A Formal Semantics for Feature Cardinalities in Feature Diagrams

Raphael Michel
CETIC Research Center
raphael.michel@cetic.be

Andreas Classen
University of Namur
acs@info.fundp.ac.be

Arnaud Hubaux
University of Namur
ahu@info.fundp.ac.be

Quentin Boucher
University of Namur
qbo@info.fundp.ac.be

ABSTRACT

Feature cardinalities in feature diagrams determine the number of times a feature and its subtree can be duplicated during configuration by an operation named "cloning".

Other authors already investigated the problem and published different proposals of semantics for this construct. However, this previous work is not easily amenable to the formal study of the various properties of feature diagrams and their derived configurations. Also, cross-tree constraint languages still need to be properly extended to account for feature cardinalities.

This paper presents an extension of an earlier formal semantics of feature diagrams by adding support for feature cardinalities.

1. INTRODUCTION

In software product lines (SPL) engineering, feature diagrams (FD) are a popular family of modelling languages used to describe variability and commonalities across products [14]. Most of these languages are graphical and depict an SPL as a tree or a DAG, where nodes are features and edges represent hierarchical decomposition of features. Variability is expressed by defining the combinations of features one can and cannot choose when configuring a product using various mechanisms such as optional features, group cardinalities and constraints. The following figure illustrates these concepts on a simple example taken from one of our industrial case studies.

Figure 1 illustrates a product line of documents. Documents can be normal documents or booklets. If the sheets must be bound, the binding feature is selected and one (and only one) of the three proposed sides (top, left or right) must be selected. This is specified by the [1..1] cardinality on the group of features under "binding". However, as expressed by the additional constraint under the FD, if the document is a booklet, no binding is allowed.

VaMoS '11, January 27-29, 2011 Namur, Belgium

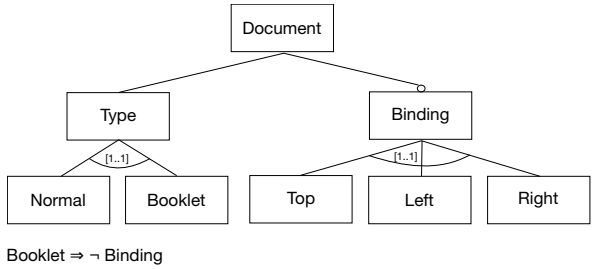

Figure 1: A simple FD

These notations have been surveyed and formalised elsewhere [17]. However, they generally lack a construct that allows to duplicate a subtree of the FD to configure a product of the SPL. The following figure extends our previous example to illustrate this concept.

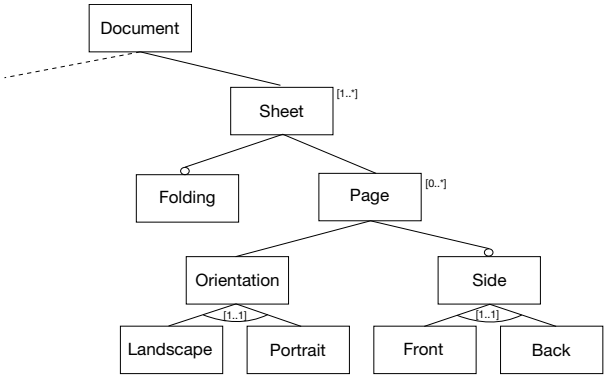

Figure 2: Feature cloning

Figure 2 represents the case where sheets and pages can appear multiple times in a document. In this case, a document is made of one or more sheets which can be folded or not. Each sheet can contain pages for which the orientation and the side of the sheet can be chosen. This FD illustrates a case where a feature that is declared once in the feature diagram will be duplicated an arbitrary number of times in the final product. This operation of duplicating features is known as *feature cloning* [9].

Feature cloning comes along with another concept illustrated in Figure 2: next to each cloneable feature, an in-

terval specifying bounds on the number of times the feature can be cloned is specified between brackets. These bounds are called *feature cardinalities*. Riebisch et al. [16] were the first to propose to extend FDs with cardinalities similar to those found in UML class diagrams.

Even if their visual representations are very similar, feature cardinalities and group cardinalities are different types of constraints. Feature cardinalities specify the number of times a feature and its subtree can be duplicated; group cardinalities require to make a choice between several alternatives.

Feature cloning and feature cardinalities allow to go one step further in modelling by specifying bounds on the number of times a feature can be included and customized before the actual system is generated, and therefore opens the door to richer models and richer configurations.

