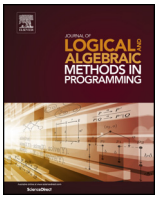
Journal of Logical and Algebraic Methods in Programming 97 (2018) 30–54

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp



A theoretical framework for cardinality-based feature models: The semantics and computational aspects ☆

Aliakbar Safilian *, Tom Maibaum, Zinovy Diskin

Department of Computing and Software, McMaster University, Canada

ARTICLE INFO

Article history:
Received 26 June 2017
Received in revised form 22 February 2018
Accepted 24 February 2018
Available online xxxx

Keywords:
Cardinality-based feature models
Formal language theory
Decidability
Complexity

ABSTRACT

Feature modeling is the most common approach for modeling software product line configurations. We propose a formal language-based formalization for the hierarchical semantics of cardinality-based feature models. We provide a transformation mapping, which allows us to transform a cardinality-based feature diagram to an appropriate regular expression. We propose a formal framework for expressing crosscutting constraints over cardinality-based feature diagrams. We then provide two kinds of semantics for constraints: the flat and the language semantics. We show how to integrate the semantics of diagrams and constraints over them. We also characterize some analysis operations over feature models in terms of operations on languages and discuss the corresponding decidability and computational complexity problems.

1. Introduction

Product line (PL) engineering [5] is a very well-known industrial approach to software/hardware design. A PL is a set of products that share some *commonalities* and exhibit some *variabilities*, where commonalities and variabilities are usually captured using entities called *features*, "system properties that are relevant to some stakeholders" [16]. The idea of this approach is that, instead of producing products individually, the common core of a PL is first produced, leaving a much smaller task to be completed, namely the adaptation of the core to a concrete application requirement. This results in several advantages, including a significant reduction in development time/cost and an increase in reusability [5].

The most common method for modeling PLs is *Feature modeling*. A *feature model* (FM) is a tree of features representing a hierarchical structure of features, which is equipped with some special annotations on the tree's elements showing the *constraints* on features, based on which valid configurations are built. An FM may also be equipped with some additional constraints called *crosscutting constraints* (CCs), a.k.a. *cross-tree constraints* [19]. As the name suggests, these constraints are defined on *incomparable* features. Two features are called incomparable if neither of them is a descendant of the other in the hierarchical structure. We distinguish between these constraints and the tree-structure in a feature model, and call the latter a *feature diagram*.[1]

Feature modeling languages could be divided into *non-attributed* (we call *pure*) and *attributed* FMs. Pure FMs are grouped into *Boolean* and *cardinality-based* FMs. Boolean FMs [28] represent product variability and commonality in terms of Boolean

☆ Financial support was provided by APC via the NECSIS project.

* Corresponding author.
E-mail addresses: safiliaa@mcmaster.ca (A. Safilian), maibaum@mcmaster.ca (T. Maibaum), diskinz@mcmaster.ca (Z. Diskin).

[1] There are some feature diagram variants that represent some simple crosscutting constrains (*inclusive* and *exclusive* constraints involving only two features) [29].

https://doi.org/10.1016/j.jlamp.2018.02.002
2352-2208/

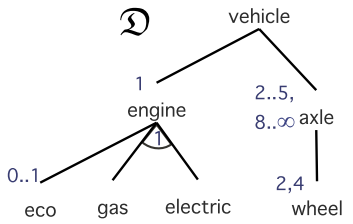


Fig. 1. A cardinality-based feature model for vehicles.

constraints: *optional/mandatory* features, *OR/XOR* decomposition operations, and Boolean CCs. These models are not expressive enough to model PLs that deal not only with the feature types, but also with the number of feature instances. In cardinality-based FMs (CFMs) [16], multiplicity constraints are used to represent the constraints of PLs' configurations. Features in attributed FMs [49] are equipped with some attributes allowing them to model extra quantitative constraints. In this paper, we restrict ourselves to CFMs. Since CFMs subsume Boolean FMs [16],[2] the research reported in this paper can be also applied to Boolean FMs.

Fig. 1 represents a cardinality-based feature diagram (CFD) for a vehicle. A vehicle must have exactly one engine and any number of axles (imagine a futuristic vehicle model), excluding zero, one, six, and seven. The multiplicity constraints "1" and "$2..5, 8..\infty$" on engine and axle, respectively, model these requirements. An engine can be optionally equipped with an ecosystem, represented by the subfeature eco with multiplicity constraint "0..1". Arcs on a set of sibling features equipped with multiplicity constraints model *groups*. An engine can be either gasoline or electric but not both (note the multiplicity constraint "1" on the group {electric, gas}). An axle has either two or four wheels (note the multiplicity constraint $2, 4$ on wheel). This CFD, denoted by $\mathfrak{D}$, is used as a running example throughout the paper. Suppose that our model also needs to satisfy the following requirement: "if an engine is equipped with an ecosystem, then the number of axles must be 4". This cannot be expressed by multiplicity constraints on the diagram's elements and should be considered as a CC. Let us denote the whole CFM (i.e., $\mathfrak{D}$ and the above CC) by $\mathfrak{M}$.

The most common semantics considered for a CFM in the literature is the set of its valid *flat* configurations, where a flat configuration of a CFM is a multiset of features satisfying the constraints of the CFM [16]. As an example, the multiset of features $\{\{\text{vehicle}, \text{engine}, \text{electric}, \text{eco}, \text{axle}^2, \text{wheel}^6\}\}$ is a valid flat configuration of the CFM $\mathfrak{M}$ above.[3] The set of all flat configurations is called the *flat semantics* of the CFM [42].

The flat semantics of a given CFM ignores the hierarchical structure of the CFM. Capturing hierarchical structures of CFMs is important for several reasons: Some analysis questions about the CFM rely explicitly on the hierarchical structure of the CFM, including the least common ancestor (LCA) of a given set of features and determining the root feature [34]. Such analysis questions can be used for extracting some important information of the CFM. For example, the LCA operation comes in handy when one wants to get a smallest subsystem of a system (modeled by an FM) such that it includes some specific properties (features). Some other operations, *specialization* and *refactoring* [50], compare two (or more) given FMs in a semantic sense.[4] Relying on a poor abstraction (like the flat semantics) to define these analysis operations would make their definitions deficient. For instance, consider two very simple Boolean FMs $\mathbf{M}_1$ and $\mathbf{M}_2$ that are defined on the same set of features $\{a, b, c\}$ as follows: All the features in both models are mandatory and their tree-structure are represented by "$a = b^{\uparrow} = c^{\uparrow}$" and "$a = b^{\uparrow}, b = c^{\uparrow}$", respectively. ($f = g^{\uparrow}$ denotes that f is the parent of g.) They are equivalent in the flat semantics. Nevertheless, they represent two different PLs, as c is a constituent of b in the latter while this is not the case in the former. There are also some other challenging tasks in the domain of feature modeling, including *reverse engineering* of FMs and *feature model management*, in which preserving the hierarchical structure of given FMs is essential: It is obvious why preserving the hierarchical structure of FMs is important in reverse engineering, as the output of a reverse engineering task must be (ideally) an FM. Given two FMs, the output of a model management operation (e.g., merge) should satisfy some invariant properties derived from the given models, including the hierarchical structures of the FMs. In light of the above discussion, we define a *faithful semantics* of a CFM as a semantics capturing both the flat semantics and the hierarchical structure of the CFM.

The main contributions of the paper are summarized as follows:

– We provide a faithful semantics for CFDs by using regular languages as the semantic domain. Because of computational properties of regular languages (e.g., their closure properties and complexity class SPACE(O(1))), we can claim that this transformation provides an efficient computational framework for reasoning about CFDs.

– We then propose a formal abstract framework for expressing CCs over CFDs. We provide two kinds of semantics for CCs, flat and language-based (using the class of context-sensitive languages as the semantic domain).

– We characterize some existing analysis operations over CFMs in terms of corresponding operations over languages and discuss their computational aspects, i.e., the corresponding decidability and complexity problems.

[2] This is because any Boolean constraint can be expressed in terms of multiplicities.

[3] We use the notations "{{" and "}}" to identify multisets – for a precise definition, please see Sect. 2.1.

[4] The former investigates their equality and the latter investigates their subsetting relation.

This paper extends and improves our previous paper [43]. The transformation procedure of CFDs to regular expressions proposed in [43] is more complicated than the procedure proposed in the current paper. [43] provides a bottom-up *algorithm* for the transformation: It starts from the leaves of a given CFD and gradually shrinks the CFD by building an intermediate representation, called a *cardinality-based regular expression diagram*, at each step.[5] It keeps doing the shrinking until we get to a singleton tree associated with a regular expression. In the current paper, we propose a *recursive function* for the transformation, which is more structured and thus easier than the approach in [43]. Moreover, in the current paper, we formally show an important property of the transformation called *faithfulness*, which is missing in [43]: We prove that the regular expression generated for a given CFD captures both the hierarchical structure and the flat semantics of the CFD. As for CCs, [43] describes the language theoretic representation of some examples of CCs. However, in the current paper, we propose a formal framework for expressing CCs over CFDs, and provide two kinds of semantics for them, including a language theoretic semantics. Furthermore, [43] investigates the decidability of seven analysis operations over CFMs without providing any formal proofs, whereas the current paper "formally" investigates the decidability of fifteen analysis operations. Moreover, we discuss the complexity classes of the decidable analysis operations in the current article, which is missing in [43]. The current paper also provides a more complete review of related work.

The rest of the paper has been organized as follows. The next section provides some background. In Sect. 3, we show how to transform a given CFD to a faithful regular expression. Sect. 4 discusses CFMs with CCs. In Sect. 5, we investigate the decidability and complexity problems of some analysis operations on CFMs. Related work is discussed in Sect. 6. Sect. 7 concludes the paper with a summary and future work. Proofs of lemmas and theorems can be found in Appendix A.

2. Background

2.1. Preliminary math notations

Let $|A|$ denote the cardinality of a set A. Let $\mathbb{N}$ denote the set of natural numbers. The cardinality of $\mathbb{N}$ is denoted by $\aleph_0$. For a relation $R \subseteq B \times C$ and a set $A \subset B$, the notation $R\big|_A$ is used to denote the restriction of R to A, i.e., $R\big|_A = R \cap (A \times C)$.

We define a set $\mathbb{N}^\infty \stackrel{\text{def}}{=} \mathbb{N} \cup \{\infty\}$ and a binary reflexive transitive relation $\leq_\infty \subset \mathbb{N}^\infty \times \mathbb{N}^\infty$ as follows:
$\forall u, l \in \mathbb{N}^\infty : (l \leq_\infty u) \Leftrightarrow (l, u \in \mathbb{N} \wedge l \leq u) \vee (u = \infty)$. For any $u, l \in \mathbb{N}^\infty$, we use the notation $l <_\infty u$ to denote $(l \leq_\infty u) \wedge (l \neq u)$. Later we will skip subindex ∞ under $<_\infty$ and $\leq_\infty$.

A multiset over a set F is a total function $m : F \to \mathbb{N}$. For any $f \in F$, $m(f)$ is called the multiplicity of f in m. $\mathcal{M}(F)$ denotes the set of all multisets over F. The set $\{f \in F : m(f) > 0\}$ is called the *domain* of m, denoted by $dom(m)$. The multiset m is called finite if $dom(m)$ is finite. 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 $a_i \in dom(m)$. We usually do not write a multiplicity 1 on an element in a multiset, e.g., $\{\{a, b^2\}\} = \{\{a^1, b^2\}\}$. The empty multiset is denoted by $\varnothing$.

The *additive union* operation on multisets, denoted by $\uplus$, is defined as follows: $(m \uplus m')(f) = m(f) + m'(f)$. Given a multiset m over F and a number $n \in \mathbb{N}$, the notation m^n denotes a multiset over F defined as $\forall f \in F : m^n(f) = n \times m(f)$. The binary relation $\preceq$ on $\mathcal{M}(F)$, called the *sub-configuration relation*, is defined as: $m \preceq m'$ iff $\forall f \in dom(m) : m'(f) = m(f)$.

2.2. The flat semantics of CFDs

This subsection formalizes the syntax and flat semantics for CFDs. The following definition formalizes the multiplicity constraints. A multiplicity constraint is usually expressed as a finite sequence $(l_1, u_1) \ldots (l_n, u_n)$, where l_i $(\forall 1 \leq i \leq n)$ is a natural number, u_i $(\forall 1 \leq i \leq n)$ is either a number or ∞ (which denotes the unbounded multiplicity), $l_i \leq u_i$, and $u_i < l_{i+1}$ $(\forall 1 \leq i < n)$. Following UML notations, we visually denote a multiplicity interval (l, u) by $l..u$ (see Fig. 1).

Definition 1 *(Multiplicity constraints [43]).* (i) The set $\mathfrak{C} \stackrel{\text{def}}{=} \{(l, u) \in \mathbb{N} \times \mathbb{N}^\infty : (l \leq u)\}$ is called the *multiplicity-set* and an element $c = (l, u) \in \mathfrak{C}$ is called a *multiplicity*. We call l and u the *lower-bound*, denoted by $low(c)$, and *upper-bound*, denoted by $up(c)$, of c, respectively.

(ii) A finite subset $C \subseteq \mathfrak{C}$ is called a *multiplicity constraint* if there is no overlap between two distinct elements of C, i.e., for any distinct elements $c, c' \in C$, $\{n \in \mathbb{N} : low(c) \leq n \leq up(c)\} \cap \{n \in \mathbb{N} : low(c') \leq n \leq up(c')\} = \varnothing$. Then, C can be enumerated as $C = \{(l_i, u_i)\}_{i \in \{1, \ldots, |C|\}}$ such that $u_i < l_{i+1}$, for all $1 \leq i \leq |C| - 1$. We call l_1 and $u_{|C|}$ the lower-bound, denoted by $low(C)$, and upper-bound, denoted by $up(C)$, of C, respectively. □

Remark 1. To make multiplicity constraints as concise as possible, we may want to replace $u_i < l_{i+1}$ in Definition 1(ii) with $l_{i+1} - u_i > 1$, e.g., instead of $(1, 2)(3, 4)$, we write $(1, 4)$. □

To ease reading, we may use the set equivalent of a multiplicity constraint C, i.e., the set $\{n \in \mathbb{N} | \exists c \in C : low(c) \leq n \leq up(c)\}$. In this sense, the membership relation from $\mathbb{N}$ to $2^{\mathfrak{C}}$ is defined as the set membership.

[5] A cardinality-based regular expression diagram is a CFD whose nodes can be any regular expression built over an alphabet. The number of steps is equal to the depth of the given diagram.

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. The following formalization is a revised version of the syntax of CFDs proposed in [43]: Unlike [43], we consider a solitary feature as a singleton group with multiplicity $(1,1)$.

Definition 2 *(CFDs [43]).* A *cardinality-based feature diagram* (CFD) is a 5-tuple $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$ consisting of the following components.

(i) F is a set of *features*, $r \in F$, and $\_^{\uparrow} : F_{-r} \to F$, where $F_{-r} \stackrel{\text{def}}{=} F \setminus \{r\}$. The tuple $T \stackrel{\text{def}}{=} (F, r, \_^{\uparrow})$ presents a tree in which F is the set of nodes, r is the root, and $f^{\uparrow}$ denotes the parent of f for any $f \in F_{-r}$.

(ii) $\mathcal{G} \subseteq 2^{F_{-r}}$ is a set of *grouped* nodes. All nodes in a group G have the same parent, denoted by $G^{\uparrow}$. All groups in $\mathcal{G}$ are disjoint, i.e., $\forall G, G' \in \mathcal{G}.\ (G \neq G') \Rightarrow (G \cap G' = \varnothing)$. Let us denote the set of groups whose parent is the same, say f, by $\mathcal{G}(f)$. Any subfeature of a given feature $f \in F$ must be included in a group of f, i.e., $\forall f \in F : \{G \in \mathcal{G} : G^{\uparrow} = f\} = f_{\downarrow}$. The only element of a singleton group is called a *solitary* feature.

(iii) $\mathcal{C} \subseteq (F_{-r} \cup \mathcal{G}) \times \mathfrak{C}$ is a left-total relation called the *multiplicity relation*. For any element $e \in F_{-r} \cup \mathcal{G}$, $\mathcal{C}(e)$ represents a multiplicity constraint. For all $G \in \mathcal{G}$, $up(\mathcal{C}(G)) \leq |G|$. In addition, the multiplicity constraint of any singleton group is always $(1,1)$, i.e., $\forall G \in \mathcal{G} : |G| = 1 \implies \mathcal{C}(G) = \{(1,1)\}$. □

Given a CFD $\mathbf{D}$ and a feature $f \in F$, the *diagram induced* by f is a CFD whose tree is the tree under f in $\mathbf{D}$'s tree and all other components are inherited from $\mathbf{D}$. For an example, the diagram induced by engine in $\mathfrak{D}$ is the CFD $\mathbf{D}^{\mathsf{engine}} = (\{\mathsf{engine}, \mathsf{electric}, \mathsf{gas}, \mathsf{eco}\}, \mathsf{engine}, \_^{\uparrow}, \{\mathsf{G}_1 = \{\mathsf{eco}\}, \mathsf{G}_2 = \{\mathsf{electric}, \mathsf{gas}\}\}, \mathcal{C})$, where $\mathsf{electric}^{\uparrow} = \mathsf{gas}^{\uparrow} = \mathsf{eco}^{\uparrow} = \mathsf{engine}$, $\mathcal{C}(\mathsf{G}_1) = \mathcal{C}(\mathsf{G}_2) = \{1\}$, $\mathcal{C}(\mathsf{eco}) = \{0,1\}$, $\mathcal{C}(\mathsf{electric}) = \mathcal{C}(\mathsf{gas}) = \{1\}$.

