BEHAVIOURAL FOUNDATIONS OF FEATURE MODELING

BEHAVIOURAL FOUNDATIONS OF FEATURE MODELING

BY

ALIAKBAR SAFILIAN, M.Sc.

A THESIS

SUBMITTED TO THE SCHOOL OF GRADUATE STUDIES IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

McMaster University

Doctor of Philosophy	Department of Computing & Software
Computer Science	McMaster University
2016	Hamilton, Ontario, Canada

TITLE:	BEHAVIOURAL FOUNDATIONS OF FEATURE MODELING
AUTHOR:	Aliakbar Safilian
SUPERVISOR:	Dr. Tom Maibaum Dr. Zinovy Diskin
NUMBER OF PAGES:	xi, 223

To my Mother, wife, and daughter

(Azizeh, Mahsa, and Sophia)

Abstract

Software product line engineering is a common method for designing complex software systems. *Feature modeling* is the most common approach to specify product lines. A feature model is a *feature diagram* (a special tree of features) plus some *crosscutting constraints*. Feature modeling languages are grouped into *basic* and *cardinality-based* models. The common understanding of the semantics of feature models is a *Boolean semantics*. We discuss a major deficiency of this semantics and fix it by applying, in turn, *modal logic*, the theory of *multisets*, and *formal language theory*. In order to adequately represent the semantics of basic models, we propose a *Kripke semantics* and show that basic feature modeling needs a *modal* rather than Boolean logic. We propose two multiset based theories for cardinality-based feature diagrams, called *flat* and *hierarchical* semantics. We show that the hierarchical semantics of a given cardinality-based diagram captures all information in the diagram. We also characterize sets of multisets, which can provide a hierarchical semantics of some diagrams. We provide three different reduction processes going from a cardinality-based diagram to an appropriate regular expression. As for crosscutting constraints, we propose a formal language interpretation of them. We also characterize some existing analysis operations over feature models in terms of operations on the corresponding languages and discuss the relevant decidability problems.

Acknowledgements

I am immensely grateful to my thesis advisors, Tom Maibaum and Zinovy Diskin, for the advice, support and encouragement they have patiently provided to me. Their advice on both research as well as on my career have been priceless.

I would like to thank the other members of my thesis examining committee, Don Batory, William M. Farmer, and Jacques Carette for their review and helpful comments.

My sincere thanks also go to Shoham Ben-David, Krzysztof Czarnecki and Ridha Khedri for several valuable discussions.

Last but not the least, I would like to thank my family. My wife has been a constant source of support and encouragement during the challenges of my PhD and life. I am truly thankful for having her in my life. Words cannot express how grateful I am to my mother for all of the sacrifices that she has made on my behalf. I would not be who I am today without the love and support of my late father.

Contents

List of Figures

Chapter 1

Introduction

This thesis provides several *theoretical* frameworks to address some challenging issues in *feature modeling* – a common approach for modeling *software product lines*. We invoke *modal logic*, *multiset theory*, and *formal language theory* to provide some appropriate semantics capturing the behavioural semantics of feature modeling.

1.1 Product Line Engineering and Feature Models

Product line engineering [PBVDL05] is a very well-known industrial approach to software/hardware design. There are many successful industrial stories applying product line engineering, e.g., "Mega-Scale Product Line Engineering at General Motors" [FKC12], "HomeAway case study" [KCB08], "LG Industrial Systems" [PBVDL05], "Lufthansa Systems" [CDMM11], and "Nokia Mobile Phones, Browsers, and Networks" [MR02, JRvdL00, Jaa02].

A *product line* is a set of products that share some commonalities along with variabilities, where commonalities and variabilities are usually captured using entities called *feautres*. There are several definitions for a feature in the literature, e.g., "a system property that is relevant to some stakeholders" [CHE05a], "a logical unit of behavior specified by a set of functional and non-functional requirements" [Bos00], "a software or hardware artifact such as requirements, architectural properties, components, or code" [HKM11a]. Some other definitions can be found in [PBVDL05, SK01, JGJ97, KLD02]. An interested reader may find in [BLR+15] a good study on features. We consider a feature as a system property that a product of the system may have.

We describe the motivation for product line engineering using a vehicle system. Consider a vehicle factory producing several different vehicle models. All models have many common features, e.g., engine, gear, body, brake, wheel, window, door, etc. They may also have some variable (a.k.a. optional) features: different engine types (e.g., gas or electric), different gear types (e.g., automatic or manual), they may be optionally equipped with an anti-skidding (abs) brake system, etc.

The idea of product line engineering is that, instead of producing products individually, the common core of a product line is produced, leaving a much smaller task to be completed, namely the adaptation of the core to a concrete application requirement. This results in a significant reduction in development time and cost [PBVDL05]. Other advantages are reusability [Bos01], reduced product risks [QC11], increased product quality [Jen07], etc.

There exist several approaches for modeling product lines, including *orthogonal variability modeling* [PBVDL05], *decision modeling* [SRG11], and *feature modeling*

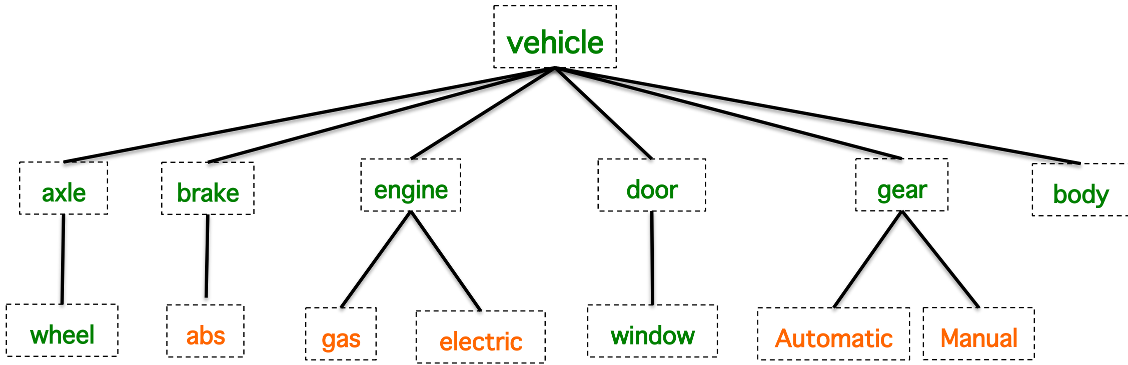


Figure 1.1: A tree of features for the vehicle product line

[KCH+90]. Feature modeling is the most common approach for modeling the commonalities and variabilities of a product line. Feature modeling was first introduced by Kang et al. in 1990 [KCH+90] and has been one of the main topics of research in software product line engineering and variability modeling ever since. Below, we describe the idea of feature modeling for modeling product lines.

A feature in a product line may be a *subfeature* of another feature. In our vehicle product line, the features wheel, abs, gas/electric, automatic/manual, and window could be seen as subfeatures of axle, brake, engine, gear, and door, respectively. By adding the product name (vehicle) as a feature, we get a *tree of features*. Figure 1.1 represents a tree of features for our vehicle product line.

There may also be some *feature constraints* on this tree. For example, some features may be *mandatory* subfeatures of their parents, e.g., brake, engine, gear, and body are mandatory subfeatures of vehicle. Some others may be *optional* subfeatures of their parents, e.g., abs and window are optional subfeatures of brake and door, respectively. Let black filled and unfilled circles on nodes denote mandatory and optional features, respectively (see Figure 1.2). We may also have some *group constraints* (a.k.a. *decomposition operations*). For example, gas and electric are in an

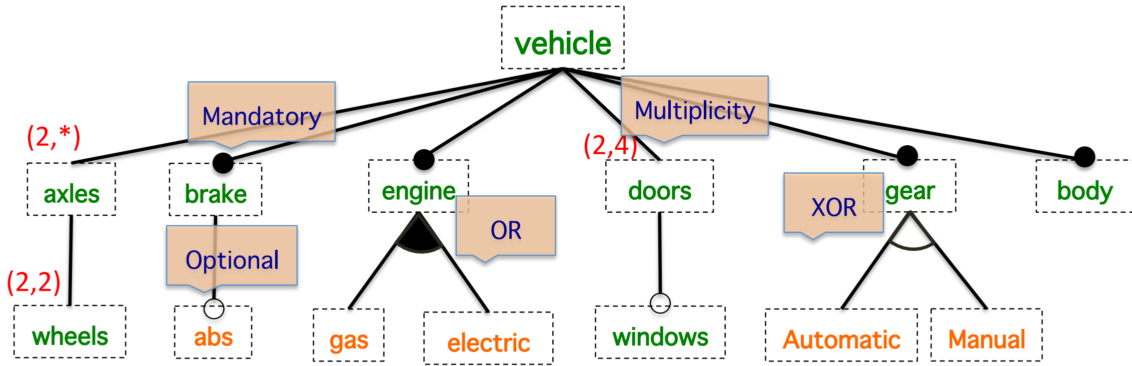


Figure 1.2: Feature constraints on features trees of vehicle product line

OR relationship, meaning that an engine can be either gas or electric or both, and automatic and manual are in an XOR relationship, meaning that a gear can be either automatic or manual but not both. We represent OR and XOR groups by filled and unfilled angles, respectively (see Figure 1.2). We may also want to deal with the number of occurrences (called *multiplicities*) of features in products. For example, let the number of occurrences of door in a vehicle be at least two and at most four. The pair $(2, 4)$ on door in Figure 1.2 represents this constraint. We already see the idea of *feature diagrams*: A feature diagram is a tree of features equipped with some annotations on features or edges showing the relationships between features.

We may also want to add some constraints involving incomparable features. (Two features of a given feature diagram are called incomparable if neither of them is a descendant of the other in the feature diagram.) Such constraints are called *cross-cutting constraints* (a.k.a. crosstree constraints). For an example, let "gas includes abs" be a crosscutting constraint over the feature diagram in Figure 1.2. It states that a gasoline engine vehicle must have an ABS brake. A feature diagram with some possible crosscutting constraints is called a *feature model*.

Feature models are grouped into *basic* and *cardinality-based* feature models. Basic feature models represent product variability and commonality in terms of Boolean constraints: optional/mandatory features, and OR/XOR decomposition operations. In cardinality-based feature models, multiplicities are used in place of traditional Boolean annotations. A more detailed background on basic and cardinality-based feature models will be given in Chapter 2.

Analysis of feature models is about extracting practically useful information from them [BBRC06]. For example, we may want to extract the following information from a given feature model: the valid products; the subfeatures of a given feature; the core features; the least common ancestor of a given set of features. Some other analyses involve two feature models and address some questions about their relationships, e.g., decide whether two given feature models represent the same products or not.

1.2 Research Questions

Industrial feature models may include thousands of features with many constraints between them, e.g., the linux kernel product line has more than 8000 features [STE$^+$10]. Therefore, we may have a very large number of possible configurations for an industrial system, e.g., 2^{8000} possible configurations[1] in the linux kernel. It becomes worse when we deal with cardinality-based feature models, as product families in cardinality-based feature models may be infinite (e.g., consider a feature in a cardinality-based feature model with no upper bound on its number of occurrences). Hence, we need to support *automated analysis* of feature models [BBRC06].

To support automated analysis of feature models, we first need to provide a formal

[1]The estimated total number of atoms in the observable Universe is between 2^{259} to 2^{273}[Loe14].

semantics for feature models. This would remove any ambiguities from the semantics of feature models and makes them processable by tools.

The common understanding of the semantics of a feature model in the literature is its product family [SHTB07], where a product family of a given feature model is a set of valid *flat configurations* of features. A flat configuration of a basic (cardinality-based, respectively) feature model is a flat set (multiset, respectively) of features satisfying the constraints of the feature model. This semantics does not capture all essential and practically important information of feature models. This is mainly because a feature model also provides a hierarchical structure for features, which is forgotten in its product family [SLB$^+$11]. For a very simple example, consider two feature models $\mathbf{M}_1$ (a is the root and b is the only mandatory child of a) and $\mathbf{M}_2$ (b is the root and a is the only mandatory child of b). $\mathbf{M}_1$ and $\mathbf{M}_2$ represent the same product family consisting of the only product $\{a, b\}$, but their hierarchical structures are different. Capturing hierarchical structures of feature models is important for several analysis operations over feature models. Indeed, any analysis operation relying on the hierarchical structure of a given feature model cannot be addressed using its product family semantics. Such analysis operations, including *least common ancestor* of a given set of features, *root feature* of a given feature model, *subfeatures* of a given feature, were explicitly characterized in the literature as necessarily relying on this information [BSRC10]. There are some other important analysis problems, in which the use of the product family semantics can be error-prone. For example, it is often important to know if one feature model $\mathbf{M}_1$ is a *refactoring* of another feature model $\mathbf{M}_2$, or a *specialization* of $\mathbf{M}_2$, or neither [TBK09]. Relying on a poor semantics like the product family semantics to define refactoring and specialization makes the definitions inadequate for their goals. Some concrete examples will be provided in Chapter 3. Another deficiency of the product family semantics is relevant to reverse engineering of feature models. Indeed, the main reason making the current state of the art approaches [SLB+11, LHLG+15] a heuristic one is mainly caused by using such a poor abstract view of feature models.

Based on the above discussion, we define a *faithful semantics* for a given feature model as a semantics capturing *all essential and practically important information* about the feature model, i.e., the product family and the hierarchical structure of the feature model.

Several formal semantics have been proposed for feature modeling, including a propositional logic encoding of basic feature models [Man02], Z-based (first order logic) semantics for basic feature models [SZFW05], algebraic based semantics for basic feature models [HKM11b], context-free grammar encoding of basic feature models [dJV02], and context-free based semantics for cardinality-based feature models [CHE05a]. However, none of them provides a faithful semantics for feature models (See Chapter 6 for a more comprehensive discussion.).

The most common methods for doing automated analysis on basic feature models are propositional logic [Bat05, MWC09, Seg08] and constraint programming based [BSTRC06b, WSB+08]. In these methods, a given feature model is translated into propositional logic formulas or a constraint programming language and then off-the-shelf tools such as Boolean Satisfaction Problem (SAT) or Constraint Satisfaction Problem (CSP) solvers are used for reasoning about the feature model. However, these encodings capture only the product family of feature models and, hence, cannot address the analysis operations relying on the hierarchy within feature models.

Automated analysis over cardinality-based feature models is much more challenging than over basic ones. As far as we know, automated analysis of cardinality-based feature models is an open problem (it has not been even partially addressed).

The above discussion lead to our main research question(s) in this thesis:

- **What is a formal and faithful semantics for feature modeling capturing more interesting and useful aspects of models?**
- **Any proposed semantics should provide (or lead us) to a framework to do automated analysis over feature models.**

1.3 Contributions

In this section, we give a summary of the main contributions made in the thesis. The contributions are grouped into the categories (i) "Modal Logic theory of basic Feature Models", (ii) "Multiset theory of Cardinalty-based Feature Diagrams", (iii) "Formal Language Theory of Cardinality-based Feature Models".

(i) Modal Logic theory of basic Feature Models. The main goal of this work is to show that Kripke structures and modal logic provide an adequate logical basis for basic feature modeling. Our main observation is that the semantics of a basic feature model should be considered as an *instantiation process* rather than its final results (products). We call intermediate states of this process *partial products*, and argue that what a feature model **M** really specifies is a partially ordered set of partial products, which we call a *partial product line* generated by **M**. The commonly considered products of **M** would be a subset of **M**'s partial product line. We then show that any partial product line can be viewed as an instance of a special type of Kripke structure, which we axiomatically define and call a *partial product Kripke structure*. The latter are specifiable by a suitable version of modal logic, which we call *partial product CTL*, as it is basically a fragment of CTL enriched with a constant (unary) modality that only holds in states representing full products. We show that any basic feature model $\mathbf{M}$ can be represented by a partial product CTL theory $\Phi(\mathbf{M})$ accurately specifying $\mathbf{M}$'s intended semantics: the main result states that for any partial product Kripke structure $\mathbf{K}$, $\mathbf{K} \models \Phi(\mathbf{M})$ iff $\mathbf{K}$ is equal to $\mathbf{M}$'s partial product line, and hence $\Phi(\mathbf{M})$ is a *sound* and *complete* representation of the feature model. In other words, $\Phi(\mathbf{M})$ provides a faithful logical theory of $\mathbf{M}$.

(ii) Multiset Theory of Cardinalty-based Feature Diagrams. A natural way to formalize the semantics of cardinality-based feature models should use a multiset theory. We propose two multiset theories for cardinality-based feature diagrams, called *flat* and *hierarchical* semantics.

Flat products are defined analogously to full (flat) products in basic feature modeling, i.e., a flat product of a given cardinality-based feature diagram is a multiset of features satisfying the subfeature relationships and multiplicity constraints. The set of all such multisets is called the *flat semantics* of the diagram. The flat semantics of a cardinality-based feature diagram provides a useful abstract view of the diagram, as it can address a large number of analysis questions about the diagram. However, it is a poor abstract view, as it does not capture some useful information about the diagram, such as the hierarchical structure.

We propose another semantics called *hierarchical products* providing a faithful semantics for cardinality-based feature diagrams. To this end, we first define a *hierarchy of multisets* built over features. A hierarchical product of a cardinality-based feature diagram is a multiset (in the corresponding multisets hierarchy) such that the rank of the multiset corresponds to the depth of the diagram. The set of all hierarchical products is called the *hierarchical semantics* of the diagram. We then prove that the hierarchical semantics of a given cardinality-based feature diagram captures *all* information about the diagram so that one can get back to the diagram from its hierarchical semantics. We also characterize sets of multisets, which can be used as the hierarchical semantics of some cardinality-based feature diagrams.

(iii) Formal Language Theory of Cardinality-based Feature Models. We invoke formal language theory [Lin11] to build some faithful semantics for cardinality-based feature modeling. This way, we can approach feature modeling problems by translating them into formal language theory problems that could be managed by well-elaborated formal language theory methods and tools. To this end, we transform cardinality-based feature diagrams to *regular expressions* and further propose a formal language interpretation of crosscutting constraints.

We have provided three types of reduction processes going from cardinality-based feature diagrams to regular expressions. Each of these transformations has its own usage and advantages. Although these transformations are different, they share some common important properties regarding the computational properties of regular languages. These properties make of the proposed frameworks good bases for addressing automated analysis over cardinality-based feature models.

Giving formal language interpretations of crosscutting constraints allows us to integrate the formal semantics of cardinality-based feature diagrams and constraints.

Accordingly, we propose a *computational hierarchy* of cardinality-based feature models which guides us in how feature models can be constructively analyzed. We also characterize some existing analysis operations over feature models in terms of languages and discuss the corresponding decidability problems.

1.4 Organization

Chapter 2 provides some background: We discuss basic and cardinality-based feature modeling using a toy example. We also provide a concise background to formal language theory and introduce some new definitions and uncommon notations for the theory used throughout the thesis.

In Chapter 3, we describe our modal logic treatment of basic feature modeling: We first argue why the Boolean semantics of feature models is a poor abstract view of models. We introduce our formal definition of the syntax of basic feature models in Section 3.1. Section 3.2 motivates the idea of partial product lines. In Section 3.3, we formally show how to get the partial product line for a given feature model. Partial product Kripke structures and CTL are introduced in Section 3.4. We present the main theoretical results in Section 3.5. Section 3.6 concludes the chapter with several other interesting applications of the modal logic view of feature modeling. The proofs of the selected theorems and propositions are given in Appendix A.

Chapter 4 discusses the multiset theories of cardinality-based feature diagrams: We first give a formal definition of the syntax of cardinality-based feature diagrams in Section 4.1. This section informally and formally discusses the flat semantics. Section 4.2 discusses and formalizes the idea of hierarchical semantics for cardinality-based feature diagrams. To this end, a hierarchy of finite multisets over a given set is proposed. The characterizations of hierarchical products and semantics are given in Sections 4.3 and 4.4, respectively. Several theorems are presented gradually leading us to show that the hierarchical semantics of a given cardinality-based feature diagram faithfully represents the diagram. Proofs of selected theorems are given in Appendix B.

We propose three different transformation procedures of cardinality-based feature diagrams to regular expressions in Chapter 5: We introduce the first transformation in Section 5.2 and show that it provides a faithful semantics for cardinality-based feature diagrams. Section 5.3 discusses the second transformation and show that it captures the flat semantics of cardinality-based feature diagrams. In Section 5.4, we propose the third transformation and show that it provides a faithful semantics for cardinality-based feature diagrams. The properties, advantages, and disadvantages of each transformation are discussed in Section 5.5. In Section 5.6, we start with a language interpretation of crosscutting constraints over cardinality-based feature diagrams. Then, a computational hierarchy of feature models are described. Section 5.7 discusses analysis operations over cardinality-based feature models in terms of languages and investigate their decidability properties. Proofs of selected theorems are given in Appendix C.

Related work and conclusions/future work are discussed in Chapter 6 and Chapter 7, respectively.

This thesis contains more than 30 lemmas and theorems. Only the main theorems are stated in the main part of the thesis. Others can be found in appendices. The reason for this is to make the thesis more readable and understandable.

A list of notations and abbreviations used in the thesis can be found on page 212.

Chapter 2

Background

In this chapter, we provide some essential background to make the later chapters readable. Section 2.1 discusses two different types of feature modeling languages, basic and cardinality-based. We describe them on a vehicle system as an example. Section 2.2 provides a concise background on some materials in the formal language theory, which are used in the thesis. Some further concepts on the theory are introduced where they are used. Along with some common concepts and notations, we will introduce some new definitions/uncommon notations which are used throughout the corresponding chapters.

2.1 Feature Modeling Languages

Feature modeling was introduced by Kang et al. in 1990 [KCH+90]. They proposed a language for feature modeling called FODA. Afterwards, many feature modeling languages with different notations have been proposed, including FORM [KKL+98], FeatuRSEB [GFd98], GP [CE00], and PLUSS [EBB05]. Schobbens et al. [SHTB07]

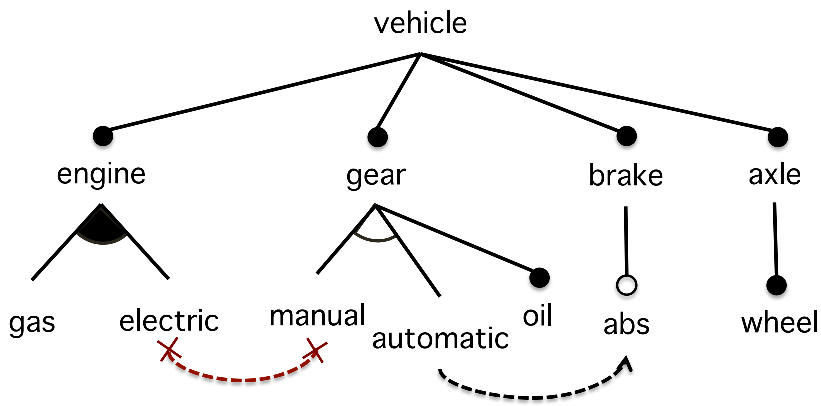


Figure 2.1: An FM of a car system

showed that all the above languages have the same expressiveness. These languages are known as *basic* feature modeling languages. Czarnecki et al in [CHE04] proposed another language called *cardinality-based* feature modeling, which provides the most expressive feature modeling language amongst the existing ones. We describe basic and cardinality-based feature models using a small part of a vehicle system as an example.

Figure 2.1 is a *basic feature model* (FM)[1] of the system. The main part of an FM, called a *basic feature diagram* (FD), is a tree of features equipped with some special annotations on the tree's elements to exhibit the relationships between features.

An edge with a black circle (•) shows a *mandatory* feature: every vehicle must include engine, gear, axle, and brake; a gear must include oil. An edge with a hollow circle (○) shows an *optional* feature: a brake can be optionally equipped with abs. These two types of edges (mandatory and optional) are called *solitary*.

Other edges are *grouped* into two groups: OR (denoted by black angles ▲) and

[1] We abbreviate basic feature models (diagrams, respect.) to FM (FD, respect.), while we will use CFM (CFD, respect.) to abbreviate cardinality-based feature models (diagrams, respect.).

XOR (denoted by hollow angles $\triangle$). The OR group $\{\mathsf{gas}, \mathsf{electric}\}$ indicates that an engine can be either gasoline or electric, or both. The XOR group $\{\mathsf{automatic}, \mathsf{manual}\}$ shows that a gear can be either automatic or manual, but not both.

A *crosscutting constraint* (CC) on an FD can be any Boolean logic formula over the set of *incomparable features* [Bat05]. Let us have the following two formulas as the CCs on our example:

– cc_1 : $\mathsf{automatic} \longrightarrow \mathsf{abs}$

– cc_2 : $\mathsf{electric} \wedge \mathsf{manual} \longrightarrow \bot$

cc_1 and cc_2 state that "a vehicle with an automatic gear must be equipped with an abs brake" and "an electric vehicle cannot have a manual gear", respectively (we have shown these two CCs by an x-ended arc and a dashed arrow in Figure 2.1, respectively.)[2]. cc_1 and cc_2 are called an *inclusive* and an *exclusive* CC, respectively.

A *product* of an FM is *usually* considered to be a set of features satisfying the constraints of the FM. The set $C = \{\mathsf{vehicle}, \mathsf{engine}, \mathsf{gear}, \mathsf{axle}, \mathsf{wheel}, \mathsf{brake}, \mathsf{oil}\}$ represents the set of the *common features* in all products of the example. This FM represents the following 8 valid products:

– $C \cup \{\mathsf{gas}, \mathsf{automatic}\}$,

– $C \cup \{\mathsf{gas}, \mathsf{automatic}, \mathsf{abs}\}$,

– $C \cup \{\mathsf{gas}, \mathsf{manual}\}$,

– $C \cup \{\mathsf{gas}, \mathsf{manual}, \mathsf{abs}\}$,

– $C \cup \{\mathsf{electric}, \mathsf{automatic}\}$,

– $C \cup \{\mathsf{electric}, \mathsf{automatic}, \mathsf{abs}\}$,

– $C \cup \{\mathsf{gas}, \mathsf{electric}, \mathsf{automatic}\}$,

[2] Note that it is not possible to visually show all CCs in this way, as a CC may involve more than two features.

– $C \cup \{\mathsf{gas}, \mathsf{electric}, \mathsf{automatic}, \mathsf{abs}\}$.

As some examples for invalid products, consider the following sets.

– $C \cup \{\mathsf{gas}, \mathsf{manual}, \mathsf{automatic}\}$,

– $C \cup \{\mathsf{electric}, \mathsf{manual}\}$,

– $C \cup \{\mathsf{electric}, \mathsf{abs}, \mathsf{automatic}\}$.

They violate the constraints XOR on $\{\mathsf{automatic}, \mathsf{manual}\}$, cc_2 , and cc_1, respectively. The set of all products of a given FM $\mathbf{M}$ is called the *product family* of the FM.

Now, suppose that we need to specify some requirements regarding the number of feature occurrences. For example, consider the following requirements:

(i) the number of axles in a vehicle cannot be one or six and there is no upper bound on it;

(ii) for each axle in a vehicle, there exists exactly two wheels.

Clearly, basic feature models like in Figure 2.1 cannot model such requirements, as they do not manage the number of occurrences of features.

To address such system requirements, Czarnecki et al. proposed *cardinality-based feature models* (CFMs) [CBUE02, CHE05a, CK05], where *multiplicity constraints* on features and groups of features, are used in place of traditional edge types (optional/mandatory features, and XOR/OR groups). Naturally, there are two types of multiplicity constraints: *feature* and *group* multiplicity constraints. A multiplicity constraint is usually expressed as a sequence of pairs (l, u), where l is a natural number, u is either a number or $*$ (representing an unbounded multiplicity) and $l \leq u$. Henceforth, we call a multiplicity constraint on a node or group a *multiplicity domain.*

Figure 2.2 provides a *cardinality-based feature diagram* (CFD) for the vehicle system including the requirements (i) and (ii). To model the OR group $\{\mathsf{gas}, \mathsf{electric}\}$ in

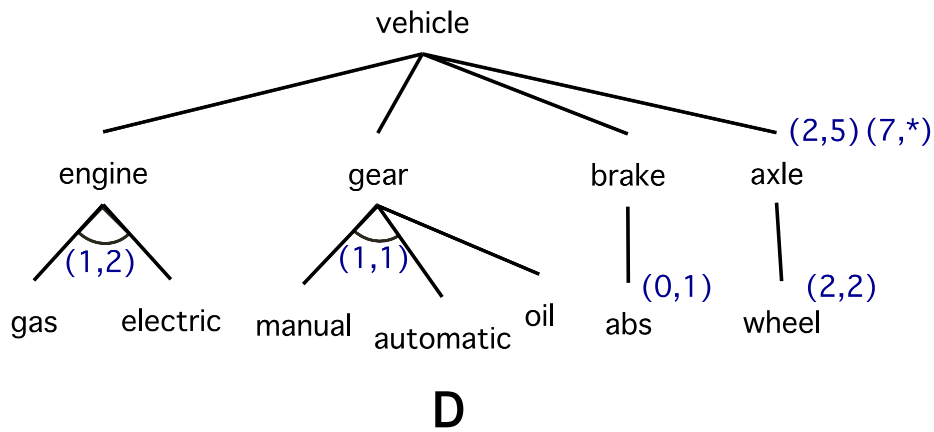


Figure 2.2: A CFM of a car system

terms of multiplicity constraints, we use the multiplicity domain $(1, 2)$ on the group. The XOR group $\{$automatic, manual$\}$ is modeled using the multiplicity domain $(1, 1)$. The feature multiplicity domain $(0, 1)$ on abs models its optional presence in a brake. The feature multiplicity domain $(2, 5)(7, *)$ on axle and $(2, 2)$ on wheel satisfy the requirements (i) and (ii), respectively. As a convention in the thesis, the multiplicity domain $(1, 1)$ is assumed if no constraint domain is shown on a feature: the multiplicity domains on engine, brake, gear, gas, electric, automatic, oil, and manual are $(1, 1)$.

CCs in a *cardinality-based feature model* (CFM) can refer to feature occurrences. Take, for example, the constraint: cc_3: "if the the engine type is electric, then the number of axles must be greater than 3". A product of a CFM is *usually* considered as a *multiset* of features satisfying the constraints. For example, the following multisets are valid products of our example, where $C = \lceil \textsf{vehicle}, \textsf{engine}, \textsf{brake}, \textsf{gear}, \textsf{oil} \rceil$.[3]

– $C \uplus \lceil \textsf{gas}, \textsf{automatic}, \textsf{abs}, \textsf{axle}^2, \textsf{wheel}^4 \rceil$,

[3] We use brackets '$\lceil$', '$\rceil$' as multiset identifiers.

– $C \uplus \lceil$ $\mathsf{electric}, \mathsf{automatic}, \mathsf{abs}, \mathsf{axle}^4, \mathsf{wheel}^8$ $\rceil$,

– $C \uplus \lceil$ $\mathsf{gas}, \mathsf{abs}, \mathsf{manual}, \mathsf{axle}^3, \mathsf{wheel}^6$ $\rceil$.

The following multisets are not valid products of the example:

– $C \uplus \lceil$ $\mathsf{electric}, \mathsf{automatic}, \mathsf{axle}^2, \mathsf{wheel}^4$ $\rceil$,

– $C \uplus \lceil$ $\mathsf{electric}, \mathsf{automatic}, \mathsf{manual}, \mathsf{axle}^4, \mathsf{wheel}^8$ $\rceil$

They violate the constraints cc_3 and the group multiplicity $(1, 1)$ on $\{\mathsf{automatic}, \mathsf{manual}\}$, respectively. Note that the set of valid products of this CFM is an infinite set.

Remark 2.1. Cardinality-based feature models are much more expressive than basic ones, as any Boolean constraint can be expressed in terms of multiplicities: a mandatory feature and an optional feature can be expressed by the multiplicity domains $(1, 1)$ and $(0, 1)$, respectively; the multiplicity domains $(1, n)$ and $(1, 1)$ model an OR and an XOR group with n elements, respectively.

2.2 Formal Languages

In this section, we provide a concise background on some materials in formal language theory, which are used in the thesis. For a more complete background, we refer the interested reader to some standard textbooks like [Lin11, Dav94, Koz97, Hop07, Coo03].

Let us, first, fix the *alphabet* (set of symbols) and denote it by Σ. Σ^* denotes the set of all finite *words* (sequences of occurrencess of symbols) built over Σ. Any subset of Σ^* is called a *language*.

The languages are grouped based on their computational properties. The most well-known are *regular*, *context-free*, *context-sensitive*, *recursive*, and *recursively enumerable* languages. It is worth mentioning that, according to the Turing-Church thesis [Cop02], we consider algorithms and Turing machines equivalent.

Recursively Enumerable Languages. A language $\mathcal{L}$ is called a *recursively enumerable* language[4] if there exists an algorithm (Turing machine) accepting the language. In other words, there is an algorithm such that it halts (terminates) for any given element (word) in $\mathcal{L}$ and outputs a symbol indicating that the input is in $\mathcal{L}$. Note that there is no guarantee that the algorithm halts for any other given words (the words that are not in the language).

Recursive Languages. A language $\mathcal{L}$ is called *recursive* (a.k.a. computable, decidable) if there exists a Turing machine such that it halts for any given word and decides whether the input is in the language or not. Obviously, the class of recursively enumerable languages is a subclass of the recursive languages class.

Context-Sensitive Languages. A language is called *context-sensitive* if there exists an algorithm written in a monotonic grammar. A grammar is monotonic if all of its production rules, a.k.a., productions, are in the form of $\Gamma \rightarrow \Theta$, where Γ and Θ are strings generated over terminals and non-terminals and Θ is not shorter than Γ. Any context-sensitive language is a recursive language.

Context-free Languages. A language is called *context-free* if it can be generated by some context-free grammar. A grammar is context-free if all of its productions are in the form of $V \rightarrow \Theta$, where V is a non-terminal symbol and Θ is a string of terminals and/or non-terminals.[5] Therefore, the class of context-free languages is a

[4] a.k.a. semi-computable, semidecidable, computably enumerable, and Turing-recognizable

[5] We could define context-free languages using *push-down automata* [Lin11]. Since we do not use push-down automata in this thesis, we do not discuss them at all.

subclass of context-sensitive languages.

Regular Languages. A language is *regular* if and only if it can be expressed by some *regular expression*, *regular grammar*, or *finite state automaton*. Any regular language is a context-free language.

Regular expressions are defined according to the following BNF expressions ($\mathbb{N}$ denotes the set of natural numbers):

$$\mathcal{R} ::= \varnothing \mid \varepsilon \mid \sigma \text{ (for any } \sigma \in \Sigma) \mid \mathcal{R} + \mathcal{R} \mid \mathcal{R}.\mathcal{R} \mid \mathcal{R}^* \mid (\mathcal{R}).$$

The expressions $\varnothing$, ε, σ (for any $\sigma \in \Sigma$) are often called *primitive* regular expressions.

The semantics of a regular expression is commonly considered as the language associated with the expression. The language associated with a regular expression $\mathcal{R}$ is denoted by $\mathcal{L}(\mathcal{R})$ and defined in an inductive way as follows:

$$\begin{gathered}
\mathcal{L}(\varnothing) = \varnothing, \\
\mathcal{L}(\varepsilon) = \{\varepsilon\}, \\
\mathcal{L}(\sigma) = \{\sigma\}, \text{ for any } \sigma \in \Sigma, \\
\mathcal{L}(\mathcal{R}_1 + \mathcal{R}_2) = \mathcal{L}(\mathcal{R}_1) \cup \mathcal{L}(\mathcal{R}_2), \\
\mathcal{L}(\mathcal{R}^*) = (\mathcal{L}(\mathcal{R}))^*, \\
\mathcal{L}((\mathcal{R})) = \mathcal{L}(\mathcal{R}), \\
\mathcal{L}(\mathcal{R}_1.\mathcal{R}_2) = \mathcal{L}(\mathcal{R}_1).\mathcal{L}(\mathcal{R}_2).
\end{gathered}$$

We extend regular expressions by a definable operation $\_^n$ (for $n \in \mathbb{N}$) on regular expressions, called *iteration*: $\mathcal{R}^n = \underbrace{\mathcal{R} \ldots \mathcal{R}}_{n}$. This will help us to give much more concise regular expressions for cardinality-based feature diagrams (see Chapter 5).

A (non-deterministic) *finite state automaton* (FSA) is a tuple (S, T, F, I) where S

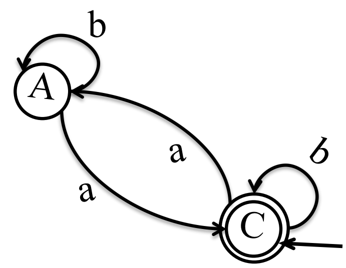


Figure 2.3: Transition graphs: example

is a finite set of states, $T : S \times (\Sigma \cup \{\varepsilon\}) \to 2^S$ is a transition function, $F \subseteq S$ is a set of final states, and $I \subseteq S$ is a set of initial states. The transition relation can be extended to $T^* : S \times \Sigma^* \to 2^S$ to deal with strings rather than a single symbol. The meaning of $T^*(s, w) = S'$ is that S' is the set of all possible states that the FSA may be in by starting at the state s and processing the word w.

Like regular expressions, the semantics of an FSA is often given via formal languages. Let $\mathcal{A} = (S, T, F, I)$ be an FSA. The language associated with $\mathcal{A}$ is denoted by $\mathcal{L}(\mathcal{A})$ and is equal to $\{w \in \Sigma^* : \exists i \in I, T^*(i, w) \cap F \neq \varnothing\}$.

Transition graphs are sometimes used to visually represent finite state automata. Figure 2.3 represents an FSA for the language $\{w \in \{a, b\}^*$: the number of occurrencess of a is even$\}$. The initial state is identified by an incoming unlabelled arrow not originating at any state. The final states are drawn with double circles.

A *regular grammar* is either a *right* or *left regular grammar*. The productions of a right (left, respectively) regular grammar must be in one of the following forms: $V \to \varepsilon$ (the same, respectively), $V \to \sigma$ (the same, respectively), $V \to \sigma V'$ ($V \to V'\sigma$, respectively), where σ is a terminal (symbol) and V, V' are non-terminals (symbols).

We also need to know the notion of *bounded regular languages*. We say a regular language $\mathcal{L}$ is a bounded regular language, if there are n words $w_1, \ldots, w_n \in \Sigma^*$ such

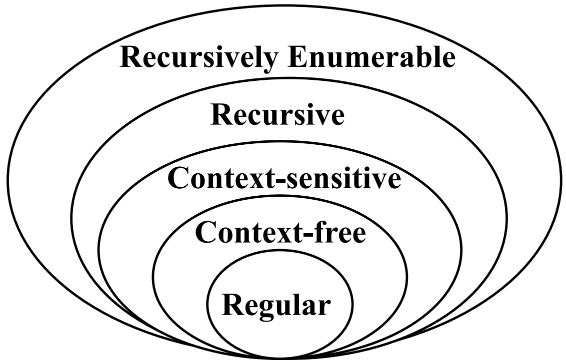


Figure 2.4: A containment hierarchy of formal languages

that $\mathcal{L} \subseteq w_1^* \ \ldots \ w_n^*$.

Figure 2.4 presents a containment hierarchy of formal languages: Regular $\subset$ Context-free $\subset$ Context-sensitive $\subset$ Recursive $\subset$ Recursively enumerable (r.e.)

The following properties of formal languages are used throughout the thesis:

Closure Properties. The class of regular languages is closed under the set operations union, intersection, complement, relative complement. It is also closed under the language operations Kleene star, concatenation, and reversal.

The class of context-free languages is closed under the set operation *union*, but is not closed under other set operations. It is also closed under Kleene star, reversal, and concatenation. This class is also closed under intersection with any regular languages.

The class of context-sensitive languages is closed under union, intersection, complement, relative complement, Kleene star, concatenation, and reversal. This class is also closed under intersection with any regular languages.

Decidability. All recursive languages (including regular, context-free, and context-sensitive languages) are decidable. Below, we state some other decidability results that are used throughout the thesis.

For a given language $\mathcal{L}$, the *emptiness problem* is to decide whether $\mathcal{L} = \varnothing$ or not. The problem is decidable in both classes of regular and context-free languages. However, it is not decidable in the class of context-sensitive languages.

Given two languages $\mathcal{L}$ and $\mathcal{L}'$, the *equality problem* is to decide whether $\mathcal{L} = \mathcal{L}'$ or not. The equality problem is decidable in the class of regular languages, but undecidable in other classes of formal languages [KN12]. However, if one of the given languages is a bounded regular and the other is context-free, then the equality problem would still be decidable [Hop69].

Given two languages $\mathcal{L}$ and $\mathcal{L}'$, the *inclusion problem* is to decide whether $\mathcal{L} \subset \mathcal{L}'$ or not. The problem is decidable in the class of regular languages, but it is undecidable in other classes of formal languages. However, if $\mathcal{L}$ is context-free and $\mathcal{L}'$ is regular, then the above problem would be decidable.

Parikh Images and Theorem. Let $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$. The *Parikh image* (a.k.a. Parikh vector) [Par61] of a given word $w \in \Sigma^*$ is the vector $(o_1, \ldots, o_n)$ where o_i denotes the number of occurrences of σ_i in w. Clearly, the Parikh image of a word can be seen as a multiset over the alphabet. Thus, we recast the original definition in the following way: The Parikh image of w is the multiset $[\sigma_1^{o_1}, \ldots, \sigma_n^{o_n}]$, where o_i denotes the number of occurrences of σ_i in w.

Parikh in [Par66] proved that the Parikh image of any context-free language, i.e., the set of Parikh images of its words, would be equal to the Parikh image of a regular language.

In the following, we introduce some other notations that are used in the later chapters. Let Σ be an alphabet.

- For any $\sigma \in \Sigma$, let σ^n denote the word $\underbrace{\sigma \ldots \sigma}_{n}$.
- $\mathbf{RE}(\Sigma)$ denotes the class of all regular expressions built over Σ.

- We use $\mathcal{R}^+$ to denote $\mathcal{R}\mathcal{R}^*$ for any regular expression $\mathcal{R}$.
- The Parikh image of a word w (a formal language $\mathcal{L}$, respectively) is denoted by $\mathrm{Par}(w)$ ($\mathrm{Par}(\mathcal{L})$, respectively).
- U_w denotes the set of alphabet symbols included in a word w.
- $\#_w(\sigma)$ denotes the number of occurrences of σ in a word w.

We will also need the following definition in Chapter 5:

Definition 2.1. For a given word w, we consider a partial order $\sqsubseteq_w \subseteq U_w \times U_w$ defined as follows: $\forall \sigma, \sigma' \in U_w$, $\sigma \sqsubseteq_w \sigma'$ iff any occurrences of σ' is preceded by some occurrencess of σ in w. $\square$

Definition 2.2. We consider a relation $\leq_{\text{seq}}$: $\Sigma^* \times \Sigma^*$ defined as $w \leq_{\text{seq}} w'$ iff w is a subsequence of w'. $\square$

Chapter 3

Modal Logic Theory of Basic Feature Models

As already discussed in the Introduction (Section 1.2), the commonly considered semantics for a basic feature models (FM) is a Boolean semantics, that is, the set of flat products represented by the FM. For formal analysis, FMs are usually encoded by propositional theories with Boolean semantics. In this chapter, 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.

The FM $\mathbf{M}_1$ in Figure 3.1 says that a car *must* have an engine and brake, and brake can be *optionally* equipped with an anti-skidding system (abs). The model specifies a product line consisting of two products: $P = \{\mathsf{car}, \mathsf{engine}, \mathsf{brake}\}$ and $P' = P \cup \{\mathsf{abs}\}$. As FMs of industrial size can be big and complex, they require tools for their management and analysis, and thus should be represented by formal objects processable by tools. A common approach is to consider features as atomic propositions, and view an FM as a theory in Boolean propositional logic (BL), whose valid valuations are to

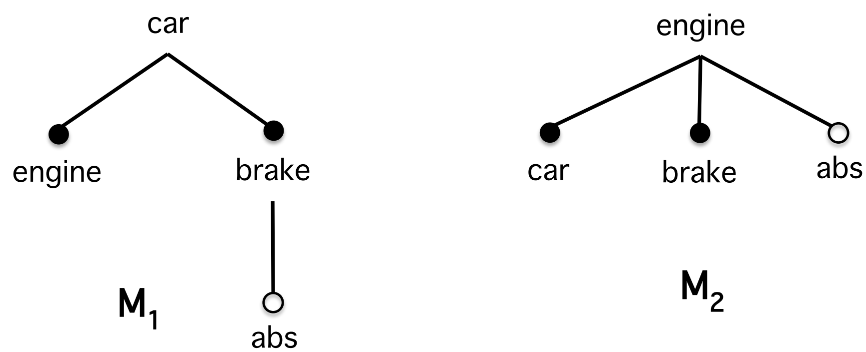


Figure 3.1: Two FMs with the same Boolean semantics

be exactly the valid products defined by the FM [Bat05]. For example, model $\mathbf{M}_1$ represents the BL theory $\Phi(\mathbf{M}_1) = \{\mathsf{engine} \rightarrow \mathsf{car}, \mathsf{brake} \rightarrow \mathsf{car}, \mathsf{abs} \rightarrow \mathsf{brake}\} \cup \{\mathsf{car} \rightarrow \mathsf{engine}, \mathsf{car} \rightarrow \mathsf{brake}\} \cup \{\mathsf{car}\}$: the first three implications encode subfeature dependencies (a feature can appear in a product only if its parent is in the product), and the next two implications encode the mandatory dependencies between features. The root feature must be always included in the product.

Now, consider the FM $\mathbf{M}_2$ in Figure 3.1. This model is essentially different from $\mathbf{M}_1$, but has the same set of products, $\{P, P'\}$ determined by an equivalent BL theory $\Phi(\mathbf{M}_2) = \{\mathsf{car} \rightarrow \mathsf{engine}, \mathsf{brake} \rightarrow \mathsf{engine}, \mathsf{abs} \rightarrow \mathsf{engine}\} \cup \{\mathsf{engine} \rightarrow \mathsf{car}, \mathsf{engine} \rightarrow \mathsf{brake}\} \cup \{\mathsf{engine}\}$: only grouping of implications has changed, but it is immaterial for BL. The core of the problem is that two semantically different dependencies (the parent feature and a mandatory subfeature) are both encoded by implication and hence are not distinguished.

We are not the first to have noticed this drawback, e.g., it is mentioned in [SLB$^+$11] (where FMs' semantics not captured by BL is called *ontological*), and probably many researchers and practitioners in the field are aware of the situation. Nevertheless, as far as we know, no alternative to the Boolean semantics of feature modeling (FM) has been proposed in the literature,[1] which we think is theoretically unsatisfactory. Even more importantly, inadequate logical foundations for FM hinder practical analyses: as important information contained in FMs (*hierarchical structure*) is not captured by their traditional 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 rather than explicated in the specification, with all its negative consequences for software testing, debugging, maintenance, and communication between the stakeholders.

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

[1] Along with propositional logic, there have been proposed some other logical approaches for treatment of FMs such as first-order logic (see Section 6.6). However, they also suffer the same problem: they take into account only the set of valid products of FMs.

iff $\mathbf{K}$ is equal to $\mathbf{M}$'s partial product line, and hence $\Phi_{\mathsf{ML}}(\mathbf{M})$ is a *sound* and *complete* representation of the FM. Then we can replace FMs by the respective ppCTL-theories, which are highly amenable to formal analysis and automated processing.

The organization of this chapter is as follows: Section 3.1 gives our formal framework for the syntax of FMs. Section 3.2 motivates the formal framework developed in the chapter: we show how the deficiency of the traditional Boolean semantics can be fixed by introducing partial products and transitions between them. In Section 3.3, the notion of partial product lines generated for given FMs is formalized. In Section 3.4, we introduce the notion of partial product Kripke structures as immediate abstractions of partial product lines, and ppCTL as a language to specify partial product Kripke structures' properties. We show, step-by-step, how to translate an FM into a ppCTL-theory, and prove our main results in Section 3.5. In Section 3.6, we discuss some other interesting practical applications of the modal logic view of FMs.

3.1 Basic Feature Models: A Formal Framework

A unified formal approach to basic feature models and their Boolean semantics is developed in [SHT06]. Our variant of the formalization of the basic feature models is designed to support our work: the structure of our modal theories will follow the structure of feature models as defined below. Typical FMs are trees of features with some extra structures, like in Figure 2.1. In our framework, mandatory features and XOR-groups are derived constructs. A mandatory feature can be seen as a singleton OR-group. An XOR-group can be expressed by an OR-group with some additional exclusive constraints between its elements.

Definition 3.1 (Feature Diagrams). A *feature diagram* (FD) is a pair $T_{\mathcal{OR}} = (T, \mathcal{OR})$ of the following components.

(i) $T = (F, r, \_^{\uparrow})$ is a tree whose nodes are *features*: F denotes the set of all features, $r \in F$ is the root, and function $\_^{\uparrow}$ maps each non-root feature $f \in F_{-r} \stackrel{\text{def}}{=} F \setminus \{r\}$ 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 $f_{\downarrow\downarrow}$, respectively.

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

(ii) $\mathcal{OR}$ is a function that assigns to each feature $f \in F$ a set $\mathcal{OR}(f) \subset 2^{f_{\downarrow}}$ (possibly empty) of *disjoint* subsets of f's children called *OR-groups*. If a group $G \in \mathcal{OR}(f)$ is a singleton $\{f'\}$ for some $f' \in f_{\downarrow}$, we say that f' is a *mandatory* subfeature of f. For example, in Figure 2.1, $\mathcal{OR}(\mathsf{gear}) = \{\{\mathsf{manual}, \mathsf{automatic}\}, \{\mathsf{oil}\}\}$, 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 of f. For example, in Figure 2.1, $\mathcal{OR}(\mathsf{brake}) = \varnothing$, and abs is an optional subfeature of brake. □

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

Definition 3.2 (Feature Models). A *feature model* (FM) is a triple $\mathbf{M} = (T_{\mathcal{OR}}, \mathcal{EX}, \mathcal{IN})$ with $T_{\mathcal{OR}}$ a feature diagram as defined in Definition 3.1, and two additional components defined below:

(i) $\mathcal{EX} \subseteq \#2^F$ is a set of *exclusive dependencies* between features. For example, in Figure 2.1, $\mathcal{EX} = \{\{\mathsf{electric}, \mathsf{manual}\}, \{\mathsf{manual}, \mathsf{automatic}\}\}$.

(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, feature model in Figure 2.1 has $\mathcal{IN} = \{\mathsf{automatic} \rightarrow \mathsf{abs}\}$.

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

$\Box$

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 $\mathbf{M} = (T, \mathcal{OR}, \mathcal{EX}, \mathcal{IN})$. If needed, we will subscript $\mathbf{M}$'s components with index $_\mathbf{M}$, e.g., write $F_\mathbf{M}$ 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 feature models over the same feature set F is denoted by $\mathcal{M}(F)$.

3.2 Partial Product Lines: Motivation

This section aims to motivate the formal framework we develop in this chapter. In Section 3.2.1, we introduce *partial products* and *partial product lines* (PPLs). We begin with PPLs generated by simple FMs, which can be readily explained in lattice-theoretic terms. Then, in Section 3.2.2, we show that PPLs generated by complex FMs are more naturally, and even necessarily, considered as transition systems.

3.2.1 Products as Processes

What is lost in the traditional Boolean encoding is the *dynamic* nature of the notion of products. An FM defines not just a set of valid products but the very way

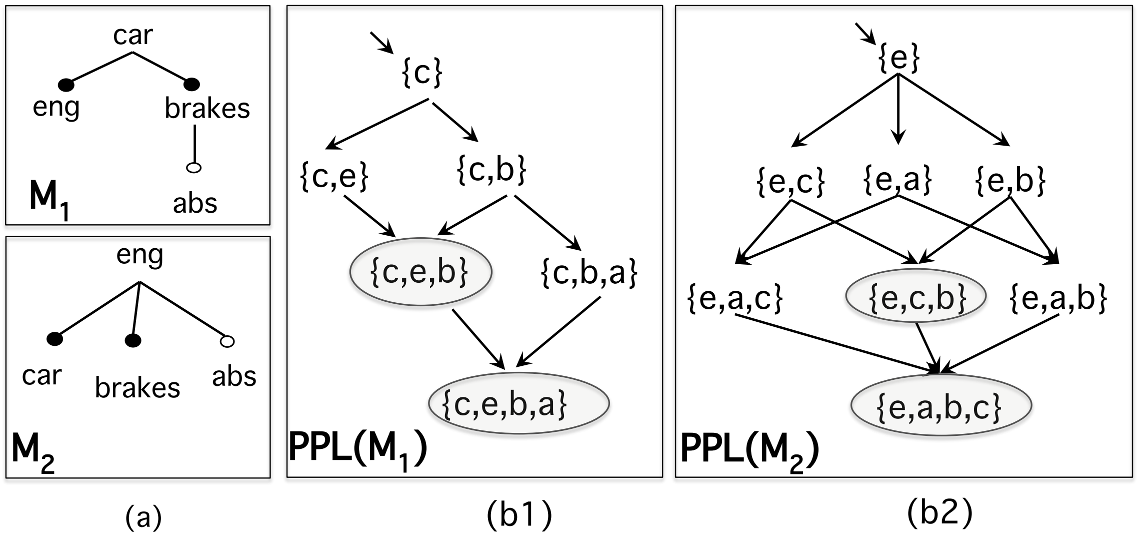


Figure 3.2: From FMs to PPLs: simple cases

these products are to be (dis)assembled step by step from constituent features. Correspondingly, a product line appears as a *transition system* initialized at the root feature (say, car for $\mathbf{M}_1$ in Figure 3.2 a) and gradually progressing towards fuller products (say, $\{\mathsf{car}\} \rightarrow \{\mathsf{car}, \mathsf{engine}\} \rightarrow \{\mathsf{car}, \mathsf{engine}, \mathsf{brake}\}$ or $\{\mathsf{car}\} \rightarrow \{\mathsf{car}, \mathsf{brake}\} \rightarrow \{\mathsf{car}, \mathsf{brake}, \mathsf{abs}\} \rightarrow \{\mathsf{car}, \mathsf{brake}, \mathsf{abs}, \mathsf{engine}\}$); we will call such sequences *instantiation paths*. The graph in Figure 3.2(b1) specifies all possible instantiation paths for $\mathbf{M}_1$ (c, e, b, a stand for car, engine, brake, abs, respectively, 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 brake, and product {c,b} is missing feature engine. 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 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 PPL determined by FM $\mathbf{M}_1$, and write $PPL(\mathbf{M}_1)$ or PPL_1. In a similar way, the PPL of the second FM, $PPL(\mathbf{M}_2)$, is built in Figure 3.2(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 $\{\mathsf{c}\}$ and $\{\mathsf{c},\mathsf{b}\}$ present in PPL_1 but absent in PPL_2, whereas $\{\mathsf{e}\}$ and $\{\mathsf{e},\mathsf{a}\}$ are present in PPL_2 but absent from PPL_1) too. This essential difference between PPLs properly reflects the essential difference between the FMs. Note that capturing the difference between the two FMs $\mathbf{M}_1$ and $\mathbf{M}_2$ is important, as they represent two different product lines: The first model ($\mathbf{M}_1$) represents a product line for cars, while the second one ($\mathbf{M}_2$) represents a product line for engines[2].