Most configurators used in feature-oriented development produce a configuration containing the desired features by allowing to choose between several variants, or deciding whether an optional feature will or will not be included in the system being generated. They allow to make choices that are valid with respect to the feature model of the SPL but do not include the ability to duplicate features and their subtrees. Cases like the one presented as example in Figure 2 cannot be handled by such configurators because of their lack of support for feature cardinalities.

From a theoretical perspective, a formal semantics is necessary to precisely evaluate language expressiveness as well as the complexity of its associated decision procedures. From a practical perspective, formal semantics is a prerequisite to unambiguous modelling as well as correct and efficient implementation of model-based automations.

Several authors already covered the topic and presented formal definitions of feature cardinalities and cloning. Zhang et al. present their own version of the semantics of cloneable features in [21] using two different patterns and a way of using BDDs [2] to verify models. In [9] and [10], Czarnecki et al. define and formalize the syntax and semantics of clone-enabled FDs. Although their definition is a major source of inspiration for our work, it suffers from accidental complexity due to (1) a lack of distinction between syntactic domain, semantic domain and semantic function, and (2) the usage of a rather operational style. Sun et al. propose a formal specification in [20] which also suffers from this lack of distinction between the elements the language. This specification is described using the Z language and then mapped to Alloy to conduct analyses. Bak et al. propose *Clafer* [4], a modelling language with support for feature cardinalities and present its semantics as a mapping to Alloy. Prior to Clafer, Anastakis et al. [1] already translated UML models to Alloy. Asikainen et al. defined Kumbang [3] as a UML Profile, and PCML (Product Configuration Modelling Language) [13]. The semantics of both these languages are defined as mappings to WCRL (Weight Constraint Rules Language) [19], a more general-purpose language which is handled by smodels [18] to conduct analyses. While all these mappings to different languages and tools like Alloy or WCRL and smodels, indeed define semantics, these semantics are implementation-dependent and not necessarily easy to re-implement on other platforms. They are also not well suited to conduct formal analyses and compare to properties of other semantics.

Although the concepts found in the literature already provide a solid foundation to work upon, we transpose them in our own terms using a universal mathematical representation, which is completely independent from any existing language or tool. As mentioned before, this universality ensures that our semantics can easily be reused to implement any language and makes it easier to analyse and compare properties of languages.

Our contribution is an adaptation of previously defined semantics [17] with support for feature cardinalities. As prescribed by the guidelines of Harel and Rumpe in [12], we propose formal definitions of the syntactic domain, the semantic domain and the semantic function.

The remainder of this paper is structured as follows. Section 2 presents our earlier definition of the semantics of FD, the foundations on which we build up this work, Section 3 and 4 describe how we formalize cardinality-based FDs and their semantics. Sections 5 and 6 identify perspectives and challenges for future work and, finally Section 7 concludes this paper.

2. BACKGROUND

In earlier work [17] a formal syntax and semantics for feature diagrams was presented. As this paper builds on this existing work and extends it, we first recall its essential elements.

Following the guidelines of Harel and Rumpe [12], the semantics is defined by distinguishing the three elements of our language: the syntactic domain, the semantic domain, and the semantic function.

2.1 Syntactic domain

The syntactic domain of our language determines everything that can be written using an FD modelling language. From an abstract point of view, any FD can be seen as a tree of features, containing a single root, and where each feature is decomposed in one or more features, except for the leaves. Features can be labeled as optional and cardinalities can be used to define the decomposition type of a feature.

DEFINITION 1 (SYNTACTIC DOMAIN $\mathcal{L}_{FD}$).
$d \in \mathcal{L}_{FD}$ is a 6-tuple $(F, r, \omega, DE, \lambda, \Phi)$ such that:

- F is the (non empty) set of features (nodes).
- $r \in F$ is the root.
- $\omega : F \to \{0, 1\}$ labels optional features with a 0
- $DE \subseteq F \times F$ is the decomposition relation between features which forms a tree. For convenience, we will use $children(f)$ to denote $\{g \mid (f, g) \in DE\}$, the set of all direct sub-features of f, and write $n \to n'$ sometimes instead of $(n, n') \in DE$.
- $\lambda : F \to \mathbb{N} \times \mathbb{N}$ indicates the decomposition type of a feature, represented as a cardinality $\langle i..j \rangle$ where i indicates the minimum number of children required in a product and j the maximum.
- Φ is a formula that captures crosscutting constraints ($\ll requires \gg$ and $\ll includes \gg$) as well as textual constraints. Without loss of generality, we consider Φ to be a conjunction of Boolean formulae on features, i.e. $\Phi \in \mathbb{B}(N)$

Furthermore, each $d \in \mathcal{L}_{FD}$ must satisfy the following well-formedness rules:

- r is the root: $\forall f \in F(\nexists f' \in F \bullet f' \rightarrow f) \Leftrightarrow f = r$,
- DE is acyclic: $\nexists f_1, .., f_k \in F \bullet f_1 \rightarrow .. \rightarrow f_k \rightarrow f_1$,
- Leaves are $\langle 0..0 \rangle$-decomposed.
- Except for the root, each node has a single parent: $\forall f \in F \setminus r : \exists! f' \in F \bullet f' \rightarrow f$

Illustration.

