



**HAL**  
open science

# Extending Boolean Variability Relationship Extraction to Multi-valued Software Descriptions

Jessie Galasso-Carbonnel, Marianne Huchard

► **To cite this version:**

Jessie Galasso-Carbonnel, Marianne Huchard. Extending Boolean Variability Relationship Extraction to Multi-valued Software Descriptions. Roberto E. Lopez-Herrejon; Jabier Martinez; Wesley Klewerton Guez Assunção; Tewfik Ziadi; Mathieu Acher; Silvia Vergilio. Handbook of Re-Engineering Software Intensive Systems into Software Product Lines, Springer International Publishing, pp.143-173, 2023, 978-3-031-11686-5. 10.1007/978-3-031-11686-5\_6 . lirmm-04089535

**HAL Id: lirmm-04089535**

**<https://hal-lirmm.ccsd.cnrs.fr/lirmm-04089535>**

Submitted on 4 May 2023

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

## Chapter 6

# Extending boolean variability relationship extraction to multi-valued software descriptions

Jessie Galasso and Marianne Huchard

**Abstract** Extracting variability information from software product descriptions is crucial when reverse engineering a software product line, e.g., for variability model synthesis. Existing methods are predominantly designed for feature-oriented product lines, where products are described by the set of distinguishable features they implement, and variability information may be expressed by logical relationships over these features. However, limits of such boolean feature-based variability modeling approaches have been highlighted, notably regarding their expressive power. In this chapter, we take a step towards more complex variability extraction and focus on extracting non-boolean variability relationships from multi-valued software descriptions. We first analyze software descriptions, variability relationships and extraction methods used in the boolean case. We attract attention to a knowledge engineering framework supporting a sound and complete feature-based variability relationship extraction method. The benefits of this framework include several extensions enabling to take into account more complex datasets than boolean ones. We explore one of these extensions to extend the traditional boolean extraction method and handle variability relationships including both boolean features and attribute values that can be used to synthesize extended variability models.

### 6.1 Introduction

Reducing the development time and cost of a software portfolio while increasing its overall quality and scope may be achieved through systematic reuse and mass-customization; it is the core of the Software Product Line Engineering (SPLE)

---

Jessie Galasso  
Université de Montréal, Canada e-mail: [jessie.galasso-carbonnel@umontreal.ca](mailto:jessie.galasso-carbonnel@umontreal.ca)

Marianne Huchard  
LIRMM, University of Montpellier, CNRS, Montpellier, France e-mail: [marianne.huchard@lirmm.fr](mailto:marianne.huchard@lirmm.fr)

approach [48]. SPLE is a development paradigm based on a set of reusable software artifacts and a common architecture with which the artifacts may be combined: different combinations lead to different software products that share similarities. Altogether, the common architecture, the reusable artifacts and the derivable products form a software product line (SPL). A central task of SPLE is to document the SPL *variability*, i.e., documenting the accepted artifact combinations and thus the derivable software products. This task produces *variability models* which are crucial for the understanding, implementation, management, maintenance and evolution of the SPL. Variability models can operate at several levels of abstraction but most commonly group artifacts under distinguishable software characteristics or behaviours called *features*. Feature-oriented product line engineering [4, 34] is a widely adopted approach for SPL implementation, where feature diagrams (FDs) [33] are the most commonly studied variability models. FDs are a family of description languages that enable to express constraints (also called *variability relationships*) over a set of features and thus delimit the scope of an SPL. In other words, FDs describe derivable software intensive systems through boolean descriptions being accepted combinations of features. However, limitations regarding the expressiveness of these “boolean” FDs have been encountered, and FD extensions were proposed to overstep them, e.g., to represent feature cardinalities [16], group cardinalities [16, 50], or multi-valued attributes [7, 16]. Aside from these extended FDs, other types of variability models try to tackle the problem of representing more detailed variability information, such as orthogonal variability models [48] or the common variability language [31].

Proactive adoption of an SPL [37] instructs to first determine its scope (i.e., establishing which artifacts will be needed and which combinations should be allowed through variability models), then implement the architecture and the artifacts, and finally begin to derive software products. However, this adoption process takes time before being able to propose working products to customers, that is why a large part of companies working with SPLE prefer to adopt an extractive approach [11]. Extractive SPL adoption [37] is about capitalizing on similar software products already developed to fasten the definition of the common software architecture, the set of reusable artifacts and the variability models through reverse engineering methods. A detailed overview of current research in SPL reverse engineering can be found in [6] in the form of a mapping study. An abundant literature addresses automated or semi-automated extraction of variability relationships between features from (boolean) software intensive-system descriptions, mostly to synthesize boolean FDs [2, 3, 16, 21, 29, 30, 38, 41, 45, 51]. Variability extraction that goes beyond boolean feature relationships is addressed by few authors: Becan et al. consider boolean features and multi-valued attributes [9], and Carbonnel et al. consider boolean features, UML-like cardinalities and multi-valued attributes [15].

In this chapter, we present a method to extract variability relationships involving more than boolean features from product descriptions. More specifically, we focus on the types of relationships to be extracted to synthesize the three prevalent FD extensions that were proposed to enhance their expressiveness. We study product descriptions that take the form of multi-valued matrices containing both boolean and non-boolean characteristics. The proposed method leverages existing boolean

variability relationships extraction methods based on Formal Concept Analysis (FCA) [25], a framework for knowledge engineering and knowledge representation. Even though the value of FCA may be difficult to apprehend at first, it is one-of-a-kind in the variability reverse engineering landscape; FCA offers a structural, reusable and extensible framework which formalizes variability extraction and representation, and thus includes most of the existing variability extraction approaches found in the literature. In the following, we first present a boolean variability extraction method based on FCA (Section 6.2), and then a way to extend this framework to handle multi-valued variability extraction (Section 6.3). In each section, we present the input product descriptions, the variability relationships to be extracted, the reverse engineering approaches found in the literature, and a sound and complete FCA-based extraction method (i.e., all variability relationships that are true for the considered set of input product descriptions are extracted, and all extracted relationships are true). We do not address the problem of synthesizing variability models such as FDs extended with attributes or cardinalities; we focus on extracting variability information independently from any representation, but that can be used to build different kinds of variability models. The soundness and completeness of the extraction method provide strong foundations for the synthesis of variability models which are consistent with the input descriptions, but do not guarantee the legibility and/or maintainability of the produced models as these quality attributes depend on the synthesis method. We conclude in Section 6.4 by summarizing the approach and drawing a few perspectives.

## 6.2 Variability in boolean descriptions

Descriptions composed of boolean feature sets have been extensively studied from the very first research works on product line engineering [33]. In this section, we focus on the variability relationships which can be extracted from this type of descriptions, and how. Feature-based boolean descriptions are discussed in Section 6.2.1, where we introduce an illustrative example derived from the robot battle programming game Robocode<sup>1</sup> [43]. Section 6.2.2 reviews the main variability relationships between features that have been studied in the literature, and Section 6.2.3 reports the main approaches for feature relationship extraction. The key principles and benefits of an extraction process involving Formal Concept Analysis are exposed in Section 6.2.4.

### 6.2.1 Boolean feature-based descriptions

In SPLE, a *feature* corresponds to a high level product's part or behavior which is relevant to any stakeholder [4]. Features are intended to be easily understandable contrary to artifacts of lower level that only experts may comprehend. They represent

---

<sup>1</sup> <https://robocode.sourceforge.io/>

noticeable characteristics which help distinguish similar products from one another; a feature is associated to a unique name and may be implemented by various (concrete) artifacts. Let us consider an example about Robocode, a programming game where developers have to implement the behaviour of their own bots (i.e., the products to be described) in order to fight bots implemented by other developers. Examples of features characterizing a bot may be its strategies of movement on the battlefield (e.g. `random movement`, `minimum risk movement`, `stop&go`, `wave surfing`), the kinds of fights it is designed for (`one-on-one fighting` or `melee fighting`), if it takes part to some community events (such as the `roborumble competition`), or the software license of the code source (`open source` or `closed license`). Let us note that all features may not be at the same level of abstraction: for instance, the feature `license` is more abstract (or generalised) than the feature `open source`.