3.2.2 PPLs: From lattices to transition systems

Generating partial product lines $PPL_{1,2}$ of FMs $\mathbf{M}_{1,2}$ in Figure 3.2 can be readily explained in lattice-theoretic terms. Let us first forget about mandatory bullets, and consider all features as optional. Then both FMs are just trees, and hence are posets,

[2] If we just ignore that $\mathbf{M}_2$ is pathological, as the feature car cannot be a subfeature of the feature engine in the reality.

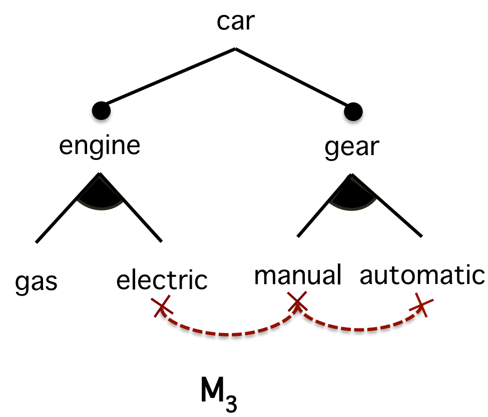


Figure 3.3: An FM: a fragment of Figure 2.1

even join semi-lattices (joins go up in feature trees). Valid products of an FM M_i are upward-closed sets of features (filters), and form a lattice (consider Figure 3.2(b1,b2) as Hasse diagrams), whose join is set union, and meet is intersection. If we freely add meets (go down) to posets $M_{1,2}$ ($\mathsf{engine} \wedge \mathsf{brake}$ etc.), and thus freely generate lattices $L(M_i)$, $i = 1, 2$, over the respective posets, then lattices $L(M_i)$ and PPL_i will be dually isomorphic (Birkhoff duality).

The forgotten mandatoriness of some features appears as incompleteness of some objects; we call them *proper* partial products. Partial products closed under mandatoriness are *full*. Thus, PPLs of simple FMs such as in Figure 3.2(a) are their filter lattices with distinguished subsets of full products. In the next section, we will argue that this lattice-theoretic view does not work for more complex FMs.

Figure 3.3 shows a fragment of the FM in Figure 2.1, in which, for uniformity, we have presented the XOR-group as an OR-group with a new crosscutting constraint added to the tree (note the $\times$-ended arc between manual and automatic[3]). To build

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

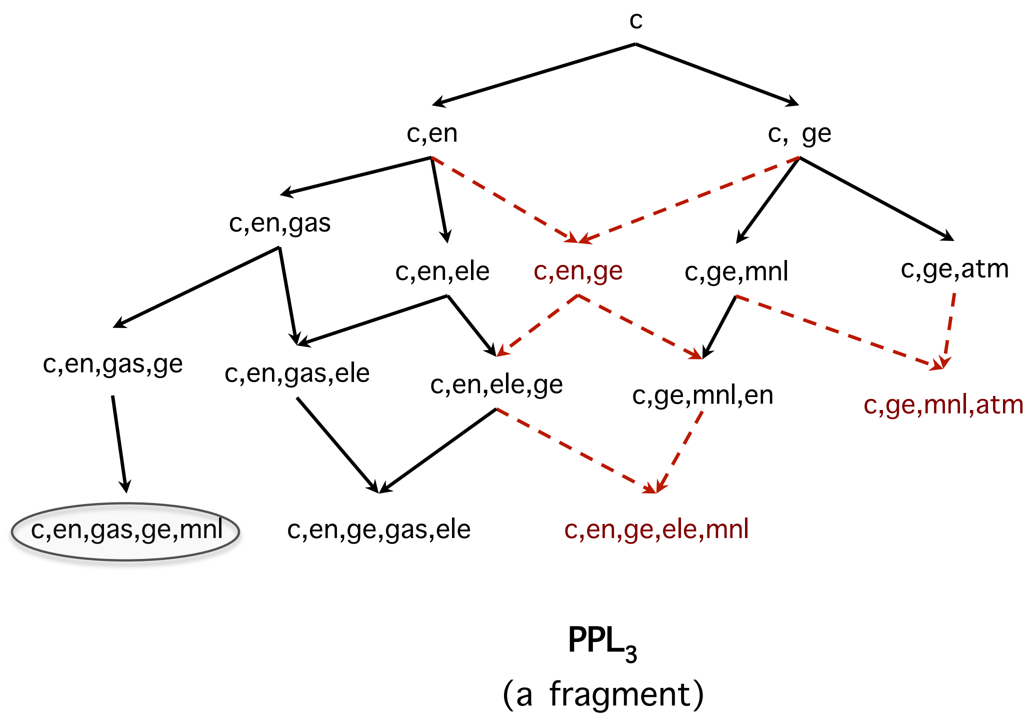


Figure 3.4: A fragment of the PPL of Figure 3.3

the PPL, we follow the idea described above, and first consider $\mathbf{M}_3$ as a pure tree-based poset with all the extra-structure (denoted by black bullets and black triangles) removed. Figure 3.4 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 engine etc.

Now let us consider how the additional structure embodied in the feature influences the PPL. Two exclusive crosscutting constraints force us to exclude the bottom central and right products from the PPL; they are shown in brown-red and the respective edges are dashed. To specify this lattice-theoretically, we add to the lattice of features a universal *bottom* element $\perp$ (a feature to be a subfeature of any feature), and write two defining equations: ($\mathsf{ele} \wedge \mathsf{manual} = \bot$) and ($\mathsf{manual} \wedge \mathsf{automatic} = \bot$). Then, in the filter lattice, the formal down-join of products $\{\mathsf{c},\mathsf{en},\mathsf{ele},\mathsf{ge}\}$ and $\{\mathsf{c},\mathsf{ge},\mathsf{mnl},\mathsf{en}\}$ "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 FM $\mathbf{M}_3$ (given by black bullets and triangles). This structure determines a subset of the PPL consisting of full products (e.g., $\{\mathsf{c},\mathsf{en}, \mathsf{gas}, \mathsf{ge}, \mathsf{mnl}\}$ in Figure 3.4) 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 (Figure 3.4). The left instantiation path, i.e., $\{\mathsf{c}\} \longrightarrow_{\mathsf{en}} \{\mathsf{c}, \mathsf{en}\} \longrightarrow_{\mathsf{ge}} P$, leading to this product 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 a 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's 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). Note that this principle substantially reduces the complexity of the PPL for a given FM

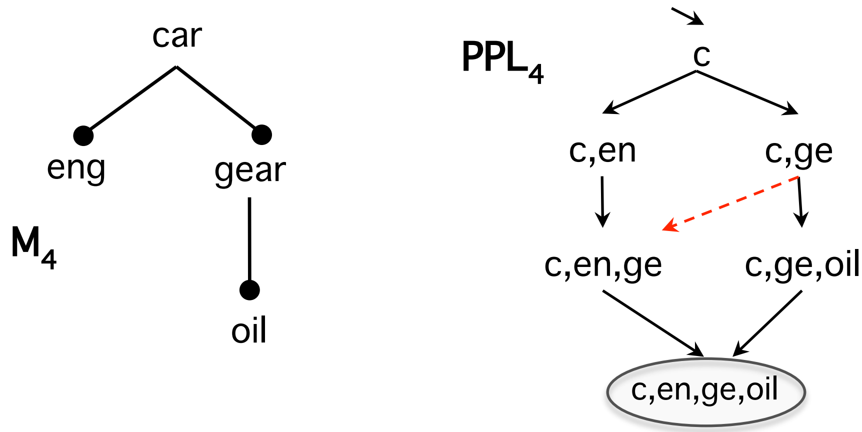


Figure 3.5: Exclusion of an edge due to I2C

without loss of any information of the FM.

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 3.5 shows a simple example with FM $\mathbf{M}_4$ 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 $\{\mathsf{c}, \mathsf{en}\}$ 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 something else (transition systems/Kripke structures and modal logic are) needed. Moreover, even if all inclusions are transitions, Boolean logic is still 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 propositional Boolean logic, it provides a more expressive language for crosscutting constraints over FMs. Later in Section 3.6, we will provide an example in which some crosscutting constraints cannot be expressed by propositional Boolean logic, but can be in 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 have a clear semantic meaning: the instantiation process can stay in a full product state indefinitely. This way, the transition relation in a PPL would be left-total, as any partial product eventually evolves into a full product. This also makes PPLs standard Kripke structures used for the semantics of CTL in which transition relations must be left-total.[4]

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

3.3 Partial Product Lines: Formally

In this section, we are going to formalize the notion of PPLs and formally show how to get a PPL for a given FM. As already discussed, both partial products and transitions are first class citizens in PPLs. In Section 3.3.1, we define a BL encoding of an FM,

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

and the corresponding notions of full and partial products. In Section 3.3.2, an FM's PPL is formally defined as a transition system.

3.3.1 Full and Partial Products

A common approach to formalizing the product line (of full products) for a given FM is to use Boolean propositional logic [Bat05]. Features are considered as atomic propositions, and dependencies between features are specified by logical formulas. For example, if a feature f' is a subfeature of feature f, we have an implication $f' \to f$ (if a product has feature f', it must have feature f as well). If $\{g_1, g_2\}$ is an OR-group of f's subfeatures, we write $f \to (g_1 \vee g_2)$; if, in addition, features g_1, g_2 are mutually exclusive, we write $g_1 \wedge g_2 \to \bot$. In this way, given an FM $\mathbf{M} = (T, \mathcal{OR}, \mathcal{EX}, \mathcal{IN})$, each of its four components gives rise to a respective propositional theory as shown in the upper four rows of Table 3.1: 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.[5] Together these theories constitute theory $\Phi^!_{\mathsf{BL}}(M)$, and a set of features P is a legal full product for $\mathbf{M}$ iff $P \models \Phi^!_{\mathsf{BL}}(\mathbf{M})$. Since publishing the seminal paper [Man02], this propositional view of basic feature modeling became common and has been used in both theoretical and practice-oriented work [Bat05, CW07, SLB+11].

Below we revise the propositional encoding of FMs: we introduce two propositional theories for, respectively, partial and full products (subsection A) and show how the $\mathsf{I2C}$-principle can be propositionally encoded (subsection B).

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

Table 3.1: Boolean theories extracted from an FM $\mathbf{M} = (T_{\mathcal{OR}}, \mathcal{EX}, \mathcal{IN})$

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

A: Enabling vs. Causality.

The encoding above has a drawback that we discussed in the introduction: two different relationships between features (being a subfeature, $f' \to f$, and being a mandatory subfeature, $f \to f'$) are similarly encoded. This implies $f \leftrightarrow f'$ for any mandatory subfeature f' of f, and leads to misrepresentation of the hierarchical structure of 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 products must respect it. [6]

Thus, we have two Boolean theories for the same FM $\mathbf{M}$. One is the theory of partial products and another is the theory of full products. The theory of *partial products* is denoted by $\Phi_{\mathsf{BL}}(\mathbf{M})$ (for now without the bang superscript) that encodes the basic structural dependencies a well-formed partial product must satisfy, and thus defines all partial products. This theory consists of three components as specified in row (all) in the Table: $\Phi_{\mathsf{BL}}(T)$ is the BL-encoding of subfeature dependencies (row (1)), $\Phi_{\mathsf{BL}}(\mathcal{EX})$ is the BL-encoding of exclusive dependencies (row (2)), and in section B we will consider yet another ingredient—the Boolean encoding of the $\mathsf{I2C}$-condition, $\Phi^{\mathsf{I2C}}_{\mathsf{BL}}(T_{\mathcal{OR}})$. The other propositional theory, $\mathbf{M}$'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 the BL-encoding $\Phi^{!}_{\mathsf{BL}}(\mathcal{OR})$ of the mandatoriness dependencies embodied in the $\mathcal{OR}$-structure (row ($3^!$)), plus the Boolean logic encoding $\Phi^{!}_{\mathsf{BL}}(\mathcal{IN})$ of the inclusive crosscutting constraints (row($4^!$)), which we treat as mandatory for only full products rather than affecting instantiation (i.e., as causal rather than enabling). With a more refined approach to feature modeling, a crosscutting constraint should be labeled as either causal or enabling, but with the current feature modeling practice, crosscutting constraints are not labeled and we thus consider them as causal, i.e., constraining full products only.

Definition 3.3 (Full Products). A *full product* over an FM $\mathbf{M} = (T_{\mathcal{OR}}, \mathcal{EX}, \mathcal{IN})$ is a set of features $P \subseteq F$ satisfying theory $\Phi^{!}_{\mathsf{BL}}(\mathbf{M})$ defined in Table 3.1.

The set of all full products is called $\mathbf{M}$'s *full product set* and denoted by $\mathcal{FP}_M$.

[6] 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 Section 6.1.

Thus, $\mathcal{FP}_{\mathbf{M}} = \{P \subseteq F : P \models \Phi^{!}_{\mathsf{BL}}(M)\}$. $\square$

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: Instantiate to Completion via Propositional Logic.

Consider once again PPL$_3$ in Figure 3.4, from which product $\{\mathsf{c}, \mathsf{en}, \mathsf{ge}\}$ is excluded as violating the $\mathsf{I2C}$ 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 further down the lattice they are combined in product $\{\mathsf{c}, \mathsf{en}, \mathsf{ele}, \mathsf{ge}\}$ below $\{\mathsf{c}, \mathsf{en}\}$, and in product $\{\mathsf{c}, \mathsf{ge}, \mathsf{mnl}, \mathsf{en}\}$ below $\{\mathsf{c}, \mathsf{ge}\}$ as well. In other words, the conflict between features en and ge is transient rather than permanent, and its propositional specification is not trivial. We solve this problem by introducing the notion of a *feature subtree* induced by a feature in Definition 3.4, and then specifying theory $\Phi^{\mathsf{I2C}}_{\mathsf{BL}}(T_{\mathcal{OR}})$ shown in row (3) in Table 3.1. 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 3.4 (Induced Subtrees). Let $T_{\mathcal{OR}} = (T, \mathcal{OR})$ be a feature diagram over a set of features F, and $f \in F$. A *feature subtree induced* by f is a pair $T^{f}_{\mathcal{OR}} = (T^f, \mathcal{OR}^f)$ with T^f being the tree under f, i.e., $T^f \stackrel{\text{def}}{=} (f_{\downarrow\downarrow} \cup \{f\}, f, \_^{\uparrow})$, and mapping $\mathcal{OR}^f$ is inherited from $\mathcal{OR}$, i.e., for any $g \in f_{\downarrow\downarrow}$, $\mathcal{OR}^f(g) = \mathcal{OR}(g)$. $\square$

Now we can specify theory $\Phi^{\mathsf{I2C}}_{\mathsf{BL}}(T_{\mathcal{OR}})$ as shown in row (3) in Table 3.1. The theory formalizes the idea that if a valid product contains two incomparable features, then at least one of these features must be fully instantiated within the product.

Definition 3.5 (**Partial Products).** A *partial product* over FM $\mathbf{M} = (T_{\mathcal{OR}}, \mathcal{EX}, \mathcal{IN})$ is a set of features $P \subseteq F$ satisfying the instantiation theory $\Phi_{\mathsf{BL}}(\mathbf{M})$ specified in row (all) in Table 3.1. (Recall that a full product is a set of features satisfying theory $\Phi^{!}_{\mathsf{BL}}(\mathbf{M})$.) We denote the set of all partial products by $\mathcal{PP}_{\mathbf{M}}$. Thus, $\mathcal{PP}_{\mathbf{M}} = \{P \subseteq F : P \models \Phi_{\mathsf{BL}}(\mathbf{M})\}$. □

Proposition 3.1. For any FM $\mathbf{M}$, $\Phi^{!}_{\mathsf{BL}}(\mathbf{M}) \models \Phi_{\mathsf{BL}}(\mathbf{M})$. Hence, full products as defined in Definition 3.3 form a subset of partial products, $\mathcal{FP}(\mathbf{M}) \subseteq \mathcal{PP}(\mathbf{M})$. □

Note that transition exclusion discussed in Section 3.2.2 cannot be explained with Boolean logic and needs a modal logic; we will give a suitable logic and show how it works in Section 3.5.

3.3.2 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 $P' = P \cup \{f\}$ is valid as well. We then write $P \longrightarrow P'$ and call the pair (P, P') *a valid (elementary* or *step) transition*.

Two necessary conditions are obvious: the parent $f^{\uparrow}$ must be in P, and f should not be in conflict with features in P, that is, $P' \models \big(\Phi_{\mathsf{BL}}(T) \cup \Phi_{\mathsf{BL}}(\mathcal{EX})\big)$. Compatibility with $\mathsf{I2C}$ is more complicated: we would need to formalize relative completeness of P in its branch, as follows.

Definition 3.6 (**Relative fullness).** Given a product P and a feature $f \notin P$, the following theory (continuing the list in Table 3.1) is defined:

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

where $T^g_{\mathcal{OR}}$ denotes the subtree induced by feature g as described in Definition 3.4. (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 FM $\mathbf{M}_4$ in Figure 3.5, for product P_1=$\{\mathsf{car}, \mathsf{engine}\}$ 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 $f_2 = \mathsf{engine}$, $P_2 \not\models \Phi^{\mathsf{I2C}}_{\mathsf{BL}}(P_2, f_2)$ because $\Phi^{!}_{\mathsf{BL}}(T^{\mathsf{gear}}_{\mathcal{OR}}) = \{\mathsf{gear} \to \mathsf{oil}\}$ and $P_2 \not\models \{\mathsf{gear} \to \mathsf{oil}\}$.

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

Definition 3.7 (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.

Theorem 3.1. If P is a valid partial product and $P \longrightarrow P'$, then P' is a valid partial product. □

Finally, we formalize partial product lines as follows:

Definition 3.8 (Partial Product Lines). Let $\mathbf{M} = (T_{\mathcal{OR}}, \mathcal{EX}, \mathcal{IN})$ be an FM. The *partial product line* (PPL) determined by M is a triple $\mathbb{P}(\mathbf{M}) = (\mathcal{PP}_{\mathbf{M}}, \longrightarrow_{\mathbf{M}}, I_{\mathbf{M}})$ with the set $\mathcal{PP}_{\mathbf{M}}$ of partial products given by Definition 3.5, transition relations $\longrightarrow_{\mathbf{M}}$ given by Definition 3.7 (so that full products, and only them, are equipped with self-loops), and the initial product $I_{\mathbf{M}} = \{r\}$ consisting of the root feature. □

3.4 Partial Product Kripke Structures and Their Logic

In this section, we introduce *partial product Kripke structures*, which are an immediate abstraction of partial product lines generated by FMs. Then we introduce a modal logic called *partial product CTL*, which is tailored for specifying partial product Kripke structures' properties.

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

Definition 3.9 (partial product Kripke Structure). Let F be a finite set (of *features*). A *partial product Kripke structure* (ppKS) over F is a triple $\mathbf{K} = (\mathcal{PP}, \longrightarrow, I)$ 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[7]. In addition, the following three conditions hold ($\longrightarrow^+$ denotes the transitive closure of $\longrightarrow$):

(*Singletonicity*) For all $P, P' \in \mathcal{PP}$, if $P \longrightarrow P'$ and $P \neq P'$, then $P' {=} P \uplus \{f\}$ for

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

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

Proposition 3.2. 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-ppKS of a ppKS.

Definition 3.10 (Sub-ppKS). Let $\mathbf{K}$, $\mathbf{K}'$ be two ppKSs. We say $\mathbf{K}$ is a sub-ppKS of $\mathbf{K}'$, denoted by $\mathbf{K} \sqsubseteq \mathbf{K}'$, iff $I_\mathbf{K} = I_{\mathbf{K}'}$, $\mathcal{PP}_\mathbf{K} \subseteq \mathcal{PP}_{\mathbf{K}'}$, and $\longrightarrow_\mathbf{K} \subseteq \longrightarrow_{\mathbf{K}'}$. □

The following proposition is an obvious corollary of Definition 3.8.

Proposition 3.3. Let $\mathbf{M} \in \mathcal{M}(F)$ be an FM. Its partial product line is an ppKS, i.e., $\mathbb{P}(\mathbf{M}) \in \mathcal{K}(F)$. □

The proposition above is not very interesting: there is a rich structure in $\mathbb{P}(\mathbf{M})$ that is not captured by the fact that $\mathbb{P}(\mathbf{M})$ is a ppKS—the class $\mathcal{K}(F)$ is too big.

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

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

Definition 3.11 (partial product CTL). Partial product CTL (ppCTL) formulas are defined using a finite set of propositional letters F, an ordinary signature of propositional connectives: constant (zero-ary) $\top$ (truth), unary $\neg$ (negation) and binary $\vee$ (disjunction) connectives, and a modal signature consisting of modal operators: constant (zero-ary) modality !, and three CTL unary modalities AX, AF, and AG. The *well-formed* ppCTL-*formulas* ϕ are given by the following grammar:

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

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

The semantics of ppCTL-formulas is given using the class $\mathcal{K}(F)$ of ppKSs built over the same set of features F. Let $\mathbf{K} \in \mathcal{K}(F)$ be a ppKS $(\mathcal{PP}, \longrightarrow, I)$. We first define a satisfaction relation $\models$ between a product $P \in \mathcal{PP}$ and a formula $\phi \in \mathsf{ppCTL}(F)$ by structural induction on ϕ. This is done in Table 3.2.

Table 3.2: Rules of satisfiability

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

3.5 ppCTL theory of a Feature Model

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

Theorem 3.2 (Soundness). $\mathbb{P}(\mathbf{M}) \models \Phi_{\mathsf{ML}}(\mathbf{M})$. □

Theorem 3.3 (Semi-completeness). $\mathbf{K} \models \Phi_{\mathsf{ML}\subseteq}(\mathbf{M})$ implies $\mathbf{K} \sqsubseteq \mathbb{P}(\mathbf{M})$. □

Theorem 3.4 (Completeness). $\mathbf{K} \models \Phi_{\mathsf{ML}}(\mathbf{M})$ *iff* $\mathbf{K} = \mathbb{P}(\mathbf{M})$. □

Completeness allows us to replace FMs by the respective ppCTL-theories, which are highly amenable to formal analysis and automated processing. *Semi-completeness* is useful (as an auxiliary intermediate step to completeness, but also) for some important practical problems in feature modeling such as *specialization* [TBK09] ($\mathbf{M}$ is a specialization of another FM $\mathbf{M}'$ if the latter subsume the former in a semantic sense), and some other analysis operations [BSRC10] over FMs. These operations are normally considered for full product lines only, but can be redefined for PPLs as well (see Section 3.6.1).

We will build theories $\Phi_{\mathsf{ML}\subseteq}(\mathbf{M})$ and $\Phi_{\mathsf{ML}}(\mathbf{M})$ from small *component* theories, which specify the respective properties of $\mathbf{M}$'s PPL in terms of ppCTL. Before we proceed to defining these theories and giving proofs, in order to provide some guidance through the proofs, we discuss, in Section 3.5.1, the structure of the entire component family, and explain how the compound theories, $\Phi_{\mathsf{ML}\subseteq}(\mathbf{M})$, $\Phi_{\mathsf{ML}}(\mathbf{M})$, and $\Phi_{\mathsf{ML}+}(\mathbf{M}) \stackrel{\text{def}}{=} \Phi_{\mathsf{ML}}(\mathbf{M}) \setminus \Phi_{\mathsf{ML}\subseteq}(\mathbf{M})$ are built from them. Then, in Section 3.5.2, we zoom into component theories and explain how they are built. The proofs can be found in Appendix A.

3.5.1 Structure of the component family

All component theories we need are referenced in Table 3.3. Its bottom row consists of the three compound theories mentioned above; the last (rightmost) column theory is the union of the theories in its row—this is a general rule for the entire table. Another general rule is that each theory in the bottom row is the union of all components above it in its column(s) (and $\Phi_{\mathsf{ML}\subseteq}(\mathbf{M})$ 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. full product line and PPL (split into Boolean and modal components), and to provide completeness wrt. full product line and PPL: a theory in the last column is

Table 3.3: Component and Compound Theories

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

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. full product line (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.

3.5.2 The Content of Component Theories

Now we specify the internal theories, and explain their meaning. Boolean theories are specified in Table 3.1. Modal theories are defined in Table 3.4 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 complete theory $\Phi_{\mathsf{ML}}(T)$. A ppKS $\mathbf{K}$ 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 $\mathcal{OR}$ 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

Table 3.4: Definitions of (basic) ppCTL theories

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

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

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

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

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

$$\Phi^{\mathsf{I2C}\nrightarrow}_{\mathsf{ML}\subseteq}(T_{\mathcal{OR}}) = \big\{ f \wedge \neg \bigwedge \Phi^{!}_{\mathsf{BL}}(T^{f}_{\mathcal{OR}}) \rightarrow \neg \mathsf{EX} g : f, g \in F, f \neq g, f^{\uparrow} = g^{\uparrow} \big\}$$

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

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

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

$\Phi^{!}_{\mathsf{BL}}(\mathcal{OR})$ (Table 3.1). Consider an OR group G. The theory $\Phi^{!}_{\mathsf{ML}\subseteq}(\mathcal{OR})$ states that if 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 (note the finality property proven in Proposition 3.2).

The nature of the theory corresponding to $\mathcal{IN}$ is like $\mathcal{OR}$'s: it also deals only with full products. The theory $\Phi^{!}_{\mathsf{ML}\subseteq}(\mathcal{IN})$ is the modal version of the 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 (note again the finality property in PPLs).

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

Recall that the theory $\Phi^{\mathsf{I2C}}_{\mathsf{BL}}(T_{\mathcal{OR}})$ (Table 3.1) guarantees that the partial products of the PPL respect the I2C principle. However, as discussed in Section 3.2.2, transitions also have to respect this principle. The modal theory $\Phi^{\mathsf{I2C}\nrightarrow}_{\mathsf{ML}\subseteq}(T_{\mathcal{OR}})$ excludes the invalid transitions due to the I2C principle (see Table 3.4). 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 $\Phi^{\mathsf{I2C}}_{\mathsf{BL}}(T_{\mathcal{OR}})$ and $\Phi^{\mathsf{I2C}\nrightarrow}_{\mathsf{ML}\subseteq}(T_{\mathcal{OR}})$.

Recall that, according to Definition 3.5, a set of features is a valid partial product iff it satisfies the Boolean theory $\Phi_{\mathsf{BL}}(\mathbf{M})$. However, any ppKS satisfying this theory does not necessarily include all valid partial products. To ensure that the ppKS includes all partial products, we add modal theories $\Phi^{\downarrow}_{\mathsf{ML}+}(T)$ 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 the I2C principle, then there must be an immediately accessible state from P including f. The corresponding completeness theory is denoted by $\Phi_{\mathsf{ML}+}(\mathbf{M})$ and is equal to $\Phi_{\mathsf{BL}}(\mathbf{M}) \cup \Phi^{\downarrow}_{\mathsf{ML}+}(T) \cup \Phi^{\leftrightarrow}_{\mathsf{ML}+}(T_{\mathcal{OR}}, \mathcal{EX})$.

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

3.6 Other Applications of the Modal Logic View

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

3.6.1 Automated Analysis of FMs

Analysis of FMs is an important practical issue, and as industrial FMs can contain thousands of features, the analysis should be automated [BSRC10]. A big group of analysis problems over FMs rely on the Boolean semantics of FMs. For example, given an FM $\mathbf{M}$, we may be interested in checking whether $PL(\mathbf{M})$ is not empty [TC09], or whether a given set of features G is a valid full product, i.e., $G \in PL(\mathbf{M})$ [KCH$^+$90]. We may also be interested in finding the set of common (core) features among all full products, $\bigcap PL(\mathbf{M})$ [TC09], or checking whether f is a core feature, i.e., $f \in \bigcap PL(\mathbf{M})$. Specifically, an important problem is to find so called *dead* features, which do not occur in any product [KCH$^+$90]. A typical practical approach to these analysis problems is to encode the FM by a Boolean theory, and then use off-the-shelf tools like SAT-solvers [Bat05].

However, there are some other important analysis problems, in which the use of the Boolean semantics can be error-prone. For example, it is often important to know if one FM $\mathbf{M}_1$ is a *refactoring* of another FM $\mathbf{M}_2$, or a *specialization* of $\mathbf{M}_2$, or neither [TBK09]. Standard definitions of refactoring and specialization are based on semantics, which in the Boolean case gives rise to defining refactoring $\mathbf{M}_1 \simeq \mathbf{M}_2$ as $PL(\mathbf{M}_1) = PL(\mathbf{M}_2)$ and specialization $\mathbf{M}_1 \preceq \mathbf{M}_2$ as $PL(\mathbf{M}_1) \subseteq PL(\mathbf{M}_2)$. However, as we have seen above, the Boolean semantics is too poor and makes the definitions above inadequate for their goals (see the example in the introduction). Hence, in practice, to investigate refactoring and specialization, engineers should work with pairs $(PL(\mathbf{M}), \mathbf{M})$, whose second component represents the feature hierarchical structure not captured by the first component. Working with such pairs brings two issues. First, it leads to obvious maintenance problems: if one of the components changes, the user must remember to propagate the changes to the other component. Second, having a syntactical "non-Boolean" object of analysis does not allow us to use SAT (or SMT) solvers. However, the PPL semantics allows the management of both issues. As our completeness theorem shows, $PPL(\mathbf{M})$ adequately captures the feature hierarchy, and hence we can analyze a single object, $PPL(\mathbf{M})$ or, equivalently, the modal theory $\Phi_{\mathsf{ML}}(\mathbf{M})$.

Finally, there are analysis problems only addressing the hierarchy, e.g., finding the *Lowest Common Ancestor* (LCA) of a set of features in the feature tree [MWCC08]. The PPL semantics allows us to analyze such a problem by using a model checker: given a set of features G and a candidate common ancestor feature c, we need to check whether the Kripke structure $PPL(\mathbf{M})$ satisfies $\bigwedge G \rightarrow c$. This way, we could get the set of common ancestors of G. Let us denote it by C. Now, to check whether an element $l \in C$ is the LCA of G, we just need to check if $PPL(\mathbf{M})$ satisfies $l \rightarrow \bigwedge C$. Other syntactical analysis problems can be approached in the same way: an FM $\mathbf{M}$ is represented by a Kripke structure $PPL(\mathbf{M})$, the problem to be analyzed is encoded by

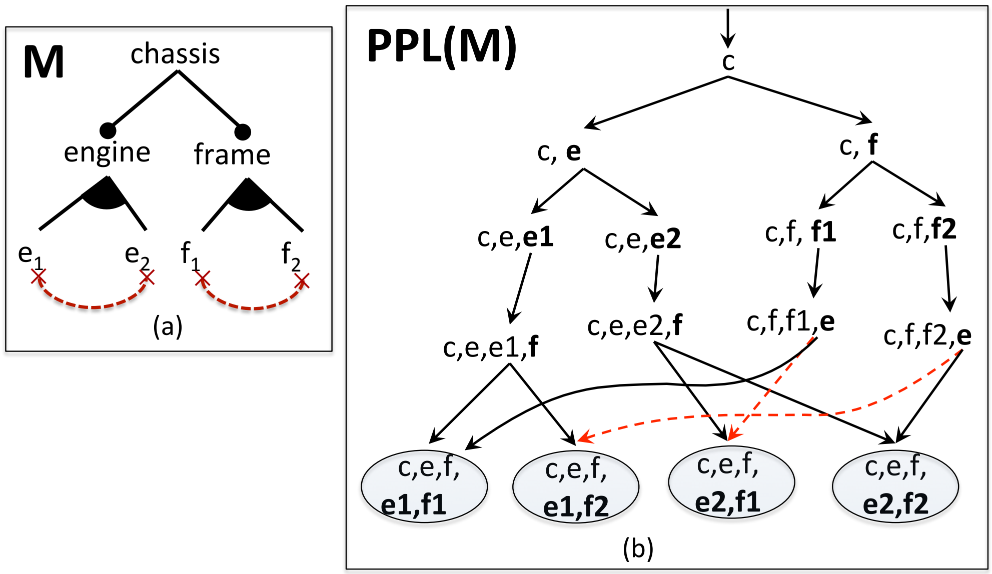


Figure 3.6: An FM of an Engine Frame (a), and its PPL (b)

a ppCTL-formula ϕ, and a model checker tool is used for checking if $PPL(\mathbf{M}) \models \phi$.

3.6.2 PL-builder vs. PL-client View

Modal properties of product lines may not be 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 important for the vendor, who should plan and provide a reasonable production of all products in the product line. For example, consider the following scenario.

Suppose we want to design a chassis with two mandatory components: an engine and a frame. An engine is of type e_1 xor e_2, and a frame is of type f_1 xor f_2, as specified in the Figure 3.6. In general, engine e_i better fits in frame f_i, $i = 1, 2$, but the frame supplier can modify the frame for an extra cost. Thus, we have four full products $P_0 \cup P_{ij}$ with $P_0 = \{c, e, f\}$ and $P_{ij} = \{e_i, f_j\}$, $i, j = 1, 2$ (c, e, and f stand for chassis, engine, and frame, resp.).

There are two ways for assembling the chassis. If we first decide on the engine type, then, for engine e_i, we may choose either to order frame f_i, or frame f_j, $j \neq i$, with a suitable modification, depending on what is cheaper (we assume that each frame type has its own supplier). Thus, from each product $P_0 \cup \{e_i\}$, $i = 1, 2$ there are two transitions as shown in Figure 3.6. However, if we first decide on the frame type, then only the engine of the respective type can be mounted on the frame, and transitions from $P_0 \cup \{f_i\}$ to $P_0 \cup \{f_i, e_j\}$ $j \neq i$ are illegal (shown dashed/red in Figure 3.6). To exclude the illegal transitions from the ppl, we need to add to the FM the following two modal CCs: $(f_i \wedge e \wedge \neg e_i) \rightarrow \mathsf{AX}\neg e_j$ for $i, j \in \{1, 2\}$ and $i \neq j$. Such constraints cannot be expressed in BL as they do not change the *set* of partial products, and only transition are affected.

3.6.3 Reverse Engineering of FMs

Reverse engineering of FMs is an active research area in feature modeling. The problem statement is as follows: given a product line, we want to build an appropriate FM representing the product line. Depending on the representation of the given product line, the current approaches are grouped into two kinds: reverse engineering of FMs from Boolean logic formulas [CW07], reverse engineering of FMs from textual descriptions of features [ASB+08, NE08]. She et al. in [SLB+11] argue that none of these approaches are complete. Indeed, the main challenge of this task is to determine an appropriate hierarchical structure of features. The Boolean logic approach is incomplete, since, as already discussed, the Boolean logic 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" [SLB+11]. To relieve the deficiencies of these approaches, the current stat-of-the-art approach [SLB+11] proposes a heuristics based approach in which both types of inputs are given as input. However, if we take the given input to be the ppCTL theory of the product line (in other words, its PPL), reverse engineering of FMs becomes simpler and more manageable. This is because the given ppCTL theory contains everything needed to build a corresponding FM. Also, our careful decomposition of an FM's structure and theories into small blocks is because it would allow better tuning of the reverse engineering process.

Chapter 4

Multiset Theory of Cardinality-Based Feature Diagrams

Basic feature modeling deals with feature "types", while we deal with feature "resources" (occurrences) in cardinality-based feature modeling. Thus, the relation between cardinality-based and basic feature modeling is roughly the relation between resources on one hand and their types on the other hand. We have already discussed two semantics for basic feature modeling, Boolean and Kripke-based, which are both "set-theoretic" (type-conscious). Moving from basic to cardinality-based feature modeling is, indeed, moving from set theory to "multiset theory" (resource-conscious). In the present chapter, we propose two multiset theories, called *flat* and *hierarchical*, for cardinality-based feature diagrams (CFDs).

The flat semantics of a CFD is the set of multisets over features satisfying the multiplicity constraints. This semantics provides a useful abstract view of the CFD, as it can address a large number of analysis questions about the CFD. However, it does not capture some other useful information such as the hierarchy of the CFD.

The hierarchical semantics of a CFD provides a faithful semantics for the CFD. This semantics is defined based on a *hierarchy of multisets* built over features. The hierarchical semantics of the CFD would be then a subset of this hierarchy. We show that the hierarchical semantics captures *all* information of the CFD so that one can retrieve the CFD from its hierarchical semantics.

The plan of this chapter is as follows. Section 4.1 gives our formal framework for the syntax of CFDs. This section also discusses the idea of flat semantics. Section 4.2 will discuss and formalize the idea of hierarchical semantics for cardinality-based feature diagrams. To this end, we propose a hierarchy of finite multistes, as a fundamental basis for formalizing hierarchical semantics. We show that the hierarchical semantics of a CFD captures all information of the CFD. Section 4.3 characterizes multisets representing hierarchical products of some CFDs. To this end, we introduce the notion of *tree-like multisets*. It is proven that a multiset is a hierarchical product of a given CFD iff it is a tree-like multiset. Section 4.4 characterizes sets of tree-like multisets representing hierarchical semantics of CFDs, namely, we show what sets of tree-like multisets are the hierarchical semantics of some CFDs. To this end, the notion of *mergeable* and *complete mergeable* tree-like multisets are introduced. We will discuss the practical application of flat and hierarchical semantics more in Section 4.5. Some examples will be provided following the formal definitions to clarify the definitions.

4.1 Cardinality-Based Feature Diagrams and their Flat Semantics

To make this section self-contained, we first provide an informal description of CFDs (see Section 2.1 for more explanation). A CFD is a tree of features in which some subsets of non-root nodes are *grouped* and other nodes are called *solitary*. In addition, non-root nodes and groups are equipped with some multiplicity constaints. In our framework, solitary nodes are derived constructs. A multiplicity constraint is usually expressed as a sequence of pairs (l, u), where l is a natural number, u is either a number or $*$ (representing an unbounded multiplicity) and $l \leq u$. We call a multiplicity constraint on a node or group a *multiplicity domain*. As an example, consider the CFD in Figure 4.1. It is a CFD over features $\mathsf{f}, \mathsf{f}_{1-6}$. G denotes a group consisting of the features $\mathsf{f}_4, \mathsf{f}_5$, and f_6, and any feature in $F \setminus \mathsf{G}$ is a solitary feature. The multiplicity domains are as follows: $(2, 3)$ on G, $(1, 2)(4, ^*)$ on f_1, $(0, 2)$ on f_2, $(3, 5)$ on f_3, and $(1, 2)$ on f_6. The multiplicity domains on the features $\mathsf{f}_{4,5}$ are both $(1, 1)$. We will use this CFD as an example to illustrate the notions discussed in this chapter.

A multiplicity domain, a sequence of intervals on natural numbers, can be expressed as a subset of natural numbers, e.g., the multiplicity domains (1,2)(4,*) and (2,3) on the feature f_1 and the group G are the sets $\mathbb{N} \setminus \{3\}$ and $\{2, 3\}$, respectively. In this chapter, we consider a multiplicity domain as a subset of natural numbers. This definition of multiplicity domains makes some further formalizations in the chapter easier to read. Note that considering any subset of natural numbers as a valid multiplicity domain makes CFDs more expressive than traditional CFDs, as not all subsets

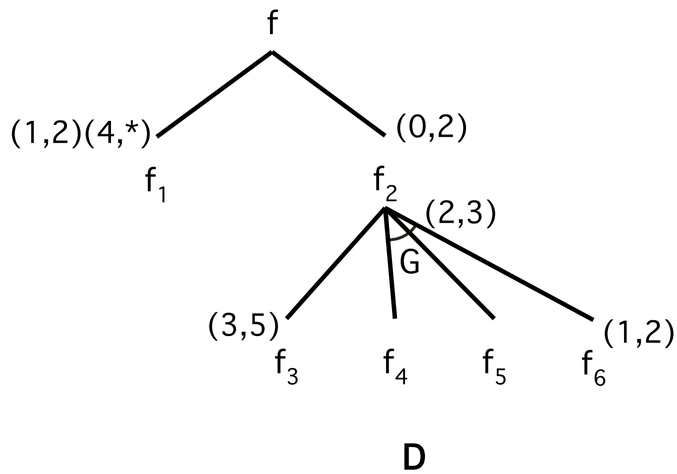


Figure 4.1: A CFD: running example

of natural numbers can be expressed as a finite sequence of intervals.[1] The following definition formalizes the syntax of CFDs.

Definition 4.1 (Cardinality-based Feature Diagrams). A *cardinality-based feature diagram* (CFD) is a 5-tuple $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$ consisting of the following components.

(i) $T = (F, r, \_^{\uparrow})$ is a *tree* with set F of nodes (called *features*), $r \in F$ is the root, and function $\_^{\uparrow}$ maps each non-root node $f \in F_{-r} \stackrel{\text{def}}{=} F \setminus \{r\}$ to its parent $f^{\uparrow}$. The inverse function that assigns to each node f the set of its children is denoted by $f_{\downarrow}$. The set of all descendants of f is denoted by $f_{\downarrow\downarrow}$.

(ii) $\mathcal{G} \subseteq 2^{F_{-r}}$ is a set of *grouped* nodes. For all $G \in \mathcal{G}$, $|G| > 1$, and all nodes in G have the same parent, denoted by $G^{\uparrow}$. All groups in $\mathcal{G}$ are disjoint, i.e., $\forall G, G' \in \mathcal{G} : (G = G') \vee (G \cap G' = \varnothing)$. The nodes that are not in a group are called *solitary* nodes. Let $\mathcal{S}$ denote the solitary nodes, i.e., $\mathcal{S} = F_{-r} \setminus \bigcup_{G \in \mathcal{G}} G$.

[1] In the next chapter, where we will propose formal language based semantics for CFMs, we will get back to the traditional definition of multiplicity domains.

(iii) $\mathcal{C} : (F_{-r} \cup \mathcal{G}) \to 2^{\mathbb{N}}$ is a total function called the *multiplicity function*. For any feature or group $e \in F_{-r} \cup \mathcal{G}$, $\mathcal{C}(e)$ represents the multiplicity constraint of e, where $\mathcal{C}(e) \neq \{0\}$ and $\mathcal{C}(e) \neq \varnothing$. In addition, for all $G \in \mathcal{G}$, $\mathcal{C}(G)$ is a finite subset of $\mathbb{N}$ and its greatest member is less than or equal $|G|$ (the number of G's members).

The class of all CFDs and all CFDs over the same set of features F are denoted by $\mathcal{D}$ and $\mathcal{D}(F)$, respectively. □

If needed, we will subscript $\mathbf{D}$'s components with index $_{\mathbf{D}}$, e.g., write $\mathcal{G}_{\mathbf{D}}$.

Remark 4.1. The original definition of CFDs in [CHE05a] has two restrictions on group multiplicities: (i) the multiplicity domain of a grouped node is always $\{1\}$ and (ii) the multiplicity domain assigned to a group is a singleton. However, we generalized CFDs in the above definition without essentially complicating the framework and enabling useful generalizations in feature modeling. □

To proceed, we first need a definition of multisets:

Definition 4.2 (Multisets). A multiset over a set A is a total function $m : A \to \mathbb{N}$, which maps an element of A to a natural number. For any $a \in A$, $m(a)$ is called the *multiplicity* of a in m. The set $\{a \in A : m(a) > 0\}$ is called *domain* of m, denoted by $dom(m)$. A multiset m is called *finite* if $dom(m)$ is finite. □

We need the *additive union* operation, denoted by $\uplus$, on multisets: $(m \uplus m')(f) = m(f) + m'(f)$. We write $m = \lceil a_1^{n_1}, a_2^{n_2}, \ldots \rceil$ to explicitly show the elements of a multiset m, where $n_i = m(a_i)$ for any $a_i \in dom(m)$. The empty multiset is denoted by $\varnothing$.

An instance of a given CFD is commonly considered as a multiset[2] of features

[2] See the formal definition of multisets in Definition 4.2.

satisfying the constraints of the CFD [CHE05a, MSDLM11]. We call such multisets *flat products* of the CFD. The flat products of a given CFD is formalized in [CHE05a] via context-free grammars (see Section 6.2). However, as far as we know, it never gained a direct definition. This is an important issue, as verification of a proposed formulation of products without having a formal definition of them is impossible. We will formalize flat products later in this section. A flat product of a CFD is a multiset of features satisfying the following constraints.

(a) The root is always included in the multiset with multiplicity 1: the multiplicity of the feature f in any valid flat product of the CFD in Figure 4.1 is always 1.

(b) If a non-root feature is included in the multiset then its parent must be included too, e.g., the presence of the node f_3 in a flat product of the CFD in Figure 4.1 implies the presence of the node f_2.

(c) A valid multiplicity of a non-root feature is given based on its multiplicity domain and the multiplicity of its parent feature in the flat product, e.g., if the multiplicity of f_2 in a flat product of the CFD in Figure 4.1 is 2 then f_3's multiplicity must be at least 6 and at most 10 in the flat product. In general, for non-root features f included in the flat product, there must be a multiplicity c in f's multiplicity domain such that its multiplicity in the flat product is equal to the product of its parent's ($f^{\uparrow}$) multiplicity and c.

(d) If the parent of a *mandatory feature* (a solitary feature with lower bound multiplicity greater than 0) is included in a flat product then it must be included too, e.g., the presence of f_2 in a flat product implies the presence of f_3 in the flat product.

(e) If a parent of a grouped set of features is included in a flat product then the presence of the grouped features must satisfy the associated group multiplicity constraint, e.g., the presence of the feature f_2 in a flat product implies the presence of 2 or 3 of the features f_4, f_5, and f_6 in the flat product.

In our running example (Figure 4.1), the following multisets are valid flat products of the CFD:

— $m_1 = \lceil \mathsf{f}, {\mathsf{f}_1}^5 \rceil$ (We consider 1 as the default multiplicity of an element in a multiset. This is why the multiplicity of f is not written.): the number of occurrences of f_1 is 5, which does not violate the multiplicity constraint ($\mathcal{C}(\mathsf{f}_1) = \mathbb{N} \setminus \{3\}$) on the feature. The presence of f_2 in a flat product is optional (the lower bound of the multiplicity domain of f_2 is 0). In this example, f_2 is excluded and so are all its children.

— $m_2 = \lceil \mathsf{f}, {\mathsf{f}_1}^5, \mathsf{f}_2, {\mathsf{f}_3}^3, \mathsf{f}_4, \mathsf{f}_5 \rceil$: unlike m_1, f_2 is included in the product with 1 occurrence. The multiplicity domain on f_3 says that the number of its occurrences must be 3, 4, or 5 for each occurrence of f_2 (its parent). The multiplicity of f_3 is 3, which satisfies the constraint. The group multiplicity on G indicates that 2 or 3 of the features f_4, f_5, and f_6 must be included in the product: the two features f_4 and f_5 are included in m_2, each with one occurrence, which satisfy the multiplicity constraints on the features.

— $m_3 = \lceil \mathsf{f}, {\mathsf{f}_1}^5, {\mathsf{f}_2}^2, {\mathsf{f}_3}^6, {\mathsf{f}_4}^2, {\mathsf{f}_5}^2 \rceil$: the difference between m_3 and m_2 comes from their multiplicities for f_2. The multiplicity of f_2 in this example is twice its multiplicity in m_2. This is why the multiplicity of the features f_3, f_4, and f_5 have been multiplied by 2.

We call the set of flat products of a given CFD the *flat semantics* of the CFD. Note that the flat semantics of our running example is an infinite set. The following definition formalizes the notion of flat products.

Definition 4.3 (Flat Products). Let $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$ be a CFD. A multiset m over F is called a *flat product* of $\mathbf{D}$ if the following conditions hold:

(i) $m(r) = 1$,

(ii) $\forall f \in F_{-r} : f \in dom(m) \implies (\exists c \in \mathcal{C}(f) : m(f) = c \times m(f^{\uparrow}))$,

(iii) $\forall f \in \mathcal{S} : 0 \notin \mathcal{C}(f) \wedge m(f^{\uparrow}) > 0 \implies m(f) > 0$,

(iv) $\forall G \in \mathcal{G} : (m(G^{\uparrow}) > 0) \implies (|dom(m) \cap G| \in \mathcal{C}(G))$.

The set of flat products of $\mathbf{D}$, denoted by $\mathcal{P}^{\text{flat}}(\mathbf{D})$, is called the *flat semantics* of $\mathbf{D}$.

□

Let us see how the above definition formalizes the description of flat products (see page 63). The condition "a" (the root is always included in the multiset with multiplicity 1) is directly formalized in Definition 4.3(i). Definition 4.3(ii) satisfies the conditions "b" (if a non-root feature is included in the multiset then its parent must be included too): suppose that a non-root feature f is included in a flat product m without its parent. This implies that $m(f^{\uparrow}) = 0$ and $m(f) > 0$, which violates Definition 4.3(ii). Definition 4.3(ii) also formalizes the condition "c" (a valid multiplicity of a non-root feature is given based on its multiplicity domain and the multiplicity of its parent feature in the flat produc). Indeed, "b" is a consequence of "c". Definition 4.3(iii) formalizes the condition "d" (if the parent of a mandatory feature is included in a flat product then it must be included too). Note that $0 \notin \mathcal{C}(f)$ for a feature f means that f is a mandatory subfeature of its parent. Definition 4.3(iv) formalizes the condition "e" (if a parent of a group is included in a flat product then the presence of the grouped features must satisfy the associated group multiplicity constraint).

We also provide a recursive definition of flat products in Lemma 4.1. To this end, we first need the following notion.

Definition 4.4 (Grouped Flat Products). Let $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$ be a CFD and $G = \{f_1, f_2, \ldots, f_k\} \in \mathcal{G}$ for $k \in \mathbb{N}$. A multiset m over F is a *grouped flat product* associated with G if there exist $c \in \mathcal{C}(G)$, $c_i \in \mathcal{C}(f_i)$, $g_i \in \{0, 1\}$, and $m_i \in \mathcal{P}^{\text{flat}}(\mathbf{D}^{f_i})$ for any $1 \leq i \leq k$ such that

$$m = \biguplus_{1 \leq i \leq k} m_i^{c_i \times g_i}, \text{ and } \sum_{1 \leq i \leq k} g_i = c$$

The set of all grouped flat products associated with G is denoted by $\mathcal{P}^{\text{flat}}(\mathbf{D}, G)$. □

Consider the group $\mathsf{G} = \{\mathsf{f}_4, \mathsf{f}_5, \mathsf{f}_6\}$ in our running example (Figure 4.1). $\mathcal{P}^{\text{flat}}(\mathbf{D}, \mathsf{G})$ consists of the following elements:

— $g_1 = \lceil \mathsf{f}_4, \mathsf{f}_5, \mathsf{f}_6 \rceil$: All elements of the group are picked, which satisfies the group's multiplicity domain $\{2, 3\}$. The multiplicity of f_6 is 1, which is in its multiplicity domain $\{1, 2\}$. Note that the multiplicities of both features f_4 and f_5 are always 1, if included.

— $g_2 = \lceil \mathsf{f}_4, \mathsf{f}_5, {\mathsf{f}_6}^2 \rceil$: This is the same as g_1 except that 2 is chosen from f_6's multiplicity domain.

— $g_3 = \lceil \mathsf{f}_4, \mathsf{f}_5 \rceil$: The two features f_4 and f_5 have been chosen, which satisfies the group multiplicity domain.

— $g_4 = \lceil \mathsf{f}_4, \mathsf{f}_6 \rceil$: This is the same as g_3 except that f_5 is replaced by f_6 with 1 occurrence.

— $g_5 = \lceil \mathsf{f}_4, {\mathsf{f}_6}^2 \rceil$: This is the same as g_4 except that 2 is chosen from f_6's multiplicity domain.

— $g_6 = \lceil \mathsf{f}_5, \mathsf{f}_6 \rceil$: This is the same as g_4 except that f_4 is replaced by f_5.

— $g_7 = \lceil \mathsf{f}_5, {\mathsf{f}_6}^2 \rceil$: This is the same as g_5 except that f_4 is replaced by f_5.

The following theorem provides a recursive presentation of flat semantics.

Lemma 4.1. Given a CFD $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$, for any multiset m over F: $m \in \mathcal{P}^{\text{flat}}(\mathbf{D})$ iff m satisfies the following conditions:

(i) $m(r) = 1$,

(ii) $\forall f \in \mathcal{S} \cap r_{\downarrow},\ \exists c \in \mathcal{C}(f),\ \exists n \in \mathcal{P}^{\text{flat}}(\mathbf{D}^f),\ \forall e \in \mathit{dom}(n) : m(e) = c \times n(e)$.

(iii) $\forall G \in \mathcal{G} \cap 2^{r_{\downarrow}},\ \exists n \in \mathcal{P}^{\text{flat}}(\mathbf{D}, G),\ \forall e \in \mathit{dom}(n) : m(e) = n(e)$. □

The following statement is a corollary of the above lemma.

Corollary 4.1. Given a CFD $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$, a flat product $m \in \mathcal{P}^{\text{flat}}(\mathbf{D})$ satisfies the following conditions:

(i) $\forall f \in \mathcal{S},\ \exists c \in \mathcal{C}(f),\ \exists n \in \mathcal{P}^{\text{flat}}(\mathbf{D}^f),\ \forall e \in \mathit{dom}(n) : m(e) = n(e) \times c \times m(f^{\uparrow})$.

(ii) $\forall G \in \mathcal{G},\ \exists n \in \mathcal{P}^{\text{flat}}(\mathbf{D}, G),\ \forall e \in \mathit{dom}(n) : m(e) = n(e) \times m(G^{\uparrow})$. □

The flat semantics of a CFD provides a useful abstract view of the CFD, as it can address a large number of analysis questions about the diagram. However, it is a poor abstract view, as it does not capture some other useful information about the diagram, such as the hierarchical structure. For an example, consider two different CFDs $\mathbf{D}_1$ and $\mathbf{D}_2$ in Figure 4.2. They are equivalent in the flat semantics, since they represent the same flat semantics $\{\lceil \mathsf{f}, \mathsf{f}_2, \mathsf{f}_3 \rceil,\ \lceil \mathsf{f}, \mathsf{f}_2{}^2, \mathsf{f}_3{}^2 \rceil\}$.