The following table gives a formal abstract representation of the example presented in Figure 1.

- F = {Document, Type, Normal, Booklet, Binding, Top, Left, Right}
- r = Document

Feature	ω	λ	Parent (DE)
Document	1	$\langle 1..2 \rangle$	
Type	1	$\langle 1..1 \rangle$	Document
Normal	0	$\langle 0..0 \rangle$	Type
Booklet	0	$\langle 0..0 \rangle$	Type
Binding	0	$\langle 1..1 \rangle$	Document
Top	0	$\langle 0..0 \rangle$	Binding
Left	0	$\langle 0..0 \rangle$	Binding
Right	0	$\langle 0..0 \rangle$	Binding

- $\Phi = \neg Booklet \vee \neg Binding$

These features can be combined in various ways to form products. The set of all possible combinations of features from a given FD is called the *semantic domain*. The following subsection describes this concept.

2.2 Semantic Domain

The semantic domain contains every product that can be derived from any possible FD expressed in terms of the abstract syntax. It is thus the set of all possible sets of features.

Definition 2 (Semantic domain $\mathcal{S}_{FD}$).

$$\mathcal{S}_{FD} = \mathcal{P}(\mathcal{P}(F))$$

The semantic domain of a simple feature diagram is often way too big to be fully represented. On a simple FD without feature cardinalities, the size of the semantic domain is 2^n (where n is the number of features). Enumerating all possible combinations of the eight features of the first example would result in 64 possible configurations. Due to cardinalities, tree decompositions, and other additional constraints, not every configuration that is part of the semantic domain refers to a valid product. That is, some configurations are discarded because they do not fulfill all the constraints defined in the FD. The mapping between the syntactic domain, and the corresponding valid configurations in the semantic domain is made through the *semantic function*.

2.3 Semantic Function

According to Harel and Rumpe [12], the semantic function maps elements from the syntactic domain to their meaning in the semantic domain. The semantic function of FDs is formally defined in [7, 17] as :

Definition 3 (Semantic function).

$$\mathcal{M} : \mathcal{L}_{FD} \rightarrow \mathcal{S}_{FD}$$

where $\mathcal{M}(d)$ is the set of all valid feature sets $c \in \mathcal{P}(F)$. Each $c \in \mathcal{M}(d)$ is such that

- *it contains the root : $r \in c$*
- *it satisfies the group cardinalities:*
  $\forall f \in c : \lambda(f) = \langle m..n \rangle$
  $\Rightarrow m - |opt_F| \leq |mand_p| \wedge |all_p| \leq n$
  where:
  - $opt_F = \{g | g \in F \wedge \omega(g) = 0 \wedge f \rightarrow g\}$
  - $mand_c = \{g | g \in F \wedge \omega(g) = 1 \wedge f \rightarrow g\}$
  - $all_c = \{g | g \in c \wedge f \rightarrow g\}$
- *it justifies each feature:*
  $\forall f \in F, g \in c : g \in children(f) \Rightarrow f \in c$
- *it satisfies Φ*

As an example, the FD depicted in Figure 1 allows only 5 valid products (among the 64 combinations) :

- $\{Document, Type, Booklet\}$
- $\{Document, Type, Normal\}$
- $\{Document, Type, Normal, Binding, Top\}$
- $\{Document, Type, Normal, Binding, Left\}$
- $\{Document, Type, Normal, Binding, Right\}$

2.4 Limitations

In this section, we presented the translation of the first example in terms of our formal semantics. The second example (illustrated in Figure 2) demonstrates the limitations of the current semantics.

- The syntactic domain lacks support for feature cardinalities, i.e., they cannot be expressed in the FD.
- As the semantic domain is defined as a set, it is impossible to include the same feature multiple times.
- When sheets and pages are added to a document, one needs to know on which sheet a given page appears. The semantics must maintain this parent-children relationship between clones.

These limitations impact both the static representation of a SPL using an FD, and the dynamic process of configuring a product. The static representation includes everything that is defined when designing the FD, that is, the model of the product line in terms of features, commonalities and variability. The configuration process is dynamic in the sense that it consists of a sequence of steps that refine the FD until a single configuration is left.