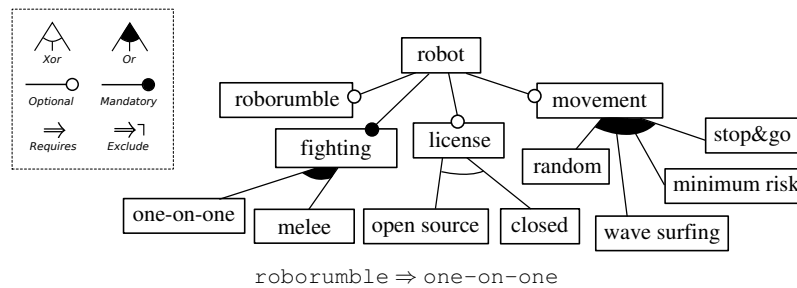
In a feature-oriented approach [34], a product is designated by the set of features it possesses. Such a set of features is called a product *configuration* and may be considered an abstract, high level description of the product. For example, `{one-on-one fighting, melee fighting, roborumble}` is a configuration. However, some configurations may not be admitted, for instance, if the artifacts implementing the features are not compatible regarding some domain or business constraints. For instance, a product cannot have both an `open source` and a `closed license`. Accepted combinations of features may be delineated through constraints defining a set of valid configurations, also called the product line scope. Constraints over a finite set of features restricting the way they can be combined are called variability relationships and express what is called the *variability* of the product line, which is traditionally documented through variability models.

## 6.2.2 Boolean variability relationships

**Feature relationships.** If one maps features to propositional variables in propositional logic [42], then any propositional formula may be used to express variability relationships in the form of constraints between features (also called feature dependencies or feature relationships). However, for a product designer, a simplified dependency language is more suitable to highlight the principal feature relationships. Hence, four kinds of feature relationships gathered most of the attention of practitioners. These relationships express situations where: ① a feature requires another feature, ② two features are mutually exclusive, ③ a feature should be refined by selecting at least one feature in a group and ④ a feature should be refined by selecting exactly one feature in a group. For example, to document Robocode variability, one may want to define feature relationships stating that: a bot participating in the `roborumble competition` requires to be designed for `one-on-one fighting`; a `random movement strategy` is not appropriate for `melee fighting`; the implemented movement strategies should be specified amongst the feature group `{random, wave surfing, stop&go, minimum risk}` (at least one but possibly more);

the source code `license` should be specified amongst the group `{open-source, closed}` (exactly one). These four kinds of relationships may be expressed by propositional formulas using respectively: binary implications for “requires” relationships (`roborumble`  $\Rightarrow$  `one-on-one` fighting), binary implications with negations for “exclude” relationships (`random` movement  $\Rightarrow \neg$  `melee` fighting), equivalences with or-connectives for “at least one” feature refinement (`movement`  $\Leftrightarrow$  (`random`  $\vee$  `wave surfing`  $\vee$  `stop&go`  $\vee$  `minimum risk`)) and equivalences with xor-connectives for “exactly one” feature refinement (`license`  $\Leftrightarrow$  (`open-source`  $\oplus$  `closed`)). Those are thus the basic feature relationships usually expressed in boolean variability models.

**Feature diagram.** The *de facto* standard for representing a product line variability in a diagram-like representation is a type of variability model called *feature diagram* (FD) [33, 57]. Fig. 6.1 presents an example of FD representing a possible view of the variability of Robocode bots.



**Fig. 6.1** Excerpt of a possible FD depicting the variability of Robocode bots

The box-shaped nodes represent the features. They are structured in a tree representing child-parent (or refinement) relationships. The edges of the tree must be decorated to express some constraints (amongst the four depicted in the top part of the box in the left-hand side of Fig. 6.1) restricting the way the features may be selected when choosing a valid configuration. A white disc states that the child-feature may be optionally selected when the parent-feature is present in the configuration, and a black disc forces the child-feature to be selected. Arcs are used to group features; when the parent feature of the group is selected: if the arc is filled with black then at least one feature must be selected from the group (also called or-group); if the arc is non-filled then exactly one feature must be selected from the group (also called xor-group). Cross-tree constraints in the form of “requires” and “exclude” constraints may be added (textually or graphically) to complete the graphical representation. The FD presented in Fig. 6.1 states that: a bot necessarily implements at least one fighting style, which may be `melee`, `one-on-one` or both. It can optionally participate to the `roborumble` competition, but if it does, then it should implement the `one-on-one` fighting style. A bot can optionally have a license which is either open source or closed, but not both. Finally, it optionally implements at least one movement strategy amongst four possible strategies.

**Link with propositional logic.** The relationships expressed in feature diagrams may be mapped to the four basic feature relationships discussed at the beginning of the section, and therefore to their representations in propositional logic [10, 16, 42]. Straightforwardly, mutually exclusive features (situation ②) are expressed through “exclude” cross-tree constraints, and feature group refinement (situations ③ and ④) are expressed through or- and xor-groups, respectively. Child-parent relationships implicitly express “require” relationships (situation ①); for instance, if one wants to select `open source` (which is a *kind-of* license) in a configuration, he or she needs to select the feature `license` beforehand, thus `open source`  $\Rightarrow$  `license`. Note that such “require” relationships may also be expressed through cross-tree constraints (such as `roborumble`  $\Rightarrow$  `one-on-one`) and mandatory relationships (all bots have a fighting style, thus `robot`  $\Rightarrow$  `fighting`). The first two columns of Table 6.2 depict this mapping: the first column shows the feature relationships in their propositional form with a particular case of situation ① when there are symmetric binary implications (i.e., logical equivalences) denoted by ①’, and the second column shows the FD syntax of the corresponding variability relationships. Third and fourth columns will be addressed later in the section. We can see that: a binary implication (①) may correspond to a child-parent relationship, an optional relationship or a “require” cross-tree constraint; an equivalence (①’) may correspond to a mandatory relationship (because it is the combination of two binary implications: one comes from the child-parent relationship and the other comes the mandatory constraint) or may correspond to two symmetric “require” cross-tree constraints; mutual exclusions (②) and groups (③ and ④) have only one representation in the feature diagram syntax.

**Semantics.** The conjunction of the “propositional logic forms” of feature relationships depicted in an FD outlines a propositional formula (called the FD’s *logical semantics*). The formula’s models correspond to the FDs’ set of valid configurations (called its *configuration semantics*). Another semantics, called the *ontological semantics*, is given by the tree structure of the diagram that conveys meanings to the modeled domain. For instance, a situation where a feature  $f_1$  requires another feature  $f_2$  (logically represented by  $f_1 \Rightarrow f_2$ ) may be represented in an FD by different relationships having different meanings, e.g., through the feature hierarchy ( $f_2$  refines  $f_1$ ), a mandatory relationship ( $f_2$  is a necessary refinement of  $f_1$ ), or through a cross-tree constraint ( $f_1$  necessitates  $f_2$  but does not generalize nor specialize it).

**Other boolean variability models.** In some cases, when one needs to express more complex feature relationships, the expressiveness of FD graphical syntax is not sufficient. A *feature model* is a *feature diagram* supplemented by a complex propositional logic formula capturing the constraints which cannot be expressed by an FD [57]. In this context, a “complex” propositional formula is a formula which cannot be written as a conjunction of “require” and “exclude” cross-tree constraints; if it is the case, then it is considered an FD (as the one of Fig. 6.1). Other formalisms have been introduced. Binary Implication Graphs (BIG), as their name suggests, graphically represent binary implications [1, 2, 20, 21, 56, 57]. Directed hypergraphs have been presented in [20]. In this representation, each binary implication is represented by a directed binary edge, while other constraints (feature groups and mutex) are represented by hyperedges. Mutex graphs [56, 57] only show pairwise incompatibilities

between features. Feature Diagram Generalized Notation and Feature Graphs [20] represent binary implications and groups, while relaxing the tree-structure constraint of traditional FDs. She et al. [57] add mutex to these representations. Equivalence Class Feature Diagrams (ECFD) have been proposed in [14] as a canonical structure for variability representation. They represent binary implications, groups, mutex as the other structures, but may also highlight generalized exclusions (i.e., groups of features that never appear together).

### ***6.2.3 Feature relationship extraction in the literature***

Becan et al. empirically show that variability model extraction approaches based on 1) logical heuristics extracting the logical semantics combined with 2) ontological heuristics relying on user or expert decisions for associating meaningful knowledge to the extracted logical relationships outperform other approaches [8]. In fact, the first one guarantees correct configuration and logical semantics, while the second one guarantees a correct ontological semantics. In this chapter, we mainly focus on the first step of this type of variability model synthesis, i.e., the extraction of variability information (feature relationships) in the form of logical relationships. A sound and complete feature relationship extraction method is crucial to ensure correct foundations for meaningful variability model synthesis.

