Towards the Extraction of Variability Information to Assist Variability Modelling of Complex Product Lines

Jessie Carbonnel, Marianne Huchard, Clémentine Nebut

Towards the Extraction of Variability Information to Assist Variability Modelling of Complex Product Lines

Jessie Carbonnel, Marianne Huchard, Clémentine Nebut
LIRMM, University of Montpellier, CNRS, Montpellier, France

https://dl.acm.org/citation.cfm?id=3168365.3168378

Abstract

Software product line engineering gathers a set of methods that rely on systematic reuse and mass customisation to reduce the development time and cost of a set of similar software systems. Boolean feature models are the *de facto* standard used to represent product line variability in terms of features, a feature being a distinguishable characteristic of one or several softwares. The extractive adoption of a product line from a set of individually developed softwares requires to extract variability information from a collection of software descriptions to model their variability. With the appearance of more and more complex software systems, software product line engineering faces new challenges including variability extraction and modelling. Extensions of boolean feature models, as multi-valued attributes or UML-like cardinalities have since been proposed to support variability modelling in complex product lines. In this paper, we propose research directions to address the issue of extracting more complex variability information, as a part of extended feature models synthesis from software descriptions. We consider the capabilities of Formal Concept Analysis, a mathematical framework for knowledge discovery, along with two of its extensions called Pattern Structures and Relational Concept Analysis, to answer this problematic. These frameworks bring theoretical foundations to complex variability extraction algorithms.

1 Introduction

Software Product Line Engineering (SPLE) [PBvdL05] is an approach based on systematic reuse and mass customisation that aims at reducing the development time and cost of a set of similar software systems. In this process, variability modelling is a central task that documents common and variable assets, along with the way they can be combined to constitute a valid software system. Representing assets by features, a feature being a distinguishable characteristic of one or several softwares, is the most commonly used variability modelling approach,

where feature models (FMs) [KCH+90] are the *de facto* standard to model variability. FMs organise a set of features in a hierarchy representing several levels of details, and express constraints between these features to depict their possible interactions in a software system.

However, software systems tend to become more and more complex [FKL+06]. This complexity may arise from the size of the software systems, or from the types of information that have to be represented to model them. This affects traditional software product line approaches, and gives new challenges in the domain of *complex software product lines* [HGR12]. In order to tackle the scale-related complexity of software systems, propositions and approaches to manage *multi product lines* (MPLs) are flourishing [HGR12, Bot13, Bos09, RSKuR08]. MPLs consist into dealing with several smaller and interconnected software product lines that compose a bigger and more complex one, in order to ease its overall management. Besides, FM extensions have been proposed these past years to improve the expressiveness of original FMs (also called boolean FMs) and cope with more complex product lines. The prevalent FM extensions are the three ones one can find in the cardinality-based feature model notation [CBUE02, CHE04], i.e., multi-valued attributes, UML-like cardinalities and references between different feature models. When multi-valued attributes and UML-like cardinalities allow to enrich the FMs capabilities to model more complex and detailed variability information, references allow to connect several different FMs to support MPLs.

Automated synthesis of boolean FMs from software system descriptions have been widely studied in the literature [ACP+12, HLE11, HLE13, LLG+15, RPK11, CW07, AMdSH+14, DDH+13, LLE14]. A significant number of these approaches rely on a phase of variability information extraction from the given descriptions, as a part of the FM synthesis. However, to the best of our knowledge, only Becan et al. [BBGA15] have studied the automated synthesis of extended FMs, or the extraction of more complex variability information to help manage complex software product lines.

Formal Concept Analysis (FCA) [GW99] is a mathematical framework widely used in knowledge discovery [PEVD10] and in software reverse engineering applications [Sne00]. From a set of objects described by a set of binary attributes, the application of FCA organises the objects depending on the attributes they share in a canonical structure that highlights their commonalities and variability. This framework has been used to extract variability information found in boolean feature models, when the objects represent software variants and when the attributes represent features [AMdSH+14, RPK11, LP07, CHN17]. Besides, FCA possesses several extensions, as for instance to take into account temporal data, similarities between data, or relationships between several different datasets.

In this paper, we propose research directions to address the problem of extracting complex variability information from software descriptions, as a part of the process of extended feature model synthesis. In what follows, we call *augmented variability information* the information that involves multi-valued attributes and UML-like cardinalities, i.e., not only boolean features. *Inter-*

model variability information is the name we give to variability information involving elements from different feature models. We assess the usage of FCA, along with two of its extensions called Pattern Structures and Relational Concept Analysis, as mathematical frameworks to extract complex (i.e., augmented and/or inter-model) variability information in the form of logical relationships. The theoretical foundations of variability extraction algorithms provided by these frameworks is what motivated our choice. The aim of this research is to ease the transition from individually developed software variants that are more and more complex, to systematic software reuse and mass customisation approaches. The remainder of this paper is organised as follows. Boolean feature models and their extensions for complex variability modelling are presented in Section 2, where we identify the kinds of logical relationships they represent. In Section 3, we present the theoretical bases of Formal Concept Analysis, Pattern Structures and Relational Concept Analysis. We expose our research plan in Section 4. Related work is discussed in Section 5, and Section 6 concludes the paper.

2 Feature Models and their Extensions

In this section, we present boolean feature models along with their three prevalent extensions that have been proposed to document and manage more complex variability (i.e., within the same FM and between different FMs). We also identify the logical semantics of each one of the variability information given by these variability models, in the form of logical relationhips.

2.1 Boolean Feature Models

Boolean Feature Models (FMs) [KCH$^+$90] are a family of graphical languages which allow to define the scope of a product line in terms of features (i.e., distinguishable characteristics) and constraints between these features. Figure 1 presents a boolean FM about e-commerce applications. A boolean FM represents a finite set of features in a hierarchy (called *feature tree*), that expresses child-parent relationships. Constraints can be represented graphically by decorating the edges of the feature tree: these constraints show how the selection of a feature may affect the selection of its child features. A black disc forces 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 selected optionally (*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, additional constraints that cannot be expressed in the feature tree can be added. They are typically *requires* and *exclude* constraints. The boolean FM of Figure 1 states that: all e-commerce applications have a catalog, that is either displayed in a

list or in a grid; an application can optionally support payment methods, credit card and check being the two possible proposed methods; it can also possess a basket management; having a basket necessitates to support payment methods, and conversely.