3. FORMALIZING CLONES

As mentioned in Section 2.4, the current syntax and semantics of FD has limitations and does not support cloning. This section describes the lacking elements of the existing semantics, how clones are going to be represented as well as how the syntax and semantics presented in Section 2 is modified to overcome the limitations.

3.1 Syntactic Domain

Feature cardinalities allow to constrain the number of clones allowed for a given feature.

The current formal syntactic domain already provides a very limited support for feature cardinalities through ω, a function which defines if a feature is optional or mandatory. Although cardinalities are commonly represented as an interval (or as a union of intervals) $\langle m..n \rangle$ where $m, n \in \mathbb{N} : m \leq n$, the optional or mandatory nature of a feature can be considered as a cardinality as well. Optionality is translated to $\langle 0..1 \rangle$ and mandatory features are labeled with $\langle 1..1 \rangle$. Extending the domain of ω from $\{0, 1\}$ to $\mathbb{N} \times (\mathbb{N} \cup \{*\})$ thus provides a way to define cardinalities for each feature without loss of the initial functionality. The union $\mathbb{N} \cup \{*\}$ as proposed by Czarnecki et al. in [11], allows to define unbounded cardinalities. The $<$ operator is then extended over this domain as $\forall n \in \mathbb{N} : n < *$. For example this is needed in the previously mentioned case studu, see Figure 2 where the number of sheets is unknown in advance and thus unbounded. This extension comes with a well-formedness rule : the root of a FD cannot be cloned and is always mandatory. More formally, $\omega(r) = \langle 1..1 \rangle$

3.2 Distinguishing Features and Clones

Before diving into the details of the semantics of feature cardinalities, we need to clarify important terms of the vocabulary we will use in the rest of this paper.

As mentioned in Section 2.2, in the previous semantics, a product was defined by the set of features that were selected. It was possible to define products as sets of features because (i) features could only be included once in a product and (ii) the FD contained the whole information about how features were decomposed. Therefore no ambiguity could exist about how the features included the product were related to each other. This allowed to abuse the term *feature* and use it to refer to the nodes of the FD, as well as to the constituting elements of a product.

With feature cardinalities however, a feature can be included more than once in a product. Figure 2 illustrates a case where a document designer needs to be able to specify the orientation of each page he wants to print. The final product can contain the features *Page*, *Portrait* and *Landscape* several times if the document contains several pages. This example already gives an intuitive feeling of the problem: if a document with two pages having different orientations is represented as { *Document, Sheet, Page, Orientation, Page, Orientation, Portrait, Landscape* }, the FD tells that a *Page* has an orientation, but not which *Page* has which *Orientation* (*Portrait* or *Landscape*). In order to be able to reason about this problem, a clear distinction between the nodes of the FD and elements that constitute the product is required: we will use the term *feature* to refer to a node of the FD and *clone* to refer to an element of a product.

An analogy can be made between clone-enabled FD and object models: intuitively, an FD defines a "type" (feature) hierarchy, and a clone is an "instance" of a feature in the object-oriented sense of the term.

3.3 Representing Clones

Since the semantic domain contains the products which are made of clones, we first need to define a suitable structure to adequately represent clones and products.

We propose a representation that maintains the tree structure of the FD and hereby preserves the parent-children relationship. This representation has a recursive definition: a clone is represented by a tuple containing its feature and a multiset of children which are themselves clones. An intuitive way to express this is by describing it as a grammatical rule : $Clone := (Feature, \{Clone^*\})$ where $Feature \in F$. Note that this definition is purely illustrative and hides the fact that the children of a clone are contained in a multiset which is by nature, unordered. The following definition formalizes the structure of clones.

Definition 4 (Clone). *If we define a clone as a tuple (feature, children) where children is a multiset of clones, then the set of all possible clones is C such that: $C \subseteq F \times powerbag(C)$*

This representation structurally forces a clone to have a single parent. This may be seen as a lack of flexibility but we believe that it also eliminates a lot of complexity, as up until now, we did not encounter any real case which requires a feature to be shared among several parents.

3.4 Semantic Domain

The semantic domain must include every possible product that can be derived from any possible FD. Without cloning, the semantic domain is thus the set of all possible sets of features, i.e. $\mathcal{P}(\mathcal{P}(F))$. As stated in [8], simply defining a configuration as a multiset of features, rather than a set, would not solve the problem because the final products must also contain information about how these features are connected to each other, which is not possible using multisets since the parent of a feature might appear more than once. As we defined a clone being a feature and its multiset of children, any clone of the root feature is an actual product. We can thus redefine the semantic domain S_{FD} as the set of all possible clones.

