# Modelling equivalence classes of feature models with concept lattices to assist their extraction from product descriptions

Jessie Carbonnel, Marianne Huchard, Clémentine Nebut

Version of Record: https://www.sciencedirect.com/science/article/pii/S0164121219300378
Manuscript_ba1da175d3a0e767bcbc67622e89b686

# Modelling Equivalence Classes of Feature Models with Concept Lattices to assist their Extraction from Product Descriptions

Jessie Carbonnel[a], Marianne Huchard[a], Clémentine Nebut[a]

[a]*LIRMM, Université de Montpellier and CNRS, Montpellier, France*

**Abstract**

Software product line engineering gathers a set of methods to help create, manage and maintain a collection of similar software systems. Variability modelling is a focal point of this paradigm, where feature models (FMs) are the prevalent notation. Migration from single system development to software product lines is a spreading topic in software engineering. To ease the migration, research has been done to automatically extract FMs from software descriptions, but most of these approaches are defined in a functional manner based on an *ad-hoc* variability analysis. In this paper, we propose a theoretical view on FM extraction from software descriptions based on Formal Concept Analysis (FCA). It is a structural framework for variability representation which allows to lay down theoretical foundation to variability extraction. We propose an original mapping between relationships expressed in FMs and the ones emphasised in FCA conceptual structures. We show that conceptual structures represent equivalence classes of FMs that steer the user choices during their synthesis, and propose a reverse engineering method based on them. We discuss its applicability and show that the combinatorial explosion of concept lattices can be avoided by the use of two sub-orders embodying the necessary information concerning variability.

*Keywords:* Software Product Lines, Reverse Engineering, Formal Concept Analysis, Variability Modelling, Feature Models

## 1. Introduction

A software family designates a collection of software systems that are similar enough to justify their management as a single entity (by considering their commonalities) rather than individually (by considering their specificities). Software product line engineering (SPLE) [1] gathers a set of methods which aims to produce such a group of similar software systems while reducing their development cost and their time to market, and increasing their quality and their supply. It is a software development paradigm based on mass customisation and systematic reuse: rather than individually developing each similar software, SPLE seeks to derive a set of *software variants* from a common set of reusable artefacts (code, requirement, architecture...) which are structured in a generic architecture. The term software product line (SPL) encompasses the generic architecture, the reusable artefacts and the set of derivable software variants. Variability modelling is a focal point of SPLE, whose purpose is to document what is common and what varies between the software variants; it aims to facilitate the management of a potentially numerous set of variants by factorising their common parts and identifying their specific ones. Several approaches exist to model variability, such as decision modelling [2] or feature modelling [3]. In this paper, we focus on the most prevalent approach, feature modelling, where a feature is a distinguishable characteristic present in one or several variants, generally representing a high level functionality. In this approach, a software variant is represented by the set of features it owns. Such a combination of features is called a valid configuration, and represents a high level functional description of a software variant of the SPL. Feature models (FMs) [3, 4] are the most commonly used notation to model SPL variability in terms of features. FMs are a family of description languages which permit to describe a

*Email addresses:* jcarbonnel@lirmm.fr (Jessie Carbonnel), huchard@lirmm.fr (Marianne Huchard), nebut@lirmm.fr (Clémentine Nebut)

set of features and relationships between them; they represent in a compact and understandable way a set of valid configurations, thus delimiting the scope of an SPL. Aside from variability representation, FMs are also used in other SPLE related tasks, as information retrieval, SPL evolution and maintenance [5], or product derivation [6].

However, in some legacy systems, different variants of the same system may have been individually developed without using systematic reuse [7]. For instance, *clone-and-own* is a common practice in industries to build variants of existing software systems: it consists in cloning existing softwares and arbitrarily removing or adding functionalities to these clones in order to satisfy a new set of requirements [8]. In these cases, practitioners may be confronted to a twofold issue. On the one hand, it is difficult to maintain and manage such a group of software variants, especially when their number and complexity increase over time. On the other hand, migrating to SPLE is an arduous task which implies, inter alia, to build a variability model from an existing collection of product variants [9]. Even with a small number of variants, a manual construction of FMs from existing products is challenging. To ease the migration, research is made about automated synthesis of FMs from product descriptions [10, 11, 12, 13, 14, 15, 16]. They usually seek to extract correct and relevant relationships between the features present in the set of initial variants, as a basis to build an FM. However, most of the proposed approaches are defined in a functional manner based on an *ad-hoc* variability analysis through heterogeneous structures and techniques.

Formal Concept Analysis (FCA) [17] is a mathematical data analysis framework for hierarchical clustering which has been used to support automated synthesis of FMs. As input, FCA takes a set of objects described by attributes, and organises these objects depending on the attributes they share: the result is a canonical structure called a concept lattice, which naturally highlights commonalities and variability of the input set of objects. FCA has successfully been applied in SPLE to structure a set of existing software variants (as objects) described by a common set of features (as attributes). Examples of its usage include FM reverse engineering [10, 13], organising SPL scenarios [18], locating features in source code [19, 20], restructuring FM variability [5], and recovering SPL architecture [21].

In this paper, we argue that most existing approaches for FM reverse engineering are encompassed in FCA, which offers a structural and reusable framework for variability extraction and representation, while being based on a solid theoretical background. To discuss this statement, we study correlations between FCA conceptual structures and FM semantics, and how they lay down theoretical foundations for FM reverse engineering. We first recall definitions of FMs and FCA on which our work is founded, and discuss the prevalent challenges in FM reverse engineering (Section 2). It leads us to two research questions: Which parts of this activity can be automated? Which parts need user intervention? We discuss FCA-based methods for automated variability extraction in the form of logical relationships from software variant descriptions, and we show that these methods are sound and complete (Section 3). Then, we study the commonalities between FCA conceptual structures and FMs, and propose an original mapping between the two formalisms. We show that a conceptual structure (namely a concept lattice, an AOC-poset or an AC-poset) entirely includes a given FM logical semantics, and therefore represents the equivalence class of FMs representing the same set of valid configurations, that we call an *equivalence class feature diagram* (ECFD for short). It allows us to highlight the parts of the extraction that may be fully-automated. To ease the representation of the interesting variability information present in conceptual structures, we propose a domain specific modelling language for ECFDs (Section 4). Based on these foundations, we propose a semi-automated FM reverse engineering method. We illustrate this process on an application on FM re-engineering borrowed from the work of Haslinger et al. [22] (Section 5). We evaluate the applicability of the proposed method in Section 6. Section 7 discusses related work and how FCA encompasses existing methods: we give hints about the benefits of considering FCA as an unifying framework, which may be used for knowledge representation and information management to further assist the migration to SPLs, without being confined to FM synthesis. Finally, Section 8 presents conclusion and future work.

This paper proposes the following contributions:

- sound and complete FCA-based methods for automated variability extraction in the form of logical relationships;
- original mapping between FCA structures and FMs leading to a theoretical characterisation of equivalence classes of FMs;
- FM reverse engineering method based on the aforementioned mapping;
- presenting FCA as a framework unifying existing FM reverse engineering methods;

- applicability study of FCA-based variability modelling for collections of existing software variants.

## 2. Background

In this section, we first present feature models, the *de facto* standard for variability modelling in terms of features. Then, we give the theoretical bases of formal concept analysis, and present three of its associated conceptual structures called concept lattices, AOC-posets and AC-posets. Finally, we recall the main challenges of FM reverse engineering.

### *2.1. Feature models*

Feature models (FMs) [3] are a family of visual description languages which allow to define the scope of a product line in terms of features, where a feature refers to a distinguishable characteristic that can be either possessed or not by a product variant. By defining a set of features and relationships between them, an FM permits to describe a finite set of *valid configurations*, i.e., a subset of features which satisfies all the constraints expressed by the relationships. Each valid configuration describes a product variant by the features it owns; thus, an FM is a compact representation of all the possible variant descriptions of a product line. There are extended versions of FMs [23, 24], but here we focus on the basic FMs [3, 4], also called boolean FMs.

A boolean FM represents a finite set of features in a hierarchy (called the *feature tree*), where the most general features are located at the top of the tree, and the most specialised ones are situated at the bottom. This hierarchy expresses child-parent relationships: it indicates that if a feature is selected to be in a configuration then its direct parent-feature has to be in the configuration as well. The edges of the feature tree are decorated to represent other relationships that show how the selection of a feature may affect the selection of its child features. Figure 1 (left) depicts the edge decorations of these relationships. A black disc requires the selection of the child feature when the parent feature is selected (*mandatory relationship*), whereas a white circle indicates that the child feature can be optionally selected (*optional relationship*). Several child features can be grouped, a group being depicted by an arc which indicates the number of features that can be selected: a black-filled arc states that at least one child feature of the group has to be selected (*or-group*), and a non-filled arc shows that exactly one child feature of the group has to be selected (*xor-group*). Finally, relationships that cannot be represented in the hierarchy (usually binary *requires* and *exclude* constraints) are called *cross-tree constraints* (CTCs), and can be added to complete the feature tree. It is noteworthy that FMs are not canonical representations, i.e., several different FMs may represent the same set of configurations. An example of a boolean FM about e-commerce applications is presented in Figure 1 (right).

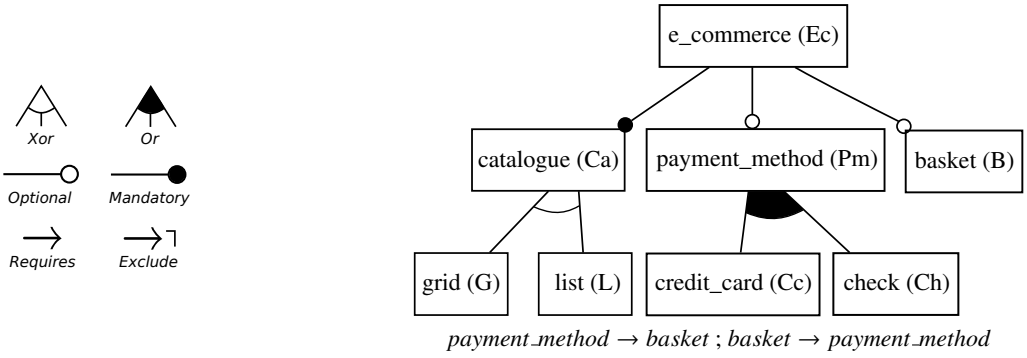



Figure 1: (Left) Boolean FM relationships; (Right) Boolean FM about e-commerce applications

We can read in this FM that: an e-commerce application necessarily possesses a catalogue; this catalogue can be displayed in a grid or in a list, but not both; an application can eventually possess payment methods (credit card, check, or both); it can also optionally have a basket; the CTCs state that if the basket feature is selected, the application must possess at least one payment method, and conversely.

The FM of our example depicts 8 valid configurations, that are displayed in Table 1. The configuration {*e_commerce*, *catalogue*, *grid*, *basket*, *payment_method*, *credit_card*} satisfies all the FM constraints, and therefore describes a possible variant. However, the configuration {*e_commerce*, *catalogue*, *grid*, *basket*} does not correspond to any variant, because the constraint $basket \rightarrow payment\_method$ is not satisfied.

*2.2. Formal concept analysis*

Formal concept analysis (FCA) [17] is a mathematical data analysis framework structuring a set of objects $O$ described by a set of binary attributes $A$. FCA is based on a binary relation $R \subseteq O \times A$ stating "which objects are described by which attributes". This relation can be represented by a cross table $O \times A$, called a *formal context*, where a cross in the cell $(o, a)$ states that $(o, a) \in R$.

**Definition 2.1 (Formal context).** *A formal context is a triple $K = (O, A, R)$ where $O$ and $A$ are two finite sets and $R \subseteq O \times A$ is a binary relation. Elements from $O$ are called the objects and elements from $A$ the attributes. A pair $(o, a)$ from $R$ states that "the object $o$ possesses the attribute $a$".*

Table 1 represents a formal context with 8 objects (rows) and 8 attributes (columns).

Table 1: Formal context depicting the 8 valid configurations of the boolean FM of Figure 1

| | Ec | Ca | G | L | Pm | Cc | Ch | B |
|---|---|---|---|---|---|---|---|---|
| v1 | x | x | x | | | | | |
| v2 | x | x | x | | x | x | | x |
| v3 | x | x | x | | x | | x | x |
| v4 | x | x | x | | x | x | x | x |
| v5 | x | x | | x | | | | |
| v6 | x | x | | x | x | x | | x |
| v7 | x | x | | x | x | | x | x |
| v8 | x | x | | x | x | x | x | x |

Ec = e_commerce, Ca = catalogue, G = grid, L = list,
Pm = payment_method, Cc = credit_card, Ch = check,
B = basket

The application of FCA on a formal context extracts a finite set of *formal concepts*.

**Definition 2.2 (Formal concept).** *Given a formal context $K = (O, A, R)$, a formal concept $C$ is a pair $(E, I)$ such that $E \subseteq O$ and $I \subseteq A$. It depicts a maximal set of objects that share a maximal set of common attributes. $E = \{o \in O \mid \forall a \in I,\ (o, a) \in R\}$ is the concept's **extent**, denoted by Ext(C), and $I = \{a \in A \mid \forall o \in E,\ (o, a) \in R\}$ is the concept's **intent**, denoted by Int(C).*

For instance, let us arbitrarily select the set of objects $\{v1,\ v2\}$ in Table 1. Now, we select all the attributes shared by this set of objects, and we obtain the following set of attributes: $\{e\_commerce,\ catalogue,\ grid\}$. Finally, let us retrieve all the objects possessing this set of attributes: we obtain $E = \{v1,\ v2,\ v3,\ v4\}$. We have extracted the formal concept composed of the pair $E = \{v1,\ v2,\ v3,\ v4\}$ and $I = \{e\_commerce,\ catalogue,\ grid\}$.

The set of all concepts that can be extracted from a formal context $K$ can be partially ordered by the set-inclusion order on the concepts' extents, also called the specialisation order $\leq_{CL}$.

**Definition 2.3 (Specialisation order).** *Given a formal context $K = (O, A, R)$ and two concepts $C_1 = (E_1, I_1)$ and $C_2 = (E_2, I_2)$ of $K$, $C_1 \leq_{CL} C_2$ if and only if $E_1 \subseteq E_2$ and $I_2 \subseteq I_1$. Then, $C_1$ is called a sub-concept of $C_2$, and $C_2$ a super-concept of $C_1$.*

Therefore, a concept inherits all the attributes of its super-concepts, and all the objects of its sub-concepts. When provided with the specialisation order $\leq_{CL}$ the set of all concepts forms a structure called a *concept lattice*.

**Definition 2.4 (Concept lattice).** *Given $C_K$ the set of all concepts extracted from a formal context $K$, the concept lattice associated with $K$, denoted by $(C_K, \leq_{CL})$, is the set of all concepts $C_K$ provided with the specialisation order $\leq_{CL}$.*

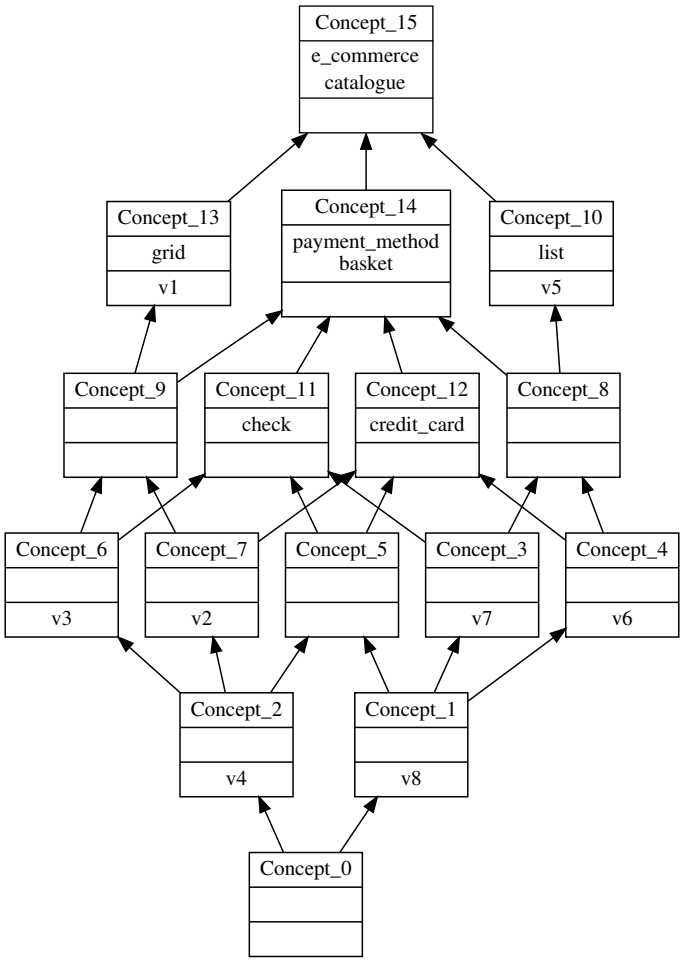



Figure 2: Concept lattice associated with the context of Table 1

Figure 2 represents the Hasse diagram of the concept lattice associated with the formal context of Table 1, from which 16 concepts have been extracted and then partially ordered. The construction tool used here is `RCAExplore`[1]. A concept is represented by a three-part box displaying the name of the concept (top part), its intent (middle part), and its extent (bottom part). An arrow between two concepts shows the specialisation order. In this representation, intents and extents of concepts are simplified: attributes (resp. objects) appear only once in the concept lattice, in the concept where they are *introduced* i.e., the greatest (resp. lowest) concept having that attribute (resp. object). In this simplified representation, the intent and the extent of a concept can then be reconstituted by inheritance. For example, the intent of *Concept_13* is Int(*Concept*_13) = {*e_commerce*, *catalogue*, *grid*}, and its extent is Ext(*Concept*_13) = {*v*1, *v*2, *v*3, *v*4}.