Definition 3 *(Diagrams induced by nodes [42]).* Let $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$ be a CFD and $f \in F$. The *CFD induced by* f is a CFD $\mathbf{D}^f = (F', f, \_^{\uparrow}\big|_{F'}, \mathcal{G}', \mathcal{C}')$, where $F' = f_{\downarrow\downarrow} \cup \{f\}$, $\mathcal{G}' = \mathcal{G} \cap 2^{F'}$, and $\mathcal{C}' = \mathcal{C}|_{F' \cup \mathcal{G}'}$. □

A *flat configuration* of a given CFD is a multiset of features satisfying the following constraints:

(i) The root is in the domain of the multiset with multiplicity 1, e.g., $m(\mathsf{vehicle}) = 1$ for $\mathfrak{D}$.

(ii) Given a non-root feature f included in m, there are some valid multiplicities $c_1, \ldots, c_n$ in the f's multiplicity constraint, where $n = m(f^{\uparrow})$, such that $m(f) = c_1 + \ldots + c_n$. As an example, supposing that there are two instances of axle in m, we must have either four, six, or eight wheels in m.

(iii) For each instance of the parent of a group, the presence of the group's features must satisfy the associated group multiplicity constraint, e.g., for each instance of engine in m: there must be either electric or gas.

Remark 2. Michel et al. in [36] pointed out that one can interpret the multiplicity constraints on features in two different ways, called *local* and *global* interpretations. The scope of the multiplicity constraint over a feature in the local and global interpretations are the feature's parent and the root (or the whole configuration), respectively. We have followed the local interpretation in which a constraint over a feature f represents the valid number of f's instances for each instance of $f^{\uparrow}$ in a valid configuration (see (ii) above). In the global interpretation, the multiplicity constraint over f denotes the set of valid numbers of instances of f in a valid configuration. From the theoretical and practical points of view, the local interpretation is more interesting than the global one. This is because any PL modeled by a CFM $\mathbf{M}$ based on a global interpretation can be easily modeled by an equivalent CFM $\mathbf{M}'$ based on a local interpretation,[6] but not vice versa. Most CFM approaches follow the local interpretation, e.g., see [17,16,42,19,40,39]. Recently, an extension of CFDs has been proposed in [47], where the scope of the multiplicity constraint over a feature can be defined relatively, i.e., on any ancestor of the feature. Such models are not in the scope of the current paper. □

The flat semantics of CFDs has been formalized in several papers, e.g., [36,43,42,53]. In the following, we provide yet again a *new* recursive definition of flat configurations. In the following definition, a *flat configuration* associated with a group is a multiset over the group's features satisfying the group's multiplicity constraint. For an example, the flat configurations associated with the group $\mathsf{G} = \{\mathsf{electric}, \mathsf{gas}\}$ in Fig. 1 consists of $\{\{\mathsf{gas}\}\}$ and $\{\{\mathsf{electric}\}\}$.

Definition 4 *(Flat configurations).* Given a CFD $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$, a multiset m over F is a *flat configuration* of $\mathbf{D}$ iff:

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

(ii) For all $G \in \mathcal{G}(r)$, there exists a multiset m_G over F, called *a flat configuration associated with group* G, such that $m_G \preceq m$ (i.e., $\forall f \in dom(m_G) : m(f) = m_G(f)$) satisfying the following condition:

[6] $\mathbf{M}'$ could be $\mathbf{M}$ whose feature multiplicity constraints are all replaced by $(0, \infty)$ and the global multiplicities are expressed as CCs.

$\exists S \subseteq G$ with $|S| \in \mathcal{C}(G)$ (selected features from the group)
$\forall f \in S, \exists c_f \in \mathcal{C}(f)$ (valid multiplicities chosen for each feature)
$\forall 1 \leq i \leq c_f, \exists m_{fi} \in \|\mathbf{D}^f\|$ (configurations picked for each feature instance) such that $m_G = \biguplus_{\substack{f \in S \\ 1 \leq i \leq c_f}} m_{fi}$.

The set of flat configurations associated with a group G is denoted by $\|\mathbf{D}, G\|$. The set of flat configurations of $\mathbf{D}$, denoted by $\|\mathbf{D}\|$, is called $\mathbf{D}$'s *flat semantics*. □

Therefore, given a CFD $\mathbf{D}$ with root r whose sets of groups are $\{G_1, \ldots, G_n\}$, a multiset m is a flat configuration of $\mathbf{D}$ iff

$$\forall 1 \leq i \leq n, \exists m_{G_i} \in \|\mathbf{D}, G_i\| : m = \{\{r\}\} \uplus \biguplus_{1 \leq i \leq n} m_{G_i} \tag{1}$$

2.3. Formal languages

This section provides a concise background for some material in formal language theory and introduces some new definitions and uncommon notations for the theory used throughout the paper.

Languages. An *alphabet* Σ is a finite set of symbols. Let Σ^* denote the set of all finite *words* (sequences of occurrences of symbols) built over Σ. The sequence of symbols with length 0 is denoted by ε. For any $\sigma \in \Sigma$, let σ^n denote the word $\sigma \ldots \sigma$ (n occurrences of σ). Any subset of Σ^* is called a *language*. Let $\Sigma(w)$ ($\Sigma(\mathcal{L})$, respectively) denote the set of symbols occurring in a given word w (language $\mathcal{L}$, respectively).

Context-sensitive languages. A language is called *context-sensitive* if it can be generated by some context-sensitive grammar. A grammar is context-sensitive if all of its production rules (a.k.a., productions) are in the form of $\Gamma \rightarrow \Theta$, where Γ and Θ are strings generated over terminals or non-terminals and Θ is not shorter than Γ.

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

Regular languages. A language is *regular* if and only if it can be expressed by some regular grammar. A regular grammar is either a right or left regular grammar. The productions of a right regular grammar must be in one of the following forms: $V \rightarrow \varepsilon$, $V \rightarrow \sigma$, $V \rightarrow \sigma V'$, where σ is a terminal (symbol) and V, V' are non-terminals (symbols). A left regular grammar is defined similarly with replacement of the last production form with $V \rightarrow V'\sigma$. One can also express regular languages via *regular expressions*. *Regular expressions* are defined by the following BNF grammar:

$$\mathcal{R} ::= \varnothing \mid \varepsilon \mid \sigma \text{ (for any } \sigma \in \Sigma) \mid \mathcal{R} + \mathcal{R} \mid \mathcal{R}.\mathcal{R} \mid \mathcal{R}^* \tag{2}$$

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

The language associated with a regular expression $\mathcal{R}$ is denoted by $\mathcal{L}(\mathcal{R})$ and defined in a recursive way as follows:

$$\begin{aligned}
\mathcal{L}(\varnothing) &= \varnothing \\
\mathcal{L}(\varepsilon) &= \{\varepsilon\} \\
\mathcal{L}(\sigma) &= \{\sigma\}, \text{for any } \sigma \in \Sigma \\
\mathcal{L}(\mathcal{R}_1 + \mathcal{R}_2) &= \mathcal{L}(\mathcal{R}_1) \cup \mathcal{L}(\mathcal{R}_2) \\
\mathcal{L}(\mathcal{R}_1.\mathcal{R}_2) = \mathcal{L}(\mathcal{R}_1).\mathcal{L}(\mathcal{R}_2) &= \{w_1.w_2 : w_1 \in \mathcal{L}(\mathcal{R}_1) \wedge w_2 \in \mathcal{L}(\mathcal{R}_2)\} \\
\mathcal{L}(\mathcal{R}^*) = (\mathcal{L}(\mathcal{R}))^* &= \bigcup_{0 \leq i} \mathcal{L}(\mathcal{R})^i, \text{where} \\
\mathcal{L}(\mathcal{R})^0 = \{\varepsilon\}, &\text{ and } \forall 1 \leq i : \mathcal{L}(\mathcal{R})^i = \mathcal{L}(\mathcal{R}).\mathcal{L}(\mathcal{R})^{i-1}
\end{aligned} \tag{3}$$

Definable operations on regular expressions. We propose some definable operations on regular expressions. This will help us to give much more concise regular expressions for CFDs. Given two regular expressions $\mathcal{R}$, $\mathcal{R}'$, and a number $n \in \mathbb{N}$:

$$\begin{aligned}
\mathcal{R}^+ &\stackrel{\text{def}}{=} \mathcal{R}\mathcal{R}^* \\
\mathcal{R}^n &\stackrel{\text{def}}{=} \mathcal{R} \ldots \mathcal{R} \text{ (n repetitions of } \mathcal{R}) \text{ for any } n \in \mathbb{N} \\
\mathcal{R} \,||\, \mathcal{R}' &\stackrel{\text{def}}{=} \mathcal{R}\mathcal{R}' + \mathcal{R}'\mathcal{R}
\end{aligned} \tag{4}$$

Given a regular expression $\mathcal{R}$ and a multiplicity domain $C = \{(l_i, n_i)\}_{1 \leq i \leq n}$:

$$\begin{aligned}
\mathcal{R}^{\langle C \rangle} &\stackrel{\text{def}}{=} \mathcal{R}_1 + \ldots + \mathcal{R}_n, \text{where} \\
\mathcal{R}_i &= \begin{cases} \mathcal{R}^{l_i} + \ldots + \mathcal{R}^{u_i} & \text{if } u_i \neq \infty \\ \mathcal{R}^{l_i}\mathcal{R}^* & \text{otherwise} \end{cases}
\end{aligned} \tag{5}$$

For an example, consider the regular expression $\mathcal{R} = f_1^4 + f_2^*$ and a multiplicity domain $C = \{(3, *)\}$. According to (5), $\mathcal{R}^{\langle C \rangle} = (f_1^4 + f_2^*)^3 (f_1^4 + f_2^*)^*$.

For a given finite set of regular expressions $X = \{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$ and a multiplicity domain C:

$$\begin{aligned} + X &\stackrel{\text{def}}{=} +_{\mathcal{R} \in X} \mathcal{R} = \mathcal{R}_1 + \ldots + \mathcal{R}_n \\ || X &\stackrel{\text{def}}{=} ||_{\mathcal{R} \in X} \mathcal{R} = \mathcal{R}_1 || \ldots || \mathcal{R}_n \\ ||^{\langle C \rangle} X &\stackrel{\text{def}}{=} +_{\substack{X' \subseteq X \\ |X'| \in C}} ||X' \end{aligned} \tag{6}$$

For example, let $X = \{\mathcal{R}, \mathcal{R}'\}$ (for some regular expressions $\mathcal{R}$ and $\mathcal{R}'$) and $C = \{(1, 2)\}$. According to the above equation, $||^{\langle C \rangle} X = \mathcal{R} + \mathcal{R}' + (\mathcal{R} || \mathcal{R}')$.

One-unambiguous regular languages. *One-unambiguous* regular expressions were defined in [10]: Given a regular expression $\mathcal{R}$, let us mark symbols with subscripts to indicate different positions of the same symbol in $\mathcal{R}$. For instance, $(a_1 + b_1)^*\ a_2\ (a_3\ b_2)^*$ is a marking of the expression $(a + b)^*\ a\ (a\ b)^*$. The reverse of marking is dropping the subscripts. Let $\mathcal{R}^\natural$ and $w^\natural$ respectively denote the reverse of a marked expression $\mathcal{R}$ and a marked word w in this sense. We say that a regular expression $\mathcal{R}$ is one-unambiguous iff for each word $w \in \mathcal{L}(\mathcal{R})$, there is only one marked word $v \in \mathcal{L}(\mathcal{R}')$ (for some marking expression $\mathcal{R}'$ of $\mathcal{R}$) such that $v^\natural = w$. For instance, $(b^* + a)^+$ is an one-unambiguous expression, but $(b^* + a)(b^* + a)^*$ is not. A regular language is called one-unambiguous if it can be generated by an one-unambiguous regular expression.

Bounded regular languages. We say a regular language $\mathcal{L}$ is a *bounded* regular language, if there are n words $w_1, \ldots, w_n \in \Sigma^*$ (for some $n \in \mathbb{N}$) such that $\mathcal{L} \subseteq w_1^* \ldots w_n^*$.

Closure properties. The class of regular languages is closed under the set operations union, intersection, complement, and 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.

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

The intersection of a context-free (context-sensitive, respectively) language with a regular language is a context-free language (context-sensitive language, respectively).

Decidability. We say that a given language $\mathcal{L}$ is decidable if there is an algorithm (Turing machine) which decides whether a given word is in $\mathcal{L}$ or not. All recursive languages (including regular, context-free, and context-sensitive languages) are decidable. Below, we state some other decidability results that are used in Sect. 5.2.

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

Given two languages $\mathcal{L}$ and $\mathcal{L}'$, the *equality problem* is to decide whether $\mathcal{L} = \mathcal{L}'$ or not. The equality problem is decidable in the class of regular languages, but undecidable in other classes of formal languages. However, if one of the given languages is a bounded regular and the other is context-free, then the equality problem is still decidable [25].

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

Parikh images. Let $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$. The *Parikh image* (a.k.a. Parikh vector) [38] of a given word $w \in \Sigma^*$ is the vector $(o_1, \ldots, o_n)$ where o_i denotes the number of occurrences of σ_i in w. Clearly, the Parikh image of a word can be seen as a multiset over the alphabet. Thus, we recast the original definition in the following way: The Parikh image of w is the multiset $\{\{\sigma_1^{o_1}, \ldots, \sigma_n^{o_n}\}\}$, where o_i denotes the number of occurrences of σ_i in w.

The Parikh image of a word w (a formal language $\mathcal{L}$, respectively) is denoted by $\mathrm{Par}(w)$ ($\mathrm{Par}(\mathcal{L})$, respectively). By the Parikh image of a regular expression $\mathcal{R}$, denoted by $\mathrm{Par}(\mathcal{R})$, we mean the Parikh image of its language, i.e., $\mathrm{Par}(\mathcal{L}(\mathcal{R}))$.

The following simple results about Parikh images of languages will be used in the proofs of the theorems in Sect. 3. For any two languages $\mathcal{L}_1, \mathcal{L}_2$:

$$\begin{aligned} \mathrm{Par}(\mathcal{L}_1 \mathcal{L}_2) &= \{m_1 \uplus m_2 : m_1 \in \mathrm{Par}(\mathcal{L}_1), m_2 \in \mathrm{Par}(\mathcal{L}_2)\} \\ \mathrm{Par}(\mathcal{L}_1 || \mathcal{L}_2) &= \mathrm{Par}(\mathcal{L}_1 \mathcal{L}_2) \end{aligned} \tag{7}$$

Parikh in [38] proved that the Parikh image of any context-free language, i.e., the set of Parikh images of its words, is equal to the Parikh image of a regular language.

Complexity classes. We briefly introduce the complexity classes used in this article.

PSPACE denotes the class of problems solvable by a deterministic Turing machine in polynomial amount of space. The class of all decision problems that can be solved by a non-deterministic (deterministic, respectively) Turing machine in polynomial time is denoted by NP (P, respectively). NL denotes the class of problems solvable by a nondeterministic Turing machine in logarithmic amount of space. It is well-known that $\mathsf{NL} \subset \mathsf{P} \subseteq \mathsf{NP} \subset \mathsf{PSPACE}$.

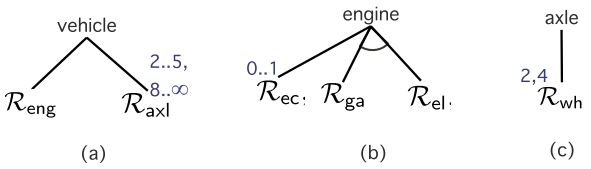


Fig. 2. (a) $\mathfrak{D}$, (b) $\mathfrak{D}^{\mathsf{engine}}$, (c) $\mathfrak{D}^{\mathsf{axle}}$.

For a given complexity class Λ, co-Λ denotes the class of problems X whose complement $\bar{X}$ belongs to Λ. Λ-hard denotes the class of problems X such that any problem in Λ is reducible to X in polynomial time. Λ-complete is the class of problems which are both Λ and Λ-hard.

For a given class Λ, the class P^{Λ} denotes the problems that can be solved by a Turing machine in polynomial time equipped with an oracle of a complete problem in Λ. The notations NP^{Λ} and coNP^{Λ} are defined analogously. A hierarchy of these generalizations of complexity classes, called *the polynomial-time hierarchy*, is defined as follows:

$$\Sigma_0^{\mathsf{P}} = \Delta_0^{\mathsf{P}} = \Pi_0^{\mathsf{P}} = \mathsf{P}$$
$$\Sigma_{i+1}^{\mathsf{P}} = \mathsf{NP}^{\Sigma_i^{\mathsf{P}}}$$
$$\Delta_{i+1}^{\mathsf{P}} = \mathsf{P}^{\Sigma_i^{\mathsf{P}}}$$
$$\Pi_{i+1}^{\mathsf{P}} = \mathsf{coNP}^{\Sigma_i^{\mathsf{P}}}$$

3. Transformation of CFDs to regular expressions

We transform a given CFD to a regular expression $\mathcal{R}$ built over the set of features such that it provides a faithful semantics for the CFD.

3.1. Transformation

Before getting to the formalization, we illustrate the transformation procedure on our running example $\mathfrak{D}$ in Fig. 1. Let us denote the desired (the result of the transformation) regular expression for $\mathfrak{D}$ by $\mathcal{R}_{\mathfrak{D}}$. Also, let $\mathcal{R}_{\mathsf{eng}}$, $\mathcal{R}_{\mathsf{axl}}$, $\mathcal{R}_{\mathsf{ec}}$, $\mathcal{R}_{\mathsf{el}}$, $\mathcal{R}_{\mathsf{ga}}$, and $\mathcal{R}_{\mathsf{wh}}$ denote the desired regular expressions for the diagrams induced by engine, axle, eco, electric, gas, and wheel, respectively.