4.2 Hierarchical Semantics

Two types of information are lost in the flat semantics of a CFD: the *tree structure*, and the *feature's types* (grouped or solitary). For an example, given a CFD, we cannot address the following questions via the CFD's flat semantics: What are the

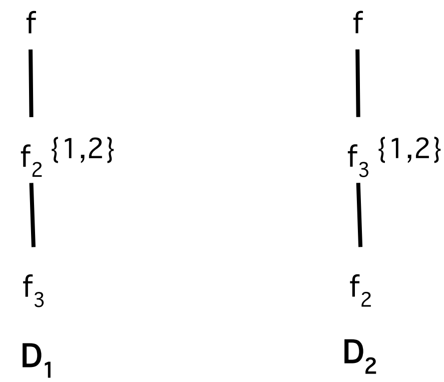


Figure 4.2: Two different CFDs with the same flat semantics

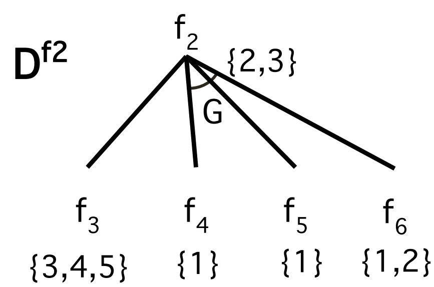


Figure 4.3: Diagram induced by a node: an example

subfeatures of a given feature? Decide whether a given feature is solitary or not? In this section, we address this problem with another semantics for CFDs, called *hierarchical semantics*.

We need the notion of *induced diagrams* defined in Definition 4.5 to continue this discussion. For a relation $R \subseteq B \times C$ and a set A, the notation $R\big|_A$ is used to denote the restriction of R to A.

Definition 4.5 (Diagram Induced by Nodes). Let $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$ be a CFD and $f \in F$. The *CFD induced by* f is a CFD $\mathbf{D}^f = (F', f, \_^{\uparrow}\big|_{F'}, \mathcal{G}', \mathcal{C}')$, where $F' = f_{\downarrow\downarrow} \cup \{f\}$, $\mathcal{G}' = \mathcal{G} \cap 2^{F'}$, and $\mathcal{C}' = \mathcal{C}|_{F' \cup \mathcal{G}'}$, i.e., its tree is the tree under f in $\mathbf{D}$'s tree and all other components are inherited from $\mathbf{D}$. $\square$

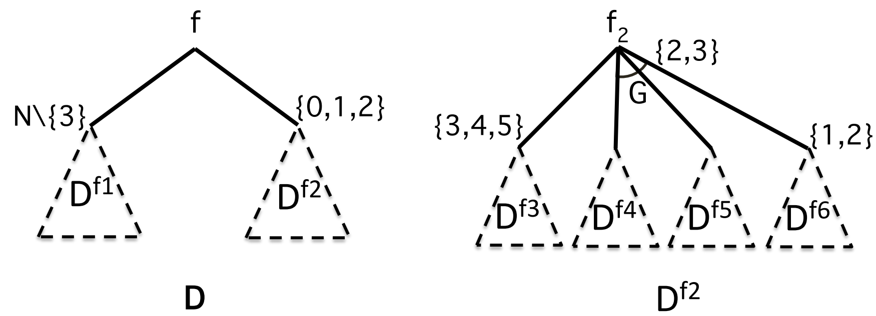


Figure 4.4: The representation of Figure 4.1 in terms of induced diagrams

For an example, $\mathbf{D}^{\mathsf{f}_2}$ in Figure 4.3 is the diagram induced by f_2 of the CFD $\mathbf{D}$ in Figure 4.1.

In the hierarchical semantics of a given CFD, the multiplicity domain of a solitary feature is considered as a multiplicity constraint on its corresponding induced diagram. Looking at Figure 4.4 (left), which represents the CFD $\mathbf{D}$ in Figure 4.1 in terms of induced diagrams: The root feature f has two children labeled by $\mathbf{D}^{\mathsf{f}_1}$ (diagram induced by f_1) and $\mathbf{D}^{\mathsf{f}_2}$ (diagram induced by f_2) with multiplicity domains $\mathbb{N} \setminus \{3\}$ and $\{0, 1, 2\}$, respectively. In this way, a hierarchical product of $\mathbf{D}$ is considered as a multiset $\lceil \mathsf{f}, h_1^{c_1}, h_2^{c_2} \rceil$, where h_1 and h_2 are a hierarchical product of $\mathbf{D}^{\mathsf{f}_1}$ and a hierarchical product of $\mathbf{D}^{\mathsf{f}_2}$, respectively, and $c_1 \in \mathbb{N} \setminus \{3\}$ (the multiplicity domain of f_1), $c_2 \in \{0, 1, 2\}$ (the multiplicity domain of f_2). Since $\mathbf{D}^{\mathsf{f}_1}$ is a singleton tree, h_1 is always equal to $\lceil \mathsf{f}_1 \rceil$.

Now, consider $\mathbf{D}^{\mathsf{f}_2}$ (the diagram induced by f_2) shown in Figure 4.4 (right). The feature f_2 has four subfeatures f_{3-6}, where f_3 is a solitary feature and the others are grouped all together under G. Thus, f_2 has two subelements: a solitary feature f_3 and a group G. To distinguish between groups and solitary features, we introduce a notion, called *grouped hierarchical products*. This way, a hierarchical product of $\mathbf{D}^{\mathsf{f}_2}$ would be a multiset $\lceil \mathsf{f}_2, h_3^{c_3}, h_G \rceil$, where h_3 and h_G are a hierarchical product of

$\mathbf{D}^{\mathsf{f}_3}$ and a grouped hierarchical product of G, respectively, and $c_3 \in \{3, 4, 5\}$ (the multiplicity domain of f_3). Since $\mathbf{D}^{\mathsf{f}_3}$ is a singleton CFD, h_3 is always equal to $\lceil \mathsf{f}_3 \rceil$. We describe how to get a grouped hierarchical product for G in the following.

Suppose that we choose f_4 and f_5 from the group G in our configuration. The corresponding grouped hierarchical product would be a multiset $\lceil h_4, h_5 \rceil$, where h_4 and h_5 are, respectively, a hierarchical product of $\mathbf{D}^{\mathsf{f}_4}$ and a hierarchical product of $\mathbf{D}^{\mathsf{f}_5}$. The multiplicities of h_4 and h_5 are both 1, as the multiplicity domains of f_4 and f_5 are both $\{1\}$. Since $\mathbf{D}^{\mathsf{f}_4}$ and $\mathbf{D}^{\mathsf{f}_5}$ are singleton trees, h_4 and h_5 would be always $\lceil \mathsf{f}_4 \rceil$ and $\lceil \mathsf{f}_5 \rceil$, respectively. Now, let us replace f_5 by f_6 in our group. Then, a grouped hierarchical product would be a multiset $\lceil h_4, h_6^{c_6} \rceil$, where h_6 is a hierarchical product of $\mathbf{D}^{\mathsf{f}_6}$ and $c_6 \in \{1, 2\}$ (the multiplicity domain of f_6). h_6 would always be equal to $\lceil \mathsf{f}_6 \rceil$, as f_6 is a leaf. This way, we "explicitly" distinguish between grouped and solitary features.

According to discussion above, a hierarchical product of the CFD in Figure 4.1 would be a multiset $\lceil \mathsf{f}, \lceil \mathsf{f}_1 \rceil^{c_1}, \lceil \mathsf{f}_2, \lceil \mathsf{f}_3 \rceil^{c_3}, \lceil \lceil \mathsf{f}_4 \rceil^{c_4 \times g_1}, \lceil \mathsf{f}_5 \rceil^{c_5 \times g_2}, \lceil \mathsf{f}_6 \rceil^{c_6 \times g_3} \rceil \rceil^{c_2} \rceil$, where c_{1-6} are valid multiplicities of f_{1-6}, respectively, and $g_{1-3} \in \{0, 1\}$ such that $2 \leq g_1 + g_2 + g_3 \leq 3$. The condition $2 \leq g_1 + g_2 + g_3 \leq 3$ ensures that the group multiplicity $\{2, 3\}$ is satisfied. Note that an element in a multiset with multiplicity 0 means that the element does not belong to the domain of the multiset, e.g., $\lceil \mathsf{f}, \lceil \mathsf{f}_1 \rceil^{c_1}, \lceil \mathsf{f}_2, \lceil \mathsf{f}_3 \rceil^{c_3}, \lceil \lceil \mathsf{f}_4 \rceil^{0}, \lceil \mathsf{f}_5 \rceil^{c_5}, \lceil \mathsf{f}_6 \rceil^{c_6} \rceil \rceil^{c_2} \rceil = \lceil \mathsf{f}, \lceil \mathsf{f}_1 \rceil^{c_1}, \lceil \mathsf{f}_2, \lceil \mathsf{f}_3 \rceil^{c_3}, \lceil \lceil \mathsf{f}_5 \rceil^{c_5}, \lceil \mathsf{f}_6 \rceil^{c_6} \rceil \rceil^{c_2} \rceil$.

In our running example (Figure 4.1), the following multisets are valid hierarchical products of the CFD:

— $h_1 = \lceil \mathsf{f}, \lceil \mathsf{f}_1 \rceil^5 \rceil$: We chose the multiplicity 5 from the multiplicity domain of f_1. Since f_1 is a leaf node, $\mathbf{D}^{\mathsf{f}_1}$ represents a single hierarchical product $\lceil \mathsf{f}_1 \rceil$. Since the lower bound of the multiplicity domain of f_2 is 0, it is completely safe not to include its corresponding hierarchical product, as done in h_1.

— $h_2 = \lceil \mathsf{f}, \lceil \mathsf{f}_1 \rceil^5, \lceil \mathsf{f}_2, \lceil \mathsf{f}_3 \rceil^3, \lceil \lceil \mathsf{f}_4 \rceil, \lceil \mathsf{f}_5 \rceil \rceil \rceil \rceil$: Unlike h_1, a hierarchical product of $\mathbf{D}^{\mathsf{f}_2}$ is included: $n = \lceil \mathsf{f}_2, \lceil \mathsf{f}_3 \rceil^3, \lceil \lceil \mathsf{f}_4 \rceil, \lceil \mathsf{f}_5 \rceil \rceil \rceil$. $\mathbf{D}^{\mathsf{f}_2}$ has a group G and a solitary node f_3. The multiplicity 3 is chosen from the multiplicity domain of f_3, i.e., $\lceil \mathsf{f}_3 \rceil$ is an element of n with multiplicity 3. The multiplicity 2 is chosen from the group's multiplicity domain. The two elements f_4 and f_5 each with 1 occurrence are included in the corresponding grouped hierarchical product $\lceil \lceil \mathsf{f}_4 \rceil, \lceil \mathsf{f}_5 \rceil \rceil$.

— $h_3 = \lceil \mathsf{f}, \lceil \mathsf{f}_1 \rceil^5, \lceil \mathsf{f}_2, \lceil \mathsf{f}_3 \rceil^3, \lceil \lceil \mathsf{f}_4 \rceil, \lceil \mathsf{f}_5 \rceil \rceil \rceil^2 \rceil$: This multiset and h_2 differ in the f_2's multiplicities. The multiplicity of f_2 in this example is two times greater than its multiplicity in h_2. This is why we have two occurrences of $\lceil \mathsf{f}_2, \lceil \mathsf{f}_3 \rceil^3, \lceil \lceil \mathsf{f}_4 \rceil, \lceil \mathsf{f}_5 \rceil \rceil \rceil$ (a hierarchical product of $\mathbf{D}^{\mathsf{f}_2}$).

Consider again the CFDs $\mathbf{D}_1$ and $\mathbf{D}_2$ in Figure 4.2. Unlike their flat semantics, their hierarchical semantics capture the differences: $\mathbf{D}_1$ contains the two hierarchical products $\lceil \mathsf{f}, \lceil \mathsf{f}_2, \lceil \mathsf{f}_3 \rceil \rceil \rceil$ and $\lceil \mathsf{f}, \lceil \mathsf{f}_2, \lceil \mathsf{f}_3 \rceil \rceil^2 \rceil$, while $\mathbf{D}_2$ contains $\lceil \mathsf{f}, \lceil \mathsf{f}_3, \lceil \mathsf{f}_2 \rceil \rceil \rceil$ and $\lceil \mathsf{f}, \lceil \mathsf{f}_3, \lceil \mathsf{f}_2 \rceil \rceil^2 \rceil$ as its hierarchical products.

We define a hierarchy of multisets over a set of urelements,[3] which will be a fundamental basis for formalizing the hierarchical semantics of CFDs. Since the set of features in a CFD is a finite set, we will always deal with finite multisets. Let $\mathcal{MS}(A)$ denote the class of all finite multisets over A.

Definition 4.6 (A Hierarchy of finite Multisets). For every nonempty set of

[3] An *urlement* is an object, which may be an element of a set, but it is not a set.

urelements A, we define a *hierarchy* $\mathcal{H}(A)$ of multisets as follows:

$$\mathcal{H}_1(A) = \mathcal{MS}(A), \quad \dots \quad \mathcal{H}_{n+1} = \mathcal{MS}(A \cup \bigcup_{0 \leq i \leq n} \mathcal{H}_i), \quad \dots .$$

$$\mathcal{H}(A) = \bigcup_{i \geq 1} \mathcal{H}_i(A)$$

The *rank of a multiset* $m \in \mathcal{H}(A)$, denoted by $rank(m)$, is equal to the least number n such that $m \in \mathcal{H}_n(A)$. Any multiset with rank 1 is called a *flat multiset* over A. □

As an example, consider the multisets $m_1 = \lceil a^3, b^3 \rceil$, $m_2 = \lceil a^2, \lceil a^2, b^3 \rceil, \lceil b \rceil^4 \rceil$, and $m_3 = \lceil a^{10}, \lceil a^2, b^3 \rceil^3, \lceil b \rceil^4, \lceil \lceil a \rceil \rceil \rceil$ in $\mathcal{H}(\{a, b\})$. We then would have: $rank(m_1) = 1$, $rank(m_2) = 2$, and $rank(m_3) = 3$ and $m_1 \uplus m_2 = \lceil a^5, b^3, \lceil a^2, b^3 \rceil, \lceil b \rceil^4 \rceil$.

Now, we are at the point where we can formalize hierarchical products of CFDs. Consider a CFD $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C}) \in \mathcal{D}(F)$. Suppose that r has n solitary subfeatures $s_1, \dots, s_n$ and k groups $G_1, \dots, G_k$. According to our informal description of hierarchical products, any multiset $m \in \mathcal{H}(F)$ is a hierarchical product of $\mathbf{D}$ if its domain consists of (i) r with 1 occurrence, (ii) a hierarchical product of $\mathbf{D}^{s_i}$ (diagram induced by s_i) with a multiplicity $c_i \in \mathcal{C}(s_i)$ for any $1 \leq i \leq n$, (iii) a grouped hierarchical product of G_j with multiplicity 1 for any $1 \leq j \leq k$. Hierarchical products and grouped hierarchical products are formalized in Definitions 4.7 and 4.8, respectively.

Definition 4.7 (Hierarchical Products). Given a CFD $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$, the set of $\mathbf{D}$'s *hierarchical products*, denoted by $\mathcal{P}(\mathbf{D})$, is defined as follows: For any multiset $m \in \mathcal{H}(F)$, $m \in \mathcal{P}(\mathbf{D})$ iff it satisfies the following conditions:

(i) $m(r) = 1$.

(ii) $\forall f \in \mathcal{S} \cap r_{\downarrow}, \exists c \in \mathcal{C}(f), \exists n \in \mathcal{P}(\mathbf{D}^f) : m(n) = c$.

(iii) $\forall G \in \mathcal{G} \cap 2^{r_\downarrow}, \exists n \in \mathcal{P}(\mathbf{D}, G) : m(n) = 1$.

(see Definition 4.8 for the definition of $\mathcal{P}(\mathbf{D}, G)$)

$\mathcal{P}(\mathbf{D})$ is called the *hierarchical semantics* of $\mathbf{D}$. $\square$

Definition 4.8 provides a definition for *grouped hierarchical products*. Consider a group with n elements $\{f_1, \ldots, f_n\}$ whose group multiplicity domain is denoted by C (note that $\forall c \in C : 1 \le c \le n$). A hierarchical product of this group would be a multiset $\lceil f_1^{c_1 \times g_1}, \ldots, f_n^{c_n \times g_n} \rceil$, where c_i is a valid multiplicity for feature f_i and $g_i \in \{0, 1\}$ ($\forall 1 \le i \le n$) such that $g_1 + \ldots + g_n \in C$.

Definition 4.8 (Grouped Hierarchical Products). Let $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$ be a CFD and $G = \{f_1, f_2, \ldots, f_k\} \in \mathcal{G}$ for some k. A *grouped hierarchical product* corresponding to G is a multiset $m \in \mathcal{H}(F)$ such that for all $1 \le i \le k$, there exist $c \in \mathcal{C}(G)$, $c_i \in \mathcal{C}(f_i)$, $g_i \in \{0, 1\}$, $m_i \in \mathcal{P}(\mathbf{D}^{f_i})$, and

(i) $dom(m) = \{m_i : g_i = 1\}$,

(ii) $\forall 1 \le i \le k : m(m_i) = c_i \times g_i$,

(iii) $g_1 + \ldots + g_k = c$.

The set of grouped hierarchical products associated with G is denoted by $\mathcal{P}(\mathbf{D}, G)$. $\square$

The following theorem is important, as it shows that hierarchical semantics provides a faithful semantics for CFDs. In Section 4.4, we will characterize hierarchical semantics of CFDs.

Theorem 4.1. Given two CFDs $\mathbf{D}$ and $\mathbf{D}'$, $(\mathcal{P}(\mathbf{D}) = \mathcal{P}(\mathbf{D}')) \iff (\mathbf{D} = \mathbf{D}')$. $\square$

The above theorem shows that, unlike the flat semantics, the hierarchical semantics of a given CFD captures all information of the CFD. However, the cardinalities of the hierarchical semantics and flat semantics of a given CFD are the same, i.e., there is a bijection between the set of hierarchical products and the set of flat products for a fixed CFD. This is shown in Theorem 4.2. Before getting to this formally, we first need the following notions.

The domain of a multiset with rank greater than 1 includes some multisets. For example consider the multiset $m = \lceil a, b, \lceil c, \lceil d, e \rceil \rceil \rceil \in \mathcal{H}_3(\{a, b, c, d, e\})$. The domain of this multiset includes the multiset $i_1 = \lceil c, \lceil d, e \rceil \rceil$. The domain of i_1 itself includes the multiset $i_2 = \lceil d, e \rceil$ whose domain is a set urelements. We call i_1 and i_2 the multiset ingredients of m.

Definition 4.9 (Multiset Ingredients of Multisets). Given a multiset $m \in \mathcal{H}(A)$ for some A, $\mathsf{MultIng}(m)$ is the smallest set of multisets in $\mathcal{H}(A)$ such that

(i) $\{n \in dom(m) : rank(n) \geq 1\} \subset \mathsf{MultIng}(m)$,

(ii) $\forall n \in \mathsf{MultIng}(m) : \mathsf{MultIng}(n) \subset \mathsf{MultIng}(m)$.

The multiplicity of a multiset $n \in \mathsf{MultIng}(m)$ in m is denoted by $\#_m(n)$. □

The following definition formalizes a notion called the *flat multiplicity* of an urelement in a multiset. An illustrating example follows the definition.

Definition 4.10 (Flat Multiplicities and Flattening). Let $m \in \mathcal{H}(A)$ for a set A of urelements. The *flat multiplicity* of an element is defined by a function $\#_{m,A} : A \to \mathbb{N}$ as $\#_{m,A}(a) = m(a) + \sum_{e \in \mathsf{MultIng}(m)} \#_{m,A}(e)$.

We define a function $\mathsf{flat}_A : \mathcal{H}(A) \to \mathcal{H}_1(A)$, which maps a given multiset $m \in \mathcal{H}(A)$ to a flat multiset as follows. For any $m \in \mathcal{H}(A) : \mathsf{flat}_A(m)(a) = \#_{m,A}(a)$. We say that $\mathsf{flat}_A(m)$ *flattens* m. □

Consider again the multiset $m = \lceil a^2, b^2, \lceil a, b \rceil^4, \lceil a^8, \lceil a \rceil^7, \lceil a^5, b^3 \rceil^3 \rceil \rceil$. The flat multiplicities of a and b are 36 and 15, respectively. Thus, $\mathsf{flat}_{\{a,b\}}(m) = \lceil a^{36}, b^{15} \rceil$.

The following theorem (Theorem 4.2) shows that the restriction of the flattening function to the domain of the hierarchical semantics of a given CFD provides a bijection between the hierarchical semantics and the flat semantics of the CFD. Consider the hierarchical products $h_1 = \lceil \mathsf{f}, \lceil \mathsf{f}_1 \rceil^5 \rceil$, $h_2 = \lceil \mathsf{f}, \lceil \mathsf{f}_1 \rceil^5, \lceil \mathsf{f}_2, \lceil \mathsf{f}_3 \rceil^3, \lceil \lceil \mathsf{f}_4 \rceil, \lceil \mathsf{f}_5 \rceil \rceil \rceil \rceil$, and $h_3 = \lceil \mathsf{f}, \lceil \mathsf{f}_1 \rceil^5, \lceil \mathsf{f}_2, \lceil \mathsf{f}_3 \rceil^3, \ \lceil \lceil \mathsf{f}_4 \rceil, \lceil \mathsf{f}_5 \rceil \rceil \rceil^2 \rceil$ of the CFD in Figure 4.1 (see page 70). Flattening them, we obtain $m_1 = \lceil \mathsf{f}, {\mathsf{f}_1}^5 \rceil$, $m_2 = \lceil \mathsf{f}, {\mathsf{f}_1}^5, \mathsf{f}_2, {\mathsf{f}_3}^3, \mathsf{f}_4, \mathsf{f}_5 \rceil$, and $m_3 = \lceil \mathsf{f}, {\mathsf{f}_1}^5, {\mathsf{f}_2}^2, {\mathsf{f}_3}^6, {\mathsf{f}_4}^2, \ {\mathsf{f}_5}^2 \rceil$, respectively, which are flat products of the CFD (see page 64).

Theorem 4.2. For any CFD $\mathbf{D} \in \mathcal{D}(F)$, the function $\mathsf{flat}_F\big|_{\mathcal{P}(\mathbf{D})}$, i.e., the restriction of flat_F to the subdomain $\mathcal{P}(\mathbf{D})$, provides a bijection from $\mathcal{P}(\mathbf{D})$ to $\mathcal{P}^{\text{flat}}(\mathbf{D})$. $\square$

4.3 Characterization of Hierarchical Products

In this section, we characterize the domain of multisets that can be hierarchical products of some CFDs. To this end, we define a notion called *tree-like multisets*.

Definition 4.11 (Tree-like Multisets). Given a set of urelements A, the set of *tree-like multisets* over A, denoted by $\mathcal{TH}(A)$, is inductively defined as follows:

(i) $\lceil a \rceil \in \mathcal{TH}(A)$, $\forall a \in A$.

(ii) $t_1 \uplus \lceil t_2^n \rceil \in \mathcal{TH}(A)$, $\forall t_1, t_2 \in \mathcal{TH}(A), \forall n \in \mathbb{N}$, if

$dom(\mathsf{flat}_A(t_1)) \cap dom(\mathsf{flat}_A(t_2)) = \varnothing$.

(iii) $t \uplus \lceil \lceil \ t_1^{n_1}, \ldots, t_k^{n_k} \ \rceil \rceil \in \mathcal{TH}(A)$, $\forall t, t_1, \ldots t_k \in \mathcal{TH}(A), \forall n_1, \ldots, n_k \in \mathbb{N}$, if

$\forall 1 \leq i \leq k : dom(\mathsf{flat}_A(t)) \cap dom(\mathsf{flat}_A(t_i)) = \varnothing$, and

$\forall 1 \leq i, j \leq k : (i \neq j) \implies (dom(\mathsf{flat}_A(t_i)) \cap dom(\mathsf{flat}_A(t_j)) = \varnothing)$. $\square$

For example, the following multisets in $\mathcal{H}(\{a, b, c\})$ are tree-like multisets:

– $t_1 = \lceil a \rceil$, $t_2 = \lceil b \rceil$, and $t_3 = \lceil c \rceil$,

– $t_4 = t_1 \uplus \lceil t_2^6 \rceil = \lceil a, \lceil b \rceil^6 \rceil$, $t_5 = t_3 \uplus \lceil t_4 \rceil = \lceil c, \lceil a, \lceil b \rceil \rceil^6 \rceil$,

– $t_6 = t_1 \uplus \lceil \lceil t_2^2, t_3 \rceil \rceil = \lceil a, \lceil \lceil b \rceil^2, \lceil c \rceil \rceil \rceil$.

The following multisets are not valid tree-like multisets:

– $n_1 = \lceil a, b \rceil$,

– $n_2 = \lceil a^3, \lceil b \rceil^6 \rceil$,

– $n_3 = \lceil a, \lceil \lceil b, c \rceil^2 \rceil \rceil$.

Definition 4.12 (Groups of Tree-like Multisets). Given a set A of urelements, $t_1, \ldots t_k \in \mathcal{TH}(A), n_1, \ldots, n_k \in \mathbb{N}$, the multiset $\lceil \lceil t_1^{n_1}, \ldots, t_k^{n_k} \rceil \rceil \in \mathcal{H}(A)$ is called a *group of tree-like multiset* over A if

$\forall 1 \leq i, j \leq k : (i \neq j) \implies (dom(\mathsf{flat}_A(t_i)) \cap dom(\mathsf{flat}_A(t_j)) = \varnothing)$.

Any element of the domain of a group tree-like multiset is called a *grouped tree-like multiset.* □

The multiset $\lceil \lceil b \rceil^2, \lceil c, [a]^3 \rceil \rceil$ is an example of a group of tree-like multiset over $\{a, b, c\}$. As noticed, the domain of a tree-like multiset includes a unique urelement with multiplicity 1. We call this element the *root* of the tree-like multiset, formalized in the following definition.

Definition 4.13 (Roots of Tree-like Multisets). Given a set A, we define a function $root : \mathcal{TH}(A) \to A$, as follows:

(i) $root(\lceil a \rceil) = a$, for any $a \in A$.

(ii) $root(t_1 \uplus \lceil t_2^n \rceil) = root(t_1)$ for any $t_1, t_2 \in \mathcal{TH}(A), n \in \mathbb{N}$ satisfying the conditions in Definition 4.11(ii).

(iii) $root(t \uplus \lceil\ t_1^{n_1}, \ldots, t_k^{n_k}\ \rceil) = root(t)$ for any $t, t_1, \ldots t_k \in \mathcal{TH}(A)$ and $n_1, \ldots, n_k \in \mathbb{N}$ satisfying the conditions in Definition 4.11(iii) □

Note that any multiset ingredient of a tree-like multiset is either a tree-like multiset or a group of tree-like multisets. As an example, the multiset $t = \lceil a, \lceil b\rceil^5, \lceil c, \lceil d\rceil^3, \lceil\lceil e\rceil, \lceil f\rceil\rceil\rceil^2\rceil$ is a tree-like multiset over the set $\{a, b, c, d, e, f\}$: $root(t) = a$; the elements $t_1 = \lceil b\rceil, t_2 = \lceil c, \lceil d\rceil^3, \lceil\lceil e\rceil, \lceil f\rceil\rceil\rceil \in dom(t)$ are both tree-like multisets with $root(t_1) = b$ and $root(t_2) = c$, respectively; the element $\lceil\lceil e\rceil, \lceil f\rceil\rceil \in dom(t_2)$ is a group of tree-like multisets.

Restriction of $\mathcal{H}(A)$ to tree-like multisets results in a hierarchy of tree-like multisets. Let us denote this hierarchy and its classes by $\mathcal{TH}(A)$ and $\mathcal{TH}_i(A)$ ($i \geq 1$), respectively. According to Definition 4.11, $\mathcal{TH}_1(A) = \{\lceil a\rceil : a \in A\}$ and $\mathcal{TH}(A) = \bigcup_i \mathcal{TH}_i(A)$. Note that $\mathcal{TH}(A)$ is **not** closed under additive union and multiset minus.

The following theorem shows that a hierarchical product of a CFD is always a tree-like multiset.

Theorem 4.3. Any hierarchical product of a given CFD over a set of features F is a tree-like multiset over F. □

For example, consider again the three hierarchical products of our running example in Figure 4.1: $h_1 = \lceil \mathsf{f}, \lceil \mathsf{f}_1\rceil^5\rceil$, $h_2 = \lceil \mathsf{f}, \lceil \mathsf{f}_1\rceil^5, \lceil \mathsf{f}_2, \lceil \mathsf{f}_3\rceil^3, \lceil\lceil \mathsf{f}_4\rceil, \lceil \mathsf{f}_5\rceil\rceil\rceil\rceil$, and $h_3 = \lceil \mathsf{f}, \lceil \mathsf{f}_1\rceil^5, \lceil \mathsf{f}_2, \lceil \mathsf{f}_3\rceil^3, \lceil\lceil \mathsf{f}_4\rceil, \lceil \mathsf{f}_5\rceil\rceil\rceil^2\rceil$. It is easy to see that h_{1-3} are all tree-like multisets.

The rest of the section is devoted to showing that any tree-like multiset is a hierarchical product of some CFDs. We show how to extract a CFD from a given tree-like multiset. This is done step by step through the following definitions. Definition 4.16, Definition 4.17, and Definition 4.18 show, respectively, how to extract the tree, groups, and multiplcities from a given tree-like multiset.

We first define the notion of a *tree-like multisest induced* by an element:

Definition 4.14 (**Tree-like Multiset Induced by Elements**). For a given tree-like multiset t over a set A, the *tree-like multiset induced* by a, denoted by t^a, is the multiset ingredient of t whose root is a. $\square$

Remark 4.2. According to Definition 4.11, the following statement follows: Let $t \in \mathcal{TH}(A)$ for a set A of urelements. For any $a \in dom(\mathsf{flat}_A(t))$, there is a unique multiset ingredient of t whose root is a. This uniqueness makes Definition 4.14 well-formed.

For an example, consider the tree-like multiset $t = \lceil a, \lceil b \rceil^5, \lceil c, \lceil d \rceil^3, \lceil \lceil e \rceil, \lceil f \rceil \rceil \rceil^2 \rceil$ over the set $\{a, b, c, d, e, f\}$. Then, we would have: $t^a = t$, $t^b = \lceil b \rceil$, $t^c = \lceil c, \lceil d \rceil^3, \lceil \lceil e \rceil, \lceil f \rceil \rceil \rceil$, $t^d = \lceil d \rceil$, $t^e = \lceil e \rceil$, and $t^f = \lceil f \rceil$.

The following definition introduces the notion of *parents* in tree-like multisets.

Definition 4.15 (**Parents of Elements in Tree-like Multisets**). For a given tree-like multiset t over a set A and $a \in dom(\mathsf{flat}_A(t)) \setminus \{root(t)\}$, the *parent of* a, denoted by $a^{\uparrow_t}$, is an element in $dom(\mathsf{flat}_A(t))$ such that

(i) if t^a is a grouped tree-like multiset under a group multiset g, then g is in the domain of the tree-like multiset induced by $a^{\uparrow_t}$, i.e., $g \in dom(t^{a^{\uparrow_t}})$.

(ii) if t^a is a tree-like multiset, then it is in the domain of the tree-like multiset induced by $a^{\uparrow_t}$, i.e., $t^a \in dom(t^{a^{\uparrow_t}})$. $\square$

Remark 4.3. According to Definition 4.11, the following statement follows obviously: Consider a tree-like multiset $t \in \mathcal{TH}(A)$ for a set A. For any $a \in dom(\mathsf{flat}_A(t)) \setminus \{root(t)\}$, there exists a unique element in $dom(\mathsf{flat}_A(t))$ satisfying (i) and (ii) in Definition 4.15. This makes Definition 4.15 well-formed. Therefore, $\_^{\uparrow_t}$ is indeed a function from $dom(\mathsf{flat}_A(t)) \setminus \{root(t)\}$ to $dom(\mathsf{flat}_A(t))$. $\square$

For an example, consider again the tree-like multiset $t = \lceil a, \lceil b \rceil^5, \lceil c, \lceil d \rceil^3, \lceil \lceil e \rceil, \lceil f \rceil \rceil \rceil^2 \rceil$ over the set $\{a, b, c, d, e, f\}$. We would have: $b^{\uparrow_t} = c^{\uparrow_t} = a$ and $d^{\uparrow_t} = e^{\uparrow_t} = f^{\uparrow_t} = c$.

Now we can see that any tree-like multiset represents a unique tree of the elements of its corresponding flat multiset. This tree is extracted using the parents of elements. The following definition shows how to do so.

Definition 4.16 (Trees Associated with Tree-like Multisets). Let t be a tree-like multiset over a set A; the *tree associated* with t, denoted by T_t, is defined as follows: $T_t = (N_t, r_t, \_^{\uparrow_t})$, where $N_t = dom(\mathsf{flat}_A(t))$, $r_t = root(t)$, and $\_^{\uparrow_t} : N_t \setminus \{r_t\} \to N_t$ is a function defined in Definition 4.15. $\square$

For an example, consider the tree-like multiset $t = \lceil a, \lceil b \rceil^5, \lceil c, \lceil d \rceil^3, \lceil \lceil e \rceil, \lceil f \rceil \rceil \rceil^2 \rceil$ $\in \mathcal{TH}(\{a, b, c, d, e, f\})$. Its tree is represented in Figure 4.5: The root of t, i.e., a, is the root of the tree. There are two elements, b and c, whose parents are a. b is a leaf in the tree, as there is no element whose parent is b. There are three elements d, e, and f whose parents are c (e and f are grouped tree-like multisets and their corresponding group $\lceil \lceil e \rceil, \lceil f \rceil \rceil$ is an element in the domain of $t^c = \lceil c, \lceil d \rceil^3, \lceil \lceil e \rceil, \lceil f \rceil \rceil \rceil$.). All the elements d, e, and f are leaves, as there is no element whose parent is either d, e, or f.

The following definition shows how to extract groups from tree-like multisets. Groups are extracted via group multiset ingredients.

Definition 4.17 (Groups Associated with Tree-like Multisets). Let t be a tree-like multiset over a set A. A set $G \subset dom(\mathsf{flat}_A(t))$ is called a *group* if there exists a group tree-like multiset $g \in \mathsf{MultIng}(t)$ such that $G = \{root(x) : x \in dom(g)\}$. We

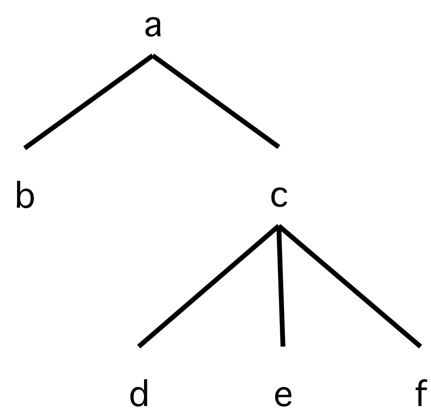


Figure 4.5: Trees associated with tree-like multisets: example

define $G^{\uparrow_t} \stackrel{\text{def}}{=} e^{\uparrow_t}$ for an element $e \in G$ and call it the parent of G.[4]

The set of all groups of t is denoted by $\mathcal{G}_t$. Let $\mathcal{G}(a)$ denote the set of all groups G whose parent is a, i.e., $\mathcal{G}(a) = \{G \in \mathcal{G}_t : G^{\uparrow_t} = a\}$. □

For an example, consider the multiset $t = \lceil a, \lceil [b\rceil, \lceil c, \lceil d\rceil^4\rceil\rceil^5, \lceil e, \lceil f\rceil^3, \lceil\lceil g\rceil, \lceil h\rceil\rceil\rceil^2\rceil$. There are two group tree-like multisets $g_1 = \lceil\lceil b\rceil, \lceil c, \lceil d\rceil^4\rceil\rceil$, $g_2 = \lceil\lceil g\rceil, \lceil h\rceil\rceil$. According to Definition 4.17, the groups corresponding to g_1 and g_2 would be, respectively, equal to the sets $G_1 = \{\ root(\lceil b\rceil),\ root(\lceil c, \lceil d\rceil^4\rceil)\ \} = \{b, c\}$ and $G_2 = \{\ root(\lceil g\rceil),\ root(\lceil h\rceil)\ \} = \{g, h\}$.

We have already shown how to extract the corresponding tree and groups from a given tree-like multiset. All we need to do now is to know how to extract multiplicities from tree-like multisets. The following definition shows how to do so.

Definition 4.18 (Multiplicities Associated with Tree-like Multisets). For a given tree-like multiset $t \in \mathcal{TH}(A)$ over a set A, we define a function $\mathcal{C}_t : \big(dom(\mathsf{flat}_A(t))\setminus$

[4]Note that $\forall e, e' \in G : e^{\uparrow_t} = e'^{\uparrow_t}$.

$\{root(t)\}\big) \cup \mathcal{G}_t \to \mathbb{N}$ as follows:

$$\mathcal{C}_t(e) = \begin{cases} |e| & \text{if } e \in \mathcal{G}_t \\ \#_t(t^e) & \text{otherwise} \end{cases}$$

Recall that t^e and $\#_t(t^e)$ denote the tree-like multiset induced by e and the multiplicity of t^e (see Definition 4.9), respectively. □

As an example, consider again the tree-like multiset $t = \lceil a, \lceil b\rceil^5, \lceil c, \lceil d\rceil^3, \lceil\lceil e\rceil, \lceil f\rceil\rceil\rceil^2\rceil$. It has only one associated group $G = \{e, f\}$. According to Definition 4.18, $\mathcal{C}_t$ is defined on $\{a, b, c, d, e, f, G\}$ as follows:

$\mathcal{C}_t(b) = \#_t(t^b) = \#_t(\lceil b\rceil) = 5.$

$\mathcal{C}_t(c) = \#_t(t^c) = \#_t(\lceil c, \lceil d\rceil^3, \lceil\lceil e\rceil, \lceil f\rceil\rceil\rceil) = 2.$

$\mathcal{C}_t(d) = \#_t(t^d)\} = \#_t(\lceil d\rceil) = 3.$

$\mathcal{C}_t(e) = \#_t(t^e)\} = \#_t(\lceil e\rceil) = 1.$

$\mathcal{C}_t(f) = \#_t(t^f)\} = \#_t(\lceil f\rceil) = 1.$

$\mathcal{C}_t(G) = \#_t(t^G) = |G| = 2.$

Now we are at the point where we can prove that any tree-like multiset is a hierarchical product of some CFD.

Theorem 4.4. For any tree-like multiset t, there is a CFD $\mathbf{D}$ such that $t \in \mathcal{P}(\mathbf{D})$ □

For an example, consider the tree-like multisets $t = \lceil a, \lceil b\rceil^5, \lceil c, \lceil d\rceil^3, \lceil\lceil e\rceil, \lceil f\rceil\rceil\rceil^2\rceil$ and $t' = \lceil a, \lceil c, \lceil\lceil e\rceil\rceil\rceil, \lceil g\rceil^3\rceil$. The CFDs $\mathbf{D}_t$ and $\mathbf{D}_{t'}$ in Figure 4.6 represents two CFDs whose hierarchical semantics include t and t', respectively.

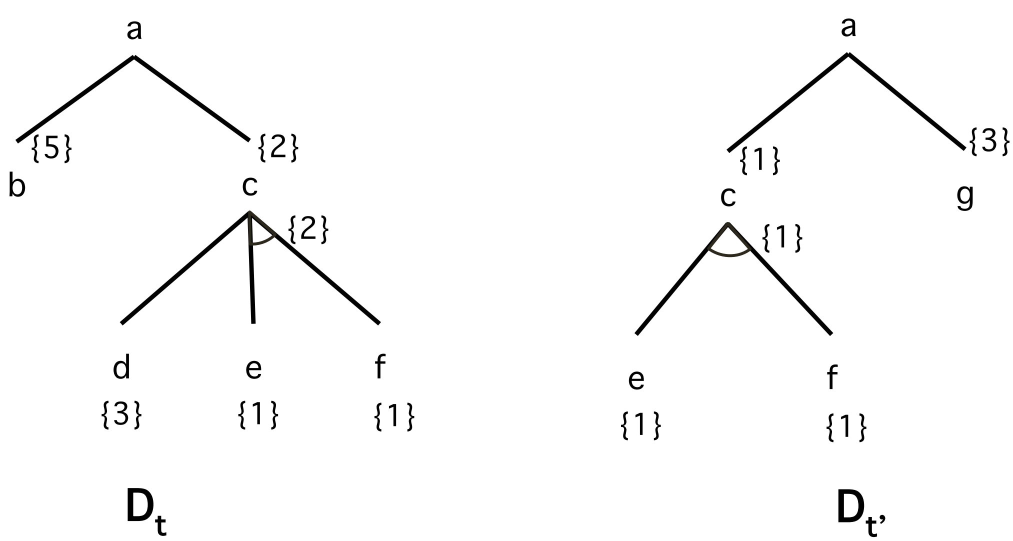


Figure 4.6: Representative CFDs of single tree-like multisets: example

4.4 Characterization of Hierarchical Semantics

In the previous section, we showed that a multiset is a hierarchical product of some CFDs if and only if it is a tree-like multiset. In this section, we are going to characterize hierarchical semantics of CFDs. That is, we want to see what sets of tree-like multisets can be the hierarchical semantics of a CFD. We first define the notions *mergeable tree-like* and *completely mergeable tree-like multisets.* A set of tree-like multisets is mergeable if it represents a subset of the hierarchical semantics of some CFDs. It is called completely mergeable if it is equal to the hierarchical semantics of a CFD.

Definition 4.19 (**Mergeable Tree-like Multisets).** We say that the elements of a (possibly infinite) set of tree-like multisets U are

(i) *mergeable* if there exists a CFD $\mathbf{D}$ such that $U \subseteq \mathcal{P}(\mathbf{D})$. We then call $\mathbf{D}$ a *representative CFD* of U.

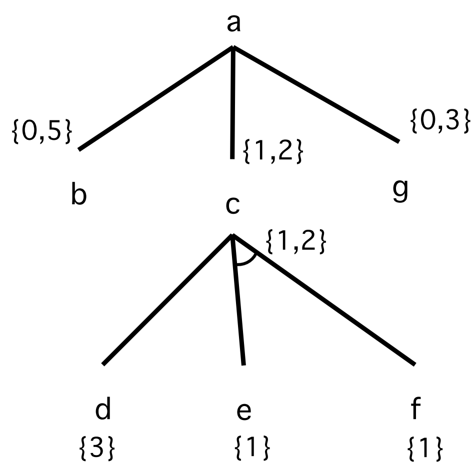


Figure 4.7: Representative CFDs of mergeable tree-like multisets: example

(ii) *completely mergeable* if there is a CFD $\mathbf{D}$ such that $U = \mathcal{P}(\mathbf{D})$. $\square$

According to Theorem 4.4, any singleton set of tree-like multisets is mergeable. Consider the tree-like multisets $t = \lceil a, \lceil b \rceil^5, \lceil c, \lceil d \rceil^3, \lceil \lceil e \rceil, \lceil f \rceil \rceil \rceil^2 \rceil$ and $t' = \lceil a, \lceil c, \lceil \lceil e \rceil \rceil \rceil$ $, \lceil g \rceil^3 \rceil$. Figure 4.7 represents a CFD whose hierarchical semantics includes t and t'. Therefore, they are mergeable. However, t and t' are not completely mergeable.

As a simple example of non-mergeable tree-like multisets, consider $n = \lceil a, \lceil b \rceil^3 \rceil$ and $n' = \lceil b, \lceil a \rceil^2 \rceil$. They are not mergeable, as their roots are different.

There is no unique CFD representing a given set of tree-like multisets. For example, replacing the multiplicity domain of node b in $\mathbf{D}$ (Figure 4.7) by any other multiplicity domains including 0 and 5 (e.g., $\mathbb{N}$), the CFD would still represent t and t'. Another example: adding an optional subfeature[5] to the node b, the CFD is still a representative of t and t'. Indeed, for a given set of mergeable tree-like multisets, there is an infinite number of representative CFDs. Therefore, a notion of minimality for representative CFDs can be useful.

[5]multiplicity domain with lower bound 0

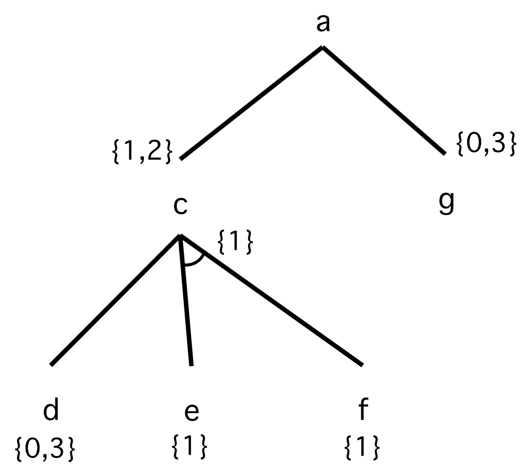


Figure 4.8: Representative CFDs of mergeable tree-like multisets: example

Definition 4.20 (Minimal Representative CFDs). A CFD $\mathbf{D}$ is called a *minimal representative* CFD of a given set of mergeable tree-like multisets U if

(i) it is a representative CFD of U, and

(ii) for any other representative CFD $\mathbf{D}'$ of U, $|\mathcal{P}(\mathbf{D})| \leq |\mathcal{P}(\mathbf{D}')|$.

Let $\mathcal{D}_{U^{\text{merge}}}$ denote the family of minimal representative CFDs of U. □

The CFD $\mathbf{D}$ in Figure 4.7 represents a minimal representative CFD of the tree-like multisets $t = \lceil a, \lceil b \rceil^5, \lceil c, \lceil d \rceil^3, \lceil \lceil e \rceil, \lceil f \rceil \rceil \rceil^2 \rceil$ and $t' = \lceil a, \lceil c, \lceil \lceil e \rceil \rceil \rceil, \lceil g \rceil^3 \rceil$. For these two tree-like multisets, there is, indeed, only one minimal representative CFD. Now, consider another tree-like multiset $t'' = \lceil a, \lceil c, \lceil d \rceil^3, \lceil \lceil e \rceil \rceil \rceil^2 \rceil$. A minimal representative CFD of t'' and t' is represented in Figure 4.8. However, this is not the only minimal CFD representing these two tree-like multisets: replacing f by another feature, say x, we obtain another minimal representative CFD of t'' and t'.

Remark 4.4. Note that, according to Theorem 4.1, there is a single minimal representative CFD of a given set of completely mergeable tree-like multisets. □

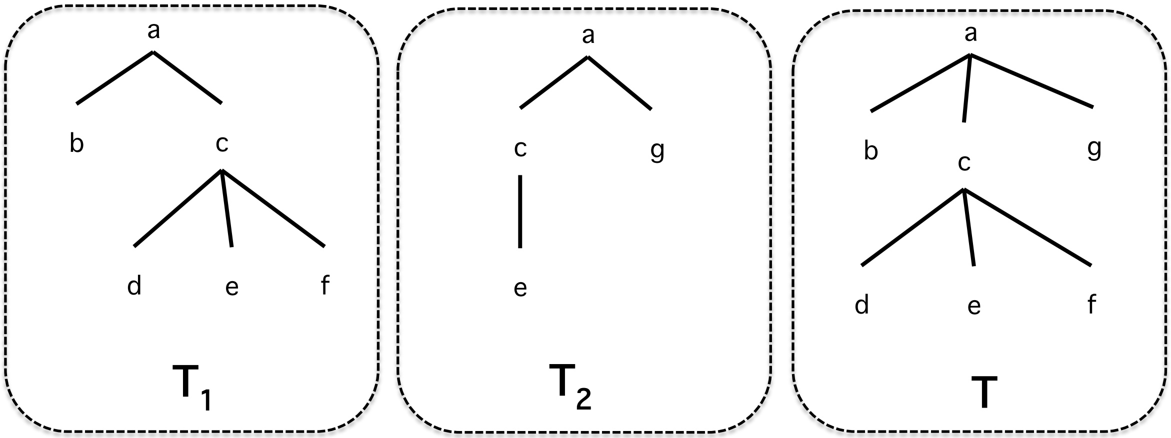


Figure 4.9: Megeable trees and their representative trees: an example

In the rest of this section, we are going to characterize mergeable tree-like multisets. To this end, we first introduce the notion of mergeable trees.

Remark 4.5. Note that a mergeable set of tree-like multisets may be infinite. However, it is always enumerable, as the hierarchical semantics of a CFD is always enumerable. This simple fact is used in the following definitions and theorems. $\square$

Definition 4.21 (Mergeable Trees). Consider an enumerable set of trees $\mathcal{T} = \{T_i : i \in I\}$, where I enumerates its elements. Let $T_i = (N_i, r_i, \_^{\uparrow_i})$, $\forall i \in I$. We say that the trees in $\mathcal{T}$ are mergeable if

(i) $\forall i, j \in I : r_i = r_j$.

(ii) $\forall i, j \in I, \forall n \in (N_i \cap N_j) \setminus \{r_1\} : n^{\uparrow_i} = n^{\uparrow_j}$.

Then the tuple $(N, r, \_^{\uparrow})$, where $N = \bigcup_{i \in I} N_i, r = r_1$, and $\_^{\uparrow} = \bigcup_{i \in I} \_^{\uparrow_i}$ is a tree. We use the notation $\mathcal{T}^{\text{merge}}$ to denote this tree and call it the *representative tree* of $\mathcal{T}$. $\square$

As an example, consider the megeable trees T_1, T_2 and their representative tree T in Figure 4.9. Taking advantage of this notion, we characterize mergeable tree-like multisets in the following theorem.

Theorem 4.5. Consider an enumerable set of tree-like multisets $U = \{t_i : i \in I\} \subset \mathcal{TH}(A)$ over a set A, where I enumerates its elements. Let $T_i = (N_i, r_i, \_^{\uparrow_i})$ and $\mathcal{G}_i$ ($\forall i \in I$) denote the t_i's associated tree and groups, respectively (see Definitions 4.16 and 4.17, respectively). The tree-like multisets in U are mergeable iff:

(i) $\forall i, j \in I : T_i, T_j$ are mergeable.

(ii) $\forall i, j \in I, \forall n \in N_i \cap N_j : (\exists G \in \mathcal{G}_i : n \in G) \implies (\exists G \in \mathcal{G}_j : n \in G)$. $\square$

The above theorem characterized mergeable tree-like multisets. However, it does not lead us to a pragmatic approach when a given set of tree-like multisets is infinite. We need to address this problem. Note that what makes the hierarchical semantics of a CFD infinite is due to some infinite multiplicity domains of some nodes, e.g., the multiplicity domain $\mathbb{N} \setminus \{3\}$ on $\mathsf{f_1}$ in the CFD in Figure 4.1. However, as we saw in Theorem 4.5, multiplicities on elements in tree-like multisets have no influence in making them mergeable or not. We will use this clue to address the problem. We first introduce the notion of *relaxed multisets*. A relaxed version of a given multiset is obtained by changing all multiplicities of its ingredients to 1. For an example, the relaxed multiset of $\lceil a, \lceil b \rceil^5, \lceil c, \lceil d \rceil^3, \lceil \lceil e \rceil, \lceil f \rceil \rceil \rceil^2 \rceil$ would be $\lceil a, \lceil b \rceil, \lceil c, \lceil d \rceil, \lceil \lceil e \rceil, \lceil f \rceil \rceil \rceil \rceil$.

Definition 4.22 (Relaxed Multisets). Given a multiset $m \in \mathcal{H}(A)$ over a set A, its *relaxed multiset*, denoted by m°, is defined as follows:

$$dom(\mathsf{flat}_A(m^\circ)) = dom(\mathsf{flat}_A(m)),$$

$$\mathsf{MultIng}(m^\circ) = \mathsf{MultIng}(m),$$

$$\forall e \in dom(m^\circ) : m^\circ(e) = 1,$$

$$\forall n \in \mathsf{MultIng}(m^\circ), \forall e \in dom(n) : n(e) = 1.$$

For a given set of multisets U, let U° denote the set $\{m^\circ : m \in U\}$. $\square$

The following proposition follows easily.

Proposition 4.1. Let $T_t = (N_t, r_t, \_^{\uparrow_t}), \mathcal{G}_t, \mathcal{C}_t$ denote tree, groups, and multiplicities associated with a given tree-like multiset $t \in \mathcal{TH}(A)$ (see Definitions 4.16, 4.17, and 4.18.). The tree and groups associated with t° are equal to $T_{t^\circ} = T_t$ and $\mathcal{G}_{t^\circ} = \mathcal{G}_t$, respectively. The multiplicities associated with t°, i.e., $\mathcal{C}_{t^\circ}$, is defined as follows:

$$\mathcal{C}_{t^\circ}(e) = \begin{cases} \{1\} & \text{if } e \in (N_t \setminus \{r_t\}) \\ \mathcal{C}_t(e) & \text{if } G \in \mathcal{G} \end{cases}$$

□

To specify whether a given set of tree-like multisets is mergeable or not, we just need to deal with its relaxed version (see Theorem 4.6(i)). More interestingly (and practically useful), the relaxed version of a set of mergeable tree-like multisets is finite (see Theorem 4.6(ii)).

Theorem 4.6. Consider an enumerable set of tree-like multisets $U \subset \mathcal{TH}(A)$ over a set A.

(i) U is mergeable iff U° is.

(ii) U is mergeable implies that U° is finite. □

Now, we want to characterize completely mergeable tree-like multisets. This is done in Theorem 4.7. Before getting to the theorem, let us see some examples. Consider the set $U = \{t_1, t_2, t_3, t_4\}$ of tree-like multisets, where t_{1-4} are as follows:

$t_1 = \lceil a, \lceil b \rceil^5, \lceil \lceil c \rceil \rceil \rceil$,

$t_2 = \lceil a, \lceil b \rceil^5, \lceil \lceil d \rceil^3 \rceil \rceil$,

$t_3 = \lceil a, \lceil b \rceil^2, \lceil \lceil c \rceil \rceil \rceil$,

$t_4 = \lceil a, \lceil b \rceil^2, \lceil \lceil d \rceil^3 \rceil \rceil$.