Definition 5 (Semantic domain). *If C is the set of all possible clones as defined in Definition 4, we define the semantic domain as $\mathcal{S}_{FD} = C$*

4. SEMANTICS OF CLONING

The introduction of cardinalities on features as well as the concept of clones and their recursive definition have a profound impact on the semantic function and require an adaptation.

Before we can do this, we need to examine what cloning actually means. As it turns out, cloning is not just a small addition to FDs, but a complex construct that affects group cardinalities and the way we think about optionality. This clearly illustrates the need for and benefit of a formal, tool independent semantics.

4.1 Group Cardinalities

Group cardinalities are an already well-known concept in the field of FD, but the fact that multiple clones of a given feature can appear due to the feature cardinalities, the meaning of group cardinalities has to be redefined precisely. Without feature cardinalities, group cardinalities define the number of features that can be chosen among a set of options under an "or" node. When features can appear

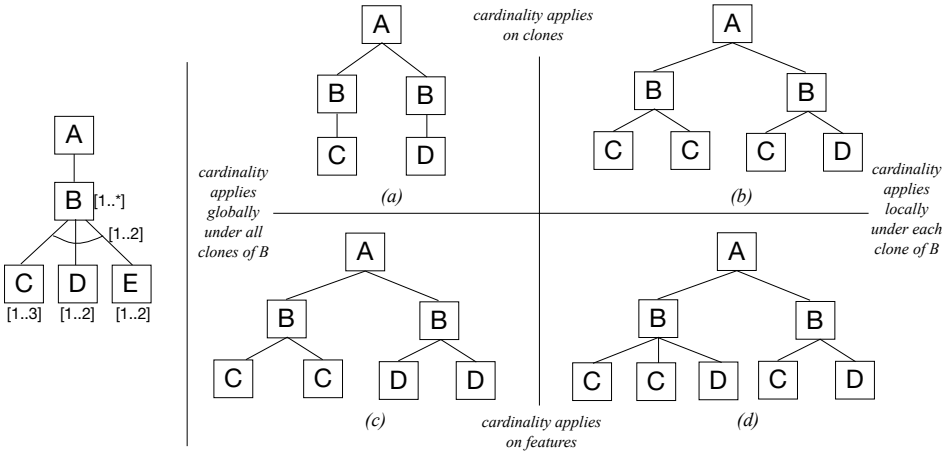

Figure 3: Possible products depending on interpretation of group cardinalities

multiple times because they are cloned, there are several ways to interpret this semantically.

The group cardinality $\langle i..j \rangle$ requires that the number of selected children be between i and j. The question is how to count these when there are clones involved. Basically, there are two questions: (a) Should all clones be counted irrespective of their parent (i.e. the cardinality applies globally), or should clones be counted under each cloned parent (i.e. the cardinality is local to each parent)? We will refer to this as the *level*. (b) Should the clones, that is, the instances of features, be counted rather than the features that were instantiated? We will refer to this as the *scope*. This leads to four different interpretations as illustrated in Figure 3.

(a) *scope: clones, level: global.* The group cardinality affects the total number of nodes under all clones of B. The scope contains all clones, at a global level. In this case, as illustrated in (a) on the schema, we count one clone of C and one clone of D, which together reach the maximal cardinality of two specified in the FD.

(b) *scope: clones, level: local* The group cardinality affects the total number of nodes under each clone of B. The scope targeted here are the clones, at a local level. On the schema labeled (b), we count two clones of C under the B at the left side, reaching the maximum cardinality, and thereby preventing to define a clone of D, and under the second B at the right side, we count two clones ($1\ C\ +\ 1\ D$), and we see that there is no possibility to add more clones of C or D even though their own cardinalities allow more than one clone of each feature.

(c) *scope: features, level: global* The group cardinality affects the total number of features under all clones of B. The scope is limited to features, at a global level. In this case, on schema (c), C and D count together for two features and thus prevent any addition of a clone of E under any of the two clones of B.

(d) *scope: features, level: local* The group cardinality affects the total number of features under each clone of B. The scope is limited to features at a local level. In this case, the cardinality is verified on the number of features present under each clone of B: as illustrated on the schema labeled (d), on each side, C and D each count for one, giving two, even though there are two clones of C on the left, they are counted as a single feature (type).

In the previous semantics, the question was not raised because without feature cardinalities, a feature in the FD leads to a single node in the product tree. We believe that, when reproducing the same FD with feature cardinalities by setting $\langle 0..1 \rangle$ or $\langle 1..1 \rangle$ cardinalities on the features, the semantics should remain unchanged, which leads us to choose the fourth option. With each different interpretation proposed, we illustrated a case where the interpretation would lead to a counter-intuitive functioning of the cardinalities.