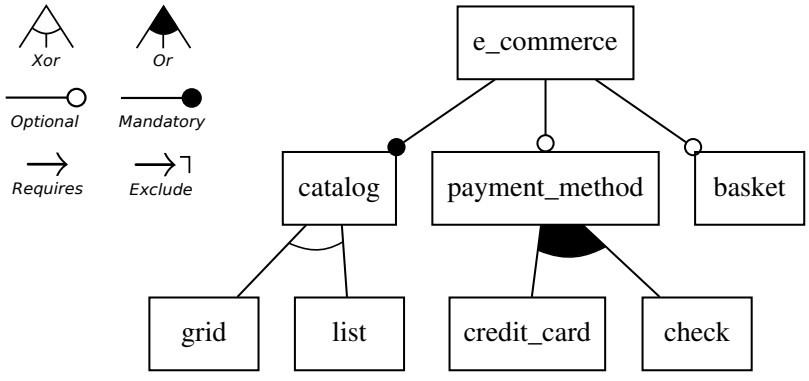

Figure 1: Boolean FM about e-commerce applications

FMs give some knowledge about the modelled domain and the interaction of some of its concepts. Extracting FM relationships that convey correct ontological knowlegde from a set of product descriptions without using external ontologies or relying on an expert intervention is quite infeasible. Besides, the parallel between boolean FMs and propositional logic has been widely studied [Man02, CW07, BSRC10]; writing boolean FMs in the form of propositional formulas allows to represent the *logical semantics* of the FM constraints and to display the feature interaction through logical relationships. Extracting logical relationships from a set of product descriptions is easier and less error-prone than extracting ontological FM relationships. Table 1 shows the logical semantics of the FM as presented in [CW07, PSA+12].

Table 1: Logical semantics of a boolean FM constraints; p represents a parent feature, c_i a child feature, and f_i any feature

FM constraints	**Logical semantics**
child-parent	$c \rightarrow p$
optional	none
mandatory	$p \rightarrow c$
or-group	$p \rightarrow \{c_1 \vee ... \vee c_n\}$
xor-group	$p \rightarrow \{c_1 \oplus ... \oplus c_n\}$
requires	$f_1 \rightarrow f_2$
exclude	$f_1 \rightarrow \neg f_2$

Therefore, we can identify in boolean FMs the following logical relationships

between features: implications ($f_1 \rightarrow f_2$, from child-parent, mandatory and requires relationships), mutex ($f_1 \rightarrow \neg f_2$, from exclude constraints), the particular case of double implications that we call co-occurrences ($f_1 \leftrightarrow f_2$, from double requires relationships), or-groups and xor-groups.

2.2 Extended Feature Models

In what follows, we will consider the two extended feature models of Figure 2. The left-hand side FM represents e-commerce applications as the one in Figure 1, but with some additional information. The feature `catalog` possesses an attribute `productsPerPage` of type integer that defines the maximum number of products depicted in a catalog page. The cardinality of the feature `catalog` states that an e-commerce has at least one catalog but can have several ones. The two group cardinalities constrain the number of features that can be selected in the corresponding group. This FM does not directly include a feature `basket`, but it is connected to another FM about `basket` management that can support a bank verification process. The constraint `credit_card` $\rightarrow$ `bank_verification` is a *requires* constraint involving a feature of each FM, and `grid` $\rightarrow$ `productsPerPage` $\geq$ `15` is a *requires* constraint involving a feature and an attribute value.

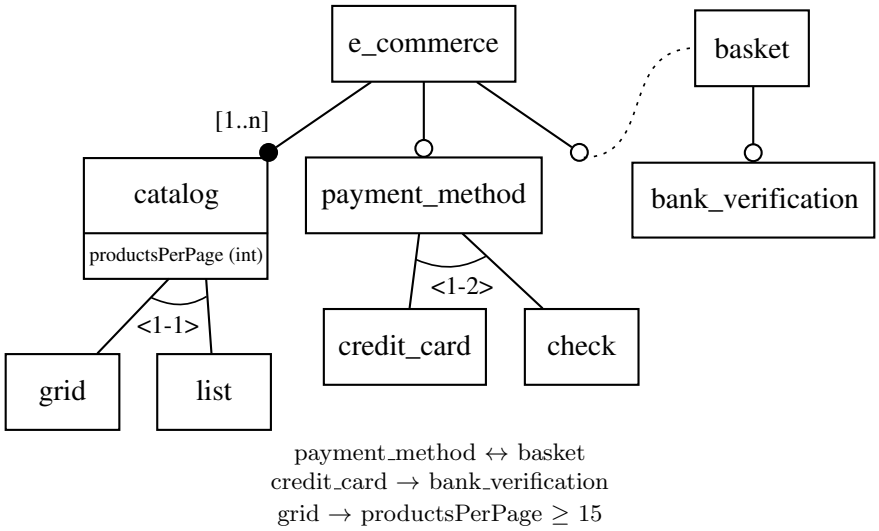

Figure 2: Extended feature models with a feature attribute, a feature cardinality, group cardinalities and a reference

2.2.1 Attributes

An extension of boolean FMs proposes to add multi-valued attributes. An attribute possesses a type (e.g., integer, string, enumeration) and is associated

with one feature of the FM. This extension permits to model more detailed information without complexifying the FM [CBUE02]. In fact, in an "all-feature view" of the FM, each attribute value would be represented as a feature. In the cases where the possible values are too numerous (as for numerical values), the number of features would be too important and the FM would be unintelligible. For instance, introducing an attribute `productsPerPage` of type integer in the feature `catalog` reduces the number of features necessary to represent this information, as shown in Figure 3.