We call *object-concepts* and *attribute-concepts* the concepts which introduce respectively at least an object or an attribute; we call *plain-concepts* the ones which introduce neither attributes nor objects. In Figure 2, *Concept_4* is an object-concept introducing *v6*, *Concept_11* an attribute-concept introducing *check*, *Concept_13* introducing *grid* and *v1* is both, and *Concept_9* is a plain-concept. In what follows, the set of all object-concepts of a context $K$ is denoted by $\mathcal{OC}_K$, and the set of all attribute-concepts is denoted by $\mathcal{AC}_K$.

In some applications, it is not necessary to take into account plain-concepts. These cases generally occur when FCA is used to organise by specialisation the elements of the input dataset, but not to emphasise clusters among them.

[1] http://dataqual.engees.unistra.fr/logiciels/rcaExplore

In such applications, one can choose to construct the concept hierarchy without the plain-concepts: the obtained structure is a sub-order called an *Attribute-Object-Concept partially ordered set (AOC-poset)* [25].

**Definition 2.5 (AOC-poset).** *Given $C_K$ the set of all concepts extracted from a formal context K, the AOC-poset is the sub-order of $(C_K, \leq_{CL})$ restricted to attribute-concepts and object-concepts: $(\mathcal{AOC}_K \cup \mathcal{OC}_K, \leq_{CL})$.*

Figure 3 presents the AOC-poset associated with the formal context of Table 1. It represents the partial order of

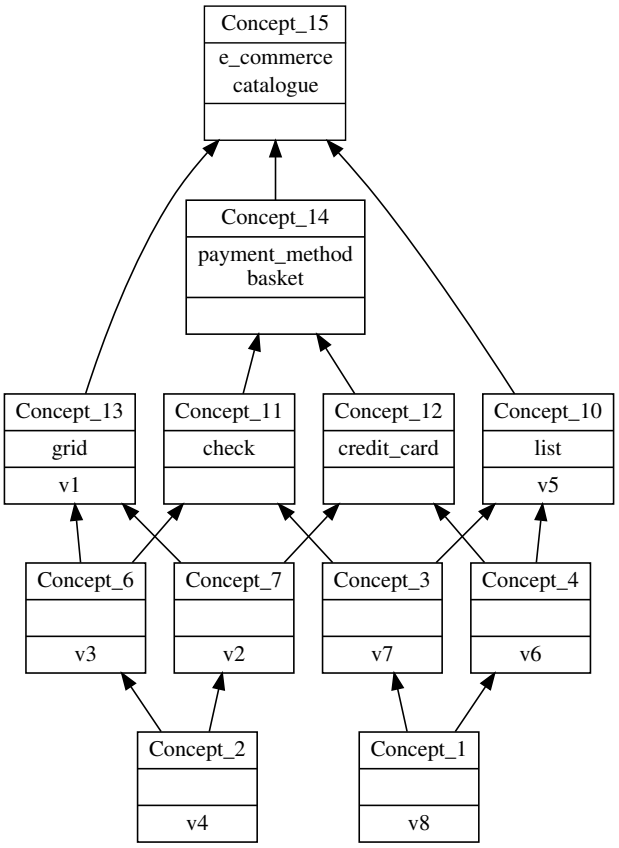



Figure 3: AOC-poset associated with the formal context of Table 1

the concepts of Figure 2, minus the plain-concepts *Concept_0*, *Concept_5*, *Concept_8* and *Concept_9*. An AOC-poset associated with a context *K* then possesses less concepts than the concept lattice associated with the same context, but it preserves the hierarchy between the elements and the formal context can be fully reconstituted from it.

Another interesting sub-order is the *Attribute-Concept partially ordered set* (AC-poset), which retains only the attribute-concepts.

**Definition 2.6 (AC-poset).** *Given $C_K$ the set of all concepts extracted from a formal context K, the AC-poset is the sub-order of $(C_K, \leq_{CL})$ restricted to attribute-concepts: $(\mathcal{AC}_K, \leq_{CL})$.*

Figure 4 shows the AC-poset of Table 1. It is the minimal conceptual structure conserving the hierarchy between attributes, and it may be seen as a binary implication graph. As for the AOC-poset, the formal context can be fully reconstituted from it. More details about dimensions of FCA structures are presented in Section 6.

### *2.3. Known challenges in FM reverse engineering*

The literature identifies two main challenges in FM reverse engineering from variant descriptions.

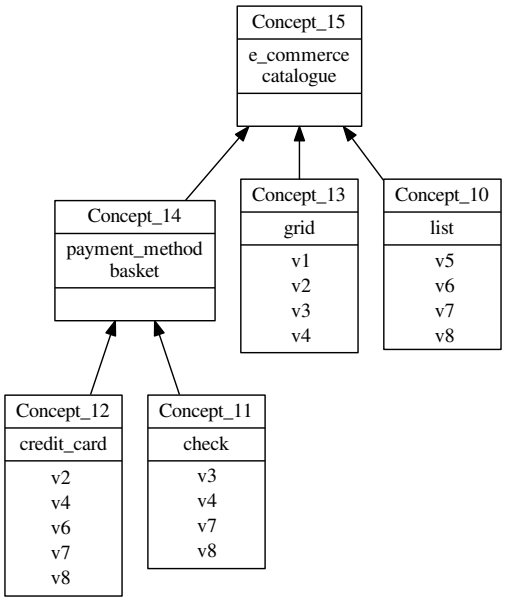



Figure 4: AC-poset associated with the formal context of Table 1

The first challenge concerns the set of valid configurations depicted by the synthesised FM, called the FM *configuration semantics* [4, 26]. It is known that FM as defined in the FODA report are not logically complete [27]. As a consequence, current methods permit to obtain FMs that describe all the configurations corresponding to a set of existing variants, but in some cases they also describe extra configurations: their configuration semantics then does not exactly correspond to the original set of variants. This can lead to some difficulties if the FM is used to perform SPLE tasks as product selection or product derivation. In these cases, a user can be allowed to select a subset of features which does not correspond to any existing variant. Adding any logical formula as a cross-tree constraint to the model can be a solution to restrict the valid configurations to the exact set of original variants. But here again, one can face two other issues: 1) cross-tree constraints can take the form of a complex and incomprehensible propositional formula, and 2) the number of cross-tree constraints can become overwhelming. An example of the second issue appears in the case study of Ryssel et al. [10] where an FM with 42 features needs 1772 cross-tree constraints to describe the exact set of original variants. However, for specific operations, such as predicting which non-existing variants may be easily derived from the set of existing ones, representing more configurations may be desirable; this concern is out of the scope of this paper. To synthesise FMs that most accurately represent the configuration semantics represented by the initial set of variants, while remaining understandable for a user is thus the first challenge.

The second challenge refers to the meaning of the feature hierarchy, also called *ontological semantics* [26]. In fact, the way features are related and how they are connected in the hierarchy also displays important information. For example, in Figure 1 (right), *credit_card* and *check* are sub-features of *payment_method* because they refine the concept represented by this feature. The meaning expressed through the hierarchy is important to maintain the SPL, understand its architecture and, as mentioned in [28], to be used as a basis for automated procedures. Therefore, the second main challenge of FM automated synthesis is to obtain a coherent feature hierarchy, i.e., that displays meaningful information regarding the domain. Semi-automated FM synthesis tries to reduce incoherence of the feature hierarchy by including users decisions in the process. A recent empirical study [15] has shown that, when it comes to FM extraction, semi-automated approaches outperform the other solutions, and thus user decisions are necessary at some point to avoid the extracted model to have a bad ontological semantics. This second challenge rises two complementary research questions:

**RQ1:** What part of the process of FM extraction can be fully-automated without altering its semantics?

**RQ2:** What part of the process of FM extraction necessitates user decisions?

FCA is a method for data analysis, knowledge representation and information management, which provides a natural hierarchical structure to any dataset composed of a set of objects and formal descriptions. It is founded

on a strong theoretical background while focusing on human-centered approaches [29], and has already shown its capacity to encompass variability information and support its extraction [21, 30, 10]. Part of question 1 is answered in the following section (Section 3) which presents a sound and complete FCA-based extraction method of feature relationships from variant descriptions. Then, Section 4 studies commonalities between FCA conceptual structures and FMs, which allows to complete the answer of question 1 and sketch a solution for question 2.

## 3. A sound and complete variability extraction method based on FCA

In the field of SPLE, FCA has been principally used to support information and logical relationship retrieval, for FM extraction from variant descriptions [30, 10], FM re-engineering [5], recovering SPL architecture [21], or feature identification in source code [20, 19]. The specialisation order between the attribute-concepts emphasises relationships between features that are true for the considered set of variants. In this section, we gather the different types of variability information which can be extracted from FCA conceptual structures. We study existing methods in order to obtain an efficient generalised process which is proved sound and complete thanks to the properties given by Ganter and Wille [17].

### *3.1. Extracting binary implications*

Ganter and Wille show that all binary implications that hold in a context can be read in the associated concept lattice, and conversely. Concept lattices thus support sound and complete extraction of binary implications [17, p.80]:

**Property 1.** *Given two features $f_1$ and $f_2$ respectively introduced in concepts $C_1$ and $C_2$, $C_2 \leq_s C_1 \iff f_2 \rightarrow f_1$*

In other words, if $C_1$ is a super-concept of $C_2$, then all variants having $f_2$ necessarily have $f_1$, because features are inherited from top to bottom in the structure. Therefore, to extract binary implications, only the attribute-concepts are needed: so they can be extracted from the AOC-poset and AC-poset as well. Binary implications can be found by following the arrows in the Hasse diagram of the conceptual structures. For instance, in Figures 2, 3 and 4, the feature *payment_method* is introduced in a super-concept of the concept introducing the feature *check*, so we can extract the implication *check* $\rightarrow$ *payment_method*.

This extraction method is used from the concept lattice by Shatnawi et al. [21], and from the AC-poset by Al-Msiedeen et al. [30] and Ryssel et al. [10]. We will extract binary implications from the AC-poset which is the smaller structure and thus supports the most efficient extraction: applying this method on an AC-poset has a complexity in $O(|A|^2)$, as $|A|$ is the maximum number of attribute-concepts and that potentially each pair of attribute-concepts has to be checked.

### *3.2. Extracting co-occurrences*

Co-occurring features are a particular case of double binary implications, that may be expressed using Property 2:

**Property 2.** *Given two features $f_1$ and $f_2$ introduced in the same concept $C$, we have $f_2 \leftrightarrow f_1$*

This property is due to the double application of Property 1, with $f_1$ and $f_2$ introduced in $C$, and therefore $f_1 \rightarrow f_2$ and $f_2 \rightarrow f_1$. It may be easily read in attribute-concepts of the associated conceptual structures. For instance, in Figures 2, 3 and 4, we can see that features *payment_method* and *basket* are co-occurrent because they are both introduced in *Concept_14*.

Co-occurring features are extracted from the concept lattice by Shatnawi et al. [21] and from the AC-poset by Al-Msiedeen et al. [30]. They are not taken into account in the work of Ryssel et al. [10]. Here again, we will use the AC-poset to extract this information. This process has a complexity in $O(|A|)$, as every attribute-concept has to be checked once to detect all co-occurrences.

### 3.3. Extracting mutual exclusions (mutex)

If two features $f_1$ and $f_2$ of a formal context are mutually exclusive (represented by an exclude CTC in FMs), then they are never present together in any variant of the formal context. This means that the set of variants having $f_1$ and the set of variants having $f_2$ are disjoint. Because the extent of an attribute-concept represents all the variants of the formal context possessing the attributes introduced in that concept, we can then naturally formulate the following property:

**Property 3.** *Given two concepts $C_1$ and $C_2$ respectively introducing features $f_1$ and $f_2$, $f_1 \rightarrow \neg f_2 \iff Ext(C_1) \cap Ext(C_2) = \varnothing$*

Testing the intersection of the extents of each pair of attribute-concepts is a sound and complete manner to extract all mutex. For instance in Figure 4, the intersection of *Concept_13* and *Concept_10* extents is empty, thus *grid* $\rightarrow \neg$ *list*. Another way to extract this information is to compute the greatest lower-bound of the two attribute-concepts $C_1$ and $C_2$. The greatest lower-bound of a pair of concepts represents a concept whose extent is the intersection of the extents' pair (because concept lattices are closed by intersection). Indeed, when the extent of their greatest lower bound is empty, it means that $f_1$ and $f_2$ never appear together in any variant. As a consequence, if the greatest lower-bound of two attribute-concepts is the bottom-concept, and that the bottom-concept has an empty extent, then attributes introduced in these concepts are mutually exclusive.

Shatnawi et al. [21] extract all the mutex by checking all incomparable concepts in a concept lattice, with a complexity in $O(|V|^3)$ with $|V| \leq 2^{min(|O|,|A|)}$. Al-Msiedeen et al. define a heuristics in $O(|A|^2)$, but the extraction is not complete. Finally, Ryssel et al. [10] add the negation of all features in a new formal context, and compute the binary implications including the negations to obtain all mutex. The number of attributes is multiplied by 2, so the complexity is still in $O(|A|^2)$, plus the computation of the AC-poset augmented with the negation.

As we try to extract all information from one structure to avoid computing several structures, we will test the intersection of attribute-concepts' extent. As intersection may be performed in $O(2|O|)$, the complexity of the process is in $O(|O|.|A|^2)$.

### 3.4. Extracting feature-groups

A group of features rooted on a parent feature means that, in any variant having the parent feature, at least one of the features from the group is always present. An or-group does not specify the maximal number of features that may be selected (thus representing a cardinality [1..n]), and a xor-group constrains this number to 1 (thus representing a cardinality [1..1]). Or-groups and xor-groups thus have the following properties: 1) each feature involved in a group implies the parent feature, and 2) in each variant, the parent feature must always be present with at least one feature of the group. Note that in an FM configuration set, each possible combination of features involved in an or-group appears at least in one valid configuration. However, as we work with potentially incomplete product descriptions, finding or-groups observing this "strong" property is very unlikely. Therefore, we only seek to detect "weak" or-groups, i.e., respecting at least the two previous properties (which is also the case of the or-group extraction in [4]). In the rest of the paper, the or-groups are considered "weak". Xor-groups must also verify the fact that in each variant the parent feature appears with at most one feature of the group; they are a particular case of or-groups. In what follows, we show how to detect or-groups, and then how to detect xor-groups among them. We will consider a feature $f_0$ and its attribute-concept $C_0$.

#### 3.4.1. Detecting or-groups

Firstly, all features from a group imply the same parent feature. Thus, if $f_0$ is a group parent, then the potential features of the group are necessarily introduced in sub-concepts of $C_0$, as they are all the features implying $f_0$.

**Property 4.** *Given a feature $f_0$ introduced in a concept $C_0$, if $f_0$ is a parent feature then a feature $f_i$ involved in the group is necessarily introduced in a concept $C_i \in \mathcal{AC}_\mathcal{K}$ such that $C_i <_{CL} C_0$ and $f_i \rightarrow f_0$.*

For instance in Figure 2, let us consider the feature *payment_method* introduced in *Concept_14*: the potential features of a group under this feature are only *check* and *credit_card*. The feature *grid* is not a candidate because it is introduced in a concept which is not a sub-concept of *Concept_14*.

Secondly, at least one feature of the group must appear with $f_0$ in any variant of the initial formal context. In other words, there must be no variant $v$ having $f_0$ without any feature introduced in a sub-concept of $C_0$. If such a variant exists, $f_0$ cannot be a group parent. This information can be read in a concept lattice: if there is an object-concept being a sub-concept of $C_0$, and no attribute-concept exists between $C_0$ and the object-concept, then $f_0$ cannot be a group parent. Moreover, $C_0$ cannot be an object-concept for the same reason. It is easier to verify this property in an AC-poset: if the union of the extents of the direct sub-concepts of $C_0$ is equal to the extent of $C_0$, then all variants having $f_0$ also have at least a feature introduced in a sub-concept of $C_0$, and $f_0$ is a group parent. Then, each subset of sub-concepts of $C_0$ in the AC-poset having the union of their extents equals to $C_0$'s extent forms an or-group.