In the literature, feature relationships extraction has been studied from various perspectives. As feature diagrams may not represent all configuration sets, most authors synthesize feature models. Several dedicated algorithms to build a feature model from a set of valid configurations can be found. She et al. [57] propose a sound and complete feature relationship extraction by enhancing the method defined by Czarnecki and Wasowski [20] which originally did not extract mutually exclusive features; these methods rely on binary decision diagrams to extract interim variability representations in the form of binary implication graph and mutex graph. Acher et al. [2] also present a dedicated algorithm for a sound and complete extraction using binary implication graphs. Haslinger et al. [29, 30] propose a recursive algorithm to build feature models without extracting feature relationships beforehand; the soundness and completeness of the extraction method is not assessed. Davril et al. [21] extract relevant features from informal documents to produce a configuration-by-feature matrix, and all valid implications. They use text-mining techniques and feature co-occurrences to identify meaningful implications. The feature group extraction is sound but not complete, and these properties are not assessed for the other feature relationships. Ferrari et al. [23] analyze product descriptions in the form of commercial texts to propose candidate optional and mandatory features to an expert. Search-based techniques are applied in several works. The authors in [38] and [41] explore these techniques and build FMs whose configuration semantics approximate a given configuration set. Assunção et al. [5] propose to use 2 multi-objective evolutionary algorithms for reverse engineering feature models. The multi-objective perspective allows the expert to tune the reverse engineering process depending on the objectives (e.g., correct



configuration semantics, legibility) they identified as important in a given context. They take into account knowledge from existing variants' source code (in the form of a weighted directed graph representing source code dependencies) to ensure that the output models' configurations correspond to well formed software variants with regard to their source code. Czarnecki et al. propose an algorithm based on data-mining techniques to build Probabilistic Feature Models in [19]. These extraction methods are not deterministic. In [58], Temple et al. elaborate a technique to build a variability model that can be configured for different contexts, using classification trees [40] and constraint solving. Other approaches use the conceptual structures built by Formal Concept Analysis (FCA), a mathematical knowledge engineering framework, to benefit from their ability to encode variability in a systematic and canonical way [3, 14, 39, 51]. FCA framework supports a sound and complete extraction of the four prevalent feature relationships, and stands out among the other proposed methods in the literature. In fact, it is not a method specifically designed for variability extraction, but rather an identification of variability information naturally embedded in a unique and canonical structure whose construction relies on mathematical properties applied on the input software descriptions. FCA encompasses most of the aforementioned extraction methods, as well as their interim variability representations (e.g., binary implication graphs, mutex graphs, FDs) used for variability analysis, as the constructed structures contain the essence of variability [14]. The elements composing these structures express, as pointed out by Uta Priss, "*a natural feature of information representation which is as fundamental to hierarchies and object/attribute structures as set theory or relational algebra are for relational databases*" [49]. This explains the spreading of Formal Concept Analysis in knowledge engineering [47] or even software reverse engineering [59]. In what follows, we detail a sound and complete variability relationship extraction in the form of logical relationships based on the FCA framework and its associated conceptual structures.

#### ***6.2.4 A sound and complete FCA-based feature relationships extraction***

Formal Concept Analysis (FCA) [25] is a knowledge engineering framework for data structuring and knowledge representation. FCA enables the elaboration of concept hierarchies, where concepts group a maximal set of objects (or entities, documents) sharing a maximal set of attributes (or properties, descriptors). Applying this framework on a set of objects described by a set of attributes enables to build data structures organizing objects depending on the attributes they share.

**Input.** FCA input is a *formal context*  $K = (O, A, J)$  with  $O$  a set of objects,  $A$  a set of attributes and  $J \subseteq O \times A$  a binary relationship, where  $(o, a) \in J$  when " $o$  owns  $a$ ". A formal context may be represented by a binary table where a cross in a cell represents attribute (feature) ownership for an object (product). Table 6.3 shows a formal context with 7 existing Robocode bots (objects as rows) described by 11 features (attributes

as columns) taken from the RoboWiki<sup>2</sup>. It states for instance that the bot named Aristocles owns the one-on-one fighting feature, but not the melee fighting one. Formal contexts may conveniently represent configuration sets in an extensional way (e.g., the bot Centaur corresponds to the configuration {melee fighting, one-on-one fighting, roborumble}).

**Table 6.1** Formal context about 7 Robocode bots gathered from RoboWiki (2019)

Robocode	melee fighting	one-on-one fighting	license	closed	open-source	movement	min-risk-movement	random-movement	stop-and-go	wave-surfing	roborumble
<b>Aristocles</b>		×	×		×	×		×			×
<b>B26354</b>	×	×	×	×		×	×			×	
<b>Centaur</b>	×	×									×
<b>Coriantumr</b>	×	×	×		×	×					×
<b>Decado</b>		×	×		×	×		×	×		
<b>DrussGT</b>		×	×		×	×				×	×
<b>Durandal</b>	×	×	×	×		×				×	×

**Step 1 - building concepts.** A concept  $C = (E, I)$  associates a maximal group of objects  $E$ , with a maximal group of attributes  $I$  they share.  $E = \{o \in O \mid \forall a \in I, (o, a) \in J\}$  is the concept’s extent and  $I = \{a \in A \mid \forall o \in E, (o, a) \in J\}$  is the concept’s intent. For example, the concept gathering “all bots having open source license” is  $C_{os} = (E_{os}, I_{os})$  with  $I_{os} = \{\text{one-on-one fighting, license, open-source, movement}\}$  and  $E_{os} = \{\text{Aristocles, Coriantumr, Decado, DrussGT}\}$ . There is no other bot than the ones in  $E_{os}$  that share all attributes of  $I_{os}$ , and there is no other attribute than the ones in  $I_{os}$  shared by all bots of  $E_{os}$ .

**Step 2 - ordering concepts.** The specialization order  $\leq_{CL}$  between concepts is based on extent inclusion (and intent containment): for two concepts  $C_1 = (E_1, I_1)$  and  $C_2 = (E_2, I_2)$ ,  $C_1 \leq_{CL} C_2$  if and only if  $E_1 \subseteq E_2$  (and equivalently  $I_1 \supseteq I_2$ ). For example,  $C_{os}$  is a super-concept of concept  $C_{rand} = (E_{rand}, I_{rand})$ , with  $I_{rand} = \{\text{one-on-one fighting, license, open-source, movement, random movement}\}$  and  $E_{rand} = \{\text{Aristocles, Decado}\}$ , which is the concept gathering all bots implementing the random movement strategy. Indeed,  $E_{rand} \subseteq E_{os}$  and  $I_{os} \subseteq I_{rand}$ . This order enables to organize concepts in a specialization/generalization fashion.

**Output conceptual structures.** The *concept lattice* is the set of all the concepts  $C_K$  of the formal context  $K$ , provided with the order  $\leq_{CL}$ . In some applications, it is not necessary to study all the concepts that can be extracted: some of them may

<sup>2</sup> <http://robowiki.net/>, last access July 2019

be more valuable than others regarding the analysis to be applied. This is the case when investigating the variability information naturally embedded in concept lattices. To analyze variability information highlighted by means of FCA, it is enough to work with an output conceptual structure restricted to two types of concepts, namely *attribute-concepts* and *object-concepts*. An attribute-concept is the concept gathering all objects having a given attribute: we say that this attribute is introduced in this concept. For instance, the concept  $C_{os}$  is the attribute-concept introducing the attribute `open source`. An *object-concept* is the concept gathering the exact attribute set of a given object (i.e., its associated configuration): we say that this object is introduced in this concept. The concept  $C_{cent} = (E_{cent}, I_{cent})$  with  $E_{cent} = \{\text{Centaur, Coriantumur, Durandal}\}$  and  $I_{cent} = \{\text{melee fighting, one-on-one fighting, roborumble}\}$  (corresponding to the configuration of Centaur) is the object-concept introducing Centaur. The AOC-poset (for *Attribute-Object Concept partially ordered set*) is the concept lattice restricted to these specific concepts. Using AOC-posets instead of concept lattices brings a considerable improvement in terms of scalability. In fact, given  $|O|$  and  $|A|$  the numbers of objects and attributes, respectively, a concept lattice may have at most  $2^{\min(|O|, |A|)}$  concepts, whereas the AOC-poset is bounded by  $|O| + |A|$ , thus avoiding an eventual exponential growth of the output conceptual structures. A graphical representation of the AOC-poset associated with the formal context of Table 6.1 is shown in Fig. 6.2; it is built with the tool RCAExplore [22]. Note that for a given formal context, there exists only one concept lattice and only one AOC-poset (i.e., they are canonical structures).