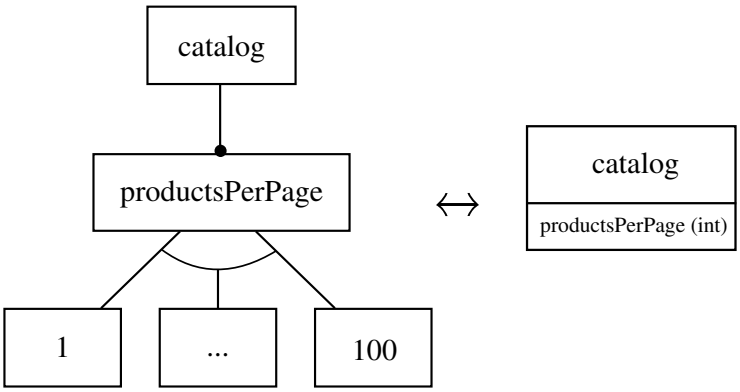

Figure 3: Representing numerous values with features (left-hand side) versus with an attribute (right-hand side)

Introducing attributes and their values in FMs allows to express complex variability information, i.e., *requires* and *exclude* constraints between features and/or attribute values. In our example of Figure 2, the constraint `grid` $\rightarrow$ `productsPerPage` $\geq$ `15` involves a feature and an attribute value. As stated by their name, feature-groups are only defined over the set of features, so they do not involve attributes. The variability information induced by attributes thus corresponds to the following logical relationships: implications, co-occurences and mutex between a feature and an attribute value, or between two attribute values. We call these additional logical relationships *augmented variability information*.

2.2.2 Cardinalities

Another extension of FMs introduces UML-like cardinalities on features and on feature-groups [CHE04].

Feature-group cardinalities depict the minimum and the maximum number of sub-features that can be selected in a group (denoted $< min - max >$). Therefore, the boolean feature model group notations for xor-groups and or-groups do not stand anymore: xor-groups are defined by a cardinality $< 1 - 1 >$, while or-groups by a cardinality $< 1 - n >$. This can be written in propositional logic by representing each combination of sub-features of the group

that is allowed by the cardinality. Let p be a feature, and $\{f_1, f_2, f_3\}$ be a feature-group with p as a parent and associated with the cardinality $< 2-3 >$. The logical relationship representing this group is:
$p \rightarrow ((f_1 \wedge f_2 \wedge \neg f_3) \vee (f_1 \wedge f_3 \wedge \neg f_2) \vee (f_2 \wedge f_3 \wedge \neg f_1) \vee (f_1 \wedge f_2 \wedge f_3))$

Feature cardinalities define the minimum and the maximum number of occurrences of a given feature in a valid configuration (denoted $[min..max]$). Except for the graphical notation that can be different, feature cardinalities can be seen as attributes: in the example of Figure 2, the cardinality of `catalog` could be represented by an attribute `occurrence` with the domain $[1..n]$. Hence, feature cardinalities could also be represented by features, as we have seen before with attribute values. Note that even though the representation is different, the information stays the same. Figure 4 depicts the different (yet equivalent) ways to represent a feature cardinality.

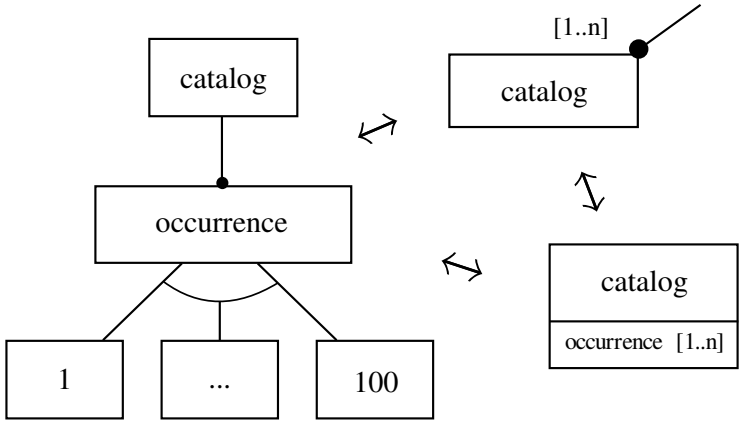

Figure 4: Different ways to represent a feature cardinality

Group-cardinalities allow to consider that all feature-groups are the same kind of variability information: not as xor-/or-groups, but as feature-groups associated with a cardinality. Thus, concerning feature groups, this FM extension does not add any new kind of variability information compared to the one found in boolean FMs, but generalises two existing ones. Moreover, as feature cardinalities can be seen as feature attributes, the kinds of variability information induced by feature cardinalities are the same as the ones induced by attributes, i.e., *augmented variability information.*

2.2.3 References

The last FM extension we consider allows the definition of *references*, which increase the FM modularity [CBUE02, CHE04]. A reference is a feature of an FM representing another FM, and is graphically represented by a connection between the feature and the other FM root feature. This extension permits to split a large FM into several more specific FMs which are connected to each

other (e.g., following separation of concerns), thus avoiding monolithic and large models which are difficult to work with. Also, this extension allows to reuse the sub-FMs in several places of the product line. Constraints as *requires* and *exclude* ones can then be defined between features of different yet connected FMs. In Figure 2, a reference (denoted by a dotted line) is defined between the root feature of the left-hand side FM about e-commerce applications, and the right-hand side FM about baskets. The cross-tree constraint `credit_card` $\rightarrow$ `bank_verification` between features of the two FMs represents what we called an *inter-model variability information*, i.e., a logical relationship between elements of separated FMs.

3 Formal Concept Analysis and its Extensions

In this section, we present the bases of Formal Concept Analysis theory and two of its extensions: Pattern Structures, that allow to extract logical relationships between more complex data types than binary attributes (e.g., multi-valued attributes, numerical values) and Relational Concept Analysis, which permits to extract relationships between elements of different datasets.

3.1 Formal Concept Analysis

Formal Concept Analysis (FCA) [GW99] is a mathematical framework for hierarchical clustering of a set of objects that are described by a set of binary attributes. As input, FCA takes a *formal context* $F = (O, A, J)$, where O is the set of objects, A is the set of binary attributes and $J \subseteq O \times A$ is a binary relationship stating "which objects possess which binary attributes". A formal context can be represented by a table $O \times A$: a cross in the cell (o, a) states that the object o possesses the binary attribute a. Table 2 presents an excerpt of a formal context where the objects represent variants of e-commerce applications, and the binary attributes represent 8 possible features characterising these softwares.