**Property 5.** *Given a feature $f_0$ introduced in the concept $C_0 = (E_0, I_0) \in \mathcal{AC}_K$, and the attribute-concepts $C_i = (E_i, I_i) \in \mathcal{AC}_K, i \in \{1, 2, \ldots, n\} | C_i <_{CL} C_0$ and $\nexists C_j \in \mathcal{AC}_K, (C_j <_{CL} C_0$ and $C_i <_{CL} C_j)$. $f_0$ is a group parent feature if and only if $E_0 = \bigcup_{i=1}^{n} E_i$.*

In Figure 2, feature *catalogue* introduced in *Concept_15* is a group parent. The extent of *Concept_13* introducing *grid* is $\{v_1 - v_4\}$[2], and the one of *Concept_10* introducing *list* is $\{v_5 - v_8\}$. Their union is equal to $\{v_1 - v_8\}$, the extent of *Concept_15*: therefore, *grid* and *list* form an or-group under *catalogue*.

*3.4.2. Detecting xor-groups*

Xor-groups are particular or-groups such that no feature of the group appears with another feature of the group in any variant. Therefore, extents of their attribute-concepts are disjoint. To detect xor-groups, we must test or-groups for this property. In the following property, we consider that if an attribute-concept introduces more than one feature (i.e., co-occurrent features), then this set of features is represented by one feature.

**Property 6.** *Given $\{C_1 = (E_1, I_1), C_2 = (E_2, I_2), \ldots, C_n = (E_n, I_n)\}$ the set of concepts from $\mathcal{AC}_K$ respectively introducing features $f_1, f_2, \ldots, f_n$. The or-group composed of the feature set $\{f_1, f_2, \ldots, f_n\}$ and of their parent feature $f_0$ is a xor-group if and only if $\nexists (C_i, C_j) | C_i, C_j \in \{C_1, C_2, \ldots, C_n\}$ and $E_i \cap E_j \neq \varnothing$.*

In Figure 2, *Concept_13* and *Concept_10* have no variants in common in their extents: the or-group previously detected is thus a xor-group.

Shatnawi et al. [21] define two heuristics to detect feature-groups, which are neither sound nor complete. Al-Msiedeen et al [30] propose their own heuristics but they face the same problems. On the contrary, Ryssel et al. [10] express the problem of finding groups as finding a cover (for or-groups) and an exact cover (for xor-groups) in the AC-poset. It is equivalent to the method discussed here, and may be performed with a complexity in $O(2^{|A|})$.

*3.4.3. On groups' minimality*

An or-group is minimal if it is composed of a minimal number of attribute-concepts, i.e., if no attribute-concept may be removed without "breaking" the aforementioned group properties. Adding a feature to a minimal or-group thus leads to a non-minimal or-group. More specifically, all subsets of features introduced in sub-concepts of $C_0$ may be added to a minimal or-group to obtain non-minimal or-groups. Therefore, knowing the minimal or-groups and the feature hierarchy is enough to represent all possible non-minimal or-groups. In what follows, we consider only the minimal or-groups for the sake of concision.

By definition, the attribute-concepts involved in a xor-group cover exactly the extent of $C_0$, so all xor-groups are minimal.

*3.5. Identifying core features*

Features which are present in all variants [5, 30], also called *core-features*, give highlights on the common generic architecture of the SPL. Core-features are not evidently represented in FMs, but they are important variability information that can easily be identified in FMs with automated analyses. The *top-concept* (i.e., the only concept being the super-concept of all other concepts) is the concept having the biggest extent in the structure, which contains all the

[2] $\{v_i - v_j\}$ for $i < j$ is a shorten notation for the set $\{v_i, v_{i+1}, \ldots, v_j\}$

variants. Therefore, the top-concept intent encompasses all features that are shared by all the considered variants. The core-features are thus naturally introduced in the top-concept. The top-concept always exists in the concept lattice, but not necessarily in the AOC-poset and the AC-poset. If a sub-hierarchy does not possess a top-concept, then there are no core-features.

**Property 7.** *Given $C_\top \in C_K$ the concept such that $\forall C_i \in C_K, C_i \leq_{CL} C_\top$, then all features $f$ introduced in $C_\top$ are inherited by all concepts of the structures and thus $\top \rightarrow f$.*

For instance, we can see in Figures 2, 3 and 4 that the feature *catalogue* introduced in the top-concept *Concept_15* is present in all e-commerce application variants.

### *3.6. Identifying dead features*

*Dead-features* are features which are not owned by any product, and then are absent of all the considered variants [5, 30]. In concept lattices, this kind of features are thus introduced in a concept having an empty extent; if such a concept exists, it corresponds to the *bottom-concept*. The bottom-concept is the only concept of the structure being the sub-concept of all other concepts. It has the biggest intent, which contains all features from the formal context, as it inherits features from all concepts of the structure. Therefore, the bottom-concept extent contains all variants described by all the features. If it has an empty extent and a nonempty intent, the features introduced in its intent thus are not shared by any variant.

**Property 8.** *Given $C_\bot = (E_\bot, I_\bot) \in C_K$ the concept such that $\forall C_i \in C_K, C_\bot \leq_{CL} C_i$, if $E_\bot = \varnothing$ then a feature $f$ introduced in $C_\bot$ is not shared by any variant and $f \rightarrow \bot$.*

In Figure 2, the bottom-concept (*Concept_0*) has an empty extent and does not introduce any feature, which means there are no unused features in this variant set. The AOC-poset and AC-poset of Figures 3 and 4 do not have a bottom-concept (because *Concept_0* of the concept lattice is a plain-concept), which leads us to the same conclusion.

In this section, we gave an overview of what variability information may be automatically extracted using FCA, and thus answers part of **RQ1**. In what follows, we study the link between boolean FMs and the extracted variability information studied here, and show how FCA properties help to complete the answer of **RQ1** and to delimit the answer of **RQ2**.

## 4. FCA structures and equivalence classes of FMs

In this section, we first analyse the way FMs and concept lattices express feature relationships. Then, we give details of a mapping between feature relationships which are expressed in these two different formalisms. In particular, we show that concept lattices represent equivalence classes of FMs having the same configuration semantics. Finally, to ease the visualisation of the useful feature relationships extracted using FCA, we introduce a simplified representation based on FCA structures, called an *Equivalence Class Feature Diagram* (ECFD).

It is noteworthy that the logical semantics presented in ECFDs may be computed (and have been computed) using propositional formulas and SAT-solvers. However, the mapping between FCA and FMs shows how an FCA structure embodies an FM logical semantics and parts of its structure, while being a compact and canonical representation of both the propositional formula and its models. Here, we study a structural framework that naturally highlights FMs logical semantics, contrarily to existing methods that are designed especially to extract this information. FCA structures allow to support functions of knowledge processing such as exploring, searching, recognising, identifying, analysing, investigating, deciding, improving, restructuring and memorising [29]. What motivates our work here is to better understand links between FMs and FCA in order to benefit from FCA qualities in knowledge representation in the field of SPLE.

### *4.1. FMs and their semantics*

*Configuration semantics.* We have seen previously that FMs describe a finite set of valid combinations of features (i.e., valid configurations) by defining relationships between them. This set of valid configurations represents the *configuration semantics* of the FM [26]. Table 1 represents the configuration semantics of the FM of Figure 1.

*Ontological semantics.* The type of the relationship that links a subset of features (e.g., refinement, requirement, mandatory selection, feature groups) has an ontological meaning: it gives knowledge on the feature interaction with regard to the modelled domain. The chosen relationship types connecting features constitute the *ontological semantics* of a FM [26]. In this document, we consider that both feature tree relationships and cross-tree constraints represent the ontological semantics of an FM. Reverse engineering FMs from product configurations is a challenging task in particular because of this ontological facet. The difficulty lies in the fact that several different FMs (i.e., having different ontological semantics) can be equivalent (i.e., having the same configuration semantics). Therefore, a domain expert intervention is needed to identify which one represents the most meaningful modelling [15, 26]. For instance, let us consider the FM of Figure 5. It has the same configuration semantics as the FM presented in Figure 1. However its ontological semantics is different: in Figure 5, the feature *payment_method* now refines *basket* instead of *e_commerce* through a mandatory relationship. This can be understood as "payment methods is a mandatory sub-feature of basket management". On the contrary, in the FM of Figure 1, their link can be understood as "payment methods and basket management are two independent sub-features of e-commerce applications, but they require each other". The challenge is to identify which FM has a more meaningful representation of e-commerce applications' variability.

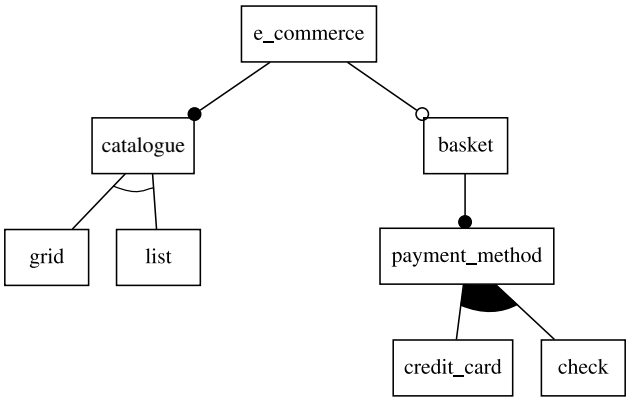


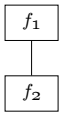
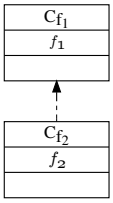
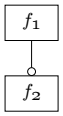
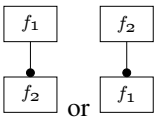
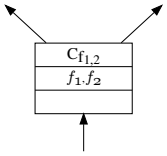
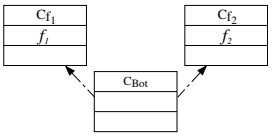

Figure 5: Example FM with the same configuration semantics as the FM of Figure 1, but with a different ontological semantics

*Logical semantics.* To our knowledge, the link between FMs and first-order logic was first made by Mannion in [31]. Ever since, FMs have been represented in the form of propositional formulas, where each feature corresponds to a propositional variable and where constraints are defined using propositional connectives. Mapping were established between FM and logical relationships [32, 4, 33] to obtain a propositional formula based on an FM such that the models of the formula correspond to the valid configurations of the FM. Since different propositional formulas having the same set of models are considered equivalent, and since an FM has a unique configuration semantics, the following proposition holds:

**Proposition 1.** *Let us consider the set of boolean FMs with $F$ as the feature set and $v_c \subseteq 2^F$ the associated set of valid configurations. Then, there exists a set of equivalent propositional formulas built on top of $F$ as the set of propositional variables, such that, given $\mathcal{F}_M$ one of these formulas and any interpretation $I : F \rightarrow \{true, false\}$, then ($I$ is a model of $\mathcal{F}_M$) iff ($\exists C \in v_c$ such that $\forall f \in F,\ I(f) = true$ iff $f \in C$).*

Each type of FM ontological relationship has an equivalence in propositional logic; thus, each ontological relationship corresponds to a logical relationship, that we will call its *logical semantics*. The columns *FMs* and *Propositional formula* of Tables 2, 3 and 4 present this mapping. Boolean FMs, as defined in Section 2, can depict 8 different ontological relationships (see Tables 2, 3 and 4): refinement relationships (1), mandatory (4) and optional (2) selection, requires (3, 5) and exclude (6) constraints, or-groups (7) and xor-groups (8). The "*FMs*" column depicts these different types of ontological relationships. Graphical relationships are illustrated by their corresponding edge decorations, and cross-tree constraints are identified by their textual notation. The "*Propositional formula*" column shows their logical semantics, as presented in [32, 4, 33]. The last column "*Concept lattices*" is discussed in Section 4.2.

Table 2: Mapping between FMs, propositional formulas, and concept lattices (except feature-groups)

| | FMs | | Prop. form. | Concept lattices | |
|---|---|---|---|---|---|
| 1 |  | $f_2 <_{FM} f_1$ | $f_2 \rightarrow f_1$ | $C_{f_2} \leq_{CL} C_{f_1}$ |  |
| 2 |  | $optional(f_1, f_2)$ | | | |
| 3 | $f_2 \rightarrow f_1$ | $f_2$ *requires* $f_1$ | | | |
| 4 |  | $mandatory(f_1, f_2)$ or $mandatory(f_2, f_1)$ | $f_1 \leftrightarrow f_2$ | $C_{f_1} =_{CL} C_{f_2}$ |  |
| 5 | $f_1 \rightarrow f_2$<br>$f_2 \rightarrow f_1$ | $f_1$ *requires* $f_2$<br>$f_2$ *requires* $f_1$ | | | |
| 6 | $f_1 \rightarrow \neg f_2$ | $exclude(f_1, f_2)$ | $f_1 \rightarrow \neg f_2$ or $f_2 \rightarrow \neg f_1$ | $Ext(C_{f_1} \sqcap C_{f_2}) = \varnothing$ |  |

In Table 2, **Row (1)** represents refinement relationships between features, i.e., that feature $f_2$ is a child-feature of $f_1$. The notation $f_2 <_{FM} f_1$ means that $f_2$ is introduced in a lower level of the branch introducing $f_1$ in the feature tree. We recall that, during a configuration selection, child features can be selected only if their parent features are already selected. **Row (2)** shows an optional relationship between a feature $f_1$ and its direct sub-feature $f_2$. This edge expresses the fact that there is no constraint on this selection: when $f_1$ is selected, its sub-feature can be selected, or not. However, $f_2$ still refines $f_1$. **Row (3)** depicts a requires CTC, which states that "if $f_2$ is selected, $f_1$ also has to be selected". These three ontological relationships have the same logical semantics, and can be represented by the implication $f_2 \rightarrow f_1$.

**Row (4)** (left) represents a mandatory relationship between a feature $f_1$ and its direct sub-feature $f_2$: when the parent feature is selected, the child feature is also selected. Note that $f_2$ also refines $f_1$. **Row (5)** shows the particular case of circular requires CTCs, i.e., when $f_1$ requires $f_2$ and $f_2$ requires $f_1$. These two types of relationships have the same logical semantics, and can be represented by a logical equivalence $f_1 \leftrightarrow f_2$.

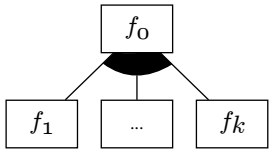
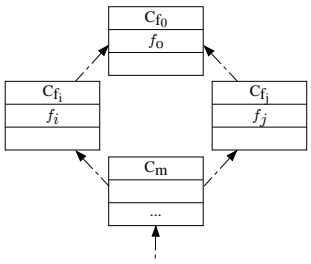
**Row (6)** depicts exclude CTCs: if $f_1$ is selected in a configuration, then $f_2$ cannot be selected, and conversely. This mutual exclusion can be logically expressed by $f_1 \rightarrow \neg f_2$ or $f_2 \rightarrow \neg f_1$.

Table 3 presents the mapping between or-groups and propositional logic. Let us consider $F = \{f_1, \ldots, f_k\}$ the set of features involved in an or-group, and $f_0$ the parent-feature of this group. All configurations having one of the features in $F$ also necessarily have $f_0$, as each feature of $F$ refines $f_0$. Conversely, configurations having the parent-feature necessarily have one of the feature of the or-group, thus it can be logically written by $f_0 \rightarrow (f_1 \vee \ldots \vee f_k)$.

Finally, Table 4 presents the mapping of xor-groups. Features involved in a xor-group have a similar behaviour as the ones involved in an or-group. But in addition, features of $F$ are mutually exclusive, as in exclude cross-tree constraints. Their logical semantics is then the same as or-groups, but with mutual exclusions between each pair of features in $F$: $f_0 \rightarrow (f_1 \oplus \ldots \oplus f_k)$.

We can see in this mapping that different types of ontological relationships have the same logical semantics; these ontological relationships can thus be represented by the same equivalent propositional formulas. However, even though their logical semantics is equivalent, their ontological semantics still represents different domain knowledges. Thus, one logical relationship may correspond to several ontological ones, as illustrated in Figure 6.

Table 3: Mapping: or-groups (7)

| FMs | |
|---|---|
|   | $or(f_0, F) \mid F = \{f_1, f_2, ..., f_k\}$ |
| **Propositional formula**: $f_0 \rightarrow (f_1 \vee \ldots \vee f_k)$ | |
| Concept lattices | |
|   | $\forall f \in F, C_f \leq_{CL} C_{f_0}.$<br>$Ext(C_{f_1}) \cup ... \cup Ext(C_{f_k}) = Ext(C_{f_0}).$<br>$\forall f_i \in F,$<br>$Ext(C_{f_1}) \cup ... \cup Ext(C_{f_k}) \setminus Ext(C_{f_i}) \neq Ext(C_{f_0}).$<br>$C_{f_0} \notin OC$ |

**Proposition 2.** *An ontological relationship can be associated to one and only one logical semantics, but a logical relationship may correspond to several different FM ontological relationships.*

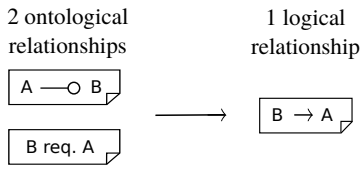



Figure 6: Asymmetric mapping between ontological relationships and logical relationships

*4.2. Concept lattices and their semantics*

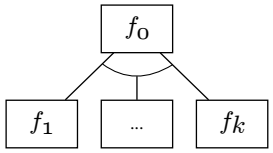
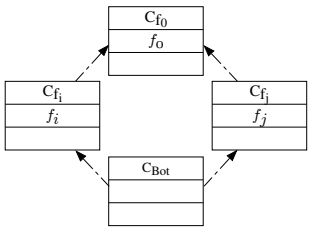
Note that, in the following section, all of the properties stated for concept lattices can be extended to AOC-posets and AC-posets, sometimes with variants which are indicated when necessary.