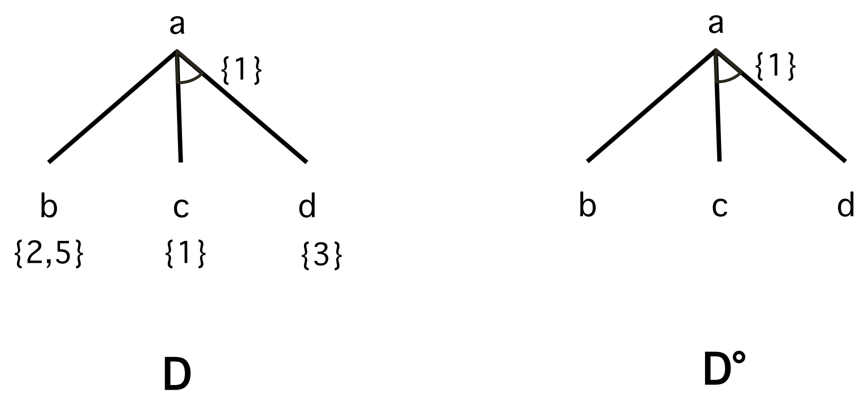


Figure 4.10: Minimal representative CFDs of U and U°

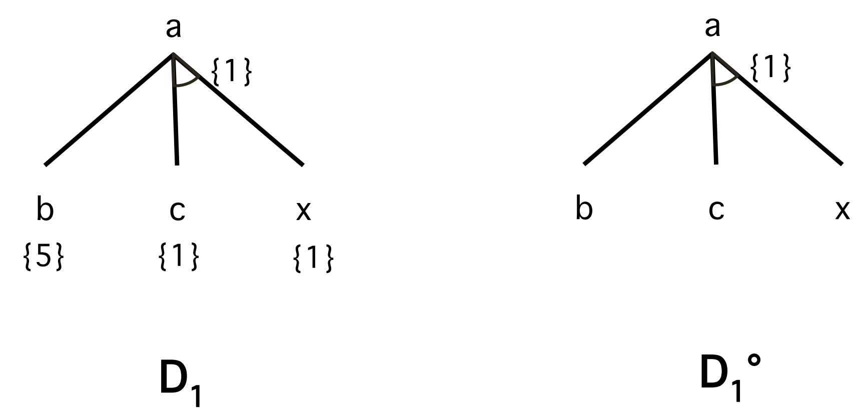


Figure 4.11: Minimal representative CFDs of U_1 and ${U_1}^\circ$

Their relaxed multisets are represented in the following (as usual, we denote the set of the following tree-like multisets by U°):

$${t_1}^\circ = {t_3}^\circ = \lceil a, \lceil b \rceil, \lceil \lceil c \rceil \rceil \rceil,$$

$${t_2}^\circ = {t_4}^\circ = \lceil a, \lceil b \rceil, \lceil \lceil d \rceil \rceil \rceil.$$

Two minimal representative CFDs of U and U° are represented in Figure 4.10 as $\mathbf{D}$ and $\mathbf{D}^\circ$, respectively.[6] Since $\mathcal{P}(\mathbf{D}) = U$ and $\mathcal{P}(\mathbf{D}^\circ) = U^\circ$, both U and U° are completely mergeable.

[6]Since U is completely mergeable, there is only one minimal representative CFD of U.

Now, consider $U_1 = U \setminus \{t_2, t_4\}$. Clearly, U_1 is not completely mergeable. A minimal representative CFD $\mathbf{D}_1$ of U_1 is represented in Figure 4.11, where x can be any feature not equal to a, b, or c. A representative CFD $\mathbf{D}_1{}^\circ$ of $U_1{}^\circ$ is also represented in Figure 4.11. Note that $U_1{}^\circ$ is not completely mergeable either.

What we saw in the above examples is indeed a general rule: For any completely mergeable tree-like multisets U, U° would be completely mergeable too. However, we cannot characterize completely mergeable multisets relying on just their relaxed multisets. Indeed, there are sets of tree-like multisets which are not completely mergeable, but their relaxed multisets are. As an example, consider the set of multisets $U_2 = U \setminus \{t_3\}$. Clearly, U_2 is not completely mergeable. The CFD $\mathbf{D}$ in Figure 4.10 is a minimal representative CFD of U_2 (recall that it is also a representative CFD of U). Since $U_2{}^\circ = U^\circ$, $U_2{}^\circ$ is completely mergeable (as U° is).

The above discussion shows that characterization of completely mergeable tree-like multisets goes via their relaxed tree-like multisets and multiplicities. We will need the following notion.

Definition 4.23 (Overall Multiplicities). Given a set of tree-like multisets $U \subset \mathcal{TH}(A)$, we define a function $\mathcal{C}_U : A \to 2^{\mathbb{N}}$ as follows: $\mathcal{C}_U(a) = \bigcup_{t \in U} \{\#_t(t^a)\}$.[7] □

Theorem 4.7. Consider an enumerable set of tree-like multisets $U \subset \mathcal{TH}(A)$ over a set A. U is completely mergeable iff

(i) U° is completely mergeable, and

(ii) $\forall t \in U^\circ, \forall a \in dom(\mathsf{flat}_A(t)), \forall c \in \mathcal{C}_U(a), \exists t' \in U : (t'^\circ = t) \wedge (\#_{t'}(t'^a) = c)$.

In Theorem 4.1, we showed that two CFDs are equal iff their hierarchical semantics

[7] Recall that t^a and $\#_t(t^a)$ denote the multiset induced by a and the multiplicity of t^a in t, resp.

are equal. This implies that there is a unique minimal representative CFD of given completely mergeable tree-like multisets. Thus, we get to the following statement, which is a corollary of Theorems 4.7 and 4.1.

Corollary 4.2. There is a bijection between the domains of CFDs and completely mergeable tree-like multisets. □

4.5 Other Applications

As mentioned in Section 4.1, the flat semantics is commonly considered as the semantics of CFDs in the literature. The most well-known formulation of flat semantics is given via context-free grammars, that is, a given CFD is transformed to a context-free grammar and then the Parikh image of its language is considered as the set of flat products of the CFD [CHE05a]. However, as far as we know, flat semantics never achieved a direct definition. This is an important issue, as verification of a proposed formulation of products without having a formal definition of them is impossible. We provided a direct definition of flat products in Definition 4.1. In Chapter 5, where we propose transformation of CFDs to regular expressions, we will take advantage of this definition to verify the proposed methods. Deciding whether a given multiset is a valid flat product for a given CFD or not is algorithmic. This is easy to see via Lemma 4.1, where a recursive terminating definition of flat products is provided. The flat semantics of a given CFD can address some analysis questions about the CFD, including “deciding whether a given multiset is a valid product of a given CFD or not”, “deciding whether a given integer is a valid multiplicity of a given feature or not”, etc. Therefore, this semantics provides a useful abstract view of CFDs. However, the flat semantics of a given CFD does not capture all useful information about the CFD. To overcome this problem, we have proposed another multisets-based semantics called the hierarchical semantics.

The hierarchical semantics of a given CFD captures all information about the CFD (see Theorem 4.1). This means that one can address any question about the CFD based on its hierarchical semantics. It is easy to see that deciding whether a given multiset is a hierarchical product of a given CFD or not is algorithmic (see the recursive definition of hierarchical products in Definition 4.7). One of the three transformations of CFDs to regular expressions discussed in Chapter 5 is based on the hierarchical semantics, which ensures that the transformation is a faithful one (see Section 5.4 and Section 5.5).

The hierarchical semantics could also be used in the reverse engineering of CFDs (an important problem in feature modeling), as the hierarchical semantics of a given CFD captures all information about the CFD. In Section 4.3 and Section 4.4, we characterized the hierarchical products and semantics of a given CFD, respectively. The proofs given for corresponding theorems are all constructive: Theorem 4.4 constructively shows that there is a CFD representing a given tree-like multiset; Theorem 4.6, whose proof is constructive, characterizes mergeable tree-like multisets; Theorem 4.7, whose proof constructively shows how to retrieve the CFD from its hierarchical semantics, characterizes completely mergeable multisets.

Another important application of the hierarchical semantics of CFDs regards *feature model management*, which is an active area in feature modeling. By feature model management, we mean feature model composition via some operations like merging, intersection, and union, etc [SBRCT08, ACLF10a, ACC+13]. Characterization of the hierarchical semantics come in handy here. As an example, suppose that we want to obtain the merge of two CFDs $\mathbf{D}_1$ and $\mathbf{D}_2$. We need to address the two following questions: Are $\mathbf{D}_1$ and $\mathbf{D}_2$ mergeable? What would be the result of their merge, if they are mergeable? To address these questions, we first obtain their hierarchical semantics $\mathcal{P}(\mathbf{D}_1)$ and $\mathcal{P}(\mathbf{D}_2)$, respectively. We then decide whether their union is mergeable or not. To this end, we take advantage of Theorem 4.6. This would address the first question. If they are mergeable, then we obtain a representative CFD of $\mathcal{P}(\mathbf{D}_1) \cup \mathcal{P}(\mathbf{D}_2)$. The proof of Theorem 4.6 constructively shows how to obtain a representative CFD of a set of mergeable tree-like multisets. As for the intersection (union, respectively) of $\mathbf{D}_1$ and $\mathbf{D}_2$, we first obtain the intersection (union, respectively) of their hierarchical semantics and then decide whether the obtained set of tree-like multisets is completely mergeable or not. To this end, we would apply Theorem 4.7.

Chapter 5

The Semantics of Cardinality-Based Feature Models via Formal Languages

In this chapter, we build semantics for CFMs via formal languages. We propose three different transformations going from CFDs to regular expressions. This provides a semantics for CFDs by using regular languages as the semantic domain. Regular languages have some nice computational properties. These properties, such as the decidability of the emptiness, inclusion, and equality problems, help us to propose algorithmic solutions for analysis operations over CFDs. In addition, the complexity class of all regular languages is SPACE(O(1)), i.e., the decision problems can be solved in constant space. Therefore, we can claim that regular expressions provide a good computational framework for reasoning about CFDs. Also, there are several off-the-shelf tools dealing with the class of regular languages. This enables us to address automated analysis over CFDs, which is a challenging issue in cardinality-based feature modeling.

The first transformation is discussed in Section 5.2. We first define a generalization of CFDs called *Cardinality-based Regular-expression Diagrams* (CRDs) in which labelling of nodes can be any regular expression built over an alphabet. Subsequently, we give a procedure to translate a given CRD to a regular expression, called *CRDs to Regular Expressions* (CRE). We also prove that the CRE regular expression generated for a given CFD captures both the flat semantics and the hierarchy of the CFD; and hence it provides a faithful semantics for the CFD.

The second transformation is discussed in Section 5.3. We first define *Ordered siblings CFDs* (osCFDs) and *Ordered siblings CRDs* (osCRDs) which are CFDs and CRDs, respectively, enriched with a partial order on nodes, called *sibling ordering*. We then provide a procedure to translate a given osCRD to a regular expression, called *osCRDs to Regular expressions* (ORE). We show that the ORE regular expression generated for a given osCFD captures the flat semantics of its underlying CFD. However, it may not capture the hierarchy of the CFD. Thus, this transformation does not provide a faithful semantics for CFDs. However, it is a cheap transformation and yet useful for many analysis questions about CFDs.

The third transformation, called *Hierarchical semantics to Regular Expressions* (HRE) is discussed in Section 5.4. This transformation is based on the hierarchical semantics discussed in Chapter 4. Thus, it provides a faithful semantics for the underlying CFD of a given osCFD. The HRE regular expression for a given osCFD is built on the set of features plus two extra symbols ($\lceil$ and $\rceil$). This differentiates HRE from ORE and CRE. The transformations will be further discussed in Section 5.5.

As for crosscutting constraints over CFDs, we propose a formal language interpretation of them (see Section 5.6). In this way, we can integrate the formal semantics of CFDs and crosscutting constraints over them. Thus, three different kinds of formal languages are associated with a given CFM, i.e., CRE, ORE, and HRE languages of the CFM. Formal language encoding of CFMs allows us to group CFMs based on their computational properties, say regular, context-free, and context-sensitive CFMs. As a result, we give a computational hierarchy of CFMs, which guides us in how to constructively analyze them. We also investigate the decidability problems of some analysis operations over CFMs. Interestingly, we noticed that not all of the investigated analysis operations are decidable in all classes of CFMs.

5.1 Cardinality-Based Feature Diagram: Syntax

A CFD is a tree of features in which some subsets of non-root nodes are grouped and other nodes are called solitary. In addition, non-root nodes and groups are equipped with some multiplicity constaints. A multiplicity constraint is usually expressed as a sequence of pairs (l, u), where l is a natural number, u is either a number or $*$ (representing an unbounded multiplicity) and $l \leq u$. We call a multiplicity constraint on a node or group a *multiplicity domain*. As an example, consider the CFD in Figure 5.1. It is a CFD over features $\mathsf{f}, \mathsf{f}_{1-6}$. G denotes a group consisting of the features $\mathsf{f}_4, \mathsf{f}_5$, and f_6, and any feature in $F \setminus \mathsf{G}$ is a solitary feature. The multiplicity domains are as follows: $(2, 3)$ on G, $(1, 2)(4, ^*)$ on f_1, $(0, 2)$ on f_2, $(3, 5)$ on f_3, and $(1, 2)$ on f_6. The multiplicity domains on the features $\mathsf{f}_{4,5}$ are both $(1, 1)$. We will use this CFD as an example to illustrate the transformation procedures in the chapter.

In Chapter 4, where two multiset theories were proposed for CFDs, we formalized

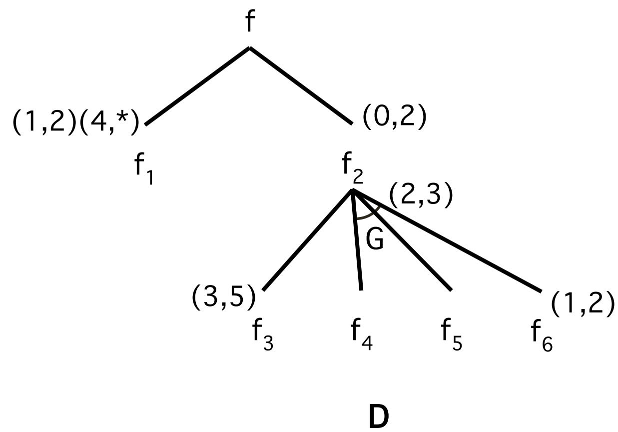


Figure 5.1: A CFD: running example for transformations

a multiplicity domain as a subset of natural numbers to make the formalizations in the chapter easier to read.[1] In this chapter, we get back to the common practical multiplicity constraints, as not all subsets of natural numbers are computable. To this end, we first need to formalize the notion of multiplicities over nodes and groups.

Definition 5.1. We define a structure $(\mathbb{N}^*, \leq_*)$ as $\mathbb{N}^* = \mathbb{N} \cup \{*\}$ the universe, and $\leq_* \subset \mathbb{N}^* \times \mathbb{N}^*$ a reflexive transitive relation defined by $\forall u, l \in \mathbb{N}^* : (l \leq_* u) \Leftrightarrow (l, u \in \mathbb{N} \wedge l \leq u) \vee (u = *)$.[2] For any $u, l \in \mathbb{N}^*$, we use the notation $l <_* u$ to denote $(l \leq_* u) \wedge (l \neq u)$. □

Definition 5.2 (Multiplicities).

(i) The *multiplicity-set* is the set $\mathfrak{C} = \{(l, u) \in \mathbb{N} \times \mathbb{N}^* : (l \leq_* u) \wedge (u \neq 0)\}$. An element $c = (l, u) \in \mathfrak{C}$ is called a *multiplicity*. We call l and u the *lower-bound*, denoted by $low(c)$, and *upper-bound*, denoted by $up(c)$, of c, respectively.

[1] This definition of constraint domains provides us with a more expressive language of CFDs, since not any subset of natural numbers can be expressed as a finite sequence of intervals of numbers.

[2] Another equivalent definition could be $\forall l, u \in \mathbb{N} : (l \leq_* u \Longleftrightarrow l \leq u) \wedge (l \leq_* *)$

(iii) A subset $C \subseteq \mathfrak{C}$ is called a *multiplicity domain* if there exists a *finite* set $I = \{1, \ldots, n\} \subset \mathbb{N}$ such that $C = \{(l_i, u_i) : i \in I\}$ in which $u_i <_* l_{i+1}$, for all $i, i+1 \in I$. We call l_1 and u_n the lower-bound, denoted by $low(C)$, and upper-bound, denoted by $up(C)$, of C, respectively.

$\square$

Definition 5.3 (Cardinality-based Feature Diagrams). A *cardinality-based feature diagram* (CFD) is a 3-tuple $\mathbf{D} = (T, \mathcal{G}, \mathcal{C})$ consisting of the following components.

(i) $T = (F, r, \_^{\uparrow})$ is a *tree* with set F of nodes (called *features*), $r \in F$ is the root, and function $\_^{\uparrow}$ maps each non-root node $f \in F_{-r} \overset{\text{def}}{=} F \setminus \{r\}$ to its parent $f^{\uparrow}$. The inverse function that assigns to each node f the set of its children is denoted by $f_{\downarrow}$. The set of all descendants of f is denoted by $f_{\downarrow\downarrow}$.

(ii) $\mathcal{G} \subseteq 2^{F_{-r}}$ is a set of *grouped* nodes. For all $G \in \mathcal{G}$, $|G| > 1$, and all nodes in G have the same parent, denoted by $G^{\uparrow}$. All groups in $\mathcal{G}$ are disjoint, i.e., $\forall G, G' \in \mathcal{G}.\ (G \neq G') \Rightarrow (G \cap G' = \varnothing)$. The nodes that are not in a group are called *solitary* nodes. Let $\mathcal{S}$ denote the solitary nodes, i.e., $\mathcal{S} = F_{-r} \setminus \bigcup_{G \in \mathcal{G}} G$.

(iii) $\mathcal{C} \subseteq (F_{-r} \cup \mathcal{G}) \times \mathfrak{C}$ is a left-total relation called the *multiplicity relation*. For any element $e \in F_{-r} \cup \mathcal{G}$, $\mathcal{C}(e)$ represents a multiplicity domain (see Definition 5.2(iii)). In addition, for all $G \in \mathcal{G}$, $up(\mathcal{C}(G)) \leq |G|$.

The class of all CFDs and all CFDs over the same set of features F are denoted by $\mathcal{D}$ and $\mathcal{D}(F)$, respectively. $\square$

We will sometimes write a CFD $\mathbf{D}$ as a 5-tuple $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$. If needed, we will subscript $\mathbf{D}$'s components with index $_{\mathbf{D}}$, e.g., write $\mathcal{G}_{\mathbf{D}}$.

Let $\mathbf{D} = (T, \mathcal{G}, \mathcal{C})$ be a CFD with $T = (F, r, \_^{\uparrow})$ and $f \in F$. We will need the following notations in later sections:

– $depth(\mathbf{D})$ denotes T's depth and $depth(f)$ denotes the f's depth in T. In our example in Figure 5.1, $depth(\mathsf{f}) = 1$, $depth(\mathsf{f}_1) = 3$, and $depth(\mathbf{D}) = 3$.

– $lev(\mathbf{D})$ denotes the set of leaf nodes, i.e., $lev(\mathbf{D}) = \{f \in F : f_\downarrow = \varnothing\}$. In Figure 5.1, $lev(\mathbf{D}) = \{\mathsf{f}_1, \mathsf{f}_3, \mathsf{f}_4, \mathsf{f}_5, \mathsf{f}_6\}$.

– $glev(\mathbf{D})$ denotes the set of grouped leaves, i.e., $glev(\mathbf{D}) = \{G \in \mathcal{G} : \forall n \in G.\ n_\downarrow = \varnothing\}$. In Figure 5.1, $glev(\mathbf{D}) = \{\{\mathsf{f}_4, \mathsf{f}_5, \mathsf{f}_6\}\}$.

5.2 The CRE Transformation

In this section, we discuss the *CRDs to Regular Expressions* (CRE) transformation. We first need to define a notion called *cardinality-based regular-expression diagrams* (CRDs). CRDs are generalization of CFDs in which labelling of nodes can be any regular expression built over an alphabet.

Definition 5.4 (Cardinality-based Regular-expression Diagrams). A *cardinality-based regular-expression diagram* (CRD) over an alphabet Σ is a 3-tuple $\mathbf{RD} = (LT_{re}, \mathcal{G}, \mathcal{C})$ of the following components:

(i) $LT_{re} = (N, r, \_^{\uparrow}, \Sigma, l_{re})$ is a *labeled tree* where N, r, $\_^{\uparrow}$, are as defined in Definition 5.3(i) (see page 97), Σ is a finite set (the alphabet), and $l_{re} : N \to \mathbf{RE}(\Sigma)$ is a function that labels each node with a regular expression built over Σ.

(ii) $\mathcal{G} \subseteq 2^{N_{-r}}$ is a set of *grouped* nodes, as defined in Definition 5.3(ii) (page 97).

(iii) $\mathcal{C} \subseteq (N_{-r} \cup \mathcal{G}) \times \mathfrak{C}$ is called the *multiplicity relation*, as defined in Definition 5.3(iii) (page 97).

The class of all CRDs over the same alphabet Σ will be denoted by $\mathcal{RD}(\Sigma)$.

If needed, we will subscript $\mathbf{RD}$'s components with index $_{\mathbf{RD}}$, e.g., write $\mathcal{G}_{\mathbf{RD}}$. □

Remark 5.1. A CFD would be a CRD, since an atomic feature could be considered as a primitive regular expression built over the set of features (as an alphabet). □

All the operations used on CFDs also work for CRDs. These notations include $depth(\mathbf{RD})$, $lev(\mathbf{RD})$, $glev(\mathbf{RD})$, $cplev(\mathbf{RD})$. Please see page 97 for their meanings. We will also need the following operations.

— $plev(\mathbf{RD})$ denotes the set of non-leaf nodes all of whose children are leaves, i.e., $plev(\mathbf{RD}) = \{n \in N : n_{\downarrow} \subseteq lev(\mathbf{RD})\}$, where N denotes the set of nodes of $\mathbf{RD}$.

– $cplev(\mathbf{RD})$ denotes the leaves all of whose siblings are leaves, i.e., $cplev(\mathbf{RD}) = \{n \in N : (n^{\uparrow})_{\downarrow} \subseteq lev(\mathbf{RD})\}$, where N denotes the set of nodes of $\mathbf{RD}$.

The CRE procedure is a bottom-up procedure and includes a finite number of steps (equal to the depth of the CRD's tree) called CRE-*shrinking* steps. Each CRE-shrinking step takes a CRD and returns another CRD such that the depth of the output's tree is less than that of the input. The output of the last step is a CRD with the singleton tree [3] whose root is labeled with a regular expression. This regular expression is called the CRE *expression* of the CRD.

A shrinking step includes three stages: (1) *eliminating multiplicities from leaves* (CRE-EML), (2) *eliminating grouped leaves* (CRE-EGL), and (3) *depth reduction* (CRE-DR). We will use the CFD in Figure 5.1 as a running example to illustrate the translation procedure.

[3] A tree consisting of a single isolated node.

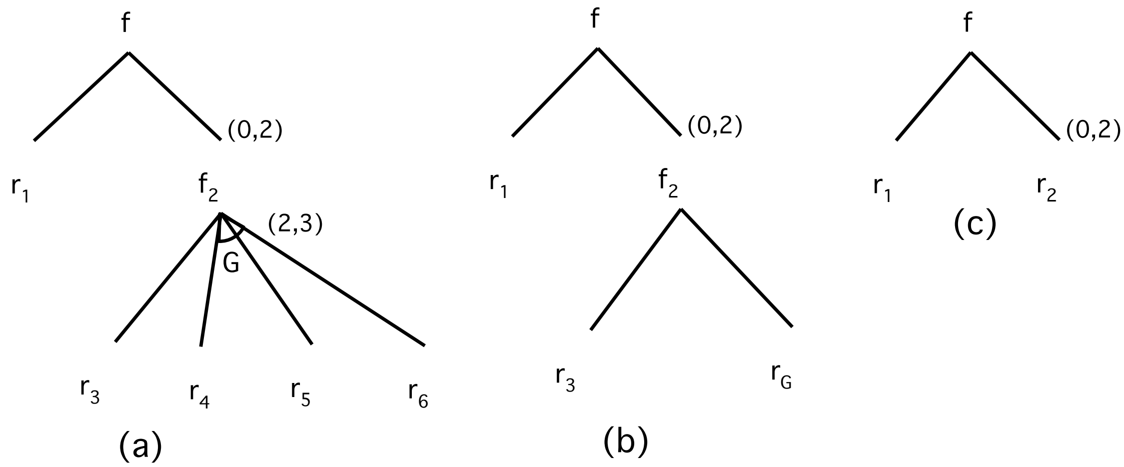


Figure 5.2: CRD to RE: shrinking procedure on Figure 4.1.

5.2.1 CRE-EML

At this stage, regular expressions corresponding to leaf nodes are computed and their multiplicity domains change to the singleton domain $\{(1,1)\}$. For an example, the regular expression corresponding to the node $\mathsf{f_1}$ (Figure 5.1) would be

$$\mathsf{f_1} + \mathsf{f_1}^2 + \mathsf{f_1}^4\mathsf{f_1}^*.$$

This regular expression represents the multiplicity constraint on this node properly, as it says that the number of occurrences of the feature $\mathsf{f_1}$ on this node must be one or two or more than three. Then, the label of the leaves are replaced by their regular expressions, computed in the above way, and their associated multiplicities change to $\{(1,1)\}$. Figure 5.2(a)[4] represents the result of this stage applied to the CFD in Figure 5.1, where

[4]Recall that our convention is to consider a multiplicity domain $(1,1)$ as the default multiplicity of a node and this is why there is no need to show such multiplicities in the figure.

$$r_1 = \mathsf{f_1} + \mathsf{f_1}^2 + \mathsf{f_1}^4\mathsf{f_1}^*,$$

$$r_3 = \mathsf{f_3}^3 + \mathsf{f_3}^4 + \mathsf{f_3}^5,$$

$$r_4 = \mathsf{f_2},$$

$$r_5 = \mathsf{f_4},$$

$$r_6 = \mathsf{f_1} + \mathsf{f_1}^2.$$

The CRE-EML stage (stage 1) is formalized via the following definitions.

Definition 5.5 (**Expressions Associated to Leaves**)**.** Given a CRD $\mathbf{RD} = (LT_{re}, \mathcal{G}, \mathcal{C})$ with $LT_{re} = (N, r, \_^{\uparrow}, \Sigma, l_{re})$, we define a total function $\mathrm{lex}_{\mathbf{RD}} : lev(\mathbf{RD}) \to \mathbf{RE}(\Sigma)$ which maps a leaf node in $\mathbf{RD}$ to a regular expression built over Σ. For a given node $n \in lev(\mathbf{RD})$ with $\mathcal{C}(n) = \{(l_i, u_i)\}_{1 \leq i \leq j}$ (for some $j \in \mathbb{N}$), $\mathrm{lex}_{\mathbf{RD}}(n)$ is defined as folllows:

$$\mathrm{lex}_{\mathbf{RD}}(n) = r_1 + \ldots + r_j, \text{ where}$$

$$r_i = \begin{cases} l_{re}(n)^{l_i} + \ldots + l_{re}(n)^{u_i} & \text{if } u_i \neq * \\ l_{re}(n)^{l_i}\big(l_{re}(n)\big)^* & \text{otherwise} \end{cases}$$

□

Definition 5.6 (CRE-**EML Stage**)**.** [5] We define a function $\mathrm{mel}^{\mathsf{CRE}} : \mathcal{RD}(\Sigma) \to \mathcal{RD}(\Sigma)$, called CRE-*EML* function, as follows. For a given CRD $\mathbf{RD} = (LT_{re}, \mathcal{G}, \mathcal{C})$

[5] We use the superscript CRE for an operation in this section to distinguish it from the operation with similar functionality in the next section. We will use the superscript ORE for the operations in the following sections, e.g., $\mathrm{mel}^{\mathsf{ORE}}$.

with $LT_{re} = (N, r, \_^{\uparrow}, \Sigma, l_{re})$,

$$\text{mel}^{\mathsf{CRE}}(\mathbf{RD}) = (LT'_{re}, \mathcal{G}, \mathcal{C}'), \text{ where}$$

$$LT'_{re} = (N, r, \_^{\uparrow}, \Sigma, l'_{re})$$

$$\mathcal{C}'(e) = \begin{cases} \{(1,1)\} & \text{if } e \in lev(\mathbf{RD}) \\ \mathcal{C}(e) & \text{otherwise} \end{cases}$$

$$l'_{re}(n) = \begin{cases} \text{lex}_{\mathbf{RD}}(n) & \text{if } n \in lev(\mathbf{RD}) \\ l_{re}(n) & \text{otherwise} \end{cases}$$

□

5.2.2 CRE-EGL

At this stage, grouped leaf nodes are replaced by new nodes with proper regular expressions. The input of this stage is the output of the first stage. For an example, consider the grouped leaves $\mathsf{G} = \{\mathsf{f}_4, \mathsf{f}_5, \mathsf{f}_6\}$ in Figure 5.1. The group multiplicity $(2, 3)$ says that at least two and at most three of the nodes involved in the group (i.e., the nodes f_4, f_5, and f_6) must be included in a valid product for each instance of their parent (i.e., the node f_2) in the product. The following regular expressions r_{2-G} and r_{3-G} model the lower and upper bounds of the multiplicity, respectively.

$$r_{2-G} = r_4r_5 + r_5r_4 + r_5r_6 + r_6r_5 + r_4r_6 + r_6r_4$$

$$r_{3-G} = r_4r_5r_6 + r_4r_6r_5 + r_5r_4r_6 + r_5r_6r_4 + r_6r_4r_5 + r_6r_5r_4$$

Thus, the regular expression corresponding to the group would be

$$r_G = r_{2-G} + r_{3-G}.$$

Then, each grouped leaf is replaced by a new node with the default multiplicity domain $\{(1,1)\}$ and is labeled with the computed regular expression. Figure 5.2(b) represents the result of applying this stage on Figure 5.2(a).

To formalize this stage, we first need to introduce the following notation. Let $Per^{(l,u)}(X)$ denote the set of all concatenation permutations S of X (a set of regular expressions) with length between l and u ($l \leq |S| \leq u$). For an example, $Per^{(1,2)}(\{r_1, r_2, r_3\})$ would be the following set of expressions: $\{r_1,\ r_2,\ r_3\} \cup \{r_1 r_2,$ $r_2 r_1,\ r_1 r_3,\ r_2 r_3,\ r_3 r_2\}$. We consider $Per^{(l,u)}(\varnothing) = \epsilon$ for any l and u. By $Per^k(X)$ (for any $k \in \mathbb{N}$), we mean $Per^{(k,k)}(X)$.

We will need the following definitions to formalize the stage CRE-EGL.

Definition 5.7 (**Grouped Leaves CRE-Expressions).** Given a CRD $\mathbf{RD} = (LT_{re}, \mathcal{G}, \mathcal{C})$ with $LT_{re} = (N, r, \_^{\uparrow}, \Sigma, l_{re})$, we define a total function $\mathrm{gex}^{\mathsf{CRE}}_{\mathbf{RD}} : glev(\mathbf{RD}) \rightarrow \mathbf{RE}(\Sigma)$ which maps a grouped set of leaves in $\mathbf{RD}$ to a regular expression built over Σ. For a given group $G \in glev(\mathbf{RD})$ with $\mathcal{C}(G) = \{(l_i, u_i)\}_{1 \leq i \leq j}$ (for some $j \in \mathbb{N}$), $\mathrm{gex}^{\mathsf{CRE}}_{\mathbf{RD}}(G)$ is defined as follows.

$$\mathrm{gex}^{\mathsf{CRE}}_{\mathbf{RD}}(G) = r_1 + \ldots + r_j, \text{ where for all } 1 \leq i \leq j:$$

$$r_i = \bigplus X_i,$$

$$X_i = Per^{(l_i, u_i)}(\{l_{re}(n) : n \in G\}).$$

$\square$

We assign a node identifier to each group all of whose elements are leaves.

Definition 5.8 (Node Identifiers for Leaf Groups). Let $\mathbf{RD} = (LT_{re}, \mathcal{G}, \mathcal{C})$ be a CRD with $LT_{re} = (N, r, \_^{\uparrow}, \Sigma, l_{re})$. For each group $G \in glev(\mathbf{RD})$, a node identifier n_G is assigned. Let N_G denote the set of these node identifiers. In other words, we have a bijection $\mathrm{gid} : N_G \to glev(\mathbf{RD})$ which assigns each grouped node in $glev(\mathbf{RD})$ to a unique node identifier in N_G. $\square$

Definition 5.9 (CRE-EGL Stage). $\mathrm{gle}^{\mathsf{CRE}} : \mathcal{RD}(\Sigma) \to \mathcal{RD}(\Sigma)$ is a total function called CRE-EGL function. For a given CRD $\mathbf{RD} = (LT_{re}, \mathcal{G}, \mathcal{C})$ with $LT_{re} = (N, r, \_^{\uparrow}, \Sigma, l_{re})$, $\mathrm{gle}^{\mathsf{CRE}}(\mathbf{RD})$ is defined as follows:

$$\mathrm{gle}^{\mathsf{CRE}}(\mathbf{RD}) = (LT'_{re}, \mathcal{G}', \mathcal{C}'), \text{ where}$$

$$LT'_{re} = (N', r, \_^{\uparrow'}, \Sigma, l'_{re}),$$

$$N' = (N - glev(\mathbf{RD})) \uplus N_G$$

$$\mathcal{G}' = \mathcal{G} - glev(\mathbf{RD})$$

$$\mathcal{C}'(e) = \begin{cases} \{(1,1)\} & \text{if } e \in N_G \\ \mathcal{C}(e) & \text{otherwise} \end{cases}$$

$$n^{\uparrow'} = \begin{cases} \mathrm{gid}(n)^{\uparrow} & \text{if } n \in N_G \\ n^{\uparrow} & \text{otherwise} \end{cases}$$

$$l'_{re}(n) = \begin{cases} \mathrm{gex}^{\mathsf{CRE}}_{\mathbf{RD}}(\mathrm{gid}(n)), & \text{if } n \in N_G \\ l_{re}(n) & \text{otherwise} \end{cases}$$

□

Remark 5.2. To compute the CRE-regular expression corresponding to a leaf group, we consider all valid permutations of its elements' regular expressions. This is the way how the CRE transformation preserves the given CFD's hierarchy.

5.2.3 CRE-DR

This stage takes the output of the second stage and returns a CRD whose depth is less than that of the input. To this end, the regular expressions corresponding to the nodes all of whose children are leaves are computed. Then, the label of such nodes are replaced by the corresponding computed regular expressions and their child nodes are eliminated from the given CRD. Let us see what the result of this stage applied to the CRD in Figure 5.2(b) would be. There is only one node, labeled by f_2, whose children are all leaves. Figure 5.2(c) shows the result, where

$$r_2 = \mathsf{f}_2(r_3 r_{\mathsf{G}} + r_{\mathsf{G}} r_3).$$

We formalize this stage via the following definitions.

Definition 5.10 (CRE**-Expressions for Parents of Leaves).** Given a CRD $\mathbf{RD} = (LT_{re}, \mathcal{G}, \mathcal{C})$ with $LT_{re} = (N, r, \_^{\uparrow}, \Sigma, l_{re})$, we define a total function $\mathrm{pex}^{\mathsf{CRE}}_{\mathbf{RD}} : plev(\mathbf{RD}) \to \mathbf{RE}(\Sigma)$, which maps a parent all of whose child nodes are leaves to a regular expression. For a given node $n \in plev(\mathbf{RD})$, $\mathrm{pex}^{\mathsf{CRE}}_{\mathbf{RD}}(n)$ is defined as follows:

$$\mathrm{pex}^{\mathsf{CRE}}_{\mathbf{RD}}(n) = l_{re}(n)(+\ Per^{|n_{\downarrow}|}(E)), \text{ where}$$

$$E = \{l_{re}(n') : n' \in n_{\downarrow}\}.$$

□

Remark 5.3. Note that, to compute the CRE-regular expression corresponding to a node $n \in plev(\mathbf{RD})$, we consider all valid permutations of its subfeatures' regular expressions. Indeed, this is the way how the CRE transformation preserves the given CFD's hierarchy.

Definition 5.11 (CRE**-DR Stage).** The function $\mathrm{dre}^{\mathsf{CRE}} : \mathcal{RD}(\Sigma) \to \mathcal{RD}(\Sigma)$ is called CRE-DR function. For a given CRD $\mathbf{RD} = (LT_{re}, \mathcal{G}, \mathcal{C})$ with $LT_{re} = (N, r, \_^{\uparrow}, \Sigma, l_{re})$, $\mathrm{dre}^{\mathsf{CRE}}(\mathbf{RD})$ is defined as follows:

$$\mathrm{dre}^{\mathsf{CRE}}(\mathbf{RD}) = (LT'_{re}, \mathcal{G}, \mathcal{C}')$$

$$LT'_{re} = (N', r, \_^{\uparrow\prime}, \Sigma, l'_{re})$$

$$N' = N - cplev(\mathbf{RD})$$

$$\_^{\uparrow\prime} = \_^{\uparrow}|_{N'}$$

$$\mathcal{C}' = \mathcal{C}|_{N' \cup \mathcal{G}}$$

$$l'_{re}(n) = \begin{cases} \mathrm{pex}^{\mathsf{CRE}}_{\mathbf{RD}}(n) & \text{if } n \in plev(\mathbf{RD}) \\ l_{re}(n) & \text{otherwise} \end{cases}$$

□

5.2.4 CRE-Shrinking Step and CRE

The CRE shrinking step is formalized as the composition of $\text{mel}^{\mathsf{CRE}}$ (CRE-EML stage), $\text{gle}^{\mathsf{CRE}}$ (CRE-EGL stage), and $\text{dre}^{\mathsf{CRE}}$ (CRE-DR stage).

Definition 5.12 (CRE**-Shrinking Step).** The function $\text{shr}^{\mathsf{CRE}} : \mathcal{RD}(\Sigma) \to \mathcal{RD}(\Sigma)$ is called CRE-shrinking function and is defined as $\text{shr}^{\mathsf{CRE}} = \text{dre}^{\mathsf{CRE}} \circ \text{gle}^{\mathsf{CRE}} \circ \text{mel}^{\mathsf{CRE}}$.[6]

□

We keep doing the shrinking steps until we get a singleton CRD. In the running example, we need to do the shrinking step once more. The final result would be the expression

$$r = \mathsf{f}(r_1 r_2' + r_2' r_1), \text{ where}$$

$$r_2' = \varepsilon + r_2 + r_2^2.$$

The notation $\mathcal{R}^{\mathsf{CRE}}(\mathbf{RD})$ is used to denote the CRE regular expression generated for a given CRD **RD**.

Now we are at the point where we can prove that the CRE regular expression interpretation of a given CFD **D** provides a faithful semantics for **D**.

Theorem 5.1. For a given CFD **D**, $\text{Par}(\mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{D}))) = \mathcal{P}^{\text{flat}}(\mathbf{D})$. □

Definition 5.13 (Preserving the Hierarchy). Consider a CFD $\mathbf{D} = (T, \mathcal{G}, \mathcal{C})$ with $T = (F, r, \_^{\uparrow})$ and let $\mathcal{L}$ be a language built over F. We say $\mathcal{L}$ *preserves the hierarchical*

[6] $\circ$ denotes composition.

structure of $\mathbf{D}$ if $\forall f, f' \in F : (f' \in f_{\downarrow\downarrow}) \iff \big(\forall w \in \mathcal{L} : (f' \in U_w) \Rightarrow (f \sqsubseteq_w f')\big)$[7]. $\square$

Theorem 5.2. For a given CFD $\mathbf{D}$, $\mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{D}))$ preserves $\mathbf{D}$'s hierarchy. $\square$

5.3 The ORE Transformation

In this section, we discuss the *Ordered sibling CFDs to Regular Expressions* (ORE) transformation. We first define *ordered siblings CFDs* (osCFDs) and *ordered siblings CRDs* (osCRDs), which are CFDs and CRDs, respectively, enriched with a partial order on nodes, called *sibling ordering*. We use the notations $\inf(R)$ and $\sup(R)$ to denote the infimum and supremum of a total ordering R, respectively.

Definition 5.14 (**Ordered Siblings CFDs).** An *ordered siblings CFD* (osCFD) is a tuple $\mathbf{OD} = (T, \mathcal{G}, \mathcal{C}, \leq_{\text{sib}})$ of the following components:

(i) $T = (F, r, \_^{\uparrow})$ is a tree, as defined in Definition 3.1.

(ii) $(T, \mathcal{G}, \mathcal{C})$ is a CFD, as defined in Definition 5.3. We call it the $\mathbf{OD}$'s *underlying CFD* and denote it by $\mathbf{OD}^{\text{cfd}}$.

(iii) $\leq_{\text{sib}}$ is a partial order on F, called *sibling ordering*, satisfying the following conditions:

(iii-i) $\forall f, f' \in F : f \leq_{\text{sib}} f' \implies f^{\uparrow} = f'^{\uparrow}$.

(iii-ii) $\forall S \in \text{Sib} : \leq_{\text{sib}} \big|_S$ (the restriction of $\leq_{\text{sib}}$ on S) is a total ordering, where $\text{Sib} = \{f_{\downarrow} \subset F : f \in F\}$.

(iii-iii) $\forall S \in \text{Sib}, \forall G \subset S, \forall f \in S \setminus G : G \in \mathcal{G} \implies$

$$(f \leq_{\text{sib}} \inf(\leq_{\text{sib}} \big|_G)) \vee (\sup(\leq_{\text{sib}} \big|_G) \leq_{\text{sib}} f).$$

The class of all osCFDs over the same set of features F will be denoted by $\mathcal{OD}(F)$.

[7] See page 24 for the definition of $\sqsubseteq_w$.

If needed, we will subscript **OD**'s components with index $_{\mathbf{OD}}$, e.g., write $\leq_{\text{sib}\mathbf{OD}}$.

□

An example could be an osCFD **OD** whose underlying CFD is the CFD **D** in Figure 5.1 and its sibling ordering $\leq_{\text{sib}}$ is the transitive closure of $\{(\mathsf{f}_1, \mathsf{f}_2), (\mathsf{f}_3, \mathsf{f}_4), (\mathsf{f}_4, \mathsf{f}_5), (\mathsf{f}_5, \mathsf{f}_6)\}$.[8] We will use this osCFD as a running example to illustrate the transformation procedure.

Let $\mathcal{OD}(\mathbf{D})$ denote the set of all osCFDs whose underlying CFDs are the same CFD **D**.

Definition 5.15 (**Ordered Siblings CRDs).** An *ordered siblings CRD* (osCRD) is a 4-tuple $\mathbf{ORD} = (LT_{re}, \mathcal{G}, \mathcal{C}, \leq_{\text{sib}})$ of the following components:

(i) $LT_{re} = (N, r, \_^{\uparrow}, \Sigma, l_{re})$ is a regular expression labeled tree as defined in Definition 5.4(i).

(ii) $(LT_{re}, \mathcal{G}, \mathcal{C})$ is a CRD, as defined in Definition 5.4. It is called the **ORD**'s *underlying CRD*, denoted by $\mathbf{ORD}^{\text{crd}}$.

(iii) $\leq_{\text{sib}}$ is a sibling ordering on N (see Definition 5.14(iii)).

The class of all osCRDs over the same alphabet Σ will be denoted by $\mathcal{ORD}(\Sigma)$.

If needed, we will subscript **ORD**'s components with index $_{\mathbf{ORD}}$, e.g., write $\leq_{\text{sib}\mathbf{ORD}}$. □

Obviously, osCRDs subsume osCFDs. Any notation used for CFDs and CRDs also works for osCFDs and osCRDs, respectively. These notations include $depth(\mathbf{ORD})$, $lev(\mathbf{ORD})$, $glev(\mathbf{ORD})$, $plev(\mathbf{ORD})$, $cplev(\mathbf{ORD})$. Please see page 97 for their meanings.

[8] As visual imaginary, one could consider a left to right ordering on siblings in Figure 5.1.

ORE is a procedure transforming a given osCRD to a regular expression. The procedure is similar to the CRE procedure. Like CRE, it is a bottom-up procedure including a finite number of steps called ORE-*shrinking* steps. Each step takes an osCRD and returns another osCRD such that the depth of the output's tree is less than that of the input. The last step returns a singleton osCRD. Each step includes three stages: (1) *eliminating multiplicities from leaves* (ORE-EML), (2) *eliminating grouped leaves* (ORE-EGL), and (3) *depth reduction* (ORE-DR).

5.3.1 ORE-EML

ORE-EML is like CRE-EML described in the previous section. It transforms a given osCRD to an osCRD whose underlying CRD is obtained by applying CRE-EML on the given osCRD's underlying CRD and the sibling ordering remains unchanged.[9]

Definition 5.16 (ORE**-EML Stage).** We define a function $\mathrm{mel}^{\mathsf{ORE}} : \mathcal{ORD}(\Sigma) \to \mathcal{ORD}(\Sigma)$, called ORE-EML function. For a given osCRD $\mathbf{ORD}$, $\mathrm{mel}^{\mathsf{ORE}}(\mathbf{ORD})$ is an osCRD $\mathbf{ORD}'$ where $\mathbf{ORD}'^{\mathrm{crd}} = \mathrm{mel}^{\mathsf{CRE}}(\mathbf{ORD}^{\mathrm{crd}})$ and $\leq_{\mathrm{sib}\mathbf{ORD}'} = \leq_{\mathrm{sib}\mathbf{ORD}}$ (see Definition 5.6 for the definition of $\mathrm{mel}^{\mathsf{CRE}}$). $\square$

Applying ORE-EML on our running example would result in an osCRD whose underlying CRD is shown in Figure 5.2(a) and sibling ordering is still a left-to-right ordering on siblings.

[9]Recall that CRE-EML does not change the tree structure of a given CRD. It just changes the labelling of the leaves.

5.3.2 ORE-EGL

Recall that for computing a regular expression corresponding to a set of grouped leaves in CRE-EGL, we consider all valid permutation of their regular expressions. In ORE-EGL, instead of considering all valid permutations, we consider the sibling ordering on the nodes to compute the corresponding regular expression. For an example, consider the grouped leaves $\mathsf{G} = \{r_4, r_5, r_6\}$ in Figure 5.2(a). Its corresponding ORE regular expression would be equal to $r_G = r'_G + r''_G$ where

$$r'_G = r_4r_5 + r_5r_6 + r_4r_6,$$

$$r''_G = r_4r_5r_6.$$

r'_G and r''_G model the lower bound and upper bound of the G's multiplicity, respectively. Note that all choices in both r'_G and r''_G observe the sibling ordering: r_4 must precedes r_5 and r_5 must precedes r_6 ($r_4 \leq_{\text{sib}} r_5 \leq_{\text{sib}} r_6$). Compare r'_G and r''_G with their CRE versions $r_4r_5+r_5r_4+r_5r_6+r_6r_5+r_4r_6+r_6r_4$ and $r_4r_5r_6+r_4r_6r_5+r_5r_4r_6+r_5r_6r_4+r_6r_4r_5+r_6r_5r_4$, respectively (page 102), in which we consider all permutations of r_4, r_5, r_6 with length 2 and 3, respectively. The number of choices for r_G is reduced from 12 in CRE to 4 in ORE. This reduction makes ORE computationally much cheaper than CRE. We will get back to this in Section 5.5.

Like CRE-EGL, G is replaced by a new node with a multiplicity $\{(1, 1)\}$ and is labeled with r_G. Figure 5.2(b) represents the result's underlying CFD. In ORE-EGL, we may need a new sibling ordering, as some nodes may be removed and some new nodes may be added. In our example, the grouped nodes labeled with r_4, r_5, and r_6 are removed and a node labeled with r_G is added. The new sibling ordering would be

$\{(r_1, \mathsf{f}_2), (r_3, r_G)\}$.

Let $Per_{\leq}^{(l,u)}(X)$ denote the set of all concatenation permutations S of X (a set of regular expressions) with length between l and u ($l \leq |S| \leq u$) considering a total ordering $\leq$ on X. For an example, $Per_{\leq}^{(2,3)}(\{a,b,c\})$ with $a \leq b \leq c$ would be the following set of expressions: $\{ab,\ ac,\ bc\} \cup \{abc\}$. We consider $Per_{\varnothing}^{(l,u)}(\varnothing) = \epsilon$ for any l and u.[10] By $Per_{\leq}^{k}(X)$ (for any $k \in \mathbb{N}$), we mean $Per_{\leq}^{(k,k)}(X)$.

Definition 5.17 (**Grouped Leaves** ORE**-Expressions).** Given an ocCRD $\mathbf{ORD} = (LT_{re}, \mathcal{G}, \mathcal{C}, \leq_{\text{sib}})$ with $LT_{\text{re}\leq} = (N, r, \_^{\uparrow}, \Sigma, l_{re})$, we define a total function $\text{gex}_{\mathbf{ORD}}^{\mathsf{ORE}} : glev(\mathbf{ORD}) \to \mathbf{RE}(\Sigma)$, which maps a grouped set of leaves in $\mathbf{ORD}$ to a regular expression built over Σ. For a given group $G \in glev(\mathbf{ORD})$ with $\mathcal{C}(G) = \{(l_i, u_i)\}_{1 \leq i \leq j}$ (for some $j \in \mathbb{N}$), $\text{gex}_{\mathbf{ORD}}^{\mathsf{ORE}}(G)$ is defined as follows.

$$\text{gex}_{\mathbf{ORD}}^{\mathsf{ORE}}(G) = r_1 + \ldots + r_j, \text{ where for all } 1 \leq i \leq j :$$

$$r_i = \sum X_i,$$

$$X_i = Per_{\leq_{\text{sib}}}^{(l_i,u_i)}(\{l_{re}(n) : n \in G\}).$$

□

We use the function gid defined in Definition 5.8 to assign a node identifier to each group all of whose elements are leaves.

Definition 5.18 (ORE**-EGL Stage).** $\text{gle}^{\mathsf{ORE}} : \mathcal{ORD}(\Sigma) \to \mathcal{ORD}(\Sigma)$ is a total function called ORE-*EGL* function. For a given osCRD $\mathbf{ORD} = (LT_{re}, \mathcal{G}, \mathcal{C}, \leq_{\text{sib}})$

[10] Clearly, $\leq$ would be also $\varnothing$ when its domain is empty.

with $LT_{\mathrm{re}\leq} = (N, r, \_^{\uparrow}, \Sigma, l_{re})$, $\mathrm{gle}^{\mathsf{ORE}}(\mathbf{ORD})$ is defined as follows:

$$\mathrm{gle}^{\mathsf{ORE}}(\mathbf{ORD}) = (LT'_{re}, \mathcal{G}', \mathcal{C}', \leq'_{\mathrm{sib}}), \text{ where}$$

$$LT'_{re} = (N', r, \_^{\uparrow'}, \Sigma, l'_{re})$$

$$N' = (N - glev(\mathbf{ORD})) \uplus N_G$$

$$\mathcal{G}' = \mathcal{G} - glev(\mathbf{ORD})$$

$$\mathcal{C}'(e) = \begin{cases} \{(1,1)\} & \text{if } e \in N_G \\ \mathcal{C}(e) & \text{otherwise} \end{cases}$$

$$n^{\uparrow'} = \begin{cases} \mathrm{gid}(n)^{\uparrow} & \text{if } n \in N_G \\ n^{\uparrow} & \text{otherwise} \end{cases}$$

$$l'_{re}(n) = \begin{cases} \mathrm{gex}^{\mathsf{ORE}}_{\mathbf{ORD}}(\mathrm{gid}(n)) & \text{if } n \in N_G \\ l_{re}(n) & \text{otherwise} \end{cases}$$

$$\leq'_{\mathrm{sib}} = \leq_{\mathrm{sib}} \big|_{N' - N_G} \cup R \cup L$$

$$R = \{(a, n) \in (N' - N_G) \times N_G : a \leq_{\mathrm{sib}} b \text{ for some } b \in \mathrm{gid}(n)\}$$

$$L = \{(n, a) \in N_G \times (N' - N_G) : b \leq_{\mathrm{sib}} a \text{ for some } b \in \mathrm{gid}(n)\}$$

□

5.3.3 ORE-DR

In this stage, we reduce the depth of a given osCRD by computing some regular expressions corresponding to the nodes all of whose child nodes are leaves. Recall that in CRE-DR, we consider all permutations of their associated regular expressions in the computation (see Definition 5.10). In ORE, instead of considering all permutations, we consider the sibling ordering on the nodes to compute the corresponding regular expressions. For an example, consider the osCRD whose underlying CRD in shown in Figure 5.2(b) with a left to right sibling ordering, i.e., $r_1 \leq_{\text{sib}} f_2$ and $r_3 \leq_{\text{sib}} r_G$. The node labeled with f_2 is the only node whose all children are leaves. Figure 5.2(c) shows the result, where $r_2 = \mathsf{f}_2(r_3 r_G)$. Consider r_2's CRE version $\mathsf{f}_2(r_3 r_{\mathsf{G}} + r_{\mathsf{G}} r_3)$ on page 105. The number of choices for r_2 is reduced from 2 in CRE to 1 in ORE.

Definition 5.19 (ORE**-Expressions for Parents of Leaves).** Given an osCRD $\mathbf{ORD} = (LT_{re}, \mathcal{G}, \mathcal{C}, \leq_{\text{sib}})$ with $LT_{\text{re}\leq} = (N, r, \_^{\uparrow}, \Sigma, l_{re})$, we define a total function $\text{pex}^{\mathsf{ORE}}_{\mathbf{ORD}} : plev(\mathbf{ORD}) \rightarrow \mathbf{RE}(\Sigma)$, which maps a parent whose child nodes are leaves to a regular expression. For a given node $n \in plev(\mathbf{ORD})$, $\text{pex}^{\mathsf{ORE}}_{\mathbf{ORD}}(n)$ is defined as follows:

$$\text{pex}^{\mathsf{ORE}}_{\mathbf{ORD}}(n) = l_{re}(n)(+\ Per^{|n_{\downarrow}|}_{\leq_{\text{sib}}}(E)), \text{ where}$$

$$E = \{l_{re}(n') : n' \in n_{\downarrow}\}.$$

□

Definition 5.20 (ORE**-DR Stage).** We define a function $\text{dre}^{\mathsf{ORE}} : \mathcal{ORD}(\Sigma) \rightarrow \mathcal{ORD}(\Sigma)$, called ORE-*DR* function, as follows: For a given osCRD $\mathbf{ORD} = (LT_{re}, \mathcal{G}, \mathcal{C},$

$\leq_{\text{sib}}$) with $LT_{\text{re}\leq} = (N, r, \_^{\uparrow}, \Sigma, l_{re})$, $\text{dre}^{\mathsf{ORE}}(\mathbf{ORD})$ is defined as follows:

$$\text{dre}^{\mathsf{ORE}}(\mathbf{ORD}) = (LT'_{re}, \mathcal{G}, \mathcal{C}', \leq'_{\text{sib}})$$

$$LT'_{re} = (N', r, \_^{\uparrow\prime}, \Sigma, l'_{re})$$

$$N' = N - \mathit{cplev}(\mathbf{ORD})$$

$$\_^{\uparrow\prime} = \_^{\uparrow}|_{N'}$$

$$\leq'_{\text{sib}} = \leq_{\text{sib}} \big|_{N'}$$

$$\mathcal{C}' = \mathcal{C}|_{N' \cup \mathcal{G}}$$

$$l'_{re}(n) = \begin{cases} \text{pex}^{\mathsf{ORE}}_{\mathbf{ORD}}(n) & \text{if } n \in \mathit{plev}(\mathbf{ORD}) \\ l_{re}(n) & \text{otherwise} \end{cases}$$

□

5.3.4 ORE-Shrinking Step and ORE

The ORE shrinking step is formalized as the composition of $\text{mel}^{\mathsf{ORE}}$ (ORE-EML stage), $\text{gle}^{\mathsf{ORE}}$ (ORE-EGL stage), and $\text{dre}^{\mathsf{ORE}}$ (ORE-DR stage).

