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

© Copyright by Aliakbar Safilian, 2016
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

Abstract iv

Acknowledgements v

1 Introduction 1
  1.1 Product Line Engineering and Feature Models . . . . . . . . . . . . . 1
  1.2 Research Questions . . . . . . . . . . . . . . . . . . . . . . . . . . 5
  1.3 Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
  1.4 Organization . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

2 Background 13
  2.1 Feature Modeling Languages . . . . . . . . . . . . . . . . . . . . . . 13
  2.2 Formal Languages . . . . . . . . . . . . . . . . . . . . . . . . . . . 18

3 Modal Logic Theory of Basic Feature Models 25
  3.1 Basic Feature Models: A Formal Framework . . . . . . . . . . . . . . 28
  3.2 Partial Product Lines: Motivation . . . . . . . . . . . . . . . . . . . 30
    3.2.1 Products as Processes . . . . . . . . . . . . . . . . . . . . . . 30
    3.2.2 PPLs: From lattices to transition systems . . . . . . . . . . . 32
## List of Figures

1.1 A tree of features for the vehicle product line . . . . . . . . . . . . . . 3  
1.2 Feature constraints on features trees of vehicle product line . . . . . 4  
2.1 An FM of a car system . . . . . . . . . . . . . . . . . . . . . . . . 14  
2.2 A CFM of a car system . . . . . . . . . . . . . . . . . . . . . . . . 17  
2.3 Transition graphs: example . . . . . . . . . . . . . . . . . . . . . . 21  
2.4 A containment hierarchy of formal languages . . . . . . . . . . . . . . 22  
3.1 Two FMs with the same Boolean semantics . . . . . . . . . . . . . . . 26  
3.2 From FMs to PPLs: simple cases . . . . . . . . . . . . . . . . . . . . 31  
3.3 An FM: a fragment of Figure 2.1 . . . . . . . . . . . . . . . . . . . . 33  
3.4 A fragment of the PPL of Figure 3.3 . . . . . . . . . . . . . . . . . . 34  
3.5 Exclusion of an edge due to I2C . . . . . . . . . . . . . . . . . . . . . 36  
3.6 An FM of an Engine Frame (a), and its PPL (b) . . . . . . . . . . . 55  
4.1 A CFD: running example . . . . . . . . . . . . . . . . . . . . . . . . 61  
4.2 Two different CFDs with the same flat semantics . . . . . . . . . . . 68  
4.3 Diagram induced by a node: an example . . . . . . . . . . . . . . . . 68  
4.4 The representation of Figure 4.1 in terms of induced diagrams . . . . 69  
4.5 Trees associated with tree-like multisets: example . . . . . . . . . . . 80  
4.6 Representative CFDs of single tree-like multisets: example . . . . . . 82
4.7 Representative CFDs of mergeable tree-like multisets: example . . . . 83
4.8 Representative CFDs of mergeable tree-like multisets: example . . . . 84
4.9 Megeable trees and their representative trees: an example . . . . . 85
4.10 Minimal representative CFDs of $U$ and $U^\circ$ . . . . . . . . . . . . . . . 88
4.11 Minimal representative CFDs of $U_1$ and $U_1^\circ$ . . . . . . . . . . . . . . 88
5.1 A CFD: running example for transformations . . . . . . . . . . . . . . 96
5.2 CRD to RE: shrinking procedure on Figure 4.1. . . . . . . . . . . . . 100
5.3 Difference between ORE and CRE: example . . . . . . . . . . . . . . 123
5.4 Faithfulness in CRE and HRE: example . . . . . . . . . . . . . . . . . 125
5.5 A Computational Hierarchy of CFMs . . . . . . . . . . . . . . . . . . . . 130
6.1 An FM adopted from [dJV02] . . . . . . . . . . . . . . . . . . . . . 138
6.2 Two FDs with the same grammar in the de Jong and Visser approach 140
6.3 An FM adopted from [HKM11a] . . . . . . . . . . . . . . . . . . . . 142
6.4 PPLs vs Staged configuration: a staged configuration . . . . . . . . . 146
6.5 PPLs vs Stage configuration: A PPL . . . . . . . . . . . . . . . . . . . . 146
B.1 $D_2$: The diagram induced by depth 2 of $D_1$ . . . . . . . . . . . . . . 169
C.2 Substitution of a Leaf Node with a CFD: An example . . . . 189
C.3 Cutting of CFD by nodes: an example . . . . . . . . . . . . . . . . . 190
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 \textit{features}. 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., \textit{engine}, \textit{gear}, \textit{body}, \textit{brake}, \textit{wheel}, \textit{window}, \textit{door}, etc. They may also have some variable (a.k.a. optional) features: different engine types (e.g., \textit{gas} or \textit{electric}), different gear types (e.g., \textit{automatic} or \textit{manual}), they may be optionally equipped with an anti-skidding (\textit{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 \textit{orthogonal variability modeling} [PBVDL05], \textit{decision modeling} [SRG11], and \textit{feature modeling}.
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
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 \textit{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 $M_1$ ($a$ is the root and $b$ is the only mandatory child of $a$) and $M_2$ ($b$ is the root and $a$ is the only mandatory child of $b$). $M_1$ and $M_2$ 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 $M_1$ is a refactoring of another feature model $M_2$, or a specialization of $M_2$, 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 Cardinality-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 $M$ can be represented by a partial product CTL theory $\Phi(M)$ accurately specifying $M$'s intended semantics: the main result states that for any partial product Kripke structure $K$, $K \models \Phi(M)$ iff $K$ is equal to $M$’s partial product line, and hence $\Phi(M)$ is a sound and complete representation of the feature model. In other words, $\Phi(M)$ provides a faithful logical theory of $M$.

(ii) Multiset Theory of Cardinality-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]
showed that all the above languages have the same expressiveness. These languages are known as \textit{basic} feature modeling languages. Czarnecki et al in [CHE04] proposed another language called \textit{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 \textit{basic feature model} (FM)\footnote{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.).} of the system. The main part of an FM, called a \textit{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 \textit{mandatory} feature: every \texttt{vehicle} must include \texttt{engine}, \texttt{gear}, \texttt{axle}, and \texttt{brake}; a \texttt{gear} must include \texttt{oil}. An edge with a hollow circle (○) shows an \textit{optional} feature: a \texttt{brake} can be optionally equipped with \texttt{abs}. These two types of edges (mandatory and optional) are called \textit{solitary}.

Other edges are \textit{grouped} into two groups: OR (denoted by black angles ▲) and
XOR (denoted by hollow angles $\Delta$). The OR group \{gas, electric\} indicates that an engine can be either gasoline or electric, or both. The XOR group \{automatic, 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:

- $cc_1 : \text{automatic} \rightarrow \text{abs}$
- $cc_2 : \text{electric} \land \text{manual} \rightarrow \bot$

$cc_1$ 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 $C = \{\text{vehicle, engine, gear, axle, wheel, brake, oil}\}$ represents the set of the common features in all products of the example. This FM represents the following 8 valid products:

- $C \cup \{\text{gas, automatic}\}$,
- $C \cup \{\text{gas, automatic, abs}\}$,
- $C \cup \{\text{gas, manual}\}$,
- $C \cup \{\text{gas, manual, abs}\}$,
- $C \cup \{\text{electric, automatic}\}$,
- $C \cup \{\text{electric, automatic, abs}\}$,
- $C \cup \{\text{gas, electric, automatic}\}$,

\(^2\) Note that it is not possible to visually show all CCs in this way, as a CC may involve more than two features.
As some examples for invalid products, consider the following sets:

- $C \cup \{\text{gas, electric, automatic, abs}\}$.
- $C \cup \{\text{gas, manual, automatic}\}$,
- $C \cup \{\text{electric, manual}\}$,
- $C \cup \{\text{electric, abs, automatic}\}$.

They violate the constraints XOR on $\{\text{automatic, manual}\}$, $cc_2$, and $cc_1$, respectively.

The set of all products of a given FM $M$ 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 $\star$ (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 $\{\text{gas, electric}\}$ in
terms of multiplicity constraints, we use the multiplicity domain \((1, 2)\) on the group. The XOR group \{\text{automatic, manual}\} is modeled using the multiplicity domain \((1, 1)\).

The feature multiplicity domain \((0, 1)\) on \text{abs} models its optional presence in a \text{brake}. The feature multiplicity domain \((2, 5)(7, *)\) on \text{axle} and \((2, 2)\) on \text{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 \text{engine}, \text{brake}, \text{gear}, \text{gas}, \text{electric}, \text{automatic}, \text{oil}, and \text{manual} are \((1, 1)\).

CCs in a \textit{cardinality-based feature model} (CFM) can refer to feature occurrences. Take, for example, the constraint: \text{cc}_3: “if 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 \textit{multiset} of features satisfying the constraints. For example, the following multisets are valid products of our example, where \(C = \left[ \text{vehicle}, \text{engine}, \text{brake}, \text{gear}, \text{oil} \right]^{\text{3}}\):

\[ C \uplus \left[ \text{gas}, \text{automatic}, \text{abs}, \text{axle}^{2}, \text{wheel}^{4} \right]^{\text{3}}, \]

\(^{3}\) We use brackets ‘[’, ‘]’ as multiset identifiers.
The following multisets are not valid products of the example:

- $C \uplus \{ \text{electric, automatic, abs, axle}^4, \text{wheel}^8 \}$
- $C \uplus \{ \text{gas, abs, manual, axle}^3, \text{wheel}^6 \}$

They violate the constraints $cc_3$ and the group multiplicity $(1, 1)$ on $\{\text{automatic, manual}\}$, 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 $\Sigma$. $\Sigma^*$ denotes the set of all finite words (sequences of occurrences of symbols) built over $\Sigma$. Any subset of $\Sigma^*$ 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 $L$ 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 $L$ and outputs a symbol indicating that the input is in $L$. 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 $L$ 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 $\Gamma$ and $\Theta$ are strings generated over terminals and non-terminals and $\Theta$ is not shorter than $\Gamma$. 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 $\Theta$ 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):

\[
\mathcal{R} ::= \emptyset \mid \varepsilon \mid \sigma \text{ (for any } \sigma \in \Sigma) \mid \mathcal{R} + \mathcal{R} \mid \mathcal{R} \cdot \mathcal{R} \mid \mathcal{R}^* \mid (\mathcal{R}).
\]

The expressions \( \emptyset, \varepsilon, \sigma \) (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:

\[
\begin{align*}
\mathcal{L}(\emptyset) &= \emptyset, \\
\mathcal{L}(\varepsilon) &= \{\varepsilon\}, \\
\mathcal{L}(\sigma) &= \{\sigma\}, \text{ for any } \sigma \in \Sigma, \\
\mathcal{L}(\mathcal{R}_1 + \mathcal{R}_2) &= \mathcal{L}(\mathcal{R}_1) \cup \mathcal{L}(\mathcal{R}_2), \\
\mathcal{L}(\mathcal{R}^*) &= (\mathcal{L}(\mathcal{R}))^*, \\
\mathcal{L}((\mathcal{R})) &= \mathcal{L}(\mathcal{R}), \\
\mathcal{L}(\mathcal{R}_1 \cdot \mathcal{R}_2) &= \mathcal{L}(\mathcal{R}_1).\mathcal{L}(\mathcal{R}_2).
\end{align*}
\]

We extend regular expressions by a definable operation \( \mathcal{R}^n \) (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 \( S \)
is a finite set of states, \( T : S \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^S \) 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 \( T^* : S \times \Sigma^* \rightarrow 2^S \) 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 \( A = (S, T, F, I) \) be an FSA. The language associated with \( A \) is denoted by \( L(A) \) and is equal to \( \{ w \in \Sigma^* : \exists i \in I, T^*(i, w) \cap F \neq \emptyset \} \).

**Transition graphs** are sometimes used to visually represent finite state automata. Figure 2.3 represents an FSA for the language \( \{ w \in \{a, b\}^* : \text{the number of occurrences of } a \text{ is even} \} \). 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: \( V \rightarrow \varepsilon \) (the same, respectively), \( V \rightarrow \sigma \) (the same, respectively), \( V \rightarrow \sigma V' \) (\( V \rightarrow V' \sigma \), respectively), where \( \sigma \) 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 \( L \subseteq w_1^* \ldots w_n^* \).

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 \( \mathcal{L} = \emptyset \) 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 \( L \) and \( L' \), the equality problem is to decide whether \( L = L' \) 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 \( L \) and \( L' \), the inclusion problem is to decide whether \( L \subset L' \) or not. The problem is decidable in the class of regular languages, but it is undecidable in other classes of formal languages. However, if \( L \) is context-free and \( L' \) 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 \( \sigma_i \) 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 \( \sigma_i \) 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 \( \Sigma \) be an alphabet.

- For any \( \sigma \in \Sigma \), let \( \sigma^n \) denote the word \( \underbrace{\sigma \ldots \sigma}_{n} \).

- \( \text{RE}(\Sigma) \) denotes the class of all regular expressions built over \( \Sigma \).
• We use $R^+$ to denote $RR^*$ for any regular expression $R$.

• The Parikh image of a word $w$ (a formal language $L$, respectively) is denoted by $\text{Par}(w)$ ($\text{Par}(L)$, respectively).

• $U_w$ denotes the set of alphabet symbols included in a word $w$.

• $\#_w(\sigma)$ denotes the number of occurrences of $\sigma$ 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 $\sigma'$ is preceded by some occurrences of $\sigma$ in $w$.

**Definition 2.2.** We consider a relation $\preceq_{\text{seq}} \subseteq \Sigma^* \times \Sigma^*$ defined as $w \preceq_{\text{seq}} w'$ iff $w$ is a subsequence of $w'$.
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 $M_1$ 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: $P = \{\text{car, engine, brake}\}$ and $P' = P \cup \{\text{abs}\}$. 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
be exactly the valid products defined by the FM [Bat05]. For example, model $M_1$ represents the BL theory $\Phi(M_1) = \{\text{engine} \rightarrow \text{car}, \text{brake} \rightarrow \text{car}, \text{abs} \rightarrow \text{brake}\} \cup \{\text{car} \rightarrow \text{engine}, \text{car} \rightarrow \text{brake}\} \cup \{\text{car}\}$: 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 $M_2$ in Figure 3.1. This model is essentially different from $M_1$, but has the same set of products, $\{P, P'\}$ determined by an equivalent BL theory $\Phi(M_2) = \{\text{car} \rightarrow \text{engine}, \text{brake} \rightarrow \text{engine}, \text{abs} \rightarrow \text{engine}\} \cup \{\text{engine} \rightarrow \text{car}, \text{engine} \rightarrow \text{brake}\} \cup \{\text{engine}\}$: 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,\textsuperscript{1} which we think is theoretically unsatisfactory. Even more importantly, inadequate logical foundations for FM hinder practical analyses: as important information contained in FMs (\textit{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 \textit{instantiation process} rather than its final result. We call intermediate states of this process \textit{partial products}, and argue that what an FM $M$ really specifies is a partially ordered set of partial products, which we call a \textit{partial product line} generated by $M$. The commonly considered products of $M$ (we call them \textit{full}) only form 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 structures, which we axiomatically define and call a \textit{partial product Kripke structure} ($\text{ppKS}$). The latter are specifiable by a suitable version of modal logic, which we call \textit{partial product CTL} ($\text{ppCTL}$), 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 $M$ can be represented by a $\text{ppCTL}$ theory $\Phi_{\text{ML}}(M)$ accurately specifying $M$’s intended semantics: the main theoretical result of the chapter states that for any $\text{ppKS} K$, $K \models \Phi_{\text{ML}}(M)$

\textsuperscript{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 $K$ is equal to $M$'s partial product line, and hence $\Phi_{\text{ML}}(M)$ 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 $T_{\text{OR}} = (T, \text{OR})$ 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 $f \in F_{\neg r} \overset{\text{def}}{=} F \setminus \{r\}$ to its parent $f_\uparrow$. The inverse function that assigns to each feature the set of its children (called subfeatures) is denoted by $f_\downarrow$; this set is empty for leaves. It is easy to see that the set of $f$’s siblings is the set $(f_\uparrow)_\downarrow \setminus \{f\}$. The set of all ancestors and all descendants of a feature $f$ are denoted by $f_\uparrow\uparrow$ and $f_\downarrow\downarrow$, respectively.

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

(ii) $\text{OR}$ is a function that assigns to each feature $f \in F$ a set $\text{OR}(f) \subset 2^{f_\downarrow}$ (possibly empty) of disjoint subsets of $f$’s children called OR-groups. If a group $G \in \text{OR}(f)$ 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, $\text{OR}(\text{gear}) = \{\{\text{manual, automatic}\}, \{\text{oil}\}\}$, and oil is a mandatory subfeature of gear.

Elements in set $O(f) \overset{\text{def}}{=} f_\downarrow \setminus \bigcup \text{OR}(f)$ are called optional subfeatures of $f$. For example, in Figure 2.1, $\text{OR}(\text{brake}) = \emptyset$, 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 $M = (T_{\text{OR}}, \mathbf{EX}, \mathbf{IN})$ with $T_{\text{OR}}$ a feature diagram as defined in Definition 3.1, and two additional components defined below:

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

(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 $(f_1 \land \ldots \land f_m) \rightarrow (g_1 \lor \ldots \lor g_n)$. For example, feature model in Figure 2.1 has $\mathcal{IN} = \{\text{automatic} \rightarrow \text{abs}\}$.

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

Thus, an FM is a tree of features $T$ endowed with three extra structures $\mathcal{OR}$, $\mathcal{EX}$, and $\mathcal{IN}$. We will sometimes write it as a quadruple $M = (T, \mathcal{OR}, \mathcal{EX}, \mathcal{IN})$. If needed, we will subscript $M$’s components with index $M$, e.g., write $F_M$ for the set of features $F$. Note that an FM is a purely syntactic object contrary to the common usage of term ‘model’ in logic.

The class of all 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
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 M₁ in Figure 3.2 a) and gradually progressing towards fuller products (say, \{car\} \rightarrow \{car, engine\} \rightarrow \{car, engine, brake\} or \{car\} \rightarrow \{car, brake\} \rightarrow \{car, brake, abs\} \rightarrow \{car, brake, abs, engine\}); we will call such sequences instantiation paths. The graph in Figure 3.2(b1) specifies all possible instantiation paths for M₁ (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., \( \{c\} \rightarrow_e \{c,e\} \), where the subscript denotes the added feature.

We call the instantiation graph described above the PPL determined by FM \( M_1 \), and write \( PPL(M_1) \) or \( PPL_1 \). In a similar way, the PPL of the second FM, \( PPL(M_2) \), is built in 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 \( \{c\} \) and \( \{c,b\} \) present in \( PPL_1 \) but absent in \( PPL_2 \), whereas \( \{e\} \) and \( \{e,a\} \) are present in \( PPL_2 \) but absent from \( PPL_1 \)) too. This essential difference between PPLs properly reflects the essential difference between the FMs. Note that capturing the difference between the two FMs \( M_1 \) and \( M_2 \) is important, as they represent two different product lines: The first model \( (M_1) \) represents a product line for cars, while the second one \( (M_2) \) 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 \( M_{1,2} \) 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 \( M_2 \) is pathological, as the feature car cannot be a subfeature of the feature engine in the reality.
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 $M_{1,2}$ (engine $\land$ 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.
the PPL, we follow the idea described above, and first consider $M_3$ as a pure tree-based poset with all the extra-structure (denoted by black bullets and black triangles) removed. Figure 3.4 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 $\bot$ (a feature to be a subfeature of any feature), and write
two defining equations: \( \text{ele} \land \text{manual} = \bot \) and \( \text{manual} \land \text{automatic} = \bot \). Then, in the filter lattice, the formal down-join of products \( \{ \text{c, en, ele, ge} \} \) and \( \{ \text{c, ge, mnl, en} \} \) “blow up” and become equal to the set of all features ("False implies everything"). The same happens with the other pair of conflicting products.

Next we consider the mandatoriness structure of FM \( M_3 \) (given by black bullets and triangles). This structure determines a subset of the PPL consisting of full products (e.g., \( \{ \text{c, en, gas, ge, mnl} \} \) in Figure 3.4) as we discussed above. In addition, mandatoriness affects the set of valid partial products as well.

Consider the product \( P = \{ \text{c, en, ge} \} \) at the center of the diagram (Figure 3.4). The left instantiation path, i.e., \( \{ \text{c} \} \rightarrow_{\text{en}} \{ \text{c, en} \} \rightarrow_{\text{ge}} P \), 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 \( \{ \text{c, ge} \} \rightarrow_{\text{en}} P \) 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
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 $M_4$ and its PPL. In the latter, the “diagonal” transition $\{c, ge\} \rightarrow \{c, en, ge\}$ violates I2C and must be removed. However, its target product is still reachable from $\{c, en\}$ 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.\footnote{A relation $R \subseteq A \times B$ is left-total if $\forall a \in A, \exists b \in B : (a, b) \in R.$}

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 $\text{BL}$ encoding of an FM,
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 $f' \rightarrow f$ (if a product has feature $f'$, it must have feature $f$ as well). If $\{g_1, g_2\}$ is an OR-group of $f$’s subfeatures, we write $f \rightarrow (g_1 \lor g_2)$; if, in addition, features $g_1, g_2$ are mutually exclusive, we write $g_1 \land g_2 \rightarrow \bot$. In this way, given an FM $M = (T, \text{OR}, \text{EX}, \text{IN})$, 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 $\text{BL}$ is needed because later we will also consider modal theories encoded by FMs. Together these theories constitute theory $\Phi^i_{\text{BL}}(M)$, and a set of features $P$ is a legal full product for $M$ iff $P \models \Phi^i_{\text{BL}}(M)$.

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 I2C-principle can be propositionally encoded (subsection B).

$^5\lor G$ and $\land G$ represent conjunction and disjunction of all formulas in a set of formulas $G$. 
Table 3.1: Boolean theories extracted from an FM $M = (T_{\text{OR}}, \mathcal{E}, \mathcal{I})$

<table>
<thead>
<tr>
<th>Theory</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) $\Phi_{\text{BL}}(T)$</td>
<td>${ \top \rightarrow r } \cup { f' \rightarrow f : f \in F, f' \in f_i }$</td>
</tr>
<tr>
<td>(2) $\Phi_{\text{BL}}(\mathcal{E})$</td>
<td>${ \bigwedge G \rightarrow \bot : G \in \mathcal{E} }$</td>
</tr>
<tr>
<td>(3') $\Phi_{\text{BL}}^i(\mathcal{O})$</td>
<td>${ f \rightarrow \bigvee G : f \in F, G \in \mathcal{O}(f) }$</td>
</tr>
<tr>
<td>(4') $\Phi_{\text{BL}}^i(\mathcal{I})$</td>
<td>${ \bigwedge G \rightarrow \bigvee G' : (G, G') \in \mathcal{I} }$</td>
</tr>
<tr>
<td>(all') $\Phi_{\text{BL}}^i(M)$</td>
<td>$\Phi_{\text{BL}}(T) \cup \Phi_{\text{BL}}(\mathcal{E}) \cup \Phi_{\text{BL}}^i(\mathcal{O}) \cup \Phi_{\text{BL}}^i(\mathcal{I})$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Theory</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>(3) $\Phi_{\text{BL}}^{\text{I2C}}(T_{\text{OR}})$</td>
<td>${ f \land g \rightarrow (\bigwedge \Phi_{\text{BL}}^i(T_{\text{OR}}^f)) \lor (\bigwedge \Phi_{\text{BL}}^i(T_{\text{OR}}^g)) : f, g \in F, f^\dagger = g^\dagger } $</td>
</tr>
<tr>
<td>(all) $\Phi_{\text{BL}}(M)$</td>
<td>$\Phi_{\text{BL}}(\text{BL})T \cup \Phi_{\text{BL}}(\mathcal{E}) \cup \Phi_{\text{BL}}^{\text{I2C}}(T_{\text{OR}})$</td>
</tr>
</tbody>
</table>

**A: Enabling vs. Causality.**

The encoding above has a drawback that we discussed in the introduction: two different relationships between features (being a subfeature, $f' \rightarrow f$, and being a mandatory subfeature, $f \rightarrow f'$) are similarly encoded. This implies $f \leftrightarrow f'$ for any mandatory subfeature $f'$ of $f$, and leads to misrepresentation of the 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
Thus, we have two Boolean theories for the same FM \( M \). One is the theory of partial products and another is the theory of full products. The theory of partial products is denoted by \( \Phi_{BL}(M) \) (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: \( \Phi_{BL}(T) \) is the BL-encoding of subfeature dependencies (row (1)), \( \Phi_{BL}(E\mathcal{X}) \) is the BL-encoding of exclusive dependencies (row (2)), and in section B we will consider yet another ingredient—the Boolean encoding of the I2C-condition, \( \Phi_{BL}^{I2C}(OR) \). The other propositional theory, \( M \)'s full product theory \( \Phi_{BL}^!(M) \), consists of four components: \( \Phi_{BL}(T) \) and \( \Phi_{BL}(E\mathcal{X}) \) as above, plus the BL-encoding \( \Phi_{BL}^!(OR) \) of the mandatoriness dependencies embodied in the \( OR \)-structure (row (3')), plus the Boolean logic encoding \( \Phi_{BL}^!(IN) \) 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 \( M = (T_{OR}, E\mathcal{X}, I\mathcal{N}) \) is a set of features \( P \subseteq F \) satisfying theory \( \Phi_{BL}^!(M) \) defined in Table 3.1.