4.2 Feature Cardinalities

Feature cardinalities define how many times a given feature can be repeated. Similarly to group cardinalities, there is a question about what is to be counted when checking the cardinality: should all clones be counted, irrespective of their parent (the cardinality applies globally), or should clones be counted under their respective parent (locally)? In

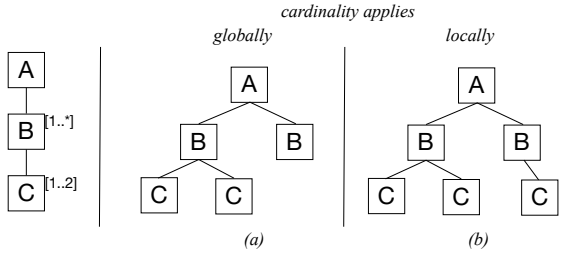

Figure 4: Possible products depending on interpretation of feature cardinalities

Figure 4, we see the difference between the global and the local level. On the left side (a), if the cardinality on C is enforced on the whole product (i.e. at the global level), the second clone of B cannot have a clone of C as child, because the first B has already the maximum number of clones of C in its children. On the other side, as illustrated on the right side (b), when feature cardinalites are local, each clone of B can have one or two clones of C as children, as the cardinality is applied under each clone of B.

We decided to choose the latter option, that is, to limit the

scope of the feature cardinalities at the "local" level. Besides the fact that this option has also been choosen by other authors [9], it also corresponds to the intuitive notion of feature cloning and allows to duplicate a subtree independently of the rest of the product. The example illustrated in Figure 2 gives a good overview of this intuition: if the cardinality on *Page* had been $\langle 1..4 \rangle$ to express the fact that only four pages can be printed on each sheet (e.g. two on each side in booklets), applying the cardinality constraint locally would prevent printing more than four pages on each sheet while applying the cardinality constraint globally would prevent to print any document having more than four pages.

4.3 Optionality

As it was already the case in our previous semantics, optionality has "priority" over group cardinalities. This means that if a feature is optional, the cardinalities of an "and" group will not prevent it to be excluded from the product.

However, as mentioned previously, feature cardinalities also have an impact on the way we see optionality. Previously, if a feature was part of an "or" group, it could always be omitted from a product: this is the essence of an "or" group. A mandatory feature could then be considered optional because the only rules enforcing the presence of mandatory features were the group cardinalities. In other words, no rule explicitly verified that a feature labelled as mandatory was actually part of the product. Feature cardinalities are more restrictive because they actually check the number of clones and this behaviour should not interfere with "or" group cardinalities. This problem is best illustrated in Figure 5. We see that the feature cardinalities require at least one clone of B, one clone of C and one clone of D to be included. If these cardinalities are applied without regard to the "or" group, the cardinality allowing to select only one of the three features becomes meaningless as none of the three options can be omitted. In order to have an "or" group working as intuitively expected, all features in the group should be considered as optional. We thus decided to artificially make all features of a group optional by allowing to exclude a feature from the group by verifying feature cardinalities only for features for which at least one clone is present.

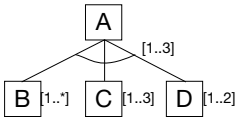

Figure 5: Feature cardinalities and "or" groups

4.4 Semantic Function

The goal of the semantic function is to make a clear distinction between elements of the semantic domain that we consider as valid products and those that are considered invalid. The rules presented previously in this section allow to make this distinction and are formalized in the semantic function redefined hereunder.

The semantic function ensures that a product (1) has the correct root and (2) satisfies the decomposition rules and the cardinalities. This point is verified using four rules. The first rule ensures that the features of the elements of the product are decomposed exactly as they are in the FD. For example, if a feature a is parent of a feature b in the FD, then each clone of b will have a clone of a as parent. The second rule expresses formally what has been discussed in Section 4.2 about how feature cardinalities are checked. The third rule describes how the group cardinalities are verified as described in Section 4.1. As the clones, and thus the products, are defined recursively, so is the part of the semantic function that refers to them. A clone satisfies the constraints, if it respects the decomposition and cardinalities and all of its children respect them as well, this recursion is allowed by the fourth rule.

In order to improve readability, we use the following notational conventions:

- $\{\!|...|\!\}$ denotes a multiset or bag and $\{...\}$ denotes a regular set.