The application of FCA on a formal context extracts a set of *formal concepts*, where each formal concept represents "a maximal set of objects sharing a maximal set of binary attributes". A formal concept is thus a pair $C = (E, I)$ where $E = \{o \in O \mid \forall a \in I, (o, a) \in J\}$ is the concept's extent and $I = \{a \in A \mid \forall o \in E, (o, a) \in J\}$ is the concept's intent. The set of formal concepts C_F of a formal context F can be partially ordered by the set-inclusion order (denoted $\leq_s$) on the concepts' extents. Given two concepts $C_1 = (E_1, I_1)$ and $C_2 = (E_2, I_2)$, $C_1 \leq_s C_2$ if and only if $E_1 \subseteq E_2$ and $I_2 \subseteq I_1$. C_1 is called a sub-concept of C_2, and C_2 a super-concept of C_1. The set of all concepts C_F of a formal context, provided with the order $\leq_s$ form a lattice structure $(C_F, \leq_s)$ called a *concept lattice*. Figure 5 presents the concept lattice obtained from Table 2.

An arrow represents a relation from a concept to one of its super-concepts, i.e., the partial order. Here, the concepts are presented in an optimised way by

Table 2: Formal context with 8 objects representing e-commerce applications described by 8 binary attributes

	e_commerce	catalog	grid	list	payment_method	credit_card	check	basket
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

displaying each object and each binary attribute only once in the structure. An object (resp. a binary attribute) is introduced in the lowest (resp. greatest) concept of the concept lattice possessing it. Therefore, a concept inherits all the objects of its sub-concepts, and all the binary attributes of its super-concepts. For instance, `Concept_11` possesses the binary attributes `check`, `payment_method`, `basket`, `e_commerce` and `catalog`, and the objects `v3`, `v4`, `v7` and `v8`.

The way the objects (i.e., similar softwares) and the binary attributes (i.e., software characteristics) are organised in the concept lattice highlights information regarding their variability. More specifically, using the proper algorithms [CHN17], one can extract logical relationships involving the binary attributes, as implications, co-occurrences, mutual exclusions (mutex) and groups (in the sense of the feature groups one can find in boolean feature models). This extraction approach is complete as it allows to extract all the logical relationships (among the four types presented before) that are true for the considered set of objects.

3.2 Pattern Structures

Pattern Structures [GK01] have been proposed as a generalisation of FCA to describe a set of objects O with data types that are more complex than binary attributes. In this approach, each object is characterised by a *description* (or *pattern*) taken from a set of descriptions (denoted D) having the same type. A set of descriptions can be of any type of data on which one can establish *similarities*. The similarity of two descriptions d_1 and d_2 of D is given by a *similarity operator* (denoted $\sqcap$) that returns the most specific description of D which generalises both d_1 and d_2. For instance, in the software engineering domain, a set of descriptions could depict programming languages, and one can define the similarity of the two descriptions *Java* and *C++* by a third description

Java $\sqcap$ *C++* = *Object Oriented Language*. The similarity operator is associated to a subsumption relation $\sqsubseteq$ that allows to partially order the set of descriptions D. In our example, *Object Oriented Language* $\sqsubseteq$ *C++*. The set of objects O, the partially ordered description set $(D, \sqcap)$ and the mapping $\delta : O \rightarrow D$ that associates each object $o \in O$ with a description $d \in D$ form a Pattern Structure. Given a Pattern Structure $PS = (O, (D, \sqcap), \delta)$, one can extract a set of *pattern concepts*, where a pattern concept represents a maximal set of objects $O' \subseteq O$ decribed by the most specific description $d \in D$ characterising all the objects of O'. The set of all pattern concepts extracted from a Pattern Structure can be partially ordered by the relation $\leq_{ps}$ as follows: given two pattern concepts $C_1 = (O_1, d_1)$ and $C_2 = (O_2, d_2)$, $C_1 \leq_{ps} C_2$ if and only if $O_1 \subseteq O_2$ and $d_2 \sqsubseteq d_1$. The set of all pattern concepts of PS provided with the partial order $\leq_{ps}$ form a *pattern concept lattice*. In a pattern concept lattice, the set of objects is structured depending on their (potentially complex) descriptions and their similarities.

Descriptions can be of atomic types (e.g., dates, numerical values, String), but it is possible to combine several sets of descriptions (that could be of different types) in a *vector of descriptions*. The similarity between two vectors of descriptions can be obtained by computing the similarity of descriptions with the same rank in the vectors:

$$< d_{1i}, d_{2i}, ..., d_{ni} > \sqcap_{dv} < d_{1j}, d_{2j}, ..., d_{nj} >$$

$$=< d_{1i} \sqcap_1 d_{1j}, d_{2i} \sqcap_2 d_{2j}, ..., d_{ni} \sqcap_n d_{nj} >.$$

Therefore, in this framework, a set of vectors of descriptions can be handled in the same way as a set of descriptions of atomic type. As vectors of descriptions can gather several different types of descriptions, they are great candidates to represent complex software configurations. Note that the same kind of mining algorithms used with traditional concept lattices can be applied on pattern concept lattices; this opens the field for extraction of logical relationships involving data types that are more complex than binary attributes (e.g., multi-valued attributes).

3.3 Relational Concept Analysis

Relational Concept Analysis (RCA) [HHNV13] extends traditional FCA to take into account several formal contexts, and to define relations between the sets of objects of these formal contexts. As for traditional FCA, each formal context describes a set of objects depending on a set of binary attributes. In addition, RCA allows to link a set of objects with another set of objects by a relationship expressed through a *relational context*. A relational context is a 3-tuple $R = (O_1, O_2, r)$ where O_1 (called the *source* set) and O_2 (called the *target* set) are two sets of objects such that there are two formal contexts (O_1, A_1, J_1) and (O_2, A_2, J_2), and where $r \subseteq O_1 \times O_2$ is a binary relationship. For instance, a formal context could represent softwares described by binary attributes as `spam-prevention` or `open source`. Another formal context could represent

programming languages described by attributes representing their paradigms. Finally, a relational context could represent the relationship "`is-written-in`", associating the softwares (*source* context) with the programming languages they are written in (*target* context).