As the set of objects (i.e., valid configurations in our case) of formal contexts can be read in concept lattices, they present a *configuration semantics* as for FMs. Also, as presented in Section 3, concept lattices highlight logical relationships between their elements, and thus include a *logical semantics* as well. A formal context may be seen as a tabular representation of all models of a set of equivalent propositional formulas; therefore, the concept lattice associated to this formal context can be seen as a structural framework embodying both the propositional formulas and their models. Since a unique concept lattice can be built from a formal context, and that a unique formal context corresponds to a concept lattice [17], the following proposition holds:

**Proposition 3.** *Let any propositional formula $\mathcal{F}$ on the variable set V, and $\mathcal{M}_\mathcal{F}$ the set of its models. There exists a unique concept lattice $\mathcal{L}_\mathcal{F}$ which is built on top of the formal context $K = (\mathcal{M}_\mathcal{F}, V, R)$ and $(\forall m \in \mathcal{M}_\mathcal{F}, \forall v \in V, (m, v) \in R$ iff $m(v) = true)$.*
*Reciprocally, let $\mathcal{L}$ be a concept lattice and $K = (O, A, R)$ its associated formal context. We can associate to $\mathcal{L}$ a set of equivalent propositional formulas, built on top of A as the set of propositional variables and such that their models are in a one-to-one mapping with O, and for a model m associated with $o \in O$, $\forall a \in A, m(a) = true$ iff $(o, a) \in R$.*

However, concept lattices only emphasise logical relationships and do not explicitly represent any kind of ontological information, therefore they do not have an *ontological semantics*.

Table 4: Mapping: xor-groups (8)

| **FMs** | |
|---|---|
|  | $xor(f_0, F) \mid F = \{f_1, f_2, ..., f_k\}$ |
| **Propositional formula**: $f_0 \rightarrow (f_1 \oplus \ldots \oplus f_k)$ | |
| **Concept lattices** | |
|  | $\forall f \in F, C_f \leq_{CL} C_{f_0}$.<br>$\forall f_i, f_j \in F \mid f_i \neq f_j,\ Ext(C_{f_i} \sqcap C_{f_j}) = \varnothing$.<br>$\{Ext(C_{f_1}), ..., Ext(C_{f_k})\}$ is a partition of $Ext(C_{f_0})$.<br>$C_{f_0} \notin OC$ |

The columns *Propositional logic* and *Concept lattices* of Tables 2, 3 and 4 present the mapping between logical relationships and how they can be read/extracted from concept lattices. They summarise the theoretical analysis presented in the first 4 subsections of Section 3; here we just discuss the patterns corresponding these relationships in a concept lattice.

**Row (1,2,3)** If the concept introducing the feature $f_2$ is a sub-concept of the concept introducing $f_1$ (denoted by $C_{f_2} <_{CL} C_{f_1}$), then all configurations having $f_2$ also have $f_1$, hence the implication $f_2 \rightarrow f_1$. Note that this implication also stands when $f_1$ is introduced in the same concept as $f_2$ (denoted by $C_{f_2} =_{CL} C_{f_1}$), but in this case we also have $f_1 \rightarrow f_2$.

**Row (4,5)** Logical equivalences can be read in concept lattices by identifying concepts introducing more than one feature: if $f_1$ and $f_2$ are both introduced in the same concept (denoted $C_{f_1} =_{CL} C_{f_2}$), then $f_1 \leftrightarrow f_2$.

**Row (6)** Mutual exclusion can be read in a concept lattice by computing the greatest lower-bound of two concepts, respectively introducing $f_1$ and $f_2$ (denoted $C_{f_1} \sqcap C_{f_2}$). In AOC-posets and AC-posets, contrarily to concept lattices, mutually exclusive features are introduced in concepts which do not have a lower bound, because the bottom-concept is empty (it is a plain concept) and thus it is not represented in the sub-orders.

Table 3 presents the mapping of or-groups logical semantics into concept lattices. Given $F$ the features involved in an or-group, and $f_0$ the parent-feature of this group. All configurations having one of the features in $F$ also necessarily have $f_0$, and conversely, configurations having the parent-feature necessarily have one of the feature of the or-group; thus, the union of the extents of concepts introducing features of $F$ is equal to the extent of the concept introducing $f_0$. Finally, the concept introducing the parent-feature of an or-group is not an object-concept, as at least one feature of $F$ has to be selected. We denoted it by $C_{f_0} \notin OC$, $C_{f_0}$ being the concept introducing the parent-feature of the group, and $OC$ the set of object-concepts of the concept lattice.

Finally, Table 4 presents the mapping of xor-groups logical semantics into concept lattices. Xor-groups are like or-groups, but the greatest lower bound of concepts introducing the features of $F$ has an empty extent. Indeed, features involved in a xor-group and features involved in an *exclude* cross-tree constraint have a similar behaviour, as they are mutually exclusive in both situations.

### *4.3. Mapping's conclusions*

Regarding the two previous subsections, a *mapping* can be established between feature relationships expressed in FMs (having ontological semantics) and feature relationships extracted from conceptual structures (having logical semantics). This mapping shows that the logical semantics of an FM can be represented as a conjunction of binary implications, equivalences, mutex, or-groups and xor-groups. We have shown in Section 3 that these five types of logical relationships can be extracted in a sound and complete manner from FCA conceptual structures. This leads us to this statement:

**Proposition 4.** *All logical relationships representing the logical semantics of boolean FMs can be read and extracted from concept lattices, AOC-posets and AC-posets.*

In the field of FM reverse engineering, such a mapping could ease the detection of potential FM relationships in FCA structures generated from product descriptions. Altogether, it permits to steer the FM extraction, and narrow the number of choices which can be taken by the domain expert (e.g., hierarchy, feature-groups). Moreover, it shows that the entire logical semantics is contained in a unique structure.

Let us consider an FM (named $\mathcal{FM}$) and a concept lattice (named $CL_{\mathcal{FM}}$) having the same configuration semantics (i.e., the concept lattice is built from the formal context depicting the FM valid configurations). We have seen previously that an FM has a unique set of valid configurations, and that a unique concept lattice can be built from a given formal context, thus the following proposition holds:

**Proposition 5.** *Given an FM and its set of configurations, there always exists one unique concept lattice associated with the formal context representing exactly this configuration set.*

The variability extraction approach using FCA is sound and complete: all logical relationships (amongst the 5 presented in the mapping) that are true for the considered set of configurations are thus extracted. Therefore, as $CL_{\mathcal{FM}}$ has the same configuration semantics as $\mathcal{FM}$, the logical semantics of $\mathcal{FM}$ can be extracted from $CL_{\mathcal{FM}}$. More precisely, one can establish a matching between the ontological relationships of $\mathcal{FM}$ and the logical relationships extracted from $CL_{\mathcal{FM}}$. This is illustrated in Figure 7.

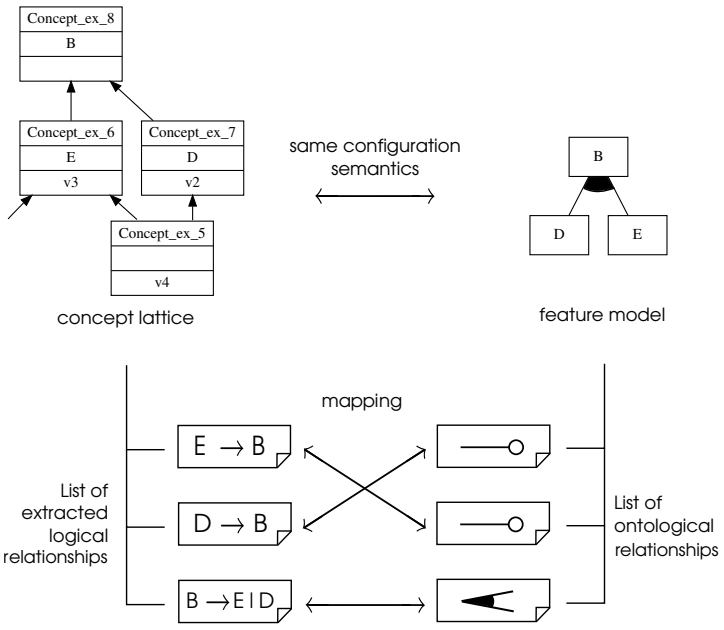



Figure 7: Some logical relationships from a concept lattice can be mapped to a corresponding FM ontological relationship

However, because of Proposition 2, one cannot associate with certainty an ontological semantics to the extracted logical dependencies without the initial FM or domain knowledge, as several choices can be possible. As a consequence, a concept lattice can be mapped to several different FMs (i.e. different ontological semantics), as long as these FMs possess the same logical semantics. As the configuration semantics can be computed from the logical semantics (i.e., list of models verifying the propositional formula), equivalent FMs (i.e. having the same configuration semantics) have all the same logical semantics, and only their ontological semantics differs. Thus, all the FMs having the same configuration semantics can be mapped to their equivalent concept lattice.

**Proposition 6.** *Let $FM$ be the set of FMs with $F$ as the feature set and $v_c \subseteq 2^F$ as the associated set of valid configurations. There exists a unique concept lattice (resp. AOC-poset and AC-poset) $\mathcal{L}_{FM}$ built on the formal context $K = (v_c, F, R)$ such that $\forall\ C \in v_c, \forall f \in F, (C, f) \in R$ iff $f \in C$.*

To sum up, one can map several FMs into the same concept lattice, if they have the same configuration semantics: each concept lattice built from a set of valid configurations represents an *equivalence class* of FMs (Figure 8).

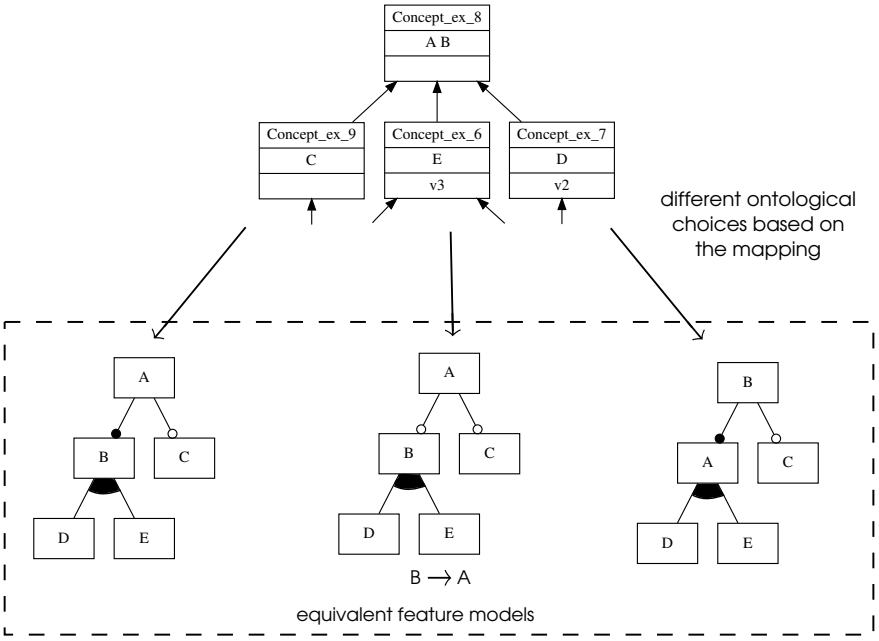



Figure 8: Concept lattices represent FM equivalence classes

### *4.4. A DSML for equivalence class of FMs*

To ease the mapping and the analysis of FCA structures, we decided to only keep the information that is necessary to represent the "equivalence classes" of FMs. In fact, this information is sometimes difficult to read in the lattice (e.g., feature-groups). We called the set of extracted information, in the form of logical relationships, an *equivalence class feature diagram* (ECFD). An ECFD hence symbolises the 5 types of information extracted from concept lattices previously introduced in Tables 2, 3 and 4. Figure 9 presents the meta-model of an ECFD.

`EquivalenceClass` can possess two types of `Element`. A `VariabilityBlock` groups a set of co-occurring `Features` that can be manipulated as a single entity. A `Group` gathers at least two variability blocks, and represents a minimal or-group, or a xor-group. An `Implication` from an element (`premise`) to a variability block (`conclusion`) states that when the features of the element are present in a configuration, the features of the block are necessarily present too. The variability blocks involved in a group cannot be the `conclusion` of the group. A group is always the premise of an implication, i.e., all blocks from the group imply the same variability block. Indeed, all variability blocks involved in a group have the same parent: an implication from a group to a variability block represents an implication from each variability block of the group toward the same parent variability block. Finally, mutual exclusions (`Mutex`) can be defined between two variability blocks. Note that the ECFD is not always a tree, and that groups can overlap.

An ECFD represents the logical semantics of a set of configurations, stemming from the information extracted from concept lattices. Thus, it represents all the FMs having the same configuration semantics, and each one of them matches this structure. Moreover, their construction is deterministic, i.e., a unique ECFD can be built from a set of variant descriptions. In the following proposition, we write that a formula belongs to a configuration by notation abuse.

**Proposition 7.** *Let $v_c$ be the configuration semantics of a set of FMs, and $F$ their feature set. Then, there exists a unique ECFD $E = (F_E, VB_E, OR_E, XOR_E, M_E, Imp_E)$ such that:*

- $F_E = F$.

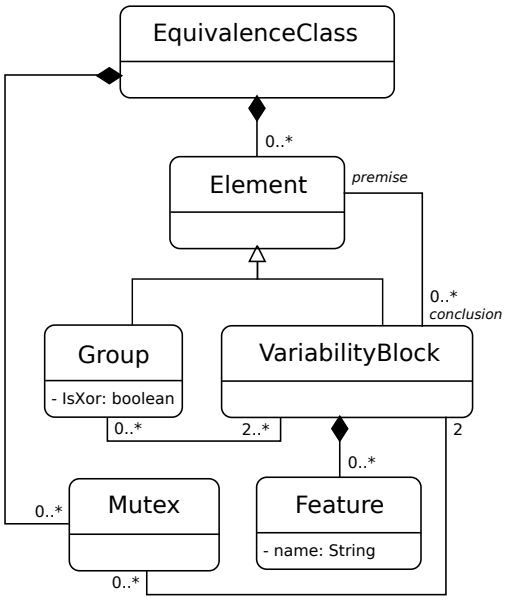



Figure 9: Meta-Model of Equivalence Class Feature Diagrams

- $VB_E$ *forms a partition of* $F_E$ *such that* $\forall f_1, f_2 \in F_E$ *with* $(\forall\ C \in v_c, f_1 \in C \iff f_2 \in C)$, *we have* $\exists b \in VB_E$ *and* $f_1 \in b, f_2 \in b$.
- $g \in OR_E$ *is a pair* $(b_0, \{b_1, \ldots, b_n\})$ *with* $b_i \in VB_E, i \in \{0, 1, \ldots, n\}$ *such that* $\forall\ C \in v_c, b_0 \in C$ *iff* $(b_1 \vee \ldots \vee b_n) \in C$ *and* $\nexists b_k | (b_0, \{b_1, \ldots, b_n\} \setminus b_k) \in OR_E$.
- $g \in XOR_E$ *is a pair* $(b_0, \{b_1, \ldots, b_n\})$ *with* $b_i \in VB_E, i \in \{0, 1, \ldots, n\}$ *such that* $\forall\ C \in v_c, b_0 \in C$ *iff* $(b_1 \oplus \ldots \oplus b_n) \in C$.
- $m \in M_E$ *is a pair* $m = (b_1, b_2), b_1, b_2 \in VB_E$ *such that* $\forall\ C \in v_c, (b_1 \notin C$ *or* $b_2 \notin C)$.
- $i \in Imp_E$ *is a pair* $i = (b_1, b_2), b_1, b_2 \in VB_E$ *such that* $\forall\ C \in v_c, (b_1 \in C$ *implies* $b_2 \in C)$.

We defined a textual and a graphical notation for the ECFD. The textual notation gathers the set of all logical relationships extracted from the concept lattice, as presented in the left column of Table 5. The graphical notation represents each kind of logical relationships with a graphical element close to the FM representation, as presented in the right column of Table 5.

Let us consider the concept lattice of Figure 2. The logical semantics of its equivalent FMs is presented in Figure 10. $G$, $L$, $Ch$ and $Cc$ are singleton variability blocks and are not represented. Variability blocks possessing more than one feature are called "composite variability blocks".

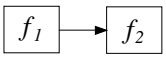
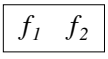
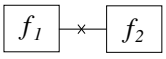
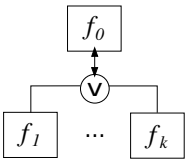
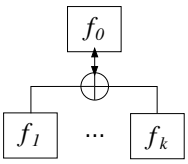
The graphical view of this ECFD is shown in Figure 11.