The set of all full products is called \( M \)'s full product set and denoted by \( FP_M \).

\(^{6}\)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, $\mathcal{FP}_M = \{ P \subseteq F : P \models \Phi^{I2C}_{BL}(M) \}$. 

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

B: Instantiate to Completion via Propositional Logic.

Consider once again $\text{PPL}_3$ in Figure 3.4, from which product $\{c, en, ge\}$ is excluded as violating the I2C principle. Note that in order to specify this exclusion propositionally, we cannot declare that features $en$ and $ge$ are mutually exclusive and write $\{en \land ge \rightarrow \bot\}$ because further down the lattice they are combined in product $\{c, en, ele, ge\}$ below $\{c, en\}$, and in product $\{c, ge, mnl, en\}$ below $\{c, ge\}$ 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 $\Phi^{I2C}_{BL}(T\mathcal{OR})$ 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 $T_{\mathcal{OR}}^f = (T^f, \mathcal{OR}^f)$ with $T^f$ being the tree under $f$, i.e., $T^f \overset{\text{def}}{=} (f_{\downarrow\downarrow} \cup \{f\}, f, \uparrow)$, and mapping $\mathcal{OR}^f$ is inherited from $\mathcal{OR}$, i.e., for any $g \in f_{\downarrow\downarrow}$, $\mathcal{OR}^f(g) = \mathcal{OR}(g)$. 

Now we can specify theory $\Phi^{I2C}_{BL}(T\mathcal{OR})$ 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 $M = (T_{OR}, E, I, N)$ is a set of features $P \subseteq F$ satisfying the instantiation theory $\Phi_{BL}(M)$ specified in row (all) in Table 3.1. (Recall that a full product is a set of features satisfying theory $\Phi^I_{BL}(M)$.) We denote the set of all partial products by $\mathcal{PP}_M$. Thus, $\mathcal{PP}_M = \{ P \subseteq F : P \models \Phi_{BL}(M) \}$. $\Box$

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

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 $P \rightarrow P'$ 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, $P' \models (\Phi_{BL}(T) \cup \Phi_{BL}(E))$. Compatibility with l2C 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:

$$(3)_{P,f} \quad \Phi^{l2C}_{BL}(P, f) \overset{\text{def}}{=} \bigcup \{ \Phi^I_{BL}(T^g_{OR}) : g \in P \cap (f^\uparrow)_i \}$$
where $T_{\mathcal{OR}}^g$ denotes the subtree induced by feature $g$ as described in Definition 3.4. (Note that set $P \cap (f^\uparrow)_1$ may be empty, and then theory $\Phi_{\mathcal{BL}}^{12c}(P, f)$ is also empty.)

We say $P$ is fully instantiated wrt. $f$ if $P \models \Phi_{\mathcal{BL}}^{12c}(P, f)$.

For example, it is easy to check that for FM $M_4$ in Figure 3.5, for product $P_1=\{\text{car, engine}\}$ and feature $f_1=\text{gear}$, we have $P_1 \models \Phi_{\mathcal{BL}}^{12c}(P_1, f_1)$ while for $P_2=\{\text{car, gear}\}$ and $f_2=\text{engine}$, $P_2 \not\models \Phi_{\mathcal{BL}}^{12c}(P_2, f_2)$ because $\Phi_{\mathcal{BL}}^{12c}(T_{\mathcal{OR}}^{\text{gear}}) = \{\text{gear} \rightarrow \text{oil}\}$ and $P_2 \not\models \{\text{gear} \rightarrow \text{oil}\}$.

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 $P \rightarrow P'$, 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) $P' \models \Phi_{\mathcal{BL}}(T)$, (a2) $P' \models \Phi_{\mathcal{BL}}(\mathcal{E} \mathcal{X})$, and (a3) $P \models \Phi_{\mathcal{BL}}^{12c}(P, f)$.

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

That is, $P \rightarrow P'$ iff $((a1) \land (a2) \land (a3)) \lor (b)$

The following result is important.

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

Finally, we formalize partial product lines as follows:

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


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 $K = (W, R, L)$ 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 $(\text{ppKS})$ over $F$ is a triple $K = (\mathcal{P}_P, \rightarrow, I)$ with $\mathcal{P}_P \subset 2^F$ a set of non-empty (partial) products, $I \in \mathcal{P}_P$ the initial singleton product (i.e., $I = \{r\}$ for some $r \in F$), and $\rightarrow \subseteq \mathcal{P}_P \times \mathcal{P}_P$ a binary left-total transition relation. In addition, the following three conditions hold ($\rightarrow^+$ denotes the transitive closure of $\rightarrow$):

Singleonicity: For all $P, P' \in \mathcal{P}_P$, if $P \rightarrow P'$ and $P \neq P'$, then $P' = P \uplus \{f\}$ for

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

44
some $f \notin P$.

(Reachability) For all $P \in \mathcal{PP}$, $I \rightarrow^+ P$, i.e., $P$ is reachable from $I$.

(Self-Loops Only) For all $P, P' \in \mathcal{PP}$, if $(P \rightarrow^+ P' \rightarrow^+ P)$, then $P = P'$, i.e., every loop is a self-loop.

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

The components of an ppKS $K$ are subscripted with $k$ if needed, e.g., $\mathcal{PP}_k$. We denote the class of all ppKSs built over a set of features $F$ by $\mathcal{K}(F)$. Note that any partial product in a ppKS eventually evolves into a full product because $F$ is finite, $\rightarrow$ 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 $P \rightarrow^* P'$, where $\rightarrow^*$ denotes the reflexive transitive closure of $\rightarrow$. 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 $P \rightarrow^* P'$.

We will also need the notion of a sub-ppKS of a ppKS.

**Definition 3.10 (Sub-ppKS).** Let $K, K'$ be two ppKSs. We say $K$ is a sub-ppKS of $K'$, denoted by $K \subseteq K'$, iff $I_K = I_{K'}, \mathcal{PP}_K \subseteq \mathcal{PP}_{K'}$, and $\rightarrow_K \subseteq \rightarrow_{K'}$.

The following proposition is an obvious corollary of Definition 3.8.

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

The proposition above is not very interesting: there is a rich structure in $\mathbb{P}(M)$ that is not captured by the fact that $\mathbb{P}(M)$ is a ppKS—the class $\mathcal{K}(F)$ is too big.
We want to characterize $P(\text{M})$ in a more precise way by defining as small as possible a class of ppKSs to which $P(\text{M})$ 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 $\phi$ are given by the following grammar:

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

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

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

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

**Theorem 3.2 (Soundness).** $\mathbb{P}(M) \models \Phi_{\text{ML}}(M)$.

**Theorem 3.3 (Semi-completeness).** $K \models \Phi_{\text{ML}}^\subseteq(M)$ implies $K \subseteq \mathbb{P}(M)$.

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

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] ($M$ is a specialization of another FM $M'$ 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 $\Phi_{\text{ML}}(M)$ and $\Phi_{\text{ML}}(M)$ from small component theories, which specify the respective properties of $M$’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, $\Phi_{\text{ML}}(M)$, $\Phi_{\text{ML}}(M)$, and $\Phi_{\text{ML}+}(M)$ 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 $\Phi_{\text{ML}}(M)$ 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

<table>
<thead>
<tr>
<th>M</th>
<th>Semi-completeness</th>
<th>To Ensure Completeness</th>
<th>Completeness</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BL</td>
<td>ML</td>
<td></td>
</tr>
<tr>
<td>$T$</td>
<td>$\Phi_{BL}(T)$</td>
<td>$\emptyset$</td>
<td>$\Phi_{ML+}(T)$</td>
</tr>
<tr>
<td>$\mathcal{E}$</td>
<td>$\Phi_{BL}(\mathcal{E})$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$\mathcal{O}$</td>
<td>$\emptyset$</td>
<td>$\Phi_{ML}(\mathcal{O})$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$\mathcal{I}$</td>
<td>$\emptyset$</td>
<td>$\Phi_{ML}(\mathcal{I})$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$\mathcal{I}2C$</td>
<td>$\Phi_{12C_{BL}}(T_{OR})$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$\mathcal{F}P_M$</td>
<td>$\emptyset$</td>
<td>$\Phi_{ML}(M)$</td>
<td>$\Phi_{ML}(M)$</td>
</tr>
<tr>
<td>$\mathcal{P}P_M$</td>
<td>$\Phi_{BL}(M)$</td>
<td>$\emptyset$</td>
<td>$\Phi_{ML}^+(M) \cup \Phi_{ML+}(T_{OR}, \mathcal{E})$</td>
</tr>
<tr>
<td>$\mathcal{P}(M)$</td>
<td>$\Phi_{ML}(M)$</td>
<td>$\Phi_{ML}(M)$</td>
<td>$\Phi_{ML}(M)$</td>
</tr>
</tbody>
</table>
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 $\Phi^\downarrow_{ML}(OR)$ modally specifies the $OR$ structure to provide semi-completeness wrt. full product line (note the $!$ superindex). For another example, $\Phi^{12C}_{BL}(T_{OR})$ is a Boolean encoding of the 12C-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 $\Phi^\downarrow_{ML+}(T)$ 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 $\Phi_{BL}(T)$ generates a complete theory $\Phi_{ML}(T)$. A ppKS $K$ satisfying $\Phi_{ML}(T)$ is guaranteed to capture the tree structure $T$.

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

The theories corresponding to $OR$ deal with full products (states with self-loop transitions). The theory $\Phi^\downarrow_{ML\subseteq}(OR)$ is the modal version of the Boolean theory...
Table 3.4: Definitions of (basic) ppCTL theories

\[
\Phi_{\text{ML}+}(T) = \{ f \land \neg \bigvee_{f} \rightarrow \text{EX}g : f, g \in F, g^{\uparrow} = f \} \\
\Phi_{\text{ML} \leq}(\text{OR}) = \{ f \rightarrow \Box f \lor G : f \in F, G \in \text{OR}(f) \} \\
\Phi_{\text{ML} \leq}(\text{IN}) = \{ \land G \rightarrow \Box f : (G, G') \in \text{IN} \} \\
\Phi_{\text{ML} \leq}(\text{M}) = \{ ! \rightarrow \land \Phi_{\text{BL}}(\text{M}) \} \\
\Phi_{\text{ML}+}(\text{M}) = \{ \land \Phi_{\text{BL}}(\text{M}) \rightarrow ! \} \\
\Phi_{\text{ML} \leq}(T_{\text{OR}}) = \{ f \land \neg \land \Phi_{\text{BL}}(T_{\text{OR}}) \rightarrow \neg \text{EX}g : f, g \in F, f \neq g, f^{\uparrow} = g^{\uparrow} \} \\
\Phi_{\text{ML} \leq}(T_{\text{OR}}, \mathcal{EX}) = \{ \land \Phi_{\text{BL}}(f) \land \neg f \land \neg \bigvee \Phi_{\text{BL}}(f) \rightarrow \text{EX}f : f \in F \}, \text{ where} \\
\Phi_{\text{BL}}(f) = \{ g \rightarrow \Phi_{\text{BL}}(T_{\text{OR}}) : g, f \in F, g^{\uparrow} = f^{\uparrow}, g \neq f \} \\
\Phi_{\text{BL}}(\mathcal{E}) = \{ \land (G \setminus \{ f \}) : G \in \mathcal{E}, f \in G \}
\]

Φ_{\text{BL}}(\text{OR}) (Table 3.1). Consider an OR group G. The theory Φ_{\text{ML} \leq}(\text{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 \text{IN} is like \text{OR}’s: it also deals only with full products. The theory Φ_{\text{ML} \leq}(\text{IN}) is the modal version of the Boolean theory Φ_{\text{BL}}(\text{IN}). Let (G, G') be an inclusive constraint. The theory Φ_{\text{ML} \leq}(\text{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 $\Phi_{\text{ML} \subseteq (O \cup R)}$ and $\Phi_{\text{ML} \subseteq (I \cup N)}$ are derivable from the theory $\Phi_{\text{ML} \subseteq (M)}$. $\Phi_{\text{ML} \subseteq (M)}$ holding in a ppKS guarantees that any full product in the ppKS is a full product of $M$. On the other hand, any ppKS satisfying the theory $\Phi_{\text{ML} \subseteq (M)} (= \Phi_{\text{ML} \subseteq (M)} \cup \Phi_{\text{ML} \subseteq (M)})$ must include all full products of $M$ and only them.

Recall that the theory $\Phi_{\text{I2C} \subseteq (T \cup R)}$ (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 $\Phi_{\text{I2C} \not\subseteq (T \cup R)}$ 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, $\Phi_{\text{I2C} \subseteq (T \cup R)}$, would be the union of $\Phi_{\text{I2C} \subseteq (T \cup R)}$.

Recall that, according to Definition 3.5, a set of features is a valid partial product iff it satisfies the Boolean theory $\Phi_{\text{BL} \subseteq (M)}$. 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 $\Phi_{\text{ML} \subseteq (T \cup R)}$ and $\Phi_{\text{ML} \subseteq (T \cup R, \mathcal{E}, \mathcal{A})}$. Consider a state $P$ and a feature $f$ such that $f \notin P$. The theory $\Phi_{\text{ML} \subseteq (T \cup R, \mathcal{E}, \mathcal{A})}$ 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 $\Phi_{\text{ML} \subseteq (M)}$ and is equal to $\Phi_{\text{ML} \subseteq (M)} \cup \Phi_{\text{I2C} \subseteq (T \cup R, \mathcal{E}, \mathcal{A})}$.

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

### 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 $M$, we may be interested in checking whether $PL(M)$ is not empty [TC09], or whether a given set of features $G$ is a valid full product, i.e., $G \in PL(M)$ [KCH+90]. We may also be interested in finding the set of common (core) features among all full products, $\bigcap PL(M)$ [TC09], or checking whether $f$ is a core feature, i.e., $f \in \bigcap PL(M)$. Specifically, an important problem is to find so called *dead* features, which do not occur in any product [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 $M_1$ is a *refactoring* of another FM $M_2$, or a *specialization* of $M_2$, 


or neither [TBK09]. Standard definitions of refactoring and specialization are based on semantics, which in the Boolean case gives rise to defining refactoring \( M_1 \simeq M_2 \) as \( PL(M_1) = PL(M_2) \) and specialization \( M_1 \preceq M_2 \) as \( PL(M_1) \subseteq PL(M_2) \).

However, as we have seen above, the Boolean semantics is too poor and makes the definitions above inadequate for their goals (see the example in the introduction). Hence, in practice, to investigate refactoring and specialization, engineers should work with pairs \((PL(M), M)\), whose second component represents the feature hierarchical structure not captured by the first component. Working with such pairs brings two issues. First, it leads to obvious maintenance problems: if one of the components changes, the user must remember to propagate the changes to the other component. Second, having a syntactical “non-Boolean” object of analysis does not allow us to use SAT (or SMT) solvers. However, the PPL semantics allows the management of both issues. As our completeness theorem shows, \( PPL(M) \) adequately captures the feature hierarchy, and hence we can analyze a single object, \( PPL(M) \) or, equivalently, the modal theory \( \Phi_{\text{mL}}(M) \).

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 \( PPL(M) \) 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 \( PPL(M) \) satisfies \( l \rightarrow \bigwedge C \). Other syntactical analysis problems can be approached in the same way: an FM \( M \) is represented by a Kripke structure \( PPL(M) \), the problem to be analyzed is encoded by
a ppCTL-formula $\phi$, and a model checker tool is used for checking if $PPL(M) \models \phi$.

### 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: $(f_i \land e \land \neg e_i) \rightarrow AX \neg e_j$ 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 \(\text{ppCTL}\) theory of the product line (in other words, its PPL), reverse engineering of FMs becomes simpler and more manageable. This is because the given \(\text{ppCTL}\) 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 constraints. 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 \(\ast\) (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 \(f, f_1...6\). \(G\) denotes a group consisting of the features \(f_4, f_5,\) and \(f_6\), and any feature in \(F \setminus G\) is a solitary feature. The multiplicity domains are as follows: \((2, 3)\) on \(G\), \((1, 2)(4, \ast)\) 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 \(f_{4,5}\) 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 \((1,2)(4,\ast)\) 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
of natural numbers can be expressed as a finite sequence of intervals. 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 $D = (F, r, \uparrow, G, C)$ 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 $f \in F_r \triangleq F \setminus \{r\}$ 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) $G \subseteq 2^{F_r}$ is a set of grouped nodes. For all $G \in G$, $|G| > 1$, and all nodes in $G$ have the same parent, denoted by $G^{\uparrow}$. All groups in $G$ are disjoint, i.e., $\forall G, G' \in G : (G = G') \lor (G \cap G' = \emptyset)$. The nodes that are not in a group are called solitary nodes. Let $S$ denote the solitary nodes, i.e., $S = F_r \setminus \bigcup_{G \in G} G$.

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) $C: (F_{-r} \cup G) \rightarrow 2^\mathbb{N}$ is a total function called the multiplicity function. For any feature or group $e \in F_{-r} \cup G$, $C(e)$ represents the multiplicity constraint of $e$, where $C(e) \neq \{0\}$ and $C(e) \neq \emptyset$. In addition, for all $G \in G$, $C(G)$ 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 $\mathcal{D}$’s components with index $\mathcal{D}$, e.g., write $\mathcal{G}_D$.

**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 $m: A \rightarrow \mathbb{N}$, 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 $\text{dom}(m)$. A multiset $m$ is called *finite* if $\text{dom}(m)$ is finite.

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

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:

- $m_1 = [f, f_1^5]$ (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 ($C(f_1) = \mathbb{N} \setminus \{3\}$) 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.

- $m_2 = [f, f_1^5, f_2, f_3^3, f_4, f_5]$: 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.

- $m_3 = [f, f_1^5, f_2^2, f_3^6, f_4^2, f_5^2]$: 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 $D = (F, r, \uparrow, G, C)$ be a CFD. A multiset $m$ over $F$ is called a flat product of $D$ if the following conditions hold:

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

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

(iii) $\forall f \in S : 0 \not\in C(f) \land m(f^\uparrow) > 0 \implies m(f) > 0,$

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

The set of flat products of $D$, denoted by $P_{\text{flat}}(D)$, is called the flat semantics of $D$. $\square$

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 $0 \not\in C(f)$ 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 $D = (F, r, \uparrow, G, C)$ be a CFD and $G = \{f_1, f_2, \ldots, f_k\} \in G$ for $k \in \mathbb{N}$. A multiset $m$ over $F$ is a grouped flat product associated with $G$ if there exist $c \in C(G), c_i \in C(f_i), g_i \in \{0, 1\}$, and $m_i \in \mathcal{P}_{\text{flat}}(D^{f_i})$ 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 $\mathcal{P}_{\text{flat}}(D, G)$.

Consider the group $G = \{f_4, f_5, f_6\}$ in our running example (Figure 4.1). $\mathcal{P}_{\text{flat}}(D, G)$ consists of the following elements:

- $g_1 = [f_4, f_5, f_6]$: 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.

- $g_2 = [f_4, f_5, f_6^2]$: This is the same as $g_1$ except that 2 is chosen from $f_6$'s multiplicity domain.

- $g_3 = [f_4, f_5]$: The two features $f_4$ and $f_5$ have been chosen, which satisfies the group multiplicity domain.

- $g_4 = [f_4, f_6]$: This is the same as $g_3$ except that $f_5$ is replaced by $f_6$ with 1 occurrence.

- $g_5 = [f_4, f_6^2]$: This is the same as $g_4$ except that 2 is chosen from $f_6$'s multiplicity domain.

- $g_6 = [f_5, f_6]$: This is the same as $g_4$ except that $f_4$ is replaced by $f_5$.

- $g_7 = [f_5, f_6^2]$: 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 $D = (F,r,↑,G,C)$, for any multiset $m$ over $F$: $m \in \mathcal{P}_{flat}(D)$ iff $m$ satisfies the following conditions:

(i) $m(r) = 1$,
(ii) $\forall f \in S \cap r_↓, \exists c \in C(f), \exists n \in \mathcal{P}_{flat}(D^f), \forall e \in \text{dom}(n) : m(e) = c \times n(e)$.
(iii) $\forall G \in \mathcal{G} \cap 2^{r_↓}, \exists n \in \mathcal{P}_{flat}(D,G), \forall e \in \text{dom}(n) : m(e) = n(e)$.

The following statement is a corollary of the above lemma.

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

(i) $\forall f \in S, \exists c \in C(f), \exists n \in \mathcal{P}_{flat}(D^f), \forall e \in \text{dom}(n) : m(e) = n(e) \times c \times m(f_↑)$.
(ii) $\forall G \in \mathcal{G}, \exists n \in \mathcal{P}_{flat}(D,G), \forall e \in \text{dom}(n) : m(e) = n(e) \times m(G_↑)$.

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 $D_1$ and $D_2$ in Figure 4.2. They are equivalent in the flat semantics, since they represent the same flat semantics $\{[f,f_2,f_3], [f,f_2^2,f_3^2]\}$.

### 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
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 $R \upharpoonright A$ is used to denote the restriction of $R$ to $A$.

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

---

Figure 4.2: Two different CFDs with the same flat semantics

Figure 4.3: Diagram induced by a node: an example
For an example, $D f^2$ in Figure 4.3 is the diagram induced by $f_2$ of the CFD $D$ 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 $D$ in Figure 4.1 in terms of induced diagrams: The root feature $f$ has two children labeled by $D f^1$ (diagram induced by $f_1$) and $D f^2$ (diagram induced by $f_2$) with multiplicity domains $\mathbb{N}\backslash \{3\}$ and $\{0,1,2\}$, respectively. In this way, a hierarchical product of $D$ is considered as a multiset $[f, h_1^c, h_2^c]$, where $h_1$ and $h_2$ are a hierarchical product of $D f^1$ and a hierarchical product of $D f^2$, respectively, and $c_1 \in \mathbb{N}\backslash \{3\}$ (the multiplicity domain of $f_1$), $c_2 \in \{0,1,2\}$ (the multiplicity domain of $f_2$). Since $D f^1$ is a singleton tree, $h_1$ is always equal to $[f_1]$.

Now, consider $D f^2$ (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 $D f^2$ would be a multiset $[f_2, h_3^c, h_G^c]$, where $h_3$ and $h_G$ are a hierarchical product of
D^f_3 and a grouped hierarchical product of G, respectively, and c_3 \in \{3, 4, 5\} (the multiplicity domain of f_3). Since D^f_3 is a singleton CFD, h_3 is always equal to \lceil 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 D^f_4 and a hierarchical product of D^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 D^f_4 and D^f_5 are singleton trees, h_4 and h_5 would be always \lceil f_4 \rceil and \lceil 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 \rceil, where h_6 is a hierarchical product of D^f_6 and c_6 \in \{1, 2\} (the multiplicity domain of f_6). h_6 would always be equal to \lceil 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 f, \lceil f_1 \rceil c_1, \lceil f_2, f_3 \rceil c_3, \lceil \lceil f_4 \rceil c_4 \times g_1, \lceil f_5 \rceil c_5 \times g_2, \lceil f_6 \rceil c_6 \times g_3 \rceil c_2 \rceil, \lceil \lceil f_5 \rceil c_5, \lceil f_6 \rceil c_6 \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 f, \lceil f_1 \rceil c_1, \lceil f_2, f_3 \rceil c_3, \lceil \lceil f_4 \rceil 0, \lceil f_5 \rceil c_5, \lceil f_6 \rceil c_6 \rceil c_2 \rceil = \lceil f, \lceil f_1 \rceil c_1, \lceil f_2, f_3 \rceil c_3, \lceil \lceil f_5 \rceil c_5, \lceil f_6 \rceil c_6 \rceil c_2 \rceil.

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

- h_1 = \lceil f, \lceil f_1 \rceil 5 \rceil: We chose the multiplicity 5 from the multiplicity domain of f_1. Since f_1 is a leaf node, D^f_1 represents a single hierarchical product \lceil 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 $h_1$.