The multiplicity constraint of a feature is indeed the multiplicity constraint on its corresponding induced diagram. Fig. 2(a) represents the CFD $\mathfrak{D}$ in Fig. 1 in terms of the induced diagrams' regular expressions: The root feature vehicle has two children labeled by $\mathcal{R}_{\mathsf{eng}}$ and $\mathcal{R}_{\mathsf{axl}}$ with the corresponding multiplicity constraints.[7] In this way, a word accepted by the regular expression $\mathcal{R}_{\mathfrak{D}}$ would be a word generated by the following regular expression:

$$\mathsf{vehicle}\ (\mathcal{R}_{\mathsf{eng}} \mid\mid \mathcal{R}_{\mathsf{axl}}^{c_1}), \text{ where } c_1 \in \mathbb{N} \setminus \{0, 1, 6, 7\}.$$

In any word of the expression, the symbol vehicle precedes any other symbols, as it is the root feature. Also the expression $\mathcal{R}_{\mathsf{eng}} \mid\mid \mathcal{R}_{\mathsf{axl}}^{c_1}$ accommodates any permutation of its operands $\mathcal{R}_{\mathsf{eng}}$ and $\mathcal{R}_{\mathsf{axl}}^{c_1}$, as they are siblings. This is the way in which this transformation preserves the hierarchical structure (and also the flat semantics) of the CFD.

To provide a regular expression generating all valid words for the diagram, we need to consider all valid multiplicity constraints on axle. The expression $(\mathcal{R}_{\mathsf{axl}}^2 + \mathcal{R}_{\mathsf{axl}}^3 + \mathcal{R}_{\mathsf{axl}}^4 + \mathcal{R}_{\mathsf{axl}}^5 + \mathcal{R}_{\mathsf{axl}}^8 \mathcal{R}_{\mathsf{axl}}^*)$ properly represent the multiplicity constraint on axle. Thus,

$$\mathcal{R}_{\mathfrak{D}} = \mathsf{vehicle}\ (\mathcal{R}_{\mathsf{eng}} \mid\mid (\mathcal{R}_{\mathsf{axl}}^2 + \ldots + \mathcal{R}_{\mathsf{axl}}^5 + \mathcal{R}_{\mathsf{axl}}^8 \mathcal{R}_{\mathsf{axl}}^*)).$$

Now, consider the diagram induced by engine shown in Fig. 2(b). The node engine has an optional expression $\mathcal{R}_{\mathsf{ec}}$ (the regular expression of the diagram induced by eco) and an XOR group of $\mathcal{R}_{\mathsf{el}}$ and $\mathcal{R}_{\mathsf{ga}}$ (the regular expressions of the diagram induced by electric and gas, respectively). Therefore,

$$\mathcal{R}_{\mathsf{eng}} = \mathsf{engine}\ ((\varepsilon + \mathcal{R}_{\mathsf{ec}}) \mid\mid (\mathcal{R}_{\mathsf{el}} + \mathcal{R}_{\mathsf{ga}})).$$

The choice between ε and $\mathcal{R}_{\mathsf{ec}}$ properly models the optional presence of eco in an engine. The choice between the expressions $\mathcal{R}_{\mathsf{el}}$ and $\mathcal{R}_{\mathsf{ga}}$ models the XOR group. Likewise, the regular expression $\mathcal{R}_{\mathsf{axl}}$ (a regular expression of the diagram induced by axle – see Fig. 2(c)) would be equal to

[7] As a common convention, we assume the multiplicity constraint "1" on an element (either a feature or a group) if no multiplicity constraint is shown on the element.

$$\mathcal{R}_{\mathsf{axl}} = \mathsf{axle}\ (\mathcal{R}^2_{\mathsf{wh}} + \mathcal{R}^4_{\mathsf{wh}}).$$

The regular expressions associated with leaves are primitive expressions, as their corresponding diagrams are singleton, e.g., $\mathcal{R}_{\mathsf{ec}} = \mathsf{eco}$ and $\mathcal{R}_{\mathsf{el}} = \mathsf{electric}$.

The following definitions formally show how to derive the regular expression for a given CFD. See equations (5) and (6) in Sect. 2.3 for the meaning of notations used in the definitions.

Definition 5 *(Regular expressions of CFDs).* Given a CFD $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$, its *regular expression*, denoted by $\mathcal{R}_{\mathbf{D}}$, is recursively defined as follows:

$$\mathcal{R}_{\mathbf{D}} \stackrel{\text{def}}{=} r\ ||_{\mathcal{R}\in\{\mathcal{R}_{\mathbf{D},G}:G\in\mathcal{G}(r)\}}\ \mathcal{R}, \text{ where}$$

$\mathcal{R}_{\mathbf{D},G}$ is called the *regular expression associated with group* G and is defined as:

$$\mathcal{R}_{\mathbf{D},G} = ||^{\langle\mathcal{C}(G)\rangle}\ \{\mathcal{R}_{\mathbf{D}^f}{}^{\langle\mathcal{C}(f)\rangle} : f \in G\} \qquad \square$$

Remark 3. According to the above definition, the regular expression associated with a solitary feature f (i.e., a singleton group) would be $\mathcal{R}_{\mathbf{D},f} = \mathcal{R}_{\mathbf{D}^f}{}^{\langle\mathcal{C}(f)\rangle}$. $\square$

Therefore, given a CFD $\mathbf{D}$ with root r whose set of groups is $\{G_1, \ldots, G_n\}$, the regular expression corresponding to $\mathbf{D}$ would be:

$$\mathcal{R}_{\mathbf{D}} = r\ (\mathcal{R}_{\mathbf{D},G_1}\ ||\ \ldots\ ||\ \mathcal{R}_{\mathbf{D},G_n}) \tag{8}$$

3.2. Faithfulness

We show that the regular expression built from a given CFD provides a faithful semantics for the CFD: the Parikh image of its language is equal to the flat semantics of the CFD and it also captures the hierarchy of features based on the ordering between the features in the language of the expression (Theorems 2 and 5, respectively). The proofs of the lemmas and theorems can be found in Appendix A.1. The following lemma is used in the proof of Theorem 2. By the depth of a CFD, we mean the depth of its underlying tree.

Lemma 1. *Suppose that for any CFD with depth less than N ($N \in \mathbb{N}$), the Parikh image of its associated regular expression is equal to its flat semantics. Now, given a CFD* $\mathbf{D}$ *with depth N and a group G:* $\mathrm{Par}(\mathcal{R}_{\mathbf{D},G}) = \|\mathbf{D}, G\|$. $\square$

The following theorem shows that $\mathcal{R}_{\mathbf{D}}$ captures the flat semantics of a given CFD $\mathbf{D}$.

Theorem 2. *For a given CFD* $\mathbf{D}$, $\mathrm{Par}(\mathcal{R}_{\mathbf{D}}) = \|\mathbf{D}\|$. $\square$

Theorem 5 shows that the hierarchical structure of a given CFD $\mathbf{D}$ can be retrieved from $\mathcal{R}_{\mathbf{D}}$. The idea is that a feature f is a subfeature of another feature f' iff, for any word in the language, any occurrences of f is preceded by some occurrences of f'. This ordering is defined in Definition 6.

Definition 6 *(Symbol ordering in words and languages).*

(i) (*Symbol ordering in words*). Given an alphabet Σ and a word w, we define a transitive irreflexive relation $\sqsubseteq^{\Sigma}_{w} \subseteq \Sigma \times \Sigma$ as follows: $\forall\sigma, \sigma' \in \Sigma(w) : \sigma \sqsubseteq_w \sigma'$ iff any occurrences of σ' is preceded by some occurrences of σ in w.

(ii) (*Symbol ordering in languages*). Given an alphabet Σ and a language $\mathcal{L}$, we define a transitive irreflexive relation $\sqsubseteq^{\Sigma}_{\mathcal{L}} \subseteq \Sigma \times \Sigma$ as follows: $\forall\sigma, \sigma' \in \Sigma(\mathcal{L}) : (\sigma \sqsubseteq^{\Sigma}_{\mathcal{L}} \sigma') \Leftrightarrow (\forall w \in \mathcal{L} : \sigma \sqsubseteq^{\Sigma}_{w} \sigma')$. This implies that $\sqsubseteq^{\Sigma}_{\mathcal{L}} = \bigcap_{w\in\mathcal{L}} \sqsubseteq^{\Sigma}_{w}$. $\square$

As an example, let $\Sigma = \{\sigma_1, \sigma_2, \sigma_3\}$, and $w_1 = \sigma_1^2\sigma_2$, $w_2 = \sigma_1\sigma_2^3\sigma_3^2$, $w_3 = \sigma_2$, $w_4 = \varepsilon$, $w_5 = \sigma_1\sigma_2\sigma_1\sigma_3\sigma_2$, and $\mathcal{L}_1 = \{w_1, w_2\}$, $\mathcal{L}_2 = \{w_3, w_5\}$, $\mathcal{L}_3 = \{w_{1-5}\}$. According to the above definition:

$$\sqsubseteq^{\Sigma}_{w_1} = \{(\sigma_1,\sigma_2), (\sigma_1,\sigma_3), (\sigma_2,\sigma_3)\},$$
$$\sqsubseteq^{\Sigma}_{w_2} = \{(\sigma_1,\sigma_2), (\sigma_2,\sigma_3)\},$$
$$\sqsubseteq^{\Sigma}_{w_3} = \{(\sigma_2,\sigma_1), (\sigma_2,\sigma_3), (\sigma_1,\sigma_3), (\sigma_3,\sigma_1)\},$$
$$\sqsubseteq^{\Sigma}_{w_4} = \Sigma\times\Sigma \setminus \{(\sigma,\sigma) : \sigma \in \Sigma\},$$
$$\sqsubseteq^{\Sigma}_{w_5} = \{(\sigma_1,\sigma_2), (\sigma_1,\sigma_3), (\sigma_2,\sigma_3)\}, \text{ and}$$
$$\sqsubseteq^{\Sigma}_{\mathcal{L}_1} = \sqsubseteq^{\Sigma}_{w_1} \cap \sqsubseteq^{\Sigma}_{w_2} = \{(\sigma_1,\sigma_2), (\sigma_2,\sigma_3)\},$$

$$\sqsubseteq^{\Sigma}_{\mathcal{L}_2} = \sqsubseteq^{\Sigma}_{w_3} \cap \sqsubseteq^{\Sigma}_{w_5} = \{(\sigma_2, \sigma_3), (\sigma_1, \sigma_3)\},$$
$$\sqsubseteq^{\Sigma}_{\mathcal{L}_3} = \bigcap_{1 \le i \le 5} \sqsubseteq^{\Sigma}_{w_i} = \{(\sigma_2, \sigma_3)\}.$$

For simplicity, we write $\sqsubseteq_w$ and $\sqsubseteq_{\mathcal{L}}$ to denote $\sqsubseteq^{\Sigma(w)}_{w}$ and $\sqsubseteq^{\Sigma(\mathcal{L})}_{\mathcal{L}}$, respectively. Given a regular expression $\mathcal{R}$ and an alphabet Σ, we mean $\sqsubseteq_{\mathcal{L}(\mathcal{R})}$ and $\sqsubseteq^{\Sigma}_{\mathcal{L}(\mathcal{R})}$ by $\sqsubseteq_{\mathcal{R}}$ and $\sqsubseteq^{\Sigma}_{\mathcal{R}}$, respectively.

We need a couple of lemmas to prove Theorem 5. In the following lemma, $com(\mathcal{L})$ for a given language $\mathcal{L}$ denotes the set of symbols common in all words in the language, i.e., $com(\mathcal{L}) = \{\sigma \in \Sigma(\mathcal{L}) : \forall w \in \mathcal{L}, \sigma \in \Sigma(w)\}$. By $com(\mathcal{R})$ for a given regular expression $\mathcal{R}$, we mean $com(\mathcal{L}(\mathcal{R}))$.

Lemma 3. *For any regular expressions $\mathcal{R}$ and $\mathcal{R}'$, $n \in \mathbb{N}$, and $C \in \mathfrak{C}$, the following statements hold. Let $\Sigma = \Sigma(\mathcal{R}) \cup \Sigma(\mathcal{R}')$.*
(i) $\sqsubseteq_{\mathcal{R}+\mathcal{R}'} = \sqsubseteq^{\Sigma}_{\mathcal{R}} \cap \sqsubseteq^{\Sigma}_{\mathcal{R}'}$
(ii) $\Sigma(\mathcal{R}) \cap \Sigma(\mathcal{R}') = \varnothing \Longrightarrow \sqsubseteq_{\mathcal{R}+\mathcal{R}'} = \sqsubseteq_{\mathcal{R}} \cup \sqsubseteq_{\mathcal{R}'}$
(iii) $\Sigma(\mathcal{R}) \cap \Sigma(\mathcal{R}') = \varnothing \Longrightarrow \sqsubseteq_{\mathcal{R}.\mathcal{R}'} = \sqsubseteq_{\mathcal{R}} \cup \sqsubseteq_{\mathcal{R}'} \cup (com(\mathcal{R}) \times com(\mathcal{R}'))$
(iv) $\Sigma(\mathcal{R}) \cap \Sigma(\mathcal{R}') = \varnothing \Longrightarrow \sqsubseteq_{\mathcal{R}||\mathcal{R}'} = \sqsubseteq_{\mathcal{R}} \cup \sqsubseteq_{\mathcal{R}'}$
(v) $\sqsubseteq_{\mathcal{R}} = \sqsubseteq_{\mathcal{R}^n} = \sqsubseteq_{\mathcal{R}^*} = \sqsubseteq_{\mathcal{R}^{\langle C \rangle}}$

Note that the equations in the above lemma involve two regular expressions. However, they can be easily generalized to any number of regular expressions.

Lemma 4. *For any CFD* $\mathbf{D}$ *and* $G \in \mathcal{G}_{\mathbf{D}}$: $\sqsubseteq_{\mathcal{R}_{\mathbf{D},G}} = \bigcup_{f \in G} \sqsubseteq_{\mathcal{R}_{\mathbf{D}f}}$. □

The following theorem shows that, for a given CFD $\mathbf{D}$, $\mathcal{R}_{\mathbf{D}}$ preserves $\mathbf{D}$'s hierarchy.

Theorem 5. *For any given CFD* $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$*, the following statement holds:* $\forall f, g \in F : g \in f_{\downarrow\downarrow} \Leftrightarrow f \sqsubseteq_{\mathcal{L}} g$*, where* $\mathcal{L} = \mathcal{L}(\mathcal{R}_{\mathbf{D}})$. □

4. Crosscutting constraints

As the name suggests, *crosscutting constraints* (CCs) over a CFD are additional constraints on the corresponding PL, which are usually defined on some incomparable features. In Sect. 4.1, we propose a formal language for expressing CCs over CFDs. We provide two kinds of semantics for constraints: the flat and the language semantics. We then discuss the expressiveness of our language in Sect. 4.2. Finally, Sect. 4.3 shows how to integrate the semantics of CFDs and CCs defined over them to provide both a flat and a language semantics for CFMs.

4.1. A formal framework for CCs

Given a feature $f \in F$, we use the notation $\#f$ to denote the number of f's instances in a multiset built over F. In our language, there are atomic CCs of the form $(n_1 \times \#f_1) < (n_2 \times \#f_2)$, where $n_1 \in \mathbb{N}, n_2 \in \mathbb{N}^{\infty}$ and $f_{1,2}$ are features. The above expression has an intuitive meaning: "The number of instances of f_1 multiplied by n_1 is less than the number of instances of f_2 multiplied by n_2". When $n_i = 1$ $(i \in \{1, 2\})$, we simply drop n_i in the expression. We build compound CCs by using Boolean connectives $\vee, \wedge, \longrightarrow$, and $\neg$.

The following definition formalizes the syntax of the language. For simplicity, we use the notation ${f_1}^{n_1} < {f_2}^{n_2}$, where $f_{1,2}$ are features and $n_{1,2}$ are numbers, to denote an atomic constraint $(n_1 \times \#f_1) < (n_2 \times \#f_2)$.

Definition 7 *(CCs: syntax).* Given a finite set of features F, a CC over F is an expression defined via the following BNF:

$$\phi := {f_1}^{n_1} < {f_2}^{n_2} \mid \phi \vee \phi \mid \neg\phi, \text{where } (n_1, n_2) \in \mathbb{N} \times \mathbb{N}^{\infty} \text{ and } f_{1,2} \in F$$

We denote the family of all such CCs over F by $\Phi(F)$. We propose two definable connectives $\longrightarrow$ (implication) and $\wedge$ (conjunction) as follows: $\phi_1 \longrightarrow \phi_2 \stackrel{\text{def}}{=} \neg\phi_1 \vee \phi_2$ and $\phi_1 \wedge \phi_2 \stackrel{\text{def}}{=} \neg(\neg\phi_1 \vee \neg\phi_2)$. □

Since there is no notion of hierarchy between features in a CC, the semantic domain of CCs in $\Phi(F)$ is faithfully the powerset of the set of multisets built over F. The following definition provides a formal semantics for CCs in $\Phi(F)$, called the flat semantics of $\Phi(F)$.

Definition 8 *(CCs: flat semantics).* We recursively define a semantic function (mapping from the syntax domain to the semantic domain), $\|\_\| : \Phi(F) \longrightarrow 2^{\mathcal{M}(F)}$ as follows. For any $n_{1,2} \in \mathbb{N}$, $f_{1,2} \in F$, and $\phi_{1,2} \in \Phi(F)$:

$$\|{f_1}^{n_1} < {f_2}^{n_2}\| = \begin{cases} \{m \in \mathcal{M}(F) : n_1 \times m(f_1) < n_2 \times m(f_2)\} & \text{if } n_2 \in \mathbb{N} \\ \mathcal{M}(F) & \text{if } n_2 = \infty \end{cases}$$

$$\|\phi_1 \vee \phi_2\| = \|\phi_1\| \cup \|\phi_2\|$$

$$\|\neg\phi_1\| = \mathcal{M}(F) \setminus \|\phi_1\| \qquad \square$$

Now, we are going to provide a language semantics for $\Phi(F)$. The language of a CC must not impose any specific order between features, as the CC has nothing to do with a hierarchy of features. Therefore, any word built over the set of features whose Parikh image (as a multiset) satisfies the CC is a valid word. Thus, the language of a CC is always a *commutative language* [13]. (Commutative grammars/languages are a formalism for generating multisets.) That is, given a CC $\phi \in \Phi(F)$, the language of ϕ, denoted by $\mathcal{L}(\phi)$, would be equal to

$$\mathcal{L}(\phi) = \{w \in F^* : \text{Par}(w) \in \|\phi\|\}. \tag{9}$$

Definition 9 *(CCs: language semantics).* Given a set of features F, the language semantics of CCs in $\Phi(F)$ is recursively defined as follows. For any $n_{1,2} \in \mathbb{N}$, $f_{1,2} \in F$, and $\phi_{1,2} \in \Phi(F)$:

$$\mathcal{L}({f_1}^{n_1} < {f_2}^{n_2}) = \begin{cases} \{w \in F^* : \#_w(f_1) \times n_1 < \#_w(f_2) \times n_2\} & \text{if } n_2 \in \mathbb{N} \\ F^* & \text{if } n_2 = \infty \end{cases}$$

$$\mathcal{L}(\phi_1 \vee \phi_2) = \mathcal{L}(\phi_1) \cup \mathcal{L}(\phi_2)$$

$$\mathcal{L}(\neg\phi_1) = F^* \setminus \mathcal{L}(\phi_1) \qquad \square$$

The following theorem shows that the class of CCs in $\Phi(F)$ is a subclass of context-sensitive languages. (See the proof in Appendix A.2.)