Definition 5.21 (ORE**-Shrinking Step).** The function $\text{shr}^{\mathsf{ORE}} : \mathcal{ORD}(\Sigma) \to \mathcal{RD}(\Sigma)$ is called ORE-*shrinking* function and is defined as $\text{shr}^{\mathsf{ORE}} = \text{dre}^{\mathsf{ORE}} \circ \text{gle}^{\mathsf{ORE}} \circ \text{mel}^{\mathsf{ORE}}$.

□

We keep doing the shrinking steps until we get a singleton osCRD labeled with a regular expression. In the running example, we need to do the shrinking step once more. The final result would be the expression

$$r = \mathsf{f}(r_1 r_2'), \text{ where}$$

$$r_2' = \varepsilon + r_2 + r_2^2.$$

The notation $\mathcal{R}^{\mathsf{ORE}}(\mathbf{ORD})$ is used to denote the ORE regular expression generated for a given osCRD $\mathbf{ORD}$.

The following theorem shows the relationship between CRE and ORE. The language of the ORE regular expression for a given CFD would be a subset of the language of the CFD's CRE regular expression. Also, the Parikh image of both regular expressions would be the same.

Theorem 5.3. For any given osCFD $\mathbf{OD}$:

(i) $\mathcal{L}(\mathcal{R}^{\mathsf{ORE}}(\mathbf{OD})) \subseteq \mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{OD}^{\mathrm{cfd}}))$

(ii) $\mathrm{Par}(\mathcal{L}(\mathcal{R}^{\mathsf{ORE}}(\mathbf{OD}))) = \mathrm{Par}(\mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{OD}^{\mathrm{cfd}})))$ □

The following statement is the corollary of Theorem 5.3 and Theorem 5.1. It shows that the Parikh image of the ORE regular expression for a given osCFD captures the flat semantics of its underlying CFD.

Corollary 5.1. For a given osCFD $\mathbf{OD}$, $\mathrm{Par}(\mathcal{L}(\mathcal{R}^{\mathsf{ORE}}(\mathbf{OD}))) = \mathcal{P}^{\mathrm{flat}}(\mathbf{OD}^{\mathrm{cfd}})$. □

5.4 The HRE Transformation

HRE transforms a given osCFD to a regular expression such that the output regular expression provides a faithful semantics for the osCFD's underlying CFD. The regular expression is built on the set of features plus two extra symbols '$\lceil$' and '$\rceil$'. This differentiates HRE from ORE and CRE, as ORE and CRE regular expressions are built over features. The HRE transformation is based on the hierarchical semantics given in Chapter 4. The HRE procedure transforms, indeed, the hierarchical semantics of a given osCFD's underlying CFD to a regular expression. Thus, it provides a faithful semantics for CFDs, as hierarchical semantics of CFDs does so. Before getting to formal definitions and results, we informally describe the procedure on our running example $\mathbf{D}$ in Figure 5.1 considering a sibling ordering $\mathsf{f}_1 \leq_{\text{sib}} \mathsf{f}_2$, $\mathsf{f}_3 \leq_{\text{sib}} \mathsf{f}_4 \leq_{\text{sib}} \mathsf{f}_5 \leq_{\text{sib}} \mathsf{f}_6$. Let $\mathbf{OD}$ denote this osCFD.

Consider the hierarchical product $h_1 = \lceil \mathsf{f}, \lceil \mathsf{f}_1 \rceil^5 \rceil$ of $\mathbf{D}$ (see page 70). Considering the subfeature ordering, this tree-like multiset can be seen as a sequence of symbols $w_1 = \lceil \mathsf{f} \lceil \mathsf{f}_1 \rceil \lceil \mathsf{f}_1 \rceil \lceil \mathsf{f}_1 \rceil \lceil \mathsf{f}_1 \rceil \lceil \mathsf{f}_1 \rceil \rceil$. Another example: consider the hierarchical product $h_2 = \lceil \mathsf{f}, \lceil \mathsf{f}_1 \rceil^5, \lceil \mathsf{f}_2, \lceil \mathsf{f}_3 \rceil^3, \lceil \lceil \mathsf{f}_4 \rceil, \lceil \mathsf{f}_5 \rceil \rceil \rceil \rceil$. Considering the sibling ordering, its corresponding sequence would be $w_2 = \lceil \mathsf{f} \lceil \mathsf{f}_1 \rceil \lceil \mathsf{f}_1 \rceil \lceil \mathsf{f}_1 \rceil \lceil \mathsf{f}_1 \rceil \lceil \mathsf{f}_1 \rceil \lceil \mathsf{f}_2 \lceil \mathsf{f}_3 \rceil \lceil \mathsf{f}_3 \rceil \lceil \mathsf{f}_3 \rceil \lceil \lceil \mathsf{f}_4 \rceil \lceil \mathsf{f}_5 \rceil \rceil \rceil \rceil$. HRE transforms the osCFD to a regular expression whose language is the set of all such sequences of symbols. We call this expression the $\mathbf{OD}$'s HRE regular expression, denoted by $\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD})$.

The HRE transformation is a top-down procedure. This is another difference between this approach and the others, CRE and ORE. We define a regular expression for each group and each node. The expressions defined for nodes and groups are called HRE *node* and *group* expressions, respectively. We use the notation $\mathcal{R}^{\mathsf{HRE}}(f)$ and $\mathcal{R}^{\mathsf{HRE}}(G)$ to denote the regular expression associated with a node n and a group G, respectively. The regular expression $\mathcal{R}^{\mathsf{HRE}}(r) = \mathcal{R}^{\mathsf{HRE}}(\mathbf{OD})$, where r denotes the root, would be our desirable expression for the whole osCFD. In our running example, we would have the following node expressions: $\mathcal{R}^{\mathsf{HRE}}(\mathsf{f})$, $\mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_1)$, $\mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_2)$, $\mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_3)$,

$\mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_4)$, $\mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_5)$, $\mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_6)$. We also have a group expression $\mathcal{R}^{\mathsf{HRE}}(\mathsf{G})$.

Each HRE node expression is defined based on its multiplicity domain, and the expressions corresponding to its sub nodes and groups. In the running example, we would have

$$\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}) = \lceil \ \mathsf{f}\ \mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_1)\ \mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_2) \rceil,$$

where $\mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_1)$ and $\mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_2)$ denote the HRE regular expressions corresponding to the f's subfeatures f_1 and f_2, respectively. Note that f precedes $\mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_1)$ and $\mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_2)$. Indeed, this is a general rule: to model subfeature relationship between features, the expression corresponding to a feature must precede any other expressions corresponding to its subfeatures.[11] Also, due to $\mathsf{f}_1 \leq_{\text{sib}} \mathsf{f}_2$, $\mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_1)$ precedes $\mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_2)$.

$$\mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_1) = \mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}^{\mathsf{f}_1}) + \left(\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}^{\mathsf{f}_1})\right)^2 + \left(\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}^{\mathsf{f}_1})\right)^4 \left(\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}^{\mathsf{f}_1})\right)^*$$

$\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}^{\mathsf{f}_1})$ denotes the HRE regular expression of the osCFD induced by f_1, which is equal to $\lceil\ \mathsf{f}_1\ \rceil$. The choices in $\mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_1)$ together model the multiplicity domain $(1,2)(4,^*)$ on the node f_1.[12]

$$\mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_2) = \varepsilon + \mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}^{\mathsf{f}_2}) + (\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}^{\mathsf{f}_2}))^2$$

[11] This general rule is observed in ORE and CRE too.

[12] Note that '(' and ')' are not in our alphabet. They are used to group expressions in regular expressions, while '⌈' and '⌉' are symbols of the alphabet.

$\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}^{\mathsf{f}_2})$ denotes the HRE regular expression corresponding to the diagram induced by f_2. $\mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_2)$ models the multiplicity domain $(0,2)$ on f_2 properly.[13]

$$\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}^{\mathsf{f}_2}) = \lceil\ \mathsf{f}_2\ \mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_3)\ \mathcal{R}^{\mathsf{HRE}}(\mathsf{G})\ \rceil$$

In the above expression, $\mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_3)$ and $\mathcal{R}^{\mathsf{HRE}}(\mathsf{G})$ denote the HRE regular expression corresponding to f_3 and the group $\mathsf{G} = \{\mathsf{f}_4, \mathsf{f}_5, \mathsf{f}_6\}$, respectively. Since f_2 is the root feature of $\mathbf{OD}^{\mathsf{f}_2}$ (the diagram induced by f_2), it must precede any other expressions corresponding to its subfeatures. Note that $\mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_3)$ precedes $\mathcal{R}^{\mathsf{HRE}}(\mathsf{G})$ due to the sibling ordering.[14] $\mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_3)$ and $\mathcal{R}^{\mathsf{HRE}}(\mathsf{G})$ are defined in the following:

$$\mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_3) = (\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}^{\mathsf{f}_3}))^3 + (\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}^{\mathsf{f}_3}))^4 + (\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}^{\mathsf{f}_3}))^5$$

$\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}^{\mathsf{f}_3})$ denotes the HRE expression corresponding to the diagram induced by f_3, which is equal to $\lceil\ \mathsf{f}_3\ \rceil$. Note that the choices between different iterations of $\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}^{\mathsf{f}_3})$ together model the multiplicity domain $(3,5)$ on f_3.

$$\mathcal{R}^{\mathsf{HRE}}(\mathsf{G}) = \lceil(G_2 + G_3)\rceil$$

The multiplicity domain $(2,3)$ on the group G says that two or three elements of the group must be included in the group's expression $\mathcal{R}^{\mathsf{HRE}}(\mathsf{G})$. These two multiplicities

[13] $(\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}^{\mathsf{f}_2}))^0 = \varepsilon$.

[14] $\mathsf{f}_3 \leq_{\mathrm{sib}} \inf(\mathsf{G})$.

are modelled, respectively, by the expressions G_2 and G_3:

$$G_2 = \mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_4)\ \mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_5) + \mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_4)\ \mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_6) + \mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_5)\ \mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_6)$$

$$\mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_4) = \mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}^{\mathsf{f}_4})$$

$$\mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_5) = \mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}^{\mathsf{f}_5})$$

$$\mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_6) = \mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}^{\mathsf{f}_6}) + (\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}^{\mathsf{f}_6}))^2$$

There are three choices for the group multiplicity 2: choosing either "$\mathsf{f}_4, \mathsf{f}_5$", "$\mathsf{f}_4, \mathsf{f}_6$", or "$\mathsf{f}_5, \mathsf{f}_6$". In the building of the expression G_2, we also consider the ordering $\mathsf{f}_4 \leq_{\text{sib}} \mathsf{f}_5 \leq_{\text{sib}} \mathsf{f}_6$. Due to the multiplicity domain $(1, 2)$ on the feature f_6, there are two different choices $\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}^{\mathsf{f}_6})$ and $(\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}^{\mathsf{f}_6}))^2$ for the f_6's HRE regular expression. $\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}^{\mathsf{f}_4})$, $\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}^{\mathsf{f}_5})$, and $\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}^{\mathsf{f}_6})$ are, respectively, equal to $\lceil\ \mathsf{f}_4\ \rceil$, $\lceil\ \mathsf{f}_5\ \rceil$, and $\lceil\ \mathsf{f}_6\ \rceil$, as they are all leaves.

$$G_3 = \mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_4)\ \mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_5)\ \mathcal{R}^{\mathsf{HRE}}(\mathsf{f}_6)$$

As for the group multiplicity 3, we must include all the group elements f_4, f_5, and f_6. Considering the sibling ordering, we would get the above expression for G_3.

It is easy to see that $w_1 = \lceil \mathsf{f}\ \lceil \mathsf{f}_1 \rceil^5 \rceil$ and $w_2 = \lceil \mathsf{f}\ \lceil \mathsf{f}_1 \rceil^5 \lceil \mathsf{f}_2\ \lceil \mathsf{f}_3 \rceil^3\ \lceil\ \lceil \mathsf{f}_4 \rceil\ \lceil \mathsf{f}_5 \rceil \rceil \rceil \rceil$ (see page 117) are in the language of $\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD})$.

To formalize the procedure, we first define an ordering on the solitary sub features and groups of a given feature in an osCFD.

Definition 5.22. Let $\mathbf{OD} = (T, \mathcal{G}, \mathcal{C}, \leq_{\text{sib}})$ be an osCFD with $T = (F, r, \_^{\uparrow})$ and $f \in F$. We define a total order $\leq^f_{\text{sib}} \subseteq \text{Ing}(f_\downarrow) \times \text{Ing}(f_\downarrow)$, where $\text{Ing}(f_\downarrow) \stackrel{\text{def}}{=} (\mathcal{S} \cap f_\downarrow) \cup (\mathcal{G} \cap 2^{f_\downarrow})$, as the smallest transitive relation satisfying the following conditions:

(i) $\forall e_1, e_2 \in \mathcal{S} \cap f_{\downarrow} : (e_1 \leq^{f}_{\text{sib}} e_2) \iff (e_1 \leq_{\text{sib}} e_2)$.

(ii) $\forall e \in \mathcal{S} \cap f_{\downarrow}, \forall G \in \mathcal{G} \cap 2^{f_{\downarrow}} : (e \leq^{f}_{\text{sib}} G) \iff (e \leq_{\text{sib}} \inf(G))$.

(ii) $\forall e \in \mathcal{S} \cap f_{\downarrow}, \forall G \in \mathcal{G} \cap 2^{f_{\downarrow}} : (G \leq^{f}_{\text{sib}} e) \iff (\sup(G) \leq_{\text{sib}} e)$. □

In our running example, we would have $\mathsf{f_1} \leq^{\mathsf{f}}_{\text{sib}} \mathsf{f_2}, \quad \mathsf{f_3} \leq^{\mathsf{f_2}}_{\text{sib}} \mathsf{G}$.

Definition 5.23 and Definition 5.24 show how to get HRE node and group regular expressions, respectively.

Definition 5.23 (HRE **Node Expressions).** Let $\mathbf{OD} = (T, \mathcal{G}, \mathcal{C}, \leq_{\text{sib}})$ be an osCFD with $T = (F, r, \_^{\uparrow})$. For a given node $f \in F$ with $\mathcal{C}(n) = \{(l_i, u_i)\}_{1 \leq i \leq j}$ (for some $j \in \mathbb{N}$), its HRE regular expression, denoted by $\mathcal{R}^{\mathsf{HRE}}(f)$, is defined as follows:

$$\mathcal{R}^{\mathsf{HRE}}(f) = r_1 + \ldots + r_j, \text{ where } \forall 1 \leq i \leq j :$$

$$r_i = \begin{cases} (\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}^f))^{l_i} + \ldots + (\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}^f))^{u_i} & \text{if } u_i \neq * \\ (\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}^f))^{l_i} \big(\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}^f)\big)^* & \text{otherwise} \end{cases}$$

See Definition 5.25 for $\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}^f)$. □

Definition 5.24 (HRE **Group Expressions).** Let $\mathbf{OD} = (T, \mathcal{G}, \mathcal{C}, \leq_{\text{sib}})$ be an osCFD with $T = (F, r, \_^{\uparrow})$. For a given group $G \in \mathcal{G}$ with $\mathcal{C}(G) = \{(l_i, u_i)\}_{1 \leq i \leq j}$ (for some $j \in \mathbb{N}$), its HRE regular expression, denoted by $\mathcal{R}^{\mathsf{HRE}}(G)$, is defined as follows:

$$\mathcal{R}^{\mathsf{HRE}}(G) = \lceil (r_1 + \ldots + r_j) \rceil, \text{ where for all } 1 \leq i \leq j :$$

$$r_i = \bigplus X_i$$

$$X_i = Per^{(l_i, u_i)}_{\leq_{\text{sib}}}(E) \text{ with } E = \{\mathcal{R}^{\mathsf{HRE}}(n) : n \in G\}$$

See Definition 5.23 for $\mathcal{R}^{\mathsf{HRE}}(n)$ for a node n. $\square$

The following definition shows how to get the HRE regular expression for a given osCFD.

Definition 5.25 (HRE **Expressions for osCFDs).** Let $\mathbf{OD} = (T, \mathcal{G}, \mathcal{C}, \leq_{\text{sib}})$ be an osCFD with $T = (F, r, \_^{\uparrow})$. Its HRE expression, denoted by $\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD})$, is defined as follows:

$$\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}) = \lceil\ r\ Per^{|E|}_{\leq^r_{\text{sib}}}(E)\ \rceil, \text{ where } E = \{\mathcal{R}^{\mathsf{HRE}}(e) : e \in \text{Ing}(r_{\downarrow})\}$$

See Definition 5.22 for $\leq^r_{\text{sib}}$ and $\text{Ing}(r_{\downarrow})$. $\square$

Remark 5.4. Note that if the tree of a given osCFD is singleton, then E in the above expression would be empty. In this case, $Per^{|E|}_{\leq^r_{\text{sib}}}(E)$ would be ε, which implies $\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD}) = \lceil r \rceil$.

5.5 Discussion on Transformations

In this section, we discuss advantages and disadvantages of the different transformations discussed in the previous sections. We will discuss them in terms of *faithfulness*, *reverse engineering*, *computational complexity*, and *automated analysis*.

(i) Faithfulness. Recall that we call a semantics of a given CFD faithful if it captures both the flat semantics and the hierarchy of the CFD.

The CRE regular expression for a given CFD provides a faithful semantics for the CFD. Indeed, it captures both the flat semantics and the hierarchy of the CFD. These have been proven in Theorems 5.1 and 5.2, respectively.

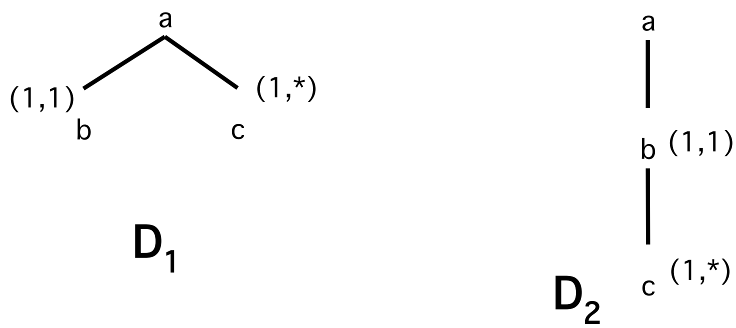


Figure 5.3: Difference between ORE and CRE: example

The ORE regular expression for a given CFD may not provide a faithful semantics for the CFD. However, as shown in Corollary 5.1, it captures the flat semantics of the CFD. Consider the two CFDs $\mathbf{D}_1$ and $\mathbf{D}_2$ in Figure 5.3. Their flat semantics are the same. However, their hierarchical structures are different. There are two osCFDs having $\mathbf{D}_1$ as their underlying CFD. Let us pick the one in which the sibling ordering is $\mathsf{b} \leq_{\text{sib}} \mathsf{c}$ and denote it by $\mathbf{OD}_1$. There is only one osCFD whose underlying CFD is $\mathbf{D}_2$ (The sibling ordering in this osCFD would be empty.). Let us denote this osCFD by $\mathbf{OD}_2$. According to the procedures described for ORE and CRE, we would have:

$$\mathcal{R}^{\mathsf{CRE}}(\mathbf{D}_1) = \mathsf{a}(\mathsf{bc}^+ + \mathsf{c}^+\mathsf{b})$$

$$\mathcal{R}^{\mathsf{CRE}}(\mathbf{D}_2) = \mathcal{R}^{\mathsf{ORE}}(\mathbf{OD}_1) = \mathcal{R}^{\mathsf{ORE}}(\mathbf{OD}_2) = \mathsf{abc}^+$$

As we see above, the CRE expressions of $\mathbf{D}_1$ and $\mathbf{D}_2$ distinguish between them, as expected. However, the ORE expressions of $\mathbf{OD}_1$ and $\mathbf{OD}_2$ are the same, which shows that ORE does not provide a faithful semantics for osCFDs' underlying CFDs. However, considering all osCFDs whose underlying CFDs are the same as a given CFD, we can get a faithful semantics for the given CFD via ORE. This is shown in the following proposition.

Proposition 5.1. For any given CFD $\mathbf{D}$: $+_{\mathbf{OD}\in\mathcal{OD}(\mathbf{D})}\ \mathcal{R}^{\mathsf{ORE}}(\mathbf{OD}) = \mathcal{R}^{\mathsf{CRE}}(\mathbf{D})$. $\square$

As an example, consider again the CFD $\mathbf{D}_1$ in Figure 5.3. As already mentioned, $\mathcal{OD}(\mathbf{D}_1)$ includes two osCFDs: $\mathbf{OD}_1$ with the sibling ordering $\mathsf{b} \leq_{\text{sib}} \mathsf{c}$ and $\mathbf{OD}'_1$ with the sibling ordering $\mathsf{c} \leq_{\text{sib}} \mathsf{b}$. Composing $\mathcal{R}^{\mathsf{ORE}}(\mathbf{OD'}_1) = \mathsf{ac^+b}$ and $\mathcal{R}^{\mathsf{ORE}}(\mathbf{OD}_1) = \mathsf{abc^+}$ via the choice operation $+$, we get to $\mathcal{R}^{\mathsf{CRE}}(\mathbf{D}_1) = \mathsf{a(bc^+ + c^+b)}$.

The HRE regular expression for a given CFD provides a faithful semantics for the CFD, as it captures the hierarchical semantics of the CFD. However, there is *subtle difference* between faithfulness of the CRE and HRE regular expressions of a given CFD. The HRE regular expression *explicitly* distinguishes between *grouped* and *solitary* features, while the CRE one does not. As an example, consider the CFD $\mathbf{D}_1$ in Figure 5.4 (consider all the multiplicity domains as the default one $(1, 1)$). The language of its CRE and HRE regular expressions would be the following sets, respectively:

$$\mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{D}_1)) = \{\ ab,\ \ ac\ \},$$

$$\mathcal{L}(\mathcal{R}^{\mathsf{HRE}}(\mathbf{D}_1)) = \{\ \lceil a\ \lceil\lceil b\rceil\rceil\rceil,\ \ \lceil a\ \lceil\lceil c\rceil\rceil\rceil\ \}.$$

We see that one can recognize the feature b (c, respectively) from the element $\lceil a, \lceil\lceil b\rceil\rceil\rceil$ ($\lceil a, \lceil\lceil c\rceil\rceil\rceil$, respectively) of the HRE language as a grouped feature, while this is not the case in the CRE language. This is because, as mentioned already, the HRE transformation explicitly models groups using the extra symbols, the brackets. Hence, the HRE transformation is the only transformation explicitly capturing the syntax of a given CFD.

(ii) Reverse Engineering. Since HRE explicitly captures the syntax of CFDs,

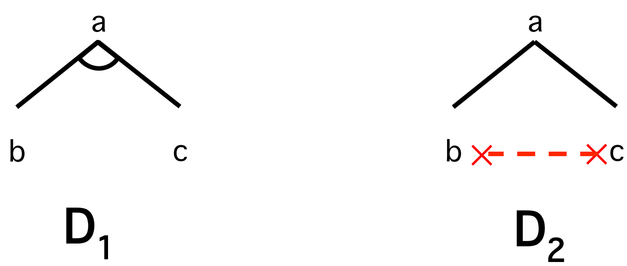


Figure 5.4: Faithfulness in CRE and HRE: example

it would be the best candidate for reverse engineering of CFMs. As a simple example, consider the CFDs $\mathbf{D}_1$ and $\mathbf{D}_2$ in Figure 5.4 (ignore the exclusive constraint between b and c in $\mathbf{D}_2$ for a while). $\mathbf{D}_2$'s HRE and CRE languages are as follows (let us suppose a left-to-right ordering on siblings as the sibling ordering on $\mathbf{D}_2$):

$$\mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{D}_2)) = \{\ abc,\ \ acb\ \},$$

$$\mathcal{L}(\mathcal{R}^{\mathsf{HRE}}(\mathbf{D}_2)) = \{\ \lceil a\ \lceil b \rceil\ \lceil c \rceil \rceil\ \}.$$

We see that both the CRE and HRE languages distinguish between $\mathbf{D}_1$ and $\mathbf{D}_2$.

Now, consider an exclusive constraint between b and c on $\mathbf{D}_2$, as shown in Figure 5.4. Let $\mathbf{M}$ denote this CFM. $\mathbf{M}$'s CRE and HRE languages are as follows: (Let $\mathcal{L}^{\mathsf{CRE}}(\mathbf{M})$ and $\mathcal{L}^{\mathsf{HRE}}(\mathbf{M})$ denote their corresponding languages, respectively.)[15]

$$\mathcal{L}^{\mathsf{CRE}}(\mathbf{M}) = \{\ ab, ac\ \},$$

$$\mathcal{L}^{\mathsf{HRE}}(\mathbf{M}) = \{\ \lceil a\ \lceil b \rceil \rceil,\ \ \lceil a\ \lceil c \rceil \rceil\ \}.$$

[15] In Section 5.6, we will discuss how to give a language interpretation of CCs and how to integrate the languages of CCs and CFMs.

As we see, $\mathcal{L}^{\mathsf{CRE}}(\mathbf{M}) = \mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{D}_1))$, while $\mathcal{L}^{\mathsf{HRE}}(\mathbf{M}) \neq \mathcal{L}(\mathcal{R}^{\mathsf{HRE}}(\mathbf{D}_1))$. Therefore, HRE does capture the difference between $\mathbf{M}$ and $\mathbf{D}_1$, while CRE does not.

In summary, both CRE and HRE transformations provide faithful semantics for CFDs. However, HRE models the syntax of a given CFD explicitly, while CRE does not. Considering CCs on CFDs and turning to CFMs, CRE languages may not work very well, but HRE languages do. This shows that HRE could be the best candidate for reverse engineering of CFMs.

(iii) Computational Complexity.[16] Recall that in stage 2 (stage 3, respectively) of the CRE transformation procedure, we consider *all* valid permutations of a given group (a node, respectively) all of whose elements (children, respectively) are leaves. Clearly, the ORE transformation does consider only one of the corresponding valid permutations in both stages 2 and 3. (Recall that ORE is given an osCFD instead of a CFD and hence consider the sibling ordering to compute the corresponding regular expressions.) Thus, the time complexity of CRE would be much more than ORE for a given CFD. The time complexity class of HRE would be the same as ORE's. This is because HRE, like ORE, works on osCFDs instead of CFDs.

According to the above informal discussion, we could say that the CRE transformation is computationally expensive, while the ORE and HRE transformations are cheap.

(iv) Automated Analysis. The CRE and ORE languages of a given CFD are built over the set of features, while the HRE's language is built over the set of features plus two extra symbols. To do analysis operations using the HRE interpretation, we will also need to manage and consider the extra symbols, as they have some semantical

[16] We are not going to provide a detailed and formal computational complexity analysis of the transformations. What we aim to do is to provide an intuition as to their complexity.

meanings.

Some analysis operations over an CFD rely on the flat semantics of the CFD (see Section 5.7). In such cases, we would prefer to work on the ORE regular expression, as it captures the flat semantics of the CFD and is cheaper than CRE. As for analysis operations regarding to the hierarchy of the CFD, we should choose either CRE or HRE.

5.6 A Computational Hierarchy of CFMs

Crosscutting Constraints (CCs) only make sense with respect to a given CFD. In the previous sections, we formalized the semantics of CFDs using formal languages (more precisely, regular languages). Hence, it makes sense to use the same framework, i.e., formal languages, to express CCs. This will allow us to integrate the semantics of CCs and CFDs. In this sense, a CC over a CFD can be any formal language built over the alphabet of the CFD. A set of CCs can be seen as the intersection of the languages expressing them.

In this way, a CFM would be basically a tuple of formal languages $(\mathcal{L}_{\mathbf{D}}, \mathcal{L}_{cc})$ with $\mathcal{L}_{\mathbf{D}}$ and $\mathcal{L}_{cc}$ denoting the formal languages of the CFD ($\mathbf{D}$) and CCs (cc), respectively. The formal language associated with the whole model, denoted by $\mathcal{L}_{\mathbf{M}}$, is then equal to $\mathcal{L}_{\mathbf{D}} \cap \mathcal{L}_{cc}$. The alphabet on which the CCs are expressed depends on the alphabet the language of the CFD is built on. In the following, we show how to translate the most common types of CCs using formal languages.

Consider a CFD $\mathbf{D}$ with a set of features F including three features f_1, f_2, and f_3. Several interesting CCs applied to the CFD are as follows:

(cc$_1$) f_1 requires f_2.

(in other words: If the number of occurrences of f_1 in a product is greater than 0, then the number of occurrences of f_2 in the product must be greater than 0).

(cc$_2$) f_1 excludes f_2.

(in other words: If the number of occurrences of f_1 in a product is greater than 0, then the number of occurrences of f_2 in the product must be 0).

(cc$_3$) If the number of occurrences of f_1 in a product is even, then the number of occurrences of f_2 in the product must be odd.

(cc$_4$) The number of occurrences of f_1 and f_2 in any product are equal.

(cc$_5$) The number of occurrences of f_1, f_2, and f_3 in any product are equal.

The first two CCs are traditional inclusive and exclusive CCs. However, they can be expressed in terms of feature occurrences, as we see in the parenthetical remarks above.

As already mentioned, the alphabet for defining CCs over a CFD is given by the alphabet over which the language of the CFD is built. Let Σ denote the underlying alphabet. Recall that $\Sigma = F$ for both CRE and ORE, while $\Sigma = F \cup \{\lceil, \rceil\}$ for HRE. In the following, we see the formal language interpretation of the above CCs. The formal language of a given CC cc is denoted by $\mathcal{L}(\mathrm{cc})$. Recall that $\#_a(w)$ denotes the number of occurrences of a letter a in a word w.

$$\mathcal{L}(cc_1) = \big\{w \in \Sigma^* : \big(\#_{f_1}(w) > 0\big) \Rightarrow \big(\#_{f_2}(w) > 0\big)\big\}.$$

$$\mathcal{L}(cc_2) = \big\{w \in \Sigma^* : \big(\#_{f_1}(w) > 0\big) \Rightarrow \big(\#_{f_2}(w) = 0\big)\big\}.$$

$$\mathcal{L}(cc_3) = \big\{ w \in \Sigma^* : (\exists n \in \mathbb{N}. \#_{f_1}(w) = 2n) \Rightarrow (\exists n \in \mathbb{N}. \#_{f_1}(w) = 2n + 1) \big\}.$$

$$\mathcal{L}(cc_4) = \big\{ w \in \Sigma^* : \#_{f_1}(w) = \#_{f_2}(w) \big\}.$$

$$\mathcal{L}(cc_5) = \big\{ w \in \Sigma^* : \#_{f_1}(w) = \#_{f_2}(w) = \#_{f_3}(w) \big\}.$$

Theorem 5.4. $\mathcal{L}(\mathrm{cc}_1)$, $\mathcal{L}(\mathrm{cc}_2)$, and $\mathcal{L}(\mathrm{cc}_3)$ are regular, $\mathcal{L}(\mathrm{cc}_4)$ is context-free, and $\mathcal{L}(\mathrm{cc}_5)$ is context-sensitive. □

Remark 5.5. What we need in cc_4 is counting the number of occurrences of f_1 and f_2. If the order of the symbols is ignored, then, according to Parikh's theorem [Par66], $\mathcal{L}(\mathrm{cc}_4)$ as a context-free language is not distinguishable from a regular language. This fact can be used in doing automated analysis of CFMs, as most of the language tools work for regular languages.

$\mathcal{L}_{\mathbf{D}}$ (the language of a CFD $\mathbf{D}$) can be obtained via three different transformations discussed in Chapter 5, i.e., it can be either the language of CRE, ORE, or HRE regular expressions of $\mathbf{D}$. As already discussed, depending on what we need to do and how, we choose one of these transformations (see Section 5.5). Thus, we define three different language-based semantics for CFMs:

Definition 5.26 (Language-based Semantics of CFMs). Given a CFM $\mathbf{M} = (\mathbf{D}, cc)$, where $\mathbf{D}$ is a CFD and cc is a set of CCs over $\mathbf{D}$, we define the following three language-based semantics for $\mathbf{M}$:

$$\mathcal{L}^{\mathsf{CRE}}(\mathbf{M}) \stackrel{\text{def}}{=} \mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{D})) \cap \mathcal{L}_{cc}$$

$$\mathcal{L}^{\mathsf{ORE}}(\mathbf{M}) \stackrel{\text{def}}{=} \mathcal{L}(\mathcal{R}^{\mathsf{ORE}}(\mathbf{D})) \cap \mathcal{L}_{cc}$$

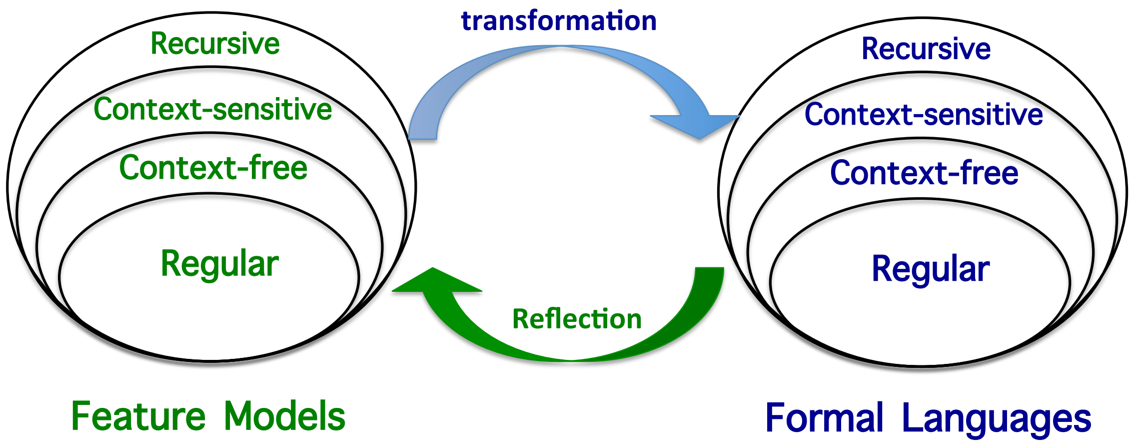


Figure 5.5: A Computational Hierarchy of CFMs

$$\mathcal{L}^{\mathsf{HRE}}(\mathbf{M}) \stackrel{\text{def}}{=} \mathcal{L}(\mathcal{R}^{\mathsf{HRE}}(\mathbf{D})) \cap \mathcal{L}_{cc}$$

They are called, respectively, the CRE, ORE, and HRE-language semantics of $\mathbf{M}$. □

The CRE and ORE languages for a given model $\mathbf{M}$ are built over the set of features, while the HRE language is over the set of features plus two extra symbols '$\lceil$' and '$\rceil$'. Recall that the ORE language of $\mathbf{M}$ can only capture the flat products of the CFM, while the CRE and HRE languages capture both the flat products and the hierarchy.

Regardless of what transformation we choose, the language of the CFD ($\mathcal{L}_{\mathbf{D}}$) is always regular, since $\mathcal{R}^{\mathsf{CRE}}(\mathbf{D})$, $\mathcal{R}^{\mathsf{ORE}}(\mathbf{D})$, and $\mathcal{R}^{\mathsf{HRE}}(\mathbf{D})$ are all regular expressions. It is a well-known fact in formal language theory that any class of languages is closed under intersection with regular languages [Dav94]. For instance, the intersection of a non-regular context-free (non-context-free context-sensitive, respectively) language with a regular language is a non-regular context-free language (non-context-free context-sensitive language, respectively). Due to this fact, the type of $\mathcal{L}_{\mathbf{M}}$ for a given CFM $\mathbf{M} = (\mathbf{D}, cc)$ is given by the type of $\mathcal{L}_{cc}$, as $\mathcal{L}_{\mathbf{D}}$ is always regular.

Now, consider a computational hierarchy of the classes of formal languages (one is shown in the right-hand side of Figure 5.5). Reflecting of the language hierarchy into the domain of feature models provides a computational hierarchy of feature models (see the left-hand side of Figure 5.5). This hierarchy is important because it guides us in how feature models can be constructively analyzed. See Section 5.7 where we discuss the decidability problems of analysis operations over CFMs.

Remark 5.6. The class of all basic feature models is a subclass of regular feature models, since the product family of a basic feature model is always finite.

Remark 5.7. It is worth mentioning that one can theoretically define even a non-recursive recursively enumerable (r.e.) CFM. For example, consider a CFM over a set of features F and $f \in F$. Let cc be a CC defined as follows: the set of valid numbers of occurrences of f is equal to the very well-known non-recursive r.e. set K [Coo03]. However, it is unlikely to find such a CC in practice.

5.7 Analysis Operations over CFMs

In this section, we discuss decidability problems corresponding to the analysis operations over CFMs.

Some analysis operations take only one CFM (along with another potential input that is not a CFM) as input and perform some analysis on the CFM. Below is a sample list of such operations:

Valid Configuration: The Valid Configuration operation takes a CFM and a flat multiset of features as inputs and decides whether it is a valid flat product of the CFM or not.

Partial Configuration: This operation takes a CFM and a flat multiset over features as inputs and decides whether it is a valid partial product of the CFM or not. A multiset m is a partial product of the CFM if there exists a flat product m' of the CFM such that $\forall a \in dom(m) : m(a) = m'(a)$.

Core Features: The Core Features operation takes a CFM and returns the set of features that are included in all products of the CFM.

Valid feature Multiplicity: The operation takes a CFM, a feature, and a natural number as inputs and decides whether the number is in the multiplicity domain of the feature or not.

Void Feature Model: This operation takes a CFM as input and decides whether its product line is empty or not.

Dead Feature: The Dead Feature operation takes a CFM and a feature and decides whether the feature is *dead* in the CFM or not. A feature f in a CFM $\mathbf{M}$ is called dead if $\nexists m \in \mathcal{P}^{\text{flat}}(\mathbf{M})$ such that $f \in dom(m)$.

Common Ancestors: The Common Ancestor operation takes a CFD and a set of features and returns their common ancestor features.

Least Common Ancestor: This operation takes a CFD and a set of features and returns their lowest common ancestor feature.

Some other operations deal with two CFMs. Such operations answer some questions about the relationships between the CFMs. The most well-known of such operations are *refactoring* and *specialization*.

Refactoring: The Refactoring operation takes two CFMs and decides whether their product line are equal or not.

Dynamic Refactoring: This operation takes two CFMs and decides whether their languages are equal or not.

Specialization: The specialization operation takes two CFMs $\mathbf{M}_1$ and $\mathbf{M}_2$ as inputs and decides whether the product line of $\mathbf{M}_1$ is a subset of the product line of $\mathbf{M}_2$ or not.

Dynamic Specialization: The operation takes two CFMs $\mathbf{M}_1$ and $\mathbf{M}_2$ as inputs and decides whether the language of $\mathbf{M}_1$ is a subset of the $\mathbf{M}_2$'s or not.

Remark 5.8. Most of the above analysis operations were originally defined and used on basic feature models. We have modified their definitions to be applicable to CFMs. As far as we know, some of the above operations are not defined in the literature. These operations include dynamic refactoring, dynamic specialization, and valid multiplicity.

Now, we address the decidability problems corresponding to the above operations. We consider the computational hierarchy of CFMs represented in Figure 5.5, i.e., the containment hierarchy Regular $\subset$ Context-free $\subset$ Context-sensitive $\subset$ Recursive.

Theorem 5.5. Given a recursive CFM, the operations Valid Product, Common Ancestors, and Least Common Ancestor are decidable. $\square$

Theorem 5.6. Given a context-free CFM $\mathbf{M}$, the operations Partial Configuration, Core Features, Valid feature Multiplicity, Void Feature Model, and Dead Feature are decidable. However, none of them is decidable in the class of context-sensitive CFMs. $\square$

Theorem 5.7. Given two CFMs $\mathbf{M}_1$ and $\mathbf{M}_2$, the following statements hold:

(i) If both are regular, then the (Dynamic) Refactoring problem between them is decidable.

(ii) If $\mathbf{M}_1$ and $\mathbf{M}_2$ are regular and context-free, respectively, then the (Dynamic) Refactoring problem is decidable iff $\mathbf{M}_1$ is bounded regular. $\square$

Remark 5.9. In general, the equality problem in the class of context-free languages is undecidable. Therefore, the Refactoring problem is not decidable in the class of context-free CFMs.

Theorem 5.8. Given two CFMs $\mathbf{M}_1$ and $\mathbf{M}_2$, the following statements hold:

(i) If both are regular, the (Dynamic) Specialization problem between them is decidable.

(ii) If $\mathbf{M}_1$ and $\mathbf{M}_2$ are regular and context-free, respectively, then the problem "is $\mathbf{M}_2$ a (dynamic) specialization of $\mathbf{M}_1$?" is decidable. □

Chapter 6

Related Work

6.1 Feature vs. Event Modeling

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

Following the survey in [vGP95], we distinguish three approaches in event modeling. The first is based on a topological notion of a configuration structure $(E, \mathcal{C})$ with E a (possibly infinite) set of *events*, and $\mathcal{C} \subset 2^E$ a family of subsets (usually finite) of events, which satisfy some closure conditions (e.g., under intersection and directed union). Sets from $\mathcal{C}$ are called *configurations* and understood as states of the system: $X \in \mathcal{C}$ is a state in which all events from X already occurred.

In the second approach, valid configurations are specified indirectly by some structure $\boldsymbol{D}$ of dependencies between events, which make some configurations invalid. Formally, some notion of *validity* of a set $X \subset E$ wrt. $\boldsymbol{D}$ is specified so that an *event structure* $(E, \boldsymbol{D})$ determines a configuration structure $\{X \subset E : X \text{ is } \textit{valid} \text{ wrt. } \boldsymbol{D}\}$.

Table 6.1: Event vs. feature modeling

Approach	Event Model	Feature Model	
		Boolean logic	Modal logic
Topological	$(E, \mathcal{C})$	$(F, \mathcal{PP}, \mathcal{FP})$	$(F, \mathcal{PP}, \rightarrow, I)$
Structural	$(E, \boldsymbol{D})$	$(F, \mathbf{M})$	
Logical	(E, Φ)	$(F, \Phi_{\mathsf{BL}}, \Phi^{!}_{\mathsf{BL}})$	(F, Φ_{ML})

Typical representatives of this approach are Winskel's prime and general event structures [Win82], and Pratt's event spaces [Pra91].

The third approach, originating in [GP93], is an ordinary encoding of sets of propositions by Boolean logical formulas. Then an event model is just a Boolean theory, i.e., a pair (E, Φ) with Φ a set of propositional formulas over set E of propositions. The left half of Table 6.1 summarizes this rough mini-survey.

Importantly, transitions between states are typically considered a derived notion: in [GP93], any set inclusion is a transition, and in [vGP95], special conditions are to hold in order for a set inclusion to be a valid transition. A notable exclusion is *event automata* in [PP95], i.e., tuples $(E, \mathcal{C}, \rightarrow, I)$ with $\rightarrow$ a *given* transition relation over configurations (states), and $I \in \mathcal{C}$ an initial state.

Feature modeling is directly related to event modeling, and actually can be seen as a special interpretation of event modeling. Indeed, features can be considered as events, (partial) products as configurations, and FMs as special event-structures: An FM $\mathbf{M} = (T_{\mathcal{OR}}, \mathcal{EX}, \mathcal{IN})$ can be seen as a special encoding of a set of dependencies analogous to $\boldsymbol{D}$ (the middle row of the table). An important distinction of the Boolean feature modeling is the presence of a special subset of *final* states (products), so that feature modeling's topological and logical counterparts are triples rather than pairs (see the Boolean column in the table). Pinna and Poigné in [PP95] mention final states (they call them *quiescent*) but do not actually use them, whereas for feature modeling, final products are a crucial ingredient.

The last column of the table describes feature modeling's basic topological and logical structures in the modal logic view: the upper row is our notion of ppKS, and the bottom one is the theory specified in Section 3.5. Our ppKS is exactly an event automaton with quiescent states, which, additionally, satisfies the conditions of Left-totality of the transition relations and Self-loops only, but Pinna and Poigné do not apply modal logic for specifying event automata's properties (and do not even mention it); they also do not consider the I2C-principle.

The comparison above shows enough similarities and differences to hope for a fruitful interaction between the two fields. We are currently investigating what feature modeling can usefully bring to event modeling; and can mention several simple findings. The presence of two separate Boolean theories allows us to formally distinguish between *enabling* and *causality* [GP93]. Also, we are not aware of propositional specifications of *transient* conflicts (discussed on page 41) such as our Boolean and modal encoding of I2C. These encodings are nothing but a compact formal specification of a transaction mechanism, which is usually considered to be non-trivial.

Remarkably, only recently similar generalizations were proposed for event modeling in the formalism of *DCR-Graphs* [HM11]. The latter also employ two relations between events, *condition* and *response*, that correspond to our *subfeature* and *mandatoriness* relations, and their *markings* roughly correspond to our partial products. DCR-Graphs also use two additional relations *include/exclude*, which allow them to model several important constructs in concurrent distributed workflow, including

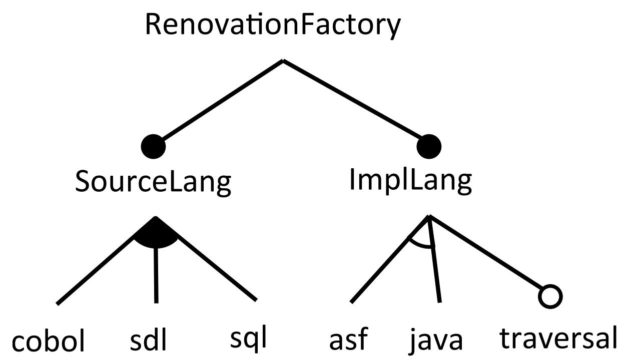


Figure 6.1: An FM adopted from [dJV02]

transient conflicts.

These observations show that a simple feature model formalism is capable of encoding complex modal theories specifying non-trivial concurrent phenomena. Specifically, a detailed comparative analysis of FMs and DCR-Graphs should be an interesting and we believe useful research task.

6.2 Grammars-based Semantics

In this section, we survey the literature relevant to the connection between feature modeling and formal languages. Indeed, it is directly related to what we have discussed in Chapter 5.

Batory and O'Malley were the first connecting software product lines to grammars [BO92]. In this work, the authors proposed a model called GenVoca, in which systems are defined by some functions that add features to programs. As shown in the paper, a set of such functions can be expressed as a grammar.

de Jong and Visser [dJV02] connected basic feature diagrams (FDs) and context-free grammars. They use textual representations of FDs written in a domain-specific

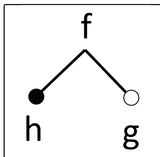
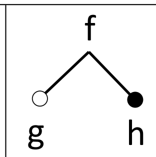
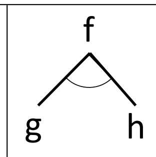
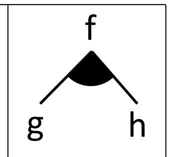
			
f$\rightarrow h[g]$	f$\rightarrow [g]h$	f$\rightarrow g \mid h$	f $\rightarrow$ t+ t $\rightarrow g \mid h$

Table 6.2: Translating FDs to iterative tree grammars

language called *feature description language* [VDK02]. The corresponding textual representation of a given FD is similar to a context-free grammar. The grammar generated for the FD in Figure 6.1, according to [dJV02], is as follows (nonterminals and terminals start with upper case and lower case symbols, respectively.):

RenovationFactory $\longrightarrow$ SourceLang ImplLang

SourceLang $\longrightarrow$ cobol | sdl | sql | cobol sdl | cobol sql | sdl sql | cobol sdl sql

ImplLang $\longrightarrow$ asf | java | asf traversal | java traversal

Batory, in [Bat05], shows the connection between FDs and iterative tree grammars [KRT08]. His and [dJV02]'s translation procedures are essentially the same. Table 6.2 gives some basic examples showing how Batory's encoding works. Terminals are denoted by italic symbols and optional features are surrounded by brackets.

In [dJV02] and [Bat05], the set of *atomic features* (features that appear in leaf nodes) is considered as terminals and other features as nonterminals. Thus, a word accepted in the above grammar generated for Figure 6.1 is a subset of {cobol, sdl, sql, asf, java, traversal}. Therefore, the language of the grammar does not represent the product line of the model. In other words, the corresponding generative grammar for

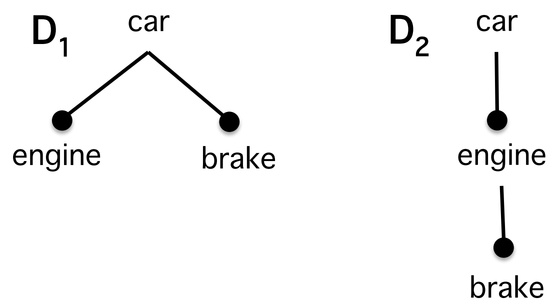


Figure 6.2: Two FDs with the same grammar in the de Jong and Visser approach

a given FD does not captures the product line semantics.

Another property of the above procedures is that they give a left-to-right ordering on siblings (the nodes with the same parent). To illustrate why this is a problem, note the left-most column in Table 6.2: the left-most feature, h, precedes the right-most feature, g. Such an ordering forces two syntactically equivalent FDs to have different semantics: the grammars of the two FDs in the first and the second columns in Table 6.2 have different associated languages. In addition, such an ordering on siblings results in the generative grammars not capturing the hierarchy of FDs, as both siblings and also subfeature relationships are ordered with the same operation (concatenation): In this sense, a feature precedes any of its subfeatures and also any of its siblings positioned after the feature (in a left-to-right ordering). As an example, consider the FDs in Figure 6.2. Their corresponding grammars in this approach would be the following grammars, respectively.

D_1 :

car $\longrightarrow$ engine brake

D_2 :

car $\longrightarrow$ engine

engine $\longrightarrow$ brake

Czarnecki et al, in [CHE05a], formalize the semantics of CFDs using context-free grammars. Unlike [Bat05] and [dJV02], this work considers the set of all features for a given CFD as the set of terminals. The generative grammar for a given CFD captures the flat semantics of the CFD. However, it gives a left-to-right ordering on siblings. Thus, this method does not capture the hierarchical structures of the CFD.

All the above approaches may result in ambiguous grammars, which makes them bad candidates for the semantics of feature modeling. However, there is a constructive way [Lin11] to fix this problem, since the languages of generated grammars are not inherently ambiguous. A context-free language is inherently ambiguous if there is no unambiguous grammar for it [Gin66].

Moreover, we consider any formal language built over the set of features as a CC over the given feature diagram. This is another difference between our approach and the above approaches. This provides the most expressive language for formally expressing CCs over cardinality-based feature diagrams. Also, this allows us to integrate the semantics of feature diagrams and crosscutting constraints.

6.3 Algebraic Approaches

Product Algebra. Höfner et al. developed an algebra, called *product family algebra*, for product lines whose basis is the structure of idempotent semirings [HKM11a]. A product family algebra over a set of features F is a 5-tuple $\mathbb{A} = (A, +, \varnothing, \times, \{\varnothing\})$ 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 considered as a term generated in this algebra.

The product line of a given FM **M** is encoded as a term in the product family algebra generated over the *prime features* of **M**; the latter are leaves in **M**'s FD. This is an important (meta)feature in the approach, which is in contrast to a common feature modeling practice. As an example, consider the FD in Figure 6.3, which is adopted from [HKM11a]. The encoded term corresponding to this feature model is as follows: $car = (manual + automatic) \times horsepower \times (1 + aircondition)$.

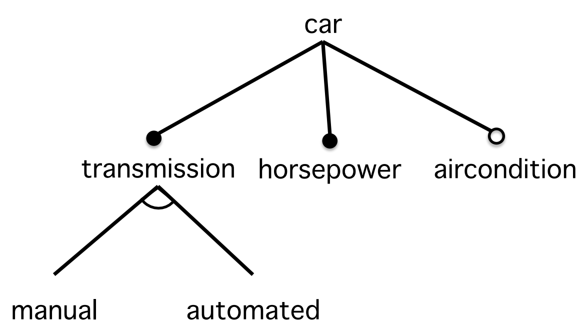


Figure 6.3: An FM adopted from [HKM11a]

To find a precise relation to semirings, we need to algebraicize our modal logic approach along the usual lines of algebraic logic — we leave this for future work. Some important distinctions can be stated immediately: For Höfner et al., a full product is a set of leaves in the feature tree, while non-leaf features are derived terms; in contrast, we follow a common feature modeling practice and consider all features in the tree to be basic. Also, their approach does not capture all semantics of FMs, as it is based on only the Boolean semantics of product lines. We believe that using Kripke structures and modal logic is simpler and easier for a product line engineer than dealing with abstract semiring algebra.

PL-CCS. Amongst algebraic models for product lines, the closest to ours is the *PL-CCS* calculus developed in [LT12, GLS08]. It is a process algebra, which extends the classical CCS by an operator $\oplus$ to model variability. $\oplus$ is a kind of choice applied at well-defined variation points. Each $\oplus$ occurence in a PL-CCS expression is equipped with a unique index, and runtime occurrences with the same index must make the same choice. This differentiates $\oplus$'s behaviour from the classical non-deterministic choice in CCS. In PL-CCS, processes are interpreted as products. The behaviour of a product line is given by a set of process definitions whose semantics is given by multi-valued Kripke structures.

There are interesting similarities and differences between PL-CCS and our ppKS. In PL-CCS, a product line's behaviour is reconstructed from an immediate product line specification. In contrast, we extract the behaviour from the feature model, which we have shown can be seen as an indirect product lines' specification providing everything needed to reconstruct the behavior. We might say that in PL-CCS, the expressive power of feature models is underestimated as they are seen in the Boolean perspective.

Importantly, PL-CCS allows for recursive definitions of processes, which makes it more expressive than our ppCTL. However, allowing recursive product definitions leads us beyond the boundaries of the tree-based feature models and our goals in the present paper. Iterative definitions are possible in cardinality-based feature models, and we built a dynamic semantics for them in Chapter 5. On the other hand, cross-cutting constraints cannot be expressed in PL-CCS, but are readily specified in our approach (we even allow for modal CCs).