Note that, if the products and/or features are numerous, the associated AOC-poset will likely be very wide, the extracted relationships too numerous, and the resulting model difficult to read. Experiments assessing the size of conceptual structures and the number of extracted relationships [14, 15] show that the method scales even with large inputs and outputs. However, the aforementioned issue is not inherent to FCA-based relationship extraction, but is related to the necessity to use separation of concerns to avoid synthesizing huge monolithic variability models. Relational Concept Analysis [27], another FCA extension that is not discussed here, provides solutions to extract variability relationships between several configuration sets, that can be obtained, for instance, by splitting a large configuration set depending on concerns. In this way, it may support the synthesis of several smaller interconnected variability models that enhance the legibility of represented variability information.

**Reading the AOC-poset.** In this graph-like representation, a concept is a 3-part box showing: the concept identifier (top-part), the concept intent (middle-part displaying attributes), and the concept extent (bottom-part displaying objects). The specialization order is represented by arrows between concepts: an arrow from a concept A (sub-concept) to a concept B (super-concept) states that “concept A specializes concept B”. Organizing concepts by specialization/generalization allows to represent factorized concepts’ intents and extents by displaying only the introduced elements: full intents and extents may be reconstituted by inheritance, from top to bottom for attributes (a concept possesses all attributes of its super-concepts) and from bottom to top for objects (a concept possesses all objects of its sub-concepts). In Fig. 6.2, the attribute-concept  $C_{os}$  introducing `open source` corresponds to

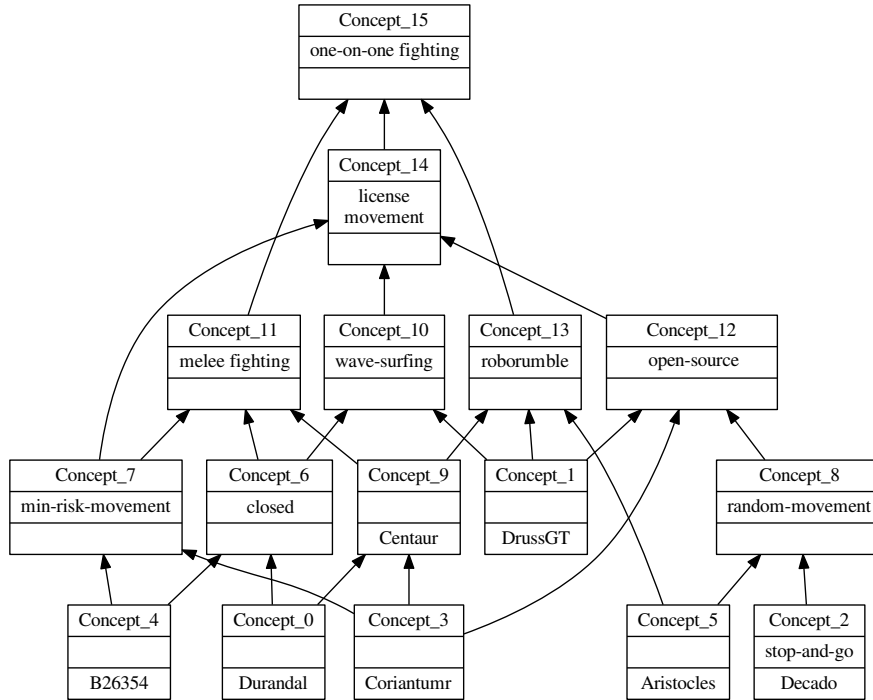


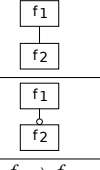
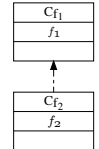
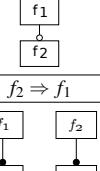
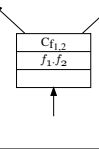
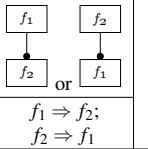
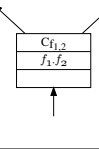
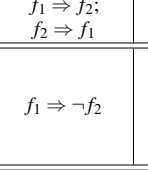
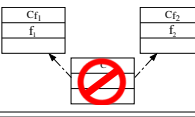
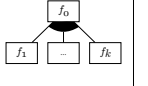
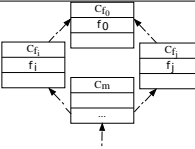
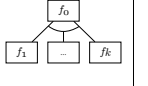
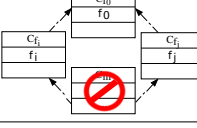
Fig. 6.2 AOC-poset associated with the formal context of Table 6.1

*Concept\_12*: attributes `one-on-one fighting`, `license` and `movement` are introduced in its super-concepts, and all objects of its extent are introduced in its sub-concepts.  $C_{rand}$  corresponds to *Concept\_8* and  $C_{cent}$  to *Concept\_9*.

**Extracting feature relationships.** By organizing objects depending on their attributes, conceptual structures naturally highlight the common and variable features amongst a set of configurations. By analyzing the structure, one can detect patterns corresponding to the feature relationships that hold in the input configuration set. Table 6.2 (columns 3 and 4) shows how the four prevalent feature relationships appear in the AOC-posets. We detail them in what follows, and we consider that  $f_i$ ,  $i \in \{0, 1, \dots, n\}$  denotes a feature and  $C_{f_i}$  the attribute-concept introducing  $f_i$ .

Situation ① (binary implications representing features that “require” other features) can be extracted from the AOC-poset by analyzing the specialization order between attribute-concepts. In fact, if an attribute-concept  $C_{f_2}$  is a sub-concept of another attribute-concept  $C_{f_1}$  (denoted by  $C_{f_2} \leq_{CL} C_{f_1}$ ), then the set of configurations having  $f_2$  is included in the set of configurations having  $f_1$  (because  $E_{f_2} \subseteq E_{f_1}$ ). Thus, all configurations having  $f_2$  also have  $f_1$  and  $(f_2 \Rightarrow f_1)$  holds. In Fig. 6.2, we can see for instance that `min-risk-movement`  $\Rightarrow$  `melee fighting` (because  $C_{concept\_7} \leq_{CL} C_{concept\_11}$ ) and `closed`  $\Rightarrow$  `license` (because  $C_{concept\_6} \leq_{CL}$

**Table 6.2** Mapping between the representations of the four prevalent feature relationships: propositional formulas (Prop. form.), feature diagrams (FD syntax) and AOC-posets adapted from [14]. The extent of a context  $C$  is denoted by  $Ext(C)$

Prop. form.	FD syntax	AOC-posets	
$\textcircled{1}$ $f_2 \Rightarrow f_1$		$C_{f_2} \leq_{CL} C_{f_1}$	
$f_2 \Rightarrow f_1$			
$\textcircled{1}'$ $f_1 \Leftrightarrow f_2$		$C_{f_1} =_{CL} C_{f_2}$	
$f_1 \Rightarrow f_2;$ $f_2 \Rightarrow f_1$			
$\textcircled{2}$ $f_1 \Rightarrow \neg f_2$ or $f_2 \Rightarrow \neg f_1$	$f_1 \Rightarrow \neg f_2$	$Ext(C_{f_1}) \cap Ext(C_{f_2}) = \emptyset$	
$\textcircled{3}$ $f_0 \Leftrightarrow (f_1 \vee \dots \vee f_k)$		$\forall f_i \in \{f_1, \dots, f_k\}, C_{f_i} \leq_{CL} C_{f_0}.$ $Ext(C_{f_1}) \cup \dots \cup Ext(C_{f_k}) = Ext(C_{f_0}).$ $C_{f_0} \notin \mathcal{O}^{\mathcal{C}}$	
$\textcircled{4}$ $f_0 \Leftrightarrow (f_1 \oplus \dots \oplus f_k)$		$\forall f_i \in \{f_1, \dots, f_k\}, C_{f_i} \leq_{CL} C_{f_0}.$ $Ext(C_{f_1}) \cup \dots \cup Ext(C_{f_k}) = Ext(C_{f_0}).$ $C_{f_0} \notin \mathcal{O}^{\mathcal{C}}$ $Ext(C_{f_1}) \cap \dots \cap Ext(C_{f_k}) = \emptyset.$	

*Concept\_14*). When such a pattern is detected in an AOC-poset built upon a configuration set, then we have candidates for refinement, optional or “require” cross-tree constraints during the synthesis of an FD. Deciding whether the minimum risk movement feature *requires* the fighting melee feature or *refines* it is left to the expert or to further processes relying on domain ontology. Situation  $\textcircled{1}'$  is a special case of situation  $\textcircled{1}$  where two features require each other. Because there is a double binary implication, then  $C_{f_2} \leq_{CL} C_{f_1}$  and  $C_{f_1} \leq_{CL} C_{f_2}$ , meaning that  $C_{f_1}$  and  $C_{f_2}$  are the same concept ( $C_{f_2} =_{CL} C_{f_1}$ ). Therefore, if two features are introduced in the same concept, then they are always present together in any configuration (co-occurring features) and  $(f_1 \Leftrightarrow f_2)$  holds. In Fig. 6.2, we observe that *license*  $\Leftrightarrow$  *movement* because they are both introduced in *Concept\_14*, meaning that if a bot defines movement strategies, it also specifies a license, and conversely.