$- h_2 = [f, [f_1]^5, [f_2, [f_3]^3, [[f_4], [f_5]]]]$: Unlike $h_1$, a hierarchical product of $D_f^2$ is included: $n = [f_2, [f_3]^3, [[f_4], [f_5]]]$. $D_f^2$ has a group $G$ and a solitary node $f_3$. The multiplicity 3 is chosen from the multiplicity domain of $f_3$, i.e., $[f_3]$ 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 $[[[f_4], [f_5]]]$.

$- h_3 = [f, [f_1]^5, [f_2, [f_3]^3, [[f_4], [f_5]]]^2]$: 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 $[f_2, [f_3]^3, [[f_4], [f_5]]]$ (a hierarchical product of $D_f^2$).

Consider again the CFDs $D_1$ and $D_2$ in Figure 4.2. Unlike their flat semantics, their hierarchical semantics capture the differences: $D_1$ contains the two hierarchical products $[f, [f_2, [f_3]]]$ and $[f, [f_2, [f_3]]^2]$, while $D_2$ contains $[f, [f_3, [f_2]]]$ and $[f, [f_3, [f_2]]^2]$ 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 urelement 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:

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

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

The rank of a multiset $m \in \mathcal{H}(A)$, denoted by $\text{rank}(m)$, 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 $m_1 = \lceil a^3, b^3 \rceil$, $m_2 = \lceil a^2, [a^2, b^3], [b]^4 \rceil$, and $m_3 = \lceil a^{10}, [a^2, b^3]^3, [b]^4, \lceil [a] \rceil \rceil$ in $\mathcal{H}(\{a, b\})$. We then would have: $\text{rank}(m_1) = 1$, $\text{rank}(m_2) = 2$, and $\text{rank}(m_3) = 3$ and $m_1 \uplus m_2 = \lceil a^5, b^3, [a^2, b^3], [b]^4 \rceil$.

Now, we are at the point where we can formalize hierarchical products of CFDs. Consider a CFD $D = (F, r, \uparrow, G, C) \in \mathcal{D}(F)$. Suppose that $r$ has $n$ solitary subfeatures $s_1, \ldots, s_n$ and $k$ groups $G_1, \ldots, G_k$. According to our informal description of hierarchical products, any multiset $m \in \mathcal{H}(F)$ is a hierarchical product of $D$ if its domain consists of (i) $r$ with 1 occurrence, (ii) a hierarchical product of $D^{s_i}$ (diagram induced by $s_i$) with a multiplicity $c_i \in C(s_i)$ 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 $D = (F, r, \uparrow, G, C)$, the set of $D$’s hierarchical products, denoted by $P(D)$, is defined as follows: For any multiset $m \in \mathcal{H}(F)$, $m \in P(D)$ iff it satisfies the following conditions:

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

(ii) $\forall f \in S \cap r_i, \exists c \in C(f), \exists n \in P(D^f) : m(n) = c$. 

72
∀ \( G \in G \cap 2^r \downarrow \), \( \exists n \in \mathcal{P}(D, G) : m(n) = 1. \)

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

\( \mathcal{P}(D) \) is called the hierarchical semantics of \( D \).

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 \( \forall c \in C : 1 \leq c \leq n \)). A hierarchical product of this group would be a multiset \( [f_1^{c_1} \times g_1, \ldots, f_n^{c_n} \times g_n] \), where \( c_i \) is a valid multiplicity for feature \( f_i \) and \( g_i \in \{0, 1\} \) (\( \forall 1 \leq i \leq n \)) such that \( g_1 + \ldots + g_n \in C \).

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

(i) \( \text{dom}(m) = \{m_i : g_i = 1\} \),
(ii) \( \forall 1 \leq i \leq k : m(m_i) = c_i \times g_i \),
(iii) \( g_1 + \ldots + g_k = c \).

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

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 \( D \) and \( D' \), \( (\mathcal{P}(D) = \mathcal{P}(D')) \iff (D = D'). \)
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 

\[ m = \lceil a, b, \lceil c, \lceil d, e \rceil \rceil \rceil \in H_3(\{a, b, c, d, e\}) \]

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 \( m \in H(A) \) for some \( A \), \( \text{MultIng}(m) \) is the smallest set of multisets in \( H(A) \) such that

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

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

The multiplicity of a multiset \( n \in \text{MultIng}(m) \) 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 \( m \in H(A) \) for a set \( A \) of urelements. The flat multiplicity of an element is defined by a function \( \#_{m, A} : A \to \mathbb{N} \) as

\[ #_{m, A}(a) = m(a) + \sum_{e \in \text{MultIng}(m)} #_{m, A}(e). \]

We define a function \( \text{flat}_A : H(A) \to H_1(A) \), which maps a given multiset \( m \in H(A) \) to a flat multiset as follows. For any \( m \in H(A) : \text{flat}_A(m)(a) = #_{m, A}(a) \). We say that \( \text{flat}_A(m) \) flattens \( m \).

Consider again the multiset \( m = \lceil a^2, b^2, [a, b]^4, [a^8, [a]^7, [a^5, b^3]^3] \rceil \). The flat multiplicities of \( a \) and \( b \) are 36 and 15, respectively. Thus, \( \text{flat}_{\{a,b\}}(m) = \lceil a^{36}, b^{15} \rceil \).
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 $h_1 = \lceil f, \lceil f_1 \rceil \rceil$, $h_2 = \lceil f, \lceil f_1 \rceil \lceil f_2 \rceil \rceil$, and $h_3 = \lceil f, \lceil f_1 \rceil \lceil f_2 \rceil \lceil f_3 \rceil \rceil$ of the CFD in Figure 4.1 (see page 70). Flattening them, we obtain $m_1 = \lceil f, f_1 \rceil$, $m_2 = \lceil f, f_1, f_2, f_3, f_4, f_5 \rceil$, and $m_3 = \lceil f, f_1, f_2, f_3, f_4, f_5 \rceil$, respectively, which are flat products of the CFD (see page 64).

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

### 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 $\mathcal{T}\mathcal{H}(A)$, is inductively defined as follows:

1. $[a] \in \mathcal{T}\mathcal{H}(A)$, $\forall a \in A$.
2. $t_1 \uplus [t_2^n] \in \mathcal{T}\mathcal{H}(A)$, $\forall t_1, t_2 \in \mathcal{T}\mathcal{H}(A), \forall n \in \mathbb{N}$, if
   \[
   \text{dom}(\text{flat}_A(t_1)) \cap \text{dom}(\text{flat}_A(t_2)) = \emptyset.
   \]
3. $t \uplus [\lceil t_{n_1}^{t_{n_k}} \rceil] \in \mathcal{T}\mathcal{H}(A)$, $\forall t, t_{n_1}, \ldots, t_{n_k} \in \mathcal{T}\mathcal{H}(A), \forall n_1, \ldots, n_k \in \mathbb{N}$, if
   \[
   \forall 1 \leq i \leq k : \text{dom}(\text{flat}_A(t)) \cap \text{dom}(\text{flat}_A(t_i)) = \emptyset, \text{ and }
   \forall 1 \leq i, j \leq k : (i \neq j) \implies (\text{dom}(\text{flat}_A(t_i)) \cap \text{dom}(\text{flat}_A(t_j)) = \emptyset).
   \]

For example, the following multisets in $\mathcal{H}([\{a, b, c\})$ are tree-like multisets:
- $t_1 = [a]$, $t_2 = [b]$, and $t_3 = [c]$,
- $t_4 = t_1 \uplus [t_2^2] = [a, [b]^6]$, $t_5 = t_3 \uplus [t_4] = [c, [a, [b]]^6]$,
- $t_6 = t_1 \uplus [[t_2^2, t_3^1]] = [a, [[b]^2, [c]]]$.

The following multisets are not valid tree-like multisets:
- $n_1 = [a, b]$,
- $n_2 = [a^3, [b]^6]$,
- $n_3 = [a, [[b, c]^2]]$.

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

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

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

The multiset $[[b]^2, [c, [a]^3]]$ 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 $\text{root} : \mathcal{T}H(A) \to A$, as follows:

(i) $\text{root}([a]) = a$, for any $a \in A$.

(ii) $\text{root}(t_1 \uplus [t_2^n]) = \text{root}(t_1)$ for any $t_1, t_2 \in \mathcal{T}H(A), n \in \mathbb{N}$ satisfying the conditions in Definition 4.11(ii).

(iii) $\text{root}(t \uplus [t_1^{n_1}, \ldots, t_k^{n_k}]) = \text{root}(t)$ for any $t, t_1, \ldots, t_k \in \mathcal{T}H(A)$ 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 \( t = \left[ a, \left[ b \right]^5, \left[ c, \left[ d \right]^3, \left[ e \right], \left[ f \right] \right]\right] \) is a tree-like multiset over the set \( \{a, b, c, d, e, f\} \): \( \text{root}(t) = a \); the elements \( t_1 = \left[ b \right], t_2 = \left[ c, \left[ d \right]^3, \left[ e \right], \left[ f \right] \right] \in \text{dom}(t) \) are both tree-like multisets with \( \text{root}(t_1) = b \) and \( \text{root}(t_2) = c \), respectively; the element \( \left[ e \right], \left[ f \right] \right] \in \text{dom}(t_2) \) 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 \( \mathcal{T}\mathcal{H}(A) \) and \( \mathcal{T}\mathcal{H}_i(A) \) \((i \geq 1)\), respectively. According to Definition 4.11, \( \mathcal{T}\mathcal{H}_1(A) = \{[a] : a \in A\} \) and \( \mathcal{T}\mathcal{H}(A) = \bigcup_i \mathcal{T}\mathcal{H}_i(A) \). Note that \( \mathcal{T}\mathcal{H}(A) \) 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: \( h_1 = \left[ f, \left[ f_1 \right]^5 \right], h_2 = \left[ f, \left[ f_1 \right]^5, \left[ f_2, \left[ f_3 \right]^3, \left[ f_4 \right], \left[ f_5 \right] \right] \right] \), and \( h_3 = \left[ f, \left[ f_1 \right]^5, \left[ f_2, \left[ f_3 \right]^3, \left[ f_4 \right], \left[ f_5 \right] \right] \right] \). 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 multiplicities from a given tree-like multiset.

We first define the notion of a *tree-like multiset 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 $a$. □

Remark 4.2. According to Definition 4.11, the following statement follows: Let $t \in T\mathcal{H}(A)$ for a set $A$ of urelements. For any $a \in \text{dom}(\text{flat}_A(t))$, 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 $t = [a, [b]^5, [c, [d]^3, [[e], [f]]]^2]$ over the set $\{a, b, c, d, e, f\}$. Then, we would have: $t^a = t, \ t^b = [b], \ t^c = [c, [d]^3, [[e], [f]]], \ t^d = [d], \ t^e = [e]$, and $t^f = [f]$.

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 $a \in \text{dom}(\text{flat}_A(t)) \setminus \{\text{root}(t)\}$, the parent of $a$, denoted by $a^{\uparrow t}$, is an element in $\text{dom}(\text{flat}_A(t))$ 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., $g \in \text{dom}(t^a^{\uparrow t})$.

(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., $t^a \in \text{dom}(t^{a^{\uparrow t}})$. □

Remark 4.3. According to Definition 4.11, the following statement follows obviously: Consider a tree-like multiset $t \in T\mathcal{H}(A)$ for a set $A$. For any $a \in \text{dom}(\text{flat}_A(t)) \setminus \{\text{root}(t)\}$, there exists a unique element in $\text{dom}(\text{flat}_A(t))$ satisfying (i) and (ii) in Definition 4.15. This makes Definition 4.15 well-formed. Therefore, $\_^{\uparrow t}$ is indeed a function from $\text{dom}(\text{flat}_A(t)) \setminus \{\text{root}(t)\}$ to $\text{dom}(\text{flat}_A(t))$. □
For an example, consider again the tree-like multiset \( t = \left[ a, [b]^5, c, [d]^3, [e], [f] \right]^2 \) over the set \{a, b, c, d, e, f\}. We would have: 
\[ b^t = c^t = a \quad \text{and} \quad d^t = e^t = f^t = c. \]

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 
\[ N_t = \text{dom}(\text{flat}_A(t)), r_t = \text{root}(t), \text{ and } _\uparrow^t : N_t \setminus \{r_t\} \rightarrow N_t \]
\( N_t \) is a function defined in Definition 4.15.

For an example, consider the tree-like multiset \( t = \left[ a, [b]^5, c, [d]^3, [e], [f] \right]^2 \in TH(\{a, b, c, d, e, f\}) \). 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 \( [[e], [f]] \) is an element in the domain of \( t^c = [c, [d]^3, [e], [f]] \)). 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 \( G \subset \text{dom}(\text{flat}_A(t)) \) is called a **group** if there exists a group tree-like multiset \( g \in \text{MultIng}(t) \) such that \( G = \{\text{root}(x) : x \in \text{dom}(g)\} \).
define $G^\uparrow = e^\uparrow$ for an element $e \in G$ and call it the parent of $G$.\footnote{Note that $\forall e, e' \in G : e^\uparrow = e'^\uparrow$.}

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., $\mathcal{G}(a) = \{ G \in \mathcal{G}_t : G^\uparrow = a \}$. \hfill \square

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

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 $t \in \mathcal{T}(A)$ over a set $A$, we define a function $C_t : \text{dom}(\text{flat}(t)) \setminus \{ \} \rightarrow \mathbb{N}$
\{root(t)\} \cup G_t \to \mathbb{N} as follows:

\[
C_t(e) = \begin{cases} 
|e| & \text{if } e \in G_t \\
\#_t(t^e) & \text{otherwise}
\end{cases}
\]

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 \(t = [a, [b]^5, [c, [d]^3, [[e], [f]]]^2]\).

It has only one associated group \(G = \{e, f\}\). According to Definition 4.18, \(C_t\) is defined on \(\{a, b, c, d, e, f, G\}\) as follows:

\[
\begin{align*}
C_t(b) &= \#_t(t^b) = \#_t([b]) = 5. \\
C_t(c) &= \#_t(t^c) = \#_t([c, [d]^3, [[e], [f]]]) = 2. \\
C_t(d) &= \#_t(t^d) = \#_t([d]) = 3. \\
C_t(e) &= \#_t(t^e) = \#_t([e]) = 1. \\
C_t(f) &= \#_t(t^f) = \#_t([f]) = 1. \\
C_t(G) &= \#_t(t^G) = |G| = 2.
\end{align*}
\]

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 \(D\) such that \(t \in \mathcal{P}(D)\) □

For an example, consider the tree-like multisets \(t = [a, [b]^5, [c, [d]^3, [[e], [f]]]^2]\) and \(t' = [a, [c, [[e]]], [g]^3]\). The CFDs \(D_t\) and \(D_{t'}\) in Figure 4.6 represents two CFDs whose hierarchical semantics include \(t\) and \(t'\), respectively.
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 $D$ such that $U \subseteq \mathcal{P}(D)$. We then call $D$ a representative CFD of $U$.
(ii) completely mergeable if there is a CFD $D$ such that $U = \mathcal{P}(D)$.

According to Theorem 4.4, any singleton set of tree-like multisets is mergeable. Consider the tree-like multisets $t = [a, [b]^5, [c, [d]^3, [e], [f]]^2]$ and $t' = [a, [c, [e]]]$, $[g]^3$. 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 $n = [a, [b]^3]$ and $n' = [b, [a]^2]$. 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 $D$ (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 $D$ 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 $D'$ of $U$, $|P(D)| \leq |P(D')|$. 

Let $D_{U_{\text{merge}}}$ denote the family of minimal representative CFDs of $U$.

The CFD $D$ in Figure 4.7 represents a minimal representative CFD of the tree-like multisets $t = [a, [b]^5, [c, [d]^3, [[e], [f]]]^2]$ and $t' = [a, [c, [[e]]], [g]^3]$. For these two tree-like multisets, there is, indeed, only one minimal representative CFD. Now, consider another tree-like multiset $t'' = [a, [c, [d]^3, [[e]]]^2]$. 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.
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.

**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 $\mathcal{T}$.

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.

![Figure 4.9: Megeable trees and their representative trees: an example](image)
Theorem 4.5. Consider an enumerable set of tree-like multisets $U = \{t_i : i \in I\} \subset \mathcal{T}(A)$ over a set $A$, where $I$ enumerates its elements. Let $T_i = (N_i, r_i, \_^i)$ and $G_i$ ($\forall i \in I$) 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) $\forall i, j \in I, \forall n \in N_i \cap N_j : (\exists G \in G_i : n \in G) \implies (\exists G \in G_j : n \in G)$.

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 $f_1$ 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 $\{a, [b]^5, [c, [d]^3, [[e, [f]]]2\}$ would be $\{a, [b], [c, [d], [[e, [f]]]]\}$.

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

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

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

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

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

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

The following proposition follows easily.
Proposition 4.1. Let $T_t = (N_t, r_t, \uparrow_t, G_t, C_t)$ 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 $t^\circ$ are equal to $T_t^\circ = T_t$ and $G_t^\circ = G_t$, respectively. The multiplicities associated with $t^\circ$, i.e., $C_t^\circ$, is defined as follows:

$$C_t^\circ(e) = \begin{cases} 
\{1\} & \text{if } e \in (N_t \setminus \{r_t\}) \\
C_t(e) & \text{if } G \in G 
\end{cases}$$

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 $U^\circ$ is.

(ii) $U$ is mergeable implies that $U^\circ$ 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:

$t_1 = [a, [b]^5, [[c]]],$
$t_2 = [a, [b]^5, [[d]^2]],$
$t_3 = [a, [b]^2, [[c]]],$
$t_4 = [a, [b]^2, [[d]^3]].$
Their relaxed multisets are represented in the following (as usual, we denote the set of the following tree-like multisets by $U^\circ$):

$$t_1^\circ = t_3^\circ = [a, [b], [[c]]],$$
$$t_2^\circ = t_4^\circ = [a, [b], [[d]]].$$

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

\[\text{Figure 4.10: Minimal representative CFDs of } U \text{ and } U^\circ\]

\[\text{Figure 4.11: Minimal representative CFDs of } U_1 \text{ and } U_1^\circ\]

\[\text{Since } U \text{ is completely mergeable, there is only one minimal representative CFD of } U.\]
Now, consider \( U_1 = U \setminus \{t_2, t_4\} \). Clearly, \( U_1 \) is not completely mergeable. A minimal representative CFD \( D_1 \) 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 \( D_1^\circ \) 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 \( U, U^\circ \) 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 \( D \) 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 \( U^\circ \) 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 \( U \subset \mathcal{T}\mathcal{H}(A) \), we define a function \( C_U : A \rightarrow 2^\mathbb{N} \) as follows: \( C_U(a) = \bigcup_{t \in U} \{#_t(t^a)\} \). \( \square \)

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

(i) \( U^\circ \) is completely mergeable, and

(ii) \( \forall t \in U^\circ, \forall a \in \text{dom}(\text{flat}_A(t)), \forall c \in C_U(a), \exists t' \in U : (t'^\circ = t) \land (#_{t'}(t'^a) = c) \).

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

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

\[ 7 \text{Recall that } t^a \text{ and } #_t(t^a) \text{ denote the multiset induced by } a \text{ and the multiplicity of } t^a \text{ in } t, \text{ resp.} \]
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 $D_1$ and $D_2$. We need to address the two following questions: Are $D_1$ and $D_2$ mergeable? What would be the result of their merge, if they are mergeable? To address these questions, we first obtain their hierarchical semantics $P(D_1)$ and $P(D_2)$, 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 $P(D_1) \cup P(D_2)$. 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 $D_1$ and $D_2$, 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 (⌈ and ⌉). 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 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 \(l \leq u\). 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 \(f, f_{1\ldots6}\). \(G\) denotes a group consisting of the features \(f_4, f_5,\) and \(f_6\), and any feature in \(F \setminus G\) 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 \(f_4, f_5\) 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
a multiplicity domain as a subset of natural numbers to make the formalizations in the chapter easier to read.\footnote{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.}

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 \(\mathbb{N}^* = \mathbb{N} \cup \{\ast\}\) the universe, and \(\leq * \subseteq \mathbb{N}^* \times \mathbb{N}^*\) a reflexive transitive relation defined by \(\forall u, l \in \mathbb{N}^* : (l \leq_* u) \iff (l, u \in \mathbb{N} \land l \leq u) \lor (u = \ast)\).\footnote{Another equivalent definition could be \(\forall l, u \in \mathbb{N} : (l \leq_* u \iff l \leq u) \land (l \leq_* \ast)\)} For any \(u, l \in \mathbb{N}^*\), we use the notation \(l <_* u\) to denote \((l \leq_* u) \land (l \neq u)\).

\[
\text{Definition 5.2 (Multiplicities).}
\]

(i) The *multiplicity-set* is the set \(\mathcal{C} = \{(l, u) \in \mathbb{N} \times \mathbb{N}^* : (l \leq_* u) \land (u \neq 0)\}\).

An element \(c = (l, u) \in \mathcal{C}\) is called a *multiplicity*. We call \(l\) and \(u\) the *lower-bound*, denoted by \(\text{low}(c)\), and *upper-bound*, denoted by \(\text{up}(c)\), of \(c\), respectively.

Figure 5.1: A CFD: running example for transformations
(iii) A subset $C \subseteq \mathcal{C}$ 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 $\text{low}(C)$, and upper-bound, denoted by $\text{up}(C)$, of $C$, respectively.

\begin{definition}[Cardinality-based Feature Diagrams] A cardinality-based feature diagram (CFD) is a 3-tuple $D = (T, \mathcal{G}, \mathcal{C})$ 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) $\mathcal{G} \subseteq 2^{F \setminus r}$ 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., $\forall G, G' \in \mathcal{G}, (G \neq G') \Rightarrow (G \cap G' = \emptyset)$. The nodes that are not in a group are called solitary nodes. Let $\mathcal{S}$ denote the solitary nodes, i.e., $\mathcal{S} = F \setminus F_r \cup \bigcup_{G \in \mathcal{G}} G$.

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

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.

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

Let $D = (T, \mathcal{G}, \mathcal{C})$ be a CFD with $T = (F, r, \uparrow)$ and $f \in F$. We will need the following notations in later sections:
– \text{depth}(\text{D}) \text{ denotes } T \text{’s depth and } \text{depth}(f) \text{ denotes the } f \text{’s depth in } T. \text{ In our example in Figure 5.1, } \text{depth}(f) = 1, \text{depth}(f_1) = 3, \text{ and } \text{depth}(\text{D}) = 3.

– \text{lev}(\text{D}) \text{ denotes the set of leaf nodes, i.e., } \text{lev}(\text{D}) = \{ f \in F : f_↓ = \emptyset \}. \text{ In Figure 5.1, } \text{lev}(\text{D}) = \{ f_1, f_3, f_4, f_5, f_6 \}.

– \text{glev}(\text{D}) \text{ denotes the set of grouped leaves, i.e., } \text{glev}(\text{D}) = \{ G \in \mathcal{G} : \forall n \in G. n_↓ = \emptyset \}. \text{ In Figure 5.1, } \text{glev}(\text{D}) = \{ \{ f_4, f_5, 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 \( \Sigma \) is a 3-tuple \( \text{RD} = (\text{LT}_{\text{re}}, \mathcal{G}, \mathcal{C}) \) of the following components:

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

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

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

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

If needed, we will subscript \( \text{RD} \)’s components with index \( \text{RD} \), e.g., write \( \mathcal{G}_{\text{RD}} \).
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 $depth(RD)$, $lev(RD)$, $glev(RD)$, $cplev(RD)$. Please see page 97 for their meanings.

We will also need the following operations.

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

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

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.
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 \(f_1\) (Figure 5.1) would be

\[ f_1 + f_1^2 + f_1^4f_1^* . \]

This regular expression represents the multiplicity constraint on this node properly, as it says that the number of occurrences of the feature \(f_1\) 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.
\[ r_1 = f_1 + f_1^2 + f_1^4 f_1^*, \]
\[ r_3 = f_3^3 + f_3^4 + f_3^5, \]
\[ r_4 = f_2, \]
\[ r_5 = f_4, \]
\[ r_6 = f_1 + f_1^2. \]

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

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

\[
\text{lex}_{\mathcal{RD}}(n) = r_1 + \ldots + r_j, \quad \text{where}
\]
\[
r_i = \begin{cases} 
  l_{re}(n)^{l_i} + \ldots + l_{re}(n)^{u_i} & \text{if } u_i \neq * \\
  l_{re}(n)^{l_i} (l_{re}(n))^* & \text{otherwise}
\end{cases}
\]

**Definition 5.6 (CRE-EML Stage).** We define a function \( \text{mel}^{\text{CRE}} : \mathcal{RD}(\Sigma) \to \mathcal{RD}(\Sigma) \), called CRE-EML function, as follows. For a given CRD \( \mathcal{RD} = (\mathcal{LT}_{re}, \mathcal{G}, \mathcal{C}) \)

\[ ^5 \text{We use the superscript CRE for an operation in this section to distinguish it from the operation with similar functionality in the next section. We will use the superscript ORE for the operations in the following sections, e.g., mel^{ORE}.} \]
with $LT_{re} = (N, r, \uparrow, \Sigma, l_{re})$,

$$me^{CRE}(RD) = (LT'_{re}, G', C'),$$

where

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

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

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

\[\square\]

### 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 $G = \{f_4, f_5, f_6\}$ 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 \( \text{Per}^{(l,u)}(X) \) 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, \( \text{Per}^{(1,2)}(\{r_1, r_2, r_3\}) \) would be the following set of expressions: \( \{r_1, r_2, r_3\} \cup \{r_1r_2, r_2r_1, r_1r_3, r_2r_3, r_3r_2\} \). We consider \( \text{Per}^{(l,u)}(\emptyset) = \epsilon \) for any \( l \) and \( u \). By \( \text{Per}^{(k,k)}(X) \) (for any \( k \in \mathbb{N} \)), we mean \( \text{Per}^{(k,k)}(X) \).