Theorem 6. *For a given CC $\phi \in \Phi(F)$, its language, i.e., $\mathcal{L}(\phi)$, is in the class of commutative context-sensitive languages.* $\square$

4.2. Discussion

In the following, we discuss the expressiveness of our proposed CC language $\Phi(F)$ over a set of features F.

4.2.1. Expressiveness of $\Phi(F)$

Our language supports inequality constraints $n_1 \times \#f < n_2 \times \#f'$ and their compositions via logical connectives. The equality relation is definable as follows: $({f_1}^{n_1} = {f_2}^{n_2}) \stackrel{\text{def}}{=} \neg({f_1}^{n_1} < {f_2}^{n_2}) \wedge \neg({f_2}^{n_2} < {f_1}^{n_1})$, which means that $n_1 \times \#f_1 = n_2 \times \#f_2$. Obviously, the relation $\leq$ is also definable: $({f_1}^{n_1} \leq {f_2}^{n_2}) \stackrel{\text{def}}{=} ({f_1}^{n_1} < {f_2}^{n_2}) \vee ({f_1}^{n_1} = {f_2}^{n_2})$.

We can also express constraints of the form $n \square \#f$, where $\square \in \{<, >, =, \leq, \geq\}$ over a CFD on F with root feature r as $r^n \square f$. (Note that the $\#r = 1$ in any configuration of the CFD.) We may want to write $n \square f$ for simplicity.

Our CC language allows for the expression of any Boolean constraints. A Boolean constraint over F is indeed a propositional logic formula built over F (as a set of atomic propositions). Boolean CCs deal with only feature types. In this sense, the validation of a feature f (i.e., an atomic proposition) in a multiset m is interpreted as True iff $f \in dom(m)$ (i.e., f is present in m). As an example, the Boolean constraint $f_1 \wedge f_2 \rightarrow \neg f_3$ over a CFD states that the presence of f_1 and f_2 in a configuration of the CFD must exclude the presence of f_3 in the configuration. To translate a given Boolean formula ψ built over F to an expression in $\Phi(F)$, we just need to replace each feature (atomic proposition) f in ψ with the expression $r^1 \leq f^1$ (i.e., $1 \leq \#f$), where r is the root feature. (As already discussed, we may want to denote it by $1 \leq f$ for simplicity.) As an example, the above Boolean constraint $f_1 \wedge f_2 \rightarrow \neg f_3$ is expressed as $(1 \leq f_1) \wedge (1 \leq f_2) \rightarrow \neg(1 \leq f_3)$, which is semantically equal to $(1 \leq f_1) \wedge (1 \leq f_2) \rightarrow (f_3 = 0)$.

4.2.2. Strengths and limitations

In this subsection, we aim mainly at comparing our CC language with other languages that are proposed to express CCs over CFMs in the literature. The best known CCs over CFDs are simple Boolean CCs $f \rightarrow f'$ and $f \rightarrow \neg f'$, called *inclusive* and *exclusive* CCs, respectively [23,29,40]. As already discussed, our language generalizes these kinds of CCs by supporting any arbitrary Boolean constraints.

Inclusive and exclusive constraints were generalized to deal with multiplicities as $(n_1, n_2) f_1 \longrightarrow (n_3, n_4) f_2$ [40,53] and $(n_1, n_2) f_1 \longrightarrow \neg(n_3, n_4) f_2$ [53], respectively. The former expresses that "if there exists at least n_1 and at most n_2 instances of f_1 in a configuration, then there must be at least n_3 and at most n_4 instances of f_2 in the configuration". The latter expresses that "if there exists at least n_1 and at most n_2 instances of f_1 in a configuration, then the number of instances of f_2 in the configuration must not be between n_3 and n_4", and vice versa. In our language, these expressions are expressed as $\phi \rightarrow \psi$ and $\phi \rightarrow \neg\psi$, respectively, where $\phi = (n_1 \leq f_1) \wedge (f_1 \leq n_2)$ and $\psi = (n_3 \leq f_2) \wedge (f_2 \leq n_4)$.

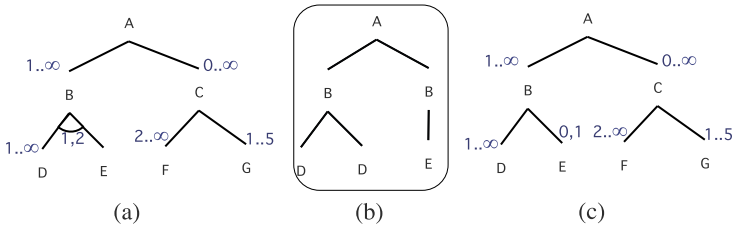


Fig. 3. (a) A CFD **D**, (b) a part of an instance of **D**, (c) a CFD.

Our language further generalizes inclusion and exclusion constraints by building more complicated constraints via logical connectives. For example, consider the following constraints expressible in our language, which, to our best knowledge, are not supported in the existing formal approaches.

$[(n_1, n_2)f_1 \longrightarrow (n_3, n_4)f_2] \ \vee \ [(n'_1, n'_2)f_1 \longrightarrow (n'_3, n'_4)f_2]$

$(n_1, n_2)f_1 \wedge (n_3, n_4)f_2 \rightarrow (n_5, n_6)f_3$

$(n_1, n_2)f_1 \rightarrow (n_3, n_4)f_2 \vee (n_5, n_6)f_3$

$(n_1, n_2)f_1 \wedge \neg(n_3, n_4)f_2 \rightarrow (n_5, n_6)f_3$

$(n_1, n_2)f_1 \rightarrow (n_3, n_4)f_2 \vee \neg(n_5, n_6)f_3$

[19] proposed CCs expressing equality between the number of feature instances, e.g., $\#f = \#f'$. Our language generalizes such constraints by allowing them to deal with multiplicities, e.g., $n \times \#f = n' \times \#f'$ and $(n_1, n_2)\#f = (n'_1, n'_2)\#f'$. Moreover, our language supports the inequality relation between the number of feature instances. Our language further generalizes such constraints by allowing their composition via logical connectives.

[36] have "informally" discussed some potential CCs over CFDs, including constraints dealing with quantifiers over the set of instances of specific features. Some examples of constraints dealing with the universal quantifier can be found in [19]. Let us call these constraints CCs with *specific scopes*, as their scopes are specific features rather than the root feature. Consider the CFD in Fig. 3(a), denoted by **D**, with the following CC: (CC_1): "for any instance of D, there must be two instances of F" (D is the scope of CC_1). To express CCs with specific scopes in a very straightforward way, one needs to identify the feature instances and use quantifiers over them. However, CCs like CC_1, which involve incomparable features, can be faithfully expressed using multiplicity constraints without quantifiers over instances: CC_1's language semantics must be commutative, since D and F are incomparable. Therefore, it is faithfully reduced to $CC'_1 : 2 \times \#D \leq \#F$", which is expressible in our CC language. Nevertheless, CCs with specific scopes involving comparable features may not be faithfully expressed in our CC language. For an example, consider the following CC over **D**: (CC_2): "There must be at least one instance of D associated with each instance of B". We cannot express CC_2 as a CC dealing with multiplicity constraint $CC'_2 : \#B \leq \#D$: See Fig. 3(b), which satisfies CC'_2, but there is an instance of B (the rightmost instance of B) whose set of children has no instance of D.

Remark 4. Scope CCs like CC_2, which rely on a specific hierarchical structure of features, cannot be expressed independently and yet faithfully. By dependency here, we mean dependency on a specific hierarchical structure. Therefore, we believe that such specific scope CCs should be directly expressed as a part of the corresponding CFD. In our example, we can simply model the CFM "**D**+ CC_2" by the CFD in Fig. 3(c). However, to address specific scope CCs in which the scope feature is not an immediate parent of the involved features, we need to extend the definition of CFDs. One such extension has been recently proposed in [47] in which multiplicity constraints are defined *relatively*, i.e., the scope of a multiplicity constraint on a feature is not necessarily its immediate parent, but can be any antecedent of the feature (see also Remark 2).

4.3. Integration of CCs and CFDs

In this subsection, we provide flat and language semantics for CFMs by integrating the semantics of CFDs and CCs. Obviously, a flat (language, respectively) semantics of a given CFM would be the intersection of the flat (language, respectively) semantics of its underlying CFD and CCs. Thus, given a CFM $\mathbf{M} = (\mathbf{D}, \phi)$, where **D** is a CFD over a set of features F and $\phi \in \Phi(F)$, its flat and language semantics, respectively denoted by $\|\mathbf{M}\|$ and $\mathcal{L}(\mathbf{M})$, would be equal to

$$\begin{aligned} \|\mathbf{M}\| &= \|\mathbf{D}\| \cap \|\phi\| \\ \mathcal{L}(\mathbf{M}) &= \mathcal{L}(\mathcal{R}_{\mathbf{D}}) \cap \mathcal{L}(\phi) = \{w \in \mathcal{L}(\mathcal{R}_{\mathbf{D}}) : \mathrm{Par}(w) \in \|\phi\|\} \end{aligned} \tag{10}$$

The language of a CFD **D** (i.e., $\mathcal{L}(\mathcal{R}_{\mathbf{D}})$) is always regular.

Since the intersection of a context-free (context-sensitive, respectively) language with a regular language is a context-free language (context-sensitive language, respectively), the type of $\mathcal{L}(\mathbf{M})$ for a given CFM $\mathbf{M} = (\mathbf{D}, \phi)$ is given by the type of $\mathcal{L}(\phi)$, as $\mathcal{L}(\mathcal{R}_{\mathbf{D}})$ is always regular.

Consider a computational hierarchy of the classes of formal languages (one is shown in the right-hand side of Fig. 4). Reflection of the language hierarchy into the domain of feature models provides a computational hierarchy of feature models (see the left-hand side of Fig. 4). This hierarchy guides us in how to constructively analyze CFMs. (See Sect. 5.) Note that

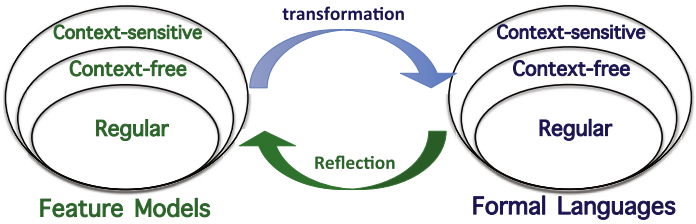


Fig. 4. A computational hierarchy of CFMs.

the class of all Boolean feature models is a subclass of regular feature models, since the product family of a Boolean feature model is always finite.

5. Analysis of CFMs: decidability and complexity

5.1. Analysis questions in the language framework

As far as we know, some of the analysis operations discussed here are not defined in the literature, including finite feature models, valid global multiplicity, dynamic refactoring, dynamic specialization, and disjointness. In the following, let $\mathbf{M}$ and $\mathbf{M}'$ be CFMs over a set of features F, $m \in \mathcal{M}(F)$, $f \in F$, $n \in \mathbb{N}$, and $F' \subset F$.

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 multiset of features as inputs and decides whether the latter is a valid flat configuration of the CFM or not. In the language framework, "m is a valid configuration of $\mathbf{M}$ iff $\exists w \in \mathcal{L}(\mathbf{M}) : \text{Par}(w) = m$".

Partial Configuration: This operation takes a CFM and a multiset of features as inputs and decides whether the latter is a valid *partial configuration* of the CFM or not. A multiset m is a partial configuration of the CFM if there exists a flat configuration m' of the CFM such that $\forall a \in dom(m) : m(a) = m'(a)$. In the language framework, "m is a valid partial configuration of $\mathbf{M}$ iff $\mathcal{L} \cap \mathcal{L}(\mathbf{M}) \neq \varnothing$, where $\mathcal{L} = \{w \in F^* | \forall f \in dom(m) : \text{Par}(w)(f) = m(f)\}$".

Core Features: The *core features* operation takes a CFM and returns the set of features that are included in all configurations of the CFM. In the language framework, "f is a core feature in $\mathbf{M}$ iff $\mathcal{L}(\mathbf{M}) \subseteq \mathcal{L}(F^* f F^*)$".

Valid Global Multiplicity: This operation takes a CFM, a feature, and a natural number as inputs and decides whether there is a valid configuration of the CFM such that the feature's multiplicity in the configuration is equal to the given number or not. If yes, then we call the given number a *valid global multiplicity* of the feature. In the language framework, "n is a valid global multiplicity of f in $\mathbf{M}$ iff $\mathcal{L}_n \cap \mathcal{L}(\mathbf{M}) \neq \varnothing$, where $\mathcal{L}_0 = \Sigma^*$, $\mathcal{L}_{i+1} = \mathcal{L}_i \ f \ \Sigma^*$, and $\Sigma = F \setminus \{f\}$".

Void Feature Model: This operation takes a CFM as input and decides whether its flat semantics is empty or not. In the language framework, "$\mathbf{M}$ is void iff $\mathcal{L}(\mathbf{M}) = \varnothing$".

Finite Feature Model: Given a CFM $\mathbf{M}$, this operation decides whether it represents a finite product line or not. Note that $\mathbf{M}$ may have some unbounded multiplicity constraints on some features, and yet, due to some CCs over its CFD, its set of configurations is finite. This operation is called *false unboundedness* in [53]. In the language framework, "$\mathbf{M}$ is a finite feature model iff $\mathcal{L}(\mathbf{M})$ is finite".

Dead Feature: The *dead feature* operation takes a CFM and a feature and decides whether the feature is *dead* in the CFM or not. A feature f in a CFM $\mathbf{M}$ is called dead if $\nexists m \in \|\mathbf{M}\|$ such that $f \in dom(m)$. In the language framework, "f is dead in $\mathbf{M}$ iff $\mathcal{L}(F^* f F^*) \cap \mathcal{L}(\mathbf{M}) = \varnothing$".

Common Ancestors: The *common ancestor* operation takes a CFD and a set of features and returns their common ancestor features. In the language framework, "f is a common ancestor of F' in $\mathbf{M}$ iff $\forall w \in \mathcal{L}(\mathbf{M}), \forall f' \in F' : f \sqsubseteq_w f'$".

Least Common Ancestor: This operation takes a CFD and a set of features and returns their lowest common ancestor feature. In the language framework, "f is the least common ancestor of F' in $\mathbf{M}$ iff $\forall w \in \mathcal{L}(\mathbf{M}), \forall l \in L : f \sqsubseteq_w l$, where L denotes the set of common ancestors of F'".

Some operations answer some questions about the relationships between two CFMs in the semantic sense:

(Dynamic) Refactoring: The *refactoring* (*dynamic refactoring*, respectively) operation takes two CFMs and decides whether their flat (language, respectively) semantics are equal or not. In the language framework, "$\mathbf{M}'$ is a refactoring (dynamic refactoring, respectively) of $\mathbf{M}$ iff $\text{Par}(\mathcal{L}(\mathbf{M})) = \text{Par}(\mathcal{L}(\mathbf{M}'))$ ($\mathcal{L}(\mathbf{M}) = \mathcal{L}(\mathbf{M}')$, respectively)".

(Dynamic) Specialization: The *specialization* (*dynamic specialization*, respectively) operation takes two CFMs $\mathbf{M}_1$ and $\mathbf{M}_2$ as inputs and decides whether the flat (language, respectively) semantics of $\mathbf{M}_1$ is a subset of the flat (language, respectively) semantics of $\mathbf{M}_2$ or not. In the language framework, "$\mathbf{M}'$ is a specialization (dynamic specialization, respectively) of $\mathbf{M}$ iff $\text{Par}(\mathcal{L}(\mathbf{M})) \subset \text{Par}(\mathcal{L}(\mathbf{M}'))$ ($\mathcal{L}(\mathbf{M}) \subset \mathcal{L}(\mathbf{M}')$, respectively)".

(Dynamic) Disjointness: The *disjointness* (*dynamic disjointness*, respectively) operation takes two CFMs as inputs and decides whether the flat (language, respectively) semantics of the two CFMs are disjoint or not. In the language framework, "$\mathbf{M}'$ and $\mathbf{M}$ are disjoint iff $\text{Par}(\mathcal{L}(\mathbf{M}')) \cap \text{Par}(\mathcal{L}(\mathbf{M})) = \varnothing$ ($\mathcal{L}(\mathbf{M}') \cap \mathcal{L}(\mathbf{M}) = \varnothing$, respectively)".

5.2. Decidability

Please find the proofs in Appendix A.3.