The logical semantics is a part of FMs that can be automatically extracted from variant descriptions (**RQ1**). Also, the ECFD allows to steer the possible choices of FM ontological semantics, and therefore the possible feature hierarchies of the final FM: it may be used to automatically infer some choices, thus reducing the number of decisions that have to be made by the user. The remaining choices (**RQ2**) are discussed in the following section.

## 5. Reverse engineering method

In this section, we describe the process of an FM reverse engineering method based on FM derivation from the ECFD obtained from a set of variant configurations. It allows us to identify when the ECFD may be used to infer a decision, and delimit the choices that remain to the user. We illustrate this process on an example about a cell phone SPL, borrowed from the work of Haslinger et al. [22].

Table 5: ECFD: a DSML for logical variability modelling

| Textual | Graphical |
|---|---|
| $f_1 \rightarrow f_2$ |  |
| $f_1 \leftrightarrow f_2$ |  |
| $f_1 \rightarrow \neg f_2$ or $f_2 \rightarrow \neg f_1$ |  |
| $f_0 \rightarrow (f_1 \vee \ldots \vee f_k)$ |  |
| $f_0 \rightarrow (f_1 \oplus \ldots \oplus f_k)$ |  |

Composite variability blocks: $Ec \leftrightarrow Ca$
$Pm \leftrightarrow B$
Implications: $[Pm, B] \rightarrow [Ec, Ca]$
$G \rightarrow [Ec, Ca]$
$L \rightarrow [Ec, Ca]$
$Ch \rightarrow [Pm, B]$
$Cc \rightarrow [Pm, B]$
Minimal or-groups: $[Pm, B] \rightarrow (Ch \vee Cc)$
Xor-group: $[Ec, Ca] \rightarrow (G \oplus L)$
Mutex: $\emptyset$

Figure 10: Textual representation of the ECFD extracted from the concept lattice of Figure 2

### *5.1. Method overview*

Our method consists in assisting step-by-step the association of FM ontological relationships to the logical relationships depicted in an ECFD; it is relying on the mapping between FM and concept lattice relationships, applied on an ECFD.

Let us consider the mapping. Given the ECFD obtained from a set of variant descriptions, a user has the possibility to browse through each logical relationship described by this ECFD (Figure 12 (1) and (2)) and to choose which ontological semantics he wants to assign to a relationship (Figure 12 (3)). For each type of logical relationship, the ontological ones which can be chosen by the user are restricted to the ones having the corresponding logical semantics, as stated in the mapping. Once all ECFD relationships are associated with an ontological semantics, the user obtains a representation corresponding to exactly one FM, as illustrated in Figure 12. Logical relationships having one possible ontological representation do not need user intervention, and may be automatically assigned.

This method has two advantages that motivated our work. On the one hand, by defining a mapping which does not change the logical semantics of the ECFD, derived FMs have similar configuration semantics. Configurations semantics may be slightly different, as the user may not retain all or-groups: thus, features involved in the discarded groups are linked by optional relationships instead, resulting in a broader set of configurations. A complex propositional formula may be added at the end to restrict the configuration semantics of the derived FM, as presented by She

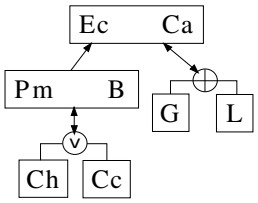



Figure 11: Graphical representation of the ECFD extracted from Figure 2

et al. [26] and Ryssel et al. [10]. On the other hand, the obtained FM has, by construction, an ontological semantics validated by the user, that we assume more meaningful than the one obtained with fully automated synthesis methods.

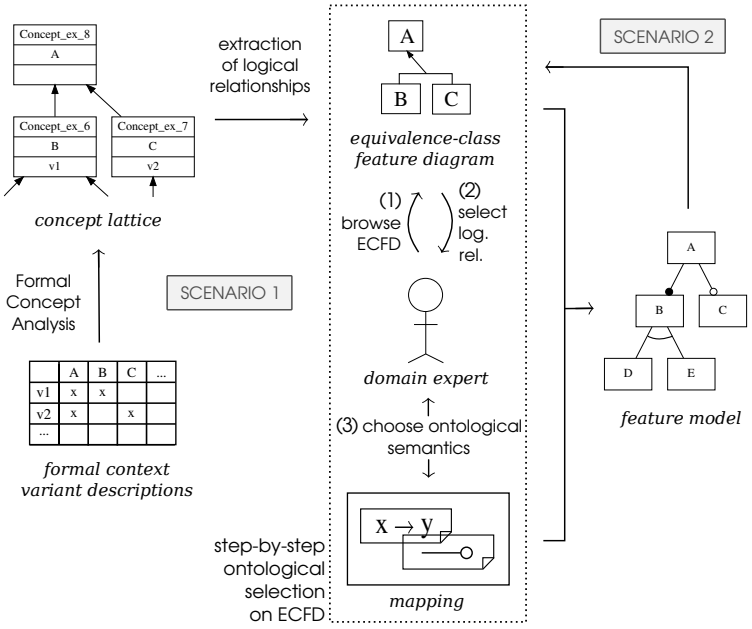



Figure 12: FM extraction method based on a step-by-step ontological semantics selection on an ECFD

We identified two scenarios which can benefit from the mapping and the ECFD:

**Scenario 1**: deriving a FM from a set of product descriptions.

**Scenario 2**: editing an FM with a correct configuration semantics, but which lacks consistency (i.e., with an incorrect ontological semantics). In fact, given an inconsistent FM, one can compute its corresponding ECFD (i.e., representing the logical semantics of the FM) and see what modifications may be done on the FM without altering its configuration semantics. This ECFD can be computed from the set of valid configurations of the FM; existing FM re-engineering is thus similar to extracting an FM from variant descriptions. However, in this scenario, the user may compare its choices in the ECFD with the existing FM ontological semantics. If the considered FM possesses too many valid configurations for them to be listed efficiently, hints about representing FM variability with FCA structures while avoiding combinatorial explosion are given in [34]. Similar process may be applied to derive the ECFD directly from the FM, therefore avoiding listing all its valid configurations.

### *5.2. Deriving FMs from an ECFD*

In what follows, we define some rules to ensure a correct FM derivation from an ECFD. Each rule is illustrated on the example of Figure 11.

1. First, the user has to indicate the root feature. He has to choose among the features depicted in variability blocks which are not premises of an implication. For example, in Figure 11, only the block containing *e_commerce*

(Ec) and *catalogue* (Ca) corresponds to this description. Therefore, the user has to choose between these two features. If the user chooses for example *catalogue* (Ca) as a root, this choice can lead to the feature models of Figure 16.

2. Also, we have seen before that different groups may overlap in the ECFD. Thus, the user has to choose which groups to keep in the final FM, while making sure a feature does not belong to more than one feature-group to respect FM semantics. In Figure 11, no group overlap, so they can be kept as is. We give two examples of ECFD having overlaping groups. Let us take as a first example an excerpt of an ECFD for another cell phone SPL, shown in the l.h.s of Figure 13. In this example, two groups overlap: the or-group $keyboard \vee Touchscreen$ and the or-group $Touchscreen \vee Speaker$. In this case, both groups are relevant so that to obtain a feature tree, the only solution is to clone the feature $Touchscreen$ and to make it appear twice in the resulting FM. If we now study another example, i.e., the excerpt of ECFD on the r.h.s of Figure 13, the same two groups overlap, but this time the group $Touchscreen \vee Speaker$ has for parent *Audiodevice*, and thus seems rather accidental. So that in this second case, the or-group $keyboard \vee Touchscreen$ can be kept, as shown in Figure 14.

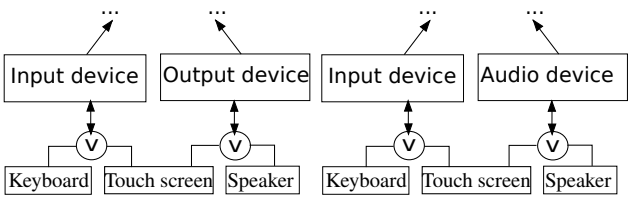



Figure 13: Two examples of ECFD with overlapping groups

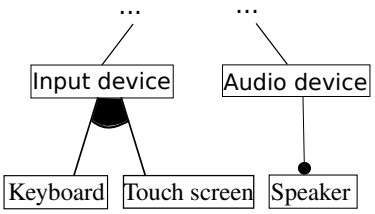



Figure 14: An excerpt of feature model from the ECFD of the r.h.s of Figure 13

3. Then, in order not to affect the logical semantics, logical relationships having only one ontological matching have priority on other ontological choices. As an example, let us slightly modify the ECFD of Figure 11, adding an arrow from $L$ towards $Cc$. We obtain the ECFD of Figure 15.
   Then, one cannot choose to assign $Cc$ as the parent feature of $L$, because $L$ is already involved in a feature-group with $G$, and this a relevant way to express the ontological semantics of the group. Therefore, the group has priority on the implication between $L$ and $Cc$, which can also be represented by a *requires* CTC.
4. Finally, when choosing the ontological semantics, the user has to care about assigning to each feature exactly one parent feature among the ones from the "super-blocks" (conclusions of the implications), or the ones sharing the same block. When a super-block contains one feature, it must be the parent of at least one feature of each sub-block. When a super-block contains more than one feature, one of these features must be the parent of at least one feature of each sub-block. Features of a sub-block have not necessarily the same parent. When several features are involved in the same feature-group, all the features from the group have the same parent feature. These rules permit to preserve the feature hierarchy of the FM. For instance, let us consider features *payment_method* (Pm) and *basket* (B) in the ECFD of Fig. 11. Let us suppose that we choose to assign $Ca$ as

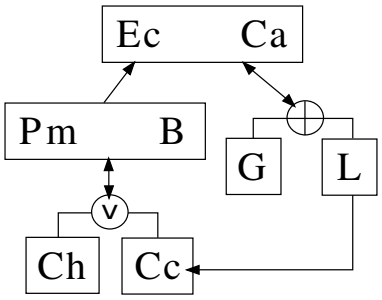



Figure 15: A modified version of the ECFD of Figure 11

the parent feature of *Pm*, and *Ec* as the parent feature of *B* (meaning that *payment_method* refines *catalogue* and that *basket* refines *payment_method*), as shown in Figure 16. Now, because *Pm* and *B* share the same box, we know from the mapping that they can be bound by a *mandatory* relationship (i.e., one of the two features refines the other) or a circular *requires* CTC (i.e., the two features are independent but require each other). Yet, a mandatory relationship cannot be chosen in this case, as it implies to designate a parent feature for *Pm* or *B*, whereas both of them already have a parent feature; in this case, a circular *requires* CTC is the only possible choice.

These rules allow to answer **RQ2**. To sum up, deriving FMs from an ECFD necessitates user's decisions for 1) selecting the feature-groups when necessary, 2) choosing a feature tree among the proposed ones (this includes choosing the root feature). Indeed, once the feature-groups have been selected, all the ontological choices concern the hierarchy. The ECFD already gives a partial hierarchy, i.e., it narrows the possible choices. Then, the user has to choose parent-feature among "super-blocks", and how to connect co-occurrent features from the same block with mandatory relationships, optional relationships and double requires CTCs. A feature ranking method as presented in [26] or in [16] may be put in place to assist the user to choose the best parent feature or even just to ease this activity. The mentioned methods use feature descriptions to compute similarity between features, and propose a ranking of the potential parent features of a given one. These methods, as they are based on external feature descriptions, allow to estimate some ontological semantics in an automated way; but, if no external information is provided, they cannot be applied. In these cases, some metrics may be computed anyway based on the information included in FCA conceptual structures. For instance, each concept can be associated with a frequency and a support, that may be used to characterise both features and feature relationships. Moreover, FCA provides a set of interestingness measures [35] that may be used to assist the user during the hierarchy choices when no ontological information is available. This is left as future work.

It is noteworthy that several works have already discussed the fact that ontological semantics of FMs resides in the hierarchy and the feature-groups [4, 26]. It is also important to state that FCA offers a theoretical background for **RQ1** and **RQ2** answers. Also, FCA allows to document the possible choices for the hierarchy and the feature-groups in a unique structure; Thus, some choices may be inferred automatically, and the ECFD permits to steer the user decisions through a graphical representation close to FMs. Here again, FCA's contribution here is to unify previous work in the same framework.

### *5.3. Examples of ontological semantics selections*

Figure 16 presents two possible ontological semantics selections of the ECFD of Figure 11. To represent the chosen ontological semantics on ECFDs, we reuse the edge decoration of FMs to represent mandatory and optional selection, and the textual notations to represent requires relationships ($\rightarrow$). To depict the chosen feature hierarchy, an arrow starts from each feature or feature-group and points towards its selected parent-feature. We highlight the selected root feature in the ECFD with an underline. The modifications added to the ECFD to show the chosen ontological semantics correspond to the aforementioned users' decisions. Feature-groups and exclude constraints keep the same notation we used in ECFD.

In the selection of the left-hand side, *Ec* is designated as the root feature. *Ca* shares the same block as *Ec*: they have to be linked by either a circular requires cross-tree constraint or a mandatory relationship. However, *Ca* cannot


Figure 16: Two different ontological semantics selections, applied on the ECFD of Figure 11. Choices of the left-hand side correspond to the FM of Figure 1, and the right-hand side to the FM of Figure 5

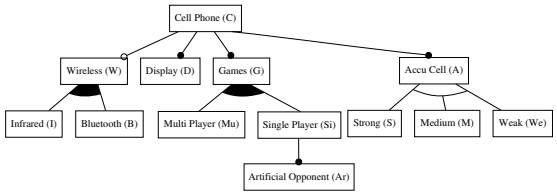
have another parent than *Ec* (because it has no "super-block"), and thus it is designated as a child-feature of *Ec*, linked by a mandatory relationship. The chosen parent-feature of the xor-group composed of *L* and *G* is *Ca* (but could have been *Ec*). *Pm* and *B* have the same parent-feature (*Ec*, but could have been *Ca*) and are linked by a circular requires CTC. Finally, the parent-feature of the or-group $\{Ch, Cc\}$ is *Pm* (but could have been *B*). These choices correspond to the FM of Figure 1.

The selection of the right-hand side shows an alternative where the user chooses to link *Pm* and *B* not by a circular requires CTC, but by a mandatory relationship from *B* (parent) to *Pm* (child). In this case, *Pm* can no longer be the child-feature of *Ec*, as it already has another parent-feature. These choices correspond to the FM of Figure 5.

### *5.4. Application: re-engineering FMs with ECFD*

Here we show how to use our method to edit an FM previously obtained with a reverse engineering method, in order to improve its ontological semantics (**scenario 2**). First, we present the input reverse engineered FM and its ontological issues. Then, we show how our method may help to steer the practitioners' choices during the FM re-engineering.

#### *5.4.1. Running example: cell phone product line*




$MultiPlayer \rightarrow Wireless\ ;\ Bluetooth \rightarrow Strong$

$MultiPlayer \nleftrightarrow Weak$

| | C | W | I | B | A | S | M | $W_e$ | D | G | $M_u$ | $S_i$ | $A_r$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | x | | | | x | x | | | x | x | | x | x |
| 2 | x | | | | x | | x | | x | x | | x | x |
| 3 | x | | | | x | | | x | x | x | | x | x |
| 4 | x | x | | x | x | x | | | x | x | | x | x |
| 5 | x | x | | x | x | x | | | x | x | x | | |
| 6 | x | x | | x | x | x | | | x | x | x | x | x |
| 7 | x | x | x | | x | x | | | x | x | | x | x |
| 8 | x | x | x | | x | x | | | x | x | x | | |
| 9 | x | x | x | | x | x | | | x | x | x | x | x |
| 10 | x | x | x | | x | | x | | x | x | | x | x |
| 11 | x | x | x | | x | | x | | x | x | x | | |
| 12 | x | x | x | | x | | x | | x | x | x | x | x |
| 13 | x | x | x | | x | | | x | x | x | | x | x |
| 14 | x | x | x | x | x | x | | | x | x | | x | x |
| 15 | x | x | x | x | x | x | | | x | x | x | | |
| 16 | x | x | x | x | x | x | | | x | x | x | x | x |

Figure 17: FM about cell phones taken from [22] and its 16 valid configurations

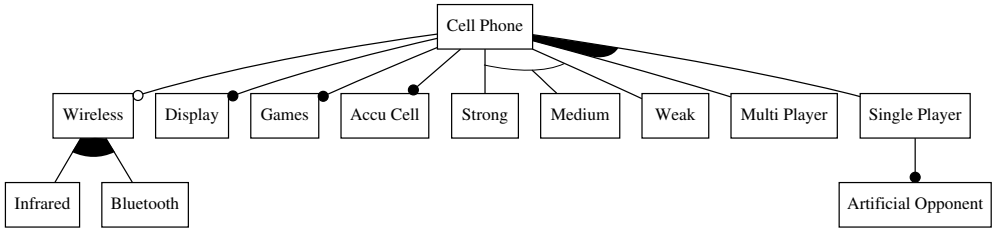
In what follows, we illustrate our method by applying it on a given reverse engineered FM in order to improve its ontological semantics. We use the example SPL about cell phones proposed by Haslinger et al. in [22].