DEFINITION 6 (SEMANTIC FUNCTION $\mathcal{M} : L_{FD} \rightarrow S_{FD}$). *$\mathcal{M}(d)$ is the set of all products such that $\forall p \in \mathcal{M}(d)$:*

- *p is topped by the root of the FD : $p = (r, D)$*
- *p satisfies the constraints of the FD : $p \models d$*

$(f, D) \models d \iff$

1. *The decomposition hierarchy is respected:*
   $\forall (g, E) \in D : f \rightarrow g$
2. *The feature cardinalities are respected:*
   $\forall g : f \rightarrow g : \omega(g) = \langle m..n \rangle$
   $\Rightarrow clones_g = 0 \ \vee \ m \leq clones_g \leq n$
   where: $clones_g = |\{\!|(g, E) \in D|\!\}|$
3. *The group cardinalities are respected: let $\lambda(f) = \langle m..n \rangle$, then*
   $m - |opt_F| \leq |mand_p| \ \wedge \ |all_p| \leq n$
   where:
   - $opt_F = \{g | g \in F \ \wedge \ \omega(g) = \langle 0..y \rangle \wedge f \rightarrow g\}$
   - $all_p = \{g | \exists (g, E) \in D\}$
   - $mand_p = all_p \setminus opt_F$
4. *For each child of f, the rules are respected:*
   $\forall (g, E) \in D : (g, E) \models d$

5. CROSS-TREE CONSTRAINTS

In addition to the changes already presented, the introduction of feature cardinalities has an impact on the cross-tree constraints.

In the previous semantics, the elements of a product were features, so these constraints were expressed by Φ, a Boolean formula over the set of features. Now that the elements of a product are clones, the meaning of a constraint expressed in terms of features is not clear anymore. For example, in the constraint $Booklet \Rightarrow Folding$, *Booklet* refers to a single feature while *Folding* is a clonable feature. It is thus unclear to what *Folding* inside the constraint refers to. Intuitively, if a document is a booklet, then all its sheets must be folded. That would mean that *Folding* is to be interpreted as "all the clones of *Folding*".

Another constraint that occurs in the document management case is that "if the document is of type Normal, then at least one page must be on the front side of a sheet". This

cannot be written $Normal \Rightarrow Front$, since that would mean that *all* pages must be on the front side of a sheet. This illustrates that existing constraint language is not expressive enough to specify the constraints one might wish to express in the presence of clones.

Essentially, the constraints which were expressed in terms of features have to be expressed in terms of clones. Basically, in the first example, the feature $Folding$ was an implicit universal quantification over the set of clones of the $Folding$ feature. For the second example, in contrast, we would need an existential quantification over the set of clones of $Front$. A constraint language should thus offer the ability to use quantifiers over the sets of clones of certain features. Moreover, to specify that clones of a certain feature should be included or excluded from a product, the language should have a construct for expressing constraints on the number of clones of a certain feature in the product.

In addition to quantification, the constraint language needs to be able to handle the relationships between clones. Take, for instance, the following constraint defined on the FD used in Figure 3: "all clones of B, which have exactly two clones of C also have at least three clones of D". In this example, the scope is in between the two extremes presented in Figure 3 and Figure 4: it neither applies locally under each clone nor globally on all clones of B, it actually applies locally under some clones of B that satisfy the condition "have exactly 2 clones of C". This simple example shows that this problem is not trivial and that it should be the object of further research into a constraint language for FD with support for feature cardinalities. This language should support formula quantification as well as lists comprehensions and their associated operators.

Other authors already investigated the use of languages like OCL as a constraint definition language for FDs [12]. However this language is much more general purpose and is not specifically tailored for FD specific constraints.

6. TOWARDS AUTOMATED TOOL SUPPORT

A formal semantics is a basis for many applications. Many possible analyses and applications have been described in [5] and studied extensively by other authors. The current reference implementation of our semantics is a configurator based on a SAT solver [15]. The SAT approach is well appropriate when using only Boolean constraints but becomes limitating when feature cardinalities and cloning are involved. The main issue we will have to face is that a single unbounded cardinality in a FD implies a theoretically infinite set of products. Therefore some properties like satisfiability might not always be verified completely.

We intend to update the TVL language [6] and its associated tools with feature cardinalities, so that it conforms to the semantics formalized in this paper and serve as a reference implementation. However, implementation details such as technologies and algorithms still have to be determined.