As input, RCA takes a *Relational Context Family* (RCF) of the form $(\mathcal{F}, \mathcal{R})$ such that $\mathcal{F}$ is a set of formal contexts $F_i = (O_i, A_i, J_i)$, $i \in \{1, 2, ..., n\}$, and $\mathcal{R}$ is a set of relational contexts $R_j = (O_k, O_l, r_j)$, $j \in \{1, 2, ..., m\}$, with $r_j \subseteq O_k \times O_l$, and O_k, O_l respectivelly being the sets of objects of the formal contexts $F_k \in \mathcal{F}$ and $F_l \in \mathcal{F}$. Given a formal context $F = (O, A, J)$, we define $rel(F)$ the set of relational contexts having O as source set. The application of RCA extends each formal context F depending on the relational contexts of $rel(F)$ to take into account the relations defined between the objects. More precisely, for each relational context $R_j = (O_k, O_l, r_j) \in \mathcal{R}$, RCA extends the formal context of O_k with *relational attributes* representing a relation based on r_j to a concept that can be extracted from the formal context of O_l. In our previous example, the formal context about softwares would be extended with relational attributes involving the concepts extracted from the programming language formal context.

The application of RCA generates a succession of contexts and concept lattices associated with the RCF $(\mathcal{F}, \mathcal{R})$. Relational attributes appear in the concept lattices as traditional binary attributes, but they can be considered as *references* towards concepts of other lattices. Therefore, the objects in these concept lattices are not only structured by their binary attributes, but also by relational attributes. In other words, these objects are still organised depending on their binary attributes, but also by the binary attributes of the objects from other formal contexts they are in relation with. For instance, the concept lattice associated with the formal context about softwares before extension would organise them depending on their spam prevention and whether they are open source or not. Then, the relationship "`is-written-in`" would extend the concept lattice to also organise the softwares depending on the paradigms of the programming languages they are written in. An example of an information that could be found in this extended concept lattice may be "all the softwares that are written in at least one object oriented programming language are open source".

Mining algorithms can be applied on extended concept lattices to extract logical relationships between binary attributes and relational attributes, thus representing logical relationships between binary attributes of different lattices. Applied on connected sets of related software systems, it would allow to extract relationships between features of different software families.

4 Research Plan

In what follows, we propose research directions to address the problem of extracting complex variability information within a set of related software descriptions, and in between different yet interconnected software families. We

propose a methodology based on Formal Concept Analysis and its extensions to extract augmented/inter-model variability information in the form of logical relationships, and we specify the expected output format.

4.1 Research Questions

We seek to extract the variability information corresponding to three FM extensions proposed to handle complex product lines. As we have seen in the introduction, the modelling needs of a complex product line are two-fold: needs to model more complex information, and needs to cope with large scale models. Thus, we decompose our goal in two research questions.

RQ1: How to extract the augmented variability information necessary to synthesise a feature model with extended modelling capabilities? Here we focus on the necessity to extract more complex information. This question concerns the extensions applicable on a single FM, i.e., the ones that complement traditional boolean FMs with multi-valued attributes and cardinalities.

RQ2: How to extract the inter-model variability information necessary to synthesise interconnected feature models? This time, we focus on the necessity to cope with the size of a large product line. The question concerns the extension consisting in defining references between several (possibly extended) FMs.

4.2 Data and Methodology

To answer these two research questions, we will work on data taken from *Product Comparison Matrices* (PCMs) [SAB13]. PCMs are matrices that depict a set of products against a set of their characteristics, hence representing a set of products of the same family in a way that ease their comparison by a user. We decided to work on PCMs for two reasons. First, they gather in their cells heterogeneous data, including both binary attributes (yes/no values that can be considered as features) and multi-valued attributes. Thus, they are interesting candidates for the extraction of augmented variability information. Also, the large number of PCMs that can be found on internet and the large scope of product families they represent allow to find links between different PCMs. More precisely, one can find PCMs having characteristics whose values represent the products of other PCMs. In these cases, one can be able to extract references and therefore inter-model variability information between several product families. The main drawback of using PCMs lies in the fact that they are not formalised and in most cases they need to be cleaned before being automatically processed [SAB13, NBA$^+$17].

RQ1: To extract augmented variability information in the form of logical relationships, we will use traditional Formal Concept Analysis along with Pattern Structures.

Traditional FCA allows to extract all implications, co-occurrences and mutex between features, as well as feature groups (or-groups and xor-groups) [CHN17,

RPK11, LP07]. Moreover, by analysing the extents of concepts introducing features involved in a feature group, it is possible to extract the corresponding group cardinality. Figure 6 presents the grammar of the variability information that can be extracted with traditional FCA. We simplify the logical relationships representing feature-groups by introducing the notation $(p, \{f_1, ..., f_n\}, < min - max >)$ to be used instead of the one aforementioned in Section 2.2.2.

Besides, Pattern Structures permit to extract, in addition to variability information extracted by traditional FCA, all implications, co-occurrences and mutex between a feature and an attribute value, and between two attribute values. Also, given a PCM's characteristics, one can extract feature cardinalities by 1) analysing the minimum and maximum number of values in the cells of a characteristic, or by 2) identifying characteristics with numerical values representing the occurrence of a feature. A feature cardinality is then handled as a multi-valued attribute with a numerical type. It is noteworthy that, as FCA is applied on Pattern Structures, all the variability information extracted with FCA can also be extracted with Pattern Structures. The grammar of the variability information that can be extracted using Pattern Structures (minus the ones extracted with traditional FCA) is presented in Figure 7.

RQ2: To extract inter-model variability information between different software families, we will use Relational Concept Analysis. By synthesising relational contexts depending on PCMs' characteristic values representing the products of other PCMs, one can build interconnected concept lattices with RCA. Then, it is possible to apply usual algorithms on extended concept lattices to extract all implications, co-occurrences and mutex between features of different software families (i.e., different PCMs). Figure 8 presents the grammar of the variability information that can be extracted with RCA. As for Pattern Structures, we only present the information that is specific to an extraction using RCA.

We sum up the variability information types that can be extracted and by which frameworks in Table 3. For now, we envision to extract logical relationships involving only features between different software families. However, research has been made about the combination of Pattern Structures and RCA [CN14]. This would permit to extract augmented variability information between software families: for instance, a co-occurrence between a feature of a software family, and an attribute value of another software family. This is left as future considerations.