In their paper, Haslinger et al. [22] propose algorithms to synthesise boolean FMs from a list of variants, organised in the form of a formal context that they called a *feature set table*. To illustrate their method, they choose to extract their input feature set table from an existing FM from `SPLOT` [36], which provides a reference with which they can compare their output FM. In what follows, we rely on this reference FM and its feature set table (i.e. list of valid configurations) to construct a conceptual structure with FCA. The FM used as reference is presented in Figure 17 (left).

In this SPL, each cell phone variant can eventually possess a *Wireless* feature, such as *Infrared* or *Bluetooth*. Both are possible. All variants own a *Display* feature and *Games*. The proposed games can be *Multi Player* or *Single Player*, a single player game always proposing to compete against an *Artificial Opponent*. Finally, each variant of this SPL possesses an *Accu Cell*, which can be either *Strong*, *Medium* or *Weak*. A variant supporting multi player games has to possess one of the wireless features and an accu cell cannot be weak. If bluetooth is available, the type of accu cell has to be strong. This FM displays 16 valid variant configurations; they are listed in the formal context presented in Figure 17 (right), analogous to the feature set table of [22].

In their paper, Haslinger et al. propose a set of algorithms to automate the synthesis of FMs from feature combinations. It is a fully automated method which produces FMs with the same configuration semantics as the original set of variants. However, the understandability and meaningfulness of the reverse engineered FMs still need to be assessed.

Figure 18 presents the output FM obtained when applying their method. Even though the reverse engineered FM




$MultiPlayer \rightarrow Wireless\,;Bluetooth \rightarrow Strong$

$MultiPlayer \rightarrow \neg Weak\,;Weak \rightarrow SinglePlayer$

Figure 18: Extracted FM from [22]

presents the same configuration semantics as Figure 17 (right), it differs in some points from the reference FM. Some of its ontological semantics is lost in the process, as, for example, the or-group $\{Strong, Medium, Weak\}$ does not refine *Accu Cell* any more, and the connection between the group and the feature does not even appear in the model. However, a good part of this semantics is still expressed in the FM: *Infrared* and *Bluetooth* refining *Wireless*, mandatory *Artificial Opponent* with *Single Player* feature. It should be pointed out that feature groups are accurate and meaningful in this example: *Strong*, *Medium* and *Weak* are mutually exclusive features, but are still part of the same group; *Multi Player* and *Weak* are also mutually exclusive, but they do not form a xor-group and so their exclusion is expressed by an exclude cross-tree constraint. But, the FM of Figure 18 also reveals a relationship which is true for the considered set of variants, but not expressed in the original FM: it appears in the form of the cross-tree constraint $Weak \rightarrow Single\ Player$, stating that variants with weak accu cells always support single player games. It is noteworthy that this kind of information can be difficult to read when one does not possess a reference FM as in this case, but only a reverse engineered one which lacks some refinement relationships.

### *5.4.2. Re-engineering FMs with ECFDs*

To guide the user into modifying the initial FM and improving its ontological semantics, the ECFD corresponding to the re-engineered FM must be extracted. Here, the set of valid configurations of the initial FM is small, so we can compute the associated AC-poset and extract the corresponding ECFD. As said previously, if the initial FM represents too much valid configurations to be listed, it is possible to 1) derive the logical semantics from the FM and 2) build the corresponding ECFD.

In this section, we follow the steps a user has to follow to edit the FM from Figure 18 and obtain the FM of Figure 17 (left). Figure 19 presents the AC-poset generated from the formal context of Figure 17 (right), and Figure 20 the corresponding ECFD.

Computing this ECFD reveals that 5 different minimal or-groups are possible, but most of them overlap:

1. $\{S, \{Si, Ar\}, I\}$
2. $\{S, \{Si, Ar\}, M\}$
3. $\{B, I\}$
4. $\{W, \{Si, Ar\}\}$

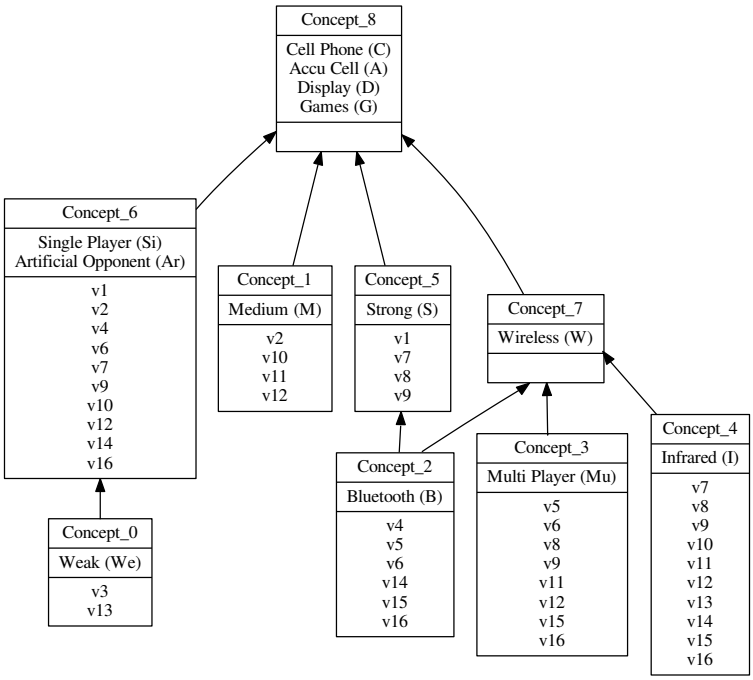



Figure 19: AC-poset associated with the formal context of Figure 17 (right)

5. $\{Mu, \{Si, Ar\}\}$

To ease the visualisation of the manipulated structure, the first choice we illustrate consists in choosing the feature groups to retain in the final FM. The user has to choose the feature-groups that seem the most consistent to him, while taking care that these groups do not overlap. To obtain the FM of Figure 17 (left), the user needs to select the two feature-groups $\{Mu, \{Si, Ar\}\}$ and $\{B, I\}$. The Figure 21 shows the previous ECFD minus the 3 or-groups not retained; it is therefore not the same ECFD as before, but an intermediate step between the first ECFD and the final FM.

Despite its apparent complexity, this diagram offers strong guidelines to the user as it retains a lot of choices:

1. the three feature-groups and the exclude constraint dependency $We \rightarrow \neg Mu$ have only one ontological semantics;
2. as seen before, the or-group $\{I, B\}$ has priority on the implication $B \rightarrow S$, the last one is thus necessarily represented by a requires CTC;
3. same conclusion can be made for $Mu \rightarrow W$.

Therefore, the user choices are reduced to:

1. select a root feature
2. the designation of a parent-feature for the feature $W$, the or-group $\{Mu, \{Si, Ar\}\}$ and the xor-group $\{We, M, S\}$;
3. select $We \rightarrow Si$ or $We \rightarrow Ar$;
4. how to express the co-occurrence of $Si$ and $Ar$;
5. how to express the co-occurrence of $C$, $G$, $D$ and $A$.

Figure 22 shows the ontological semantics of the extracted FM obtained with Haslinger reverse engineering method.

Let us see what kind of modifications our method permits to make, and select the meaningful ones, i.e., the ones corresponding to the reference FM. We have seen previously that the reverse engineered FM of Figure 18 lacks coherence regarding the parent-feature of the or-group $\{Multi\ player, Single\ player\}$ and the xor-group $\{Weak, Medium, Strong\}$, which is the root *Cell phone* for both of them. Though, one can see in the ECFD that *Cell phone* is not the only possible parent-feature for these groups, and that they can also refine either *Games*, *Display* or *Accu cell*. Features

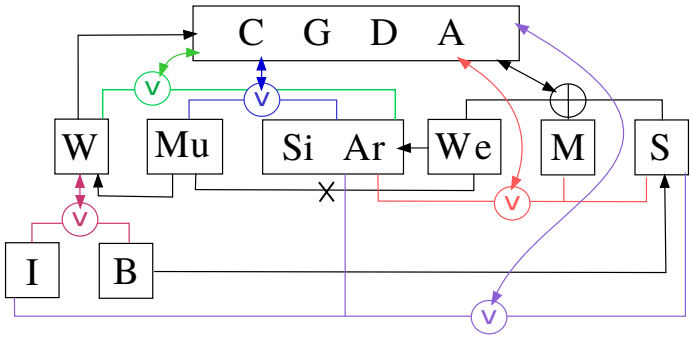



Figure 20: Equivalence class feature diagram of the SPL about cell phones

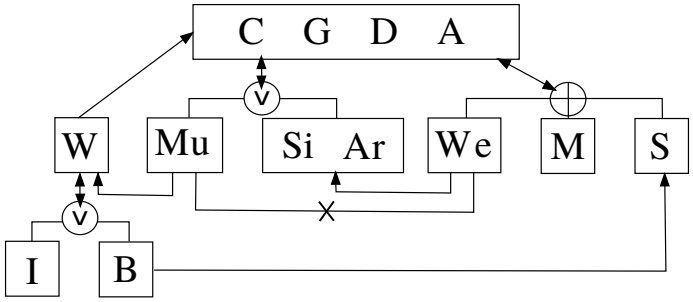



Figure 21: Equivalence class feature diagram of the SPL about cell phones without the non-consistent feature groups

*Multi player* and *Single player* refer to video game options, and thus the feature *Games* is more coherent as a parent-feature. Besides, the features *Weak*, *Medium* and *Strong* refer to a cell phone battery, and therefore refine the feature *Accu cell*. With these two modifications, we obtain the ontological semantics selection of Figure 23, corresponding to the initial FM depicted in Figure 17.

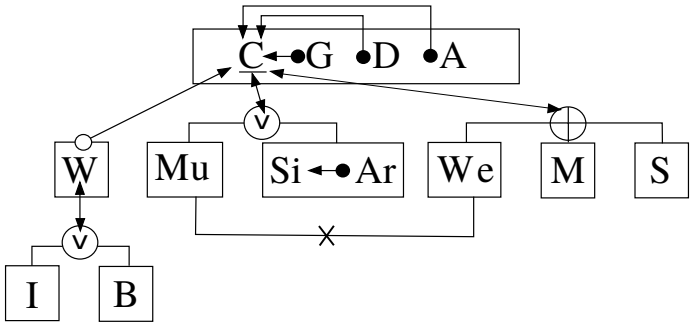
## 6. Applicability study of FCA on product descriptions

In previous sections, we have defined formal concept analysis and showed how two of its associated structures can be used to assist consistent feature models extraction from product descriptions. In what follows, we study their applicability in the software product line reverse engineering context. We discuss two aspects: 1) the size of these structures when generated from product descriptions, and 2) the viability of their generation and management with existing algorithms. Finally, we study sizes and characteristics of ECFDs corresponding to existing FMs taken from the SPLOT repository.

### *6.1. Dimensions of the conceptual structures*

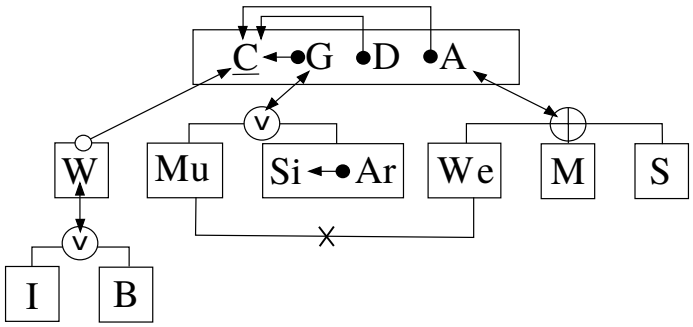
FCA is known to produce structures which exponentially grow with the size of the input. They can be difficult to generate and to exploit when their size is too important. In what follows, we define the size of a structure by its number of concepts. The size of a concept lattice and an AOC-poset depends on the number of attributes and of objects, and the form of the formal context. Let $|O|$ be the number of objects (here the variants) in a formal context, and $|A|$ the number of attributes (here the features). In a concept lattice, the number of generated concepts can reach $2^{min(|O|,|A|)}$. However, in an AOC-poset, this number cannot exceed $|O| + |A|$, and $|A|$ for an AC-poset.

Here, we present an experiment to determine an order of magnitude of the size of these two structures when they are used to organise a set of existing variants. Product descriptions can be found abundantly on the internet in the form of tabular data, called Product Comparison Matrices (PCMs). PCMs permit to compare a set of similar products depending on a set of features. They have been studied in the context of SPLE as product descriptions [37, 38]. Websites as Wikipedia propose numerous PCMs, including ones about software systems.




$B \rightarrow S$ ; $Mu \rightarrow W$; $We \rightarrow Si$

Figure 22: Ontological semantics of the extracted FM obtained with the Haslingers reverse engineering method presented in Figure 18




$B \rightarrow S$ ; $Mu \rightarrow W$ ; $We \rightarrow Si$

Figure 23: Ontological semantics choices, corresponding to the initial FM of Figure 17

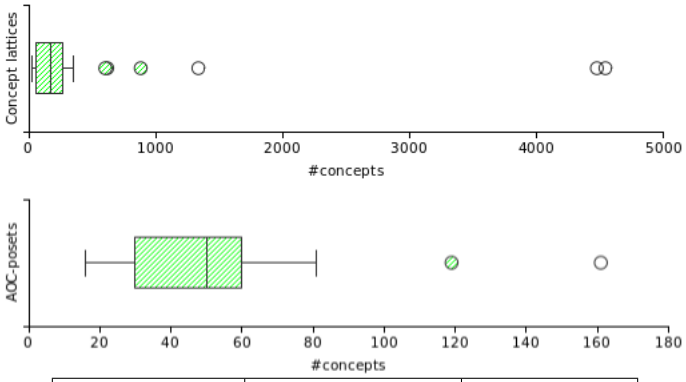
For this experiment, we took 40 Wikipedia's PCMs about software systems, and we converted them into formal contexts[3]. Because features displayed in PCMs can be multi-valued, we applied *binary scaling* on them to obtain binary attributes: a feature with $n$ different values in the PCM thus produces $n$ binary attributes (one attribute per possible value) in the formal context. These 40 PCMs were chosen randomly among all the PCMs present in the software comparison category of Wikipedia[4]. In the selected PCMs, the number of displayed products varies from 9 to 90, and the number of features from 6 to 21. Formal contexts converted from these PCMs possess from 10 to 239 binary attributes. From the obtained formal contexts, we built both concept lattices and AOC-posets, and gathered their number of concepts in Figure 24.

In this figure, we can see that most of the concept lattices possess from 50 to 300 concepts. In rare cases, they can reach about 1000 or 4500 concepts. The size of an AOC-poset is naturally smaller than the size of its associated concept lattice, but in the case of data obtained from product descriptions, the gap is very wide. In fact, with the same set of formal contexts, most of the produced AOC-posets possess from 30 to 60 concepts. Moreover, the largest obtained AOC-poset does not exceed 165 concepts. In our experiments, on average, AOC-posets are about 10 times smaller than concept lattices with the same data.

As a conclusion, despite the exponential growth of concept lattices, their dimension when structuring existing variant descriptions remains practicable even for the worst cases. Yet, AOC-posets are a recommended alternative to concept lattices in our approach, as they also permit to represent feature model equivalence classes but with a significantly smaller structure.

[3] http://www.lirmm.fr/recherche/equipes/marel/datasets/fca-and-pcm

[4] https://en.wikipedia.org/wiki/Category:Software_comparisons, last access in January 2018




| | Concept lattices | AOC-posets |
|---|---|---|
| Mean | 427.37 | 50.3 |
| Median | 174 | 50 |
| Minimum | 21 | 16 |
| Maximum | 4551 | 161 |
| First Quartile | 49 | 29.25 |
| Third Quartile | 282.5 | 61.25 |

Figure 24: Number of computed concepts with data from 40 Wikipedia's product comparison matrices

### *6.2. Complexity and performance*

We discuss in this section existing algorithms which permit to generate FCA structures from formal contexts and to perform operations on these structures. We study their complexity and their performance to compute structures of similar sizes as the one shown in Section 6.1.

In [39], Kuznetsov and Obiedkov compare the complexity and the performance of several algorithms generating concept lattices (*i.e.* computing the set of all concepts and ordering them). In their study, the authors take into account the number of attributes $|A|$, the number of objects $|O|$ and the density of the formal context $((|R| \,/\, (|O||A|)) \times 100$, with $|R|$ the size of the relation). In our experiments with Wikipedia's PCMs, contexts possess on average 60 attributes, 36 objects and a density of 1. Among all algorithms studied in [39], and with formal contexts which possess 100 attributes, 500 objects and a density of 4, the Bordat algorithm is the fastest to generate the concept lattice. Time complexity of Bordat algorithm is in $O(|O||A|^2|L|)$ (with $L$ the number of generated concepts), and it takes less than one second to compute the concept lattice associated with this kind of formal contexts.