6.4 Feature Transition Systems

In a series of papers summarized in [CCH+12, CCH+14, CCS+13, CHSL11, CHS+10], Classen *et al* proposed an elegant and effective solution to checking a given product line of transition systems in a single run of a model checker rather than checking each of the transition systems separately. In their setting, the entire product line is encoded as a *feature transition system* (FTS), in which transitions are labeled by both actions and Boolean expressions over features as Boolean variables. A truth assignment to the feature variables defines the behaviour of a single product, and the FTS as a whole represents the entire product line. They also defined a logic fCTL to allow CTL properties to refer to specific products in the line (those that include given features), and extended the model checking procedures to support checking FTSs against fCTL properties. Their tools are capable of reporting, in a single model checking run, all products for which a property holds, as well as those for which it fails to hold. In [CHL+14], Cordy *et al* extend a common model checking framework known as CEGAR, to support FTSs as well. Thus, FTS and our ppKS are orthogonal ideas: for the former, a product is a TS, while for us a product is a set of features without any functional properties. These two ideas can be combined in a single formalism, but we leave it for future work.

In contrast to the results cited above, our work is not concerned with the *functional behaviour* of the products. Rather, we concentrate on the semantics of the product *line* and the relationship between products and partial products in the line. In [CCH+14, CHSL11] the authors define fCTL, to allow CTL properties to refer to specific products in the line – those which include given features. This language differs both in intention and in structure from our ppCTL that uses a special symbol to identify final products.

6.5 Staged Configurations

Czarnecki *et al* introduced and developed the concept of *(multi-level) staged configuration* in [CHE04, CHE05b]: given an FM **M**, its full products are instantiated via consecutive specializations (called stages) of **M** by either discarding an optional feature or accepting it and hence making it mandatory for the stage at hand and all consecutive stages. This process is continued until a fully specialized FM (representing only one configuration) is reached.

The idea was further developed by Hubaux *et al* [HCH09], who proposed to map feature models to tasks and conditions of workflows. Their approach supports parallel execution of stages and choice between them, and iterative configurations.

Figure 6.4 presents an example of staged configuration for an FM **M**: In the first stage, a decision has been made between manual and automatic and manual is chosen. Note the right-hand FD in stage 1, where manual is now mandatory and automatic has disappeared. In the second stage, the optional feature pow has been discarded. The final result has been shown in the right-hand FD in stage 2.

Although both PPLs and configuration stages show how to instantiate full products, they are essentially different. Configuration paths are sequences of *feature models* with *decreasing variability*, whereas instantiation paths in PPLs are sequences of *products* with *increasing commonality.* Figure 6.5 shows the PPL of **M** in Figure 6.4. Also, 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 observing

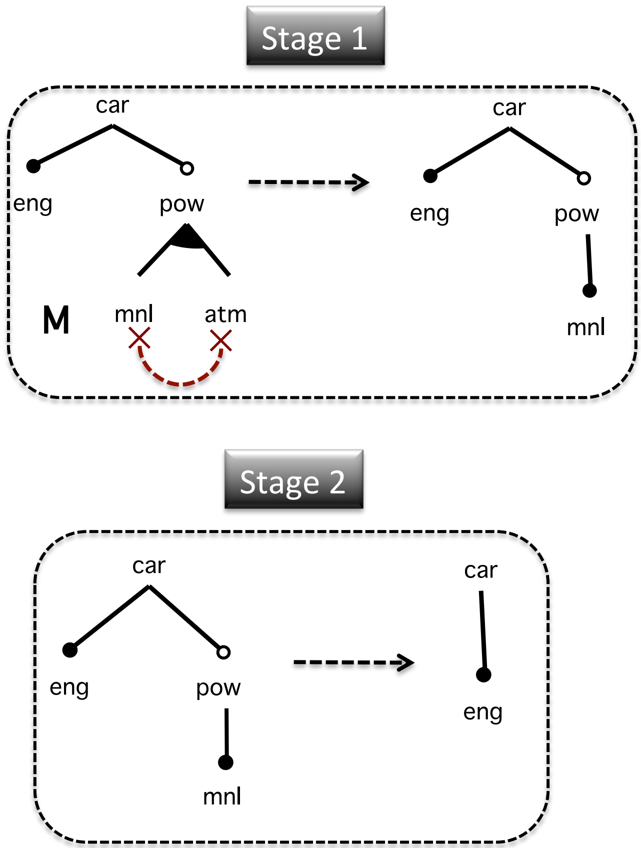


Figure 6.4: PPLs vs Staged configuration: a staged configuration

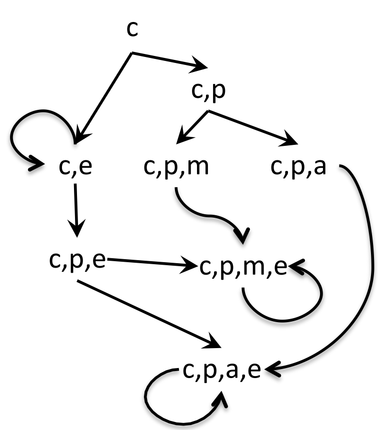


Figure 6.5: PPLs vs Stage configuration: A PPL

the constraints and the I2C-principle. For example, consider again Figure 6.4: in the first stage, we make choices between manual and automatic 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 be included in a partial product without its parent.

Thus, the two frameworks aim at different goals and are somewhat orthogonal (but, of course, PPLs cover variability too as full products are included into PPLs).

6.6 Other Formal Semantics

We have already discussed some formal semantics for feature modeling: propositional logic (Chapter 3), grammar-based (Section 6.2) and algebraic approaches (Section 6.3). There are some other approaches formalizing the semantics of basic feature modeling, including first-order logic, constraint programming, and description logic. We briefly discuss them in this section.

First Order Logic. Sun et al. propose a formal semantics for basic feature models using first-order logic (FOL) in Z [SZFW05]. They also proposed using the Alloy analyzer to reason about the consistency of a given feature model and its configurations. However, their FOL encoding of basic feature models captures only product lines (i.e., it does not capture their hierarchies.). Some other works like [GMB06] have applied an Alloy analyzer to give a theory for and reason about basic feature models.

Constraint Programming. Benavides et al. [BTC05] were the first using constraint programming to formalize basic feature models: A given FM is encoded as a constraint satisfaction problem (CSP) [Tsa93]. A CSP consists of a set of variables, a set of finite universes associated with variables, and a set of constraints on variables.

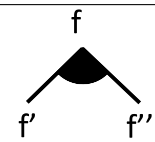
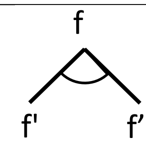
Optional	Mandatory	OR group	XOR group
			
if $(f = 0)$ $f' = 0$	$f = f'$	if $(f > 0)$ Sum(f', f'') in $\{1..2\}$ else $f' = 0, f'' = 0$	if $(f > 0)$ Sum(f', f'') in $\{1..1\}$ else $f' = 0, f'' = 0$

Table 6.3: Translating FDs to CSP

CSP solvers are used to determine whether there exists a solution for a given CSP or not. To encode a given FM into a CSP, the set of variables is considered as the set of features, the domains for each variable would be the set {0,1}, and the constraints between features are encoded into a constraint in CSP. Table 6.3 shows how to do this. Finally, the constraint $r == 1$ is added for the root feature r. As for CCs, inclusive and exclusive constraints would be in the form of $(f_1 > 0 \wedge \ldots \wedge f_n > 0) \rightarrow (f > 0)$ and $(f_1 > 0 \wedge \ldots \wedge f_n > 0) \rightarrow (f = 0)$, respectively.

Like the BL approach, the CSP-based approach for a given FM takes into account only the product line of the FM. Some empirical results show that CSP-based and BL-based automated analysis provide similar performance [BSTRC06a].

Description Logic. Utilizing description logic to encode the product line of a given basic feature model was proposed by Wang et al. in [WLS+05]. They showed how to translate an FM into OWL-DL ontologies [MVH+04], a decidable fragment of OWL. Some other works giving a description logic semantics for basic feature models are [FZ06] and [NEB+11].

Chapter 7

Conclusion and Future Work

7.1 Conclusion

(i) Modal Logic view of Basic Feature Modeling. We have presented a novel behavioural view of basic feature models, in which a product is an instantiation process rather than its final result. We called the states of these transition systems *partial products*, and showed that the set of partial products together with a set of (carefully defined) valid *transitions* between them can be considered as a special Kripke structure, whose properties are specifiable by a special fragment of CTL enriched with a constant modality. We called the logic ppCTL.

Our main result shows that a basic feature model can be considered as a compact representation of a rather complex ppCTL-theory. Thus, the logic of basic feature modeling is modal rather than Boolean.

We have also discussed several concrete tasks in basic feature modeling, which would be improved by the use of the modal logic view of feature models. These tasks include analysis of feature models, reverse engineering of feature models, and the developer vs. client view.

(ii) Multiset Theories of Cardinality-based Feature Diagrams. We have proposed two levels of generalization for cardinality-based feature diagrams. In the first generalization (Chapter 4), we have relaxed some constraints on group multiplicities. We believe that this simple generalization provides a more succinct and expressive tool for system modeling. The second generalization, called CRDs, has been proposed in Chapter 5 (see (iii) below.).

We have proposed two multiset theories for CFDs, called *flat* and *hierarchical*. A flat product of a given CFD is a multiset of features satisfying the multiplicity and subfeature constraints of the CFD. The set of all such multisets is called the *flat semantics* of the CFD. The flat semantics of a given CFD does not capture the CFD's hierarchy.

To define hierarchical products, we first defined a hierarchy of multisets over a finite set whose first class is the set finite multisets of features and other classes are defined as the set of all finite multisets built over the union of the previous classes. A hierarchical product of a CFD is defined as a multiset (in the corresponding multisets hierarchy) such that the rank of the multiset is given by the depth of the CFD and the multiplicities satisfy the multiplicity constraints of the CFD. The set of all hierarchical products is called the *hierarchical semantics* of the CFD. The hierarchical semantics of a given CFD provides a faithful semantics for the CFD.

We have proven that there is a bijection between flat and hierarchical semantics of a given CFD, i.e., a hierarchical product is a hierarchical version of a flat product.

To characterize a multiset being a hierarchical product of a CFD, we proposed the notion of *tree-like multisets*: We have proven that a multiset can be a hierarchical product of some CFDs iff it is a tree-like multiset. Also, we have characterized a set of tree-like multisets being the hierarchical semantics of a CFD.

We have proven that the hierarchical semantics of a CFD provides the most faithful semantics. Indeed, one can get back to the CFD from its hierarchical semantics. We have also discussed several possible practical applications of the mutliset theories of CFDs.

(iii) Formal Language Theories of Cardinality-based Feature Models. We have proposed a generalization of CFDs (which emerged in the CRE regular expression translation procedure) called CRDs, in which the labels of nodes can be any regular expressions built over the set of features. We believe that CRDs provide us with a way of modeling much more complicated systems, in which we need to deal with structural (non-atomic) features, e.g., programming codes, etc. It also provides us with a tool to treat multi product line engineering (see Section 7.2).

We have provided three types of reduction processes, which allow us to go from a CFD to a regular expression. These procedures are denoted by CRE, ORE, and HRE.

CRE works for CRDs. The CRE expression for a given CFD is built over the set of features and has two main properties: it captures the hierarchical structure of the CFD; it also captures the flat semantics of the CFD. These properties enable us to confidently claim that this translation faithfully captures the semantics of CFDs.

The second procedure, ORE, works for osCRDs (*ordered siblings CRDs*).[1] An osCRD is a CRD enriched with a partial order on its siblings. The ORE regular expression for a given osCFD (*ordered sibling CFD*) is built over the set of features such that it captures the flat semantics of the osCRD's underlying CFD.

[1] This kind of models are defined to formalize the ORE transformation.

HRE takes an osCFD and outputs a regular expression built over the set of features plus two extra symbols: opening and closing square brackets. The HRE procedure transforms, indeed, the hierarchical semantics of a given osCFD's underlying CFD to a regular expression. Thus, it provides a faithful semantics for CFDs.

We have also discussed the advantages of each of the above transformations in Chapter 5. Some advantages of these three transformations are common between them: Regular languages have some nice computational properties. These properties, such as the decidability of the emptiness, inclusion, and equality problems, help us to propose algorithmic solutions for analysis operations over CFDs. In addition, the complexity class of all regular languages is SPACE(O(1)), i.e., the decision problems can be solved in constant space. Due to these nice computational properties, we can also claim that regular expressions provide a nice computable framework for reasoning about CFDs.

As for CCs, we have proposed a formal language interpretation of them. In this way, we could integrate the formal semantics of CFDs and CCs. Also, it allows us to group CFMs based on their semantics, which guides us in how to constructively analyze them.

We have characterized some existing analysis operations over CFMs in terms of on the language frameworks. This allows us to use some off-the-shelf language tools to do analysis of CFMs. Note that automated support for analysis over CFMs were always considered a challenging issue. We have also investigated the decidability problems of the introduced analysis operations over CFMs. We noted that some analysis operations are not decidable in all classes of CFMs.

7.2 Open Problems

In this section, we describe several problems that we believe are mathematically interesting and whose solutions would be practically useful.

(*i*) **Complete Axiomatic System for ppCTL.** Finding a sound and complete axiomatic system for ppCTL is theoretically interesting. It would be also important in practice to do automated analysis over basic feature models (see (ii) below). As we know, ppCTL is a fragment of CTL plus a constant modality !. Several sound and complete axiomatic systems have been proposed for CTL, including [EH88], [BF97], and [LS01]. We can take advantage of these axiomatic systems to approach a sound and complete axiomatic system for ppCTL.

(*ii*) **Automated Analysis of basic Feature Models.** To implement analysis operations over a given feature model $\mathbf{M}$, one could apply either a model checker or theorem prover. To apply a model checker, we would need to transform $\mathbf{M}$ to its PPL $\mathbb{P}(\mathbf{M})$ and characterize given analysis problems in terms of ppCTL formulas. We plan to implement the analysis operations over some realistic examples using existing model checking tools. To take advantage of theorem provers, we first need to have a complete axiomatic system for our logic. There exist some theorem provers such as BDDCTL [Mar05], CTL-RP [ZHD09], and MLSolver [FL10], which can be used for reasoning about the CTL formulas.

(*iii*) **The Class of ppKSs Produced by the Class of basic Feature Models**. One of the questions that have been left open in the thesis, is to axiomatically define the class of ppKSs produced by the class of feature models. We plan to address this problem.

(*iv*) **Process Algebras for ppKSs and basic Feature Models**. Industrial systems are often very complex. Therefore, software companies usually design their systems by utilizing smaller systems, which themselves are produced by other companies [ACLF10b]. Therefore, their corresponding feature models could be seen as a compound of several smaller feature models. In this sense, proper process algebras for defining complex feature models and their corresponding ppKSs become fundamental and essential. This would be also important in the area of multi product line engineering (see viii, below).

(*v*) **Strong version of I2C.** Recall that the current version of the I2C principle says that two incomparable features can be included together in a partial product if at least "one" of them has been already completely instantiated. The current version of this principle is unavoidable, if we would like to realize a step-by-step computation. Note that this is why ppKSs are enforced to satisfy the singletonicity condition (see Definition 3.9). However, in some contexts like concurrent systems, it also makes sense to consider a *stronger* version of the I2C principle: two incomparable features can be included together in a partial product if "both" of them have been already completely instantiated. We plan to specify such a stronger version of the I2C-principle, in which a full product instantiation is always a transaction (which corresponds to replacing disjunction by conjunction in the definition of theory $\Phi^{\mathsf{I2C}}_{\mathsf{BL}}(T_{\mathcal{OR}})$, row (3) in Table 3.1). To address this problem, we would first need to modify the definition of ppKSs, as the singletonicity condition would not hold anymore. The logic would be the same. However, the ppCTL theory of a given basic feature model satisfying the strong I2C principle would change (this would be the most challenging issue in this problem.).

(*vi*) **A New Modal Logic view of Event-based Modeling.** We have discussed the intriguing similarities between event modeling and feature modeling in

Section 6.1. We are considering investigating a new modal logic view of event-based models as one of our future tasks. We believe that ppKSs provide a good tool to model the behaviour of event-based concurrent systems. Indeed, we believe that ppKSs can address several challenging issues in the area including distinguishing between *causality* and *enabling*, *interleaving* and *true concurrency*, *choice* and *conflict*, modeling *dynamic conflicts*, and *transactions*. In this sense, ppCTL (or an enriched version of the logic[2]) could be considered as a logical specification language for concurrent systems.

(*vii*) **Linear Logic Theory of Cardinality-based Feature Models.** Girard in 1987 developed *linear logic* in [Gir87], which is a substructural logic[3]. The logic is interesting from a logical point of view and for computer science. The logic is sometimes called a "resource-conscious" logic [Tro92], as a logical formula in linear logic represents certain types of resources and, unlike classical logics, resources cannot be used as often as one likes. Although linear logic is a substructural logic, both classical and intuitionistic logic can be faithfully embedded into linear logic. This is because the logic also supports finitely many uses of resources of the same type by two modality-like operators (! and ?) called *exponentials*. This ability differentiates linear logic from all other substructural logics.

Conjunction and disjunction operators in all well-known logics other than linear logic are idempotent. This fact becomes clear via their set-theoretic semantics: Formulas, conjunction and disjunction are, interpreted as sets, intersection, and union,

[2]Probably, ppCTL will need to be enriched with some past-time modalities to express the concurrent phenomena in a simple and natural way.

[3]A substructural logic is a logic lacking one of the structural rules such as weakening and contraction.

respectively. To get a reasonable logical treatment of CFMs, we needed to move from sets to multisets (see Chapter 4). This means that we need a logic in which conjunction and/or disjunction are not idempotent. Clearly, the best existing candidate would be linear logic. We plan to give a linear logic theory of multiset semantics of CFDs, discussed in Chapter 4. In this sense, we can also give a logical formulation of CCs: a CC over an CFD can be any linear logic formula built over the set of features.

(*viii*) **Multi Product Line Engineering via Formal Languages.** *Multi product line engineering* is an active area in feature modeling. A multi product line is a *product line of product lines* [ACLF10b]. We are going to give a formal language treatment for multi product line engineering. This could be done via CRDs and their CRE formal language-based semantics discussed in Chapter 5.

Recall that a CRD is a labelled CFD, where labels can be any regular expressions built over an alphabet. We showed that a CFD (and hence a basic FM — note that any propositional logic formula can be easily transformed into a regular expression) is a regular expression built over features. Thus, a CRD can be interpreted as a feature diagram of feature models modeling a multi product line.

(*ix*) **Feature Model Management.** *Feature model management* is an active area in feature modeling. By feature model management, we mean feature model composition via some operators like *merging*, *intersection*, and *union*, etc [SBRCT08, ACLF10a, ACC$^+$13].

Based on the closure properties of regular languages, say closure under intersection, union, complement, etc., we believe that our formal language framework is a very good candidate for managing feature models. We also plan to treat feature model management categorically.

(*x*) **Computational Complexity of Analysis Operations.** The computational complexity problem of analysis operations would be a crucial issue in implementing them for CFMs, and this needs to be investigated.

(*xi*) **OCL Definable Languages.** In the literature, the object-constraint language (OCL) has been proposed for expressing CCs in CFMs [CK05]. Our next mission is to discover the OCL-definable languages. It can be also fruitful for the model driven engineering (MDE) area (see xii, below), since the MDE community uses mainly OCL to express constraints. This way, we can investigate the expressiveness of OCL in terms of languages. Our conjecture is that there should be some practical CCs that cannot be expressed in OCL. Bellow, we provide some hints to support our conjectures.

It is a well-known *conjecture* that, theoretically, OCL is first order logic (FOL) plus transitivity and counting. FOL-definability leads to the class of star-free regular languages [DG08]. Considering transitivity, the class of OCL-definable languages would be equal to the class of regular languages. Considering the counting operation and equality, some context-free and sensitive languages are also covered. However, not all context-free languages can be expressed using only counting and equality. All the above conjectures need to be investigated theoretically.

(*xii*) **Metamodeling vs. Grammars.** The subject of transformation between metamodels and grammars (generally, formal languages) is an interesting practical subject in model driven engineering (MDE). As an example of its practical usefulness, one can consider bridging the gap between program codes (usually represented as grammars) and metamodels. Several relevant results have been published [AP$^+$04, WK06, Sch06, Kun08, BW13]. However, none of them perfectly addresses the problem and the problem is still a challenging and an open one.

CFMs can be interpreted as UML class models [CK05]. Indeed, UML class models could be considered as a generalization of CFMs. We believe that the research reported in the chapters 5 and 4 can be generalized to address the connection between metamodels and formal languages.

(*xiii*) **Automated Analysis of CFMs.** The theorems 5.5 to 5.8 (Chapter 5) are very important, as they state that "what operations would be decidable (*algorithmic*) in what classes of CFMs". Now a reasonable and important expectation is to address automated analysis of CFMs, which is a challenging and open issue in cardinality-based feature modeling [BSRC10, QRD13]. We believe that our formal language framework provides a nice computable framework for reasoning about CFMs.

There are several off-the-shelf language tools, including HKC [BP13], LIBVATA [LŠV12], RABIT [ACC+11], ALASKA [DWDMR08], GOAL [TCT+08], FSA6 [vN02], FAT [Hil09], JFLAP [RF06], and [GNS+15], which can be used to support automated analysis over CFMs based on our language-based semantics. We plan to implement the analysis operations over some realistic examples using formal language tools.

As a starting point, we briefly discuss how to use off-the-shelf tools to address automated analysis over regular CFMs. As discussed in Section 5.7, the class of regular CFMs is the only class over which all the analysis operations are decidable.

Most of the existing tools take finite state automata (FSA) as inputs, we first need to translate a given regular expression to an FSA. Some tools such as FSA6 [vN02] can be used to address this problem. Since CFDs and their CCs are translated to two different languages, we would also need to calculate their intersection. FAT and FSA6 are appropriate for implementing the intersection operation for two FSAs.

The Valid Configuration problem on a CFM is reduced to the membership problem on the CFM's language interpretations. FAT, JFLAP, and FSA6 address this problem.

The Void Feature Model problem is reduced to the emptiness problem on languages. The emptiness problem for a given language $\mathcal{L}$ can be seen as the equality problem between $\mathcal{L}$ and the empty language. The equality problem over FSA is supported by HKC.

The Dead Feature problem for a given CFM $\mathbf{M}$ and a feature f, can be reduced to the decision problem $\mathcal{L}(\mathbf{M}) \cap \mathcal{L}(F^* f F^*) = \varnothing$. Thus, the problem is the composition of intersection and emptiness problems, which are supported by FSA6 and HKC, respectively.

The refactoring (specialization, respectively) problem between two CFMs is simply reduced to the equality (inclusion, respectively) problem between their languages. The equality (inclusion, respectively) problem between FSA is supported by HKC.

(*xiv*) **Behavioural Feature Models.** By a *behavioural feature model*, we mean a feature model all of whose features possess a behaviour. Some specific feature interaction may be considered: (i) the behaviour of a feature may be affected by a selection of another feature in a product; (ii) the behaviour of a feature in a product may be affected by the behaviour of a selected feature in a valid product. Some papers relevant to this area are [BAS15, SA14] in which the behaviour of a feature (modelled by a finite state automaton) may be affected by a selection of another feature (the case (i), above). One of the challenging questions in this subject is to integrate the behaviours of features to get a single behaviour model for the whole feature model. We believe that CRDs provide a tool to address this problem in the systems where behaviours are expressed using finite state automata.[4] Note that in our transformation of CRDs to regular expressions (CRE), we do not consider the feature interaction. To be general, we would also need to consider interactions between features.

In a broader context, we would like to address a much more general problem in which behaviours of features are not restricted to finite state automata. In this context, we should apply a categorical approach.

[4]Finite state automata, left/right linear grammars, and regular expressions are three different tools for expressing regular languages. It has been proven that they all define the same class of formal languages [Lin11].

Appendix A

Proofs of Chapter 3

Theorem 3.1. If P is a valid partial product and $P \longrightarrow P'$, then P' is also a valid partial product.

Proof of Theorem 3.1. 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 $P' \models \bigcup\big\{\Phi^{!}_{\mathsf{BL}}(T^{g}_{\mathcal{OR}})\colon\, g \in P, g^{\uparrow}{=}f^{\uparrow})\big\}$. The above statement, along with $P \models \Phi^{\mathsf{I2C}}_{\mathsf{BL}}(T_{\mathcal{OR}})$, implies that $P' \models \big\{f{\wedge}g \to (\bigwedge\Phi^{!}_{\mathsf{BL}}(T^{f}_{\mathcal{OR}})){\vee}(\bigwedge\Phi^{!}_{\mathsf{BL}}(T^{g}_{\mathcal{OR}}))$ $\square$

Proposition 3.2. For all $P \in \mathcal{PP}$, there exists a full product P' such that $P \longrightarrow^{*} P'$, where $\longrightarrow^{*}$ is the reflexive transitive closure of of $\longrightarrow$.

Proof of Proposition 3.2. This is because a ppKS has a finite number of states, but infinite paths (as its transition relation is left-total). As all loops are self loops, a non-self-looped product must always get to a self-looped one through the transitions. $\square$

Theorem 3.2 (Soundness). $\mathbb{P}(\mathbf{M}) \models \Phi_{\mathsf{ML}}(\mathbf{M})$.

Proof of Theorem 3.2. To prove this theorem, we need to show that $\mathbb{P}(\mathbf{M})$ satisfies any components of the theory $\Phi_{\mathsf{ML}}(M)$.

(a) $\mathbb{P}(\mathbf{M}) \models \Phi_{\mathsf{BL}}(\mathbf{M})$ is obvious by to Definition 3.5. Thus, all the Boolean theories from Table 3.3 are satisfied by $\mathbb{P}(\mathbf{M})$.

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

Let $P \in \mathcal{PP}_{\mathbf{M}}$ 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 3.6) 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 3.7, there is a transition $P \longrightarrow_{\mathbf{M}} P'$. Therefore, $P \models \mathsf{EX}g$.

(c) $\mathbb{P}(\mathbf{M}) \models \Phi^{!}_{\mathsf{ML}}(\mathbf{M})$ follows obviously, since the set of states with self-loops in $\mathbb{P}(\mathbf{M})$ is equal to the set of all full products of $\mathbf{M}$. Note that this also implies that $\mathbb{P}(\mathbf{M})$ 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 $\Phi^{!}_{\mathsf{ML}\subseteq}(\mathbf{M})$.

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

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

Let f and P be a feature and a partial product of $\mathbf{M}$, respectively, such that $f \notin P$, $P \models \Phi_{\mathsf{BL}}^{\mathsf{I2C}}(f)$, and $P \not\models \bigvee \Phi_{\mathsf{BL}}^{\mathcal{EX}}(f)$. Thus, according to Definition 3.7, there exists a transition $P \longrightarrow_{\mathbf{M}} P \cup \{f\}$, which implies $P \models \mathsf{EX}\ f$. This results in $\mathbb{P}(\mathbf{M}) \models \Phi_{\mathsf{ML+}}^{\leftrightarrow}(T_{\mathcal{OR}}, \mathcal{EX})$.

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

Theorem 3.3 (Semi-completeness). $\mathbf{K} \models \Phi_{\mathsf{ML}\subseteq}(\mathbf{M})$ implies $\mathbf{K} \sqsubseteq \mathbb{P}(\mathbf{M})$.

Proof of Theorem 3.3. Let $\mathbf{K} \models \Phi_{\mathsf{ML}\subseteq}(\mathbf{M})$. $I_{\mathbf{K}} = I_{\mathbf{M}}$, since, due to $\mathbf{K} \models \Phi_{\mathsf{BL}}(T)$, $\mathbf{K} \models r$ (r is the root feature of $\mathbf{M}$).

Since $\mathbf{K} \models \Phi_{\mathsf{BL}}(\mathbf{M})$, according to Definition 3.5, $\mathcal{PP}_{\mathbf{K}} \subseteq \mathcal{PP}_{\mathbf{M}}$.

Now, we are going to show that $\longrightarrow_{\mathbf{K}} \subseteq \longrightarrow_{\mathbf{M}}$.

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

Consider a transition $P \longrightarrow_{\mathbf{K}} P'$, where $P' = P \cup \{f\}$ for a feature $f \notin P$. We want to show that there is a transition $P \longrightarrow_{\mathbf{M}} P'$ in $\mathbb{P}(\mathbf{M})$. Again, note that any state in $\mathbf{K}$ is a partial product of $\mathbf{M}$. To prove this statement, according to Definition 3.7, we need to show that (a1) $P' \models \Phi_{\mathsf{BL}}(T)$, (a2) $P' \models \Phi_{\mathsf{BL}}(\mathcal{EX})$, and (a3) $P \models \Phi_{\mathsf{BL}}^{\mathsf{I2C}}(P, f)$. (a1) and (a2) is an immediate corollary of $\mathbf{K} \models \Phi_{\mathsf{BL}}(\mathbf{M})$. To prove (a3), we need to show that for any siblings g with $g \in P$, $P \models \Phi_{\mathsf{BL}}^{!}(T_{\mathcal{OR}}^{g})$ (see Definition 3.6). Assume by a way of contradiction that $P \not\models \Phi_{\mathsf{BL}}^{!}(T_{\mathcal{OR}}^{g})$, i.e., g is not completely instantiated in P. Since $\mathbf{K} \models \Phi_{\mathsf{ML}\subseteq}^{\mathsf{I2C}\nrightarrow}(T_{\mathcal{OR}})$, $g \in P$, and $P \not\models \Phi_{\mathsf{BL}}^{!}(T_{\mathcal{OR}}^{g})$, there must not be a transition $P \longrightarrow_{\mathbf{K}} P'$. This leads us to a contradiction. Thus, (a3) holds.

Based on the above reasonings, $\longrightarrow_{\mathbf{K}} \subseteq \longrightarrow_{\mathbf{M}}$. $\square$

To prove Theorem 3.4 (the completeness theorem), we will first need the following lemmas A.1 and A.2.

Lemma A.1. $\mathbf{K} \models \Phi^{\circ}_{\mathsf{ML}}(\mathbf{M})$ *implies* $\mathcal{PP}_{\mathbf{K}} = \mathcal{PP}_{\mathbf{M}}$.

Proof of Lemma A.1. Let $\mathbf{K} \models \Phi^{\circ}_{\mathsf{ML}}(\mathbf{M})$. By Theorem 3.3, $\mathcal{PP}_{\mathbf{K}} \subseteq \mathcal{PP}_{\mathbf{M}}$. Now we need to show that $\mathcal{PP}_{\mathbf{M}} \subseteq \mathcal{PP}_{\mathbf{K}}$:

Let $P \in \mathcal{PP}_{\mathbf{M}}$ 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 $\{\mathsf{car}, \mathsf{engine}, \mathsf{gear}, \mathsf{manual}, \mathsf{oil}\}$ in the FM in Figure 2.1. We do have the following formulas corresponding to $\Phi(T)$: $\mathsf{engine} \to \mathsf{car}$, $\mathsf{gear} \to \mathsf{car}$, $\mathsf{manual} \to \mathsf{gear}$, and $\mathsf{oil} \to \mathsf{gear}$, which clearly represent the subtree $(\mathsf{engine}) \to \mathsf{car} \leftarrow (\mathsf{manual} \to \mathsf{gear} \leftarrow \mathsf{oil})$.

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 engine, since it is completely disassembled in $\{\mathsf{car}, \mathsf{engine}, \mathsf{gear}, \mathsf{manual}, \mathsf{oil}\}$. In this example, the traversal would result in the sequence $\langle \mathsf{car}, \mathsf{gear}, \mathsf{manual}, \mathsf{oil}, \mathsf{engine} \rangle$.

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 $\mathbf{K}$ and so P itself. To this end, we use the following inductive reasoning:

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

(*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 $\mathbf{K}$ and there exists a path $\{f_1\} \longrightarrow_{\mathbf{K}} \cdots \longrightarrow_{\mathbf{K}} \{f_1, \ldots, f_i\}$. 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 $\mathbf{K}$ and there exists the path $\{f_1\} \longrightarrow_{\mathbf{K}} \cdots \longrightarrow_{\mathbf{K}} P' \longrightarrow_{\mathbf{K}} P' \cup \{f_{i+1}\}$. To this end, we need to show that $P' \cup \{f_{i+1}\} \in \mathcal{PP}_{\mathbf{K}}$ and there exists a transition $P' \longrightarrow_{\mathbf{K}} P' \cup \{f_{i+1}\}$. 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 ($f_{i+1}^{\uparrow} = f_i$) due to $\mathbf{K} \models \Phi^{\downarrow}_{\mathsf{ML+}}(T)$, there is a transition $P' \longrightarrow_{\mathbf{K}} P' \cup \{f_{i+1}\}$. Hence, $\{f_1, \ldots, f_{i+1}\} \in \mathcal{PP}_{\mathbf{K}}$.

(R-2). As $\forall g \in P' : (g^{\uparrow} = f_{i+1}^{\uparrow}) \Rightarrow (P' \models \Phi^{!}_{\mathsf{BL}}(T^{g}_{\mathcal{OR}}))$ (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 $P' \models \Phi^{\mathsf{I2C}}_{\mathsf{BL}}(f_{i+1}) \wedge \neg \bigvee \Phi^{\mathcal{EX}}_{\mathsf{BL}}(f_{i+1}) \wedge \neg f_{i+1}$, and $\mathbf{K} \models \Phi^{\leftrightarrow}_{\mathsf{ML+}}(T_{\mathcal{OR}}, \mathcal{EX})$, there is a state $\{f_1, \ldots, f_{i+1}\} \in \mathcal{PP}_{\mathbf{K}}$ such that $P' \longrightarrow_{\mathbf{K}} P' \cup \{f_{i+1}\}$. Hence, $P \in \mathcal{PP}_{\mathbf{K}}$. $\square$

Lemma A.2. $\mathbf{K} \models \Phi_{\mathsf{ML}}(\mathbf{M})$ *implies* $\longrightarrow_{\mathbf{K}} = \longrightarrow_{\mathbf{M}}$.

Proof of Lemma A.2. Let $\mathbf{K} \models \Phi_{\mathsf{ML}}(\mathbf{M})$. There are two types of transitions in a ppKS: self-loop transitions and others. Note that self-loop transitions denote full products. We show that (1) full products of both $\mathbb{P}(\mathbf{M})$ and $\mathbf{K}$ are the same, i.e., the set of their self-loops are the same, (2) Non-loop transitions in $\mathbf{K}$ and $\mathbb{P}(\mathbf{M})$ are the same. (1) is obvious, since $\mathbf{K} \models \Phi^{!}_{\mathsf{ML}}(\mathbf{M})$ (note Table 3.1). In the following we also show that the statement (2) holds.

According to Theorem 3.3, $\longrightarrow_{\mathbf{K}} \subseteq \longrightarrow_{\mathbf{M}}$. Now what we need is to prove that any non-loop transition in $\mathbb{P}(\mathbf{M})$ is also a transition in $\mathbf{K}$. Note that, according to Lemma A.1, $\mathcal{PP}_{\mathbf{K}} = \mathcal{PP}_{\mathbf{M}}$. Consider a transition $P \longrightarrow_{\mathbf{M}} P'$, where $P' = P \cup \{f\}$ for a feature $f \notin P$. We want to show that there is a transition $P \longrightarrow_{\mathbf{K}} P'$ in $\mathbf{K}$. According to Definition 3.7, $P' \models \Phi_{\mathsf{BL}}(T) \cup \Phi_{\mathsf{BL}}(\mathcal{EX})$, and $P \models \Phi^{\mathsf{I2C}}_{\mathsf{BL}}(P, f)$. Thus, there are two choices:

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

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

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

(ii): Since $P' \models \Phi_{\mathsf{BL}}(\mathcal{EX})$, $P \models \neg \bigvee \Phi^{\mathcal{EX}}_{\mathsf{BL}}(f)$. 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 $P \models \Phi^{\mathsf{I2C}}_{\mathsf{BL}}(f)$. Hence, due to $\Phi^{\leftrightarrow}_{\mathsf{ML+}}(T_{\mathcal{OR}}, \mathcal{EX})$, there exists a transition $P \longrightarrow_{\mathbf{K}} P'$.

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

Theorem 3.4 (Completeness). $\mathbf{K} \models \Phi_{\mathsf{ML}}(M)$ iff $\mathbf{K} = \mathbb{P}(\mathbf{M})$.

Proof. Lemma A.1 shows that $\mathbf{K} \models \Phi^{\circ}_{\mathsf{ML}}(\mathbf{M})$ implies $\mathcal{PP}_{\mathbf{K}} = \mathcal{PP}_{\mathbf{M}}$. Lemma A.2

proves that $\mathbf{K} \models \Phi_{\mathsf{ML}}(\mathbf{M})$ implies $\longrightarrow_{\mathbf{K}} = \longrightarrow_{\mathbf{M}}$. Hence, $\mathbf{K} \models \Phi_{\mathsf{ML}}(M)$ implies $\mathbf{K} = \mathbb{P}(\mathbf{M})$. Considering the soundness theorem (Theorem 3.2), the completeness theorem is proven. $\square$

Appendix B

Proofs of Chapter 4

We first introduce some notations and complementary definitions used in the proofs.

Consider a multiset $m \in \mathcal{H}(A)$ for a set A. Let $x \in dom(m) \cup \mathsf{MultIng}(m)$. The notation $m[x/y]$ is used to denote a multiset generated by replacing any occurrence of x in m by an element $y \in \mathcal{H}(A)$. Formally,

$$m[x/y](e) = \begin{cases} m(x) + m(y) & \text{if } e = y \\ 0 & \text{if } e = x \\ m(e) & \text{otherwise} \end{cases}$$

As an example, consider the multisets $m = \lceil a^3, b^3 \rceil$. According to the above definition, $m[a/\lceil b \rceil] = \lceil \lceil b \rceil^3, b^3 \rceil$.

The following definition will be used in the proof of Lemma 4.1, Lemma **??**, and Lemma C.1.

Definition B.1 (Upper Diagram Induced by Depth). Let $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$ be a CFD and $1 \leq k \leq depth(\mathbf{D})$. The *upper diagram induced by* k is a CFD

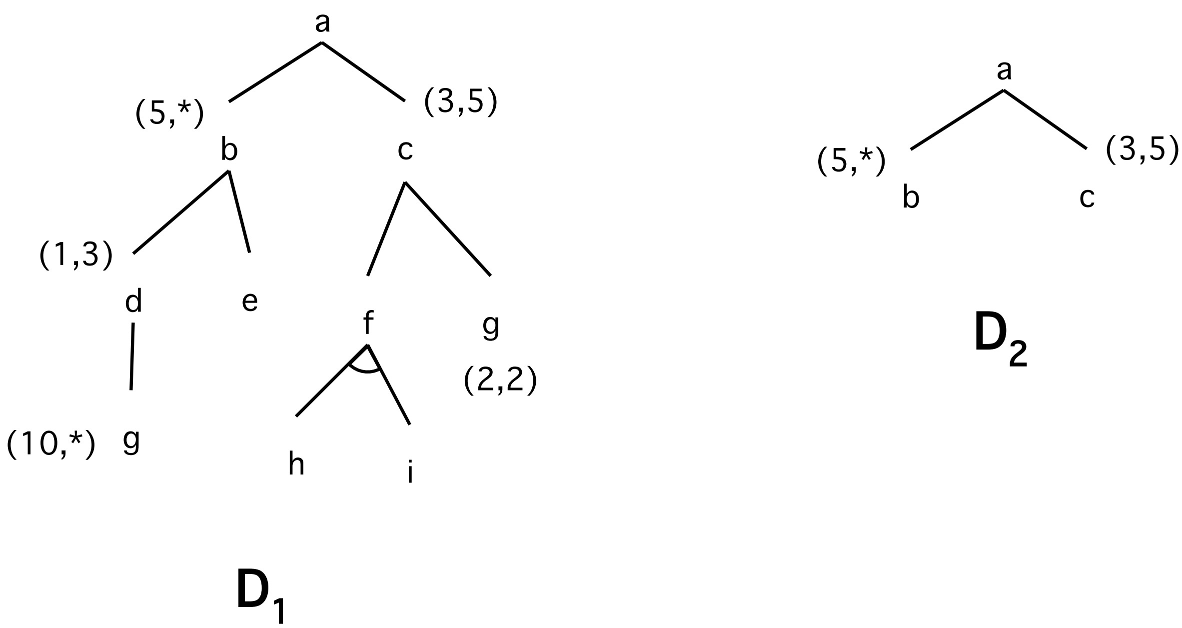


Figure B.1: $\mathbf{D}_2$: The diagram induced by depth 2 of $\mathbf{D}_1$

$\mathbf{D}_{-k} = (F', r, \_^{\uparrow}|_{F'}, \mathcal{G}', \mathcal{C}')$, where $F' = \{f \in F : \mathit{depth}(n) \leq k\}$, $\mathcal{G}' = \mathcal{G} \cap 2^{F'}$, and $\mathcal{C}' = \mathcal{C}|_{F' \uplus \mathcal{G}'}$, i.e., its tree is a subtree of $\mathbf{D}$'s tree where the nodes are in depth less than or equal to k; all other components are inherited from $\mathbf{D}$. □

For example, $\mathbf{D}_2$ is the upper diagram induced by depth 3 of $\mathbf{D}_1$ in Figure B.1.

Lemma 4.1. Given a CFD $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$, for any multiset m over F: $m \in \mathcal{P}^{\text{flat}}(\mathbf{D})$ iff m satisfies the following conditions:

(i) $m(r) = 1$,

(ii) $\forall f \in \mathcal{S} \cap r_{\downarrow},\ \exists c \in \mathcal{C}(f),\ \exists n \in \mathcal{P}^{\text{flat}}(\mathbf{D}^f),\ \forall e \in \mathit{dom}(n) : m(e) = c \times n(e)$.

(iii) $\forall G \in \mathcal{G} \cap 2^{r_{\downarrow}},\ \exists n \in \mathcal{P}^{\text{flat}}(\mathbf{D}, G),\ \forall e \in \mathit{dom}(n) : m(e) = n(e)$.

Proof of Lemma 4.1. For any CFD $\mathbf{D}$ and any flat multisets m over F, we show that both the following statements hold:

(1) $m \in \mathcal{P}^{\text{flat}}(\mathbf{D}) \implies m$ satisfies Th-(i), (ii), and (iii).[1]

(2) m satisfies Th-(i), (ii), (iii) $\implies m \in \mathcal{P}^{\text{flat}}(\mathbf{D})$.

Proof of (1):

We prove (1) by the following inductive reasoning on the depth of CFDs.

(*base case*): Consider a CFD $\mathbf{D}$ with $depth(\mathbf{D}) = 1$ and r as its root, i.e., $F_{\mathbf{D}} = \{r\}$ and any other components are empty. The only flat product is $m = \lceil r \rceil$. Holding each of the conditions Th-(i), (ii), and (iii) follows obviously, as $m(r) = 1$, $\mathcal{S} \cap r_{\downarrow} = \varnothing$, and $\mathcal{G} \cap 2^{r_{\downarrow}} = \varnothing$.

(*hypothesis*): Assume that for any CFD $\mathbf{D}$ with $depth(\mathbf{D}) < k$ (for some k), any $m \in \mathcal{P}^{\text{flat}}(\mathbf{D})$ satisfies the conditions Th-(i), (ii), and (iii).

(*inductive step*): We show that for any CFD $\mathbf{D}$ with $depth(\mathbf{D}) = k$, any $m \in \mathcal{P}^{\text{flat}}(\mathbf{D})$ satisfies the conditions Th-(i), (ii), and (iii).

Let $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$ be a CFD with $depth(\mathbf{D}) = k$ and $m \in \mathcal{P}^{\text{flat}}(\mathbf{D})$. Holding Th-(i) is clear. Let $\mathbf{D'} = \mathbf{D}_{-k}$ (the upper induced diagram of $\mathbf{D}$ by depth k, see Definition B.1) and $E = \{f \in F : depth(f) = k\}$. Let also $\mathcal{S}$ denote the set of solitary features in $\mathbf{D}$, i.e., $\mathcal{S} = \mathcal{S}_{\mathbf{D}}$.

Th-(ii):

(S-1): There exists $m' \in \mathcal{P}^{\text{flat}}(\mathbf{D'})$ such that $\forall f \in F \setminus E : m(f) = m'(f)$.

Since $depth(\mathbf{D'}) = k - 1$, due to the hypothesis,

$\forall f \in \mathcal{S} \cap r_{\downarrow} \setminus E, \exists c \in \mathcal{C}(f), \exists n' \in \mathcal{P}^{\text{flat}}(\mathbf{D'}^f),\ \forall e \in dom(n') : m'(e) = c \times n'(e)$.

Due to S-1 and the fact that $(F, r, \_^{\uparrow})$ is a tree of features,

(S-2): $\forall f \in \mathcal{S} \cap r_{\downarrow} \setminus E, \exists c \in \mathcal{C}(f), \exists n' \in \mathcal{P}^{\text{flat}}(\mathbf{D'}^f),\ \forall e \in dom(n') : m(e) = c \times n'(e)$.

Consider an arbitrary feature $f \in \mathcal{S} \cap r_{\downarrow} \setminus E$. There are unique $c \in \mathcal{C}(f)$ and

[1] Th-(i), (ii), and (iii) are abbreviations for Theorem 4.1(i), (ii), and (iii), respectively.

$n' \in \mathcal{P}^{\text{flat}}(\mathbf{D'}^f)$ satisfying (S-2).[2]

We define a multiset n'' as follows: $n'' = n' \uplus \lceil e^i : (e \in E \cap \mathcal{S}) \wedge (e^{\uparrow} \in dom(n')) \wedge (i = m(e)/c) \rceil$. According to (S-2), $\forall e \in dom(n'') : m(e) = c \times n''(e)$.

According to Def-(ii) and (iii)[3] and the assumption that n' is a flat product of $\mathbf{D'}^f$, there exists $n \in \mathbf{D}^f$ such that $\forall e \in (F \setminus E) \cup (E \cap \mathcal{S}) : n''(e) = n(e)$.

Therefore, according to above and (S-2),

$\forall f \in \mathcal{S} \cap r_{\downarrow}, \exists c \in \mathcal{C}(f), \exists n \in \mathcal{P}^{\text{flat}}(\mathbf{D}^f),\ \forall e \in dom(n) : m(e) = c \times n(e).$

Thus, Th-(ii) holds.

Th-(iii):

Let $G \in \mathcal{G} \cap 2^{r_{\downarrow}}$. We show that $\exists n \in \mathcal{P}^{\text{flat}}(\mathbf{D}, G),\ \forall e \in dom(n) : m(e) = n(e)$. There are the two following cases:

(a) $k = depth(\mathbf{D}) > 2$,

(b) $k = depth(\mathbf{D}) = 2$.

In the former case, $G \in \mathcal{G}_{\mathbf{D'}} \cap 2^{r_{\downarrow}}$. According to S-1, there exists $m' \in \mathcal{P}^{\text{flat}}(\mathbf{D'})$ such that $\forall f \in F \setminus E : m(f) = m'(f)$.

Since $depth(\mathbf{D'}) = k - 1$, due to the hypothesis,

$\exists n' \in \mathcal{P}^{\text{flat}}(\mathbf{D'}, G), \forall e \in dom(n') : n'(e) = m'(e)$.

Let $n = \left(\biguplus_{f \in X} \lceil f^{m(f)} \rceil\right) \uplus n'$, where $X = E \cap dom(m) \cap \{f_{\downarrow\downarrow} : f \in G\}$.

Clearly, $n \in \mathcal{P}^{\text{flat}}(\mathbf{D}, G)$.

Since $dom(n') \cap E = \varnothing$ and $\forall f \in F \setminus E : m(f) = m'(f)$, we get to

$\forall e \in dom(n) : n(e) = m(e)$. Thus, Th-(iii) holds in case (a).

---

[2] n' and c in (S-2) are unique multiset and multiplicity, respectively, for a given $f \in \mathcal{S} \cap r_{\downarrow} \setminus E$ satisfying the statement.

[3] Def-(i), (ii), and (iii) stand for Definition 4.3(i), (ii), and (iii), respectively.

Now, consider the case (b), where $depth(\mathbf{D}) = 2$. In this case, $G \subseteq E$.

Let $dom(m) \cap G = \{f_1, \ldots, f_j\}$ for some j.

Let $n = \biguplus_{1 \leq i \leq j} \lceil f_i^{m(f_i)} \rceil$.

Due to Def-(iii), $\forall 1 \leq i \leq j : m(f) \in \mathcal{C}(f)$: (1)

Since f_i is a leaf node in $\mathbf{D}$ for any $1 \leq i \leq j$, $\lceil f_i \rceil \in \mathcal{P}^{\text{flat}}(\mathbf{D}^{f_i})$: (2)

Due to Def-(iv), $j = |dom(m) \cap G| \in \mathcal{C}(G)$: (3)

(1), (2), and (3) together imply that $n \in \mathcal{P}^{\text{flat}}(\mathbf{D}, G)$. Since $\forall e \in dom(n) : m(e) = n(e)$, Th-(iii) holds in case (b) too.

Proof of (2):

Assume that a multiset m over the set of features satisfies Th-(i), (ii), (iii). We show that it also satisfies Def (ii), (iii), and (iv).

Def-(ii): Recall that Def-(ii) says that $\forall f \in F_{-r} : f \in dom(m) \implies (\exists c \in \mathcal{C}(f) : m(f) = c \times m(f^{\uparrow}))$.

Let $f \in F_{-r}$ and $f \in dom(m)$. Then, either $f \in \mathcal{S}$ or $\exists G \in \mathcal{G} : f \in G$.

Let us first consider the case $f \in \mathcal{S}$: Th-(ii) implies that there exists $c \in \mathcal{C}(f)$ and $n \in \mathcal{P}^{\text{flat}}(\mathbf{D}^f)$ such that $\forall e \in dom(n) : m(e) = c \times m(f^{\uparrow}) \times n(e)$. Since $T = (F, r, \_^{\uparrow})$ is a tree of features, $m(f) = n(f) \times c \times m(f^{\uparrow})$. Note that f is the root feature of $\mathbf{D}^f$, which means that, according to Definition 4.3, $n(f) = 1$. Thus, $m(f) = c \times m(f^{\uparrow})$ and Def-(ii) holds.

Now, let us consider the latter case, i.e., $\exists G \in \mathcal{G} : f \in G$. Consider such a G and let $G = \{f_1, f_2, \ldots, f_k\}$ for some k such that $f_1 = f$.

Th-(iii) and Th-(ii) together imply that there exists $n \in \mathcal{P}^{\text{flat}}(\mathbf{D}, G)$ such that $n^{m(f^{\uparrow})} \subseteq m$.

According to Definition 4.4, there exist $c \in \mathcal{C}(G)$, $c_i \in \mathcal{C}(f_i)$, $g_i \in \{0, 1\}$, and

$m_i \in \mathcal{P}^{\text{flat}}(\mathbf{D}^{f_i})$ such that $n = \biguplus_{1 \leq i \leq k} m_i^{c_i \times g_i}$, and $\sum_i g_i = c$.

Since $f \in dom(m)$, g_1 must be 1. (Note that $\mathbf{D}$ is an unlabelled tree of features.) Thus, $n(f) = m_1(f) \times c_1$.

Since f is the root feature of $\mathbf{D}^{f_1}$ and $m_1 \in \mathcal{P}^{\text{flat}}(\mathbf{D}^{f_1})$, $m_1(f) = 1$. Therefore, $n(f) = c_1$.

Since T is an unlabelled tree, $m(f) = n(f) \times m(f^{\uparrow})$. Therefore, $m(f) = c_1 \times m(f^{\uparrow})$ and Def-(ii) holds.

Def-(iii): Recall that Def-(iii) says that $\forall f \in \mathcal{S} : 0 \notin \mathcal{C}(f) \wedge m(f^{\uparrow}) > 0 \implies m(f) > 0$.

Let f be a solitary mandatory feature (i.e., $0 \notin \mathcal{C}(f)$) and its parent is in m (i.e., $m(f^{\uparrow}) > 0$). We want to show that f is in m too.

The conditions Th-(ii) and (iii) imply that there exists $c \in \mathcal{C}(f)$ and $n \in \mathcal{P}^{\text{flat}}(\mathbf{D}^f)$ such that $\forall e \in dom(n) : m(e) = c \times m(f^{\uparrow}) \times n(e)$. Therefore, $m(f) = n(f) \times c \times m(f^{\uparrow})$, as $f \in dom(n)$.

Since f is the root feature of $\mathbf{D}^f$, $n(f) = 1$ and $m(f) = c \times m(f^{\uparrow})$.

Since $0 \notin \mathcal{C}(f)$ and so $m(f^{\uparrow}) > 0$, $m(f) > 0$. Def-(iii) holds.

Def-(iv): Recall that Def-(iv) says that $\forall G \in \mathcal{G} : (m(G^{\uparrow}) > 0) \implies (|dom(m) \cap G| \in \mathcal{C}(G))$.

Consider an arbitrary group $G = \{f_1, f_2, \ldots, f_k\}$ with $m(G^{\uparrow}) > 0$.

The conditions Th-(ii) and (iii) imply that there exists $n \in \mathcal{P}^{\text{flat}}(\mathbf{D}, G)$ such that $\forall e \in dom(n) : m(e) = m(G^{\uparrow}) \times n(e)$.

According to Definition 4.4, there exist $c \in \mathcal{C}(G)$, $c_i \in \mathcal{C}(f_i)$, $g_i \in \{0, 1\}$, and $m_i \in \mathcal{P}^{\text{flat}}(\mathbf{D}^{f_i})$ such that $n = \biguplus_{1 \leq i \leq k} m_i^{c_i \times g_i}$, and $\sum_i g_i = c$.

The condition $\sum_i g_i = c$ implies that $|dom(m) \cap G| \in \mathcal{C}(G)$. Hence, Def-(iv) holds.

□

Theorem 4.1. Given two CFDs $\mathbf{D}$ and $\mathbf{D}'$, $\mathcal{P}(\mathbf{D}) = \mathcal{P}(\mathbf{D}') \implies \mathbf{D} = \mathbf{D}'$.

Proof of Theorem 4.1. Let $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$ and $\mathbf{D}' = (F', r', \_^{\uparrow'}, \mathcal{G}', \mathcal{C}')$ be two CFDs such that $\mathcal{P}(\mathbf{D}) = \mathcal{P}(\mathbf{D}')$.

Obviously, $\bigcup_{m \in \mathcal{P}(\mathbf{D})} dom(\mathsf{flat}_F(m)) = F$ and $\bigcup_{m' \in \mathcal{P}(\mathbf{D}')} dom(\mathsf{flat}_{F'}(m')) = F'$. Since $\mathcal{P}(\mathbf{D}) = \mathcal{P}(\mathbf{D}')$, $F = F'$. (S-1)

We give an inductive reasoning based on $depth(\mathbf{D})$ (the depth of $\mathbf{D}$) to show that $\mathbf{D} = \mathbf{D}'$.