Theorem 7. *Given a CFM, the operations Valid Configuration, Common Ancestors, and Least Common Ancestor are decidable.* □

Theorem 8. *Given a context-free CFM* $\mathbf{M}$*, the operations Partial Configuration, Core Features, Valid global Multiplicity, Void Feature Model, Finite Feature Model, and Dead Feature are decidable. However, none of them is decidable in the class of context-sensitive non-context-free CFMs.* □

Theorem 9. *Given two CFMs* $\mathbf{M}_1$ *and* $\mathbf{M}_2$*, the following statements hold:*
(i) If both are context-free, then the Refactoring problem is decidable.
(ii) If both are regular, then the Dynamic Refactoring problem between them is decidable.
(iii) If $\mathbf{M}_1$ *and* $\mathbf{M}_2$ *are regular and context-free, respectively, then the Dynamic Refactoring problem is decidable iff* $\mathbf{M}_1$ *is bounded regular.* □

Remark 5. 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 10. *Given two CFMs* $\mathbf{M}_1$ *and* $\mathbf{M}_2$*, the following statements hold:*
(i) If $\mathbf{M}_1$ *and* $\mathbf{M}_2$ *are context-free, then both the Specialization and Disjointness problems are decidable.*
(ii) If $\mathbf{M}_1$ *and* $\mathbf{M}_2$ *are regular and context-free, respectively, then the problems "is* $\mathbf{M}_2$ *a dynamic specialization of* $\mathbf{M}_1$*?" and "are* $\mathbf{M}_1$ *and* $\mathbf{M}_2$ *dynamically disjoint?" are decidable.* □

5.3. Complexity

The proofs of the results can be found in Sect. A.4.

Proposition 11. *The Valid Configuration problem in the regular, context-free, and context-sensitive CFMs are* P, NP-complete, *and* PSPACE-complete, *respectively.* □

Remark 6. Nevertheless, the Valid Configuration problem would be P in a subclass of context-sensitive CFMs, those models whose languages are mildly context-sensitive, as such context-sensitive languages are parsable in polynomial time [22]. □

The Partial Configuration and Valid Global Multiplicity are both reduced to the intersection non-emptiness problem.[8]

Proposition 12. *The operations Partial Configuration and Valid Global Multiplicity are* NL *and* P *problems in the class of regular and context-free CFMs, respectively.* □

Proposition 13. *The Dead Feature operation is a* NL *and* P *problem in the class of regular and context-free CFMs, respectively.* □

Proposition 14. *The Core Feature operation is a* P *problem in the class of regular CFMs. The lower bound time complexity of the operation in the class of context-free CFMs is exponential.* □

Proposition 15. *The Void Feature Model operation in the class of context-free CFMs is a* P *problem.* □

Proposition 16. *The Finite Feature Model operation in the class of context-free CFMs is a* P *problem.* □

Proposition 17. *The Common Ancestor operation is a* P *problem in all classes of CFMs.* □

Proposition 18. *The Refactoring operation is a* coNP-complete *and* Π_2^{P}-complete *problem in the class of regular and context-free CFMs, respectively.* □

Proposition 19. *The Specialization operation is a* P *and* coNP-complete *problem in the class of regular and context-free CFMs, respectively.* □

[8] The intersection non-emptiness problem of two languages $\mathcal{L}$ and $\mathcal{L}'$ is defined as $\mathcal{L} \cap \mathcal{L}' \neq \varnothing$.

Proposition 20. *The Disjointness operation is a* coNP *and* Π_2^P *problem in the class of regular and context-free CFMs, respectively.* □

Proposition 21. *Given two regular CFMs, the dynamic refactoring problem over them is in the* CSL-complete *class.* □

Remark 7. The Dynamic Refactoring problem is also decidable if one of the given CFMs is context-free and the other is bounded regular (see Theorem 9(iii)). We do not know the complexity class of this problem. However, fixing the bounded regular CFM, the problem would be in the NP-hard class. [27]. □

Proposition 22. *The lower bound time complexity of the Dynamic Specialization problem is exponential. However, it is* PSPACE-complete *when it is restricted to the regular CFMs.* □

Proposition 23. *The Dynamic Disjointness operation is a* P *problem if one of the given CFMs is context-free and the other is regular. However, it is a* NL *problem if restricted to the class of regular CFMs.* □

6. Related work

In this section, we discuss some related approaches that deal with formal semantics and/or automated analysis of CFMs. Sect. 6.1 discusses formal grammar-based approaches for modeling CFMs. In Sect. 6.2, we discuss some approaches that are based on multiset theory. Sect. 6.3 discusses some CSP-based approaches. Finally, in Sect. 6.4, we discuss some UML-based approaches for modeling variability in PLs.

6.1. Grammar-based approaches

Czarnecki et al., in [16], were the first formalizing the semantics of CFDs using context-free grammars. The generative grammar for a given CFD captures the flat semantics of the CFD. However, it gives a left-to-right ordering on siblings (the nodes with the same parent). Such an ordering forces two syntactically equivalent feature diagrams to have different language semantics. In addition, such an ordering on siblings results in the generative grammars not capturing the hierarchy of diagrams, 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). Also, the authors have not considered CCs in their theory, which is important, as CCs play an important role in feature modeling. Moreover, their approach may result in ambiguous grammars, which makes them bad candidates for the semantics of feature modeling.

6.2. Multiset-based approaches

[36] formalizes the syntax of CFDs and propose a multiset-based semantics for them. However, their approach does not support CCs. They have informally discussed some potential CCs over CFDs, including CCs with specific scopes (see Sect. 4.2).

[42] proposes another multiset based theory for a given CFD, which is called the *hierarchical theory* of the CFD. The theory is based on a defined hierarchy of multisets over the set of features. The hierarchical theory of the CFD is a subset of this hierarchy. The authors also showed that the theory captures all information of the diagram (it even semantically distinguishes between the grouped and solitary features). This semantics provides a promising theoretical framework to address some challenging issues in feature model management and reverse engineering of CFDs. However, the authors have not considered CCs in their theory.

[53] formalizes a valid configuration of a given CFM **M** by a tuple $(m, <)$, where m is a multiset of features and $<$ is declared as a binary relation over m, which aims to preserve the hierarchical structure of the CFM between the instances of features in m.[9] As for CCs, their formalization support inclusion and exclusion CCs of the forms $\phi : (n_1, n_2) f_1 \rightarrow (n_3, n_4) f_2$ and $\phi' : (n_1, n_2) f_1 \rightarrow \neg(n_3, n_4) f_2$, respectively. However, our language for expressing CCs over CFDs is much more expressive than the language proposed in [53] (see Sect. 4.2 for a detailed discussion). It is easy to see that the language interpretations of ϕ and ϕ' are always regular,[10] i.e., the CC language in [53] is a subclass of regular languages. Note that, according to Theorem 6, the class of CCs supported in our language is a subclass of context-sensitive languages. Also, there are even some regular CCs expressible in our language, but not expressible in the language of [53] (take for example the expressions $f_1 \rightarrow f_2 \vee f_3$ and $(n_1, n_2) f_1 \rightarrow f_2 \vee (n_3, n_4) f_3$). In other words, the class of CFMs supported in [53] is a subclass of our regular CFMs.

A couple of anomaly questions were discussed in [53] and its accompanying tool paper [44], including *consistent* CFMs (a given CFM is consistent iff it is not void) and *false unboundedness* on a feature (a given CFM has false unboundedness

[9] However, the definition of $<$ over m does not seem precise, as a multiset does not distinguish between different instances of an element. An element of a multiset m is a tuple $(f, m(f))$.

[10] $\phi_1 : n < \#f < n'$ is regular for any feature f and any numbers n, n'. Since regular languages are closed under union and complement, any expression of the forms $(n_1, n_2) f_1 \rightarrow (n_3, n_4) f_2$ and $\phi' : (n_1, n_2) f_1 \rightarrow \neg(n_3, n_4) f_2$ are regular.

if there is a feature with upper bound multiplicity ∞, but the number of the CFM's configurations is finite). The false unboundedness problem is reducible to the Finite Feature Model operation discussed in Sect. 5, and vice versa. The authors claim that any semantic inconsistency, including the above questions, can be expressed via a notion called *dead multiplicity* (a multiplicity n on a feature f or group is called dead if there is no valid configuration of the CFM in which the multiplicity of f is n). Clearly, this problem can be reduced to our Valid Global Multiplicity problem. Accordingly, they have defined an interesting notion called normal forms of CFMs by "narrowing its declared multiplicity constraints down to actual ones" [53]. As for automated bound analysis (e.g., false unboundedness), the article proposes using ILP-solvers (ILP stands for Integer Linear Programming). If the interval under investigation is unbounded (i.e., the upper bound is ∞), the problem can be reduced to the Finite Feature Model operation. Otherwise, it is reduced to the Valid Global Multiplicity operation. Papadimitriou in [37] showed that Integer Programming is in the NP class. We have shown that the Finite Feature Model operation (rephrasing of the false unboundedness operation) is a P problem in our framework (see Proposition 16). Recall that we have shown that the Valid Global Multiplicity operation is an NL problem in the class of regular CFMs (see Proposition 12). Thus, their solution for bound analysis is not computationally better than ours. They have also studied another anomaly in CFMs called interval gaps, which could be seen as a generalization of the dead multiplicity problem (instead of checking a single multiplicity, a finite multiplicity interval is considered). The authors proposed using SMT-solvers to address this problem. This problem can be reduced to a finite number (equal to the length of the given interval) of concrete Valid Global Multiplicity problems. Since the Valid Global Multiplicity problem in our framework is an NL problem in the class of regular CFMs, the interval gap analysis problem would be in P in our language framework. On the other hand, it is very well-known that SAT is an NP-complete problem [11]. Therefore, our solution for addressing interval gap analysis would not be computationally worse than the solution proposed in [44,53].

6.3. CSP-based approaches

Dhungana et al. in [19] propose a semi-interactive approach for flat configuration of CFMs by transforming the constraints of a given CFM to a generative constraint satisfaction problem (GCSP).[11] However, it is not formally shown how to do this transformation (abstract models of CFMs and CCs over them were not even proposed). Also, the computational complexity of this approach would be an issue, as mentioned in the paper.

[29] proposes the use of constraint logic programming solvers to address some analysis questions about CFMs. Boolean CCs and some CCs dealing with number of feature instances are supported in this paper. However, as they mentioned, their approach cannot address questions about infinite PLs.

In [40], a given CFM is translated to a constraint satisfaction problem (CSP) and then CSP-solvers are used to address some analysis questions over the CFM, e.g., the number of configurations, and validity of a configuration. Simple Boolean inclusion and exclusion CCs involving only two features have been defined/used in this paper. However, our CC language generalizes Boolean constraints (see Sect. 4.2). Also, two kinds of CCs dealing with multiplicities have been formally defined: $(n_1, n_2)f_1 \rightarrow (n_3, n_4)f_2$ and $f_1 \rightarrow op(n_3, n_4)f_2$, where op denotes a mathematical operator like $\times$. The former is clear and it is generalized in our CC language. By the latter, as far as we understood, the authors mean "$(\#n_1\ op\ \#f_1) \leq \#f_2 \leq (\#n_2\ op\ \#f_1)$".[12] As we saw in Sect. 4.2, such constraints are expressible in our language. Moreover, we have generalized them by allowing Boolean composition of them.

6.4. UML class diagrams

UML class diagrams have been also used to model variability in PLs. In the following, we discuss some related work.

As far as we know, Czarnecki et al. in [17] were the first who proposed translation from CFDs to UML class diagrams. However, the translation process was not precisely presented/defined. As the authors argue, this translation allows them to use OCL as a language for expressing the multiplicity constraints. Nevertheless, as the transformation of a given CFD to a class diagram was not formally defined, the authors did not show how to automatically generate the corresponding OCL constraints. Moreover, "OCL is much more general purpose and is not tailored for CFM specific constraints" [36]. Indeed, we first need an abstract model of CCs and then show how to transform them to OCL, which is missing in this work. The authors created a prototype, which extends a feature model plug-in in Eclipse [14], to address a couple of analysis questions over CFMs: computing the number of configurations of a given CFM, and propagating configuration choices (an interactive tool for helping the user to configure a CFM).

Gomez and Ramos in [23] present a metamodel for CFMs and provide an automated approach to transform a given CFM to a domain variability model (which represents the CFM as a class diagram). This transformation allows them to use existing MDE and generative programming approaches [15] to support the development of PLs. However, they supported a restricted form of CCs over CFDs. The CCs supported in their approach include Boolean inclusion/exclusion CCs involving only two incomparable features, and *Use* constraints between two incomparable features. A Use constraint f_1Use $(n_1, n_2)f_2$ is interpreted as "for each instance of f_1 there are between n_1 and n_2 instances of f_2". Note that all these constraints are expressible in our CC language (see Sect. 4.2). They have shown how to translate them to appropriate OCL constraints.

[11] GCSP [20] is an extension of the classical constraint satisfaction problem (CSP) designed to address the dynamic aspects of problems.

[12] See also [39], where the authors use the same language for expressing CCs.

Clafer [6] is a general-purpose modeling language, which combines the concepts of feature modeling, metamodeling, and class diagrams. Clafer uses Alloy, Z3 SMT, and CSP-solvers as underlying reasoning mechanisms.

7. Conclusion

7.1. Summary

In this paper, we provided a set-theoretic recursive definition for the flat semantics of CFMs. This semantics can be used to address a large number of analysis questions about a given CFM. However, as discussed in the introduction, it does not capture all useful information about the CFM. To overcome this problem, we proposed a language-based semantics for CFMs.

We provided a recursive function transforming a given CFD to a regular expression. It was shown that both the flat semantics and the hierarchical structure of the CFD can be retrieved from its regular expression. Therefore, it provides a faithful semantics for the CFD.

We surveyed the literature to determine what kinds of CCs practitioners may need to express over CFDs. We then proposed a formal framework for expressing CCs. As shown, this language subsumes all the current languages for expressing CCs. Two kinds of semantics were provided for this language: the flat and the language semantics. By intersecting the flat semantics (the language semantics, respectively) of a given CFD and that of the CCs expressed over the CFD, we provided the flat semantics (the language semantics, respectively) for the whole CFM.

We also characterized some existing analysis operations over CFMs in terms of the language framework. This allows us to use some off-the-shelf language tools to do analysis on CFMs, which is generally thought to be a challenging issue. We also investigated the decidability problems of the introduced analysis operations for different kinds of CFMs. Moreover, we investigated the complexity problems of decidable analysis questions.

7.2. Other applications and future work

We discuss several concrete tasks in feature modeling, which would benefit from using the language view of CFMs.

Automated analysis. Automated analysis of CFMs is still an open problem [7,40]. We believe that our approach provides a promising framework to address this problem. There are several off-the-shelf language tools, including HKC [9], LIBVATA [32], RABIT [1], ALASKA [18], GOAL [51], FSA6 [52], FAT [24], JFLAP [41], and [21], 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.

Reverse engineering. As far as we know, the current state of the art approaches for reverse engineering (e.g., [46]) are heuristic-based and work only for Boolean feature modeling. The reason for making them heuristic based is mainly caused by using a poor abstract view of feature models, the flat semantics. The language semantics discussed in this article could be used in the reverse engineering of CFMs (and Boolean feature models as well), as the language semantics of a given CFM captures all essential information about the CFM, including the flat semantics and the hierarchical structure of the CFM. To this end, we need to propose a procedure to retrieve the hierarchical structure and the constraints from the language associated with an CFM. We believe that the proof of the faithfulness theorem guides us in how to do so.

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. [45,2,3]. Given two CFMs, the output of each of these management operations (e.g., merge) should satisfy some invariant properties derived from the models, including the hierarchical structures of the models. 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.

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 [4,55,8]. However, none of them perfectly addresses the problem and the problem is still open and challenging. CFMs can be interpreted as UML class models [17]. Indeed, UML class models could be considered as a generalization of CFMs. We believe that the research reported in the paper can be generalized to address the connection between metamodels and formal languages.

Acknowledgements

We are very grateful to anonymous reviewers of our FM'15 paper for useful comments. Special thanks go to the editor and anonymous reviewers of this journal version for careful reading, numerous useful comments, suggestions, and stimulating criticism.

Appendix A. Proofs

A.1. Proofs of faithfulness

Lemma 1. *Suppose that for any CFD with depth less than N ($N \in \mathbb{N}$), the Parikh image of its associated regular expression is equal to its flat semantics. Now, given a CFD* $\mathbf{D}$ *with depth N and a group G:* $\mathrm{Par}(\mathcal{R}_{\mathbf{D},G}) = \|\mathbf{D}, G\|$.

Proof. Let $G = \{f_1, \ldots, f_n\}$ for some $n \in \mathbb{N}$ and $m \in \|\mathbf{D}, G\|$. We show that $m \in \mathrm{Par}(\mathcal{R}_{\mathbf{D},G})$. Without loss of generality, we suppose that the features f_{1-k} for some $k \in \mathcal{C}(G)$ have been selected in m. Therefore, according to Definition 4, there are $c_i \in \mathcal{C}(f_i)$ (for all $1 \le i \le k$) and $m_{ij} \in \|\mathbf{D}^{f_i}\|$ (for all $1 \le j \le c_i$) such that $m = \biguplus_{1\le i\le k} \biguplus_{1\le j\le c_i} m_{ij}$.

$$
\begin{aligned}
&\Longrightarrow \text{(due to hypothesis[13])}\ \forall\langle 1 \le i \le k\rangle, \forall\langle 1 \le c_i \le k\rangle : m_{ij} \in \mathrm{Par}(\mathcal{R}_{\mathbf{D}^{f_i}}) \\
&\Longrightarrow \forall 1 \le i \le k : (\biguplus_{1\le j\le c_i} m_{ij}) = m_{i1} \uplus \ldots \uplus m_{ic_i} \in \mathrm{Par}(\mathcal{R}^{c_i}_{\mathbf{D}^{f_i}}) \\
&\Longrightarrow \forall 1 \le i \le k, \text{ since } c_i \in \mathcal{C}(f_i) : \biguplus_{1\le j\le c_i} m_{ij} \in \mathrm{Par}(\mathcal{R}_{\mathbf{D}^{f_i}}{}^{\langle \mathcal{C}(f_i)\rangle}) \\
&\Longrightarrow m \in \mathrm{Par}(\mathcal{R}_{\mathbf{D}^{f_i}}{}^{\langle \mathcal{C}(f_i)\rangle} \ \ldots\ \mathcal{R}_{\mathbf{D}^{f_k}}{}^{\langle \mathcal{C}(f_k)\rangle}) \\
&\Longrightarrow \text{(according to (7))}\ m \in \mathrm{Par}(\mathcal{R}_{\mathbf{D}^{f_i}}{}^{\langle \mathcal{C}(f_i)\rangle} \,||\, \ldots \,||\, \mathcal{R}_{\mathbf{D}^{f_k}}{}^{\langle \mathcal{C}(f_k)\rangle}) \\
&\Longrightarrow \text{(since } k \in \mathcal{C}(G))\ m \in ||^{\langle \mathcal{C}(G)\rangle}\{\mathcal{R}(\mathbf{D}^f)^{\langle \mathcal{C}(f)\rangle} : f \in G\} \\
&\Longrightarrow \text{(according to Definition 5) } m \in \mathrm{Par}(\mathcal{R}_{\mathbf{D},G}).
\end{aligned}
$$

To show that $m \in \|\mathbf{D}, G\|$ for any $m \in \mathrm{Par}(\mathcal{R}_{\mathbf{D},G})$, we just need to reverse the above proof. □

Theorem 2. *For a given CFD* $\mathbf{D}$, $\mathrm{Par}(\mathcal{R}_{\mathbf{D}}) = \|\mathbf{D}\|$.