Situation  $\textcircled{2}$  (mutually exclusive features) is revealed in the AOC-poset by analyzing the intersection of the extents of two attribute-concepts. If the intersection of  $C_{f_1}$  extent and  $C_{f_2}$  extent is empty (denoted by  $Ext(C_{f_1}) \cap Ext(C_{f_2}) = \emptyset$ ), then it means

that the configurations having  $f_1$  and the configurations having  $f_2$  are disjoint. In other words,  $f_1$  and  $f_2$  never appear together in any configuration and  $(f_1 \Rightarrow \neg f_2)$  holds. In Fig. 6.2, we can see that  $\text{wave-surfing} \Rightarrow \neg \text{random-movement}$  (because  $\text{Ext}(\text{Concept}_{10}) \cap \text{Ext}(\text{Concept}_8) = \emptyset$ ).

Situation ③ (“at least one” feature group refinement, also called or-groups) is more complex to identify. Let  $\{f_1, \dots, f_k\}$  be the features involved in an or-group, and  $f_0$  the parent-feature of this group. A first property characterizing or-groups is that each configuration having one feature in  $\{f_1, \dots, f_k\}$  should also have  $f_0$ . Thus, in the AOC-poset, the concepts  $\{C_{f_1}, \dots, C_{f_k}\}$  must be sub-concepts of  $C_{f_0}$ . A second or-group property is that each configuration possessing the parent-feature  $f_0$  must also have at least one feature of the or-group. This can be seen in the AOC-poset when the union of the extents of concepts  $\{C_{f_1}, \dots, C_{f_k}\}$  is equal to the extent of  $C_{f_0}$ . A “true” or-group would respect a third property, stating that all  $\{f_1, \dots, f_k\}$  combinations appear amongst the objects (configurations) in the AOC-poset. However, as we consider possibly incomplete configuration sets as input, having product descriptions documenting all possible combinations is very unlikely. Thus, we relax this third constraint and look for patterns corresponding to the two first or-group properties in the AOC-poset. In other words, we look for an attribute-concept and a set of its sub-concepts such that a) the latter are also attribute-concepts and b) the union of their extents is equal to the extent of the first attribute-concept. For example, let us consider in Fig. 6.2 *Concept\_7* (min-risk-movement), *Concept\_8* (random-movement) and *Concept\_10* (wave-surfing): the union of their extents is the configuration set corresponding to the set of bots having identified movement strategies (all bots except Centaur). They form a candidate or-group with parent *movement* introduced in their super-concept *Concept\_14*. Thus, ( $\text{movement} \Leftrightarrow (\text{min-risk-movement} \vee \text{wave-surfing} \vee \text{random-movement})$ ) holds. Notice that the candidate does not contain the feature *stop-and-go* (*Concept\_2*) but that it could be integrated in the group; the feature could also refine *random-movement* due to a complementary constraint establishing that *stop-and-go* implies *random-movement*. Notice also that *Concept\_12* (open source) may replace *Concept\_7* and satisfy the or-group properties: ( $\text{movement} \Leftrightarrow (\text{min-risk-movement} \vee \text{wave-surfing} \vee \text{open source})$ ) holds too. The candidates or-groups may be numerous and need an ontological evaluation in order to detect the meaningful ones. Other constraints may be added to help select the best candidates, e.g., set of candidates that do not overlap, minimal groups.

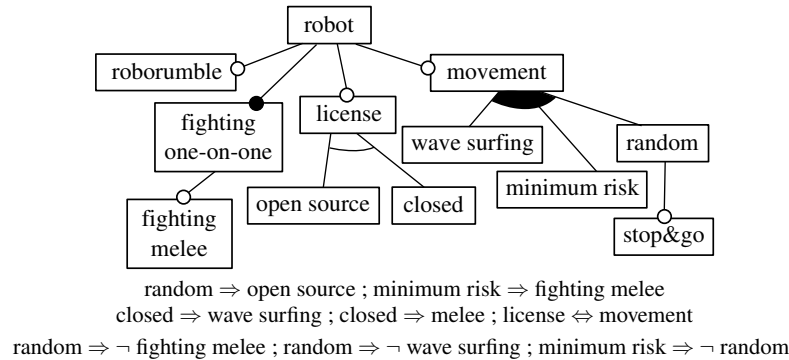
Situation ④ (“exactly one” feature group refinement, also called xor-group) is a particular sub-case of the situation ③ where all pairs of features of  $\{f_1, \dots, f_k\}$  are mutually exclusive. In addition to the two aforementioned or-group properties, concepts  $\{C_{f_1}, \dots, C_{f_k}\}$  must have disjoint extents. In Fig. 6.2, *Concept\_14* (license) with its sub-concepts *Concept\_6* (closed) and *Concept\_12* (open source) form a xor-group candidate: ( $\text{license} \Leftrightarrow (\text{closed} \oplus \text{open source})$ ) holds.

By analyzing concepts and how they are situated to each other, we can identify variability information that can be represented by propositional formulas over the set of features. Lattice theory guarantees that all detected relationships are true in the considered input, and that all true relationships can be read in FCA conceptual

structures [25]. This enables the sound and complete extraction of all prevalent feature relationships from a configuration set; variability model synthesis methods using these relationships can thus guarantee a logical semantics and a configuration semantics consistent with the input descriptions.

**Feature diagram synthesis.** Thanks to the mapping established in Table 6.2, it is possible to guide an expert during a feature diagram synthesis, a task that can be seen as choosing an FD syntax for each feature relationship identified in the AOC-poset. Figure 6.3 presents an example of FD that can be extracted from the configurations gathered in Table 6.3 by building and analyzing the associated AOC-poset. It differs from the one exposed in Fig. 6.1 because it is based on existing configurations and ontological choices corresponding to feature relationship patterns identified in the AOC-poset. For instance, Table 6.3 does not contain any *fighting* feature stating if the bot has at least one identified fighting styles, but it does contain two more specific fighting styles: thus, *fighting one-on-one* and *fighting melee* are not grouped in Fig. 6.3. Moreover, in the studied set of Robocode configurations, all bots are defined for one-on-one fighting style: the corresponding feature may be identified as mandatory in this case. Also, as feature *stop&go* implies *random*, we decided to put it as an optional feature of *random*, but we could have chosen to add *stop&go* in the or-group and to add a “require” cross-tree constraint. Some extracted constraints may be seen as “accidental”, i.e., true for the considered set of configurations but not for the domain. For instance, the constraint  $\text{closed} \Rightarrow \text{wave surfing}$ , stating that if the bot’s source code is proprietary, then it must implement the wave surfing movement strategy, is likely to be coincidental. Note that, because FD syntax cannot represent all configuration sets, the configuration semantics of this FD is not exactly the same as the one presented in the input Table 6.1. However, as the extraction of the FD logical relationships is sound and complete, the configuration semantics of the resulting model is as close as possible as the input configuration set. If the domain expert building the FD from the extracted logical relationships considers that some of them should not appear in the graphical part of the model (e.g., for pertinence or legibility concerns), they can be kept as complex cross-tree constraints to preserve a consistent configuration semantics.