We will need the following definitions to formalize the stage \( \text{CRE-EGL}. \)

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

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

\[
r_i = + \ X_i,
\]

\[
X_i = \text{Per}^{(l_i, u_i)}(\{l_{re}(n) : n \in G\}).
\]
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 $G \in \text{glev}(\mathbf{RD})$, a node identifier $n_G$ is assigned. Let $N_G$ denote the set of these node identifiers. In other words, we have a bijection $\text{gid} : N_G \rightarrow \text{glev}(\mathbf{RD})$ which assigns each grouped node in $\text{glev}(\mathbf{RD})$ to a unique node identifier in $N_G$.

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

$$\text{gle}^{\text{CRE}}(\mathbf{RD}) = (LT_{re}', \mathcal{G}', \mathcal{C}')$$

where

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

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

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

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

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

$$r_2 = f_2(r_3 r_G + r_G r_3).$$

We formalize this stage via the following definitions.

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

\[
\text{pex}_{\text{RD}}^{\text{CRE}}(n) = \text{l}_{re}(n) (\text{+ Per}_{\{n\}}(E)), \quad \text{where}
\]

\[
E = \{\text{l}_{re}(n') : n' \in n\_n\}.
\]

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

\[
\text{Definition 5.11 (CRE-DR Stage).} \quad \text{The function } \text{dre}_\text{CRE} : \mathcal{RD}(\Sigma) \rightarrow \mathcal{RD}(\Sigma) \text{ is called CRE-DR function. For a given CRD } \text{RD} = (LT_{re}, \mathcal{G}, \mathcal{C}) \text{ with } LT_{re} = (N, r, \_\uparrow, \Sigma, l_{re}), \text{ \( \text{dre}_{\text{CRE}}(\text{RD}) \) is defined as follows:}
\]

\[
\text{dre}_{\text{CRE}}(\text{RD}) = (LT'_{re}, \mathcal{G}, \mathcal{C}')
\]

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

\[
N' = N - cplev(\text{RD})
\]

\[
\_\uparrow' = \_\uparrow |_{N'}
\]

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

\[
l'_{re}(n) = \begin{cases} 
\text{pex}_{\text{RD}}^{\text{CRE}}(n) & \text{if } n \in plev(\text{RD}) \\
\text{l}_{re}(n) & \text{otherwise}
\end{cases}
\]

106
5.2.4 CRE-Shrinking Step and CRE

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

**Definition 5.12 (CRE-Shrinking Step).** The function \( \text{shr}^{\text{CRE}} : \mathcal{R}(\Sigma) \rightarrow \mathcal{R}(\Sigma) \) is called CRE-shrinking function and is defined as \( \text{shr}^{\text{CRE}} = \text{dre}^{\text{CRE}} \circ \text{gle}^{\text{CRE}} \circ \text{mel}^{\text{CRE}} \).

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

\[
r = f(r_1r'_2 + r_2r_1), \text{ where }
\]

\[
r'_2 = \varepsilon + r_2 + r_2^2.
\]

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

Now we are at the point where we can prove that the CRE regular expression interpretation of a given CFD \( \mathcal{D} \) provides a faithful semantics for \( \mathcal{D} \).

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

**Definition 5.13 (Preserving the Hierarchy).** Consider a CFD \( \mathcal{D} = (T, \mathcal{G}, \mathcal{C}) \) 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 \(D\) if \(\forall f, f' \in F : (f' \in f_{\Downarrow}) \iff (\forall w \in L : (f' \in U_w) \Rightarrow (f \sqsubseteq_w f'))\).

**Theorem 5.2.** For a given CFD \(D\), \(L(R^{\text{CRE}}(D))\) preserves \(D\)’s hierarchy.

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 \(\text{OD} = (T, G, C, \leq_{\text{sib}})\) of the following components:

(i) \(T = (F, r, \uparrow)\) is a tree, as defined in Definition 3.1.

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

(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) \(\forall S \in \text{Sib} : \leq_{\text{sib}} \mid_S\) (the restriction of \(\leq_{\text{sib}}\) on \(S\)) is a total ordering, where \(\text{Sib} = \{f_{\Downarrow} \subseteq F : f \in F\}\).

(iii-iii) \(\forall S \in \text{Sib}, \forall G \subseteq S, \forall f \in S \setminus G : G \in G \implies (f \leq_{\text{sib}} \inf(\leq_{\text{sib}} \mid_G)) \lor (\sup(\leq_{\text{sib}} \mid_G) \leq_{\text{sib}} f)\).

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

\(^7\text{See page 24 for the definition of } \sqsubseteq_w\).
If needed, we will subscript OD's components with index \( OD \), e.g., write \( \leq_{\text{sib}OD} \).

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 \( \{(f_1, f_2), (f_3, f_4), (f_4, f_5), (f_5, f_6)\} \).\(^8\) We will use this osCFD as a running example to illustrate the transformation procedure.

Let \( \mathcal{OD}(D) \) 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 \( \text{ORD} = (LT_{re}, G, C, \leq_{\text{sib}}) \) 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) \( (LT_{re}, G, C) \) is a CRD, as defined in Definition 5.4. It is called the ORD’s underlying CRD, denoted by \( \text{ORD}^{\text{crd}} \).

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

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

If needed, we will subscript ORD’s components with index \( ORD \), e.g., write \( \leq_{\text{sib}ORD} \).

Obviously, osCRDs subsume osCFDs. Any notation used for CFDs and CRDs also works for osCFDs and osCRDs, respectively. These notations include \( \text{depth}(ORD) \), \( \text{lev}(ORD) \), \( \text{glev}(ORD) \), \( \text{plev}(ORD) \), \( \text{cplev}(ORD) \). 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 \( \text{mel}^{\text{ORE}} : \text{ORD}(\Sigma) \rightarrow \text{ORD}(\Sigma) \), called ORE-EML function. For a given osCRD \( \text{ORD} \), \( \text{mel}^{\text{ORE}}(\text{ORD}) \) is an osCRD \( \text{ORD}' \) where \( \text{ORD}'^{\text{crd}} = \text{mel}^{\text{CRE}}(\text{ORD}^{\text{crd}}) \) and \( \leq_{\text{sib} \text{ORD}'} = \leq_{\text{sib} \text{ORD}} \) (see Definition 5.6 for the definition of \( \text{mel}^{\text{CRE}} \)).

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.

---

9Recall 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 $G = \{r_4, r_5, r_6\}$ in Figure 5.2(a). Its corresponding ORE regular expression would be equal to $r_G = r_G' + r_G''$ where

$$r_G' = r_4 r_5 + r_5 r_6 + r_4 r_6,$$

$$r_G'' = r_3 r_5 r_6.$$

$r_G'$ 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 $r_4 r_5 + r_5 r_4 + r_5 r_6 + r_6 r_5 + r_4 r_6 + r_6 r_4$ and $r_4 r_5 r_6 + r_4 r_6 r_5 + r_5 r_4 r_6 + r_5 r_6 r_4 + r_6 r_4 r_5 + r_6 r_5 r_4$, 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, 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 \(Per_{\leq}^{(l,u)}(\emptyset) = \epsilon\) for any \(l\) and \(u\).\(^{10}\) By \(Per_{\leq}^{k}(X)\) (for any \(k \in \mathbb{N}\)), we mean \(Per_{\leq}^{k}(X)\).

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

\[
gex_{\text{ORE-ORD}}(G) = r_1 + \ldots + r_j, \text{ where for all } 1 \leq i \leq j:
\]

\[
r_i = + X_i,
\]

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

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).** \(gle_{\text{ORE}} : \mathcal{OD}(\Sigma) \rightarrow ORD(\Sigma)\) is a total function called ORE-EGL function. For a given osCRD \(ORD = (LT_{re}, \mathcal{G}, \mathcal{C}, \leq_{sib})\)

\(^{10}\)Clearly, \(\leq\) would be also \(\emptyset\) when its domain is empty.
with $LT_{\leq} = (N, r, \uparrow, \Sigma, l_{re})$, $\text{gle}^{\text{ORE}}(\text{ORD})$ is defined as follows:

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

where

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

$$N' = (N - \text{glev}(\text{ORD})) \cup N_G$$

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

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

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

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

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

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

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

□

113
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 \(r_2 = f_2(r_3r_G)\). Consider \(r_2\)’s CRE version \(f_2(r_3r_G + r_Gr_3)\) 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 \(ORD = (LT_{re}, G, C, \leq_{\text{sib}})\) with \(LT_{re\leq} = (N, r, \uparrow, \Sigma, l_{re})\), we define a total function \(pex_{ORD} : plev(ORD) \rightarrow \text{RE}(\Sigma)\), which maps a parent whose child nodes are leaves to a regular expression. For a given node \(n \in plev(ORD)\), \(pex_{ORD}(n)\) is defined as follows:

\[
pex_{ORD}(n) = l_{re}(n)(\sum Pe^{l_{re}(n)}_{\leq_{\text{sib}}}(E)), \quad \text{where}\]

\[
E = \{l_{re}(n') : n' \in n\}.
\]

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

\[
\]
≤_{\text{sib}}) with \( LT_{re} \leq (N, r, \uparrow, \Sigma, l_{re}) \), \( \text{dre}^{\text{ORE}}(\text{ORD}) \) is defined as follows:

\[
\text{dre}^{\text{ORE}}(\text{ORD}) = (LT'_{re}, G, C', \leq'_{\text{sib}})
\]

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

\[
N' = N - cplev(\text{ORD})
\]

\[
\uparrow' = \uparrow|_{N'}
\]

\[
\leq'_{\text{sib}} = \leq_{\text{sib}}|_{N'}
\]

\[
C' = C|_{N' \cup G}
\]

\[
l'_{re}(n) = \begin{cases} 
\text{pex}^{\text{ORE}}_{\text{ORD}}(n) & \text{if } n \in \text{plev}(\text{ORD}) \\
\text{l}_{re}(n) & \text{otherwise}
\end{cases}
\]

\[\square\]

### 5.3.4 ORE-Shrinking Step and ORE

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

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

\[\square\]

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

\[ r = f(r_1r'_2), \text{ where} \]

\[ r'_2 = \varepsilon + r_2 + r_2^2. \]

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

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

**Theorem 5.3.** For any given osCFD \text{OD}:

(i) \( \mathcal{L}(\mathcal{R}^\text{ORE}(\text{OD})) \subseteq \mathcal{L}(\mathcal{R}^\text{CRE}(\text{OD}_\text{cfd})) \)

(ii) \( \text{Par}(\mathcal{L}(\mathcal{R}^\text{ORE}(\text{OD}))) = \text{Par}(\mathcal{L}(\mathcal{R}^\text{CRE}(\text{OD}_\text{cfd}))) \)

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

**Corollary 5.1.** For a given osCFD \text{OD}, \( \text{Par}(\mathcal{L}(\mathcal{R}^\text{ORE}(\text{OD}))) = \mathcal{P}^\text{flat}(\text{OD}_\text{cfd}) \).

### 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 ‘[’ and ‘]’. This
differentiates \textbf{HRE} from \textbf{ORE} and \textbf{CRE}, as \textbf{ORE} and \textbf{CRE} regular expressions are built over features. The \textbf{HRE} transformation is based on the hierarchical semantics given in Chapter 4. The \textbf{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 \( \textbf{D} \) in Figure 5.1 considering a sibling ordering \( f_1 \leq_{\text{sib}} f_2, f_3 \leq_{\text{sib}} f_4 \leq_{\text{sib}} f_5 \leq_{\text{sib}} f_6 \). Let \( \textbf{OD} \) denote this osCFD.

Consider the hierarchical product \( h_1 = [f, [f_1]^5] \) of \( \textbf{D} \) (see page 70). Considering the subfeature ordering, this tree-like multiset can be seen as a sequence of symbols \( w_1 = [f[f_1][f_1][f_1][f_1][f_1]] \). Another example: consider the hierarchical product \( h_2 = [f, [f_1]^5, [f_2, [f_3]^3, [[f_4], [f_5]]]] \). Considering the sibling ordering, its corresponding sequence would be \( w_2 = [f[f_1][f_1][f_1][f_1][f_1][f_2][f_3][f_3][f_3][f_4][f_5]][f_5] \). \textbf{HRE} transforms the osCFD to a regular expression whose language is the set of all such sequences of symbols. We call this expression the \( \textbf{OD} \)'s \textbf{HRE} regular expression, denoted by \( \mathcal{R}^{\text{HRE}}(\textbf{OD}) \).

The \textbf{HRE} transformation is a top-down procedure. This is another difference between this approach and the others, \textbf{CRE} and \textbf{ORE}. We define a regular expression for each group and each node. The expressions defined for nodes and groups are called \textbf{HRE} node and group expressions, respectively. We use the notation \( \mathcal{R}^{\text{HRE}}(f) \) and \( \mathcal{R}^{\text{HRE}}(G) \) to denote the regular expression associated with a node \( n \) and a group \( G \), respectively. The regular expression \( \mathcal{R}^{\text{HRE}}(r) = \mathcal{R}^{\text{HRE}}(\textbf{OD}) \), 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: \( \mathcal{R}^{\text{HRE}}(f), \mathcal{R}^{\text{HRE}}(f_1), \mathcal{R}^{\text{HRE}}(f_2), \mathcal{R}^{\text{HRE}}(f_3), \)
\( R^{\text{HRE}}(f_4), R^{\text{HRE}}(f_5), R^{\text{HRE}}(f_6) \). We also have a group expression \( R^{\text{HRE}}(G) \).

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

\[
R^{\text{HRE}}(\text{OD}) = \lceil f \ R^{\text{HRE}}(f_1) R^{\text{HRE}}(f_2) \rceil,
\]

where \( R^{\text{HRE}}(f_1) \) and \( R^{\text{HRE}}(f_2) \) denote the HRE regular expressions corresponding to the f’s subfeatures \( f_1 \) and \( f_2 \), respectively. Note that \( f \) precedes \( R^{\text{HRE}}(f_1) \) and \( R^{\text{HRE}}(f_2) \). 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 \( f_1 \leq_{\text{sib}} f_2 \), \( R^{\text{HRE}}(f_1) \) precedes \( R^{\text{HRE}}(f_2) \).

\[
R^{\text{HRE}}(f_1) = R^{\text{HRE}}(\text{OD}^{f_1}) + (R^{\text{HRE}}(\text{OD}^{f_1}))^2 + (R^{\text{HRE}}(\text{OD}^{f_1}))^4 (R^{\text{HRE}}(\text{OD}^{f_1}))^* 
\]

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

\[
R^{\text{HRE}}(f_2) = \varepsilon + R^{\text{HRE}}(\text{OD}^{f_2}) + (R^{\text{HRE}}(\text{OD}^{f_2}))^2
\]

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

¹²Note that ‘(‘ and ‘)’ are not in our alphabet. They are used to group expressions in regular expressions, while ‘[‘ and ‘]’ are symbols of the alphabet.
$\mathcal{R}_{\text{HRE}}(\text{OD}^{f_2})$ denotes the HRE regular expression corresponding to the diagram induced by $f_2$. $\mathcal{R}_{\text{HRE}}(f_2)$ models the multiplicity domain $(0, 2)$ on $f_2$ properly.\(^{13}\)

$$\mathcal{R}_{\text{HRE}}(\text{OD}^{f_2}) = [ f_2 \mathcal{R}_{\text{HRE}}(f_3) \mathcal{R}_{\text{HRE}}(G) ]$$

In the above expression, $\mathcal{R}_{\text{HRE}}(f_3)$ and $\mathcal{R}_{\text{HRE}}(G)$ denote the HRE regular expression corresponding to $f_3$ and the group $G = \{f_4, f_5, f_6\}$, respectively. Since $f_2$ is the root feature of $\text{OD}^{f_2}$ (the diagram induced by $f_2$), it must precede any other expressions corresponding to its subfeatures. Note that $\mathcal{R}_{\text{HRE}}(f_3)$ precedes $\mathcal{R}_{\text{HRE}}(G)$ due to the sibling ordering.\(^{14}\)$\(^{14}\) $\mathcal{R}_{\text{HRE}}(f_3)$ and $\mathcal{R}_{\text{HRE}}(G)$ are defined in the following:

$$\mathcal{R}_{\text{HRE}}(f_3) = (\mathcal{R}_{\text{HRE}}(\text{OD}^{f_3}))^3 + (\mathcal{R}_{\text{HRE}}(\text{OD}^{f_3}))^4 + (\mathcal{R}_{\text{HRE}}(\text{OD}^{f_3}))^5$$

$\mathcal{R}_{\text{HRE}}(\text{OD}^{f_3})$ denotes the HRE expression corresponding to the diagram induced by $f_3$, which is equal to $[ f_3 ]$. Note that the choices between different iterations of $\mathcal{R}_{\text{HRE}}(\text{OD}^{f_3})$ together model the multiplicity domain $(3, 5)$ on $f_3$.

$$\mathcal{R}_{\text{HRE}}(G) = [(G_2 + G_3)]$$

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 $\mathcal{R}_{\text{HRE}}(G)$. These two multiplicities

\(^{13}\)(\mathcal{R}_{\text{HRE}}(\text{OD}^{f_2}))^0 = \varepsilon.\(^{14}\) $f_3 \preceq_{\text{sh}} \text{inf}(G)$.\(^{14}\)
are modelled, respectively, by the expressions $G_2$ and $G_3$:

\[
G_2 = \mathcal{R}^{HRE}(f_4) \mathcal{R}^{HRE}(f_5) + \mathcal{R}^{HRE}(f_4) \mathcal{R}^{HRE}(f_6) + \mathcal{R}^{HRE}(f_5) \mathcal{R}^{HRE}(f_6)
\]

\[
\mathcal{R}^{HRE}(f_4) = \mathcal{R}^{HRE}(OD_{f_4})
\]

\[
\mathcal{R}^{HRE}(f_5) = \mathcal{R}^{HRE}(OD_{f_5})
\]

\[
\mathcal{R}^{HRE}(f_6) = \mathcal{R}^{HRE}(OD_{f_6}) + (\mathcal{R}^{HRE}(OD_{f_6}))^2
\]

There are three choices for the group multiplicity 2: choosing either “$f_4, f_5$”, “$f_4, f_6$”, or “$f_5, f_6$”. In the building of the expression $G_2$, we also consider the ordering $f_4 \leq_{sib} f_5 \leq_{sib} f_6$. Due to the multiplicity domain $(1, 2)$ on the feature $f_6$, there are two different choices $\mathcal{R}^{HRE}(OD_{f_6})$ and $(\mathcal{R}^{HRE}(OD_{f_6}))^2$ for the $f_6$’s HRE regular expression. $\mathcal{R}^{HRE}(OD_{f_6})$, $\mathcal{R}^{HRE}(OD_{f_6})$, and $\mathcal{R}^{HRE}(OD_{f_6})$ are, respectively, equal to $[f_4]$, $[f_5]$, and $[f_6]$, as they are all leaves.

\[
G_3 = \mathcal{R}^{HRE}(f_4) \mathcal{R}^{HRE}(f_5) \mathcal{R}^{HRE}(f_6)
\]

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 $w_1 = [f[f_1]^5]$ and $w_2 = [f[f_1]^5[f_2[f_3]^3[f_4][f_5]]]$ (see page 117) are in the language of $\mathcal{R}^{HRE}(OD)$.

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 $OD = (T, \mathcal{G}, C, \leq_{sib})$ be an osCFD with $T = (F, r, \uparrow)$ and $f \in F$. We define a total order $\leq_{sib} f \subseteq \text{Ing}(f_1) \times \text{Ing}(f_1)$, where $\text{Ing}(f_1) \overset{\text{def}}{=} (S \cap f_1) \cup (\mathcal{G} \cap 2^{f_1})$, where
as the smallest transitive relation satisfying the following conditions:

(i) \( \forall e_1, e_2 \in S \cap f_↓ : (e_1 \leq^f_{sib} e_2) \iff (e_1 \leq_{sib} e_2) \).

(ii) \( \forall e \in S \cap f_↓, \forall G \in G \cap 2^f_↓ : (e \leq^f_{sib} G) \iff (e \leq_{sib} \inf(G)) \).

(ii) \( \forall e \in S \cap f_↓, \forall G \in G \cap 2^f_↓ : (G \leq^f_{sib} e) \iff (\sup(G) \leq_{sib} e) \).

In our running example, we would have \( f_1 \leq^f_{sib} f_2, \quad f_3 \leq^f_{sib} G \).

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

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

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

See Definition 5.25 for \( \mathcal{R}^\text{HRE}(OD^f) \).

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

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

\[
r_i = + X_i
\]

\[
X_i = \text{Per}^{(l_i, u_i)}_{\leq_{sib}}(E) \text{ with } E = \{\mathcal{R}^\text{HRE}(n) : n \in G\}
\]
The following definition shows how to get the HRE regular expression for a given osCFD.

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

$$\mathcal{R}^{\text{HRE}}(OD) = \lceil r \cdot \frac{\text{Per}_{\leq_{\text{sib}}} | \{ e : e \in \text{Ing}(r_{\downarrow}) \} }{r} \rceil,$$

where $E = \{ \mathcal{R}^{\text{HRE}}(e) : e \in \text{Ing}(r_{\downarrow}) \}$.

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

**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, $\frac{\text{Per}_{\leq_{\text{sib}}} | \{ e : e \in \text{Ing}(r_{\downarrow}) \} }{r}$ would be $\varepsilon$, which implies $\mathcal{R}^{\text{HRE}}(OD) = \lceil r \rceil$.

### 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.
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 $D_1$ and $D_2$ in Figure 5.3. Their flat semantics are the same. However, their hierarchical structures are different. There are two osCFDs having $D_1$ as their underlying CFD. Let us pick the one in which the sibling ordering is $b \leq_{\text{sib}} c$ and denote it by $OD_1$. There is only one osCFD whose underlying CFD is $D_2$ (The sibling ordering in this osCFD would be empty.). Let us denote this osCFD by $OD_2$. According to the procedures described for ORE and CRE, we would have:

$$R_{\text{CRE}}(D_1) = a(bc^* + c^*b)$$

$$R_{\text{CRE}}(D_2) = R_{\text{ORE}}(OD_1) = R_{\text{ORE}}(OD_2) = abc^*$$

As we see above, the CRE expressions of $D_1$ and $D_2$ distinguish between them, as expected. However, the ORE expressions of $OD_1$ and $OD_2$ 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 $D$: $\sum_{OD \in \mathcal{OD}(D)} R_{\text{ORE}}(OD) = R_{\text{CRE}}(D)$. □

As an example, consider again the CFD $D_1$ in Figure 5.3. As already mentioned, $\mathcal{OD}(D_1)$ includes two osCFDs: $OD_1$ with the sibling ordering $b \leq_{\text{sib}} c$ and $OD'_1$ with the sibling ordering $c \leq_{\text{sib}} b$. Composing $R_{\text{ORE}}(OD'_1) = ac^+b$ and $R_{\text{ORE}}(OD_1) = abc^+$ via the choice operation $+$, we get to $R_{\text{CRE}}(D_1) = a(bc^+ + c^+b)$.