Concerning AOC-posets, Berry *et al.* compare in [40] four algorithms generating Galois sub-hierarchy (as AOC-posets) from formal contexts. This study takes into account the same three formal context aspects as in [39]. From formal contexts possessing 100 attributes, 500 objects and a density of 1, the four presented algorithms generate AOC-posets in less than 50 ms. All these algorithms have a time complexity in $O(n^3)$.

The studied product descriptions extracted from PCMs of Wikipedia produce sparse formal contexts with a number of attributes and objects generally inferior to 100. The presented studies show that it is possible to generate concept lattices and AOC-posets in less than one second from this type of input. Moreover, dimensions of the generated structures permit to perform important operations rapidly. Therefore, the generation and manipulation of FCA structures are thus practicable when modelling variability of a set of existing variants.

### *6.3. Computing ECFDs from SPLOT repository FMs*

In this section, we seek to evaluate 1) if ECFDs can be entirely computed for small and large sets of valid configurations, 2) if extracted feature-groups are consistent, and 3) the number of choices proposed by an ECFD compared to the number of possible FMs it represents. To assess their consistency, tested sets of valid configurations are derived from existing FMs to compare the extracted relationships with the initial FMs. We conduct an experiment

on FMs gathered from the SPLOT repository, from which we compute the ECFDs associated with their set of valid configurations. To compute ECFDs from existing FMs, we established the process presented in Figure 25.

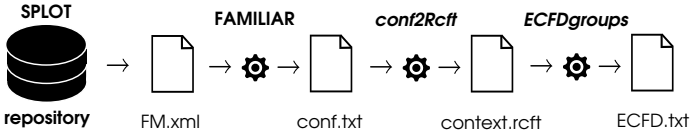



Figure 25: Process to compute ECFDs from FMs taken from the SPLOT repository

During this process, we first use two existing tools: SPLOT [36] and Familiar [41]. SPLOT proposes, aside from SPLE tools, a repository of about 700 feature models[5] created by the community. It permits to retrieve these feature models in the form of xml files. Feature models in xml format can be processed by Familiar, a tool for FM analysis which permits, among other operations, to compute the set of all valid configurations. Then, we used two programs that we developed for this process. The first one, *conf2RCFT*, takes a text file containing a list of configurations obtained with Familiar and translates it in a formal context in RCFT format. RCFT is a file format to represent formal contexts, that are used by the tool RCAExplore to build FCA structures. The second program, *ECFDgroups*, takes an RCFT file as input and computes the feature-groups and mutex of the ECFD, following the mapping of Section 4. We applied this process on 10 different feature models from SPLOT, which are presented on Table 6. The selected FMs possess from 10 features to 36 features, and from 8 valid configurations (small SPL) to 8480 valid configurations (large SPL). For each feature model, we give the number of features, configurations, mandatory relationships, optional relationships, xor-groups, or-groups and cross-tree constraints in the first part of the table. We then compute the ECFD and indicate the number of obtained xor-groups, or-groups, and mutex. For this experiment, we compute only groups and mutex, as the complexity to compute implications and co-occurrences is trivial in comparison. To characterise the extracted ECFD, we display the number of overlapping groups, cases where variability blocks possess more than one parent (`# multi-parents`), and composite variability blocks (i.e., having more than one feature). We retrieve the number of composite variability blocks and their size: (1:4) in the table states that there is 1 composite block of size 4. We have seen that the choices given by an ECFD are reduced 1) to choose a parent for each feature when necessary and 2) choose the feature groups to retain. In the worst case, the number of choices is thus:

$$\#\text{ features} + \#\text{ multi-parents} + \#\text{ groups}$$

To be compared to the number of choices, we give a lower bound of the number of possible FMs that may be derived from the ECFD. Composites variability blocks of size $n$ may have $n(n-1)$ possible combinations of mandatory relationships, and $\frac{n(n-1)}{2}$ possible cases where two features are optional but linked by a double requires relationship. Number of different combinations of retained feature groups is equal to $2^m$, $m$ being the number of extracted groups. We thus consider the following formula to compute a lower bound of number of possible FMs:

$$\prod_{b \in VB_E} \left(\frac{3|b|(|b|-1)}{2}\right) \times 2^{|xor|+|or|}$$

It is noteworthy that all ECFDs have been computed despite the complexity of feature-groups detection algorithm and the size of large SPLs, which can reach around 9000 configurations. Our results show that the number of groups may vary between the FM and its ECFD. For example, one *xor*-group of the ECFD may combine several *xor*-groups of the FM when there are additional constraints (it is the case for *Online-book-shopping*). *Smart Home* is another interesting case: we extract the same number of or-groups than in the initial FM, but we detect that two of them overlap in the ECFD. Therefore, the user can only select at most 4 of them. It is possible that cross-tree constraints involving feature belonging to feature-groups "break" the logical semantics of the feature-groups, and therefore these groups are not detected in the ECFD. This is left as future work. 3 of the 10 studied FMs have one case of multi-parents, they seem to be occasional. Overlapping groups are more common but not too numerous. On average, 30%

[5]Last accessed on January 2018

Table 6: Characteristics of ECFDs for FMs from SPLOT

| Feature Model | # features | # opt. rel. | # mand. rel. | # or-groups | # xor-groups | # CTCs | # configurations | # or (ECFD) | # xor (ECFD) | # mutex (ECFD) | # overlapping groups | # multi-parents | # composite blocks | # ECFD choices | lower bound # FMs |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Tang Eshop | 10 | 2 | 3 | 1 | 1 | 2 | 13 | 2 | 1 | 1 | 2 | 1 | (1:4) | 14 | 144 |
| Martini Eshop | 11 | 1 | 5 | 1 | 1 | 1 | 8 | 2 | 1 | 1 | 2 | 0 | (1:6) | 14 | 360 |
| Toacy Eshop | 12 | 1 | 3 | 2 | 1 | 0 | 48 | 2 | 1 | 0 | 0 | 0 | (1:4) | 15 | 144 |
| Mobile Games | 16 | 10 | 1 | 1 | 0 | 1 | 3645 | 1 | 0 | 0 | 0 | 0 | (1:2) | 17 | 6 |
| Web Game | 16 | 5 | 6 | 2 | 0 | 2 | 84 | 3 | 0 | 0 | 0 | 0 | (1:6)(1:2) | 19 | 1080 |
| Smart Home | 22 | 5 | 3 | 5 | 0 | 2 | 8480 | 5 | 0 | 0 | 2 | 1 | (1:2)(1:3) | 28 | 864 |
| Bicycle | 27 | 6 | 5 | 0 | 5 | 2 | 1152 | 1 | 5 | 14 | 0 | 0 | (1:6) | 33 | 2880 |
| Automotive system | 31 | 3 | 8 | 1 | 7 | 9 | 1344 | 9 | 7 | 9 | 8 | 1 | (1:7)(2:2) | 48 | 1.2E7 |
| Robot Calibration | 33 | 0 | 10 | 1 | 7 | 11 | 648 | 2 | 7 | 3 | 3 | 0 | (1:14) | 42 | 1.4E5 |
| Online-book-shopping | 36 | 2 | 21 | 0 | 5 | 3 | 90 | 1 | 4 | 1 | 0 | 0 | (1:18)(2:4) | 41 | 2.6E5 |

of the extracted groups overlap. Sometimes, a more consequent number of groups may overlap, for instance for *Automotive system*, from which 16 groups are extracted but 8 of them overlap. Heuristics to propose maximal subsets of groups that do not overlap for the user may be useful to guide him during its choices. The comparison between the number of choices proposed by an ECFD and the number of FMs that may be derived from it (two last columns of Table 6) highlights how much the use of ECFD allows to steer the user decisions. For instance, in the case of *Web Game*, the ECFD proposes at most 19 choices to the user, who then derive one FM amongst 1080. It is noteworthy that in all studied cases, the number of choices proposed by ECFDs are practicable and manageable by an end-user compared to the lower bound of the number of potential FMs, which is in some cases impossible to be handled by a human being.

## 7. Related Work

It is usually possible to distinguish two sub-processes in FM extraction approaches from variant descriptions: the first one consists in extracting variability information, i.e., relationships between features and the second one seeks to synthesise an FM from these extracted relationships. In the literature, some authors start the extraction process from variant descriptions (i.e., on the form of collections of feature sets), and others work directly from feature relationships (i.e., on the form of propositional formulas). Here, we focus on the approaches starting from variant descriptions, but first we introduce two methods of variability extraction from feature relationships, as they are widely reused in the other approaches.

### *7.1. Extracting variability from feature relationships*

In [4], Czarnecki and Wasowski consider feature relationships in the form of a propositional formula, and they propose a fully automated method to synthesise an FM from the formula. The extracted FM corresponds to a "generalised notation" where groups may overlap and where the feature tree may be a directed acyclic graph (DAG), thus being different from the one introduced in the FODA report. They first compute an implication graph from the propositional formula to extract a feature hierarchy in the form of a DAG, then compute feature groups and combine them with the previous DAG to obtain a FM. Their method is based on Binary Decision Diagrams (BDD) to perform operations on propositional formulas. Their method successfully identifies all binary implications and co-occurrences that are true in the considered propositional formula: it is sound and complete. Their feature group extraction relies on the computation of prime implicants, that allow to extract all minimal or-groups. All xor-groups are then identified

among the extracted or-groups by checking if the features are mutually exclusive. Because they extract all minimal or-groups, and that they construct aside an implication graph, non-minimal or-groups may be derived by combining these two structures. Even though our approach produces the same results, FCA embodies this information in a unique knowledge representation, including mutually exclusive features, which are not considered in this work. Their method is deterministic because they rely on a generalised notation of FM, which avoids user intervention for group and hierarchy choices. Their generalised notation is thus similar to our ECFD, except that we extract all mutex relationships, and that our method is able to derive a final FM respecting the revisited FODA's notation in [4].

She et al. [42] revisit the BDD-based method of [4], and complete it to extract mutex. In addition to the binary implication graph, they compute a mutex graph, i.e., a graph where nodes represent features and edges represent mutual exclusion between features. The mutex graph documents all existing mutex. They also use this graph to detect xor-groups, which are or-groups where the features form a clique in the mutex graph. Thus, they extract all logical relationships of the logical semantics of a FM. All their extracting methods are sound and complete. They represent the extracted logical semantics in the form of a feature graph, which is a symbolic representation of all FMs compatible with the initial feature relationships. The ECFD may be seen as a full graphical representation of a feature graph. Also, they design two algorithms: one to extract feature graphs from Conjunctive Normal Form, and the other from Disjunctive Normal Form. The derivation of feature hierarchies from a feature graph is not studied in their paper. In comparison, our approach relies on a mathematical framework for knowledge representation, which provides a unique structure encompassing the ones used here by She et al. Moreover, FCA structures and their correlations with FM variability delimit the possible derivable FMs and permit to guide the user during this task.

In [26], She et al. extract FMs from both feature relationships and feature descriptions. Their approach is based on the work of Czarnecki and Wasowski [4], but the construction of the feature hierarchy and the allocation of feature groups are not fully automated and depend on user decisions. More specifically, the extraction of the FM logical semantics relies on the propositional formula representing feature relationships, and they use feature descriptions to approximate the ontological semantics and assist the user into his decisions. In fact, they propose heuristics to identify best candidates to be the parent feature of a given feature by relying on their descriptions, and thus assist users in the construction of the feature hierarchy. First, all the candidate features are retrieved based on the propositional formula by computing the binary implication graph. Then, for each feature, the heuristics rank the other features depending on their similarity with the given one, thanks to a similarity function which takes into account the words in feature descriptions. Xor-groups are then automatically computed by using a mutex graph, but once again it is the user who chooses which groups are retained in the final model. This time, no heuristics is defined: all the xor-groups are presented to the user who chooses which one to keep in the hierarchy. Or-groups and co-occurrences are not studied in this paper. Partially automated strategies and Natural Language Processing techniques permit here to obtain more consistent FMs than with a fully automated method. Our work is similar to theirs, because we also propose a method relying on user decisions to build the final FM hierarchy. In comparison, we did not define any heuristics to guide the user choice as they do, because we do not consider feature descriptions in our input dataset. But our method may greatly benefit from the usage of their method to further assist the user into building the hierarchy. However, we rely on a unique AC-poset to extract FM logical semantics: our method covers all the necessary logical relationships and extract them in a sound and complete manner. Also, they propose to add a propositional formula $\phi$ in the cross tree constraints to make the final FM sound: we may be inspired by this aspect to make the configuration semantics of the ECFD and the derived FM sound. As FCA conceptual structures capture the logical semantics, we may be able to derive a formula $\phi$ representing the missing constraints to make the ECFD sound: this is left as future work.

### *7.2. Extracting variability from variant descriptions*

Acher et al. [11] work on extracting FMs from matrices representing variant descriptions. They construct the feature hierarchy by producing one FM for each variant, and then merging all these FMs into one. Each FM is built depending on variant descriptions and a collection of specifications defined in a dedicated language by practitioners. This parametrisation seeks to breath some ontological semantics into the feature hierarchy. Then, they compute the propositional formula representing the set of variant descriptions and use the BDD-based method from [4]. Their method then combines the hierarchy obtained after merging and the extracted logical relationships to obtain an FM. Contrarily to the work of [4], their final FMs may have mutex, although they do not specify how they compute them. In [28], the authors assess the necessity for a FM to have a meaningful hierarchy, feature groups and constraints. Because several different FMs can be extracted from the same set of configurations, they propose to configure the procedure of

extraction with a set of expected properties for the output FM. The extraction method combines their previous work [11] with the expected properties, which allows to make decisions during the synthesis. These two papers consider user choices in the FM extraction, but contrary to our method based on interactions during the synthesis process, these choices are made before the extraction through a dedicated language. The importance of FM meaning has been more recently discussed by Becan et al. in [15]. They propose a hybrid method relying on both logical heuristics that are fully automated, and ontological heuristics that need user decisions. Logical heuristics extract co-occurring features, minimal feature groups and a representation of all possible feature trees. Then, ontological heuristics help the user into choosing a feature tree by ranking for each feature their potential parent-features, and to detect if a group of features are siblings. They show empirically that hybrid FM extractions relying on user intervention outperform the other approaches. As for the work of [26], our method does not propose any ranking method to help the user choose the hierarchy and the feature groups, and could benefit from being combined with the techniques exposed here. In [14], Haslinger et al. propose a recursive algorithm to build FMs from a collection of feature sets (i.e., configurations). This algorithm first constructs the basis of the FM by identifying a root feature and features which are common to all products. Then, the rest of the FM is generated recursively from top to bottom: for each feature in the current hierarchy, it computes its direct descendants, then identify xor, or and optional relationships among them and add them in the feature tree. Mutex are not considered in this paper. Evaluations conducted by the authors reveal that the obtained FMs represent the same configurations as the original collection of feature sets. It is debatable as 1) they do not compute cross-tree constraints, and 2) FMs are known to not be logically complete. Also, the FM meaning still needs to be assessed, as well as the soundness and completeness of the feature relationship identification. They extend this method in [22] to synthesise FMs with requires and exclude cross-tree constraints, but do not cover the cases of circular requires cross-tree constraints.

Davril et al. [16] present an automated approach to deal with FM generation when variant configurations are not formally documented. The first step of their approach consists in mining relevant features from a collection of informal documents found in public repositories: they extract feature descriptors as they appear in input documents and group them using clustering algorithms to identify a final set of features, from which they create a product-by-feature matrix. Then, their approach is similar to [26] in the sense that they base the FM extraction both on the product-by-feature matrix and feature descriptions. First, they mine frequent itemsets to extract a set of binary implications from this matrix and to construct a binary implication graph: it is then used as a support to extract a feature hierarchy. This extraction is neither sound nor complete as they use a threshold to parametrised the extraction. Co-occurrences are detected based on the obtained binary implication graph. They also mine disjunctive rules by computing prime implicants, in order to obtain or-groups. They do not compute mutex: they argue that their variant descriptions are incomplete, and that detecting mutually exclusive features in the matrix does not necessarily mean that the features are actually mutually exclusive with regard to the domain. Xor-groups are not computed either. They use text-mining techniques and co-occurrence between features to help identify a relevant hierarchy; it provides good results even though it does not reach the meaningfulness of a manually built feature hierarchy.

Some authors study search-based techniques to build FMs from product descriptions. In [43], Linsbauer et al. use genetic programming to synthesise FMs. First, a population of FMs is randomly generated based on the set of existing features. From this population, FMs are selected to go through a phase of crossovers and mutations from which results a set of new FMs. These FMs are then evaluated, the best candidates being the ones describing the set of configurations which are most similar to the initial set. The best candidates are added to the initial population and the procedure is repeated. Lopez-Herrejon et al. [12] propose two functions to evaluate three search-based techniques to synthesise FMs: evolutionary algorithm, hill climbing and random search. The FMs built with these methods may depict less configurations, or more configurations than the original set. Moreover, the ontological semantics is not studied in these methods. Contrary to search-based techniques, FCA-based methods are deterministic and provide the user with a structure describing how the ECFD have been constructed and why.