5 Related Work

FM extraction from product descriptions: Several dedicated algorithms for the synthesis of boolean FMs can be found in the litterature [ACP$^+$12, DDH$^+$13, HLE11, HLE13]. Some authors use search-based methods to perform the extraction, as [LLE14] with genetic programming, and [LLG$^+$15] assessing evolutionary algorithms, hill climbing and random search for this task. Loesch and Plodereder [LP07] first use FCA to analyse and reorganise an FM through

Table 3: List of variability information types depending on the frameworks permiting to extract them

Variability information	FCA	Pattern Structures	RCA
implication	X	X	X
co-occurrence	X	X	X
mutex	X	X	X
feature group and group cardinality	X	X	X
augmented implication		X	
augmented co-occurrence		X	
augmented mutex		X	
inter-model implication			X
inter-model co-occurrence			X
inter-model mutex			X

the variability information provided by conceptual structures built from its set of valid configurations. Ryssel et al. [RPK11] and then Al-Msie'Deen et al. [AMHS+14] propose to extract an FM from a set of software descriptions encoded in a formal context. All these works have in common the fact that they synthesise boolean FMs representing constraints between features, whereas we aim to extract more complex constraints to later build extended FMs. To the best of our knowledge, Becan et al. [BBGA15] are the only ones that propose a method to extract extended FMs (in the form of attributed FMs) from product descriptions. They propose dedicated algorithms to extract, in addition to regular boolean FMs relationships, a set of implications involving at least one attribute value. In comparison, our approach would allow to extract mutex and co-occurrences involving attribute values, as well as implications, mutex and co-occurrences involving elements of separate FMs. Also, the frameworks we use for these tasks give mathematical foundations to the variability extraction algorithms, and lie on a single and canonical structure that naturally emphasises the variability of a set of variants.

Pattern Structures: Ganter and Kuznetsov [GK01] first introduced Pattern Structures for an application on graphs representing molecules. Since, it has been used fo knowledge discovery on complex data types, e.g., mining sequential data [BEJ+16], RDF triples classification [BCNR17, RATN] or functional dependencies extraction [BKN14].

Relational Concept Analysis: RCA was first used to build class model abstractions [AFHN06]. It has been applied for restructuring ontologies [STNB11, RHVN11], refactoring design defects [MHVG08] or extracting tranformation rules from model transformation examples [SDH+12].

6 Conclusion

In this paper, we studied three extensions of FMs (multi-valued attributes, cardinalities and references) to cope with complex product line variability modelling. We presented Formal Concept Analysis as a mathematical framework that brings theoretical foundations to the extraction of variability information from product descriptions. We also presented two FCA extensions, Pattern Structures and Relational Concept Analysis that allow to extract variability information involving cardinalities, multi-valued attributes and features from different yet connected software families. We linked variability information that can be extracted by these three frameworks with the one found in extended FMs. We proposed research directions to extract complex variability information using these frameworks, as a part of extended FM synthesis from product descriptions.

In the future, we plan to apply this methodology on existing datasets representing software family descriptions. We will analyse the extracted variability information and study different methods to separate the consistent from the inconsistent ones, and to provide a ranking of the most pertinent information for the expert. As we have said before, we will study the combination of Pattern Structures and Relational Concept Analysis to extract augmented variability information between different extended FMs. Finally, we will consider the second part of the feature model synthesis that consists in building the FM from the extracted information.

References

[ACP+12] Mathieu Acher, Anthony Cleve, Gilles Perrouin, Patrick Heymans, Charles Vanbeneden, Philippe Collet, and Philippe Lahire. On extracting feature models from product descriptions. In *Proc. of the 6th Int. Workshop on Variability Modelling of Software-Intensive Systems (VaMoS'12)*, pages 45–54, 2012.

[AFHN06] Gabriela Arévalo, Jean-Rémy Falleri, Marianne Huchard, and Clémentine Nebut. Building Abstractions in Class Models: Formal Concept Analysis in a Model-Driven Approach. In *Proc. of the 9th Int. Conference on Model Driven Engineering Languages and Systems (MoDELS'06)*, pages 513–527, 2006.

[AMdSH+14] Ra'Fat Al-Msie 'deen, Abdelhak-Djamel Seriai, Marianne Huchard, Christelle Urtado, Sylvain Vauttier, and Ahmad Al-Khlifat. Concept lattices: A representation space to structure software variability. In *Proc. of the 5th Int. Conference on Information and Communication Systems (ICICS'14)*, pages 1–6, 2014.

[AMHS+14] Ra'Fat Al-Msie'deen, Marianne Huchard, Abdelhak Seriai, Christelle Urtado, and Sylvain Vauttier. Reverse Engineering Feature

Models from Software Configurations using Formal Concept Analysis. In *Proc. of the 11th Int. Conference on Concept Lattices and Their Applications (CLA'14)*, pages 95–106, 2014.

[BBGA15] Guillaume Bécan, Razieh Behjati, Arnaud Gotlieb, and Mathieu Acher. Synthesis of attributed feature models from product descriptions. In *Proc. of the 19th Int. Conference on Software Product Line (SPLC'15)*, pages 1–10, 2015.

[BCNR17] Quentin Brabant, Miguel Couceiro, Amedeo Napoli, and Justine Reynaud. From Meaningful Orderings in the Web of Data to Multi-level Pattern Structures. In *Proc. of 23rd Int. Symposium on Foundations for Intelligent Systems (ISMIS'17)*, pages 622–631, 2017.

[BEJ+16] Aleksey Buzmakov, Elias Egho, Nicolas Jay, Sergei O. Kuznetsov, Amedeo Napoli, and Chedy Raïssi. On mining complex sequential data by means of FCA and pattern structures. *Int. Journal of General Systems*, 45(2):135–159, 2016.

[BKN14] Jaume Baixeries, Mehdi Kaytoue, and Amedeo Napoli. Characterizing functional dependencies in formal concept analysis with pattern structures. *Annals of Mathematics and Artificial Intelligence*, 72(1-2):129–149, 2014.

[Bos09] Jan Bosch. From software product lines to software ecosystems. In *Proc. of the 13th Int. Software Product Lines Conference (SPLC'09)*, pages 111–119, 2009.

[Bot13] Goetz Botterweck. Variability and Evolution in Systems of Systems. In *Proc. of the 1st Workshop on Advances in Systems of Systems (AiSoS'13)*, pages 8–23, 2013.

[BSRC10] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6):615–636, 2010.

[CBUE02] Krzysztof Czarnecki, Thomas Bednasch, Peter Unger, and Ulrich W. Eisenecker. Generative Programming for Embedded Software: An Industrial Experience Report. In *Proc. of the 1st ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering (GPCE'02)*, pages 156–172, 2002.