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 $D_1$ 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:

$$L(R_{\text{CRE}}(D_1)) = \{ ab, ac \},$$

$$L(R_{\text{HRE}}(D_1)) = \{ [a [[b]]], [a [[c]]] \}.$$  

We see that one can recognize the feature $b$ ($c$, respectively) from the element $[a, [[b]]]$ ($[a, [[c]]]$, 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,
it would be the best candidate for reverse engineering of CFMs. As a simple example, consider the CFDs $D_1$ and $D_2$ in Figure 5.4 (ignore the exclusive constraint between $b$ and $c$ in $D_2$ for a while). $D_2$’s HRE and CRE languages are as follows (let us suppose a left-to-right ordering on siblings as the sibling ordering on $D_2$):

\[
\mathcal{L}(\mathcal{R}^{CRE}(D_2)) = \{ \text{abc, acb} \},
\]

\[
\mathcal{L}(\mathcal{R}^{HRE}(D_2)) = \{ \lceil a \rceil \lceil b \rceil \lceil c \rceil \}. 
\]

We see that both the CRE and HRE languages distinguish between $D_1$ and $D_2$.

Now, consider an exclusive constraint between $b$ and $c$ on $D_2$, as shown in Figure 5.4. Let $M$ denote this CFM. $M$’s CRE and HRE languages are as follows: (Let $\mathcal{L}^{CRE}(M)$ and $\mathcal{L}^{HRE}(M)$ denote their corresponding languages, respectively.)\(^{15}\)

\[
\mathcal{L}^{CRE}(M) = \{ \text{ab, ac} \},
\]

\[
\mathcal{L}^{HRE}(M) = \{ \lceil a \rceil \lceil b \rceil, \lceil a \rceil \lceil c \rceil \}. 
\]

\(^{15}\)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, \( L^{\text{CRE}}(M) = L(\mathcal{R}^{\text{CRE}}(D_1)) \), while \( L^{\text{HRE}}(M) \neq L(\mathcal{R}^{\text{HRE}}(D_1)) \). Therefore, \text{HRE} does capture the difference between \( M \) and \( D_1 \), while \text{CRE} does not.

In summary, both \text{CRE} and \text{HRE} transformations provide faithful semantics for CFDs. However, \text{HRE} models the syntax of a given CFD explicitly, while \text{CRE} does not. Considering CCs on CFDs and turning to CFMs, \text{CRE} languages may not work very well, but \text{HRE} languages do. This shows that \text{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 \text{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 \text{ORE} transformation does consider only one of the corresponding valid permutations in both stages 2 and 3. (Recall that \text{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 \text{CRE} would be much more than \text{ORE} for a given CFD. The time complexity class of \text{HRE} would be the same as \text{ORE}’s. This is because \text{HRE}, like \text{ORE}, works on osCFDs instead of CFDs.

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

(iv) Automated Analysis. The \text{CRE} and \text{ORE} languages of a given CFD are built over the set of features, while the \text{HRE}’s language is built over the set of features plus two extra symbols. To do analysis operations using the \text{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.
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 \((L_D, L_{cc})\) with \(L_D\) and \(L_{cc}\) denoting the formal languages of the CFD \(D\) and CCs \(cc\), respectively. The formal language associated with the whole model, denoted by \(L_M\), is then equal to \(L_D \cap L_{cc}\). 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 \(D\) 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).

\[(cc_2) \ f_1 \text{ excludes } f_2.\]

(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).

\[(cc_3) \ \text{If the number of occurrences of } f_1 \text{ in a product is even, then the number of occurrences of } f_2 \text{ in the product must be odd.}\]

\[(cc_4) \ \text{The number of occurrences of } f_1 \text{ and } f_2 \text{ in any product are equal.}\]

\[(cc_5) \ \text{The number of occurrences of } f_1, f_2, \text{ and } f_3 \text{ in any product are equal.}\]

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 \( \Sigma \) denote the underlying alphabet. Recall that \( \Sigma = F \) for both CRE and ORE, while \( \Sigma = F \cup \{[,]\} \) 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 \( \mathcal{L}(cc) \). Recall that \( \#_a(w) \) denotes the number of occurrences of a letter \( a \) in a word \( w \).

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

\[
\mathcal{L}(cc_2) = \{ w \in \Sigma^* : (#_{f_1}(w) > 0) \Rightarrow (#_{f_2}(w) = 0) \}.
\]
\[ L(cc_3) = \{ w \in \Sigma^* : (\exists n \in \mathbb{N}. \#f_1(w) = 2n) \Rightarrow (\exists n \in \mathbb{N}. \#f_1(w) = 2n + 1) \}. \]

\[ L(cc_4) = \{ w \in \Sigma^* : \#f_1(w) = \#f_2(w) \}. \]

\[ L(cc_5) = \{ w \in \Sigma^* : \#f_1(w) = \#f_2(w) = \#f_3(w) \}. \]

**Theorem 5.4.** \( L(cc_1), L(cc_2), \) and \( L(cc_3) \) are regular, \( L(cc_4) \) is context-free, and \( L(cc_5) \) 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], \( L(cc_4) \) 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.

\( L_D \) (the language of a CFD \( D \)) can be obtained via three different transformations discussed in Chapter 5, i.e., it can be either the language of \( \text{CRE}, \text{ORE}, \) or \( \text{HRE} \) regular expressions of \( D \). 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 \( M = (D, cc) \), where \( D \) is a CFD and \( cc \) is a set of CCs over \( D \), we define the following three language-based semantics for \( M \):

\[ L^\text{CRE}(M) \overset{\text{def}}{=} L(R^\text{CRE}(D)) \cap L_{cc} \]

\[ L^\text{ORE}(M) \overset{\text{def}}{=} L(R^\text{ORE}(D)) \cap L_{cc} \]
They are called, respectively, the CRE, ORE, and HRE-language semantics of $M$.

The CRE and ORE languages for a given model $M$ are built over the set of features, while the HRE language is over the set of features plus two extra symbols ‘[’ and ‘]’. Recall that the ORE language of $M$ 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 ($L_D$) is always regular, since $R^{CRE}(D)$, $R^{ORE}(D)$, and $R^{HRE}(D)$ 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 $L_M$ for a given CFM $M = (D, cc)$ is given by the type of $L_{cc}$, as $L_D$ 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 $\forall a \in \text{dom}(m): m(a) = m'(a)$.

**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 $M$ is called dead if $\not\exists m \in \mathcal{P}^{\text{flat}}(M)$ such that $f \in \text{dom}(m)$.

**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 $M_1$ and $M_2$ as inputs and decides whether the product line of $M_1$ is a subset of the product line of $M_2$ or not.

Dynamic Specialization: The operation takes two CFMs $M_1$ and $M_2$ as inputs and decides whether the language of $M_1$ is a subset of the $M_2$'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 $\text{Regular} \subset \text{Context-free} \subset \text{Context-sensitive} \subset \text{Recursive}$.

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

**Theorem 5.6.** Given a context-free CFM $M$, 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.

**Theorem 5.7.** Given two CFMs $M_1$ and $M_2$, the following statements hold:

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

(ii) If $M_1$ and $M_2$ are regular and context-free, respectively, then the (Dynamic) Refactoring problem is decidable iff $M_1$ is bounded regular.
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 $M_1$ and $M_2$, the following statements hold:

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

(ii) If $M_1$ and $M_2$ are regular and context-free, respectively, then the problem “is $M_2$ a (dynamic) specialization of $M_1$?” 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 \(\mathcal{C} \subseteq 2^E\) 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 \(D\) of dependencies between events, which make some configurations invalid. Formally, some notion of validity of a set \(X \subseteq E\) wrt. \(D\) is specified so that an event structure \((E, D)\) determines a configuration structure \(\{X \subseteq E : X\ is\ valid\ wrt.\ D\}\).
Table 6.1: Event vs. feature modeling

<table>
<thead>
<tr>
<th>Approach</th>
<th>Event Model</th>
<th>Feature Model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Boolean logic</td>
<td>Modal logic</td>
</tr>
<tr>
<td>Topological</td>
<td>$(E, C)$</td>
<td>$(F, PP, F \rightarrow, I)$</td>
</tr>
<tr>
<td>Structural</td>
<td>$(E, D)$</td>
<td>$(F, M)$</td>
</tr>
<tr>
<td>Logical</td>
<td>$(E, \Phi)$</td>
<td>$(F, \Phi_{BL}, \Phi_{BL}^t)$, $(F, \Phi_{ML})$</td>
</tr>
</tbody>
</table>

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 $(E, \Phi)$ with $\Phi$ 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 $(E, C, \rightarrow, I)$ with $\rightarrow$ a given transition relation over configurations (states), and $I \in C$ 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 $M = (T_{OR}, \mathcal{E}, \mathcal{T}, N)$ can be seen as a special encoding of a set of dependencies analogous to $D$ (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
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

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.
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 $\rightarrow$ SourceLang ImplLang

SourceLang $\rightarrow$ cobol | sdl | sql | cobol sdl | cobol sql | sdl sql | cobol sdl sql

ImplLang $\rightarrow$ 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
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.

\[ D_1 : \]
\[ \text{car} \rightarrow \text{engine} \text{ brake} \]

\[ D_2 : \]
\[ \text{car} \rightarrow \text{engine} \]
\[ \text{engine} \rightarrow \text{brake} \]
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 \( A = (A, +, \emptyset, \times, \{\emptyset\}) \) where \( A = 2^{2^F} \) (power set of power set of features), \( \emptyset \) represents the empty product line, \( \{\emptyset\} \) 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 $(\mathcal{A}, +, 0)$ and $(\mathcal{A}, \times, 1)$ 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 $\mathbf{M}$ is encoded as a term in the product family algebra generated over the prime features of $\mathbf{M}$; the latter are leaves in $\mathbf{M}$’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: $\text{car} = (\text{manual} + \text{automatic}) \times \text{horsepower} \times (1 + \text{aircondition})$.

![Figure 6.3: An FM adopted from [HKM11a]](image)

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 $\oplus$ occurrence 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
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.
Figure 6.4: PPLs vs Staged configuration: a staged configuration

Figure 6.5: PPLs vs Stage configuration: A PPL
the constraints and the \textit{l2C}-principle. For example, consider again Figure 6.4: in the first stage, we make choices between \texttt{manual} and \texttt{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.
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 \( (f_1 > 0 \land \ldots \land f_n > 0) \rightarrow (f > 0) \) and \( (f_1 > 0 \land \ldots \land f_n > 0) \rightarrow (f = 0) \), 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
(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 multiset 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). 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.

\[^1\text{This kind of models are defined to formalize the ORE transformation.}\]
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 $M$, one could apply either a model checker or theorem prover. To apply a model checker, we would need to transform $M$ to its PPL $\mathbb{P}(M)$ and 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 $\Phi_{I2C}(\text{OR})$, 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 ppK\$s provide a good tool to model the behaviour of event-based concurrent systems. Indeed, we believe that ppK\$s 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.

\textit{(vii) Linear Logic Theory of Cardinality-based Feature Models.} Girard in 1987 developed \textit{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 \textit{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. Below, 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 [LSV12], 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 \( \mathcal{L}(\mathbf{M}) \cap \mathcal{L}(F^* f F^*) = \emptyset \). 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.\textsuperscript{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.

\textsuperscript{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 $P \rightarrow P'$, 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 $P' \models \Phi_{BL}(T) \cup \Phi_{BL}(\mathcal{L}X)$ and $P \models \Phi_{BL}^{IC}(P, f)$. We need to prove that $P' \models \Phi_{BL}^{IC}(T_{OR})$.

Let $g \in P$ be an arbitrary feature with $g^\uparrow = f^\uparrow$, i.e., $g \in P \cap (f^\uparrow)_L$. By definition of relative fullness, if $P \models \Phi_{BL}^{IC}(P, f)$, then definitely $P \models \Phi_{BL}^1(T_{OR}^g)$ (one of the union’s components). This implies $P' \models \Phi_{BL}^1(T_{OR}^g)$, and hence $P' \models \bigcup \{\Phi_{BL}^1(T_{OR}^g) : g \in P, g^\uparrow = f^\uparrow\}$. The above statement, along with $P \models \Phi_{BL}^{IC}(T_{OR})$, implies that $P' \models \{f \land g \rightarrow (\land \Phi_{BL}^1(T_{OR}^f)) \lor (\land \Phi_{BL}^1(T_{OR}^g))\}$.

Proposition 3.2. For all $P \in \mathcal{P}$, there exists a full product $P'$ such that $P \rightarrow^* P'$, where $\rightarrow^*$ is the reflexive transitive closure of $\rightarrow$.

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. □

**Theorem 3.2 (Soundness).** \( \mathbb{P}(M) \models \Phi_{\text{ML}}(M) \).

**Proof of Theorem 3.2.** To prove this theorem, we need to show that \( \mathbb{P}(M) \) satisfies any components of the theory \( \Phi_{\text{ML}}(M) \).

(a) \( \mathbb{P}(M) \models \Phi_{\text{BL}}(M) \) is obvious by to Definition 3.5. Thus, all the Boolean theories from Table 3.3 are satisfied by \( \mathbb{P}(M) \).

(b) \( \mathbb{P}(M) \models \Phi_{\text{ML}+}(T) \):

Let \( P \in \mathcal{PP}_M \) and \( P \models f \land \neg \lor f_i \) and \( g \in f_i \). We want to show that \( P \models \text{EX}g \). Let \( P' = P \cup \{g\} \). According to (a), \( P \models \Phi_{\text{BL}}(T) \cup \Phi_{\text{BL}}(\mathcal{E}x') \). Since the \( g \)'s parent is already in \( P' \), adding \( g \) to \( P \) does not violate \( \Phi_{\text{BL}}(T) \). Since exclusive constraints are defined on incomparable features, adding \( g \) to \( P \) also does not violate \( \Phi_{\text{BL}}(\mathcal{E}x') \). Therefore, \( P' \models \Phi_{\text{BL}}(T) \cup \Phi_{\text{BL}}(\mathcal{E}x') \). Since all subfeatures of \( f \) are absent in \( P \), \( \Phi_{\text{IC}}^T(P,f) = \emptyset \) (note Definition 3.6) and hence \( P \models \Phi_{\text{BL}}^T(P,f) \). Since \( P' \models \Phi_{\text{BL}}(T) \cup \Phi_{\text{BL}}(\mathcal{E}x') \) and \( P \models \Phi_{\text{IC}}^T(P,f) \), according to Definition 3.7, there is a transition \( P \xrightarrow[M]{} P' \). Therefore, \( P \models \text{EX}g \).

(c) \( \mathbb{P}(M) \models \Phi_{\text{ML}}(M) \) follows obviously, since the set of states with self-loops in \( \mathbb{P}(M) \) is equal to the set of all full products of \( M \). Note that this also implies that \( \mathbb{P}(M) \) satisfies both theories \( \Phi_{\text{ML}}(\mathcal{O}R) \) and \( \Phi_{\text{IC}}^T \), since these two theories are derivable from the theory \( \Phi_{\text{IC}}^T \).

(d) \( \mathbb{P}(M) \models \Phi_{\text{IC}}(T_{OR}, \mathcal{E}x') \) follows obviously. Indeed, this theory guarantees that there would not be an invalid transition due to \( \text{IC} \) principle.

(e) \( \mathbb{P}(M) \models \Phi_{\text{ML}}(T_{OR}, \mathcal{E}x') \):
Let $f$ and $P$ be a feature and a partial product of $M$, respectively, such that $f \notin P$, $P \models \Phi_{BL}^{EC}(f)$, and $P \not\models \bigvee \Phi_{BL}^T(f)$. Thus, according to Definition 3.7, there exists a transition $P \rightarrow_M P \cup \{f\}$, which implies $P \models \mathcal{E}X f$. This results in $P(M) \models \Phi_{ML_+}^+(T_{OR}, \mathcal{E}\mathcal{X})$.

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

Theorem 3.3 (Semi-completeness). $K \models \Phi_{ML_+}(M)$ implies $K \subseteq P(M)$.

Proof of Theorem 3.3. Let $K \models \Phi_{ML_+}(M)$. $I_K = I_M$, since, due to $K \models \Phi_{BL}(T)$, $K \models r$ ($r$ is the root feature of $M$).

Since $K \models \Phi_{BL}(M)$, according to Definition 3.5, $\mathcal{P}\mathcal{P}_K \subseteq \mathcal{P}\mathcal{P}_M$.

Now, we are going to show that $\rightarrow_K \subseteq \rightarrow_M$.

Due to $K \models \Phi_{ML_+}^i(M)$ and $\mathcal{P}\mathcal{P}_K \subseteq \mathcal{P}\mathcal{P}_M$, any self-loop transitions $P \rightarrow_K P$ in $K$ is a self-loop transition $P \rightarrow_M P$ in $P(M)$.

Consider a transition $P \rightarrow_K P'$, where $P' = P \cup \{f\}$ for a feature $f \notin P$. We want to show that there is a transition $P \rightarrow_M P'$ in $P(M)$. Again, note that any state in $K$ is a partial product of $M$. To prove this statement, according to Definition 3.7, we need to show that (a1) $P' \models \Phi_{BL}(T)$, (a2) $P' \models \Phi_{BL}(\mathcal{E}\mathcal{X})$, and (a3) $P \models \Phi_{BL}^{EC}(P,f)$. (a1) and (a2) is an immediate corollary of $K \models \Phi_{BL}(M)$. To prove (a3), we need to show that for any siblings $g$ with $g \in P$, $P \models \Phi_{BL}^i(T_{OR}^g)$ (see Definition 3.6). Assume by a way of contradiction that $P \not\models \Phi_{BL}^i(T_{OR}^g)$, i.e., $g$ is not completely instantiated in $P$. Since $K \models \Phi_{ML_+}^i(T_{OR})$, $g \in P$, and $P \not\models \Phi_{BL}^i(T_{OR}^g)$, there must not be a transition $P \rightarrow_K P'$. This leads us to a contradiction. Thus, (a3) holds.

163
Based on the above reasonings, \( \rightarrow K \subseteq \rightarrow M \).

To prove Theorem 3.4 (the completeness theorem), we will first need the following lemmas A.1 and A.2.

**Lemma A.1.** \( K \models \Phi^o_{\text{ML}}(M) \) implies \( \mathcal{P}_K = \mathcal{P}_M \).

**Proof of Lemma A.1.** Let \( K \models \Phi^o_{\text{ML}}(M) \). By Theorem 3.3, \( \mathcal{P}_K \subseteq \mathcal{P}_M \). Now we need to show that \( \mathcal{P}_M \subseteq \mathcal{P}_K \):

Let \( P \in \mathcal{P}_M \) 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 \( \{ \text{car, engine, gear, manual, oil} \} \) in the FM in Figure 2.1. We do have the following formulas corresponding to \( \Phi(T) \): \( \text{engine} \rightarrow \text{car}, \text{gear} \rightarrow \text{car}, \text{manual} \rightarrow \text{gear}, \) and \( \text{oil} \rightarrow \text{gear} \), which clearly represent the subtree \( (\text{engine}) \rightarrow \text{car} \leftarrow (\text{manual} \rightarrow \text{gear} \leftarrow \text{oil}) \).

We do a pre-order depth-first traversal of \( T_P \) of a special kind complying l2C-principle: in each level of the tree, all the nodes that are completely instantiated must be visited before the other nodes. In the running example, \text{gear} must be visited before \text{engine}, since it is completely disassembled in \( \{ \text{car, engine, gear, manual, oil} \} \). In this example, the traversal would result in the sequence \( \langle \text{car, gear, manual, oil, engine} \rangle \).

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

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

\( (R-2) \exists(j < i) : f_j = f_{i+1}^\uparrow & \forall g \in \{ f_1, \ldots, f_i \} : (g^\uparrow = f_{i+1}^\uparrow) \Rightarrow (\{ f_1, \ldots, f_i \} \models \Phi^o_{\text{BL}}(T^g_{\overline{OR}})), \text{i.e., } g \text{ is completely instantiated in } \{ f_1, \ldots, f_i \} \).
We prove that any prefix subsequence of $S_P$ is a partial product of $K$ and so $P$ itself. To this end, we use the following inductive reasoning:

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

**Hypothesis:** Assume that, for some $1 \leq i < n$, any prefix of the sequence $\langle f_1, \ldots, f_i \rangle$ is a state in $K$ and there exists a path $\{f_1\} \rightarrow_k \cdots \rightarrow_k \{f_1, \ldots, f_i\}$.

Let $P' = \{f_1, \ldots, f_i\}$.

**Inductive step:** We want to prove that any prefix of the sequence $\langle f_1, \ldots, f_i, f_{i+1} \rangle$ is a state in $K$ and there exists the path $\{f_1\} \rightarrow_k \cdots \rightarrow_k P' \rightarrow_k P' \cup \{f_{i+1}\}$. 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 $K \models \Phi^{\downarrow}_{\text{ML}}(T)$, there is a transition $P' \rightarrow_k P' \cup \{f_{i+1}\}$. Hence, $\{f_1, \ldots, f_{i+1}\} \in \mathcal{P}_K$.

**(R-2).** As $\forall g \in P' : (g \uparrow = f_{i+1} \uparrow) \Rightarrow (P' \models \Phi^{\downarrow}_{\text{BL}}(T_{\text{EX}}))$ (note (R-2) above), $P' \models \Phi^{\downarrow}_{\text{IC}}(f_{i+1})$.

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

Since $P' \models \Phi^{\downarrow}_{\text{BL}}(f_{i+1}) \land \neg \bigvee \Phi^{\downarrow}_{\text{BL}}(f_{i+1}) \land \neg f_{i+1}$, and $K \models \Phi^{\downarrow}_{\text{ML}}(T_{\text{OR}}, \mathcal{E})$, there is a state $\{f_1, \ldots, f_{i+1}\} \in \mathcal{P}_K$ such that $P' \rightarrow_k P' \cup \{f_{i+1}\}$. Hence, $P \in \mathcal{P}_K$.

**Lemma A.2.** $K \models \Phi_{\text{ML}}(M)$ implies $\rightarrow_k = \rightarrow_M$.
Proof of Lemma A.2. Let $K \models \Phi_{\text{ML}}(M)$. There are two types of transitions in a ppKS: self-loop transitions and others. Note that self-loop transitions denote full products. We show that (1) full products of both $P(M)$ and $K$ are the same, i.e., the set of their self-loops are the same, (2) Non-loop transitions in $K$ and $P(M)$ are the same. (1) is obvious, since $K \models \Phi^t_{\text{ML}}(M)$ (note Table 3.1). In the following we also show that the statement (2) holds.

According to Theorem 3.3, $\rightarrow_K \subseteq \rightarrow_M$. Now what we need is to prove that any non-loop transition in $P(M)$ is also a transition in $K$. Note that, according to Lemma A.1, $\mathcal{P}\mathcal{P}_K = \mathcal{P}\mathcal{P}_M$. Consider a transition $P \rightarrow_M P'$, where $P' = P \cup \{f\}$ for a feature $f \notin P$. We want to show that there is a transition $P \rightarrow_K P'$ in $K$. According to Definition 3.7, $P' \models \Phi_{\text{BL}}(T) \cup \Phi_{\text{BL}}(\mathcal{E}\mathcal{X})$, and $P \models \Phi^t_{\text{BL}}(P, f)$. Thus, there are two choices:

(i) $\Phi^t_{\text{BL}}(P, f) = \emptyset$
(ii) $\Phi^t_{\text{BL}}(P, f) \neq \emptyset$

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

(ii): Since $P' \models \Phi_{\text{BL}}(\mathcal{E}\mathcal{X})$, $P \models \neg \bigvee \Phi^t_{\text{BL}}(f)$. Also, $P \models \Phi^t_{\text{BL}}(P, f)$ implies that $P \models \Phi^t_{\text{BL}}(T_{\text{OR}})$ for any $g \in P \cap (f)^+, \bigcup_{1}$, which means $P \models \Phi^t_{\text{BL}}(f)$. Hence, due to $\Phi^t_{\text{ML}}(T_{\text{OR}}, \mathcal{E}\mathcal{X})$, there exists a transition $P \rightarrow_K P'$.

(i) and (ii) implies that any non-loop transition in $P(M)$ is also a transition in $K$. Hence, $\rightarrow_M \subseteq \rightarrow_K$. 

Theorem 3.4 (Completeness). $K \models \Phi_{\text{ML}}(M)$ iff $K = P(M)$.

Proof. Lemma A.1 shows that $K \models \Phi_{\text{ML}}(M)$ implies $\mathcal{P}\mathcal{P}_K = \mathcal{P}\mathcal{P}_M$. Lemma A.2
proves that $K \models \Phi_{ML}(M)$ implies $\rightarrow^K = \rightarrow^M$. Hence, $K \models \Phi_{ML}(M)$ implies $K = \mathbb{P}(M)$. Considering the soundness theorem (Theorem 3.2), the completeness theorem is proven.
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 $x \in \text{dom}(m) \cup \text{MultIng}(m)$. 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 $m = [a^3, b^3]$. According to the above definition, $m[a/[b]] = [[b]^3, b^3]$.