Proof. We prove this theorem by the following inductive reasoning on the depth of CFDs.
(*base case*): For any singleton CFD $\mathbf{D}$ with root r, $\mathrm{Par}(\mathcal{R}_{\mathbf{D}}) = \mathrm{Par}(r) = \{\ \{\{r\}\}\ \} = \|\mathbf{D}\|$.
(*hypothesis*): Assume that for any CFD $\mathbf{D}$ with depth less than N, $\mathrm{Par}(\mathcal{R}_{\mathbf{D}}) = \|\mathbf{D}\|$.
(*inductive step*): We show that for any CFD $\mathbf{D}$ with depth N, $\mathrm{Par}(\mathcal{R}_{\mathbf{D}}) = \|\mathbf{D}\|$.
Consider an arbitrary CFD with depth N and root r whose set of groups is $\{G_1, \ldots, G_n\}$. According to (8),

$$\mathcal{R}_{\mathbf{D}} = r\ (\mathcal{R}_{\mathbf{D},G_1} \,||\, \ldots \,||\, \mathcal{R}_{\mathbf{D},G_n}) \tag{A.1}$$

Consider a word $w \in \mathcal{L}(\mathcal{R}_{\mathbf{D}})$. According to (A.1),

$$
\begin{aligned}
&\mathrm{Par}(w) \in \mathrm{Par}(r\ (\mathcal{R}_{\mathbf{D},G_1} \,||\, \ldots \,||\, \mathcal{R}_{\mathbf{D},G_n})) \\
&\Longrightarrow \text{(according to (7))}\ \mathrm{Par}(w) \in \mathrm{Par}(r\ \ \mathcal{R}_{\mathbf{D},G_1}\ \ \ldots\ \ \mathcal{R}_{\mathbf{D},G_n}) \\
&\Longrightarrow [\forall 1 \le t \le n, \exists w_{G_t} \in \mathcal{R}_{\mathbf{D},G_t}] : \mathrm{Par}(w) = \mathrm{Par}(r\ \ w_{G_1}\ \ \ldots\ \ w_{G_n}) \\
&\Longrightarrow \mathrm{Par}(w) = \{\{r\}\} \uplus \biguplus_{1\le i\le n} \mathrm{Par}(w_G)
\end{aligned}
$$

According to Lemma 1 and the hypothesis together, $m_{G_t} \overset{\text{def}}{=} \mathrm{Par}(w_{G_t}) \in \|\mathbf{D}, G_t\|$ for any $1 \le t \le n$.

$$
\begin{aligned}
&\Longrightarrow \mathrm{Par}(w) = \{\{r\}\} \uplus \biguplus_{1\le t\le n} m_{G_t} \\
&\Longrightarrow \text{(according to (1))}\ \mathrm{Par}(w) \in \|\mathbf{D}\|
\end{aligned}
$$

To show that $m \in \mathrm{Par}(\mathcal{R}_{\mathbf{D}})$ for any $m \in \|\mathbf{D}\|$, we just need to reverse the above proof. □

Lemma 3. *For any regular expressions* $\mathcal{R}$ *and* $\mathcal{R}'$, $n \in \mathbb{N}$, *and* $C \in \mathfrak{C}$, *the following statements hold. Let* $\Sigma = \Sigma(\mathcal{R}) \cup \Sigma(\mathcal{R}')$.
(i) $\sqsubseteq_{\mathcal{R}+\mathcal{R}'} \ = \ \sqsubseteq^{\Sigma}_{\mathcal{R}} \cap \sqsubseteq^{\Sigma}_{\mathcal{R}'}$
(ii) $\Sigma(\mathcal{R}) \cap \Sigma(\mathcal{R}') = \varnothing \Longrightarrow \sqsubseteq_{\mathcal{R}+\mathcal{R}'} \ = \ \sqsubseteq_{\mathcal{R}} \cup \sqsubseteq_{\mathcal{R}'}$
(iii) $\Sigma(\mathcal{R}) \cap \Sigma(\mathcal{R}') = \varnothing \Longrightarrow \sqsubseteq_{\mathcal{R}.\mathcal{R}'} \ = \ \sqsubseteq_{\mathcal{R}} \cup \sqsubseteq_{\mathcal{R}'} \cup (com(\mathcal{R}) \times com(\mathcal{R}'))$
(iv) $\Sigma(\mathcal{R}) \cap \Sigma(\mathcal{R}') = \varnothing \Longrightarrow \sqsubseteq_{\mathcal{R}||\mathcal{R}'} \ = \ \sqsubseteq_{\mathcal{R}} \cup \sqsubseteq_{\mathcal{R}'}$
(v) $\sqsubseteq_{\mathcal{R}} \ = \ \sqsubseteq_{\mathcal{R}^n} \ = \ \sqsubseteq_{\mathcal{R}^*} \ = \ \sqsubseteq_{\mathcal{R}^{\langle C\rangle}}$.

[13] Since the depth of $\mathbf{D}^{f_i}$ $(1 \le i \le k)$ is less than N.

Proof. Note that $\Sigma(\mathcal{R}_1 \star \mathcal{R}_2) = \Sigma(\mathcal{R}_1) \cup \Sigma(\mathcal{R}_2) = \Sigma$, where $\star \in \{+, ., ||\}$.

Proof of (i): $\forall \sigma_1, \sigma_2 \in \Sigma$:

$$\sigma_1 \sqsubseteq_{\mathcal{R}+\mathcal{R}'} \sigma_2 \Leftrightarrow (\forall w \in \mathcal{L}(\mathcal{R}+\mathcal{R}') : \sigma_1 \sqsubseteq^{\Sigma}_{w} \sigma_2) \Leftrightarrow$$
$$(\forall w \in \mathcal{L}(\mathcal{R}) : \sigma_1 \sqsubseteq^{\Sigma}_{w} \sigma_2) \wedge (\forall w \in \mathcal{L}(\mathcal{R}') : \sigma_1 \sqsubseteq^{\Sigma}_{w} \sigma_2) \Leftrightarrow$$
$$(\sigma_1 \sqsubseteq^{\Sigma}_{\mathcal{R}} \sigma_2) \wedge (\sigma_1 \sqsubseteq^{\Sigma}_{\mathcal{R}'} \sigma_2)$$

This implies that $\sqsubseteq_{\mathcal{R}+\mathcal{R}'} = \sqsubseteq^{\Sigma}_{\mathcal{R}} \cap \sqsubseteq^{\Sigma}_{\mathcal{R}'}$.

Proof of (ii): According to (i), $\sqsubseteq_{\mathcal{R}+\mathcal{R}'} = \sqsubseteq^{\Sigma}_{\mathcal{R}} \cap \sqsubseteq^{\Sigma}_{\mathcal{R}'}$. According to Definition 6:

$$\sqsubseteq^{\Sigma}_{\mathcal{R}} = \sqsubseteq_{\mathcal{R}} \cup A \cup B,$$
$$\sqsubseteq^{\Sigma}_{\mathcal{R}'} = \sqsubseteq_{\mathcal{R}'} \cup A' \cup B', \text{ where}$$
$$A = \{(f, f') : f \in \Sigma(\mathcal{R}), f' \in \Sigma(\mathcal{R}')\},$$
$$B = \Sigma(\mathcal{R}')^2 \setminus \{(f', f') : f' \in \Sigma(\mathcal{R}')\}$$
$$A' = \{(f', f) : f \in \Sigma(\mathcal{R}), f' \in \Sigma(\mathcal{R}')\},$$
$$B' = \Sigma(\mathcal{R})^2 \setminus \{(f, f) : f \in \Sigma(\mathcal{R})\}$$

Since $\Sigma(\mathcal{R}) \cap \Sigma(\mathcal{R}') = \varnothing$:

$$A \cap A' = B \cap B' = A \cap B' = A' \cap B = A \cap \sqsubseteq_{\mathcal{R}'} = A' \cap \sqsubseteq_{\mathcal{R}} = \varnothing$$
$$B \cap \sqsubseteq_{\mathcal{R}'} = \sqsubseteq_{\mathcal{R}'}, \quad B' \cap \sqsubseteq_{\mathcal{R}} = \sqsubseteq_{\mathcal{R}} \tag{A.2}$$
$$\sqsubseteq_{\mathcal{R}} \cap \sqsubseteq_{\mathcal{R}'} = \varnothing$$

(A.2) implies that

$$A \cap \sqsubseteq^{\Sigma}_{\mathcal{R}'} = \varnothing \text{ and } A' \cap \sqsubseteq^{\Sigma}_{\mathcal{R}} = \varnothing$$
$$B \cap \sqsubseteq^{\Sigma}_{\mathcal{R}'} = \sqsubseteq_{\mathcal{R}'} \text{ and } B' \cap \sqsubseteq^{\Sigma}_{\mathcal{R}} = \sqsubseteq_{\mathcal{R}} \tag{A.3}$$

Since $\Sigma(\mathcal{R}) \cap \Sigma(\mathcal{R}') = \varnothing$, according to (A.2) and (A.3), $\sqsubseteq^{\Sigma}_{\mathcal{R}} \cap \sqsubseteq^{\Sigma}_{\mathcal{R}'} = \sqsubseteq_{\mathcal{R}} \cup \sqsubseteq_{\mathcal{R}'}$.

Proof of (iii): For any words w and w' with $\Sigma(w) \cap \Sigma(w') = \varnothing$, and $\sigma_{1,2} \in \Sigma(w) \cup \Sigma(w')$: $\sigma_1 \sqsubseteq_{w.w'} \sigma_2$ iff one the following statements hold:

- $\sigma_{1,2} \in \Sigma(w) \wedge \sigma_1 \sqsubseteq_{w} \sigma_2$.
- $\sigma_{1,2} \in \Sigma(w') \wedge \sigma_1 \sqsubseteq_{w'} \sigma_2$.
- $\sigma_1 \in \Sigma(w) \wedge \sigma_2 \in \Sigma(w')$.

$$\Longrightarrow \sqsubseteq_{w.w'} = \sqsubseteq_{w} \cup \sqsubseteq_{w'} \cup \{(\sigma_1 \in \Sigma(w), \sigma_2 \in \Sigma(w'))\}. \tag{A.4}$$
$$\forall \sigma_1, \sigma_2 \in \Sigma : \ \sigma_1 \sqsubseteq_{\mathcal{R}.\mathcal{R}'} \sigma_2 \Leftrightarrow$$
$$(\forall w \in \mathcal{L}(\mathcal{R}.\mathcal{R}') : \sigma_1 \sqsubseteq_{w} \sigma_2) \Leftrightarrow \tag{A.5}$$
$$(\forall w \in \mathcal{L}(\mathcal{R}), \forall w' \in \mathcal{L}(\mathcal{R}') : \sigma_1 \sqsubseteq_{w.w'} \sigma_2)$$

According to (A.4) and (A.5), (iii) holds.

Proof of (iv):

$$\sqsubseteq_{\mathcal{R}||\mathcal{R}'} =$$
$$\sqsubseteq_{\mathcal{R}.\mathcal{R}'+\mathcal{R}'.\mathcal{R}} \overset{\text{according to (i)}}{=} \sqsubseteq^{\Sigma}_{\mathcal{R}.\mathcal{R}'} \cap \sqsubseteq^{\Sigma}_{\mathcal{R}'.\mathcal{R}} = \sqsubseteq_{\mathcal{R}.\mathcal{R}'} \cap \sqsubseteq_{\mathcal{R}'.\mathcal{R}}$$
$$\overset{\text{(iii)}}{=} \sqsubseteq_{\mathcal{R}} \cup \sqsubseteq_{\mathcal{R}'} \cup [(\text{com}(\mathcal{R}) \times \text{com}(\mathcal{R}')) \cap (\text{com}(\mathcal{R}') \times \text{com}(\mathcal{R}))]$$
$$= \sqsubseteq_{\mathcal{R}} \cup \sqsubseteq_{\mathcal{R}'}$$

Proof of (v): Obviously, $\forall \sigma_{1,2} \in \Sigma(\mathcal{R})$: $\sigma_1 \sqsubseteq_{w} \sigma_2 \Leftrightarrow \sigma_1 \sqsubseteq_{w^n} \sigma_2$.

$$\forall \sigma_{1,2} \in \Sigma(\mathcal{R}) : \ \sigma_1 \sqsubseteq_{\mathcal{R}} \sigma_2 \Leftrightarrow$$
$$(\forall w \in \mathcal{L}(\mathcal{R}) : \sigma_1 \sqsubseteq_{w} \sigma_2) \Leftrightarrow$$
$$(\forall w \in \mathcal{L}(\mathcal{R}) : \sigma_1 \sqsubseteq_{w^n} \sigma_2) \Leftrightarrow \tag{A.6}$$
$$(\forall w \in \mathcal{L}(\mathcal{R}^n) : \sigma_1 \sqsubseteq_{w} \sigma_2) \Leftrightarrow$$
$$\sigma_1 \sqsubseteq_{\mathcal{R}^n} \sigma_2$$

Since $\mathcal{R}^* = +_{0\leq i}\ \mathcal{R}^i$, according to (i) and (A.6), $\sqsubseteq_{\mathcal{R}^*}=\sqsubseteq_{\mathcal{R}}$.
Since $\mathcal{R}^{\langle C\rangle} = +_{i\in C}\ \mathcal{R}^i$, according to above and (i), $\sqsubseteq_{\mathcal{R}^{\langle C\rangle}}=\bigcap_{i\in C}\sqsubseteq_{\mathcal{R}^i}=\sqsubseteq_{\mathcal{R}}$. □

Lemma 4. *For any CFD* $\mathbf{D}$ *and* $G\in\mathcal{G}_{\mathbf{D}}$: $\sqsubseteq_{\mathcal{R}_{\mathbf{D},G}} = \bigcup_{f\in G}\sqsubseteq_{\mathcal{R}_{\mathbf{D}f}}$.

Proof. We first show that, in a given CFD $\mathbf{D}$, for any group $G\in\mathcal{G}_{\mathbf{D}}$ with $\mathcal{C}(G)=\{(1,1)\}$ or $\mathcal{C}(G)=(|G|,|G|)$, the statement holds (equations (A.7) and (A.8), respectively).

According to Definition 5, $\mathcal{R}_{\mathbf{D},G} = +_{f\in G}\ \mathcal{R}_{\mathbf{D}f}$. Since $\forall f,f'\in G:(f\neq f')\implies \Sigma(\mathcal{R}_{\mathbf{D}f})\cap\Sigma(\mathcal{R}_{\mathbf{D}f'})=\varnothing$, according to Lemma 3(ii)

$$(\mathcal{C}(G)=\{(1,1)\})\implies(\sqsubseteq_{\mathcal{R}_{\mathbf{D},G}}=\bigcup_{f\in G}\sqsubseteq_{\mathcal{R}_{\mathbf{D}f}}) \tag{A.7}$$

Now, consider a group $G=\{f_1,\ldots,f_n\}\in\mathcal{G}_{\mathbf{D}}$ with $\mathcal{C}(G)=\{(|G|,|G|)\}=\{(n,n)\}$. According to Definition 5, $\mathcal{R}_{\mathbf{D},G}=\mathcal{R}_{\mathbf{D}f_1}\ ||\ldots||\ \mathcal{R}_{\mathbf{D}f_n}$. Since $\forall f,f'\in G:(f\neq f')\implies\Sigma(\mathcal{R}_{\mathbf{D}f})\cap\Sigma(\mathcal{R}_{\mathbf{D}f'})=\varnothing$, according to Lemma 3(vi),

$$(\mathcal{C}(G)=\{(|G|,|G|)\})\implies(\sqsubseteq_{\mathcal{R}_{\mathbf{D},G}}=\bigcup_{f\in G}\sqsubseteq_{\mathcal{R}_{\mathbf{D}f}}) \tag{A.8}$$

We are going to generalize (A.7) and (A.8) by proving the statement for any group with any arbitrary singleton multiplicity (i.e., a group G with $|\mathcal{C}(G)|=1$). Consider a group $G\in\mathcal{G}_{\mathbf{D}}$ with $\mathcal{C}(G)=(i,i)$, where $1\leq i\leq|G|$. We show that the statement of the lemma holds by inductive reasoning as follows.

(*base case*): It is inferable according to (A.7) and (A.8) that for any group G with $|G|=2$ and any singleton multiplicity constraint (i.e., either 1 or 2), $\sqsubseteq_{\mathcal{R}_{\mathbf{D},G}}=\bigcup_{f\in G}\sqsubseteq_{\mathcal{R}_{\mathbf{D}f}}$.

(*hypothesis*): Suppose that for some $n\in\mathbb{N}$ and any group G with $2\leq|G|\leq n$ and any valid singleton multiplicity constraint for G (i.e., $\mathcal{C}(G)=\{(i,i)\}$, where $1\leq i\leq n$), $\sqsubseteq_{\mathcal{R}_{\mathbf{D},G}}=\bigcup_{f\in G}\sqsubseteq_{\mathcal{R}_{\mathbf{D}f}}$.