[CHE04] Krzysztof Czarnecki, Simon Helsen, and Ulrich W. Eisenecker. Staged Configuration Using Feature Models. In *Proc. of the 3rd Int. Conference on Software Product Lines (SPLC'04)*, pages 266–283, 2004.

[CHN17] Jessie Carbonnel, Marianne Huchard, and Clémentine Nebut. Analyzing Variability in Product Families through Canonical Feature Diagrams. In *Proc. of the 29th Int. Conference on Software Engineering & Knowledge Engineering (SEKE'17)*, pages 185–190, 2017.

[CN14] Víctor Codocedo and Amedeo Napoli. A proposition for combining pattern structures and relational concept analysis. In *Proc. of the 12th Int. Conference on Formal Concept Analysis (ICFCA'14)*, pages 96–111, 2014.

[CW07] Krzysztof Czarnecki and Andrzej Wasowski. Feature Diagrams and Logics: There and Back Again. In *Proc. of the 11th Int. Conference on Software Product Lines (SPLC'07)*, pages 23–34, 2007.

[DDH+13] Jean-Marc Davril, Edouard Delfosse, Negar Hariri, Mathieu Acher, Jane Cleland-Huang, and Patrick Heymans. Feature model extraction from large collections of informal product descriptions. In *Proc. of the 9th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'13)*, pages 290–300, 2013.

[FKL+06] Steven Fraser, Gregor Kiczales, Ricardo Lopez, Peter G. Neumann, Linda M. Northrop, Martin C. Rinard, Douglas C. Schmidt, and Kevin J. Sullivan. The ultra challenge: software systems beyond big. In *Companion to the 21th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'06)*, pages 929–933, 2006.

[GK01] Bernhard Ganter and Sergei O. Kuznetsov. Pattern Structures and Their Projections. In *Proc. of the 9th Int. Conference on Conceptual Structures (ICCS'01)*, pages 129–142, 2001.

[GW99] Bernhard Ganter and Rudolf Wille. *Formal concept analysis - mathematical foundations*. Springer, 1999.

[HGR12] Gerald Holl, Paul Grünbacher, and Rick Rabiser. A systematic review and an expert survey on capabilities supporting multi product lines. *Information & Software Technology*, 54(8):828–852, 2012.

[HHNV13] Mohamed Rouane Hacène, Marianne Huchard, Amedeo Napoli, and Petko Valtchev. Relational concept analysis: mining concept lattices from multi-relational data. *Annals of Mathematics and Artificial Intelligence*, 67(1):81–108, 2013.

[HLE11] Evelyn Nicole Haslinger, Roberto E. Lopez-Herrejon, and Alexander Egyed. Reverse Engineering Feature Models from Programs' Feature Sets. In *Proc. of the 18th Working Conference on Reverse Engineering (WCRE'11)*, pages 308–312, 2011.

[HLE13] Evelyn Nicole Haslinger, Roberto Erick Lopez-Herrejon, and Alexander Egyed. On Extracting Feature Models from Sets of Valid Feature Combinations. In *Proc. of the 16th Int. Conference on Fundamental Approaches to Software Engineering (FASE'13), Held as Part of the European Joint Conferences on Theory and Practice of Software (ETAPS'13)*, pages 53–67, 2013.

[KCH+90] Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-021, 1990.

[LLE14] Lukas Linsbauer, Roberto Erick Lopez-Herrejon, and Alexander Egyed. Feature Model Synthesis with Genetic Programming. In *Proc. of the 6th Int. Symposium on Search-Based Software Engineering (SSBSE'14)*, pages 153–167, 2014.

[LLG+15] Roberto Erick Lopez-Herrejon, Lukas Linsbauer, José A. Galindo, José Antonio Parejo, David Benavides, Sergio Segura, and Alexander Egyed. An assessment of search-based techniques for reverse engineering feature models. *Journal of Systems and Software*, 103:353–369, 2015.

[LP07] Felix Loesch and Erhard Ploedereder. Restructuring Variability in Software Product Lines using Concept Analysis of Product Configurations. In *Proc. of the 11th European Conference on Software Maintenance and Reengineering, Software Evolution in Complex Software Intensive Systems (CSMR'07)*, pages 159–170, 2007.

[Man02] Mike Mannion. Using First-Order Logic for Product Line Model Validation. In *Proc. of the 2nd Int. Conference on Software Product Lines (SPLC'02)*, pages 176–187, 2002.

[MHVG08] Naouel Moha, Amine Rouane Hacene, Petko Valtchev, and Yann-Gaël Guéhéneuc. Refactorings of design defects using relational concept analysis. In *Proc. of the 6th Int. Conference in Formal Concept Analysis (ICFCA'08)*, pages 289–304, 2008.

[NBA+17] Sana Ben Nasr, Guillaume Bécan, Mathieu Acher, João Bosco Ferreira Filho, Nicolas Sannier, Benoit Baudry, and Jean-Marc Davril. Automated extraction of product comparison matrices from informal product descriptions. *Journal of Systems and Software*, 124:82–103, 2017.

[PBvdL05] Klaus Pohl, Günter Böckle, and Frank J van der Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer Science & Business Media, 2005.

[PEVD10] Jonas Poelmans, Paul Elzinga, Stijn Viaene, and Guido Dedene. Formal concept analysis in knowledge discovery: a survey. In *Proc. of the 8th Int. Conference on Conceptual Structures (ICCS'10)*, pages 139–153, 2010.

[PSA+12] Jeff Z Pan, Steffen Staab, Uwe Aßmann, Jürgen Ebert, and Yuting Zhao. *Ontology-driven software development*. Springer Science & Business Media, 2012.

[RATN] Justine Reynaud, Mehwish Alam, Yannick Toussaint, and Amedeo Napoli. A Proposal for Classifying the Content of the Web of Data Based on FCA and Pattern Structures. In *Proc. of 23rd Int. Symposium on Foundations for Intelligent Systems (ISMIS'17)*, pages 684–694.

[RHVN11] Mohamed Rouane-Hacene, Petko Valtchev, and Roger Nkambou. Supporting ontology design through large-scale FCA-based ontology restructuring. In *Conceptual Structures for Discovering Knowledge*, pages 257–269. Springer, 2011.