7. CONCLUSION

Previous formalizations of cardinality based feature models suffered from complexity by lack of distinction between the syntactic domain, the semantic domain and the semantic function. Others simply defined semantics as mappings to tools or other languages. We propose a formalization independent from any tool or language. The previous version of formal semantics of FDs defined in [17] lacked support for feature cardinalities and cloning, an important construction in feature-oriented development. In this paper we addressed this issue by proposing modifications to enrich the previous work in a consistent manner. We extended the semantics so that it now handles feature cardinalities and clones. We also identified new challenges that come with the fundamental changes that we have made, among which the need for an extended constraint language to complement our current definition as well as theoretical problems due to the possibly infinite size of the domains to explore when performing formal analyses.

Acknowledgements

This work is sponsored by the Interuniversity Attraction Poles Programme of the Belgian State, Belgian Science Policy, under the MoVES project, the Walloon Region under the NAPLES project and a FIRST DOC.A Fund, and the FNRS.

8. REFERENCES

[1] K. Anastasakis, B. Bordbar, G. Georg, and I. Ray. On challenges of model transformation from uml to alloy. *Software and System Modeling*, 9(1):69–86, 2010.

[2] H. R. Andersen. An introduction to binary decision diagrams. Technical report, Course Notes on the WWW, 1997.

[3] T. Asikainen, T. Männistö, and T. Soininen. Kumbang: A domain ontology for modelling variability in software product families. *Adv. Eng. Inform.*, 21:23–40, January 2007.

[4] K. Bak, K. Czarnecki, and A. Wasowski. Feature and class models in Clafer: Mixed, specialized, and coupled. In *Accepted for publication at the 3rd International Conference on Software Language Engineering (SLE 2010)*.

[5] D. Benavides, S. Segura, and A. Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Inf. Syst.*, 35:615–636, September 2010.

[6] Q. Boucher, A. Classen, P. Faber, and P. Heymans. Introducing TVL, a text-based feature modelling language. In *Proceedings of the Fourth International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'10), Linz, Austria, January 27-29*, pages 159–162. University of Duisburg-Essen, January 2010.

[7] A. Classen, Q. Boucher, and P. Heymans. A text-based approach to feature modelling: Syntax and semantics of TVL (in press). *Science of Computer Programming, Special Issue on Software Evolution*, 2010. doi:10.1016/j.scico.2010.10.005.

[8] A. Classen, A. Hubaux, and P. Heymans. A formal semantics for multi-level staged configuration. In *Third International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'09)*, January 2009.

[9] K. Czarnecki, S. Helsen, and U. Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 10(1):7–29, 2005.

[10] K. Czarnecki, S. Helsen, and U. Eisenecker. Staged configuration through specialization and multilevel configuration of feature models. *Software Process: Improvement and Practice*, 10(2):143–169, 2005.

[11] K. Czarnecki and C. H. P. Kim. Cardinality-based feature modeling and constraints: a progress report. In *International Workshop on Software Factories at OOPSLA'05*, San Diego, California, USA, 2005. ACM.

[12] D. Harel and B. Rumpe. Modeling languages: Syntax, semantics and all that stuff, part i: The basic stuff. Technical report, Mathematics & Computer Science, Weizmann Institute Of Science, Mathematics & Computer Science, Weizmann Rehovot, Israel, August 2000.

[13] M. Heiskala, A. Anderson, V. Huhtinen, J. Tiihonen, and A. Martio. A tool for comparing configurable products. In *International Conference On Engineering Design (ICED) 2003*, 2003.

[14] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature-Oriented Domain Analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, SEI, Carnegie Mellon University, November 1990.

[15] D. Le Berre. SAT4j: a reasoning engine in Java based on the SATisfiability problem (SAT). http://www.sat4j.org.

[16] M. Riebisch, K. Böllert, D. Streitferdt, and I. Philippow. Extending feature diagrams with uml multiplicities. In *6th World Conference on Integrated Design & Process Technology (IDPT2002)*, June 2002.

[17] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, and Y. Bontemps. Feature diagrams: A survey and a formal semantics. In *Requirements Engineering Conference, 2006. RE 2006. 14th IEEE International*, pages 136–145, 2006.

[18] P. Simons, I. Niemelá, and T. Soininen. Extending and implementing the stable model semantics. *Artif. Intell.*, 138:181–234, June 2002.

[19] T. Soininen, I. Niemelä, J. Tiihonen, and R. Sulonen. Representing configuration knowledge with weight constraint rules. In *Answer Set Programming'01*, 2001.

[20] J. Sun, H. Zhang, and H. Wang. Formal semantics and verification for feature modeling. In *Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems*, pages 303–312, Washington, DC, USA, 2005. IEEE Computer Society.

[21] W. Zhang, H. Yan, H. Zhao, and Z. Jin. A BDD-based approach to verifying clone-enabled feature models' constraints and customization. pages 186–199. 2008.