(*inductive step*): We show that for any group G with $|G|=n+1$ and any valid singleton multiplicity constraint for G (i.e., $\mathcal{C}(G)=\{(i,i)\}$, where $1\leq i\leq n+1$), $\sqsubseteq_{\mathcal{R}_{\mathbf{D},G}}=\bigcup_{f\in G}\sqsubseteq_{\mathcal{R}_{\mathbf{D}f}}$.

According to (A.7) and (A.8), that statement holds for $i=n+1$ and $i=1$. Now, we prove it for an arbitrary $2\leq i\leq n$. Let $G=\{f_1,\ldots,f_{n+1}\}$. For any $1\leq j\leq n+1$, we denote the expression $\mathcal{R}_{\mathbf{D}f_j}$ by $\mathcal{R}_j$. Suppose that the multiplicity constraint of G is $\{(i,i)\}$ for an arbitrary $2\leq i\leq n$. Therefore, $\mathcal{R}_{\mathbf{D},G}=||^{\langle i\rangle}\{\mathcal{R}_1,\ldots,\mathcal{R}_{n+1}\}$. Obviously,

$$\begin{aligned}&\mathcal{R}_{\mathbf{D},G} = \mathcal{R}+\mathcal{R}',\text{ where}\\ &\mathcal{R} = ||^{\langle i\rangle}\{\mathcal{R}_1,\ldots,\mathcal{R}_n\},\\ &\mathcal{R}' = \mathcal{R}_{n+1}\ ||\ \mathcal{R}'',\text{ and }\mathcal{R}'' = ||^{\langle i-1\rangle}\{\mathcal{R}_1,\ldots,\mathcal{R}_n\}\end{aligned}$$

According to Lemma 3(i),

$$\sqsubseteq_{\mathcal{R}_{\mathbf{D},G}} = \sqsubseteq^{\Sigma}_{\mathcal{R}}\cap\sqsubseteq^{\Sigma}_{\mathcal{R}'},\text{ where }\Sigma=\bigcup_{1\leq j\leq n+1}\Sigma(\mathcal{R}_j) \tag{A.9}$$

According to Definition 6,

$$\begin{aligned}&\sqsubseteq^{\Sigma}_{\mathcal{R}} = \sqsubseteq_{\mathcal{R}}\ \cup\\ &\{(\sigma,\sigma'):\sigma\in\Sigma\setminus\Sigma(\mathcal{R}_{n+1})\wedge\sigma'\in\Sigma(\mathcal{R}_{n+1})\}\ \cup\\ &\Sigma(\mathcal{R}_{n+1})\times\Sigma(\mathcal{R}_{n+1})\setminus\{(\sigma,\sigma):\sigma\in\Sigma(\mathcal{R}_{n+1})\}\end{aligned} \tag{A.10}$$

Since $\Sigma(\mathcal{R}_{n+1})\cap\Sigma(\mathcal{R}'')=\varnothing$, according to Lemma 3(vi), $\sqsubseteq^{\Sigma}_{\mathcal{R}'} = \sqsubseteq_{\mathcal{R}_{n+1}}\cup\sqsubseteq_{\mathcal{R}''}$.

According to Definition 5, $\mathcal{R}$ and $\mathcal{R}''$ are equal to the regular expressions associated with two groups with the same cardinality n and singleton multiplicity constraints i and $i-1$, respectively. Therefore, due to the hypothesis,

$$\sqsubseteq_{\mathcal{R}} = \sqsubseteq_{\mathcal{R}''} = \bigcup_{1\leq j\leq n}\sqsubseteq_{\mathcal{R}_j} \tag{A.11}$$

The last two equations together imply that

$$\sqsubseteq^{\Sigma}_{\mathcal{R}'} = \bigcup_{1\leq j\leq n+1}\sqsubseteq_{\mathcal{R}_j} \tag{A.12}$$

Since $\mathcal{R}_{n+1}\subseteq\Sigma(\mathcal{R}_{n+1})^2\setminus\{(\sigma,\sigma):\sigma\in\Sigma(\mathcal{R}_{n+1})\}$, according to (A.11) and (A.10),

$$\bigcup_{1\le j\le n+1} \sqsubseteq_{\mathcal{R}_j} \subset \sqsubseteq_{\mathcal{R}}^{\Sigma} \tag{A.13}$$

According to (A.9), (A.12), and (A.13):

$$\sqsubseteq_{\mathcal{R}_{\mathbf{D},G}} = \bigcup_{1\le j\le n+1} \sqsubseteq_{\mathcal{R}_j} \tag{sing}$$

So far, we have proven the statement of the lemma for any group with a singleton multiplicity constraint. Now, we want to show that it holds for any multiplicity constraint. Consider a group G with $\mathcal{C}(G) = \{i_1, \ldots, i_n\}$ for some $1 \le i_j \le |G|$ $(1 \le j \le n)$. According to Definition 5, $\mathcal{R}_{\mathbf{D},G} = \mathcal{R}_1 + \ldots + \mathcal{R}_n$, where $\mathcal{R}_j$ denotes the regular expression associated with G with singleton multiplicity constraint i_j. Note that for any $1 \le j \le n$, $\Sigma(\mathcal{R}_j) = \bigcup_{f\in G} \Sigma(\mathcal{R}_{\mathbf{D}^f})$. Therefore, according to (sing) and Lemma 3(i),

$$\sqsubseteq_{\mathcal{R}_{\mathbf{D},G}} = \bigcup_{1\le j\le n} \sqsubseteq_{\mathcal{R}_j} \qquad \square \tag{gen}$$

Theorem 5. *For any given CFD* $\mathbf{D} = (F, r, \_^{\uparrow}, \mathcal{G}, \mathcal{C})$*, the following statement holds:* $\forall f, g \in F : g \in f_{\downarrow\downarrow} \Leftrightarrow f \sqsubseteq_{\mathcal{L}} g$*, where* $\mathcal{L} = \mathcal{L}(\mathcal{R}_{\mathbf{D}})$.

Proof. We prove the theorem by the following inductive reasoning on the depth of CFDs.
(*base case*): For any singleton CFD the statement obviously holds.
(*hypothesis*): Assume that for any CFD $\mathbf{D}$ with depth less than N $(\in \mathbb{N})$, $\mathcal{R}_{\mathbf{D}}$ preserves $\mathbf{D}$'s hierarchy.
(*inductive step*): We show that $\mathcal{R}_{\mathbf{D}}$ preserves the hierarchy of a CFD $\mathbf{D}$ with depth N.
Consider an arbitrary CFD $\mathbf{D}$ with depth N and root r whose set of groups is $\{G_1, \ldots, G_n\}$ for some $n \in \mathbb{N}$.
According to (8), $\mathcal{R}_{\mathbf{D}} = r\ (\mathcal{R}_{\mathbf{D},G_1} \mid\mid \ldots \mid\mid \mathcal{R}_{\mathbf{D},G_n})$.

$$\Longrightarrow \text{(Lemma 3)}\ \sqsubseteq_{\mathcal{R}_{\mathbf{D}}} = \sqsubseteq_{\mathcal{R}}, \text{where } \mathcal{R} = r\ (\mathcal{R}_{\mathbf{D},G_1} \mid\mid \ldots \mid\mid \mathcal{R}_{\mathbf{D},G_n})$$
$$\Longrightarrow \sqsubseteq_{\mathcal{R}_{\mathbf{D}}} = \{(r, f) : f \in r_{\downarrow\downarrow}\} \cup \sqsubseteq_{\mathcal{R}_{\mathbf{D},G_1}} \cup \sqsubseteq_{\mathcal{R}_{\mathbf{D},G_n}}$$
$$\Longrightarrow \text{(Lemma 4)}\ \sqsubseteq_{\mathcal{R}_{\mathbf{D}}} = \{(r, f) : f \in r_{\downarrow\downarrow}\} \cup \bigcup_{f\in r_{\downarrow}} \sqsubseteq_{\mathcal{R}_{\mathbf{D}^f}}$$

Since, for each $f \in r_{\downarrow}$, the depth of $\mathbf{D}^f$ is less than N, due to the hypothesis, $\mathcal{R}_{\mathbf{D}^f}$ preserves the hierarchy of $\mathbf{D}^f$. Adding the set $A = \{(r, f) : f \in r_{\downarrow\downarrow}\}$, since $A \cap \bigcup_{f\in r_{\downarrow}} \sqsubseteq_{\mathcal{R}_{\mathbf{D}^f}} = \varnothing$, the relation $\sqsubseteq_{\mathcal{R}_{\mathbf{D}}}$ preserves the hierarchy of $\mathbf{D}$. $\square$

A.2. Sect. 4

Theorem 6. *For a given CC* $\phi \in \Phi(F)$*, its language, i.e.,* $\mathcal{L}(\phi)$*, is in the class of context-sensitive languages.*

Proof. Consider an atomic CC $\phi = ({f_1}^{n_1} < {f_2}^{n_2})$. In some cases, the language may be in the class of regular languages, e.g., if f_1 (or f_2) is the root feature, then the language would be a regular one. In general, the language of ϕ, i.e., $\mathcal{L}(\phi) = \{w \in F^* : \#_w(f_1) \times n_1 < \#_w(f_2) \times n_2\}$, is a context-free language.[14]

Since the class of context-free languages is closed under union, the language of a compound CC built using disjunction and atomic CCs is always context-free.

As the class of context-free languages is not closed under complement, the language of a compound CC involving negation would not necessarily be in the class of context-free languages. However, since the class of context-sensitive languages (a superclass of context-free languages) is closed under complement, it is always in the class of context-sensitive languages.

According to the above, the class of languages of CCs in $\Phi(F)$ would be a subclass of context-sensitive languages. $\square$

A.3. Decidability results

Theorem 7. *Given a CFM, the operations Valid Configuration, Common Ancestors, and Least Common Ancestor are decidable.*

Proof. Let $\mathbf{M}$ be a CFM over a set of features F.

Recall that the Valid Configuration operation is reduced to a *membership problem* in the context of formal languages. Since the membership problem is decidable in the class of context-sensitive languages, the problem would be decidable in the class of CFMs generating context-sensitive languages.

[14] This is a well-known exercise, see for example [33].

Recall that we reduced the Common Ancestors problem to the following problem: Given a set of features F', a feature $f \in F$ is a common ancestor of the features in F' iff $\forall w \in \mathcal{L}(\mathbf{M}), \forall f' \in F' : f \sqsubseteq_w f'$. We can reduce it to the following (elegant) problem:

The problem deals with only the underlying CFD. Let $\mathbf{D}$ denote the CFD of $\mathbf{M}$. Since the problem has nothing to do with multiplicities, it is sufficient to work with the relaxed CFD of $\mathbf{D}$, denoted by $\mathbf{D}^\circ$ (converting all multiplicities of solitary features to 1). Thus, the above problem is reduced to "f is a common ancestor of the features in F' iff $\forall w \in \mathcal{L}(\mathbf{D}^\circ), \forall f' \in F' : f \sqsubseteq_w f'$". Since $\mathcal{L}(\mathbf{D}^\circ)$ and F' are finite, the common ancestors problem is decidable in all classes of CFMs. We can reason in the same way to show that the Least Common Ancestor problem in decidable in all classes of CFMs. □

Theorem 8. *Given a context-free CFM* $\mathbf{M}$*, the operations Partial Configuration, Core Features, Valid global Multiplicity, Void Feature Model, Finite Feature Model, and Dead Feature are decidable. However, none of them is decidable in the class of context-sensitive non-context-free CFMs.*

Proof. Let $\mathbf{M}$ be a context-free CFM over a set of features F.

Partial Configuration: Let $\mathcal{L}$ denote the set of all prefixes of the words of $\mathcal{L}(\mathbf{M})$. It is a well-known fact that $\mathcal{L}$ is a context-free language.[15] The set of partial configurations would be equal to the Parikh image of $\mathcal{L}$. Thus, the Partial Configuration problem is decidable.

Core Features: Consider a feature $f \in F$. We want to determine whether f is included in all configurations or not. Recall that the problem is reduced to the inclusion problem $\mathcal{L}(\mathbf{M}) \subseteq \mathcal{L}(F^* f F^*)$. Since the language $\mathcal{L}(F^* f F^*)$ is regular and $\mathcal{L}(\mathbf{M})$ is context-free, the above inclusion problem is decidable (see Preliminaries, Sect. 2.3).

Valid Global Multiplicity: Let n and f be a number and a feature, respectively. The problem is reduced to the decision problem $\mathcal{L}_n \cap \mathcal{L}(\mathbf{M}) \neq \varnothing$. The language $\mathcal{L}_n \cap \mathcal{L}(\mathbf{M})$ is always context-free, as $\mathcal{L}_n$ is a regular language and $\mathcal{L}(\mathbf{M})$ is context-free. Since the emptiness problem in the class of context-free languages is decidable, the decision problem is decidable.

Void Feature Model: The problem is reduced to the emptiness problem in the class of context-free languages. Since the emptiness problem is decidable in this class, the problem is decidable.

Finite Feature Model: Given a CFM $\mathbf{M}$, the problem is reduced to finiteness problem $\mathcal{L}(\mathbf{M}) = \varnothing$. It is well-known in the theory of formal languages that the set $\{G : G \text{ is a context-free grammar and } |\mathcal{L}(G)| < \infty\}$ is decidable: given a context-free grammar, we first convert it to an equivalent Chomsky normal form (note that this procedure is decidable [31]). Then, we build a directed graph which represents the connectivity between non-terminals of the grammar via its productions (since the set of productions in a grammar is finite, building this graph is trivially decidable). Now, the original problem is reduced to decide whether this directed graph is cyclic or not. There are some effective methods for this problem in graph theory (see for example [12]).

Dead Feature: Suppose that we want to determine whether a given feature f is dead or not. The problem is reduced to the emptiness problem $\mathcal{L}(F^* f F^*) \cap \mathcal{L}(\mathbf{M}) = \varnothing$. Note that $\mathcal{L}(F^* f F^*)$ is regular, which implies that $\mathcal{L}(F^* f F^*) \cap \mathcal{L}(\mathbf{M})$ is context-free. Since the emptiness problem of context-free languages is decidable, the problem is decidable.

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 and/or inclusion problems in formal language theory. Since these problems are not decidable in the class of non-context-free context-sensitive languages (see Preliminaries), the above problems would not be decidable in the class of context-sensitive CFMs. □

Theorem 9. *Given two CFMs* $\mathbf{M}_1$ *and* $\mathbf{M}_2$*, the following statements hold:*

(i) If both are context-free, then the Refactoring problem is decidable.

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

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

Proof. (i) According to [38], the Parikh image of any context-free language is a semilinear set, i.e., it is equal to the Parikh image of a regular language. Since the equality problem between regular languages (and so semilinear sets) are decidable, the Refactoring problem between two context-free CFMs is decidable.

(ii) The equality problem between regular languages is decidable [33].

(iii) Hopcroft in [25] showed that for two given context-free languages $\mathcal{L}_1$ and $\mathcal{L}_2$, if one of them, say $\mathcal{L}_1$, is a bounded regular language, then the equality problem between these two languages is decidable. □

Theorem 10. *Given two CFMs* $\mathbf{M}_1$ *and* $\mathbf{M}_2$*, the following statements hold:*

(i) If $\mathbf{M}_1$ *and* $\mathbf{M}_2$ *are context-free, then both the Specialization and Disjointness problems are decidable.*

[15] To prove this, we take the grammar of $\mathcal{L}(\mathbf{M})$ in Chomsky Normal Form and for every configuration $A \to BC$, add configurations $A_\varepsilon \to BC_\varepsilon$ and $A_\varepsilon \to B_\varepsilon$. Also, for every configuration $A \to f$ (for some terminals f), we consider the configuration $A_\varepsilon \to f$. Finally, we change the starting variable S to S_ε and add the configuration $S_\varepsilon \to \varepsilon$. The context-free grammar generated in this way represents the language $\mathcal{L}$. Thus, $\mathcal{L}$ is decidable.

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

Proof. (i) According to [38], the Parikh image of any context-free language is a semilinear set, i.e., it is equal to the Parikh image of a regular language (Parikh theorem).

Recall that the specialization problem "$\mathbf{M}_1$ is a specialization of $\mathbf{M}_2$" is reduced to the problem $\text{Par}(\mathcal{L}(\mathbf{M}_1)) \subseteq \text{Par}(\mathcal{L}(\mathbf{M}_2))$. Since $\mathbf{M}_1$ and $\mathbf{M}_2$ are context-free, according to the Parikh theorem, the problem is reduced to the inclusion problem on regular languages, which is decidable.

Recall that the disjointness problem between CFMs $\mathbf{M}_1$ and $\mathbf{M}_2$ is reduced to $\text{Par}(\mathcal{L}(\mathbf{M}_1)) \cap \text{Par}(\mathcal{L}(\mathbf{M}_2)) = \varnothing$. Since $\mathbf{M}_1$ and $\mathbf{M}_2$ are context-free, according to the Parikh theorem, the problem is receded to the intersection emptiness problem on regular languages, which is decidable.