(*base case*): Let $\mathbf{D} = 1$, i.e., $F = \{r\}$, and $\_^{\uparrow} = \mathcal{G} = \mathcal{C} = \varnothing$. According to (S-1), $F' = \{r\}$, which implies that $r' = r$, $\_^{\uparrow'} = \mathcal{G}' = \mathcal{C}' = \varnothing$. Therefore, $\mathbf{D} = \mathbf{D}'$.

(*hypothesis*): Assume that for some $n \in \mathbb{N}$ and for any $depth(\mathbf{D}) < n$: $\mathcal{P}(\mathbf{D}) = \mathcal{P}(\mathbf{D}') \implies \mathbf{D} = \mathbf{D}'$.

(*inductive step*): We want to show that if $depth(\mathbf{D}) = n$, then $\mathbf{D} = \mathbf{D}'$.

Let us suppose that r in $\mathbf{D}$ (r' in $\mathbf{D}'$, respectively) has k (x, respectively) solitary subfeatures $f_1, \ldots, f_k$ ($f'_1, \ldots, f'_x$, respectively) and t (y, respectively) groups $\{G_1, \ldots, G_t\}$ ($\{G'_1, \ldots, G'_y\}$, respecively). According to Definition 4.7,

$$\mathcal{P}(\mathbf{D}) = \{\lceil r, m_1^{c_1}, \ldots, m_k^{c_k}, g_1, \ldots, g_t\rceil, \text{ where}$$

$\forall 1 \leq i \leq k, \forall 1 \leq j \leq t:$

$m_i \in \mathcal{P}(\mathbf{D}^{f_i}), c_i \in \mathcal{C}(f_i), g_j \in \mathcal{P}(\mathbf{D}, G_j)\}$ (C)

$$\mathcal{P}(\mathbf{D}') = \{\lceil r', m_1^{c_1}, \ldots, m_x^{c_x}, g_1, \ldots, g_y\rceil, \text{ where}$$

$\forall 1 \leq i \leq x, \forall 1 \leq j \leq y :$

$m_i \in \mathcal{P}(\mathbf{D'}^{f'_i}), c_i \in \mathcal{C}'(f'_i), g_j \in \mathcal{P}(\mathbf{D'}, G'_j)\}$. (C')

Consider an arbitrary hierarchical product $m = \lceil r, m_1^{c_1}, \ldots, m_k^{c_k}, g_1, \ldots, g_t \rceil$, where m_i $(1 \leq i \leq k)$ and g_j $(1 \leq j \leq t)$ satisfy the conditions in (C). Since for any $1 \leq i \leq k$ and $1 \leq j \leq t$: $rank(m_i) \in \mathcal{H}(F) \wedge rank(g_j) \in \mathcal{H}(F)$, r is the only urelement in the domain of m, i.e., $m \in \mathcal{P}(\mathbf{D'}) : dom(m') \cap F' = \{r'\}$. Likewise, for any $m \in \mathcal{P}(\mathbf{D'}) : dom(m') \cap F' = \{r'\}$. Since $\mathcal{P}(\mathbf{D}) = \mathcal{P}(\mathbf{D'})$, $r = r'$.

For any CFD, the domain of any hierarchical product of an induced diagram by a node f includes f with multiplicity 1 and its all other elements are multisets. Also, the domain of a grouped hierarchical product of a CFD is a set of multisets, i.e., it does not include any urelement. This implies the following statements:

(i) $k = x$ and $t = y$,

(ii) $\forall 1 \leq i \leq k, \exists 1 \leq j \leq k : \mathcal{P}(\mathbf{D}^{f_i}) = \mathcal{P}(\mathbf{D'}^{f'_j}) \wedge \mathcal{C}(f_i) = \mathcal{C}'(f'_j)$,

(iii) $\forall 1 \leq i \leq t, \exists 1 \leq i' \leq t : \mathcal{P}(\mathbf{D}, G_i) = \mathcal{P}(\mathbf{D'}, G'_j)$.

(ii) implies that the sets of r's solitary subfeatures in both $\mathbf{D}$ and $\mathbf{D'}$ are the same. Without loss of generality, suppose that $\forall 1 \leq i \leq k : f_i = f'_i$. Since $\forall 1 \leq i \leq k : \mathcal{P}(\mathbf{D}^{f_i}) = \mathcal{P}(\mathbf{D'}^{f'_i})$ and $depth(\mathbf{D}^{f_i}) < n$, due to the hypothesis, $\mathbf{D}^{f_i} = \mathbf{D'}^{f_i}$.

(iii) implies that the set of groups of r in $\mathbf{D}$ and $\mathbf{D'}$ are the same. Without loss of generality, we suppose that $\forall 1 \leq i \leq t : G_i = G'_i$. Consider an $1 \leq i \leq t$ and let $G_i = G'_i = \{q_1, \ldots, q_z\}$. According to Definition 4.8,

$\mathcal{P}(\mathbf{D}, G_i) = \{\lceil m_1^{c_1 \times l_1}, \ldots, m_z^{c_z \times l_z} \rceil : \forall 1 \leq j \leq z.\ m_j \in \mathcal{P}(\mathbf{D}^{g_j}), c_j \in \mathcal{C}(g_j), l_j \in \{0, 1\},\ \text{and } l_1 + \ldots + l_z \in \mathcal{C}(G_i)\}$.

$\mathcal{P}(\mathbf{D}', G_i) = \{\lceil m_1^{c_1 \times l_1}, \ldots, m_z^{c_z \times l_z} \rceil : \forall 1 \leq j \leq z.\ m_j \in \mathcal{P}(\mathbf{D}'^{g_j}), c_j \in \mathcal{C}'(g_j), l_i \in \{0, 1\},\ \text{and } l_1 + \ldots + l_z \in \mathcal{C}'(G_i)\}$.

$\mathcal{P}(\mathbf{D}, G_i) = \mathcal{P}(\mathbf{D}', G_i)$ implies that $\forall 1 \leq j \leq z : \mathcal{P}(\mathbf{D}^{g_j}) = \mathcal{P}(\mathbf{D}'^{g_j})$ and $\mathcal{C}(g_j) = \mathcal{C}'(g_j), \mathcal{C}(G_i) = \mathcal{C}'(G_i)$. Since $depth(\mathbf{D}^{g_i}) < n$, due to the hypothesis, $\mathbf{D}^{g_i} = \mathbf{D}'^{g_i}$.

According to above, since r in both $\mathbf{D}$ and $\mathbf{D}'$ have the same set of solitary subfeatures and groups whose corresponding induced diagrams are the same with the same multiplicities, $\mathbf{D} = \mathbf{D}'$. □

Theorem 4.2. For any CFD $\mathbf{D} \in \mathcal{D}(F)$, the function $\mathsf{flat}_F\big|_{\mathcal{P}(\mathbf{D})}$, i.e., the restriction of flat_F to the subdomain $\mathcal{P}(\mathbf{D})$, provides a bijection between $\mathcal{P}(\mathbf{D})$ and $\mathcal{P}^{\text{flat}}(\mathbf{D})$.

Proof. We use an inductive reasoning based on the depth of CFDs to show this.

(*base case*): The statement obviously holds for any CFD with singleton tree, i.e., a CFD with depth 1.

(*hypothesis*): Assume that the statement holds for any CFD $\mathbf{D}$ with $1 \leq depth(\mathbf{D}) < d$ for some $d \in \mathbb{N}$.

(*inductive step*): We show that $\mathsf{flat}_F\big|_{\mathcal{P}(\mathbf{D})}$ provides a bijection from $\mathcal{P}(\mathbf{D})$ to $\mathcal{P}^{\text{flat}}(\mathbf{D})$ for any CFD $\mathbf{D}$ with $depth(\mathbf{D}) = d$.

Let $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$ be a CFD with $depth(\mathbf{D}) = d$ and $\mathcal{S} \subseteq F_{-r}$ denote the set of its solitary features. Suppose that $\mathcal{S} \cap r_{\downarrow} = \{f_1, \ldots, f_i\}$ (solitary subfeatures of the root) and $\mathcal{G} \cap 2^{r_{\downarrow}} = \{G_1, \ldots, G_j\}$ (groups subelements of the root) for some $i, j \in \mathbb{N}$.

Consider a hierarchical product $h \in \mathcal{P}(\mathbf{D})$. According to Definition 4.7, h is a multiset $\lceil r, h_1^{c_1}, \ldots, h_i^{c_i}, g_1, \ldots, g_j \rceil$, where $h_k \in \mathcal{P}(\mathbf{D}^{f_k})$, $c_k \in \mathcal{C}(f_k)$ $(1 \leq k \leq i)$, and $g_t \in \mathcal{P}(\mathbf{D}, G_t)$ $(1 \leq t \leq j)$.

According to Definition 4.10, $\mathsf{flat}_F(h) = \lceil r \rceil \uplus \biguplus_{1 \leq k \leq i} (\mathsf{flat}_F(h_k))^{c_k} \uplus \biguplus_{1 \leq t \leq j} \mathsf{flat}_F(g_t)$.

Since $h_k \in \mathcal{P}(\mathbf{D}^{f_k})$ and $depth(\mathbf{D}^{f_k}) < d$ for $1 \leq k \leq i$, due to the hypothesis, $\mathsf{flat}_F(h_k)$ is a flat product of the diagram induced by f_k, i.e., $\mathsf{flat}_F(h_k) \in \mathcal{P}^{\text{flat}}(\mathbf{D}^{f_k})$.

Let $G_t = \{g_1, \ldots, g_l\}$ for $1 \leq t \leq j$.

According to Definition 4.8, $g_t = \lceil m_1^{c'_1 \times t_1}, \ldots, m_l^{c'_n \times t_l} \rceil$, where $m_k \in \mathcal{P}(\mathbf{D}^{g_k})$, $c'_k \in \mathcal{C}(g_k)$, $t_k \in \{0, 1\}$, and $t_1 + \ldots + t_l \in \mathcal{C}(G_t)$ $(1 \leq k \leq l)$. According to Definition 4.10, $\mathsf{flat}_F(g_t) = \mathsf{flat}_F(m_1)^{c'_1 \times t_1} \uplus \ldots \uplus \mathsf{flat}_F(m_l)^{c'_n \times t_n}$. Since $depth(\mathbf{D}^{g_k}) < d$ for any $1 \leq k \leq l$, due the hypothesis, $\mathsf{flat}_F(m_k) \in \mathcal{P}^{\text{flat}}(\mathbf{D}^{g_k})$. This implies that, according to Definition 4.4, $\mathsf{flat}_F(g_t) \in \mathcal{P}^{\text{flat}}(\mathbf{D}, G_t)$.

According to above, $\mathsf{flat}_F(h) = \lceil r \rceil \uplus \biguplus_{1 \leq k \leq i} m_k^{c_k} \uplus \biguplus_{1 \leq t \leq j} n_t$, where $m_k = \mathsf{flat}_F(h_k)$ $(1 \leq k \leq i)$ is a flat product of the diagram induced by f_k, i.e., $m_k \in \mathcal{P}^{\text{flat}}(\mathbf{D}^{f_k})$ and n_t $(1 \leq t \leq j)$ is a flat grouped product of G_t, i.e., $n_t = \mathsf{flat}_F(g_t) \in \mathcal{P}^{\text{flat}}(\mathbf{D}, G_t)$. Due to Lemma 4.1, $\mathsf{flat}_F(h) \in \mathcal{P}^{\text{flat}}(\mathbf{D})$. Therefore, $\mathsf{flat}_F\big|_{\mathcal{P}(\mathbf{D})}$ maps each hierarchical product of $\mathbf{D}$ to a flat product of $\mathbf{D}$. In the following, we show that $\mathsf{flat}_F\big|_{\mathcal{P}(\mathbf{D})}$ is an injective function.

Consider two different hierarchical products $h, h' \in \mathcal{P}(\mathbf{D})$ such that $\mathsf{flat}_F(h) = \mathsf{flat}_F(h')$. According to Definition 4.10 and Definition 4.7,

$\mathsf{flat}_F(h) = \lceil r \rceil \uplus \biguplus_{1 \leq k \leq i} \mathsf{flat}_F(h_k)^{c_k} \uplus \biguplus_{1 \leq t \leq j} \mathsf{flat}_F(g_t)$, and

$\mathsf{flat}_F(h') = \lceil r \rceil \uplus \biguplus_{1 \leq k \leq i} \mathsf{flat}_F(h'_k)^{c'_k} \uplus \biguplus_{1 \leq t \leq j} \mathsf{flat}_F(g'_t)$, where

$\forall 1 \leq k \leq i, \forall 1 \leq t \leq j$: $h_k, h'_k \in \mathcal{P}(\mathbf{D}^{f_k})$, $c_k, c'_k \in \mathcal{C}(f_k)$, and $g'_t, g_t \in \mathcal{P}(\mathbf{D}, G_t)$.

Note that for any two distinct subelements (solitary and/or group subelements) of the root, their hierarchical and flat products are built on disjoint subsets of features (a CFD is a special tree of features). Therefore, $\mathsf{flat}_F(h) = \mathsf{flat}_F(h')$ implies that for any $1 \leq k \leq i, 1 \leq t \leq j$: $\mathsf{flat}_F(h_k) = \mathsf{flat}_F(h'_k)$, $c_k = c'_k$, and $\mathsf{flat}_F(g_t) = \mathsf{flat}_F(g'_t)$. Due to hypothesis, this implies that for any $1 \leq k \leq i, 1 \leq t \leq j$: $h_k = h'_k$, $c_k = c'_k$, and $g_t = g'_t$. Therefore, $h = h'$, which implies that the restriction of the function $\mathsf{flat}_F\big|_{\mathcal{P}(\mathbf{D})}$ is an injective function from $\mathcal{P}(\mathbf{D})$ to $\mathcal{P}^{\text{flat}}(\mathbf{D})$.

According to Definition 4.7 and Lemma 4.1, $|\mathcal{P}(\mathbf{D})| = |\mathcal{P}^{\text{flat}}(\mathbf{D})|$ (recursive definitions of hierarchical and flat products of $\mathbf{D}$) for any CFD $\mathbf{D}$, i.e., the cardinalities of the sets of flat and hierarchical products of $\mathbf{D}$ are the same. Therefore, the restriction of the flattening function to the hierarchical semantics of $\mathbf{D}$ is a surjective function, as it is injective and the cardinalities of the domain and codomain are the same.

According to above, $\mathsf{flat}_F\big|_{\mathcal{P}(\mathbf{D})} : \mathcal{P}(\mathbf{D}) \to \mathcal{P}^{\text{flat}}(\mathbf{D})$ is a bijection. □

Theorem 4.3. Any hierarchical product of a given CFD over a set of features F is a tree-like multiset over F.

Proof of Theorem 4.3. We use an inductive reasoning based on the depth of CFDs to deal with this theorem.

(*base case*): Obviously, the statement holds for any CFD $\mathbf{D}$ with $\mathit{depth}(\mathbf{D}) = 1$.

(*hypothesis*): We assume that for any CFD $\mathbf{D}$ with $1 \leq \mathit{depth}(\mathbf{D}) < n$, the statement holds.

(*inductive step*): Let $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$ be a CFD and $\mathit{depth}(\mathbf{D}) = n$. We show that any hierarchical product $m \in \mathcal{P}(\mathbf{D})$ is a tree-like multiset.

Consider the CFD $\mathbf{D}' \stackrel{\text{def}}{=} \mathbf{D}_{-n}$ (upper diagram Induced by depth n). Due to Definition 4.7, for any hierarchical product $m \in \mathcal{P}(\mathbf{D})$, there exists $m' \in \mathcal{P}(\mathbf{D}')$ such that m is obtained by replacing any feature $f \in \{f \in F : depth(f) = n-1\}$ in m' with an $x \in \mathcal{P}(\mathbf{D}^f)$.

Due to the hypothesis, any $x \in \mathcal{P}(\mathbf{D}^f)$ is a tree-like multiset. Thus, according to Definition 4.11, m would be a tree-like multiset. □

Theorem 4.4. For any tree-like multiset t, there is a CFD $\mathbf{D}$ such that $t \in \mathcal{P}(\mathbf{D})$.

Proof of Theorem 4.4. Let t be a tree-like multiset. We want to show that there is a CFD whose hierarchical semantics includes t.

Let $T = (N, r, \_^{\uparrow})$ and $\mathcal{G}$ denote the tree and groups associated with t, respectively: $N = N_t$, $r = r_t$, $\_^{\uparrow} = \_^{\uparrow_t}$, and $\mathcal{G} = \mathcal{G}_t$.[4] We also define a function $\mathcal{C} : (N \setminus \{r\} \cup \mathcal{G}) \to 2^{\mathbb{N}}$ as follows. $\forall e \in (N \setminus \{r\} \cup \mathcal{G}) : \mathcal{C}(e) = \{\mathcal{C}_t(e)\}$, where $\mathcal{C}_t : (N \setminus \{r\}) \cup \mathcal{G}$ is defined in Definition 4.18.

The tuple $\mathbf{D} = (T, \mathcal{G}, \mathcal{C})$ would be a CFD except that there may be some singleton groups (note that singleton groups are not allowed in CFDs–see Definition 4.1(ii)). Let us call a CFD in which singleton groups are allowed a *CFD plus* (CFD$^+$). The semantics of CFD$^+$s can be defined via hierarchical semantics of CFDs. Note that the definition of hierarchical semantics for CFDs (Definition 4.7) can be directly used on CFD$^+$s. In this sense, the tuple $(T, \mathcal{G}, \mathcal{C})$ represents a singleton hierarchical semantics, as all multiplicities are singleton. It is easy to see that its singleton hierarchical product is t.

[4] See Definitions 4.16 and 4.17, respectively.

Thus, $\mathbf{D}$ is a CFD plus representing t as its single hierarchical product. We show that this tuple is a substructure of some CFDs. Indeed, to get a CFD whose hierarchical semantics includes the single hierarchical product of the tuple, we just need to add one (or more than one) feature(s) to singleton groups. We formally show how this works in the following.

Let $\mathcal{G}^1 = \{G \in \mathcal{G} : |G| = 1\}$ and N' be a set of symbols (features) with $N' \cap N = \varnothing$ and $|N'| = |\mathcal{G}^1|$. Consider a bijection $l : \mathcal{G}^1 \to N'$. We build a CFD $\mathbf{D'} = (N' \cup N, r, \_^{\uparrow'}, \mathcal{G}', \mathcal{C}')$ as follows.

$$\forall n \in N \cup N' :\uparrow' (n) = \begin{cases} l^{-1}(n)^{\uparrow} & \text{if } n \in N' \\ n^{\uparrow} & \text{otherwise} \end{cases}$$

$$\mathcal{G}' = (\mathcal{G} \setminus \mathcal{G}^1) \cup \{G \cup \{l(G)\} : G \in \mathcal{G}^1\}$$

$\mathcal{C}' : ((N' \setminus \{r\}) \cup \mathcal{G}') \to 2^{\mathbb{N}}$ is defined as follows.

$$\forall e \in (N' \setminus \{r\}) \cup \mathcal{G}') : \mathcal{C}'(e) = \begin{cases} \mathcal{C}(e) & \text{if } e \in N \vee e \in \mathcal{G} \setminus \mathcal{G}^1 \\ \{1\} & \text{otherwise} \end{cases}$$

Clearly, $\mathbf{D'}$ is a CFD and $\mathbf{D}$ is a substructure of $\mathbf{D'}$. Thus, $t \in \mathcal{P}(\mathbf{D'})$. The theorem is proven! $\square$

Theorem 4.5. Consider an enumerable set of tree-like multisets $U = \{t_i : i \in I\} \subset \mathcal{TH}(A)$ over a set A, where I enumerates its elements. Let $T_i = (N_i, r_i, \_^{\uparrow_i})$ and $\mathcal{G}_i$ ($\forall i \in I$) denote the t_i's associated tree and groups, respectively (see Definitions 4.16 and 4.17, respectively). The tree-like multisets in U are mergeable iff:

(i) $\forall i, j \in I : T_i, T_j$ are mergeable.

(ii) $\forall i, j \in I, \forall n \in N_i \cap N_j : (\exists G \in \mathcal{G}_i : n \in G) \implies (\exists G \in \mathcal{G}_j : n \in G)$.

Proof of Theorem 4.5. We prove the statement for $I = \{1, 2\}$. The proof can be easily extended to any enumerating set $I \subseteq \mathbb{N}$. Let $U = \{t_1, t_2\}$. We need to show that the following statements hold:

(1) t_1 and t_2 are mergeable $\implies$ (i) and (ii) hold.

(2) (i) and (ii) hold $\implies$ t_1 and t_2 are mergeable.

Proof of (1):

Suppose that t_1 and t_2 are megeable. According to Definition 4.19, there exists a CFD $\mathbf{D} = (T, \mathcal{G}, \mathcal{C})$ with $T = (F, r, \_^{\uparrow})$ such that $t_1, t_2 \in \mathcal{P}(\mathbf{D})$. This implies the following statements:

(S-1) T_1 and T_2 are subtrees of T such that their roots are equal to the root of T. Formally, $N_1 \cup N_2 \subseteq F$, $r_1 = r_2 = r$, and $\forall n \in N_1 \cap N_2 \setminus \{r\} : n^{\uparrow_1} = n^{\uparrow_2} = n^{\uparrow}$. Thus (i) holds.

(S-2) For any urelement $a \in A$, if its corresponding induced tree in t_1 (i.e., ${t_1}^a$) or t_2 (i.e., ${t_2}^a$) is a grouped tree-like multiset, then a must be a grouped feature in $\mathbf{D}$. Formally, $\forall a \in A : (\exists G \in \mathcal{G}_1 : a \in G) \vee (\exists G \in \mathcal{G}_2 : a \in G) \implies (\exists G \in \mathcal{G} : a \in G)$. Clearly, this implies that $\forall n \in N_1 \cap N_2 : (\exists G_1 \in \mathcal{G}_1 : n \in G_1) \implies (\exists G_2 \in \mathcal{G}_2 : n \in G_2)$. Therefore, (ii) holds.

Due to (S-1) and (S-2), (1) is proven.

Proof of (2):

Suppose that (i) and (ii) hold. We show that t_1 and t_2 are mergeable. To this end, we construct a CFD whose hierarchical semantics includes both t_1 and t_2.

Let $N' = N_1 \cup N_2$, $r' = r_1$ (note that $r_1 = r_2$), and $\_^{\uparrow'} : N' \setminus \{r'\} \to N'$ defined as

$\_^{\uparrow'} = \_^{\uparrow_1} \cup \_^{\uparrow_2}$. Note that $(N', r', \_^{\uparrow'}) = \{T_1, T_2\}^{\text{merge}}$ (see Definition 4.21).

Let $\mathcal{G}' = \mathcal{G}_{\sqcap} \bigcup \mathcal{G}_{\sqcup}$, where

$$\mathcal{G}_{\sqcap} = \{G_1 \cup G_2 : (G_1 \in \mathcal{G}_1) \wedge (G_2 \in \mathcal{G}_2) \wedge (G_1 \cap G_2 \neq \varnothing)\},$$

$$\mathcal{G}_{\sqcup} = \{G \in \mathcal{G}_1 \cup \mathcal{G}_2 : (\forall G' \in \mathcal{G}_{\sqcap} : G' \cap G = \varnothing)\},$$

To merge two CFDs, we also need to merge their groups. According to Definition 5.3, two different groups in a CFD must share no elements. Thus, we have to merge all groups in $\mathcal{G}_1$ and $\mathcal{G}_2$ that share some elements. $\mathcal{G}_{\sqcap}$ does so. Any other groups in either $\mathcal{G}_1$ and $\mathcal{G}_2$ must have to be considered as a group in the merged CFD. Such groups are obtained via $\mathcal{G}_{\sqcup}$. There may be some singleton elements in $\mathcal{G}$. Note that, according to Definition 5.3, singleton groups are not allowed in a CFD. Below, we address this problem.

Let $\mathcal{G}'' = \{G \in \mathcal{G} : |G| = 1\}$ and N'' be a set of symbols (features) with $N'' \cap N' = \varnothing$ and $|N''| = |\mathcal{G}''|$. Consider a bijection $l : \mathcal{G}'' \to N''$.

We define a tuple $\mathbf{D} = (N, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$, where:

$N = N'' \cup N'$,

$r = r'$,

$\mathcal{G} = (\mathcal{G}' \setminus \mathcal{G}'') \cup \{G \cup \{l(G)\} : G \in \mathcal{G}''\}$,

$\_^{\uparrow} : N \setminus \{r\} \to N$, defined as:

$$\forall n \in N : n^{\uparrow} = \begin{cases} l^{-1}(n)^{\uparrow'} & \text{if } n \in N'' \\ n^{\uparrow'} & \text{otherwise} \end{cases}$$

$$\forall e \in N \cup \mathcal{G} : \mathcal{C}(e) = \begin{cases} \{0\} \cup \mathcal{C}_1(e) & \text{if } (e \in N_1 \setminus N_2) \vee (e \in \mathcal{G}_\sqcup \cap \mathcal{G}_1) \\ \{0\} \cup \mathcal{C}_2(e) & \text{if } (e \in N_2 \setminus N_1) \vee (e \in \mathcal{G}_\sqcup \cap \mathcal{G}_2) \\ \mathcal{C}_1(e) \cup \mathcal{C}_2(e) & \text{if } (e \in N_1 \cap N_2) \vee (e \in \mathcal{G}_\sqcap) \\ \{1\} & \text{otherwise} \end{cases}$$

where $\mathcal{C}_1$ and $\mathcal{C}_2$ denote the multiplicities associated with t_1 and t_2, respectively (see Definition 4.18).

It is easy to see that the tuple $\mathbf{D} = (N, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$ is a CFD. It is obvious that t_1 and t_2 are two hierarchical products of $\mathbf{D}$. Thus, t_1 and t_2 are two mergeable tree-like multisets.

The proof is easily extendable to any enumerating set $I \subseteq \mathbb{N}$, as a set of tree-like multisets are mergeable iff each pairs of tree-like multisets are mergeable. □

Theorem 4.6. Consider an emeumrable set of tree-like multisets $U \subset \mathcal{TH}(A)$ over a set A of urelements.

(i) U is mergeable iff U° is.

(ii) U is mergeable implies that U° is finite.

Proof of Theorem 4.6. Let $U = \{t_i : i \in I\} \subset \mathcal{TH}(A)$, where $I \subseteq \mathbb{N}$ enumerates the elements of U. Let $T_i = (N_i, r_i, \_^{\uparrow_i})$, $\mathcal{G}_i$, and $\mathcal{C}_i$, for any $i \in I$, represent the t_i's associated tree, groups, and multiplicities , respectively – see Definitions 4.16, 4.17, and 4.18.

Proof of (i):

For any $i \in I$, let $T_i{}^\circ$ and $\mathcal{G}_i{}^\circ$ denote the tree and groups associated with $t_i{}^\circ$ (the relaxed multiset of t_i). According to Proposition 4.1, $\forall i \in I : \mathcal{G}_i{}^\circ = \mathcal{G}_i$ and $T_i{}^\circ = T_i$.

According to Theorem 4.5,

U is mergeable

$\iff$

– $\forall i, j \in I : T_i, T_j$ are mergeable.

– $\forall i, j \in I, \forall n \in N_i \cap N_j : (\exists G \in \mathcal{G}_i : n \in G) \implies (\exists G \in \mathcal{G}_j : n \in G)$.

$\iff$

– $\forall i, j \in I : T_i{}^\circ, T_j{}^\circ$ are mergeable.

– $\forall i, j \in I, \forall n \in N_i \cap N_j : (\exists G \in \mathcal{G}_i{}^\circ : n \in G) \implies (\exists G \in \mathcal{G}_j{}^\circ : n \in G)$.

$\iff$

According to Theorem 4.5, U° is meageable.

<u>*Proof of (ii)*</u>:

Suppose that the elements of U are megeable. Let $\mathbf{D} \in \mathbf{D}_{U^{\text{merge}}}$, i.e., $\mathbf{D}$ is a minimal representative CFD of U. Let $\mathbf{D} = (T, \mathcal{G}, \mathcal{C})$ with $T = (N, r, \_^{\uparrow})$.

According to (i), the elements of U° are megeable. Recall that the only difference between a tree-like multiset and its relaxed multiset is in their multiplicities, i.e., their trees and groups would be the same. We build a representative CFD $\mathbf{D}^\circ$ of U°, as follows:

$\mathbf{D}^\circ = (T, \mathcal{G}, \mathcal{C}^\circ)$ where

$$\forall e \in (N \setminus \{r\}) \cup \mathcal{G} : \mathcal{C}^\circ(e) = \begin{cases} \mathcal{C}(e) & \text{if } e \in \mathcal{G} \\ \{0, 1\} & \text{otherwise} \end{cases}$$

Clearly, $\mathbf{D}^\circ$ is a representative CFD of U°, since $\mathbf{D}$ is a minimal representative CFD of U and all feature multiplicities in $\mathbf{D}^\circ$ are $\{0,1\}$. Since there is no feature in $\mathbf{D}^\circ$ with an infinite multiplicity domain, $\mathcal{P}(\mathbf{D}^\circ)$ would be finite. Thus, U° is finite, since $U \subseteq \mathcal{P}(\mathbf{D}^\circ)$. $\square$

Theorem 4.7. Consider an emeumerable set of tree-like multisets $U \subset \mathcal{TH}(A)$ over a set A of urelements. U is completely mergeable iff

(i) U° is completely mergeable, and

(ii) $\forall t \in U^\circ, \forall a \in dom(\mathsf{flat}_A(t)), \forall c \in \mathcal{C}_U(a), \exists t' \in U : (t'^\circ = t) \wedge (\#_{t'}(t'^a) = c)$.

Proof. Suppose that U is completely mergeable, which means that there is some CFD $\mathbf{D} = (T, \mathcal{G}, \mathcal{C})$ with $(F, r, \_^\uparrow)$ representing U. We want to show that the statements (i) and (ii) hold.

We build a CFD $\mathbf{D}^\circ = (T, \mathcal{G}, \mathcal{C}^\circ)$, where $\mathcal{C}^\circ : (F \setminus \{r\}) \cup \mathcal{G} \to 2^{\mathbb{N}}$ is defined as follows:

$$\forall e \in (F \setminus \{r\}) \cup \mathcal{G} : \mathcal{C}^\circ(e) = \begin{cases} \mathcal{C}(e) & \text{if } e \in \mathcal{G} \\ \{0,1\} & \text{if } (e \notin \mathcal{G}) \wedge (0 \notin \mathcal{C}(e)) \\ \{1\} & \text{if } (e \notin \mathcal{G}) \wedge (0 \notin \mathcal{C}(e)) \end{cases}$$

It follows obviously that $\mathcal{P}(\mathbf{D}^\circ) = U^\circ$. Therefore, U° is completely mergeable.

Now, consider a tree-like multiset $t \in U^\circ$, $a \in dom(\mathsf{flat}_A(t))$, and $c \in \mathcal{C}_U(a)$. We want to show that there exists $t' \in U$ such that $t'^\circ = t$ and $\#_{t'}(t^a) = c$.

$t \in U^\circ$ implies that $t \in \mathcal{P}(\mathbf{D}^\circ)$. a is a feature in $\mathbf{D}^\circ$ involved in t and c is a valid multiplicity of the feature a in $\mathbf{D}$ (see the definition of overall multiplicities in Definition 4.23).

Since $t \in U^\circ$, there is some $t'' \in U$ such that $t''^\circ = t$. If $\#_{t''}(t''^a) = c$, then the statement (ii) is proven. Suppose that $\#_{t''}(t''^a) \neq c$. t'' is a hierarchical product of $\mathbf{D}$. Thus, for any $f \in dom(\mathsf{flat}_A(t))$ (including a), $\#_{t''}(t''^f) \in \mathcal{C}(f)$. According to Definition 4.7, replacing $\#_{t''}(t''^f)$ by any other valid multiplicity in the multiplicity domain of f would give us another valid hierarchical product of $\mathbf{D}$. Let us define t' by replacing $\#_{t''}(t''^a)$ by c. $t' \in \mathcal{P}(\mathbf{D})$ and thus $t \in U$. The statement (ii) is proven.

Proving that U is completely mergeable if the statements (i) and (ii) hold is very straightforward: Suppose that (i) and (ii) hold. Therefore, there exists a CFD $\mathbf{D}^\circ$ such that $\mathcal{P}(\mathbf{D}^\circ) = U^\circ$. Note that the multiplicity domain of any feature in $\mathbf{D}^\circ$ is either $\{0, 1\}$ or $\{1\}$. Now, we define a CFD $\mathbf{D}$ by replacing the multiplicity of any feature a in $\mathbf{D}^\circ$ by $\mathcal{C}_U(a)$ (overall multiplicity og a, see Definition 4.23). Clearly, according to (ii), $\mathcal{P}(\mathbf{D}) = U$. Therefore, U is completely mergeable. $\square$

Appendix C

Proofs of Chapter 5

We will need the following notations in the proofs.

For any regular expressions $\mathcal{R}$ (languages $\mathcal{L}$, respectively). $\Sigma(\mathcal{R})$ ($\Sigma(\mathcal{L})$, respectively) denotes the alphabet which $\mathcal{R}$ ($\mathcal{L}$, respectively) is built on.

We will also need the following definitions.

Definition C.1 (**Substitution of an Element).** Let F and F' be two finite sets and $m \in \mathcal{H}_1(F), m' \in \mathcal{H}_1(F')$. For a given $f \in F$, the *substitution* of f with m' in m is a multiset in $\mathcal{H}_1(F \cup F')$, denoted by $m[f \mapsto_{\mathrm{P}} m']$, specified as follows: each occurrence of f in m is substituted by m'. $\square$

As an example, let $F = \{f_1, f_2, f_3\}$, $F' = \{f'_1, f'_2\}$, $m = [[{f_1}^2, f_2], [f_1, {f_2}^3]]$, and $m' = [[{f'_1}^3]]$. Then, $m[f_1 \mapsto_{\mathrm{P}} m'] = [[{f'_1}^6, f_2], [{f'_1}^3, {f_2}^3]]$.

Let $F'' = \{f_1, \ldots, f_k\} \subseteq F$ and *sub* be a function, which maps each element f_i of F'' to a multiset m_i. We usually write $m[f \mapsto_{\mathrm{P}} sub(f) : \forall f \in F'']$ to mean $m[f_1 \mapsto_{\mathrm{P}} m_1] \ldots [f_k \mapsto_{\mathrm{P}} m_k]$.

Definition C.2 (**Substitution of a Letter with a Language).** Let $\mathcal{L}$ and $\mathcal{L}'$

be two languages and $\sigma \in \Sigma(\mathcal{L})$. The *Substitution of* σ in $\mathcal{L}$ with $\mathcal{L}'$ is a language, denoted by $\mathcal{L}[\sigma \mapsto_{\mathrm{L}} \mathcal{L}']$, equal to: $\mathcal{L}[\sigma \mapsto_{\mathrm{L}} \mathcal{L}'] = \{w \in \mathcal{L} : \sigma \notin w\} \bigcup \{ww'w'' : (w\sigma w'' \in \mathcal{L}) \wedge (w' \in \mathcal{L}')\}$. $\square$

For an example, consider $\Sigma = \{a, b, c\}$ and the two languages $\mathcal{L} = \{a^n b^n : n \in \mathbb{N}\}$ and $\mathcal{L}' = \{c^n : n \in \mathbb{N}\}$: $\mathcal{L}[b \mapsto_{\mathrm{L}} \mathcal{L}'] = \{a^n c^m : m \text{ is divisible by } n\}$.

Let $\Sigma' = \{\sigma_1, \ldots, \sigma_k\}$ be a subset of Σ and *sub* be a function, which maps each letter σ_i of Σ' to a language $\mathcal{L}_i$. We write $\mathcal{L}[\sigma \mapsto_{\mathrm{L}} sub(\sigma) : \forall\sigma \in \Sigma']$ to mean $\mathcal{L}[\sigma_1 \mapsto_{\mathrm{L}} \mathcal{L}_1] \ldots [\sigma_k \mapsto_{\mathrm{L}} \mathcal{L}_k]$.

Definition C.3 (**Substitution of a Letter with an Expression).** Let $\mathcal{R}$ and $\mathcal{R}'$ be two regular expressions and $\sigma \in \Sigma(\mathcal{R})$. The *Substitution of* σ *with* $\mathcal{R}'$ is a regular expression denoted by $\mathcal{R}[\sigma \mapsto_{\mathrm{E}} \mathcal{R}']$ and specified as follows: any instance of σ in $\mathcal{R}$ is replaced by $\mathcal{R}'$. $\square$

For example, consider an alphabet $\Sigma = \{a, b, c\}$ and the two regular expressions $\mathcal{R} = (a + bc)^*$ and $\mathcal{R}' = c^*$: $\mathcal{R}[a \mapsto_{\mathrm{E}} \mathcal{R}'] = (c^* + bc)^*$[1]

Let $\Sigma' = \{\sigma_1, \ldots, \sigma_k\}$ be a subset of $\Sigma(\mathcal{R})$ and *sub* be a function, which maps each letter σ_i of Σ' to a regular expression $\mathcal{R}_i$. We write $\mathcal{R}[\sigma \mapsto_{\mathrm{E}} sub(\sigma) : \forall\sigma \in \Sigma']$ to mean $\mathcal{R}[\sigma_1 \mapsto_{\mathrm{E}} \mathcal{R}_1] \ldots [\sigma_k \mapsto_{\mathrm{E}} \mathcal{R}_k]$.

Definition C.4 (**Substitution of a Leaf Node with a CFD).** Consider two CFDs $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$ and $\mathbf{D}' = (F', f, \_^{\uparrow'}, \mathcal{G}', \mathcal{C}')$ and let $f \in lev(\mathbf{D})$ (f is a leaf node in $\mathbf{D}$) such that $F \cap F' = \{f\}$. The *substitution of* f *with* $\mathbf{D}'$ is a CFD, denoted by $\mathbf{D}[f \mapsto_{\mathrm{D}} \mathbf{D}']$, equal to $(F \cup F',\ r,\ \_^{\uparrow} \cup \_^{\uparrow'},\ \mathcal{G} \cup \mathcal{G}',\ \mathcal{C} \cup \mathcal{C}')$. $\square$

[1]This regular expression would be semantically equal to $(c + b)^*$.

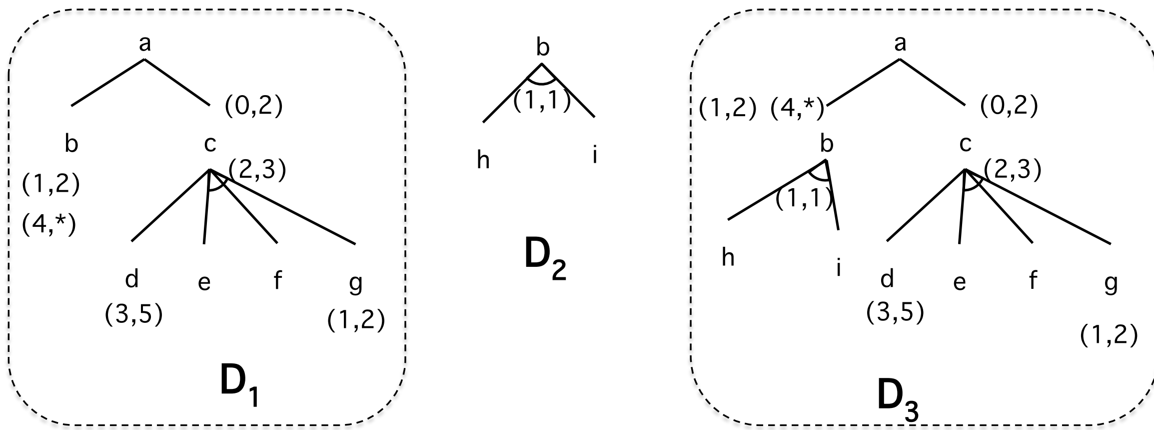


Figure C.2: Substitution of a Leaf Node with a CFD: An example

As an example, consider the CFDs $\mathbf{D}_1$ and $\mathbf{D}_2$ in Figure C.2. The substitution of the leaf node b in $\mathbf{D}_1$ with $\mathbf{D}_2$ would be $\mathbf{D}_3$ represented in Figure C.2.

Let $\mathbf{D}$ be a CFD over a set of features F and $F' = \{f_1, \ldots, f_k\}$ a subset of its set of leaf nodes, i.e., $F' \subseteq lev(F)$. Consider CFDs $\mathbf{D}_1, \ldots, \mathbf{D}_k$ over set of features $F_1, \ldots, F_k$, respectively, such that for all i the root of $\mathbf{D}_i$ is f_i and for all distinct indices $1 \leq i, j \leq k$: $F \cap F_i = \{f_i\}$ and $F_i \cap F_j = \varnothing$. Let sub be a function, which maps each element f_i in F' to the CFD $\mathbf{D}_i$. For succinctness, we usually write $\mathbf{D}[f \mapsto_{\mathrm{D}} sub(f) : \forall f \in F']$ to mean $\mathbf{D}[f_1 \mapsto_{\mathrm{D}} \mathbf{D}_1] \ldots [f_k \mapsto_{\mathrm{D}} \mathbf{D}_k]$.

Definition C.5 (**CFDs Cut by Nodes**)**.** Let $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$ be a CFD and $f \in F$. The CFD cut by f is a CFD $\mathbf{D}^{-f} = (F', r, \_^{\uparrow}|_{F'}, \mathcal{G}', \mathcal{C}\big|_{\mathcal{G}' \uplus N'})$, where $F' = F \backslash f_{\downarrow\downarrow}$ and $\mathcal{G}' = \mathcal{G} \cap 2^{F'}$, i.e., its tree is the tree of $\mathbf{D}$ in which the tree under f is excluded; all other components are inherited from $\mathbf{D}$. $\square$

As an example, $\mathbf{D}_2$ in Figure C.3 is the CFD cut by c in $\mathbf{D}_1$ in Figure C.3. The following propositions and lemmas come in handy in the proofs of the main theorems. Proposition C.1 follows obviously.

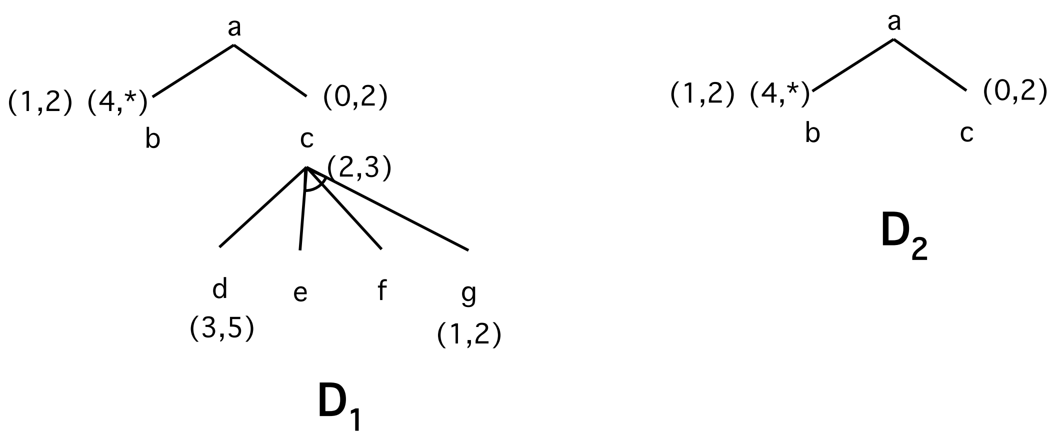


Figure C.3: Cutting of CFD by nodes: an example

Proposition C.1. Let $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$ be a CFD and $f \in F$. Then, the following statements hold:

(i) $\mathbf{D} = \mathbf{D}^{-f}[f \mapsto_{\mathrm{D}} \mathbf{D}^{f}]$.

(ii) $\mathcal{P}^{\text{flat}}(\mathbf{D}) = \mathcal{P}^{\text{flat}}(\mathbf{D}^{-f})[f \mapsto_{\mathrm{P}} \mathcal{P}^{\text{flat}}(\mathbf{D}^{f})]$.

(iii) $\mathcal{R}^{\mathsf{CRE}}(\mathbf{D}) = \mathcal{R}^{\mathsf{CRE}}(\mathbf{D}^{-f})[f \mapsto_{\mathrm{E}} \mathcal{R}^{\mathsf{CRE}}(\mathbf{D}^{f})]$.

(iv) $\mathcal{R}^{\mathsf{ORE}}(\mathbf{D}) = \mathcal{R}^{\mathsf{ORE}}(\mathbf{D}^{-f})[f \mapsto_{\mathrm{E}} \mathcal{R}^{\mathsf{ORE}}(\mathbf{D}^{f})]$. □

Lemma C.1. Let $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$ be a CFD and $1 \leq d \leq \mathit{depth}(\mathbf{D})$. Then, $\mathbf{D} = \mathbf{D}_{-d}[f \mapsto_{\mathrm{D}} \mathbf{D}^{f} : \forall f \in F']$, where $F' = \{f \in F : \mathit{depth}(f) = d\}$, i.e., the nodes with depth d. [2] □

Proof of Lemma C.1. Let $F' = \{f_1, \ldots, f_i\}$. We define a set of CFDs $\{\mathbf{D}_0, \mathbf{D}_1, \ldots, \mathbf{D}_i\}$ recursively as follows: $\mathbf{D}_0 = \mathbf{D}$ and for any $1 \leq j \leq i$: $\mathbf{D}_j = (\mathbf{D}_{j-1})^{-f_j}$. Note that $\mathbf{D}_{-d} = \mathbf{D}_i$. Due to Proposition C.1(i), $\mathbf{D}_j = \mathbf{D}_{j-1}[f_j \mapsto_{\mathrm{D}} \mathbf{D}^{f_j}]$, for any $1 \leq j \leq i$. Therefore, $\mathbf{D} = \mathbf{D}_0 = \mathbf{D}_{-d}[f_1 \mapsto_{\mathrm{D}} \mathbf{D}_1] \ldots [f_i \mapsto_{\mathrm{D}} \mathbf{D}_i]$, which is equal to $\mathbf{D}_{-d}[f \mapsto_{\mathrm{D}} \mathbf{D}^{f} : \forall f \in F']$. □

[2] $\mathbf{D}_{-d}$ denotes the upper diagram induced by depth d in $\mathbf{D}$ – see Definition B.1.

Lemma C.2. *Let* $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$ *be a CFD and* $0 \leq d \leq depth(\mathbf{D})$*. Then,* $\mathcal{P}^{\text{flat}}(\mathbf{D}) = \mathcal{P}^{\text{flat}}(\mathbf{D}_{-d})[f \mapsto_{\text{P}} \mathcal{P}^{\text{flat}}(\mathbf{D}^{f}) : \forall f \in F']$*, where* $F' = \{f \in F : depth(f) = d\}$*, i.e., the nodes with depth* d*.*

Proof of Lemma C.2. Let $F' = \{f_1, \ldots, f_i\}$. We define a set of CFDs $\{\mathbf{D}_0, \mathbf{D}_1, \ldots, \mathbf{D}_i\}$ recursively as follows: $\mathbf{D}_0 = \mathbf{D}$ and for any $1 \leq j \leq i$: $\mathbf{D}_j = (\mathbf{D}_{j-1})^{-f_j}$. Note that $\mathbf{D}_{-d} = \mathbf{D}_i$. Due to Proposition C.1(ii), $\mathcal{P}^{\text{flat}}(\mathbf{D}_{j-1}) = \mathcal{P}^{\text{flat}}(\mathbf{D}_j)[f_j \mapsto_{\text{P}} \mathcal{P}^{\text{flat}}(\mathbf{D}^{f_j})]$. Therefore, $\mathcal{P}^{\text{flat}}(\mathbf{D}) = \mathcal{P}^{\text{flat}}(\mathbf{D}_0) = \mathcal{P}^{\text{flat}}(\mathbf{D}_{-d})[f_1 \mapsto_{\text{P}} \mathcal{P}^{\text{flat}}(\mathbf{D}_1)] \ldots [f_i \mapsto_{\text{P}} \mathcal{P}^{\text{flat}}(\mathbf{D}_i)]$, which is equal to $\mathcal{P}^{\text{flat}}(\mathbf{D}_{-d})[f \mapsto_{\text{P}} \mathcal{P}^{\text{flat}}(\mathbf{D}^{f}) : \forall f \in F']$. $\square$

Lemma C.3. For any CFD $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$ and $0 \leq d \leq depth(\mathbf{D})$: $\mathcal{R}^{\mathsf{CRE}}(\mathbf{D}) = \mathcal{R}^{\mathsf{CRE}}(\mathbf{D}_{-d})[f \mapsto_{\text{E}} \mathcal{R}^{\mathsf{CRE}}(\mathbf{D}^{f}) : \forall f \in F']$, where $F' = \{f \in F : depth(f) = d\}$, i.e., the nodes with depth d.

Proof of Lemma C.3. Let $F' = \{f_1, \ldots, f_i\}$. We define a set of CFDs $\{\mathbf{D}_0, \mathbf{D}_1, \ldots, \mathbf{D}_i\}$, recursively, as follows: $\mathbf{D}_0 = \mathbf{D}$ and for any $1 \leq j \leq i$: $\mathbf{D}_j = (\mathbf{D}_{j-1})^{-f_j}$. Note that $\mathbf{D}_{-d} = \mathbf{D}_i$. Due to Proposition C.1(iii), $\mathcal{R}^{\mathsf{CRE}}(\mathbf{D}_{j-1}) = \mathcal{R}^{\mathsf{CRE}}(\mathbf{D}_j)[f_j \mapsto_{\text{E}} \mathcal{R}^{\mathsf{CRE}}(\mathbf{D}^{f_j})]$. Therefore, $\mathcal{R}^{\mathsf{CRE}}(\mathbf{D}) = \mathcal{R}^{\mathsf{CRE}}(\mathbf{D}_0) = \mathcal{R}^{\mathsf{CRE}}(\mathbf{D}_{-d})[f_1 \mapsto_{\text{E}} \mathcal{R}^{\mathsf{CRE}}(\mathbf{D}_1)] \ldots [f_i \mapsto_{\text{E}} \mathcal{R}^{\mathsf{CRE}}(\mathbf{D}_i)]$, which is equal to $\mathcal{R}^{\mathsf{CRE}}(\mathbf{D}_{-d})[f \mapsto_{\text{E}} \mathcal{R}^{\mathsf{CRE}}(\mathbf{D}^{f}) : \forall f \in F']$. $\square$

The proofs for the main theorems of Chapter 5 are given in the following.

Theorem 5.1. For a given CFD $\mathbf{D}$, $\text{Par}(\mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{D}))) = \mathcal{P}^{\text{flat}}(\mathbf{D})$.

Proof of Theorem 5.1. We use an inductive reasoning based on the depth of CFDs to prove this theorem.

(*basic step*): Obviously, the statement $\text{Par}(\mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{D}))) = \mathcal{P}^{\text{flat}}(\mathbf{D})$ holds for any CFD $\mathbf{D}$ with $depth(\mathbf{D}) = 1$.

(*hypothesis*): Assume that $\mathrm{Par}(\mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{D}))) = \mathcal{P}^{\text{flat}}(\mathbf{D})$ for any CFD $\mathbf{D}$ with $1 \leq \mathit{depth}(\mathbf{D}) \leq d$ for some $d \in \mathbb{N}$.

(*inductive step*): We want to prove that for any CFD $\mathbf{D}$ with $\mathit{depth}(\mathbf{D}) = d + 1$ the statement holds, i.e., $\mathrm{Par}(\mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{D}))) = \mathcal{P}^{\text{flat}}(\mathbf{D})$.

Let $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$ be a CFD with $\mathit{depth}(\mathbf{D}) = d + 1$.

Due to Lemma C.1, $\mathbf{D} = \mathbf{D}_{-d}[f \mapsto_{\mathrm{D}} \mathbf{D}^f : \forall f \in F']$, where $F' = \{f \in F : \mathit{depth}(f) = d\}$, i.e., the nodes with depth d.

Due to Lemma C.3, $\mathcal{R}^{\mathsf{CRE}}(\mathbf{D}) = \mathcal{R}^{\mathsf{CRE}}(\mathbf{D}_{-d})[f \mapsto_{\mathrm{E}} \mathcal{R}^{\mathsf{CRE}}(\mathbf{D}^f) : \forall f \in F']$.

Therefore, $\mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{D})) = \mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{D}_{-d}))[f \mapsto_{\mathrm{L}} \mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{D}^f)) : \forall f \in F']$.

Obviously, $\mathrm{Par}(\mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{D}))) = \mathrm{Par}(\mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{D}_{-d})))[f \mapsto_{\mathrm{P}} \mathrm{Par}(\mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{D}^f))) : \forall f \in F']$.

Since $\mathit{depth}(\mathbf{D}_{-d}) = d$, according to the hypothesis, $\mathrm{Par}(\mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{D}_{-d}))) = \mathcal{P}^{\text{flat}}(\mathbf{D}_{-d})$.

Since for any $f \in F'$: $\mathit{depth}(\mathbf{D}^f) < d$, according to the hypothesis, $\mathrm{Par}(\mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{D}^f))) = \mathcal{P}^{\text{flat}}(\mathbf{D}^f)$.

Therefore, $\mathrm{Par}(\mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{D}))) = \mathcal{P}^{\text{flat}}(\mathbf{D}_{-d})[f \mapsto_{\mathrm{P}} \mathcal{P}^{\text{flat}}(\mathbf{D}^f) : \forall f \in F']$.

Due to Lemma C.2, since $\mathcal{P}^{\text{flat}}(\mathbf{D}) = \mathcal{P}^{\text{flat}}(\mathbf{D}_{-d})[f \mapsto_{\mathrm{P}} \mathcal{P}^{\text{flat}}(\mathbf{D}^f) : \forall f \in F']$, $\mathrm{Par}(\mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{D}))) = \mathcal{P}^{\text{flat}}(\mathbf{D})$. The theorem is proven. $\square$

Theorem 5.2. For a given CFD $\mathbf{D}$, $\mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{D}))$ preserves $\mathbf{D}$'s hierarchy.

Proof of Theorem 5.2. Consider a CFD $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$. We need to prove the following statements for any $f, f' \in F$:

(1) $(f' \in f_{\downarrow}) \Rightarrow \big(\forall w \in \mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{D})) : (f' \in U_w) \Rightarrow (f \sqsubseteq_w f')\big)$.

(2) $\big(\forall w \in \mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{D})) : (f' \in U_w) \Rightarrow (f \sqsubseteq_w f')\big) \Rightarrow (f' \in f_{\downarrow\downarrow})$.