Formal concept analysis is first used for reverse engineering FMs by Ryssel et al. in [10]. From a formal context representing the set of configurations, they build a graph with attribute-concepts as nodes (i.e., an AC-poset), from which they extract a feature hierarchy. As the AC-poset is not necessarily a tree, they remove some features considered redundant until they obtain a hierarchy where all edges represent optional relationships. Thus, the final FM may not possess all features displayed in the descriptions. In our approach, we keep all existing features, and rely on user decision to build a correct hierarchy. Feature-groups and other relationships are then derived from the extent of attribute-concepts. The authors compute additional textual constraints, in the form of complex implications, to restrain

valid configurations depicted by the FMs and obtain a set of valid configurations equivalent to the one depicted in the formal context. Extraction methods of all relationships are sound and complete, but they do not compute co-occurrences. In [13], Al-Msie'deen et al. propose algorithms to extract different kinds of feature groups from an AOC-poset associated with a formal context displaying product descriptions. However, the extracted FMs only possess two levels of hierarchy and exclusively depict logical relationships. Morever, their feature-group extraction relies on heuristics and is not sound nor complete.

In comparison with what can be found in the literature, our method is based on a unique structure embodying all relationships depicting FM logical semantics, which supports a sound and complete extraction of these relationships. Therefore, it encompasses most of the extractive methods presented here, along with their intermediate structures (e.g., implication graph, mutex graph, feature diagram) used for variability analysis. FCA is a structural framework which, contrary to most of the existing approaches for FM synthesis, supports this process without being only designed for this specific task. Therefore, it is a reusable and extensible framework supporting numerous knowledge processing functions [29], which can help in the migration towards SPLs, aside from FM synthesis, and consistently with it. This unique knowledge representation structure, along with its mapping with FMs relationships, guides in few steps a user into choosing a feature hierarchy without relying on external information as feature descriptions.

Table 7: Synthesis of existing methods for FM extraction

| Ref. | **Input** | **Synthesis** | **Conf. sem.** | **Onto. sem.** | **Bin. Impl.** | **Co-occ.** | **Mutex** | **Or-groups** | **Xor-groups** |
|---|---|---|---|---|---|---|---|---|---|
| Cza-07 [4] | Rel. | Auto. | More or equal | Not assessed | s+c | s+c | / | s+c | s+c |
| She-14 [42] | Rel. | Auto. | More or equal | Not assessed | s+c | s+c | s+c | s+c | s+c |
| She-11 [26] | Rel. | Semi auto. | Equal | Assessed | s+c | / | s+c | / | s+c |
| Ach-12 [11] | Desc. | Semi auto. | More or equal | Assessed | s+c | s+c | ? | s+c | s+c |
| Ach-13 [28] | Desc. | Semi auto. | More or equal | Assessed | s+c | s+c | ? | s+c | s+c |
| Bec-16 [15] | Desc. | Semi auto. | More or equal | Assessed | s+c | s+c | ? | s+c | s+c |
| Has-11 [14] | Desc. | Auto. | More or equal | Not assessed | ? | ? | / | ? | ? |
| Has-13 [22] | Desc. | Auto. | More or equal | Not assessed | ? | ? | ? | ? | ? |
| Dav-13 [16] | Desc. | Auto. | ? | Not assessed | ∅ | ∅ | / | s | / |
| Lin-14 [43] | Desc. | Auto. | ? | Not assessed | ? | ? | ? | ? | ? |
| Lop-15 [12] | Desc. | Auto. | More or less | Not assessed | ? | ? | ? | ? | ? |
| Rys-11 [10] | Desc. | Auto. | Equal | Not assessed | s+c | / | s+c | s+c | s+c |
| AlM-14 [13] | Desc. | Auto. | More | Not assessed | s+c | s+c | s | ∅ | ∅ |
| *Proposed* | *Desc.* | *Auto.* | *More or equal* | *Assessed* | *s+c* | *s+c* | *s+c* | *s+c* | *s+c* |

Table 7 gathers some information relative to all the presented methods. Column *Input* states if the method uses variant descriptions as input (*Desc.*), or feature relationships (*Rel.*). Column *Synthesis* specifies if the method is fully automated (*Auto.*) or semi-automated (*Semi.*). Then, column *Conf. sem.* determines if the method produces FMs depicting more, less or same valid configurations than the input set. Column *Onto. sem.* tells if the consistency of the FM is assessed or not. The five last columns represent the five relationships of the FMs logical semantics. The letter "s" indicates that the extracting method is sound, and the letter "c" that it is complete. When the method is neither sound nor complete, the cell contains the symbol "∅". A "/" is used when the relationship is not studied in the corresponding paper. A "?" is used when the relationship is studied, but not its soundness and completeness.

## 8. Conclusion and future work

In this paper, we studied FCA as a structural and reusable framework for variability extraction and representation, in the context of migration from single system development towards software product line approaches. We studied

the different semantics of boolean features models and three FCA conceptual structures, as well as some of their properties. More specifically, we focused on the matching between feature dependencies expressed in feature models and the ones that can be extracted from FCA conceptual structures. We showed that conceptual structures include the logical semantics of their equivalent FMs, and therefore represent equivalence classes of FMs describing the same set of valid configurations. To ease the representation of the useful logical information extracted from conceptual structures to represent these equivalence classes, we introduce a diagrammatic representation of this information that we called an equivalence class feature diagram (ECFD). Also, we established a mapping between features relationships found in the two models, that permits to match FM relationships in the conceptual structures built on top of the FM configurations. We then proposed a process based on ECFDs and the established mapping to assist feature model extraction from product descriptions, while preserving the configuration semantics and allowing domain experts to assign step-by-step the chosen ontological semantics to the feature model. We argued that most existing approaches for FM reverse engineering are encompassed in FCA, which thus represents an unifying framework that lays down theoretical foundations for FM reverse engineering. We assessed the applicability of our method depending on the size of the generated structures and the complexity of the applied operations. Conceptual structures generated from product descriptions are easily computed with existing algorithms, and their size is practicable, especially for AOC-posets. Among operations applied on these structures to extract ECFDs, computing feature-groups is the only one with an exponential complexity with the size of the problem; however, it scales well in practice, even with large collections of product descriptions. We implement ECFD extraction to compare their characteristics against the corresponding feature models, notably concerning the feature-groups as their number may vary between the two representations. Indeed, we found out that combinations of several feature relationships in a feature model (e.g., *requires* constraint between features involved in a *xor*-group) may "alter" the extracted logical relationships, and therefore some feature-groups are not detected in the ECFD. Also, we computed the number of choices given by ECFDs and compared them to the number of different FMs that may be derived from the same ECFDs: these experiments showed that in practice, ECFDs are strong guides for FM reverse engineering.

In the future, we plan to study the possible combinations of feature relationships to be able to detect and extract all feature-groups from product descriptions and conceptual structures. We would like to complete the mapping to take into account these singularities and therefore improve our method. Also, combining our approach with feature descriptions and natural language processing techniques to provide metrics and rankings to help the user during his decisions could strongly improve the FM derivation process from an ECFD. Deriving a propositional formula $\phi$ from a conceptual structure in order to characterise constraints not supported by FMs and ECFDs is also considered, as it may be useful in some applications. A full implementation of the proposed approach to conduct qualitative evaluations is also considered. Finally, we would like to expand this work to take into account feature model extensions, as UML-like cardinalities, multi-valued attributes or references between several FMs. We plan to study some FCA extensions that may be useful to extract logical relationships from complex data (i.e., not only binary features).

## References


[1] K. Pohl, G. Böckle, F. J. van der Linden, Software Product Line Engineering: Foundations, Principles, and Techniques, Springer Science & Business Media, 2005.

[2] K. Schmid, R. Rabiser, P. Grünbacher, A comparison of decision modeling approaches in product lines, in: Proceedings of the 5th Workshop on Variability Modelling of Software-Intensive Systems (VaMoS'11), ACM, 2011, pp. 119–126.

[3] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, A. S. Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study, Tech. rep., Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst (1990).

[4] K. Czarnecki, A. Wasowski, Feature diagrams and logics: There and back again, in: Proceedings of the 11th International Conference on Software Product Lines (SPLC'07), 2007, pp. 23–34.

[5] F. Loesch, E. Ploedereder, Restructuring Variability in Software Product Lines using Concept Analysis of Product Configurations, in: Proceedings of the 11th European Conference on Software Maintenance and Reengineering (CSMR'07), 2007, pp. 159–170.

[6] A. Jansen, R. Smedinga, J. van Gurp, J. Bosch, First class feature abstractions for product derivation, IEE Proceedings - Software 151 (4) (2004) 187–198.

[7] T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, A. Wasowski, A survey of variability modeling in industrial practice, in: Proc. of the 7th Int. Workshop on Variability Modelling of Software-intensive Systems (VaMoS'13), 2013, pp. 7:1–7:8.

[8] Y. Dubinsky, J. Rubin, T. Berger, S. Duszynski, M. Becker, K. Czarnecki, An exploratory study of cloning in industrial software product lines, in: 17th European Conference on Software Maintenance and Reengineering, CSMR 2013, Genova, Italy, March 5-8, 2013, 2013, pp. 25–34.

[9] C. W. Krueger, Easing the transition to software mass customization, in: Proceedings of the 4th International Workshop on Software Product-Family Engineering (PFE'01), 2001, pp. 282–293.

[10] U. Ryssel, J. Ploennigs, K. Kabitzsch, Extraction of feature models from formal contexts, in: Workshop Proceedings of the 15th International Conference on Software Product Lines (SPLC'11), 2011, p. 4.
[11] M. Acher, A. Cleve, G. Perrouin, P. Heymans, C. Vanbeneden, P. Collet, P. Lahire, On extracting feature models from product descriptions, in: Proceedings of the 6th International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS'12), 2012, pp. 45–54.
[12] R. E. Lopez-Herrejon, L. Linsbauer, J. A. Galindo, J. A. Parejo, D. Benavides, S. Segura, A. Egyed, An assessment of search-based techniques for reverse engineering feature models, Journal of Systems and Software 103 (2015) 353–369.
[13] R. Al-Msie'deen, M. Huchard, A. Seriai, C. Urtado, S. Vauttier, Reverse Engineering Feature Models from Software Configurations using Formal Concept Analysis, in: Proceedings of the 11th International Conference on Concept Lattices and Their Applications (CLA'14), 2014, pp. 95–106.
[14] E. N. Haslinger, R. E. Lopez-Herrejon, A. Egyed, Reverse Engineering Feature Models from Programs' Feature Sets, in: Proceedings of the 18th Working Conference on Reverse Engineering (WCRE'11), 2011, pp. 308–312.
[15] G. Bécan, M. Acher, B. Baudry, S. B. Nasr, Breathing ontological knowledge into feature model synthesis: an empirical study, Empirical Software Engineering 21 (4) (2016) 1794–1841.
[16] J. Davril, E. Delfosse, N. Hariri, M. Acher, J. Cleland-Huang, P. Heymans, Feature model extraction from large collections of informal product descriptions, in: Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, (ESEC/FSE'13), 2013, pp. 290–300.
[17] B. Ganter, R. Wille, Formal concept analysis - mathematical foundations, Springer, 1999.
[18] N. Niu, S. M. Easterbrook, Concept analysis for product line requirements, in: Proceedings of the 8th International Conference on Aspect-Oriented Software Development (AOSD'09), 2009, pp. 137–148.
[19] Y. Xue, Z. Xing, S. Jarzabek, Feature location in a collection of product variants, in: Proceedings of the 19th Working Conference on Reverse Engineering (WCRE'12), 2012, pp. 145–154.
[20] R. Al-Msie'deen, A. Seriai, M. Huchard, C. Urtado, S. Vauttier, H. E. Salman, Mining Features from the Object-Oriented Source Code of a Collection of Software Variants Using Formal Concept Analysis and Latent Semantic Indexing, in: Proceedings of the 25th International Conference on Software Engineering and Knowledge Engineering (SEKE'13), 2013, pp. 244–249.
[21] A. Shatnawi, A. Seriai, H. A. Sahraoui, Recovering software product line architecture of a family of object-oriented product variants, Journal of Systems and Software 131 (2017) 325–346.
[22] E. N. Haslinger, R. E. Lopez-Herrejon, A. Egyed, On Extracting Feature Models from Sets of Valid Feature Combinations, in: Proceedings of the 16th International Conference on Fundamental Approaches to Software Engineering (FASE'13), 2013, pp. 53–67.
[23] K. Czarnecki, S. Helsen, U. W. Eisenecker, Staged Configuration Using Feature Models, in: Proceedings of the 3rd International Conference on Software Product Lines (SPLC'04), 2004, pp. 266–283.
[24] K. Czarnecki, S. Helsen, U. W. Eisenecker, Formalizing cardinality-based feature models and their specialization, Software Process: Improvement and Practice 10 (1) (2005) 7–29.
[25] W. Petersen, A Set-Theoretical Approach for the Induction of Inheritance Hierarchies, Electronic Notes in Theoretical Computer Science 53 (2001) 296–308.
[26] S. She, R. Lotufo, T. Berger, A. Wasowski, K. Czarnecki, Reverse engineering feature models, in: Proceedings of the 33rd International Conference on Software Engineering, (ICSE'11), 2011, pp. 461–470.
[27] P. Schobbens, P. Heymans, J. Trigaux, Y. Bontemps, Generic semantics of feature diagrams, Computer Networks 51 (2) (2007) 456–479.
[28] M. Acher, B. Baudry, P. Heymans, A. Cleve, J. Hainaut, Support for reverse engineering and maintaining feature models, in: Proceedings of the 7th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'13), 2013, pp. 20:1–20:8.
[29] U. Priss, Formal concept analysis in information science, ARIST 40 (1) (2006) 521–543.
[30] R. Al-Msie'Deen, M. Huchard, A.-D. Seriai, C. Urtado, S. Vauttier, A. Al-Khlifat, Concept lattices: a representation space to structure software variability, in: Proceedings of the 5th International Conference on Information and Communication Systems (ICICS'14), IEEE, 2014, pp. 1–6.
[31] M. Mannion, Using First-Order Logic for Product Line Model Validation, in: Proceedings of the 2nd International Conference on Software Product Lines (SPLC'02), 2002, pp. 176–187.
[32] D. S. Batory, Feature models, grammars, and propositional formulas, in: Proceedings of the 9th International Conference on Software Product Lines (SPLC'05), 2005, pp. 7–20.
[33] C. Wende, K. Siegemund, E. Thomas, Y. Zhao, J. Z. Pan, F. S. Parreiras, T. Walter, K. Miksa, P. Sabina, U. Aßmann, Ontology reasoning for consistency-preserving structural modelling, in: J. Z. Pan, S. Staab, U. Aßmann, J. Ebert, Y. Zhao (Eds.), Ontology-Driven Software Development, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, Ch. 9, pp. 193–218.
[34] J. Carbonnel, K. Bertet, M. Huchard, C. Nebut, FCA for software product lines representation: Mixing product and characteristic relationships in a unique canonical representation, in: Proceedings of the Thirteenth International Conference on Concept Lattices and Their Applications, Moscow, Russia, July 18-22, 2016., 2016, pp. 109–122.
[35] S. O. Kuznetsov, T. P. Makhalova, On interestingness measures of formal concepts, Inf. Sci. 442-443 (2018) 202–219.
[36] M. Mendonca, M. Branco, D. Cowan, S.P.L.O.T.: Software Product Lines Online Tools, in: Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications (OOPSLA '09), 2009, pp. 761–762.
[37] N. Sannier, M. Acher, B. Baudry, From comparison matrix to Variability Model: The Wikipedia case study, in: Proceedings of the 28th International Conference on Automated Software Engineering (ASE'13), 2013, pp. 580–585.
[38] S. B. Nasr, G. Bécan, M. Acher, J. B. F. Filho, N. Sannier, B. Baudry, J. Davril, Automated extraction of product comparison matrices from informal product descriptions, Journal of Systems and Software 124 (2017) 82–103.
[39] S. O. Kuznetsov, S. A. Obiedkov, Comparing performance of algorithms for generating concept lattices, Journal of Experimental & Theoretical Artificial Intelligence 14 (2-3) (2002) 189–216.
[40] A. Berry, A. Gutierrez, M. Huchard, A. Napoli, A. Sigayret, Hermes: a simple and efficient algorithm for building the AOC-poset of a binary relation, Annals of Mathematics and Artificial Intelligence 72 (1-2) (2014) 45–71.
[41] M. Acher, P. Collet, P. Lahire, R. B. France, Familiar: A domain-specific language for large scale management of feature models, Science of

Computer Programming 78 (6) (2013) 657–681.
[42] S. She, U. Ryssel, N. Andersen, A. Wasowski, K. Czarnecki, Efficient synthesis of feature models, Information & Software Technology 56 (9) (2014) 1122–1143.
[43] L. Linsbauer, R. E. Lopez-Herrejon, A. Egyed, Feature model synthesis with genetic programming, in: Proceedings of the 6th International Symposium on Search-Based Software Engineering (SSBSE'14), 2014, pp. 153–167.