(ii) The problem "$\mathbf{M}_2$ is a specialization of $\mathbf{M}_1$" is reducible to the problem $\mathcal{L}(\mathbf{M}_2) \subseteq \mathcal{L}(\mathbf{M}_1)$. In other words, it is equivalent to the problem of determining whether $\mathcal{L}(\mathbf{M}_2) \cap \overline{\mathcal{L}(\mathbf{M}_1)} = \varnothing$ or not, where $\overline{\mathcal{L}(\mathbf{M}_1)}$ denotes the complement of $\mathcal{L}(\mathbf{M}_1)$. Since the class of regular languages is closed under complement, $\overline{\mathcal{L}(\mathbf{M}_1)}$ is regular. Thus, $\mathcal{L}(\mathbf{M}_2) \cap \overline{\mathcal{L}(\mathbf{M}_1)}$ 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. □

A.4. Complexity results

Proposition 11. *The Valid Configuration problem in the regular, context-free, and context-sensitive CFMs are* P, NP-complete, *and* PSPACE-complete, *respectively.*

Proof. Given a CFM $\mathbf{M}$ over a set of features F and a multiset $m \in \mathcal{M}(F)$, we want to decide whether $m \in \text{Par}(\mathcal{L}(\mathbf{M}))$ or not. The Parikh image membership problem in the class of regular, context-free, and context-sensitive languages are P [30], NP-complete [30], and PSPACE-complete [22], respectively. □

Proposition 12. *The operations Partial Configuration and Valid Global Multiplicity are* NL *and* P *problems in the class of regular and context-free CFMs, respectively.*

Proof. These operations are both reduced to the intersection non-emptiness problem $\mathcal{L}(\mathbf{M}) \cap \mathcal{L} \neq \varnothing$ (where $\mathcal{L}$ is a regular language) for a given CFM $\mathbf{M}$ along with a multiset and an integer respectively. The intersection non-emptiness problem for a fixed number of regular languages (here two languages) is in the NL class [54]. The intersection non-emptiness problem for a fixed number of regular languages (here one language) and one context-free language is in the complexity class P [48]. □

Proposition 13. *The Dead Feature operation is an* NL *and* P *problem in the class of regular and context-free CFMs, respectively.*

Proof. Given a CFM $\mathbf{M}$ over a set of features F and a feature $f \in F$, this operation is reduced to the intersection emptiness problem $\mathcal{L}(\mathbf{M}) \cap \mathcal{L}(F^* f F^*) = \varnothing$. Note that $F^* f F^*$ is a regular expression. The complexity of the intersection non-emptiness problem has been discussed in the proof of Proposition 12. Since the intersection emptiness problem is the complement of the intersection non-emptiness problem, according to the above discussion: (1) The intersection emptiness problem for a fixed number of regular languages is in the coNL class; (2) the intersection emptiness problem for a fixed number of regular languages and one context-free language is in the complexity class coP. As $\text{NL} = \text{coNL}$ and $\text{P} = \text{coP}$, the statement of the proposition is proven. □

Proposition 14. *The Core Feature operation is a* P *problem in the class of regular CFMs. The lower bound time complexity of the operation in the class of context-free CFMs is exponential.*

Proof. Given a CFM $\mathbf{M}$ over a set of features F and a feature $f \in F$, the Core Feature operation is reduced to the inclusion problem $\mathcal{L}(\mathbf{M}) \subseteq \mathcal{L}(F^* f F^*)$. It is provable that the regular expression $F^* f F^*$ is one-unambiguous. Hovland in [26] proposed a polynomial-time algorithm for the inclusion problem $\mathcal{L} \subseteq \mathcal{L}'$, where $\mathcal{L}'$ is an one-unambiguous regular language. The lower bound time complexity of the inclusion problem $\mathcal{L} \subseteq \mathcal{L}'$, where $\mathcal{L}, \mathcal{L}'$ are context-free and regular, respectively, is exponential [27]. □

Proposition 15. *The Void Feature Model operation in the class of context-free CFMs is a* P *problem.*

Proof. A given CFM $\mathbf{M}$ is void iff $\mathcal{L}(\mathbf{M}) = \varnothing$, which is decidable in the class of context-free CFMs (Theorem 8). Testing a context-free grammar or a regular expression to determine if it generates a fixed finite language (including $\varnothing$) is in P [27]. □

Proposition 16. *The Finite Feature Model operation in the class of context-free CFMs is a* P *problem.*

Proof. A given CFM $\mathbf{M}$ is finite iff $\mathcal{L}(\mathbf{M})$ is finite. We have discussed the outline of an algorithmic procedure in the proof of Theorem 8 for this problem. The procedure includes the following steps: (1) converting a given context-free grammar to an equivalent Chomsky normal form; (2) building the connectivity graph for the non-terminals of the grammar; (3) detecting cycles in the directed graph. There are some polynomial time algorithm in terms of the number of non-terminals for (1) (e.g., see [31]). The size (number of the non-terminals) of the Chomsky normal form would be $O(n^2)$, where n is the size of the original grammar. Obviously, (2) can be computed in linear time. A very well-known efficient algorithm for detecting cycles in a given directed graph (i.e., the step (3)) is based on depth-first-search method [12], which is $\Theta(|V| + |E|)$ (V and E are the set of vertices and edges of the directed graph). Since $|V| \in O(n^2)$ and $|E| \in O(|V|^2)$, the complexity of the whole algorithm would polynomial. □

Proposition 17 *The Common Ancestor operation is a* P *problem in all classes of CFMs.*

Proof. Given a CFM $\mathbf{M}$ over a set of features F, a subset $F' \subset F$, and a feature f, deciding whether f is a common ancestor of F' is reduced to the problem $\forall w \in \mathcal{L}(\mathbf{M}), \forall f' \in F' : f \sqsubseteq_w f'$. According to the proof of Theorem 7, this can be further reduced to $\forall f' \in F' : f \sqsubseteq_{w^\circ} f'$, where w° is the single word of $\mathcal{L}(\mathbf{M}^\circ)$, i.e., the language of the relaxed CFM $\mathbf{M}$. Obviously, testing this problem would need $O(n^2)$ time, where $n = |F|$. □

Proposition 18. *The Refactoring operation is a* coNP-complete *and* Π_2^{P}-complete *problem in the class of regular and context-free CFMs, respectively.*

Proof. Given two CFMs, the Refactoring problem is reduced to the equality problem over the Parikh images of their languages, respectively. [30] showed that the equality problem on the Parikh images of two given regular languages is in the complexity class coNP-complete. The operation is in the complexity class Π_2^{P}-complete in the class of context-free languages [30]. □

Proposition 19. *The Specialization operation is a* P *and* coNP-complete *problem in the class of regular and context-free CFMs, respectively.*

Proof. Given two CFMs, the specialization problem is reduced to the inclusion problem over the Parikh images of their languages. [30] showed that the inclusion problem over the Parikh images of two given regular languages are in the complexity class P. It is also shown in [30] that the problem is in the complexity class coNP-complete in the class of context-free languages. □

Proposition 20. *The Disjointness operation is a* coNP *and* Π_2^{P} *problem in the class of regular and context-free CFMs, respectively.*

Proof. Given two CFMs, the disjointness problem is reduced to the disjointness problems over the Parikh images of their languages. [30] showed that the disjointness problem on the Parikh images of two given regular languages is in the complexity class coNP. It is also shown that the problem is in the complexity class Π_2^P in the class of context-free languages. □

Proposition 21. *Given two regular CFMs, the dynamic refactoring problem over them is in the* CSL-complete *class.*

Proof. Given two CFMs, the Dynamic refactoring is reduced to the equality problem over their languages. Deciding whether two given regular languages are equal or not is in the complexity class CSL-complete [27]. □

Proposition 22. *The lower bound time complexity of the Dynamic Specialization problem is exponential. However, it is* PSPACE-complete *when it is restricted to the regular CFMs.*

Proof. Given two CFMs $\mathbf{M}$ and $\mathbf{M}'$, the former is the Dynamic Specialization of the latter iff $\mathcal{L}(\mathbf{M}) \subseteq \mathcal{L}(\mathbf{M}')$. The inclusion problem in the class of regular languages is known as a PSPACE-complete problem [35]. The lower bound time complexity of the inclusion problem for a context-free language in a regular language is exponential [27]. □

Proposition 23. *The Dynamic Disjointness operation is a* P *problem if one of the given CFMs is context-free and the another is regular. However, it is an* NL *problem if restricted to the class of regular CFMs.*

Proof. Given two CFMs $\mathbf{M}$ and $\mathbf{M}'$, we say that they are Dynamically Disjoint if $\mathcal{L}(\mathbf{M}) \cap \mathcal{L}(\mathbf{M}') = \varnothing$. The complexity of the intersection emptiness problem has been already discussed in the proof of Proposition 13. □

References

[1] P.A. Abdulla, Y.-F. Chen, L. Clemente, L. Holík, C.-D. Hong, R. Mayr, T. Vojnar, Advanced Ramsey-based Büchi automata inclusion testing, in: CONCUR 2011 – Concurrency Theory, Springer, 2011, pp. 187–202.
[2] M. Acher, P. Collet, P. Lahire, R. France, Composing feature models, in: Software Language Engineering, Springer, 2010, pp. 62–81.
[3] M. Acher, B. Combemale, P. Collet, O. Barais, P. Lahire, R.B. France, Composing your compositions of variability models, in: Model-Driven Engineering Languages and Systems, Springer, 2013, pp. 352–369.
[4] M. Alanen, I. Porres, et al., A Relation Between Context-Free Grammars and Meta Object Facility Metamodels, Turku Centre for Computer Science, 2004.
[5] S. Apel, D. Batory, C. Kästner, G. Saake, Feature-Oriented Software Product Lines, Springer, 2016.
[6] K. Bak, Z. Diskin, M. Antkiewicz, K. Czarnecki, A. Wasowski, Clafer: unifying class and feature modeling, Softw. Syst. Model. 15 (3) (2016) 811–845.
[7] D. Benavides, S. Segura, A. Ruiz-Cortés, Automated analysis of feature models 20 years later: a literature review, Inf. Syst. 35 (6) (2010) 615–636.
[8] A. Bergmayr, M. Wimmer, Generating metamodels from grammars by chaining translational and by-example techniques, in: MDEBE@ MoDELS, 2013, pp. 22–31.
[9] F. Bonchi, D. Pous, Checking NFA equivalence with bisimulations up to congruence, ACM SIGPLAN Not. 48 (1) (2013) 457–468.
[10] A. Brüggemann-Klein, D. Wood, One-unambiguous regular languages, Inf. Comput. 142 (2) (1998) 182–206.
[11] S.A. Cook, The complexity of theorem-proving procedures, in: Proceedings of the Third Annual ACM Symposium on Theory of Computing, ACM, 1971, pp. 151–158.
[12] T.H. Cormen, Introduction to Algorithms, MIT Press, 2009.
[13] S. Crespi-Reghizzi, D. Mandrioli, Commutative grammars, Calcolo 13 (2) (1976) 173–189.
[14] K. Czarnecki, M. Antkiewicz, C.H.P. Kim, S. Lau, K. Pietroszek, FMP and FMP2RSM: eclipse plug-ins for modeling features using model templates, in: Companion to the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, ACM, 2005, pp. 200–201.
[15] K. Czarnecki, U.W. Eisenecker, Generative Programming: Methods, Tools, and Applications, ACM Press/Addison–Wesley Publishing Co., New York, NY, USA, 2000.
[16] K. Czarnecki, S. Helsen, U. Eisenecker, Formalizing cardinality-based feature models and their specialization, Softw. Process Improv. Pract. 10 (1) (2005) 7–29.
[17] K. Czarnecki, C.H.P. Kim, Cardinality-based feature modeling and constraints: a progress report, in: International Workshop on Software Factories, 2005, pp. 16–20.
[18] M. De Wulf, L. Doyen, N. Maquet, J.-F. Raskin, Alaska, in: Automated Technology for Verification and Analysis, Springer, 2008, pp. 240–245.
[19] D. Dhungana, A. Falkner, A. Haselbock, Configuration of cardinality-based feature models using generative constraint satisfaction, in: 2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA, IEEE, 2011, pp. 100–103.
[20] G. Fleishanderl, G.E. Friedrich, A. Haselbock, H. Schreiner, M. Stumptner, Configuring large systems using generative constraint satisfaction, IEEE Intell. Syst. Appl. 13 (4) (1998) 59–68.
[21] G. Gange, J.A. Navas, P. Schachte, H. Søndergaard, P.J. Stuckey, A tool for intersecting context-free grammars and its applications, in: NASA Formal Methods, Springer, 2015, pp. 422–428.
[22] M.R. Garey, D.S. Johnson, Computers and Intractability, vol. 29, W.H. Freeman, New York, 2002.
[23] A. Gómez, I. Ramos, Cardinality-based feature modeling and model-driven engineering: fitting them together, VaMoS 37 (2010) 61–68.
[24] S. Hilber, Finite automata tool, http://cl-informatik.uibk.ac.at/software/fat/, 2009.
[25] J.E. Hopcroft, On the equivalence and containment problems for context-free languages, Math. Syst. Theory 3 (2) (1969) 119–124.
[26] D. Hovland, The inclusion problem for regular expressions, J. Comput. Syst. Sci. 78 (6) (2012) 1795–1813.
[27] H.B. Hunt, D.J. Rosenkrantz, T.G. Szymanski, On the equivalence, containment, and covering problems for the regular and context-free languages, J. Comput. Syst. Sci. 12 (2) (1976) 222–268.
[28] K.C. Kang, S.G. Cohen, J.A. Hess, W.E. Novak, A.S. Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study, Tech. rep., DTIC Document, 1990.
[29] A.S. Karataş, H. Oğuztüzün, A. Doğru, From extended feature models to constraint logic programming, Sci. Comput. Program. 78 (12) (2013) 2295–2312.
[30] E. Kopczynski, A.W. To, Parikh images of grammars: complexity and applications, in: 2010 25th Annual IEEE Symposium on Logic in Computer Science, LICS, IEEE, 2010, pp. 80–89.
[31] M. Lange, H. Leiß, To CNF or not to CNF? an efficient yet presentable version of the CYK algorithm, Informatica Didactica 8 (2009) 2008–2010.
[32] O. Lengál, J. Šimáček, T. Vojnar, Vata: a library for efficient manipulation of non-deterministic tree automata, in: Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2012, pp. 79–94.
[33] P. Linz, An Introduction to Formal Languages and Automata, Jones & Bartlett Publishers, 2011.
[34] M. Mendonca, A. Wasowski, K. Czarnecki, D. Cowan, Efficient compilation techniques for large scale feature models, in: Proceedings of the 7th International Conference on Generative Programming and Component Engineering, ACM, 2008, pp. 13–22.
[35] A.R. Meyer, L.J. Stockmeyer, The equivalence problem for regular expressions with squaring requires exponential space, in: IEEE Conference Record of 13th Annual Symposium on Switching and Automata Theory, 1972, IEEE, 1972, pp. 125–129.
[36] R. Michel, A. Classen, A. Hubaux, Q. Boucher, A formal semantics for feature cardinalities in feature diagrams, in: Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems, ACM, 2011, pp. 82–89.
[37] C.H. Papadimitriou, On the complexity of integer programming, J. ACM 28 (4) (1981) 765–768.
[38] R.J. Parikh, On context-free languages, J. ACM 13 (4) (1966) 570–581.
[39] C. Quinton, A. Pleuss, D.L. Berre, L. Duchien, G. Botterweck, Consistency checking for the evolution of cardinality-based feature models, in: Proceedings of the 18th International Software Product Line Conference – Volume 1, ACM, 2014, pp. 122–131.
[40] C. Quinton, D. Romero, L. Duchien, Cardinality-based feature models with constraints: a pragmatic approach, in: Proceedings of the 17th International Software Product Line Conference, ACM, 2013, pp. 162–166.
[41] S.H. Rodger, T.W. Finley, JFLAP: An Interactive Formal Languages and Automata Package, Jones & Bartlett Learning, 2006.
[42] A. Safilian, T. Maibaum, Hierarchical multiset theories of cardinality-based feature diagrams, in: 2016 10th International Symposium on Theoretical Aspects of Software Engineering, TASE, IEEE, 2016, pp. 136–143.
[43] A. Safilian, T. Maibaum, Z. Diskin, The semantics of cardinality-based feature models via formal languages, in: FM 2015: Formal Methods, Springer, 2015, pp. 453–469.
[44] T. Schnabel, M. Weckesser, R. Kluge, M. Lochau, A. Schürr, Cardygan: tool support for cardinality-based feature models, in: Proceedings of the Tenth International Workshop on Variability Modelling of Software-intensive Systems, ACM, 2016, pp. 33–40.
[45] S. Segura, D. Benavides, A. Ruiz-Cortés, P. Trinidad, Automated merging of feature models using graph transformations, in: Generative and Transformational Techniques in Software Engineering II, Springer, 2008, pp. 489–505.
[46] S. She, R. Lotufo, T. Berger, A. Wasowski, K. Czarnecki, Reverse engineering feature models, in: ICSE 2011, IEEE, 2011, pp. 461–470.
[47] G. Sousa, W. Rudametkin, L. Duchien, Extending feature models with relative cardinalities, in: Proceedings of the 20th International Systems and Software Product Line Conference, ACM, 2016, pp. 79–88.

[48] J. Swernofsky, M. Wehar, On the complexity of intersecting regular, context-free, and tree languages, in: International Colloquium on Automata, Languages, and Programming, Springer, 2015, pp. 414–426.
[49] M.H. ter Beek, A. Legay, A.L. Lafuente, A. Vandin, Statistical analysis of probabilistic models of software product lines with quantitative constraints, in: Proceedings of the 19th International Conference on Software Product Line, ACM, 2015, pp. 11–15.
[50] T. Thum, D. Batory, C. Kastner, Reasoning about edits to feature models, in: IEEE 31st International Conference on Software Engineering, 2009, ICSE 2009, IEEE, 2009, pp. 254–264.
[51] Y.-K. Tsay, Y.-F. Chen, M.-H. Tsai, W.-C. Chan, C.-J. Luo, Goal extended: towards a research tool for omega automata and temporal logic, in: Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2008, pp. 346–350.
[52] G. van Noord, Fsa6.2xx: finite state automata utilities, http://odur.let.rug.nl/vannoord/Fsa/fsa.html, 2002. (Accessed 3 October 2003).
[53] M. Weckesser, M. Lochau, T. Schnabel, B. Richerzhagen, A. Schürr, Mind the gap! automated anomaly detection for potentially unbounded cardinality-based feature models, in: Fundamental Approaches to Software Engineering, Springer, 2016, pp. 158–175.
[54] M. Wehar, Hardness results for intersection non-emptiness, in: International Colloquium on Automata, Languages, and Programming, Springer, 2014, pp. 354–362.
[55] M. Wimmer, G. Kramler, Bridging grammarware and modelware, in: Satellite Events at the MoDELS 2005 Conference, Springer, 2006, pp. 159–168.