Note that (1) implies $(f' \in f_{\downarrow\downarrow}) \Rightarrow \big(\forall w \in \mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{D})) : (f' \in w) \Rightarrow (f \sqsubseteq_w f')\big)$.

Proof of (1):

Since $\mathbf{D}$ is an unlabelled tree of features, for any $i \leq \mathit{depth}(\mathbf{D})$, $(\mathrm{shr}^{\mathsf{CRE}})^i(\mathbf{D})$ is a CRD where the labels of two different nodes are two different regular expressions built over two disjoint alphabets. Let us call such CRDs *disjoint labeled CRDs* (DL-CRD).

It is obvious that for any DL-CRD $\mathbf{RD}$ and $i \leq \mathit{depth}(\mathbf{RD})$, $(\mathrm{shr}^{\mathsf{CRE}})^i(\mathbf{RD})$ is also a DL-CRD. To prove (1), we prove a more general statement stated as follows: "Consider a DL-CRD $\mathbf{RD} = (LT_{re}, \mathcal{G}, \mathcal{C})$ with $LT_{re} = (N, r, \_^{\uparrow}, \Sigma, l_{re})$. Let $n', n'' \in N$ with $l_{re}(n') = R'$ and $l_{re}(n'') = R''$ such that $n'' \in n'^{\uparrow}$. Then,
$\forall w \in \mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{RD})), \forall w'' \in \mathcal{L}(R'') : (w'' \leq_{\mathrm{seq}} w) \Rightarrow [\exists w' \in \mathcal{L}(R') : w'.w'' \leq_{\mathrm{seq}} w]$."

Let $w \in \mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{RD}))$ and $w'' \in \mathcal{L}(R'')$ such that $w'' \leq_{\mathrm{seq}} w$. We need to show that $\exists w' \in \mathcal{L}(R') : w'.w'' \leq_{\mathrm{seq}} w$.

Since $\mathbf{RD}$ and $(\mathrm{shr}^{\mathsf{CRE}})^i(\mathbf{RD})$ for any $i \leq \mathit{depth}(\mathbf{RD})$ are DL-CRDs, $\mathcal{R}^{\mathsf{CRE}}(\mathbf{RD}) = R.(R'.(R''.R^{(3)} + R^{(4)}) + R^{(5)})$ for some regular expresisons $R, R^{(3)}, R^{(4)}, R^{(5)}$ (note the definition $\mathrm{dre}^{\mathsf{CRE}}$ in Definition 5.11 and Definition 5.10) such that the regular expressions $R, R', R', R'', R^{(3)}, R^{(4)}$, $R^{(5)}$ are built over disjoint alphabets. Since $w'' \leq_{\mathrm{seq}} w$, $w \in \mathcal{L}(R.R'.R''.R^{(3)})$. The statement is proven, since R' precedes R'' in $R.R'.R''.R^{(3)}$, i.e., $\exists w' \in \mathcal{L}(R') : w'.w'' \leq_{\mathrm{seq}} w$.

Proof of (2):

We show $\neg(f' \in f_{\downarrow\downarrow}) \Rightarrow \neg\big(\forall w \in \mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{D})) : (f' \in U_w) \Rightarrow (f \sqsubseteq_w f')\big)$, which is equivalent to (2).

Suppose that $f' \notin f_{\downarrow\downarrow}$. Let k be the minimum of $\mathit{depth}(f)$ and $\mathit{depth}(f')$. Let $d = \mathit{depth}(\mathbf{D})$ and $(\mathrm{shr}^{\mathsf{CRE}})^{d-k}(\mathbf{D}) = \mathbf{RD}'$.

There are two leaves ℓ and ℓ' in $\mathbf{RD}'$ with labels R and R' in $\mathbf{RD}'$ such that $f \in \Sigma(R)$ and $f' \in \Sigma(R')$. Since $\mathbf{RD}'$ is an DL-CRD, $\Sigma(R') \cap \Sigma(R) = \varnothing$.

Note that by applying the shrinking step on $\mathbf{D}$ $d-k$ times (where $k = min(depth(f), depth(f')))$ the parents of both ℓ and ℓ' would be the same and equal to the least common ancestor of f and f', i.e., ℓ and ℓ are siblings in $\mathbf{RD}'$. Let $p = \ell^{\uparrow} = \ell'^{\uparrow}$. There are the following choices for ℓ and ℓ':

(i) Both are solitary nodes.

(ii) One of them, say ℓ, is in a group and another one, ℓ', is a solitary node.

(iii) Both are in a same group G.

(iv) One of them, say ℓ, is in a group G and another, ℓ', is in another group G'.

Let $\mathbf{RD}'_1 = \mathrm{gle}^{\mathsf{CRE}} \circ \mathrm{mel}^{\mathsf{CRE}}(\mathbf{RD}')$ (applying the first and second stages of shrinking steps on $\mathbf{RD}'$). There are two leaves ℓ_1 and ℓ'_1 with labels R_1 and R'_1 in $\mathbf{RD}'_1$ such that $f \in \Sigma(R_1)$ and $f' \in \Sigma(R'_1)$. Note that $\Sigma(R_1) \cap \Sigma(R'_1) = \varnothing$ and all leaves in $\mathbf{RD}'_1$ are solitary with multiplicities $(1, 1)$.

Now let us apply the function $\mathrm{dre}^{\mathsf{CRE}}$ on $\mathbf{RD}'_1$ to get $(\mathrm{shr}^{\mathsf{CRE}})^{d-k+1}(\mathbf{D})$. Since the function $\mathrm{dre}^{\mathsf{CRE}}$ considers all valid permutations of the p's child nodes, there is a leaf node ℓ'' in $(\mathrm{shr}^{\mathsf{CRE}})^{d-k+1}(\mathbf{D})$ labeled with a regular expression $R_{\ell''}$ in the form of $R_{\ell''} = R^{(2)} + R_1.R'_1.R^{(3)} + R'_1.R_1.R^{(4)}$.

Since $\Sigma(R_1) \cap \Sigma(R'_1) = \varnothing$, there are two words $w_1, w_2 \in \mathcal{L}(R_{\ell''})$ such that $f \sqsubseteq_{w_1} f'$ and $f' \sqsubseteq_{w_2} f$. Thus, keeping doing the shrinking steps until getting $\mathrm{shr}^{\mathsf{CRE}}(\mathbf{D})$, there would be a word $w \in \mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{D}))$ such that $f' \in w$ but $\neg(f \sqsubseteq_w f')$. The statement (2) is proven. $\square$

Theorem 5.3. For any given osCFD $\mathbf{OD}$:

(i) $\mathcal{L}(\mathcal{R}^{\mathsf{ORE}}(\mathbf{OD})) \subseteq \mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{OD}^{\text{cfd}}))$.

(ii) $\text{Par}(\mathcal{L}(\mathcal{R}^{\mathsf{ORE}}(\mathbf{OD}))) = \text{Par}(\mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{OD}^{\text{cfd}})))$.

Proof of Theorem 5.3. CRE and ORE differ in two stages, stages 2 and 3:

(1) In calculating the expressions corresponding to a group of leaves in their second stages (see ORE-EGL and CRE-EGL).

(2) In calculating the expressions in stage 3 in which an expression is computed for a given node all of whose children are leaves (see ORE-DR and CRE-DR).

The difference is that we consider all valid permutations of the corresponding elements (in (1): elements of a group; in (2): the children of a node) in CRE while, in ORE, we consider a subset of these permutations conforming the sibling ordering.

According to above, any word in $\mathcal{L}(\mathcal{R}^{\mathsf{ORE}}(\mathbf{OD}))$ belongs also to $\mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{OD}^{\text{cfd}}))$, which implies that $\mathcal{L}(\mathcal{R}^{\mathsf{ORE}}(\mathbf{OD})) \subseteq \mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{OD}^{\text{cfd}}))$. The statement (i) is proven.

(i) implies that $\text{Par}(\mathcal{L}(\mathcal{R}^{\mathsf{ORE}}(\mathbf{OD}))) \subseteq \text{Par}(\mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{OD}^{\text{cfd}})))$. Therefore, for any word $w \in \mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{OD}^{\text{cfd}}))$, there is a word $w' \in \mathcal{L}(\mathcal{R}^{\mathsf{ORE}}(\mathbf{OD}))$ such that their Parikh image are the same, i.e., $\text{Par}(w) = \text{Par}(w')$. To get w' from w, consider a permutation of w satisfying the sibling ordering. Thus, (ii) is proven, i.e., $\text{Par}(\mathcal{L}(\mathcal{R}^{\mathsf{ORE}}(\mathbf{OD}))) = \text{Par}(\mathcal{L}(\mathcal{R}^{\mathsf{CRE}}(\mathbf{OD}^{\text{cfd}})))$. □

Theorem 5.4. $\mathcal{L}(\text{cc}_1)$, $\mathcal{L}(\text{cc}_2)$, and $\mathcal{L}(\text{cc}_3)$ are regular, $\mathcal{L}(\text{cc}_4)$ is context-free, and $\mathcal{L}(\text{cc}_5)$ is context-sensitive.

Proof of Theorem 5.4. A language is regular iff it can be expressed by some regular expressions, regular grammars, or finite state automata (FSA). Let $F = \{f_1, \ldots, f_n\}$ for some $n \geq 3$.

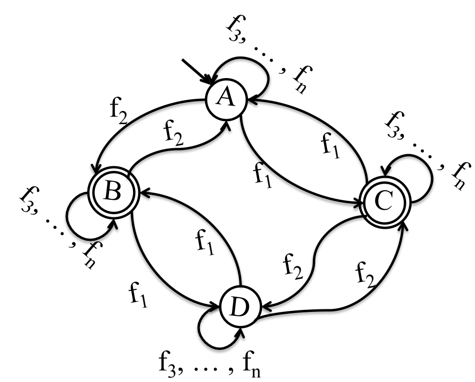
$\mathcal{L}(\mathrm{cc}_1)$ can be expressed by the following regular expression, where $r = (f_1 + \ldots + f_n)^*$:

$$f_2^* + rf_1rf_2r + rf_2rf_1r.$$

$\mathcal{L}(\mathrm{cc}_2)$ can be expressed by the following regular expression:

$$(f_2 + \ldots + f_n)^* + (f_1 + f_3 + \ldots + f_n)^* + (f_3 + \ldots + f_n)^*.$$

The following FSA accepts $\mathcal{L}(\mathrm{cc}_3)$. The initial state is identified by an incoming unlabelled arrow not originating at any state. The final states are drawn with double circles.



$\mathcal{L}(\mathrm{cc}_4)$ and $\mathcal{L}(\mathrm{cc}_5)$ are very well-known context-free and context-sensitive languages, respectively [Lin11]. □

Theorem 5.5. Given a recursive CFM, the operations Valid Product, Common Ancestors, and Least Common Ancestor are decidable.

Proof of Theorem 5.5. Let **M** be a CFM over a set of features F.

Recall that Valid Configuration operation is reduced to a *membership problem* in the context of formal languages (see page **??**). Since membership problem is decidable in the class of recursive languages, the problem would be decidable in the class of recursive CFMs.

Recall that we reduced the Common Ancestors problem to the following problem: Given a set of features F', a feature $f \in F$ is a common ancestor of the features in F' iff $\forall w \in \mathcal{L}^{\mathsf{ORE}}(\mathbf{M}), \forall f' \in F' : f \sqsubseteq_w f'$. This problem is decidable in no classes of CFMs when $\mathcal{L}^{\mathsf{ORE}}(\mathbf{M})$ is infinite. However, we can reduce it to the following smart problem:

The problem deals with only the underlying CFD. Let $\mathbf{D}$ denote the CFD of $\mathbf{M}$. Since the problem has nothing to do with multiplicities, it is sufficient to work with $\mathbf{D}^\circ$, i.e., the relaxed CFD of $\mathbf{D}$. Thus, the above problem is reduced to "f is a common ancestor of the features in F' iff $\forall w \in \mathcal{L}^{\mathsf{ORE}}(\mathbf{D}^\circ), \forall f' \in F' : f \sqsubseteq_w f'$". Since $\mathcal{L}^{\mathsf{ORE}}(\mathbf{D}^\circ)$ and F' are finite, the common ancestors problem is decidable in all classes of CFMs. We can follow the same way to show that the Least Common Ancestor problem in decidable in all classes of CFMs. $\square$

Theorem 5.6. Given a context-free FM $\mathbf{M}$, the operations Partial Configuration, Core Features, Valid feature Multiplicity, Void Feature Model, and Dead Feature are decidable. However, none of them is decidable in the class of context-sensitive CFMs.

Proof of Theorem 5.6. Let F be the set of features of $\mathbf{M}$.

Partial Configuration: Let $\mathcal{L}$ denote the set of all prefixes of the words of $\mathcal{L}^{\mathsf{CRE}}(\mathbf{M})$. $\mathcal{L}$ is a context-free language. To prove this, we take the grammar of $\mathcal{L}^{\mathsf{CRE}}(\mathbf{M})$ in Chomsky Normal Form and for every production $A \to BC$, add productions $A_\varepsilon \to$

BC_ε and $A_\varepsilon \to B_\varepsilon$. Also, for every production $A \to f$ (for some terminals f), we consider the production $A_\varepsilon \to f$. Finally, we change the starting variable S to S_ε and add the production $S_\varepsilon \to \varepsilon$. The context-free grammar generated in this way represents the language $\mathcal{L}$. Thus, $\mathcal{L}$ is decidable. The set of partial configurations would be equal to the bag interpretation of $\mathcal{L}$. Thus, the Partial Configuration problem is decidable.

Core Features: Consider a subset $C \subseteq F$. We want to determine whether C is included in all products or not. Let $C = \{f_1, \ldots, f_n\}$ for some $n \in \mathbb{N}$, $\mathcal{L} = \mathcal{L}(F^* f_1^* F^* \ldots f_n^* F^*)$. The problem is reduced to determining whether $\mathcal{L}^{\mathsf{CRE}}(\mathbf{M}) \subseteq \mathcal{L}$ or not. In other words, the problem is reduced to determining whether $\mathcal{L}^{\mathsf{CRE}}(\mathbf{M}) \cap \mathcal{L}^c = \varnothing$ or not ($\mathcal{L}^c$ denotes the complement of $\mathcal{L}$). Note that $\mathcal{L}$ is a regular language and so is $\mathcal{L}^c$. Hence, the language $\mathcal{L}^{\mathsf{CRE}}(\mathbf{M}) \cap \mathcal{L}^c$ is context-free. Since the emptiness problem in the class of context-free languages is decidable, the original problem, i.e., determining if C is included in all products, is decidable. Since the number of subsets of F is finite, the problem of finding the set of Core Features is also decidable.

Valid feature Multiplicity: Recall that the Valid feature Multiplicity problem is reduced to an *emptiness* problem: Given a feature f and $n \in \mathbb{N}$, n can be a valid multiplicity of f iff $\mathcal{L} \cap \mathcal{L}^{\mathsf{CRE}}(\mathbf{M}) \neq \varnothing$, where $\mathcal{L} = \mathcal{L}((F \setminus \{f\})^* f^n (F \setminus \{f\})^*)$. Since the emptiness problem is decidable in the class of context-free languages, the Valid feature Multiplicity problem would be decidable in the class of context-free CFMs.

Void Feature Model: Since the emptiness problem is decidable in the class of context-free languages, the Void Feature Model problem would be decidable.

Dead Feature: Let $\mathcal{L} = \mathcal{L}(F^* f F^*)$. The problem of determining whether the feature f is a dead feature of $\mathbf{M}$ or not is, indeed, to determine whether $\mathcal{L} \cap \mathcal{L}^{\mathsf{CRE}}(\mathbf{M}) =$

$\varnothing$ or not. Note that $\mathcal{L}$ is regular. Hence, $\mathcal{L} \cap \mathcal{L}^{\mathsf{CRE}}(\mathbf{M})$ is context-free. Since the emptiness problem of context-free languages is decidable, the Dead Feature problem is decidable too.

In the class of Context-Sensitive CFMs: Note that the above analysis operations are not decidable in other classes of CFMs. Recall that all the above operations are reduced to emptiness problem in the formal language theory. Since emptiness problem is not decidable in the class of non-context-free context-sensitive languages [Dav94], the above operations would not be decidable in the class of context-sensitive CFMs. □

Theorem 5.7. Given two CFMs $\mathbf{M}_1$ and $\mathbf{M}_2$, the following statements hold:

(i) If both are regular, then the (Dynamic) Refactoring problem between them is decidable.

(ii) If $\mathbf{M}_1$ and $\mathbf{M}_2$ are regular and context-free, respectively, then the (Dynamic) Refactoring problem is decidable iff $\mathbf{M}_1$ is bounded regular.

Proof of Theorem 5.7.

(i) The equality problem between regular languages is decidable [Lin11].

(ii) Hopcroft in [Hop69] showed that for two given context-free languages $\mathcal{L}_1$ and $\mathcal{L}_2$, if one of them, say $\mathcal{L}_1$, is a bounded regular language, then the equality problem between these two languages is decidable. □

Theorem 5.8. Given two CFMs $\mathbf{M}_1$ and $\mathbf{M}_2$, the following statements hold:

(i) If both are regular, the (Dynamic) Specialization problem between them is decidable.

(ii) If $\mathbf{M}_1$ and $\mathbf{M}_2$ are regular and context-free, respectively, then the problem "is $\mathbf{M}_2$ a (dynamic) specialization of $\mathbf{M}_1$?" is decidable.

Proof of Theorem 5.8.

(i) The inclusion problem in the class of regular languages is decidable [MS72]. Thus, the Specialization problem is decidable in the class of regular languages.

(ii) The problem "$\mathbf{M}_2$ is a specialization of $\mathbf{M}_1$" is reducible to the problem $\mathcal{L}^{\mathsf{CRE}}(\mathbf{M}_2) \subseteq \mathcal{L}^{\mathsf{CRE}}(\mathbf{M}_1)$. In other words, it is equivalent to the problem of determining whether $\mathcal{L}^{\mathsf{CRE}}(\mathbf{M}_2) \cap \mathcal{L}^{\mathsf{CRE}}(\mathbf{M}_1)^c = \varnothing$ or not. Since the class of regular languages is closed under complement, $\mathcal{L}^{\mathsf{CRE}}(\mathbf{M}_1)^c$ is regular. Thus, $\mathcal{L}^{\mathsf{CRE}}(\mathbf{M}_2) \cap \mathcal{L}^{\mathsf{CRE}}(\mathbf{M}_1)^c$ is context-free. Since the emptiness problem in the class of context-free languages is decidable, the Specialization problem in this case would be decidable. $\square$

Bibliography

[ACC$^+$11] Parosh Aziz Abdulla, Yu-Fang Chen, Lorenzo Clemente, Lukáš Holík, Chih-Duo Hong, Richard Mayr, and Tomáš Vojnar. Advanced ramsey-based büchi automata inclusion testing. In *CONCUR 2011–Concurrency Theory*, pages 187–202. Springer, 2011.

[ACC$^+$13] Mathieu Acher, Benoit Combemale, Philippe Collet, Olivier Barais, Philippe Lahire, and Robert B France. Composing your compositions of variability models. In *Model-Driven Engineering Languages and Systems*, pages 352–369. Springer, 2013.

[ACLF10a] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert France. Composing feature models. In *Software Language Engineering*, pages 62–81. Springer, 2010.

[ACLF10b] 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.

[AP$^+$04] Marcus Alanen, Ivan Porres, et al. *A relation between context-free grammars and meta object facility metamodels.* Turku Centre for Computer Science, 2004.

[ASB$^+$08] V. Alves, C. Schwanninger, L. Barbosa, A. Rashid, P. Sawyer, P. Rayson, C. Pohl, and A. Rummler. An exploratory study of information retrieval techniques in domain analysis. In *SPLC'08.*, 2008.

[BAS15] Sandy Beidu, Joanne M Atlee, and Pourya Shaker. Incremental and commutative composition of state-machine models of features. In *Modeling in Software Engineering (MiSE), 2015 IEEE/ACM 7th International Workshop on*, pages 13–18. IEEE, 2015.

[Bat05] Don Batory. *Feature models, grammars, and propositional formulas.* Springer, 2005.

[BBRC06] Don Batory, David Benavides, and Antonio Ruiz-Cortes. Automated analysis of feature models: challenges ahead. *Communications of the ACM*, 49(12):45–47, 2006.

[BF97] Alexander Bolotov and Michael Fisher. A resolution method for ctl branching-time temporal logic. In *Temporal Representation and Reasoning, 1997.(TIME'97), Proceedings., Fourth International Workshop on*, pages 20–27. IEEE, 1997.

[BLR+15] Thorsten Berger, Daniela Lettner, Julia Rubin, Paul Grünbacher, Adeline Silva, Martin Becker, Marsha Chechik, and Krzysztof Czarnecki. What is a feature?: a qualitative study of features in industrial software product lines. In *Proceedings of the 19th International Conference on Software Product Line*, pages 16–25. ACM, 2015.

[BO92] Don Batory and Sean O'malley. The design and implementation of hierarchical software systems with reusable components. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 1(4):355–398, 1992.

[Bos00] Jan Bosch. *Design and Use of Software Architectures: Adopting and Evolving a Product-line Approach*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[Bos01] Jan Bosch. Software product lines: organizational alternatives. In *Proceedings of the 23rd International Conference on Software Engineering*, pages 91–100. IEEE Computer Society, 2001.

[BP13] Filippo Bonchi and Damien Pous. Checking nfa equivalence with bisimulations up to congruence. *ACM SIGPLAN Notices*, 48(1):457–468, 2013.

[BSRC10] 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.

[BSTRC06a] David Benavides, Sergio Segura, Pablo Trinidad, and Antonio Ruiz-Cortés. A first step towards a framework for the automated analysis of feature models. *Proc. Managing Variability for Software Product Lines: Working With Variability Mechanisms*, pages 39–47, 2006.

[BSTRC06b] David Benavides, Sergio Segura, Pablo Trinidad, and Antonio Ruiz-Cortés. Using java csp solvers in the automated analyses of feature models. In *Generative and Transformational Techniques in Software Engineering*, pages 399–408. Springer, 2006.

[BTC05] David Benavides, Pablo Trinidad, and Antonio Ruiz Cortés. Using constraint programming to reason on feature models. In *SEKE*, pages 677–682, 2005.

[BW13] Alexander Bergmayr and Manuel Wimmer. Generating metamodels from grammars by chaining translational and by-example techniques. In *MDEBE@ MoDELS*, pages 22–31, 2013.

[CBUE02] Krzysztof Czarnecki, Thomas Bednasch, Peter Unger, and Ulrich Eisenecker. Generative programming for embedded software: An industrial experience report. In *Generative Programming and Component Engineering*, pages 156–172. Springer, 2002.

[CCH+12] A. Classen, M. Cordy, P. Heymans, A. Legay, and P. Schobbens. Model checking software product lines with SNIP. *In STTT*, 14(5):589–612, 2012.

[CCH+14] A. Classen, M. Cordy, P. Heymans, A. Legay, and P. Schobbens. Formal semantics, modular specification, and symbolic verification of product-line behaviour. *Sci. Comput. Program.*, 80:416–439, 2014.

[CCS+13] A. Classen, M. Cordy, P. Schobbens, P. Heymans, A. Legay, and J. Raskin. Featured transition systems: Foundations for verifying variability-intensive systems and their application to LTL model checking. *In IEEE Transactions on Software Engineering*, 39(8):1069–1089, 2013.

[CDMM11] Gary Chastek, Patrick Donohoe, John D McGregor, and Dirk Muthig. Engineering a production method for a software product line. In *Software Product Line Conference (SPLC), 2011 15th International*, pages 277–286. IEEE, 2011.

[CE00] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative programming: methods, tools, and applications*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[CHE04] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Staged configuration using feature models. In *Software Product Lines*, pages 266–283. Springer, 2004.

[CHE05a] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 10(1):7–29, 2005.

[CHE05b] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Staged configuration through specialization and multilevel configuration of feature models. *Software Process: Improvement and Practice*, 10(2):143–169, 2005.

[CHL+14] M. Cordy, P. Heymans, A. Legay, P. Schobbens, B. Dawagne, and M. Leucker. Counterexample guided abstraction refinement of product-line behavioural models. In *FSE*, pages 190–201, 2014.

[CHS+10] A. Classen, P. Heymans, P. Schobbens, A. Legay, and J. Raskin. Model checking lots of systems: efficient verification of temporal properties in software product lines. In *ICSE*, pages 335–344, 2010.

[CHSL11] A. Classen, P. Heymans, P. Schobbens, and A. Legay. Symbolic model checking of software product lines. In *ICSE*, pages 321–330. ACM, 2011.

[CK05] Krzysztof Czarnecki and Chang Hwan Peter Kim. Cardinality-based feature modeling and constraints: A progress report. In *International Workshop on Software Factories*, pages 16–20, 2005.

[Coo03] S Barry Cooper. *Computability theory*. CRC Press, 2003.

[Cop02] B Jack Copeland. The church-turing thesis. *Stanford encyclopedia of philosophy*, 2002.

[CW07] 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.

[Dav94] Martin Davis. *Computability, complexity, and languages: fundamentals of theoretical computer science*. Academic Press, 1994.

[DG08] Volker Diekert and Paul Gastin. First-order definable languages. *Logic and Automata: History and Perspectives*, 2:261, 2008.

[dJV02] Merijn de Jonge and Joost Visser. Grammars as feature diagrams. In *ICSR7 Workshop on Generative Programming*, pages 23–24. Citeseer, 2002.

[DWDMR08] Martin De Wulf, Laurent Doyen, Nicolas Maquet, and Jean-François Raskin. Alaska. In *Automated Technology for Verification and Analysis*, pages 240–245. Springer, 2008.

[EBB05] Magnus Eriksson, Jürgen Börstler, and Kjell Borg. The pluss approach: domain modeling with features, use cases and use case realizations. In *SPLC 2005*, SPLC'05, pages 33–44, Berlin, Heidelberg, 2005. Springer-Verlag.

[EH88] E Allen Emerson and Joseph Y Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *Computer and System Science*, 30:1–24, 1988.

[FKC12] Rick Flores, Charles Krueger, and Paul Clements. Mega-scale product line engineering at general motors. In *Proceedings of the 16th International Software Product Line Conference-Volume 1*, pages 259–268. ACM, 2012.

[FL10] Oliver Friedmann and Martin Lange. A solver for modal fixpoint logics. *Electronic Notes in Theoretical Computer Science*, 262:99–111, 2010.

[FZ06] Shaofeng Fan and Naixiao Zhang. Feature model based on description logics. In *Knowledge-Based Intelligent Information and Engineering Systems*, pages 1144–1151. Springer, 2006.

[GFd98] Martin L Griss, John Favaro, and Massimo d'Alessandro. Integrating feature modeling with the rseb. In *Software Reuse, 1998. Proceedings. Fifth International Conference on*, pages 76–85. IEEE, 1998.

[Gin66] Seymour Ginsburg. *The Mathematical Theory of Context Free Languages.[Mit Fig.]*. McGraw-Hill Book Company, 1966.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[GLS08] A. Gruler, M. Leucker, and K. Scheidemann. Modeling and model checking software product lines. In *Formal Methods for Open Object-Based Distributed Systems*, pages 113–131. Springer, 2008.

[GMB06] Rohit Gheyi, Tiago Massoni, and Paulo Borba. A theory for feature models in alloy. In *First alloy workshop*, pages 71–80, 2006.

[GNS+15] Graeme Gange, Jorge A Navas, Peter Schachte, Harald Søndergaard, and Peter J Stuckey. A tool for intersecting context-free grammars and its applications. In *NASA Formal Methods*, pages 422–428. Springer, 2015.

[GP93] Vineet Gupta and Vaughan R. Pratt. Gages accept concurrent behavior. In *FOCS*, pages 62–71, 1993.

[HCH09] Arnaud Hubaux, Andreas Classen, and Patrick Heymans. Formal modelling of feature configuration workflows. In *Proceedings of the 13th International Software Product Line Conference*, pages 221–230. Carnegie Mellon University, 2009.

[Hil09] Sandra Hilber. Finite automata tool, http://cl-informatik.uibk.ac.at/software/fat/. 2009.

[HKM11a] Peter Höfner, Ridha Khédri, and Bernhard Möller. An algebra of product families. *Software and System Modeling*, 10(2):161–182, 2011.

[HKM11b] Peter Höfner, Ridha Khédri, and Bernhard Möller. Supplementing product families with behaviour. *Int. J. Software and Informatics*, 5(1-2):245–266, 2011.

[HM11] T. Hildebrandt and R. Mukkamala. Declarative event-based workflow as distributed dynamic condition response graphs. *arXiv preprint arXiv:1110.4161*, 2011.

[Hop69] John E. Hopcroft. On the equivalence and containment problems for context-free languages. *Mathematical systems theory*, 3(2):119–124, 1969.

[Hop07] John E Hopcroft. *Introduction to automata theory, languages, and computation*. Pearson Addison Wesley, 2007.

[Jaa02] Ari Jaaksi. Developing mobile browsers in a product line. *IEEE software*, (4):73–80, 2002.

[Jen07] Paul Jensen. Experiences with product line development of multi-discipline analysis software at overwatch textron systems. In *Software Product Line Conference, 2007. SPLC 2007. 11th International*, pages 35–43. IEEE, 2007.

[JGJ97] Ivar Jacobson, Martin Griss, and Patrik Jonsson. *Software Reuse: Architecture, Process and Organization for Business Success*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1997.

[JRvdL00] Mehdi Jazayeri, Alexander Ran, and Frank van der Linden. *Software Architecture for Product Families: Principles and Practice*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[KCB08] Charles W Krueger, Dale Churchett, and Ross Buhrdorf. Homeaway's transition to software product line practice: Engineering and business results in 60 days. In *Software Product Line Conference, 2008. SPLC'08. 12th International*, pages 297–306. IEEE, 2008.

[KCH+90] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, DTIC Document, 1990.

[KKL+98] Kyo C Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin, and Moonhang Huh. Form a feature oriented reuse method with domain-specific reference architectures. *Annals of Software Engineering*, 5(1):143–168, 1998.

[KLD02] Kyo C Kang, Jaejoon Lee, and Patrick Donohoe. Feature-oriented product line engineering. *IEEE software*, (4):58–65, 2002.

[KN12] Bakhadyr Khoussainov and Anil Nerode. *Automata theory and its applications*, volume 21. Springer Science & Business Media, 2012.

[Koz97] Dexter Kozen. *Automata and computability*. Springer, 1997.

[KRT08] Alexander Koller, Michaela Regneri, and Stefan Thater. Regular tree grammars as a formalism for scope underspecification. In *ACL*, pages 218–226. Citeseer, 2008.

[Kun08] Andreas Kunert. Semi-automatic generation of metamodels and models from grammars and programs. *Electronic Notes in Theoretical Computer Science*, 211:111–119, 2008.

[LHLG+15] Roberto E Lopez-Herrejon, Lukas Linsbauer, José A Galindo, José A Parejo, David Benavides, Sergio Segura, and Alexander Egyed. An assessment of search-based techniques for reverse engineering feature models. *Journal of Systems and Software*, 103:353–369, 2015.

[Lin11] Peter Linz. *An introduction to formal languages and automata*. Jones & Bartlett Publishers, 2011.

[Loe14] Abraham Loeb. The habitable epoch of the early universe. *International Journal of Astrobiology*, 13(04):337–339, 2014.

[LS01] Martin Lange and Colin Stirling. Focus games for satisfiability and completeness of temporal logic. In *Logic in Computer Science, 2001. Proceedings. 16th Annual IEEE Symposium on*, pages 357–365. IEEE, 2001.

[LŠV12] Ondřej Lengál, Jiří Šimáček, and Tomáš Vojnar. Vata: A library for efficient manipulation of non-deterministic tree automata. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 79–94. Springer, 2012.

[LT12] M. Leucker and D. Thoma. A formal approach to software product families. In *ISoLA*, pages 131–145. Springer, 2012.

[Man02] Mike Mannion. Using first-order logic for product line model validation. In *Software Product Lines*, pages 176–187. Springer, 2002.

[Mar05] Will Marrero. Using bdds to decide ctl. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 222–236. Springer, 2005.

[MR02] Alessandro Maccari and Claudio Riva. Architectural evolution of legacy product families. In *Software Product-Family Engineering*, pages 64–69. Springer, 2002.

[MS72] Albert R Meyer and Larry J Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Switching and Automata Theory, 1972., IEEE Conference Record of 13th Annual Symposium on*, pages 125–129. IEEE, 1972.

[MSDLM11] Raúl Mazo, Camille Salinesi, Daniel Diaz, and Alberto Lora-Michiels. Transforming attribute and clone-enabled feature models into constraint programs over finite domains. In *6th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, 2011.

[MVH+04] Deborah L McGuinness, Frank Van Harmelen, et al. Owl web ontology language overview. *W3C recommendation*, 10(2004-03):10, 2004.

[MWC09] Marcilio Mendonca, Andrzej Wasowski, and Krzysztof Czarnecki. Sat-based analysis of feature models is easy. In *Proceedings of the 13th International Software Product Line Conference*, pages 231–240. Carnegie Mellon University, 2009.

[MWCC08] Marcilio Mendonca, Andrzej Wasowski, Krzysztof Czarnecki, and Donald Cowan. Efficient compilation techniques for large scale feature models. In *Proceedings of the 7th international conference on Generative PROGRAMMING and component engineering*, pages 13–22. ACM, 2008.

[NE08] N. Niu and S. Easterbrook. On-demand cluster analysis for product line functional requirements. In *SPLC*, 2008.

[NEB+11] Mahdi Noorian, Alireza Ensan, Ebrahim Bagheri, Harold Boley, and Yevgen Biletskiy. Feature model debugging based on description logic reasoning. In *DMS*, volume 11, pages 158–164, 2011.

[Par61] Rohit J Parikh. Language generating devices. *Quarterly Progress Report*, 60:199–212, 1961.

[Par66] Rohit J Parikh. On context-free languages. *Journal of the ACM (JACM)*, 13(4):570–581, 1966.

[PBVDL05] Klaus Pohl, Günter Böckle, and Frank Van Der Linden. *Software product line engineering: foundations, principles, and techniques*. Springer, 2005.

[PP95] G. Michele Pinna and Axel Poigné. On the nature of events: Another perspective in concurrency. *Theor. Comput. Sci.*, 138(2):425–454, 1995.

[Pra91] Vaughan R. Pratt. Event spaces and their linear logic. In *AMAST*, pages 3–25, 1991.

[QC11] Gerard Quilty and MO Cinneide. Experiences with software product line development in risk management software. In *Software Product Line Conference (SPLC), 2011 15th International*, pages 251–260. IEEE, 2011.

[QRD13] Clément Quinton, Daniel Romero, and Laurence Duchien. Cardinality-based feature models with constraints: a pragmatic approach. In *Proceedings of the 17th International Software Product Line Conference*, pages 162–166. ACM, 2013.

[RF06] Susan H Rodger and Thomas W Finley. *JFLAP: an interactive formal languages and automata package*. Jones & Bartlett Learning, 2006.

[SA14] Pourya Shaker and Joanne M Atlee. Behaviour interactions among product-line features. In *Proceedings of the 18th International Software Product Line Conference-Volume 1*, pages 242–246. ACM, 2014.

[SBRCT08] Sergio Segura, David Benavides, Antonio Ruiz-Cortés, and Pablo Trinidad. Automated merging of feature models using graph transformations. In *Generative and Transformational Techniques in Software Engineering II*, pages 489–505. Springer, 2008.

[Sch06] Markus Scheidgen. Cmof-model semantics and language mapping for mof 2.0 implementations. In *Model-Based Development of Computer-Based Systems and Model-Based Methodologies for Pervasive and Embedded Software, 2006. MBD/MOMPES 2006. Fourth and Third International Workshop on*, pages 10–pp. IEEE, 2006.

[Seg08] Sergio Segura. Automated analysis of feature models using atomic sets. In *SPLC (2)*, pages 201–207, 2008.

[SHT06] P-Y Schobbens, Patrick Heymans, and J-C Trigaux. Feature diagrams: A survey and a formal semantics. In *Requirements Engineering, 14th IEEE international conference*, pages 139–148. IEEE, 2006.

[SHTB07] Pierre-Yves Schobbens, Patrick Heymans, Jean-Christophe Trigaux, and Yves Bontemps. Generic semantics of feature diagrams. *Computer Networks*, 51(2):456–479, 2007.

[SK01] Juha Savolainen and Juha Kuusela. Violatility analysis framework for product lines. In *ACM SIGSOFT Software Engineering Notes*, volume 26, pages 133–141. ACM, 2001.

[SLB+11] Steven She, Rafael Lotufo, Thorsten Berger, Andrzej Wasowski, and Krzysztof Czarnecki. Reverse engineering feature models. In *ICSE 2011*, pages 461–470. IEEE, 2011.

[SRG11] Klaus Schmid, Rick Rabiser, and Paul Grünbacher. A comparison of decision modeling approaches in product lines. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*, pages 119–126. ACM, 2011.

[STE+10] Julio Sincero, Reinhard Tartler, Christoph Egger, Wolfgang Schröder-Preikschat, and Daniel Lohmann. Facing the linux 8000 feature nightmare. In *Proceedings of ACM European Conference on Computer Systems (EuroSys 2010), Best Posters and Demos Session*, 2010.

[SZFW05] Jing Sun, Hongyu Zhang, Yuan Fang, and Li Hai Wang. Formal semantics and verification for feature modeling. In *Engineering of Complex Computer Systems, 2005. ICECCS 2005. Proceedings. 10th IEEE International Conference on*, pages 303–312. IEEE, 2005.

[TBK09] T. Thum, D. Batory, and C. Kastner. Reasoning about edits to feature models. In *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, pages 254–264. IEEE, 2009.

[TC09] Pablo Trinidad and Antonio Ruiz Cortés. Abductive reasoning and automated analysis of feature models: How are they connected?. *VaMoS*, 9:145–153, 2009.

[TCT+08] Yih-Kuen Tsay, Yu-Fang Chen, Ming-Hsien Tsai, Wen-Chin Chan, and Chi-Jian Luo. Goal extended: Towards a research tool for omega automata and temporal logic. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 346–350. Springer, 2008.

[Tro92] Anne Sjerp Troelstra. *Lectures on Linear Logic*. CSLI Leacture Notes No 29, Center for the Study of Language and Information, Stanford University, 1992.

[Tsa93] Edward Tsang. *Foundations of constraint satisfaction*, volume 289. Academic press London, 1993.

[VDK02] Arie Van Deursen and Paul Klint. Domain-specific language design requires feature descriptions. *Journal of Computing and Information Technology*, 10(1):1–17, 2002.

[vGP95] Rob J. van Glabbeek and Gordon D. Plotkin. Configuration structures. In *LICS*, pages 199–209, 1995.

[vN02] Gertjan van Noord. Fsa6. 2xx: Finite state automata utilities. *http://odur. let. rug. nl/vannoord/Fsa/fsa. html, accessed*, 3(10):2003, 2002.

[Win82] Glynn Winskel. *Event structure semantics for CCS and related languages*. Springer, 1982.

[WK06] Manuel Wimmer and Gerhard Kramler. Bridging grammarware and modelware. In *Satellite Events at the MoDELS 2005 Conference*, pages 159–168. Springer, 2006.

[WLS+05] Hai Wang, Yuan Fang Li, Jing Sun, Hongyu Zhang, and Jeff Pan. A semantic web approach to feature modeling and verification. In *Workshop on Semantic Web Enabled Software Engineering (SWESE'05)*, 2005.

[WSB$^+$08] Jules White, Douglas C Schmidt, David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortés. Automated diagnosis of product-line configuration errors in feature models. In *Software Product Line Conference, 2008. SPLC'08. 12th International*, pages 225–234. IEEE, 2008.

[ZHD09] Lan Zhang, Ullrich Hustadt, and Clare Dixon. A refined resolution calculus for ctl. In *Automated Deduction–CADE-22*, pages 245–260. Springer, 2009.

List of Notations and Abbreviations

A list of mathematical notations and abbreviations used throughout the thesis can be found here. The page on which a notation is used for the first time is appended to the entry.

$+$	Choice operation on regular expression (pp 20)
$\#2^F$	Incomparable nodes in a tree with F as the nodes (pp 29)
$\#_m(n)$	Multiplicity of an ingredient n in a multiset m (pp 73)
$\mathcal{C}_t$	Multiplicities associated with a tee-like multiset t (pp 80)
$\mathcal{C}_U$	Overall multiplicities of a set of multisets of U (pp 88)
$\mathcal{FP}_{\mathbf{M}}$	Full products of $\mathbf{M}$ (pp 40)
AF	Future modality (pp 46)
AG	Global modality (pp 46)
$\mathcal{G}_t$	Groups associated with a tee-like multiset t (pp 79)

AX	Next modality (pp 46)
$\mathcal{PP}_{\mathbf{M}}$	Partial products of $\mathbf{M}$ (pp 41)
$\mathbb{P}(\mathbf{M})$	Partial Product Line of an FM $\mathbf{M}$ (pp 43)
$\mathcal{L}^{\mathrm{Par}}$	The Parikh image of a language $\mathcal{L}$ (pp 24)
w^{Par}	The Parikh image of a word w (pp 24)
$\Phi^{\mathsf{I2C}}_{\mathsf{BL}}(T_{\mathcal{OR}})$	Boolean theory of $\mathsf{I2C}$ (pp 39)
$\Phi^{\mathsf{I2C}}_{\mathsf{BL}}(P, f)$	Boolean Theory of fully instantiated P wrt. f (pp 42)
$\Phi_{\mathsf{BL}}(\mathbf{M})$	Boolean theory of partial products (pp 39)
$\Phi_{\mathsf{BL}}(\mathcal{EX})$	Boolean theory of exclusive constraints (pp 39)
$\Phi_{\mathsf{BL}}(T)$	Boolean theory of a tree (pp 39)
$\perp$	Logical false (pp 15)
$\cap$	Intersection operation on sets (pp 21)
$\mathfrak{C}$	The multiplicity set (pp 96)
$\mathcal{D}(F)$	The family of CFDs over F (pp 61)
$\mathcal{D}(F)$	The family of CFDs over F (pp 96)
$f_{\downarrow}$	Children of a node f in a tree (pp 29)
$\leq_{*}$	An ordering relation on $\mathbb{N}^{*}$ (pp 95)
$cplev(\mathbf{RD})$	Leaves all of whose siblings are leaves in a CFD $\mathbf{D}$ (pp 98)

$\cup$	Union operation on sets (pp 15)
$\mathbf{D}^{-f}$	Cutting of a CFD $\mathbf{D}$ by a node f (pp 189)
$depth(\mathbf{D})$	Depth of a CFD $\mathbf{D}$ (pp 97)
$dom(m)$	Domain of a multiset m (pp 61)
$\mathrm{dre}^{\mathsf{CRE}}$	CRE-DR function (pp 105)
$\mathrm{dre}^{\mathsf{ORE}}$	ORE-DR function (pp 114)
$\exists$	Existence quantifier (pp 21)
$F^{d\leftrightarrow}$	Nodes with depth d in a CFD (pp 172)
$\mathsf{ppKS}(F)$	Set of all ppCTL-formulas over F (pp 46)
$\mathcal{K}(F)$	Class of ppKSs over F (pp 45)
$\mathcal{L}(\mathcal{R})$	Language of a regular expression $\mathcal{R}$ (pp 20)
$\mathcal{L}_{\mathbf{D}}$	Language of a CFD $\mathbf{D}$ (pp 126)
$\mathcal{L}_{\mathbf{M}}$	Language of a CFM $\mathbf{M}$ (pp 126)
$\mathcal{L}_{cc}$	Language of a CCs cc (pp 126)
$\mathcal{L}^{\mathsf{CRE}}(\mathbf{M})$	CRE Language of a CFM $\mathbf{M}$ (pp 128)
$\mathcal{L}^{\mathsf{HRE}}(\mathbf{M})$	HRE Language of a CFM $\mathbf{M}$ (pp 128)
$\mathcal{L}^{\mathsf{ORE}}(\mathbf{M})$	ORE Language of a CFM $\mathbf{M}$ (pp 128)
$\mathcal{M}(F)$	The class of all FMs over F (pp 30)

$\Phi_{\mathsf{ML}+}(\mathbf{M})$	$\Phi_{\mathsf{ML}}(\mathbf{M}) \setminus \Phi_{\mathsf{ML}\subseteq}(\mathbf{M})$ (pp 48)
$\Phi_{\mathsf{ML}\subseteq}(\mathbf{M})$	Soundness ML theory of $\mathbf{M}$ (pp 46)
$\Phi_{\mathsf{ML}}(\mathbf{M})$	Completeness ML theory of $\mathbf{M}$ (pp 46)
$\forall$	Universal quantifier (pp 46)
$f_{\downarrow\downarrow}$	Grandchildren of a node f in a tree (pp 29)
$\mathcal{G}deg(C)$	Degree of depth of groups (pp 172)
$\mathcal{G}dep(\mathbf{D})$	Set of depth of groups (pp 172)
$\text{gex}^{\mathsf{CRE}}_{\mathbf{RD}}$	Mapping of grouped leaves to regular expressions in a CFD $\mathbf{D}$ (pp 102)
$\text{gex}^{\mathsf{ORE}}_{\mathbf{ORD}}$	Mapping of grouped leaves to ORE regular expressions in a CFD $\mathbf{D}$ (pp 111)
$\text{gle}^{\mathsf{CRE}}$	CRE-EGL function (pp 103)
$\text{gle}^{\mathsf{ORE}}$	ORE-EGL function (pp 112)
$f^{\uparrow\uparrow}$	Grandparents of a node f in a tree (pp 29)
$glev(\mathbf{D})$	Grouped leaves of a CFD $\mathbf{D}$ (pp 97)
$\mathbf{D}_{-k}$	Induced CFD by depth k (pp 167)
t^a	Induced tree-like multiset by a in t (pp 77)
$\inf(R)$	Infimum of a total order R (pp 107)
$\leq$	Usual ordering on natural numbers (pp 95)

$lev(\mathbf{D})$	Set of leaves of a CFD $\mathbf{D}$ (pp 97)
$\mathrm{lex}_{\mathbf{RD}}$	Mapping of leaves to regular expressions in a CFD $\mathbf{D}$ (pp 100)
$low(C)$	Lower bound of a multiplicity domain C (pp 96)
$low(c)$	Lower bound of a multiplicity c (pp 96)
$\longrightarrow$	Logical implication (pp 15)
$\longrightarrow^*$	Reflexive transitive closure of transition relation $\longrightarrow$ (pp 45)
$\longrightarrow^+$	Transitive closure of transition relation $\longrightarrow$ (pp 44)
$\max\limits_{a \in A}(a)$	Maximum element of a set A (pp 172)
$\mathrm{mel}^{\mathsf{CRE}}$	CRE-EML function (pp 101)
$\mathrm{mel}^{\mathsf{ORE}}$	ORE Multiplicity eliminator function (pp 109)
$\mathcal{D}_{U^{\mathrm{merge}}}$	family of minimal representative CFDs of U (pp 83)
$\mathcal{T}^{\mathrm{merge}}$	Merged tree of trees $\mathcal{T}$ (pp 84)
$\models$	Logical satisfaction relation (pp 9)
$\mathcal{H}(A)$	Hierarchy of finite multisets over A (pp 71)
$\mathsf{MultIng}(m)$	Multiset ingredients of m (pp 73)
$\mathcal{MS}(A)$	Class of finite multisets over A (pp 70)
$\mathbb{N}$	The set of natural numbers (pp 20)
$\mathbb{N}^*$	$\mathbb{N} \cup \{*\}$ (pp 95)

$\neg$	Logical negation (pp 46)
$\#_w(\sigma)$	The number of occurrences of σ in a word w (pp 24)
$\mathcal{ORD}(\Sigma)$	Class of osCRDs over Σ (pp 108)
$\mathcal{OD}(\mathbf{FD})$	Set of all osCFDs all of whose underlying CFDs are $\mathbf{D}$ (pp 108)
$\mathcal{OD}(F)$	Class of osCFDs over F (pp 108)
$Per^k_{\leq}(X)$	$Per^{(k,k)}_{\leq}(X)$ (pp 111)
$Per^k(X)$	$Per^{(k,k)}(X)$ (pp 102)
$Per^{(l,u)}_{\leq}(X)$	Concatenation permutations with length between l and u of X considering a total ordering on X (pp 111)
$Per^{(l,u)}(X)$	Concatenation permutations with length between l and u of X (pp 102)
$\mathrm{pex}^{\mathsf{CRE}}_{\mathbf{RD}}$	Mapping of parents of leaves to CRE-Expressions (pp 105)
$\mathrm{pex}^{\mathsf{ORE}}_{\mathbf{ORD}}$	Mapping of parents of leaves to ORE regular expressions (pp 113)
$\mathcal{P}^{\mathrm{flat}}(\mathbf{D})$	Flat semantics of a CFD $\mathbf{D}$ (pp 64)
$\mathcal{P}^{\mathrm{flat}}(\mathbf{D}, G)$	Grouped flat products of G in $\mathbf{D}$ (pp 65)
$\preceq$	Specialization relation on FMs (pp 53)
$plev(\mathbf{RD})$	Non-leaf nodes all of whose children are leaves in a CFD $\mathbf{D}$ (pp 98)
$a^{\uparrow}$	Parent of a in a tee-like multiset (pp 77)
$f^{\uparrow}$	Parent of a node f in a tree (pp 29)

$\mathcal{P}(\mathbf{D})$	Hierarchical semantics of a CFD $\mathbf{D}$ (pp 72)
$\mathcal{P}(\mathbf{D}, G)$	Grouped hierarchical products of G in $\mathbf{D}$ (pp 72)
$rank(m)$	Rank of a multiset m (pp 71)
$\mathcal{RD}(\Sigma)$	The class of all CRDs over the same alphabet Σ (pp 97)
$\mathbf{RE}(\Sigma)$	The class regular expressions built over Σ (pp 23)
$\mathcal{R}^+$	$\mathcal{R}\mathcal{R}^*$ for a regular expression $\mathcal{R}$ (pp 24)
$\mathcal{R}^{\mathsf{CRE}}(\mathbf{RD})$	CRE regular expression of a CFD $\mathbf{D}$ (pp 106)
$\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD})$	HRE regular expression of an osCFD $\mathbf{OD}$ (pp 116)
$\mathcal{R}^{\mathsf{HRE}}(\mathbf{OD})$	HRE regular expression of an osCFD $\mathbf{OD}$ (pp 121)
$\mathcal{R}^{\mathsf{HRE}}(f)$	HRE node expression of a node f in an osCFD (pp 120)
$\mathcal{R}^{\mathsf{HRE}}(G)$	HRE group expression of a group G in an osCFD (pp 120)
m°	Relaxed multiset of m (pp 85)
U°	Set of relaxed multisets of U (pp 85)
$m[x/y]$	Replacement of an element with another element in a multiset (pp 166)
$f\big\|_A$	Restriction of a function f to a subdomain A (pp 67)
$R\big\|_A$	restriction of R on A (pp 108)
$\mathcal{R}^{\mathsf{ORE}}(\mathbf{ORD})$	ORE regular expression generated for an osCRD $\mathbf{ORD}$ (pp 115)
$root(t)$	Root of a tree-like multiset t (pp 75)

$\sqsubseteq_w$	A partial order on the domain of a word w (pp 24)
$\setminus$	Setminus operation (pp 29)
C^{set}	Set interpretation of a multiplicity domain C (pp 96)
$\mathrm{shr}^{\mathsf{CRE}}$	CRE-Shrinking function (pp 106)
$\mathrm{shr}^{\mathsf{ORE}}$	ORE-shrinking function (pp 114)
$\Sigma(\mathcal{L})$	The alphabet which $\mathcal{L}$ is built on (pp 187)
$\Sigma(\mathcal{R})$	The alphabet which $\mathcal{R}$ is built on (pp 187)
$\simeq$	Refactoring relation on FMs (pp 53)
$\leq_{\mathrm{seq}}$	Subsequence relation on words (pp 24)
$\subseteq$	Sub-ppKS relation (pp 45)
$\subseteq$	Subset relation on sets (pp 21)
$\mapsto_{\mathrm{D}}$	Substitution of a leaf node with a CFD (pp 188)
$\mapsto_{\mathrm{E}}$	Substitution of a letter in a regular expression with another expression (pp 188)
$\mapsto_{\mathrm{L}}$	Substitution of a letter in a language with another language (pp 188)
$\mapsto_{\mathrm{P}}$	Substitution of an element in a flat product with another flat product (pp 187)
$\sup(R)$	Supremum of a total order R (pp 107)

$\top$	Logical truth (pp 46)
$\mathcal{TH}(A)$	Hierarchy of tree-like multisets over A (pp 74)
$T^{f}_{\mathcal{OR}}$	Induced subfeature tree by f (pp 41)
$\mathbf{OD}^{\mathrm{cfd}}$	Underlying CFD of an osCFD **OD** (pp 107)
$\mathbf{ORD}^{\mathrm{crd}}$	Underlying CRD of an osCRD **ORD** (pp 108)
$up(C)$	Upper bound of a multiplicity domain C (pp 96)
$up(c)$	Upper bound of a multiplicity c (pp 96)
$\uplus$	Additive union operator on multisets (pp 17)
ε	Empty string (pp 20)
$\varnothing$	Empty regular expression (pp 20)
$\vee$	Logical OR (pp 30)
$\wedge$	Logical conjunction (pp 15)
*	Kleene star (pp 20)
F_{-r}	For a set F and $r \in F$ (pp 29)
$flat_A$	Flattening function over A (pp 73)
$P \longrightarrow P'$	Transition from P to P' (pp 43)
T_t	Tree associated with a tree-like multiset t (pp 78)
U_w	The domain of a word w (pp 24)

BL	Boolean Propositional Logic (pp 25)
ppCTL	partial product CTL (pp 46)
CRE	CRDs to Regular Expressions (pp 93)
ppKS	partial product Kripke Structure (pp 44)
HRE	Hierarchical semantics to Regular Expressions (pp 93)
I2C	Instantiate to Completion (pp 35)
ML	Modal Logic (pp 46)
ORE	osCFDs to Regular Expressions (pp 93)
CC	Crosscutting Constraint (pp 15)
CFD	Cardinality-based Feature Diagram (pp 16)
CFM	Cardinality-based Feature Model (pp 17)
CRD	Cardinality-based Regular expression Diagaram (pp 93)
FD	basic Feature Diagram (pp 14)
FM	basic Feature Model (pp 14)
FSA	Finite State Automaton (pp 21)
osCFD	Ordered siblings CFD (pp 93)
osCRD	Ordered sibling CRD (pp 93)
PPL	Partial Product Line (pp 30)

r.e.	Recursively enumerable (pp 22)