**Other variability information contained in AOC-posets.** AOC-posets and more generally conceptual structures highlight different types of information which are useful regarding variability management in general. If we look at the intent of each concept (i.e., the middle part), these sub-sets of features actually represent either a valid configuration (if the concept is an object-concept) or a partial configuration, i.e., a sub-set of features shared by several valid configurations. This information may be useful for feeding decision processes, e.g., to guide a user when choosing a product configuration, or to imagine new possible configurations which can be easy to build from existing artifacts. Also, if we analyze the place of the concepts in the structure, it may give information on the usage and distribution of features in the configuration set. For instance, if a feature is introduced in a concept in the top of the structure, then it is more likely to be inherited by numerous concepts, and therefore to be commonly found in configurations. Conversely, if a feature is introduced in the bottom of the structure, then it will be present in possibly less configurations and could be identified



**Fig. 6.3** Example of FD synthesized from the AOC-poset built upon Robocode configurations of Table 6.3

as a specific or unpopular feature. Conceptual structures result from the application of a data analysis framework: their construction is *structural*; they are not built for variability analysis, but by following some mathematical properties that naturally highlight the intrinsic variability of a configuration set. They offer a unique dual view of variability, by organizing both features and configurations in one structure, that makes them a suitable representation to study the variability of existing products developed without any reuse strategy. Besides, FCA conceptual structures are not only interim representations of variability: they come with a framework relying on strong mathematical foundations, a set of management operations and numerous extensions enabling to take into account more complex input than boolean datasets.

### 6.3 Variability in non-boolean descriptions

In the previous section, we focused on variability relationships in the form of constraints over a set of features, and how to extract them from boolean descriptions. However, representing product line variability through boolean features and feature diagrams has shown some limits regarding expressiveness. This is even more the case with the increasing complexity of software intensive systems, which give birth to new issues about complex product lines and complex variability modeling [32]. In this section, we study more expressive variability relationships (used in FD extensions) which may be extracted from multi-valued descriptions. We first introduce product descriptions containing multi-valued characteristics (Section 6.3.1). These non-boolean descriptions necessitate a more complex modeling framework than the one previously presented for boolean feature sets in order to fully apprehend their intrinsic variability. Then, we present three prevalent FD extensions that seek to enhance the expressiveness of traditional boolean models, and we identify the new variability relationships introduced by these extensions (Section 6.3.2). We also discuss other types of variability models that may take into account more than boolean features.



Then, we survey main directions in non-boolean variability extraction (Section 6.3.3) before discussing how to extend FCA-based variability extraction to take into account the expressive variability relationships (Section 6.3.4).

### 6.3.1 Multi-valued descriptions

Software descriptions may include boolean features but also more complex information: characterizing products using multi-valued attributes such as numerical ones, or with symbolic values enhanced with metadata is a common practice. Product comparison matrices (PCMs) are representative of such complex product configurations [46, 53]. PCMs describe product properties in a tabular form where rows show the product configurations, columns show the product characteristics and cells contain values. A PCM characteristic can be associated with a type (that may be boolean, numerical or symbolic) and a scope for its corresponding set of possible values. Examples of such PCMs used in the product line community include Wikipedia PCMs<sup>3</sup> [53], gathered descriptions of variants generated with JHipster v3.6.1<sup>4</sup> [28], and the Robocode descriptions<sup>5</sup> [43] used here. Table 6.3 is a multi-valued matrix (PCM) that extends and adapts the previous boolean example about Robocode. Features (boolean characteristics) are shown in the first six columns. Movement strategies now are described in one column representing a multi-valued attribute with symbolic values. The last column indicates numerical values for the roborumble PWIN (probability of win). When a product possesses several values for one characteristic, they are split by a coma; the symbol ‘\*’ states that there is no known value.

Modeling complex variability necessitates to express more than only *feature relationships*: we now consider *multi-valued relationships* that should involve boolean attributes (features) as well as multi-valued attributes.

### 6.3.2 Multi-valued variability relationships

In the same way as the literature identified feature relationships corresponding to the prevalent variability information, we identify the multi-valued relationships necessary to model more complex variability. This is done through the analysis of the three FD extensions that seek to enhance the boolean case expressiveness, as we consider that these extensions represent the designers’ needs in terms of non-boolean variability.

**FD with attributes.** A first extension associates *multi-valued attributes* to features [16]. The attribute has a type, such as string, enumeration or integer. Attributes enable to define more detailed information without making the model more complex. In fact,

<sup>3</sup> [https://en.wikipedia.org/wiki/Category:Software\\_comparisons](https://en.wikipedia.org/wiki/Category:Software_comparisons), last accessed July 2019

<sup>4</sup> <https://github.com/xdevroey/jhipster-dataset/tree/master/v3.6.1>

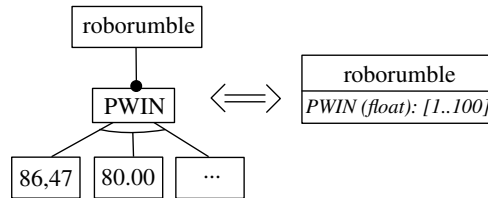
<sup>5</sup> [https://github.com/but4reuse/RobocodeSPL\\_teaching](https://github.com/but4reuse/RobocodeSPL_teaching)

**Table 6.3** PCM about the 7 previous robots of Robocode Wiki (2018) with their roborumble PWIN. Some boolean attributes (features) about movement strategy were transformed into one multi-valued attribute having symbolic values

Robocode	one-on-one fighting	melee fighting	roborumble	license	open-source	closed	movement	PWIN
<b>Aristocles</b>	×		×	×	×		random	86.47
<b>B26354</b>	×	×		×		×	minimum risk, wave surfing	*
<b>Centaur</b>	×	×	×				*	80.00
<b>Coriantumr</b>	×	×	×	×	×		minimum risk	77.24
<b>Decado</b>	×			×	×		random, stop & go	*
<b>DrussGT</b>	×		×	×	×		wave surfing	100.00
<b>Durandal</b>	×	×	×	×		×	wave surfing	79.48

modeling the same variability information with only boolean features would end up with a large model more difficult to read, as illustrated in Fig. 6.4.

With this extension, we can write (in addition to feature relationships) binary implications, co-occurrences and mutex either between a feature and an attribute value, or between two attribute values (denoted by `attribute:value`). As or- and xor-groups represent feature refinements, attribute values cannot be involved in such groups.



**Fig. 6.4** Representing detailed information without making the model more complex by using attributes: (left) representation of PWIN values with features only, (right) representation of PWIN values with a multi-valued attribute

**FD with feature cardinalities.** A second extension introduces UML-like cardinalities on features [18], stating that a feature may occur several times in a configuration and defining the minimum and maximum number of these occurrences. A number of occurrences associated with a feature may be seen as an attribute with integer values: the variability relationships necessary to represent this extension are the same as for FDs with attributes.

**FD with group cardinality.** The third extension proposes to refine feature groups with *feature-group cardinality* in the form  $\langle min - max \rangle$  indicating that a product can contain between *min* and *max* child features in the group. The boolean FD notations for xor-groups and or-groups are respectively replaced by  $\langle 1 - 1 \rangle$  groups and  $\langle 1 - n \rangle$  groups. Thanks to this extension, feature relationships ③ and ④ exposed at the beginning of Section 6.2.2 may be merged in one kind of more abstract feature relationship, stating that a feature should be refined by “between *min* and *max*” features in a group. Instead of representing the logical semantics of or- and xor-groups with  $\vee$ -connectives and  $\oplus$ -connectives respectively, one may now specify which combinations of features from a group are allowed by the cardinality. For instance, a group with parent  $p$ , a cardinality  $\langle 2 - 3 \rangle$  and children  $\{f_1, f_2, f_3\}$  is represented through the following formula:

$$p \Leftrightarrow ((f_1 \wedge f_2 \wedge \neg f_3) \vee (f_1 \wedge f_3 \wedge \neg f_2) \vee (f_2 \wedge f_3 \wedge \neg f_1) \vee (f_1 \wedge f_2 \wedge f_3))$$

To consider these three extensions, it is necessary to extend the previous four feature relationships so that binary implications, co-occurrences and mutual exclusions can involve attribute values as well as features. Figure 6.5 presents the grammar of these variability relationships. We simplify the logical relationships representing feature-groups by introducing the notation  $(p, \{f_1, \dots, f_n\}, \langle min - max \rangle)$  to be used instead of the one introduced before.