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 $D = (F, r, \uparrow, \mathcal{G}, \mathcal{C})$ be a CFD and $1 \leq k \leq \text{depth}(D)$. The upper diagram induced by $k$ is a CFD
Figure B.1: $D_2$: The diagram induced by depth 2 of $D_1$

$D_{-k} = (F', r, \uparrow_{F'}, G', C')$, where $F' = \{f \in F : \text{depth}(n) \leq k\}$, $G' = G \cap 2^F$, and $C' = C|_{F' \cup G'}$, i.e., its tree is a subtree of $D$’s tree where the nodes are in depth less than or equal to $k$; all other components are inherited from $D$.

For example, $D_2$ is the upper diagram induced by depth 3 of $D_1$ in Figure B.1.

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

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

(ii) $\forall f \in S \cap r_1$, $\exists c \in C(f)$, $\exists n \in \mathcal{P}_{\text{flat}}(D_f)$, $\forall e \in \text{dom}(n) : m(e) = c \times n(e)$.

(iii) $\forall G \in G \cap 2^r$, $\exists n \in \mathcal{P}_{\text{flat}}(D, G)$, $\forall e \in \text{dom}(n) : m(e) = n(e)$.

Proof of Lemma 4.1. For any CFD $D$ and any flat multisets $m$ over $F$, we show that both the following statements hold:
(1) $m \in \mathcal{P}^{\text{flat}}(D) \implies m \text{ satisfies Th-}(i), (ii), \text{ and (iii)}.\footnote{Th-(i), (ii), \text{ and (iii)} \text{ are abbreviations for Theorem 4.1(i), (ii), \text{ and (iii)}, respectively.}}$

(2) $m \text{ satisfies Th-}(i), (ii), (iii) \implies m \in \mathcal{P}^{\text{flat}}(D)$.

Proof of (1):

We prove (1) by the following inductive reasoning on the depth of CFDs.

(base case): Consider a CFD $D$ with $\text{depth}(D) = 1$ and $r$ as its root, i.e., $F_D = \{r\}$ 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 $m(r) = 1$, $S \cap r_\downarrow = \emptyset$, and $G \cap 2^r = \emptyset$.

(hypothesis): Assume that for any CFD $D$ with $\text{depth}(D) < k$ (for some $k$), any $m \in \mathcal{P}^{\text{flat}}(D)$ satisfies the conditions Th-(i), (ii), and (iii).

(inductive step): We show that for any CFD $D$ with $\text{depth}(D) = k$, any $m \in \mathcal{P}^{\text{flat}}(D)$ satisfies the conditions Th-(i), (ii), and (iii).

Let $D = (F, r, \uparrow, G, C)$ be a CFD with $\text{depth}(D) = k$ and $m \in \mathcal{P}^{\text{flat}}(D)$. Holding Th-(i) is clear. Let $D' = D_{\leq k}$ (the upper induced diagram of $D$ by depth $k$, see Definition B.1) and $E = \{f \in F : \text{depth}(f) = k\}$. Let also $S$ denote the set of solitary features in $D$, i.e., $S = S_D$.

Th-(ii):

(S-1): There exists $m' \in \mathcal{P}^{\text{flat}}(D')$ such that $\forall f \in F \setminus E : m(f) = m'(f)$.

Since $\text{depth}(D') = k - 1$, due to the hypothesis,

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

Due to S-1 and the fact that $(F, r, \uparrow)$ is a tree of features,

(S-2): $\forall f \in S \cap r_\downarrow \setminus E, \exists c \in C(f), \exists n' \in \mathcal{P}^{\text{flat}}(D'_f), \forall e \in \text{dom}(n') : m(e) = c \times n'(e)$.

Consider an arbitrary feature $f \in S \cap r_\downarrow \setminus E$. There are unique $c \in C(f)$ and
\(n' \in \mathcal{P}^{\text{flat}}(\mathbf{D}^f)\) satisfying (S-2).\(^2\)

We define a multiset \(n''\) as follows: \(n'' = n' \uplus \left[ e^i : (e \in E \cap S) \land (e^i \in \text{dom}(n')) \land (i = m(e)/c) \right]\). According to (S-2), \(\forall e \in \text{dom}(n'') : m(e) = c \times n''(e)\).

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 \(\forall e \in (F \setminus E) \cup (E \cap S) : n''(e) = n(e)\).

Therefore, according to above and (S-2),
\[
\forall f \in S \cap r_\downarrow, \exists c \in C(f), \exists n \in \mathcal{P}^{\text{flat}}(\mathbf{D}^f), \forall e \in \text{dom}(n) : m(e) = c \times n(e).
\]

Thus, Th-(ii) holds.

**Th-(iii):**

Let \(G \in \mathcal{G} \cap 2^r\). We show that \(\exists n \in \mathcal{P}^{\text{flat}}(\mathbf{D}, G), \forall e \in \text{dom}(n) : m(e) = n(e)\).

There are the two following cases:

(a) \(k = \text{depth}(\mathbf{D}) > 2\),

(b) \(k = \text{depth}(\mathbf{D}) = 2\).

In the former case, \(G \in \mathcal{G}_{\mathbf{D}_r} \cap 2^r\). 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 \(\text{depth}(\mathbf{D'}) = k - 1\), due to the hypothesis,
\[
\exists n' \in \mathcal{P}^{\text{flat}}(\mathbf{D'}, G), \forall e \in \text{dom}(n') : n'(e) = m'(e).
\]

Let \(n = \left( \biguplus_{f \in X} [f^{m(f)}] \right) \uplus n'\), where \(X = E \cap \text{dom}(m) \cap \{f_\downarrow : f \in G\}\).

Clearly, \(n \in \mathcal{P}^{\text{flat}}(\mathbf{D}, G)\).

Since \(\text{dom}(n') \cap E = \emptyset\) and \(\forall f \in F \setminus E : m(f) = m'(f)\), we get to \(\forall e \in \text{dom}(n) : n(e) = m(e)\). Thus, Th-(iii) holds in case (a).

\(^2\) \(n'\) and \(c\) in (S-2) are unique multiset and multiplicity, respectively, for a given \(f \in S \cap r_\downarrow \setminus E\) 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 $\text{depth}(D) = 2$. In this case, $G \subseteq E$.

Let $\text{dom}(m) \cap G = \{f_1, \ldots, f_j\}$ for some $j$.

Let $n = \bigcup_{1 \leq i \leq j} \lceil f_i^m(f_i) \rceil$.

Due to Def-(iii), $\forall 1 \leq i \leq j : m(f) \in C(f)$: (1)

Since $f_i$ is a leaf node in $D$ for any $1 \leq i \leq j$, $\lceil f_i \rceil \in P^\text{flat}(D)$: (2)

Due to Def-(iv), $j = |\text{dom}(m) \cap G| \in C(G)$: (3)

(1), (2), and (3) together imply that $n \in P^\text{flat}(D, G)$. Since $\forall e \in \text{dom}(n) : m(e) = n(e)$, 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 $\forall f \in F_{-r} : f \in \text{dom}(m) \implies (\exists c \in C(f) : m(f) = c \times m(f^\uparrow))$.

Let $f \in F_{-r}$ and $f \in \text{dom}(m)$. Then, either $f \in S$ or $\exists G \in G : f \in G$.

Let us first consider the case $f \in S$: Th-(ii) implies that there exists $c \in C(f)$ and $n \in P^\text{flat}(D^f)$ such that $\forall e \in \text{dom}(n) : m(e) = c \times m(f^\uparrow) \times n(e)$. 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 $D^f$, 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., $\exists G \in G : f \in G$. 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 $n \in P^\text{flat}(D, G)$ such that $n^{m(f^\uparrow)} \subseteq m$.

According to Definition 4.4, there exist $c \in C(G)$, $c_i \in C(f_i)$, $g_i \in \{0, 1\}$, and
\( m_i \in \mathcal{P}^{\text{flat}}(D^f) \) such that \( n = \bigcup_{1 \leq i \leq k} m_i^{c_i \times g_i} \), and \( \sum_i g_i = c \).

Since \( f \in \text{dom}(m) \), \( g_1 \) must be 1. (Note that \( D \) is an unlabelled tree of features.)

Thus, \( n(f) = m_1(f) \times c_1 \).

Since \( f \) is the root feature of \( D^f \) and \( m_1 \in \mathcal{P}^{\text{flat}}(D^f) \), \( m_1(f) = 1 \). 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 \( \forall f \in \mathcal{S} : 0 \notin \mathcal{C}(f) \land m(f^\uparrow) > 0 \implies m(f) > 0 \).

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 \( n \in \mathcal{P}^{\text{flat}}(D^f) \) such that \( \forall e \in \text{dom}(n) : m(e) = c \times m(f^\uparrow) \times n(e) \). Therefore, \( m(f) = n(f) \times c \times m(f^\uparrow) \), as \( f \in \text{dom}(n) \).

Since \( f \) is the root feature of \( D^f \), \( n(f) = 1 \) 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 \( \forall G \in \mathcal{G} : (m(G^\uparrow) > 0) \implies (|\text{dom}(m) \cap G| \in \mathcal{C}(G)) \).

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 \( n \in \mathcal{P}^{\text{flat}}(D, G) \) such that \( \forall e \in \text{dom}(n) : m(e) = m(G^\uparrow) \times n(e) \).

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 \( m_i \in \mathcal{P}^{\text{flat}}(D^{f_i}) \) such that \( n = \bigcup_{1 \leq i \leq k} m_i^{c_i \times g_i} \), and \( \sum_i g_i = c \).

The condition \( \sum_i g_i = c \) implies that \( |\text{dom}(m) \cap G| \in \mathcal{C}(G) \). Hence, Def-(iv) holds.
Theorem 4.1. Given two CFDs $D$ and $D'$, $\mathcal{P}(D) = \mathcal{P}(D') \implies D = D'$.

**Proof of Theorem 4.1.** Let $D = (F, r, \uparrow, G, C)$ and $D' = (F', r', \uparrow', G', C')$ be two CFDs such that $\mathcal{P}(D) = \mathcal{P}(D')$.

Obviously, $\bigcup_{m \in \mathcal{P}(D)} \text{dom}(\text{flat}_F(m)) = F$ and $\bigcup_{m' \in \mathcal{P}(D')} \text{dom}(\text{flat}_{F'}(m')) = F'$. Since $\mathcal{P}(D) = \mathcal{P}(D')$, $F = F'$. (S-1)

We give an inductive reasoning based on $\text{depth}(D)$ (the depth of $D$) to show that $D = D'$.

**(base case):** Let $D = 1$, i.e., $F = \{r\}$, and $\uparrow = G = C = \emptyset$. According to (S-1), $F' = \{r\}$, which implies that $r' = r$, $\uparrow' = G' = C' = \emptyset$. Therefore, $D = D'$.

**(hypothesis):** Assume that for some $n \in \mathbb{N}$ and for any $\text{depth}(D) < n$: $\mathcal{P}(D) = \mathcal{P}(D') \implies D = D'$.

**(inductive step):** We want to show that if $\text{depth}(D) = n$, then $D = D'$.

Let us suppose that $r$ in $D$ ($r'$ in $D'$, respectively) has $k$ ($x$, respectively) solitary subfeatures $f_1, \ldots, f_k$ ($f'_1, \ldots, f'_x$, respectively) and $t$ ($y$, respectively) groups $\{G_1, \ldots, G_t\}$ ($\{G'_1, \ldots, G'_y\}$, respectively). According to Definition 4.7,

$$\mathcal{P}(D) = \{[r, m^c_1, \ldots, m^c_k, g_1, \ldots, g_t], \text{ where}$$

$$\forall 1 \leq i \leq k, \forall 1 \leq j \leq t :$$

$$m_i \in \mathcal{P}(D^{f_i}), c_i \in \mathcal{C}(f_i), g_j \in \mathcal{P}(D, G_j)\} (C)$$

$$\mathcal{P}(D') = \{[r', m'^c_1, \ldots, m'^c_x, g_1, \ldots, g_y], \text{ where}$$
\(\forall 1 \leq i \leq x, \forall 1 \leq j \leq y:\)

\[ m_i \in \mathcal{P}(D^{f'_i}), c_i \in \mathcal{C}'(f'_i), g_j \in \mathcal{P}(D', G'_j). \] (C')

Consider an arbitrary hierarchical product \(m = [r, m^c_1, \ldots, m^c_k, g_1, \ldots, g_t]\), 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 \(1 \leq j \leq t:\)

\(\text{rank}(m_i) \in \mathcal{H}(F) \land \text{rank}(g_j) \in \mathcal{H}(F), r\) is the only urelement in the domain of \(m\), i.e., \(m \in \mathcal{P}(D'): \text{dom}(m') \cap F' = \{r'\}\). Likewise, for any \(m \in \mathcal{P}(D'): \text{dom}(m') \cap F' = \{r'\}\). Since \(\mathcal{P}(D) = \mathcal{P}(D')\), \(r = r'\).

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) \(\forall 1 \leq i \leq k, \exists 1 \leq j \leq k: \mathcal{P}(D^{f_i}) = \mathcal{P}(D'^{f'_i}) \land \mathcal{C}(f_i) = \mathcal{C}'(f'_i),\)

(iii) \(\forall 1 \leq i \leq t, \exists 1 \leq i' \leq t: \mathcal{P}(D, G_i) = \mathcal{P}(D', G'_j).\)

(ii) implies that the sets of \(r\)'s solitary subfeatures in both \(D\) and \(D'\) are the same. Without loss of generality, suppose that \(\forall 1 \leq i \leq k : f_i = f'_i\). Since \(\forall 1 \leq i \leq k : \mathcal{P}(D^{f_i}) = \mathcal{P}(D'^{f'_i})\) and \(\text{depth}(D^{f_i}) < n\), due to the hypothesis, \(D^{f_i} = D'^{f'_i}\).

(iii) implies that the set of groups of \(r\) in \(D\) and \(D'\) 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,
\[ \mathcal{P}(D, G_i) = \{ [m_1^{c_1 \times l_1}, \ldots, m_z^{c_z \times l_z}] : \forall 1 \leq j \leq z. m_j \in \mathcal{P}(D^g_j), c_j \in \mathcal{C}(g_j), l_j \in \{0, 1\}, \text{ and } l_1 + \ldots + l_z \in \mathcal{C}(G_i) \}. \]

\[ \mathcal{P}(D', G_i) = \{ [m_1^{c_1 \times l_1}, \ldots, m_z^{c_z \times l_z}] : \forall 1 \leq j \leq z. m_j \in \mathcal{P}(D'^g_j), c_j \in \mathcal{C}'(g_j), l_i \in \{0, 1\}, \text{ and } l_1 + \ldots + l_z \in \mathcal{C}'(G_i) \}. \]

\[ \mathcal{P}(D, G_i) = \mathcal{P}(D', G_i) \text{ implies that } \forall 1 \leq j \leq z : \mathcal{P}(D^g_j) = \mathcal{P}(D'^g_j) \text{ and } \mathcal{C}(g_j) = \mathcal{C}'(g_j), \mathcal{C}(G_i) = \mathcal{C}'(G_i). \text{ Since } \text{depth}(D^g_i) < n, \text{ due to the hypothesis, } D^g_i = D'^g_i. \]

According to above, since \( r \) in both \( D \) and \( D' \) have the same set of solitary subfeatures and groups whose corresponding induced diagrams are the same with the same multiplicities, \( D = D' \).

\[ \square \]

**Theorem 4.2.** For any CFD \( D \in \mathcal{D}(F) \), the function \( \text{flat}_F|_{\mathcal{P}(D)} \), i.e., the restriction of \( \text{flat}_F \) to the subdomain \( \mathcal{P}(D) \), provides a bijection between \( \mathcal{P}(D) \) and \( \mathcal{P}^{\text{flat}}(D) \).

**Proof.** We use an inductive reasoning based on the depth of CFDs to show this.

(\textit{base case}): The statement obviously holds for any CFD with singleton tree, i.e., a CFD with depth 1.

(\textit{hypothesis}): Assume that the statement holds for any CFD \( D \) with \( 1 \leq \text{depth}(D) < d \) for some \( d \in \mathbb{N} \).

(\textit{inductive step}): We show that \( \text{flat}_F|_{\mathcal{P}(D)} \) provides a bijection from \( \mathcal{P}(D) \) to \( \mathcal{P}^{\text{flat}}(D) \) for any CFD \( D \) with \( \text{depth}(D) = d \).

Let \( D = (F, r, \uparrow, G, C) \) be a CFD with \( \text{depth}(D) = d \) and \( S \subseteq F_r \) denote the set of its solitary features. Suppose that \( S \cap r = \{f_1, \ldots, f_i\} \) (solitary subfeatures of the
root) and $G \cap 2^{r_i} = \{G_1, \ldots, G_j\}$ (groups subelements of the root) for some $i, j \in \mathbb{N}$.

Consider a hierarchical product $h \in \mathcal{P}(D)$. According to Definition 4.7, $h$ is a multiset $\{r, h_1^{c_1}, \ldots, h_i^{c_i}, g_1, \ldots, g_j\}$, where $h_k \in \mathcal{P}(D^{f_k})$, $c_k \in \mathcal{C}(f_k)$ ($1 \leq k \leq i$), and $g_t \in \mathcal{P}(D, G_t)$ ($1 \leq t \leq j$).

According to Definition 4.10, $\text{flat}_F(h) = \left[ r \right] \sqcup \bigcup_{1 \leq k \leq i} (\text{flat}_F(h_k))^{c_k} \sqcup \bigcup_{1 \leq t \leq j} \text{flat}_F(g_t)$.

Since $h_k \in \mathcal{P}(D^{f_k})$ and $\text{depth}(D^{f_k}) < d$ for $1 \leq k \leq i$, due to the hypothesis, $\text{flat}_F(h_k)$ is a flat product of the diagram induced by $f_k$, i.e., $\text{flat}_F(h_k) \in \mathcal{P}^{\text{flat}}(D^{f_k})$.

Let $G_t = \{g_1, \ldots, g_l\}$ for $1 \leq t \leq j$.

According to Definition 4.8, $g_t = [m_1^{c_1 \times t_1}, \ldots, m_l^{c_l \times t_l}]$, where $m_k \in \mathcal{P}(D^{g_k})$, $c_k \in \mathcal{C}(g_k)$, $t_k \in \{0, 1\}$, and $t_1 + \ldots + t_l \in \mathcal{C}(G_t)$ ($1 \leq k \leq l$). According to Definition 4.10, $\text{flat}_F(g_t) = \text{flat}_F(m_1)^{c_1 \times t_1} \sqcup \ldots \sqcup \text{flat}_F(m_l)^{c_l \times t_l}$. Since $\text{depth}(D^{g_k}) < d$ for any $1 \leq k \leq l$, due the hypothesis, $\text{flat}_F(m_k) \in \mathcal{P}^{\text{flat}}(D^{g_k})$. This implies that, according to Definition 4.4, $\text{flat}_F(g_t) \in \mathcal{P}^{\text{flat}}(D, G_t)$.

According to above, $\text{flat}_F(h) = \left[ r \right] \sqcup \bigcup_{1 \leq k \leq i} m_k^{c_k} \sqcup \bigcup_{1 \leq t \leq j} n_t$, where $m_k = \text{flat}_F(h_k)$ ($1 \leq k \leq i$) is a flat product of the diagram induced by $f_k$, i.e., $m_k \in \mathcal{P}^{\text{flat}}(D^{f_k})$ and $n_t$ ($1 \leq t \leq j$) is a flat grouped product of $G_t$, i.e., $n_t = \text{flat}_F(g_t) \in \mathcal{P}^{\text{flat}}(D, G_t)$.

Due to Lemma 4.1, $\text{flat}_F(h) \in \mathcal{P}^{\text{flat}}(D)$. Therefore, $\text{flat}_F|_{\mathcal{P}(D)}$ maps each hierarchical product of $D$ to a flat product of $D$. In the following, we show that $\text{flat}_F|_{\mathcal{P}(D)}$ is an injective function.

Consider two different hierarchical products $h, h' \in \mathcal{P}(D)$ such that $\text{flat}_F(h) = \text{flat}_F(h')$. According to Definition 4.10 and Definition 4.7,

$\text{flat}_F(h) = \left[ r \right] \sqcup \bigcup_{1 \leq k \leq i} \text{flat}_F(h_k)^{c_k} \sqcup \bigcup_{1 \leq t \leq j} \text{flat}_F(g_t)$, and

$\text{flat}_F(h') = \left[ r \right] \sqcup \bigcup_{1 \leq k \leq i} \text{flat}_F(h'_k)^{c'_k} \sqcup \bigcup_{1 \leq t \leq j} \text{flat}_F(g'_t)$, where

$\forall 1 \leq k \leq i, \forall 1 \leq t \leq j: h_k, h'_k \in \mathcal{P}(D^{f_k}), c_k, c'_k \in \mathcal{C}(f_k)$, and $g'_t, g_t \in \mathcal{P}(D, G_t)$.
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, $\text{flat}_F(h) = \text{flat}_F(h')$ implies that for any $1 \leq k \leq i, 1 \leq t \leq j$: $\text{flat}_F(h_k) = \text{flat}_F(h'_k), c_k = c'_k$, and $\text{flat}_F(g_t) = \text{flat}_F(g'_t)$. 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 $\text{flat}_F|_{\mathcal{P}(D)}$ is an injective function from $\mathcal{P}(D)$ to $\mathcal{P}^{\text{flat}}(D)$.

According to Definition 4.7 and Lemma 4.1, $|\mathcal{P}(D)| = |\mathcal{P}^{\text{flat}}(D)|$ (recursive definitions of hierarchical and flat products of $D$) for any CFD $D$, i.e., the cardinalities of the sets of flat and hierarchical products of $D$ are the same. Therefore, the restriction of the flattening function to the hierarchical semantics of $D$ is a surjective function, as it is injective and the cardinalities of the domain and codomain are the same.

According to above, $\text{flat}_F|_{\mathcal{P}(D)} : \mathcal{P}(D) \rightarrow \mathcal{P}^{\text{flat}}(D)$ 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 $D$ with $\text{depth}(D) = 1$.

*(hypothesis):* We assume that for any CFD $D$ with $1 \leq \text{depth}(D) < n$, the statement holds.

*(inductive step):* Let $D = (F, r, \uparrow, G, C)$ be a CFD and $\text{depth}(D) = n$. We show that any hierarchical product $m \in \mathcal{P}(D)$ is a tree-like multiset.
Consider the CFD $D' \overset{\text{def}}{=} D_{-n}$ (upper diagram Induced by depth $n$). Due to Definition 4.7, for any hierarchical product $m \in \mathcal{P}(D)$, there exists $m' \in \mathcal{P}(D')$ such that $m$ is obtained by replacing any feature $f \in \{f \in F : \text{depth}(f) = n - 1\}$ in $m'$ with an $x \in \mathcal{P}(D')$.

Due to the hypothesis, any $x \in \mathcal{P}(D')$ 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 $D$ such that $t \in \mathcal{P}(D)$.

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 $\mathcal{G} = \mathcal{G}_t$. We also define a function $C : (N \setminus \{r\} \cup \mathcal{G}) \rightarrow 2^N$ as follows. $\forall e \in (N \setminus \{r\} \cup \mathcal{G}) : C(e) = \{C_t(e)\}$, where $C_t : (N \setminus \{r\} \cup \mathcal{G})$ is defined in Definition 4.18.

The tuple $D = (T, \mathcal{G}, C)$ 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 $(T, \mathcal{G}, C)$ represents a singleton hierarchical semantics, as all multiplicities are singleton. It is easy to see that its singleton hierarchical product is $t$.

---

\[ See \ Definitions \ 4.16 \ and \ 4.17, \ respectively. \]
Thus, $D$ 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 $G_1 = \{ G \in G : |G| = 1 \}$ and $N'$ be a set of symbols (features) with $N' \cap N = \emptyset$ and $|N'| = |G^1|$. Consider a bijection $l : G^1 \rightarrow N'$. We build a CFD $D' = (N' \cup N, r, \uparrow', G', C')$ as follows.

$$
\forall n \in N \cup N' : \uparrow'(n) = \begin{cases} 
    l^{-1}(n) & \text{if } n \in N' \\
    n & \text{otherwise}
\end{cases}
$$

$$
G' = (G \setminus G^1) \cup \{ G \cup \{ l(G) \} : G \in G^1 \}
$$

$$
C' : ((N' \setminus \{ r \}) \cup G') \rightarrow 2^N \text{ is defined as follows.}
$$

$$
\forall e \in (N' \setminus \{ r \}) \cup G' : C'(e) = \begin{cases} 
    C(e) & \text{if } e \in N \lor e \in G \setminus G^1 \\
    \{ 1 \} & \text{otherwise}
\end{cases}
$$

Clearly, $D'$ is a CFD and $D$ is a substructure of $D'$. Thus, $t \in \mathcal{P}(D')$. The theorem is proven! \hfill \Box

**Theorem 4.5.** Consider an enumerable set of tree-like multisets $U = \{ t_i : i \in I \} \subset \mathcal{T}\mathcal{H}(A)$ over a set $A$, where $I$ enumerates its elements. Let $T_i = (N_i, r_i, \uparrow_i)$ and $G_i$ ($\forall i \in I$) 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 mergeable. According to Definition 4.19, there exists a CFD \( D = (T, \mathcal{G}, \mathcal{C}) \) with \( T = (F, r, \uparrow) \) such that \( t_1, t_2 \in \mathcal{P}(D) \). 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 \( D \).