[RPK11] Uwe Ryssel, Joern Ploennigs, and Klaus Kabitzsch. Extraction of feature models from formal contexts. In *Workshop Proc. (Volume 2) of the 15th Int. Conference on Software Product Lines (SPLC'11)*, pages 4:1–4:8, 2011.

[RSKuR08] Marko Rosenmüller, Norbert Siegmund, Christian Kästner, and Syed Saif ur Rahman. Modeling dependent software product lines. In *Proc. of the GPCE Workshop on Modularization, Composition and Generative Techniques for Product Line Engineering (McGPLE'08)*, pages 13–18, 2008.

[SAB13] Nicolas Sannier, Mathieu Acher, and Benoit Baudry. From comparison matrix to Variability Model: The Wikipedia case study. In *Proc. of the 28th Int. Conference on Automated Software Engineering (ASE'13)*, pages 580–585, 2013.

[SDH+12] Hajer Saada, Xavier Dolques, Marianne Huchard, Clémentine Nebut, and Houari A. Sahraoui. Generation of Operational Transformation Rules from Examples of Model Transformations. In *Proc. of the 15th Int. Conference on Model Driven Engineering Languages and Systems (MODELS'12)*, pages 546–561, 2012.

[Sne00] Gregor Snelting. Software reengineering based on concept lattices. In *Proc. of the 4th European Conference on Software Maintenance and Reengineering (CSMR'00)*, pages 3–10, 2000.

[STNB11] Lian Shi, Yannick Toussaint, Amedeo Napoli, and Alexandre Blansché. Mining for Reengineering: An Application to Semantic Wikis Using Formal and Relational Concept Analysis. In *Proc. of the 8th Extended Semantic Web Conference (ESWC'11)*, pages 421–435, 2011.

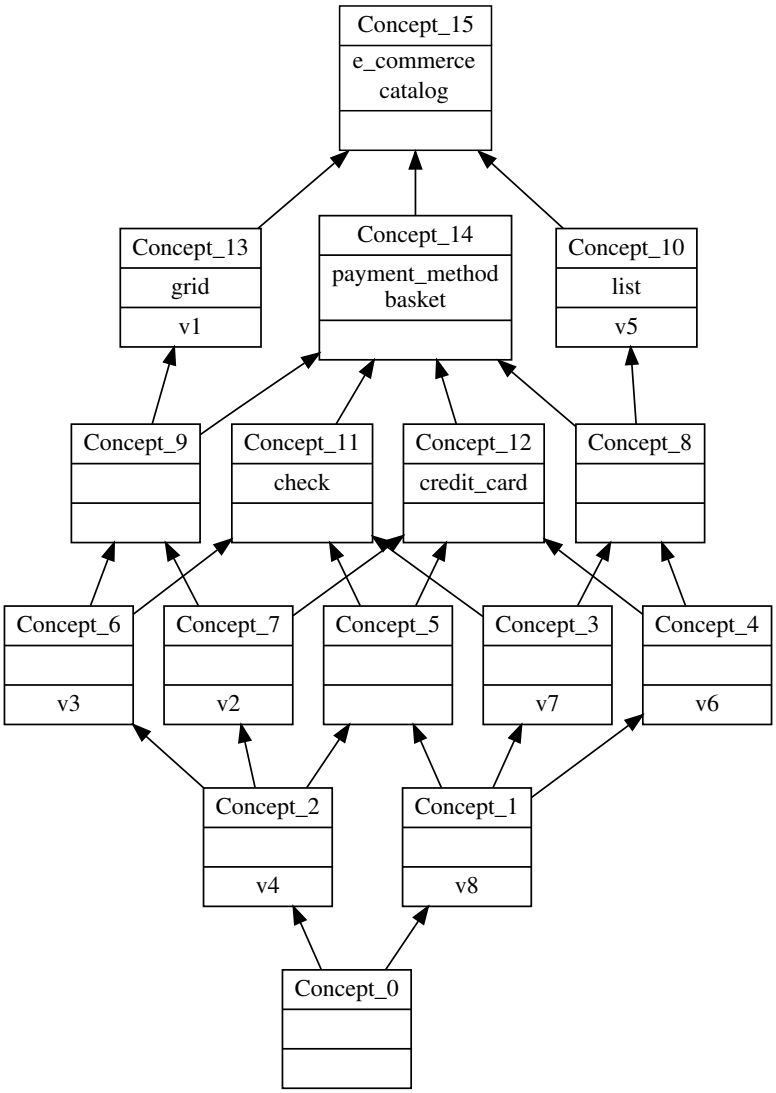

Figure 5: Concept lattice associated with the formal context of Table 2

variability information	:=	*relationship**
relationship	:=	*implication* — *co-occurrence* — *mutex* — *group*
implication	:=	*feature* '→' *feature*
co-occurrence	:=	*feature* '↔' *feature*
mutex	:=	*feature* '→' '¬' *feature*
group	:=	'(' *feature* ',' '{' *feature_set* '}' ',' *cardinality* ')'
feature_set	:=	*feature* — *feature_set* ',' *feature*
feature	:=	**feature_name**
cardinality	:=	'<' **nb_min** ',' **nb_max** '>'

Figure 6: Grammar of variability information that can be extracted with traditional FCA

variability information	:=	*relationship**
relationship	:=	*implication* — *co-occurrence* — *mutex*
implication	:=	*augmented element* '→' *augmented element*
co-occurrence	:=	*augmented element* '↔' *augmented element*
mutex	:=	*augmented element* '→' '¬' *augmented element*
augmented element	:=	*feature* — *attribute*
feature	:=	**feature_name**
attribute	:=	**attribute_name** '=' **value**

Figure 7: Grammar of the augmented variability information that can be extracted with Pattern Structures

variability information	:=	*relationship**
relationship	:=	*implication* — *co-occurrence* — *mutex*
implication	:=	*inter-model element* '→' *inter-model element*
co-occurrence	:=	*inter-model element* '↔' *inter-model element*
mutex	:=	*inter-model element* '→' '¬' *inter-model element*
inter-model element	:=	*software family* ':' *feature*
feature	:=	**feature_name**
software family	:=	**family_name**

Figure 8: Grammar of the inter-model variability information that can be extracted with RCA