---

<i>variability relationships</i>	:= <i>relationship</i> *
<i>relationship</i>	:= <i>binary implication</i>   <i>co-occurrence</i>   <i>mutex</i>   <i>group</i>
<i>binary implication</i>	:= <i>element</i> ‘ $\Rightarrow$ ’ <i>element</i>
<i>co-occurrence</i>	:= <i>element</i> ‘ $\Leftrightarrow$ ’ <i>element</i>
<i>mutex</i>	:= <i>element</i> ‘ $\Rightarrow$ ’ ‘ $\neg$ ’ <i>element</i>
<i>group</i>	:= ‘(’ <i>feature</i> ‘,’ ‘{’ <i>feature_set</i> ‘}’ ‘,’ <i>cardinality</i> ‘)’
<i>feature_set</i>	:= <i>feature</i>   <i>feature_set</i> ‘,’ <i>feature</i>
<i>element</i>	:= <i>feature</i>   <i>attribute</i>
<i>feature</i>	:= <b>feature_name</b>
<i>attribute</i>	:= <b>attribute_name: value</b>
<i>cardinality</i>	:= ‘(’ <b>nb_min</b> ‘-’ <b>nb_max</b> ‘)’

---

**Fig. 6.5** Grammar of variability relationships for representing FD extensions

**Other formalisms for non-boolean variability.** We survey a few alternative formalisms for complex variability description without pretending to be exhaustive. A survey on variability modeling in Cyber-Physical Systems (CPSs) and pointers to other surveys can be found in [52]. Decision Models (DMs) are textual descriptions focusing only on variability decisions [17, 55]. They are organized in tabular form, where each row includes a question (such as “*has PWIN?*”), the type of the expected answer (such as *Integer*) and its range (such as  $[0, 100]$ ). It may also present constraints about the required form of the answer (e.g., cardinality or implication) and conditions on when to consider the question. Variability relationships expressed by such models

are the same as the three aforementioned FD extensions. An Orthogonal Variability Model (OVM) also focuses on variability only [48]. It introduces variation points (e.g. *movement*) and their variants (e.g. *random*, *wave surfing*). Variation points can be optional or mandatory. Optional variants may be part of a group with a cardinality. Therefore, the variability relationships of Fig. 6.5 are sufficient to represent OVM variability. Common Variability Modeling (CVL) [31] proposes three interrelated models: base model (UML model or any MOF based Domain Specific Language model), variability model, and resolution model (for selection). It allows the description of cardinalities on features and groups and the introduction of attributes (under the name of *variable*). CVL models rely on separation of concerns to split one variability model in several smaller models referencing each other: this notion of references is not taken into account in Fig. 6.5. Constraint Satisfaction Problem [54] and Constraint Logic Programming [35] also have been used to consider non-boolean descriptions. Variability relationships of Fig. 6.5 may be used to build such models, but the models may represent more complex constraints as the one studied here.

### 6.3.3 *Multi-valued variability extraction in the literature*

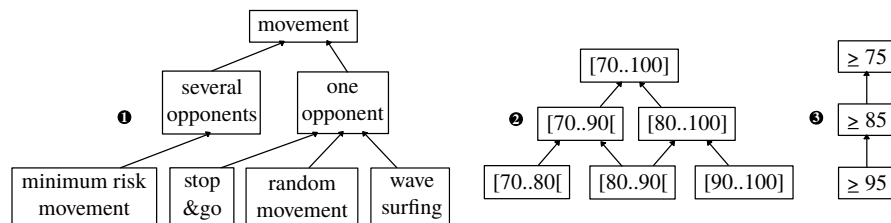
Becan et al. [9] and Carbonnel et al. [15] both extract variability relationships involving features and attributes. Becan et al. [9] propose dedicated algorithms to extract attributed feature models. They start from configuration matrices: they compute feature groups, implications between features and mutually exclusive features; then they extract all implications involving an attribute value. They use different structures: a binary implication graph and a mutex-graph. Carbonnel et al. [15], in addition, compute mutex between features and/or attribute values, and co-occurrences between features and/or attributes. They use a single structure, which is a conceptual structure and do not synthesize an attributed feature model. This approach is detailed in the next section. The MoVa2PL approach [44] extracts dependencies as well as *requires* and *exclude* constraints from feature spanning over several model variants graph decomposition (including cardinalities), and may define CVL compliant models. To the best of our knowledge, there is no approach for extracting OVM models from product descriptions. Constraint satisfaction problem acquisition [12] is a promising field which should be investigated and connected to complex variability extraction in the future.

### 6.3.4 *A sound and complete FCA-based multi-valued variability extraction*

As real datasets are often more complex than boolean descriptions, many extensions have been proposed over the years for the FCA data analysis framework. To process multi-valued attributes, we can mention the scaling of numerical data [25], the use

of value taxonomies [26], and the more general and complete framework extension called Pattern Structures [24]. These approaches may overlap and have common theoretical foundations that may lead to some mappings between them. Dealing with a PCM information as the one presented in Table 6.3 (called a *multi-valued context* in FCA) can be done with several of these frameworks. In this chapter, we choose to explore the approach based on value taxonomies, for pedagogical purposes and to avoid introducing a full complex data analysis framework such as Pattern Structures.

**Value taxonomies.** The key principle of the value taxonomy approach is the elaboration of taxonomies (partial orders, or scales) on the attribute values. This task may be summarized by “establishing, naming and organizing groups of values in a hierarchy”. The goal of this approach is to group the different values of a given attribute under more abstract values; in this way, relationships may be established not only on the attribute values present on the multi-valued descriptions, but also between groups of these values. Value taxonomies may come from different sources depending what the variability analysis aims to highlight. They may be based on *is-a* relationships for symbolic values, or extracted from external resources such as ontologies or other documents. For example, if we consider Robocode movement strategies, one may introduce the new value `several opponents`, that is more abstract than the ones introduced in Table 6.3 and encompasses all strategies suited for dealing with several bots during a fight. When studying the RoboWiki, one may determine that `minimum risk movement` belongs to this group, while `random movement`, `wave surfing` and `stop&go` rather belong to a group that could be identified by the value `one opponent`. Alternatively they could be organized depending on the complexity of their implementation algorithms, or by other movement strategies they were inspired from. Movement strategies grouped by the number of opponents against which they are more efficient are presented in Fig. 6.6 ❶. It states that a bot which implements the `wave surfing` strategy (third level) thus implements a strategy suited for fighting `one opponent` (second level) and thus implement a movement strategy (first level).



**Fig. 6.6** Examples of a taxonomy for the symbolic values of the attribute `movement`, and two taxonomies for the numerical values of the attribute `PWIN`

Taxonomies for numerical values correspond to strict or partial ordering of the values. Several schemes have been studied in FCA for building these orders [25]. In nominal scaling, each value  $v$  of a multi-valued attribute  $m$  is converted to an

attribute  $m:v$ . It is used in our example for `movement` values, generating 5 columns labeled with the prefix “`movement :`”. Nominal scaling may be seen as an approach to apply by default when no groups of values may be defined or retrieved. In ordinal scaling, the ordinary number order is used. Values of a multi-valued attribute  $m$  are described by expressions of the form  $m > n$ . It is shown in Fig. 6.6 ③ for `PWIN` values. The figure indicates that a value described by attribute `PWIN  $\geq$  95q` is also described by attribute `PWIN  $\geq$  85` and by attribute `PWIN  $\geq$  75`. Notice that, to elaborate this scale, design choices are required to define the bounds that serve in the description. Here, 75, 85 and 95 have been chosen. Figure 6.6 ② describes another type of scale on `PWIN` values. It is based on a description of values by their membership to intervals that split the value set. Here again, the interval bounds have to be determined by domain experts or using techniques like box-plot [36]. Moreover, the intervals are grouped and organized through a hierarchical structure in order to enrich the description and allow more similarities to be discovered between values. For example 75 and 85 cannot be grouped with using the bottom intervals, but they can be grouped using the level 2 interval  $[70..90]$ . Other kinds of scales can be found in [25, 36]. In general, any similarity/distance relationships between values both symbolic or numerical, can be used to form a scale.

**Binary conversion.** These value taxonomies may be used to convert the multi-valued context in a binary context. In this way, theoretical properties and algorithms of the standard FCA scheme (as presented in Section 6.2.4) apply on the transformed dataset. To achieve the binary conversion, each element from the scale (i.e., each value in the taxonomy) becomes a boolean attribute of the final binary context. Compared to a naive solution which does not use value taxonomies and where each attribute value is transformed in a boolean feature, this allows to extract additional relationships taking into account groups of attributes values. Let us choose the scale ① for values of `movement` and the scale ③ for values of `PWIN`. Then, the boolean attributes of Table 6.3 along with the values of attributes `movement` and `PWIN` enhanced with the taxonomy values form the boolean attributes of the equivalent binary context of Table. 6.4; its associated AOC-poset is presented in Fig. 6.7.