Formally, \( \forall a \in A : (\exists G \in \mathcal{G}_1 : a \in G) \lor (\exists G \in \mathcal{G}_2 : a \in G) \implies (\exists G \in \mathcal{G} : a \in G) \).

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 \( \uparrow' : N' \setminus \{r'\} \rightarrow N' \) defined as
\( T' = T_1 \cup T_2 \). Note that \((N', r', \cdot') = \{T_1, T_2\}^{\text{merge}}\) (see Definition 4.21).

Let \( G' = G_1 \cup G_2 \), where
\[
G_1 = \{G_1 \cup G_2 : (G_1 \in G_1) \land (G_2 \in G_2) \land (G_1 \cap G_2 \neq \emptyset)\},
\]
\[
G_2 = \{G \in G_1 \cup G_2 : (\forall G' \in G_1 : G' \cap G = \emptyset)\}.
\]

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 \( G_1 \) and \( G_2 \) that share some elements. \( G_1 \) does so. Any other groups in either \( G_1 \) and \( G_2 \) must have to be considered as a group in the merged CFD. Such groups are obtained via \( G_2 \). There may be some singleton elements in \( G \). Note that, according to Definition 5.3, singleton groups are not allowed in a CFD. Below, we address this problem.

Let \( G'' = \{G \in G : |G| = 1\} \) and \( N'' \) be a set of symbols (features) with \( N'' \cap N' = \emptyset \) and \( |N''| = |G''| \). Consider a bijection \( l : G'' \to N'' \).

We define a tuple \( D = (N, r, \cdot, G, C) \), where:
\[
N = N'' \cup N',
\]
\[
r = r',
\]
\[
G = (G' \setminus G'') \cup \{G \cup \{l(G)\} : G \in G''\},
\]
\[
\cdot : N \setminus \{r\} \to N, \text{ defined as:}
\]
\[
\forall n \in N : n' = \begin{cases} 
  l^{-1}(n)' & \text{if } n \in N'' \\
  n' & \text{otherwise}
\end{cases}
\]
∀e ∈ N ∪ G : C(e) =

\begin{align*}
\{0\} \cup C_1(e) & \quad \text{if } (e ∈ N_1 \setminus N_2) \lor (e ∈ G_1 \cap G_2) \\
\{0\} \cup C_2(e) & \quad \text{if } (e ∈ N_2 \setminus N_1) \lor (e ∈ G_1 \cap G_2) \\
C_1(e) \cup C_2(e) & \quad \text{if } (e ∈ N_1 \cap N_2) \lor (e ∈ G_1) \\
\{1\} & \quad \text{otherwise}
\end{align*}

where C_1 and 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 D = (N, r, ↑, G, C) is a CFD. It is obvious that t_1 and t_2 are two hierarchical products of D. Thus, t_1 and t_2 are two mergeable tree-like multisets.

The proof is easily extendable to any enumerating set I ⊆ N, as a set of tree-like multisets are mergeable iff each pairs of tree-like multisets are mergeable.

\textbf{Theorem 4.6.} Consider an enumerable set of tree-like multisets U ⊂ TH(A) over a set A of urelements.

(i) U is mergeable iff U^o is.

(ii) U is mergeable implies that U^o is finite.

\textbf{Proof of Theorem 4.6.} Let U = \{t_i : i ∈ I\} ⊂ TH(A), where I ⊆ N enumerates the elements of U. Let T_i = (N_i, r_i, ↑_i), G_i, and C_i, for any i ∈ I, represent the t_i’s associated tree, groups, and multiplicities, respectively – see Definitions 4.16, 4.17, and 4.18.

\textit{Proof of (i):}
For any $i \in I$, let $T_i^o$ and $G_i^o$ denote the tree and groups associated with $t_i^o$ (the relaxed multiset of $t_i$). According to Proposition 4.1, $\forall i \in I : G_i^o = G_i$ and $T_i^o = T_i$.

According to Theorem 4.5,

$U$ is mergeable

\[ \iff \]

- $\forall i, j \in I : T_i, T_j$ are mergeable.
- $\forall i, j \in I, \forall n \in N_i \cap N_j : (\exists G \in G_i : n \in G) \implies (\exists G \in G_j : n \in G)$.

\[ \iff \]

- $\forall i, j \in I : T_i^o, T_j^o$ are mergeable.
- $\forall i, j \in I, \forall n \in N_i \cap N_j : (\exists G \in G_i^o : n \in G) \implies (\exists G \in G_j^o : n \in G)$.

\[ \iff \]

According to Theorem 4.5, $U^o$ is mergeable.

Proof of (ii):

Suppose that the elements of $U$ are mergeable. Let $D \in D_{U^{merge}}$, i.e., $D$ is a minimal representative CFD of $U$. Let $D = (T, G, C)$ with $T = (N, r, \uparrow)$.

According to (i), the elements of $U^o$ are mergeable. 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 $D^o$ of $U^o$, as follows:

$D^o = (T, G, C^o)$ where

$$
\forall e \in (N \setminus \{r\}) \cup G : C^o(e) = \begin{cases} 
C(e) & \text{if } e \in G \\
\{0, 1\} & \text{otherwise}
\end{cases}
$$

Clearly, $D^o$ is a representative CFD of $U^o$, since $D$ is a minimal representative CFD
of \( U \) and all feature multiplicities in \( D^o \) are \( \{0, 1\} \). Since there is no feature in \( D^o \) with an infinite multiplicity domain, \( P(D^o) \) would be finite. Thus, \( U^o \) is finite, since \( U \subseteq P(D^o) \). 

\[ \]

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

(i) \( U^o \) is completely mergeable, and

(ii) \( \forall t \in U^o, \forall a \in dom(\text{flat}_A(t)), \forall c \in C_U(a), \exists t' \in U : (t'^o = t) \land (#_v(t'^a) = c) \).

Proof. Suppose that \( U \) is completely mergeable, which means that there is some CFD \( D = (T, G, C) \) with \( (F, r, \uparrow) \) representing \( U \). We want to show that the statements (i) and (ii) hold.

We build a CFD \( D^o = (T, G, C^o) \), where \( C^o : (F \setminus \{r\}) \cup G \rightarrow 2^N \) is defined as follows:

\[
\forall e \in (F \setminus \{r\}) \cup G : C^o(e) = \begin{cases} 
C(e) & \text{if } e \in G \\
\{0, 1\} & \text{if } (e \notin G) \land (0 \notin C(e)) \\
\{1\} & \text{if } (e \notin G) \land (0 \notin C(e)) 
\end{cases}
\]

It follows obviously that \( P(D^o) = U^o \). Therefore, \( U^o \) is completely mergeable.

Now, consider a tree-like multiset \( t \in U^o, a \in dom(\text{flat}_A(t)) \), and \( c \in C_U(a) \). We want to show that there exists \( t' \in U \) such that \( t'^o = t \) and \( #_v(t'^a) = c \).

\( t \in U^o \) implies that \( t \in P(D^o) \). \( a \) is a feature in \( D^o \) involved in \( t \) and \( c \) is a valid multiplicity of the feature \( a \) in \( D \) (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 \( \#_{\nu'}(t''^a) = c \), then the statement (ii) is proven. Suppose that \( \#_{\nu'}(t''^a) \neq c \). \( t'' \) is a hierarchical product of \( D \). Thus, for any \( f \in dom(\text{flat}_A(t)) \) (including \( a \)), \( \#_{\nu'}(t''^f) \in C(f) \). According to Definition 4.7, replacing \( \#_{\nu'}(t''^f) \) by any other valid multiplicity in the multiplicity domain of \( f \) would give us another valid hierarchical product of \( D \). Let us define \( t' \) by replacing \( \#_{\nu'}(t''^a) \) by \( c \). \( t' \in P(D) \) 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 \( D^\circ \) such that \( P(D^\circ) = U^\circ \). Note that the multiplicity domain of any feature in \( D^\circ \) is either \( \{0, 1\} \) or \( \{1\} \). Now, we define a CFD \( D \) by replacing the multiplicity of any feature \( a \) in \( D^\circ \) by \( C_U(a) \) (overall multiplicity og \( a \), see Definition 4.23). Clearly, according to (ii), \( P(D) = U \). 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 $m[f \mapsto_p m']$, specified as follows: each occurrence of $f$ in $m$ is substituted by $m'$.

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, $m[f_1 \mapsto_p m'] = [[f'_1^6, f_2], [f_1'^3, f_2^3]]$.

Let $F'' = \{f_1, \ldots, f_k\} \subseteq F$ and $\text{sub}$ be a function, which maps each element $f_i$ of $F''$ to a multiset $m_i$. We usually write $m[f \mapsto_p \text{sub}(f) : \forall f \in F'']$ to mean $m[f_1 \mapsto_p m_1] \ldots [f_k \mapsto_p m_k]$.

**Definition C.2 (Substitution of a Letter with a Language).** Let $\mathcal{L}$ and $\mathcal{L}'$
be two languages and $\sigma \in \Sigma(L)$. The Substitution of $\sigma$ in $L$ with $L'$ is a language, denoted by $L[\sigma \mapsto L']$, equal to: $L[\sigma \mapsto L'] = \{w \in L : \sigma \not\in w\} \cup \{ww'' : (w\sigma w'' \in L) \land (w' \in L')\}$.

For an example, consider $\Sigma = \{a, b, c\}$ and the two languages $L = \{a^n b^n : n \in \mathbb{N}\}$ and $L' = \{c^n : n \in \mathbb{N}\}$: $L[b \mapsto L'] = \{a^n c^m : m \text{ is divisible by } n\}$.

Let $\Sigma' = \{\sigma_1, \ldots, \sigma_k\}$ be a subset of $\Sigma(R)$ and $\text{sub}$ be a function, which maps each letter $\sigma_i$ of $\Sigma'$ to a regular expression $R_i$. We write $L[\sigma \mapsto L \text{ sub}(\sigma) : \forall \sigma \in \Sigma']$ to mean $L[\sigma_1 \mapsto L_1] \ldots [\sigma_k \mapsto L_k]$.

**Definition C.3 (Substitution of a Letter with an Expression).** Let $R$ and $R'$ be two regular expressions and $\sigma \in \Sigma(R)$. The Substitution of $\sigma$ with $R'$ is a regular expression denoted by $R[\sigma \mapsto E R']$ and specified as follows: any instance of $\sigma$ in $R$ is replaced by $R'$.

For example, consider an alphabet $\Sigma = \{a, b, c\}$ and the two regular expressions $R = (a + bc)^*$ and $R' = c^*$. $R[a \mapsto E R'] = (c^* + bc)^*$.\(^1\)

Let $\Sigma' = \{\sigma_1, \ldots, \sigma_k\}$ be a subset of $\Sigma(R)$ and $\text{sub}$ be a function, which maps each letter $\sigma_i$ of $\Sigma'$ to a regular expression $R_i$. We write $R[\sigma \mapsto E \text{ sub}(\sigma) : \forall \sigma \in \Sigma']$ to mean $R[\sigma_1 \mapsto E R_1] \ldots [\sigma_k \mapsto E R_k]$.

**Definition C.4 (Substitution of a Leaf Node with a CFD).** Consider two CFDs $D = (F, r, \uparrow, G, C)$ and $D' = (F', f, \uparrow', G', C')$ and let $f \in \text{lev}(D)$ ($f$ is a leaf node in $D$) such that $F \cap F' = \{f\}$. The substitution of $f$ with $D'$ is a CFD, denoted by $D[f \mapsto_D D']$, equal to $(F \cup F', r, \uparrow \cup \uparrow', G \cup G', C \cup C')$.\(^\Box\)

\(^1\)This regular expression would be semantically equal to $(c + b)^*$. 

188
As an example, consider the CFDs $D_1$ and $D_2$ in Figure C.2. The substitution of the leaf node $b$ in $D_1$ with $D_2$ would be $D_3$ represented in Figure C.2.

Let $D$ 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., $F' \subseteq \text{lev}(F)$. Consider CFDs $D_1, \ldots, D_k$ over set of features $F_1, \ldots, F_k$, respectively, such that for all $i$ the root of $D_i$ is $f_i$ and for all distinct indices $1 \leq i, j \leq k$: $F \cap F_i = \{f_i\}$ and $F_i \cap F_j = \emptyset$. Let $\text{sub}$ be a function, which maps each element $f_i$ in $F'$ to the CFD $D_i$. For succinctness, we usually write $D[f \mapsto_D \text{sub}(f) : \forall f \in F']$ to mean $D[f_1 \mapsto_D D_1] \ldots [f_k \mapsto_D D_k]$.

**Definition C.5 (CFDs Cut by Nodes).** Let $D = (F, r, \uparrow, G, C)$ be a CFD and $f \in F$. The CFD cut by $f$ is a CFD $D^{-f} = (F', r, \uparrow|_{F'}, G', C|_{G' \cup N'})$, where $F' = F \setminus f \downarrow$ and $G' = G \cap 2^{F'}$, i.e., its tree is the tree of $D$ in which the tree under $f$ is excluded; all other components are inherited from $D$.

As an example, $D_2$ in Figure C.3 is the CFD cut by $c$ in $D_1$ in Figure C.3. The following propositions and lemmas come in handy in the proofs of the main theorems. Proposition C.1 follows obviously.
**Proposition C.1.** Let \( D = (F, r, \downarrow, G, C) \) be a CFD and \( f \in F \). Then, the following statements hold:

(i) \( D = D^{-f}[f \mapsto_D D'] \).

(ii) \( P^{\text{flat}}(D) = P^{\text{flat}}(D^{-f})[f \mapsto_P P^{\text{flat}}(D')] \).

(iii) \( R^{\text{CRE}}(D) = R^{\text{CRE}}(D^{-f})[f \mapsto_E R^{\text{CRE}}(D')] \).

(iv) \( R^{\text{ORE}}(D) = R^{\text{ORE}}(D^{-f})[f \mapsto_E R^{\text{ORE}}(D')] \).

**Lemma C.1.** Let \( D = (F, r, \downarrow, G, C) \) be a CFD and \( 1 \leq d \leq \text{depth}(D) \). Then, \( D = D_{-d}[f \mapsto_D D'] : \forall f \in F' \), where \( F' = \{ f \in F : \text{depth}(f) = d \} \), i.e., the nodes with depth \( d \).

**Proof of Lemma C.1.** Let \( F' = \{ f_1, \ldots, f_i \} \). We define a set of CFDs \( \{D_0, D_1, \ldots, D_i\} \) recursively as follows: \( D_0 = D \) and for any \( 1 \leq j \leq i \): \( D_j = (D_{j-1})^{-f_j} \).

Note that \( D_{-d} = D_i \). Due to Proposition C.1(i), \( D_j = D_{j-1}[f_j \mapsto_D D_j'] \), for any \( 1 \leq j \leq i \). Therefore, \( D = D_0 = D_{-d}[f_1 \mapsto_D D_1] \ldots [f_i \mapsto_D D_i] \), which is equal to \( D_{-d}[f \mapsto_D D'] : \forall f \in F' \). 

---

\( D_{-d} \) denotes the upper diagram induced by depth \( d \) in \( D \) – see Definition B.1.
Lemma C.2. Let $D = (F, r, \uparrow, G, C)$ be a CFD and $0 \leq d \leq \text{depth}(D)$. Then, 
\[ \mathcal{P}^{\text{flat}}(D) = \mathcal{P}^{\text{flat}}(D_{-d})[f \mapsto \mathcal{P}^{\text{flat}}(D^f) : \forall f \in F'], \] where $F' = \{f \in F : \text{depth}(f) = d\}$, 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 \(\{D_0, D_1, \ldots, D_i\}\) recursively as follows: $D_0 = D$ and for any $1 \leq j \leq i$: $D_j = (D_{j-1})^{-f_j}$. Note that $D_{-d} = D_i$. Due to Proposition C.1(ii), $\mathcal{P}^{\text{flat}}(D_{j-1}) = \mathcal{P}^{\text{flat}}(D_j)[f_j \mapsto \mathcal{P}^{\text{flat}}(D_{f_j})]$. Therefore, $\mathcal{P}^{\text{flat}}(D) = \mathcal{P}^{\text{flat}}(D_0) = \mathcal{P}^{\text{flat}}(D_{-d})[f_1 \mapsto \mathcal{P}^{\text{flat}}(D_1)] \ldots [f_i \mapsto \mathcal{P}^{\text{flat}}(D_i)]$, which is equal to $\mathcal{P}^{\text{flat}}(D_{-d})[f \mapsto \mathcal{P}^{\text{flat}}(D^f) : \forall f \in F']$.

Lemma C.3. For any CFD $D = (F, r, \uparrow, G, C)$ and $0 \leq d \leq \text{depth}(D)$: $\mathcal{R}^{\text{CRE}}(D) = \mathcal{R}^{\text{CRE}}(D_{-d})[f \mapsto E \mathcal{R}^{\text{CRE}}(D^f) : \forall f \in F']$, where $F' = \{f \in F : \text{depth}(f) = d\}$, 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 \(\{D_0, D_1, \ldots, D_i\}\), recursively, as follows: $D_0 = D$ and for any $1 \leq j \leq i$: $D_j = (D_{j-1})^{-f_j}$. Note that $D_{-d} = D_i$. Due to Proposition C.1(iii), $\mathcal{R}^{\text{CRE}}(D_{j-1}) = \mathcal{R}^{\text{CRE}}(D_j)[f_j \mapsto E \mathcal{R}^{\text{CRE}}(D_{f_j})]$. Therefore, $\mathcal{R}^{\text{CRE}}(D) = \mathcal{R}^{\text{CRE}}(D_0) = \mathcal{R}^{\text{CRE}}(D_{-d})[f_1 \mapsto E \mathcal{R}^{\text{CRE}}(D_1)] \ldots [f_i \mapsto E \mathcal{R}^{\text{CRE}}(D_i)]$, which is equal to $\mathcal{R}^{\text{CRE}}(D_{-d})[f \mapsto E \mathcal{R}^{\text{CRE}}(D^f) : \forall f \in F']$.

The proofs for the main theorems of Chapter 5 are given in the following.

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

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 $\text{Par}(\mathcal{L}(\mathcal{R}^{\text{CRE}}(D))) = \mathcal{P}^{\text{flat}}(D)$ holds for any CFD $D$ with depth($D$) = 1.
(hypothesis): Assume that $\text{Par}(\mathcal{L}(\mathcal{R}^{\text{CRE}}(D))) = \mathcal{P}^{\text{flat}}(D)$ for any CFD $D$ with $1 \leq \text{depth}(D) \leq d$ for some $d \in \mathbb{N}$.

(inductive step): We want to prove that for any CFD $D$ with $\text{depth}(D) = d + 1$ the statement holds, i.e., $\text{Par}(\mathcal{L}(\mathcal{R}^{\text{CRE}}(D))) = \mathcal{P}^{\text{flat}}(D)$.

Let $D = (F, r, \uparrow, G, C)$ be a CFD with $\text{depth}(D) = d + 1$.

Due to Lemma C.1, $D = D - d[f \mapsto_D D'] : \forall f \in F'$, where $F' = \{f \in F : \text{depth}(f) = d\}$, i.e., the nodes with depth $d$.

Due to Lemma C.3, $\mathcal{R}^{\text{CRE}}(D) = \mathcal{R}^{\text{CRE}}(D - d)[f \mapsto_E \mathcal{R}^{\text{CRE}}(D')] : \forall f \in F'$.

Therefore, $\mathcal{L}(\mathcal{R}^{\text{CRE}}(D)) = \mathcal{L}(\mathcal{R}^{\text{CRE}}(D - d))[f \mapsto_L \mathcal{L}(\mathcal{R}^{\text{CRE}}(D')) : \forall f \in F']$.

Obviously, $\text{Par}(\mathcal{L}(\mathcal{R}^{\text{CRE}}(D))) = \text{Par}(\mathcal{L}(\mathcal{R}^{\text{CRE}}(D - d)))[f \mapsto_p \text{Par}(\mathcal{L}(\mathcal{R}^{\text{CRE}}(D')))) : \forall f \in F']$.

Since $\text{depth}(D - d) = d$, according to the hypothesis, $\text{Par}(\mathcal{L}(\mathcal{R}^{\text{CRE}}(D - d))) = \mathcal{P}^{\text{flat}}(D - d)$.

Since for any $f \in F'$: $\text{depth}(D') < d$, according to the hypothesis, $\text{Par}(\mathcal{L}(\mathcal{R}^{\text{CRE}}(D')))) = \mathcal{P}^{\text{flat}}(D')$.

Therefore, $\text{Par}(\mathcal{L}(\mathcal{R}^{\text{CRE}}(D))) = \mathcal{P}^{\text{flat}}(D - d)[f \mapsto_p \mathcal{P}^{\text{flat}}(D') : \forall f \in F']$.

Due to Lemma C.2, since $\mathcal{P}^{\text{flat}}(D) = \mathcal{P}^{\text{flat}}(D - d)[f \mapsto_p \mathcal{P}^{\text{flat}}(D') : \forall f \in F']$, $\text{Par}(\mathcal{L}(\mathcal{R}^{\text{CRE}}(D))) = \mathcal{P}^{\text{flat}}(D)$. The theorem is proven. \hfill $\square$

**Theorem 5.2.** For a given CFD $D$, $\mathcal{L}(\mathcal{R}^{\text{CRE}}(D))$ preserves $D$’s hierarchy.

**Proof of Theorem 5.2.** Consider a CFD $D = (F, r, \uparrow, G, C)$. We need to prove the following statements for any $f, f' \in F$:

1. $(f' \in f_{\downarrow}) \Rightarrow (\forall w \in \mathcal{L}(\mathcal{R}^{\text{CRE}}(D)) : (f' \in U_w) \Rightarrow (f \sqsubseteq_w f'))$.
2. $(\forall w \in \mathcal{L}(\mathcal{R}^{\text{CRE}}(D)) : (f' \in U_w) \Rightarrow (f \sqsubseteq_w f')) \Rightarrow (f' \in f_{\downarrow})$.

Note that (1) implies $(f' \in f_{\downarrow}) \Rightarrow (\forall w \in \mathcal{L}(\mathcal{R}^{\text{CRE}}(D)) : (f' \in w) \Rightarrow (f \sqsubseteq_w f'))$. 

192
Proof of (1):

Since D is an unlabelled tree of features, for any \( i \leq \text{depth}(D) \), \((\text{shr}^{\text{CRE}})^i(D)\) 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 \textit{disjoint labeled CRDs} (DL-CRD).

It is obvious that for any DL-CRD \( RD \) and \( i \leq \text{depth}(RD) \), \((\text{shr}^{\text{CRE}})^i(RD)\) is also a DL-CRD. To prove (1), we prove a more general statement stated as follows:

"Consider a DL-CRD \( RD = (LT_{re}, G, \mathcal{C}) \) with \( LT_{re} = (N, r, \uparrow, \Sigma, l_{re}) \). Let \( n', n'' \in N \) such that \( l_{re}(n') = R' \) and \( l_{re}(n'') = R'' \) such that \( n'' \in n'^\uparrow \). Then,

\[
\forall w \in \mathcal{L}(\mathcal{R}^{\text{CRE}}(RD)), \forall w'' \in \mathcal{L}(R'') : (w'' \leq_{\text{seq}} w) \Rightarrow [\exists w' \in \mathcal{L}(R') : w'.w'' \leq_{\text{seq}} w].
\]

Let \( w \in \mathcal{L}(\mathcal{R}^{\text{CRE}}(RD)) \) and \( w'' \in \mathcal{L}(R'') \) such that \( w'' \leq_{\text{seq}} w \). We need to show that \( \exists w' \in \mathcal{L}(R') : w'.w'' \leq_{\text{seq}} w \).

Since \( RD \) and \((\text{shr}^{\text{CRE}})^i(RD)\) for any \( i \leq \text{depth}(RD) \) are DL-CRDS, \( \mathcal{R}^{\text{CRE}}(RD) = R.(R'.(R''.R^{(3)}+R^{(4)})+R^{(5)}) \) for some regular expresions \( R, R^{(3)}, R^{(4)}, R^{(5)} \) (note the definition dre^{CRE} in Definition 5.11 and Definition 5.10) such that the regular expressions \( R, R', R'', R^{(3)}, R^{(4)}, R^{(5)} \) are built over disjoint alphabets. Since \( w'' \leq_{\text{seq}} w \), \( w \in \mathcal{L}(R.R'.R''.R^{(3)}) \). The statement is proven, since \( R' \) precedes \( R'' \) in \( R.R'.R''.R^{(3)} \), i.e., \( \exists w' \in \mathcal{L}(R') : w'.w'' \leq_{\text{seq}} w \).

Proof of (2):

We show \( \neg(f' \in f_{\perp\perp}) \Rightarrow \neg(\forall w \in \mathcal{L}(\mathcal{R}^{\text{CRE}}(D)) : (f' \in U_w) \Rightarrow (f \sqsubseteq_w f')) \), which is equivalent to (2).

Suppose that \( f' \not\in f_{\perp\perp} \). Let \( k \) be the minimum of \( \text{depth}(f) \) and \( \text{depth}(f') \). Let \( d = \text{depth}(D) \) and \( (\text{shr}^{\text{CRE}})^{d-k}(D) = RD' \).
There are two leaves $\ell$ and $\ell'$ in $RD'$ with labels $R$ and $R'$ in $RD'$ such that $f \in \Sigma(R)$ and $f' \in \Sigma(R')$. Since $RD'$ is an DL-CRD, $\Sigma(R') \cap \Sigma(R) = \emptyset$.

Note that by applying the shrinking step on $D$ $d-k$ times (where $k = \min(\text{depth}(f), \text{depth}(f'))$) the parents of both $\ell$ and $\ell'$ would be the same and equal to the least common ancestor of $f$ and $f'$, i.e., $\ell$ and $\ell$ are siblings in $RD'$. Let $p = \ell^t = \ell'^t$. There are the following choices for $\ell$ and $\ell'$:

(i) Both are solitary nodes.

(ii) One of them, say $\ell$, is in a group and another one, $\ell'$, is a solitary node.

(iii) Both are in the same group $G$.

(iv) One of them, say $\ell$, is in a group $G$ and another, $\ell'$, is in another group $G'$.

Let $RD'_1 = gle^{CRE} \circ mel^{CRE}(RD')$ (applying the first and second stages of shrinking steps on $RD'$). There are two leaves $\ell_1$ and $\ell'_1$ with labels $R_1$ and $R'_1$ in $RD'_1$ such that $f \in \Sigma(R_1)$ and $f' \in \Sigma(R'_1)$. Note that $\Sigma(R_1) \cap \Sigma(R'_1) = \emptyset$ and all leaves in $RD'_1$ are solitary with multiplicities $(1, 1)$.

Now let us apply the function $dre^{CRE}$ on $RD'_1$ to get $(\text{shr}^{CRE})^{d-k+1}(D)$. Since the function $dre^{CRE}$ considers all valid permutations of the $p$’s child nodes, there is a leaf node $\ell''$ in $(\text{shr}^{CRE})^{d-k+1}(D)$ 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 $\Sigma(R_1) \cap \Sigma(R'_1) = \emptyset$, there are two words $w_1, w_2 \in L(R_{\ell''})$ such that $f \sqsubseteq w_1$ $f'$ and $f' \sqsubseteq w_2$ $f$. Thus, keeping doing the shrinking steps until getting $\text{shr}^{CRE}(D)$, there would be a word $w \in L(R^{CRE}(D))$ such that $f' \in w$ but $\neg(f \sqsubseteq w$ $f')$. The statement (2) is proven. □
Theorem 5.3. For any given osCFD OD:

(i) $L(R_{ORE}(OD)) \subseteq L(R_{CRE}(OD_{cfd}))$.

(ii) $\text{Par}(L(R_{ORE}(OD))) = \text{Par}(L(R_{CRE}(OD_{cfd})))$.

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 $L(R_{ORE}(OD))$ belongs also to $L(R_{CRE}(OD_{cfd}))$, which implies that $L(R_{ORE}(OD)) \subseteq L(R_{CRE}(OD_{cfd}))$. The statement (i) is proven.

(i) implies that $\text{Par}(L(R_{ORE}(OD))) \subseteq \text{Par}(L(R_{CRE}(OD_{cfd})))$. Therefore, for any word $w \in L(R_{CRE}(OD_{cfd}))$, there is a word $w' \in L(R_{ORE}(OD))$ 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., $\text{Par}(L(R_{ORE}(OD))) = \text{Par}(L(R_{CRE}(OD_{cfd}))))$.

Theorem 5.4. $L(cc_1)$, $L(cc_2)$, and $L(cc_3)$ are regular, $L(cc_4)$ is context-free, and $L(cc_5)$ 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 $n \geq 3$. 

195
\( \mathcal{L}(cc_1) \) can be expressed by the following regular expression, where \( r = (f_1 + \ldots + f_n)^* \):

\[
f_2^* + rf_1 r f_2 r + rf_2 r f_1 r.
\]

\( \mathcal{L}(cc_2) \) 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 \( \mathcal{L}(cc_3) \). 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}(cc_4) \) and \( \mathcal{L}(cc_5) \) 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 $\forall w \in L_{ORE}(M), \forall f' \in F' : f \sqsubseteq_w f'$. This problem is decidable in no classes of CFMs when $L_{ORE}(M)$ is infinite. However, we can reduce it to the following smart problem:

The problem deals with only the underlying CFD. Let $D$ denote the CFD of $M$. Since the problem has nothing to do with multiplicities, it is sufficient to work with $D^\circ$, i.e., the relaxed CFD of $D$. Thus, the above problem is reduced to "$f$ is a common ancestor of the features in $F'$ iff $\forall w \in L_{ORE}(D^\circ), \forall f' \in F' : f \sqsubseteq_w f'$". Since $L_{ORE}(D^\circ)$ 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.

**Theorem 5.6.** Given a context-free FM $M$, 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 $M$.

*Partial Configuration:* Let $L$ denote the set of all prefixes of the words of $L_{CRE}(M)$. $L$ is a context-free language. To prove this, we take the grammar of $L_{CRE}(M)$ in Chomsky Normal Form and for every production $A \rightarrow BC$, add productions $A_\varepsilon \rightarrow$
$BC_\varepsilon$ and $A_\varepsilon \rightarrow B_\varepsilon$. Also, for every production $A \rightarrow f$ (for some terminals $f$), we consider the production $A_\varepsilon \rightarrow f$. Finally, we change the starting variable $S$ to $S_\varepsilon$ and add the production $S_\varepsilon \rightarrow \varepsilon$. 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 $n \in \mathbb{N}$, $\mathcal{L} = \mathcal{L}(F^* f_1^* F^* \cdots f_n^* F^*)$. The problem is reduced to determining whether $\mathcal{L}^{\text{CRE}}(M) \subseteq \mathcal{L}$ or not. In other words, the problem is reduced to determining whether $\mathcal{L}^{\text{CRE}}(M) \cap \mathcal{L}^c = \emptyset$ 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 $\mathcal{L}^{\text{CRE}}(M) \cap \mathcal{L}^c$ 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 $\mathcal{L} \cap \mathcal{L}^{\text{CRE}}(M) \neq \emptyset$, 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 $M$ or not is, indeed, to determine whether $\mathcal{L} \cap \mathcal{L}^{\text{CRE}}(M) = \emptyset$.
∅ or not. Note that \( \mathcal{L} \) is regular. Hence, \( \mathcal{L} \cap \mathcal{L}^{\text{CRE}}(M) \) 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.

\[ \square \]

**Theorem 5.7.** Given two CFMs \( M_1 \) and \( M_2 \), the following statements hold:

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

(ii) If \( M_1 \) and \( M_2 \) are regular and context-free, respectively, then the (Dynamic) Refactoring problem is decidable iff \( M_1 \) 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. \( \square \)

**Theorem 5.8.** Given two CFMs \( M_1 \) and \( M_2 \), the following statements hold:

(i) If both are regular, the (Dynamic) Specialization problem between them is decidable.
(ii) If $M_1$ and $M_2$ are regular and context-free, respectively, then the problem “is $M_2$ a (dynamic) specialization of $M_1$?” 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 “$M_2$ is a specialization of $M_1$” is reducible to the problem $L^{\text{CRE}}(M_2) \subseteq L^{\text{CRE}}(M_1)$. In other words, it is equivalent to the problem of determining whether $L^{\text{CRE}}(M_2) \cap L^{\text{CRE}}(M_1)^c = \emptyset$ or not. Since the class of regular languages is closed under complement, $L^{\text{CRE}}(M_1)^c$ is regular. Thus, $L^{\text{CRE}}(M_2) \cap L^{\text{CRE}}(M_1)^c$ 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. \qed
Bibliography


[BSRC06b] David Benavides, Sergio Segura, Pablo Trinidad, and Antonio Ruiz-Cortés. Using java csp solvers in the automated analyses of feature


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)

C_t Multiplicities associated with a tee-like multiset t (pp 80)

C_U Overall multiplicities of a set of multisets of U (pp 88)

\mathcal{FP}_M Full products of M (pp 40)

AF Future modality (pp 46)

AG Global modality (pp 46)

G_t Groups associated with a tee-like multiset t (pp 79)
\( \text{AX} \)  
Next modality (pp 46)

\( \mathcal{P}_M \)  
Partial products of \( M \) (pp 41)

\( \mathbb{P}(M) \)  
Partial Product Line of an FM \( M \) (pp 43)

\( \mathcal{L}^{\text{Par}} \)  
The Parikh image of a language \( \mathcal{L} \) (pp 24)

\( w^{\text{Par}} \)  
The Parikh image of a word \( w \) (pp 24)

\( \Phi_{\text{Bl}}^{\text{I2C}}(T_{\text{OR}}) \)  
Boolean theory of \( \text{I2C} \) (pp 39)

\( \Phi_{\text{Bl}}^{\text{I2C}}(P,f) \)  
Boolean Theory of fully instantiated \( P \) wrt. \( f \) (pp 42)

\( \Phi_{\text{Bl}}(M) \)  
Boolean theory of partial products (pp 39)

\( \Phi_{\text{Bl}}(\mathcal{E} \mathcal{X}) \)  
Boolean theory of exclusive constraints (pp 39)

\( \Phi_{\text{Bl}}(T) \)  
Boolean theory of a tree (pp 39)

\( \bot \)  
Logical false (pp 15)

\( \cap \)  
Intersection operation on sets (pp 21)

\( \mathcal{C} \)  
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)

\( f_i \)  
Children of a node \( f \) in a tree (pp 29)

\( \leq_\ast \)  
An ordering relation on \( \mathbb{N}^\ast \) (pp 95)

\( cplev(\text{RD}) \)  
Leaves all of whose siblings are leaves in a CFD \( D \) (pp 98)
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>∪</td>
<td>Union operation on sets (pp 15)</td>
</tr>
<tr>
<td>$D^{-f}$</td>
<td>Cutting of a CFD $D$ by a node $f$ (pp 189)</td>
</tr>
<tr>
<td>$\text{depth}(D)$</td>
<td>Depth of a CFD $D$ (pp 97)</td>
</tr>
<tr>
<td>$\text{dom}(m)$</td>
<td>Domain of a multiset $m$ (pp 61)</td>
</tr>
<tr>
<td>$\text{dre}^{\text{CRE}}$</td>
<td>CRE-DR function (pp 105)</td>
</tr>
<tr>
<td>$\text{dre}^{\text{ORE}}$</td>
<td>ORE-DR function (pp 114)</td>
</tr>
<tr>
<td>$\exists$</td>
<td>Existence quantifier (pp 21)</td>
</tr>
<tr>
<td>$F^{d_{\leftrightarrow}}$</td>
<td>Nodes with depth $d$ in a CFD (pp 172)</td>
</tr>
<tr>
<td>$\text{ppKS}(F)$</td>
<td>Set of all $\text{ppCTL}$-formulas over $F$ (pp 46)</td>
</tr>
<tr>
<td>$K(F)$</td>
<td>Class of $\text{ppKS}$s over $F$ (pp 45)</td>
</tr>
<tr>
<td>$\mathcal{L}(\mathcal{R})$</td>
<td>Language of a regular expression $\mathcal{R}$ (pp 20)</td>
</tr>
<tr>
<td>$\mathcal{L}_D$</td>
<td>Language of a CFD $D$ (pp 126)</td>
</tr>
<tr>
<td>$\mathcal{L}_M$</td>
<td>Language of a CFM $M$ (pp 126)</td>
</tr>
<tr>
<td>$\mathcal{L}_{cc}$</td>
<td>Language of a CCs $cc$ (pp 126)</td>
</tr>
<tr>
<td>$\mathcal{L}^{\text{CRE}}(M)$</td>
<td>CRE Language of a CFM $M$ (pp 128)</td>
</tr>
<tr>
<td>$\mathcal{L}^{\text{HRE}}(M)$</td>
<td>HRE Language of a CFM $M$ (pp 128)</td>
</tr>
<tr>
<td>$\mathcal{L}^{\text{ORE}}(M)$</td>
<td>ORE Language of a CFM $M$ (pp 128)</td>
</tr>
<tr>
<td>$\mathcal{M}(F)$</td>
<td>The class of all FMs over $F$ (pp 30)</td>
</tr>
</tbody>
</table>
$\Phi_{\text{ML}^+}(M)$  $\Phi_{\text{ML}}(M) \setminus \Phi_{\text{ML}^+}(M)$ (pp 48)

$\Phi_{\text{ML}^+}(M)$  Soundness ML theory of M (pp 46)

$\Phi_{\text{ML}}(M)$  Completeness ML theory of M (pp 46)

\(\forall\)  Universal quantifier (pp 46)

\(f_{\downarrow\downarrow}\)  Grandchildren of a node \(f\) in a tree (pp 29)

\(G_{\text{deg}}(C')\)  Degree of depth of groups (pp 172)

\(G_{\text{dep}}(D)\)  Set of depth of groups (pp 172)

\(g\text{ex}_{\text{RD}}\)  Mapping of grouped leaves to regular expressions in a CFD \(D\) (pp 102)

\(g\text{ex}_{\text{ORD}}\)  Mapping of grouped leaves to ORE regular expressions in a CFD \(D\) (pp 111)

\(g\text{le}_{\text{CRE}}\)  CRE-EGL function (pp 103)

\(g\text{le}_{\text{ORE}}\)  ORE-EGL function (pp 112)

\(f_{\uparrow\uparrow}\)  Grandparents of a node \(f\) in a tree (pp 29)

\(g\text{lev}(D)\)  Grouped leaves of a CFD \(D\) (pp 97)

\(D_{-k}\)  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)
$lev(D)$ Set of leaves of a CFD $D$ (pp 97)

$\text{lex}_{RD}$ Mapping of leaves to regular expressions in a CFD $D$ (pp 100)

$\text{low}(C)$ Lower bound of a multiplicity domain $C$ (pp 96)

$\text{low}(c)$ Lower bound of a multiplicity $c$ (pp 96)

$\rightarrow$ Logical implication (pp 15)

$\rightarrow^*$ Reflexive transitive closure of transition relation $\rightarrow$ (pp 45)

$\rightarrow^+$ Transitive closure of transition relation $\rightarrow$ (pp 44)

$max(a)_{a \in A}$ Maximum element of a set $A$ (pp 172)

$\text{mel}^{CRE}$ CRE-EML function (pp 101)

$\text{mel}^{ORE}$ ORE Multiplicity eliminator function (pp 109)

$D_{U\text{merge}}$ Family of minimal representative CFDs of $U$ (pp 83)

$\mathcal{T}_{\text{merge}}$ Merged tree of trees $\mathcal{T}$ (pp 84)

$|=|$ Logical satisfaction relation (pp 9)

$\mathcal{H}(A)$ Hierarchy of finite multisets over $A$ (pp 71)

$\text{MultIng}(m)$ 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)

$\mathbb{N}^*$ $\mathbb{N} \cup \{\ast\}$ (pp 95)
\(\neg\) Logical negation (pp 46)

\(#_w(\sigma)\) The number of occurrences of \(\sigma\) in a word \(w\) (pp 24)

\(\textit{ORD}(\Sigma)\) Class of osCRDs over \(\Sigma\) (pp 108)

\(\textit{OD}(\textit{FD})\) Set of all osCFDs all of whose underlying CFDs are \(D\) (pp 108)

\(\textit{OD}(F)\) Class of osCFDs over \(F\) (pp 108)

\(\text{Per}^k_{\leq}(X)\) \(\text{Per}^{(k,k)}_{\leq}(X)\) (pp 111)

\(\text{Per}^k(X)\) \(\text{Per}^{(k,k)}(X)\) (pp 102)

\(\text{Per}^{(l,u)}_{\leq}(X)\) Concatenation permutations with length between \(l\) and \(u\) of \(X\) considering a total ordering on \(X\) (pp 111)

\(\text{Per}^{(l,u)}(X)\) Concatenation permutations with length between \(l\) and \(u\) of \(X\) (pp 102)

\(\text{pex}_{\text{CRE}}\) Mapping of parents of leaves to \(\text{CRE}\)-Expressions (pp 105)

\(\text{pex}_{\text{ORE}}\) Mapping of parents of leaves to \(\text{ORE}\) regular expressions (pp 113)

\(\mathcal{P}^\text{flat}(D)\) Flat semantics of a CFD \(D\) (pp 64)

\(\mathcal{P}^\text{flat}(D,G)\) Grouped flat products of \(G\) in \(D\) (pp 65)

\(\preceq\) Specialization relation on FMs (pp 53)

\(\text{plev}(\text{RD})\) Non-leaf nodes all of whose children are leaves in a CFD \(D\) (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)
\( \mathcal{P}(D) \) Hierarchical semantics of a CFD \( D \) (pp 72)

\( \mathcal{P}(D, G) \) Grouped hierarchical products of \( G \) in \( D \) (pp 72)

\( \text{rank}(m) \) Rank of a multiset \( m \) (pp 71)

\( \mathcal{RD}(\Sigma) \) The class of all CRDs over the same alphabet \( \Sigma \) (pp 97)

\( \mathcal{RE}(\Sigma) \) The class regular expressions built over \( \Sigma \) (pp 23)

\( \mathcal{R}^+ \) \( \mathcal{R} \mathcal{R}^* \) for a regular expression \( \mathcal{R} \) (pp 24)

\( \mathcal{R}^{CRE}(RD) \) CRE regular expression of a CFD \( D \) (pp 106)

\( \mathcal{R}^{HRE}(OD) \) HRE regular expression of an osCFD \( OD \) (pp 116)

\( \mathcal{R}^{HRE}(OD) \) HRE regular expression of an osCFD \( OD \) (pp 121)

\( \mathcal{R}^{HRE}(f) \) HRE node expression of a node \( f \) in an osCFD (pp 120)

\( \mathcal{R}^{HRE}(G) \) HRE group expression of a group \( G \) in an osCFD (pp 120)

\( m^\circ \) Relaxed multiset of \( m \) (pp 85)

\( U^\circ \) Set of relaxed multisets of \( U \) (pp 85)

\( m[x/y] \) Replacement of an element with another element in a multiset (pp 166)

\( f|_A \) Restriction of a function \( f \) to a subdomain \( A \) (pp 67)

\( R|_A \) restriction of \( R \) on \( A \) (pp 108)

\( \mathcal{R}^{ORE}(ORD) \) ORE regular expression generated for an osCRD \( ORD \) (pp 115)

\( \text{root}(t) \) Root of a tree-like multiset \( t \) (pp 75)
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\subseteq_w)</td>
<td>A partial order on the domain of a word (w) (pp 24)</td>
</tr>
<tr>
<td>(\setminus)</td>
<td>Setminus operation (pp 29)</td>
</tr>
<tr>
<td>(C^{\text{set}})</td>
<td>Set interpretation of a multiplicity domain (C) (pp 96)</td>
</tr>
<tr>
<td>(\text{shr}^{\text{CRE}})</td>
<td>CRE-Shrinking function (pp 106)</td>
</tr>
<tr>
<td>(\text{shr}^{\text{ORE}})</td>
<td>ORE-shrinking function (pp 114)</td>
</tr>
<tr>
<td>(\Sigma(\mathcal{L}))</td>
<td>The alphabet which (\mathcal{L}) is built on (pp 187)</td>
</tr>
<tr>
<td>(\Sigma(\mathcal{R}))</td>
<td>The alphabet which (\mathcal{R}) is built on (pp 187)</td>
</tr>
<tr>
<td>(\simeq)</td>
<td>Refactoring relation on FMs (pp 53)</td>
</tr>
<tr>
<td>(\leq_{\text{seq}})</td>
<td>Subsequence relation on words (pp 24)</td>
</tr>
<tr>
<td>(\subseteq)</td>
<td>Sub-(\text{ppKS}) relation (pp 45)</td>
</tr>
<tr>
<td>(\subseteq)</td>
<td>Subset relation on sets (pp 21)</td>
</tr>
<tr>
<td>(\mapsto_{\text{D}})</td>
<td>Substitution of a leaf node with a CFD (pp 188)</td>
</tr>
<tr>
<td>(\mapsto_{\text{E}})</td>
<td>Substitution of a letter in a regular expression with another expression (pp 188)</td>
</tr>
<tr>
<td>(\mapsto_{\text{L}})</td>
<td>Substitution of a letter in a language with another language (pp 188)</td>
</tr>
<tr>
<td>(\mapsto_{\text{P}})</td>
<td>Substitution of an element in a flat product with another flat product (pp 187)</td>
</tr>
<tr>
<td>(\text{sup}(R))</td>
<td>Supremum of a total order (R) (pp 107)</td>
</tr>
</tbody>
</table>
$\top$ Logical truth (pp 46)

$\mathcal{TH}(A)$ Hierarchy of tree-like multisets over $A$ (pp 74)

$T^f_{\text{OR}}$ Induced subfeature tree by $f$ (pp 41)

$\text{OD}^{\text{crd}}$ Underlying CFD of an osCFD $\text{OD}$ (pp 107)

$\text{ORD}^{\text{crd}}$ Underlying CRD of an osCRD $\text{ORD}$ (pp 108)

$\text{up}(C)$ Upper bound of a multiplicity domain $C$ (pp 96)

$\text{up}(c)$ Upper bound of a multiplicity $c$ (pp 96)

$\sqcup$ Additive union operator on multisets (pp 17)

$\varepsilon$ Empty string (pp 20)

$\emptyset$ Empty regular expression (pp 20)

$\lor$ Logical OR (pp 30)

$\land$ Logical conjunction (pp 15)

$*$ Kleene star (pp 20)

$F_{-r}$ For a set $F$ and $r \in F$ (pp 29)

$\text{flat}_A$ Flattening function over $A$ (pp 73)

$P \rightarrow P'$ 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)
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BL</td>
<td>Boolean Propositional Logic</td>
<td>25</td>
</tr>
<tr>
<td>ppCTL</td>
<td>partial product CTL</td>
<td>46</td>
</tr>
<tr>
<td>CRE</td>
<td>CRDs to Regular Expressions</td>
<td>93</td>
</tr>
<tr>
<td>ppKS</td>
<td>partial product Kripke Structure</td>
<td>44</td>
</tr>
<tr>
<td>HRE</td>
<td>Hierarchical semantics to Regular Expressions</td>
<td>93</td>
</tr>
<tr>
<td>I2C</td>
<td>Instantiate to Completion</td>
<td>35</td>
</tr>
<tr>
<td>ML</td>
<td>Modal Logic</td>
<td>46</td>
</tr>
<tr>
<td>ORE</td>
<td>osCFDs to Regular Expressions</td>
<td>93</td>
</tr>
<tr>
<td>CC</td>
<td>Crosscutting Constraint</td>
<td>15</td>
</tr>
<tr>
<td>CFD</td>
<td>Cardinality-based Feature Diagram</td>
<td>16</td>
</tr>
<tr>
<td>CFM</td>
<td>Cardinality-based Feature Model</td>
<td>17</td>
</tr>
<tr>
<td>CRD</td>
<td>Cardinality-based Regular expression Diagram</td>
<td>93</td>
</tr>
<tr>
<td>FD</td>
<td>basic Feature Diagram</td>
<td>14</td>
</tr>
<tr>
<td>FM</td>
<td>basic Feature Model</td>
<td>14</td>
</tr>
<tr>
<td>FSA</td>
<td>Finite State Automaton</td>
<td>21</td>
</tr>
<tr>
<td>osCFD</td>
<td>Ordered siblings CFD</td>
<td>93</td>
</tr>
<tr>
<td>osCRD</td>
<td>Ordered sibling CRD</td>
<td>93</td>
</tr>
<tr>
<td>PPL</td>
<td>Partial Product Line</td>
<td>30</td>
</tr>
</tbody>
</table>
r.e. \hspace{1cm} \text{Recursively enumerable (pp 22)}