Rows of the final context still correspond to the product configurations from the multi-valued context shown Table 6.3. A relation between a product configuration  $c$  and an attribute  $a$  is then established (and shown with a cross in the table) depending on the attribute kind. For a boolean attribute  $a$ ,  $c$  owns  $a$  in the final context when  $c$  owns  $a$  in the multi-valued context. This is illustrated by the 6 first columns (Table 6.4). For a nominal scale,  $c$  owns  $m:v$  in the final context when  $c$  owns  $m$  with value  $v$  in the multi-valued context. This is illustrated by columns 7-11 for `movement`. For an ordinal scale,  $c$  owns the attribute  $m > n$  in the final context when  $c$  owns for  $m$  a value  $v > n$  in the multi-valued context. The last 4 columns illustrate the ordinal scale for `PWIN`. A  $(c, a)$  relation in this scheme verifies the following general property: if we have  $(c, a_l)$  and  $a_g$  more general than  $a_l$  in the taxonomy,  $(c, a_g)$  is also true. This applies to all value taxonomies, beyond the examples we have shown in this section.

**Reading multi-valued relationships in the AOC-poset.** The AOC-poset can be interpreted as in the boolean case. As a consequence of the value taxonomies, a concept introducing an attribute  $a_g$ , with  $a_g$  more general than another attribute  $a_l$  in

**Table 6.4** Formal context obtained after applying binary conversion to the multi-valued context of Table 6.3

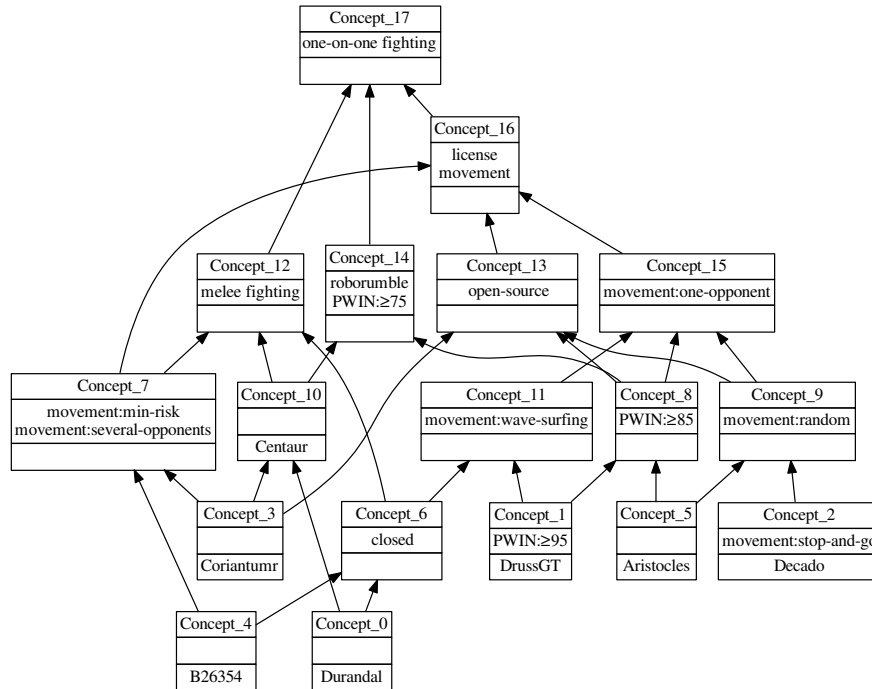
Robocode	one-on-one fighting	melee fighting	roborumble	license	open-source	closed	movement:random	movement:min-risk	movement:wave-surfing	movement:stop-and-go	PWIN: $\geq$ 95	PWIN: $\geq$ 85	PWIN: $\geq$ 75	movement	movement:one-opponent	movement:several-opponents
<b>Aristocles</b>	x		x	x	x		x					x	x	x	x	
<b>B26354</b>	x	x		x		x		x	x					x	x	x
<b>Centaur</b>	x	x	x										x			
<b>Coriantumr</b>	x	x	x	x	x			x					x	x		x
<b>Decado</b>	x			x	x		x		x					x	x	
<b>DrussGT</b>	x		x	x	x				x		x	x	x	x	x	
<b>Durandal</b>	x	x	x	x		x			x				x	x	x	

the taxonomy, is a super-concept of the concept introducing  $a_l$  (if one exists). For example, in Fig. 6.7, *Concept\_15*, introducing  $PWIN \geq 75$ , is a super-concept of *Concept\_10* which introduces  $PWIN \geq 85$ . Therefore, the taxonomies are included in the AOC-poset.

The mapping between propositional formulas and the AOC-poset still applies in this context. For example, applying the rule from Table 6.2, row ①, we can deduce  $PWIN \geq 85 \Rightarrow \text{open-source}$  because *Concept\_10* introduces  $PWIN \geq 85$  and its super-concept *Concept\_14* introduces *open-source*. From Table 6.2, row ②, as *Concept\_11* (introducing *movement:random*) and *Concept\_11* (introducing  $PWIN \geq 95$ ) have disjoint extents, we can deduce  $\text{movement:random} \Rightarrow \neg PWIN \geq 95$ .

Group cardinalities may be extracted by analyzing the intent of the sub-concepts of the concept introducing the parent of the group. For instance, by analyzing the sub-concepts of *Concept\_16* introducing *movement*, one may notice that a bot never possesses more than two initial values for the attribute *movement*, hence suggesting a  $\langle 1 - 2 \rangle$  cardinality.

In the FD extended with multi-valued attributes, each attribute is associated with a feature of the FD. In the AOC-poset, when the most general value of a value taxonomy (i.e., group representing all values of an attribute) implies a feature, then we can deduce that the attribute corresponding to the value taxonomy may be associated to this feature. For instance, in *Concept\_14* of Fig 6.7, the element representing all values of *PWIN* (i.e.,  $PWIN \geq 75$ ) is co-occurrent with the feature *roborumble*, suggesting to associate the attribute *PWIN* to this feature.



**Fig. 6.7** AOC-poset associated with the context of Table 6.4

The extracted logical relationships may be quite numerous due to the sound and complete extraction method based on potentially incomplete input descriptions [15]. However, redundancy elimination techniques based on grouped attribute values introduced through taxonomies may be applied to reduce their number without losing information. For instance, one can extract the two binary implications  $PWIN: \geq 95 \Rightarrow \text{movement:one opponent}$  and  $PWIN: \geq 85 \Rightarrow \text{movement:one opponent}$ , but only the second may be kept as the first one is included in it. Depending on whether the input descriptions possess numerous features or attributes having values which can be easily organized in taxonomies, redundancy elimination may reduce up to 50% of some types of variability relationships. Redundancy elimination combined with filtering methods allowing the expert to ignore or focus on relationships between given features and/or attributes may help their analysis. A previous study shows that a filtering method as simple as removing relationships between features and/or attributes considered unrelated by the expert reduces their number from 30% to 65%.

FCA is a structural framework which naturally highlights variability and that can be extended to take into account more complex input descriptions. We have seen how value taxonomies may help handle multi-valued descriptions more efficiently and extract multi-valued variability relationships represented by extended FDs. The more



complete FCA extension called Pattern Structure generalizes the taxonomy approach to take into account any description on which similarities may be defined. In this way, one can imagine handling more complex artifacts such as slices of FDs, or versioning data. Other extensions such as Relational Concept Analysis [27] can be investigated to extend the extracted variability relationships and build interconnected variability models such as FDs with references or CVL models [13].

## 6.4 Conclusion

In this chapter, we focused on the process of extracting variability information in the form of logical relationships from product descriptions. More specifically, we tackled the concern of extracting complex variability relationships from non-boolean descriptions. To do so, we extend existing extraction methods focusing on the traditional boolean case to complex variability representations. Thus, this chapter was divided in two main symmetric parts: the first part discussed the traditional boolean case, and the second part examined the extraction in a more complex case.

In the first part, we presented the boolean product descriptions as defined in feature-oriented product line approaches. Then, we summarized the four main studied feature relationships (i.e., variability relationships based on boolean descriptions) to express variability in terms of features. After that, we reviewed methods found in the literature that seek to extract these feature relationships. As revealed by a study on feature model synthesis, methods that first focus on extracting the logical foundation of descriptions' intrinsic variability before relying on experts to breath ontological meaning to them outperform the other methods. Thus, we directed attention to Formal Concept Analysis, a knowledge engineering framework for knowledge representation and extraction, as it includes and formalizes existing methods extracting variability information in the form of logical relationships. We then presented a sound and complete extraction method based on Formal Concept Analysis and its associated conceptual structures. An advantage of using this framework in the boolean case is that it is extensible, and thus may be applied to more complex datasets (i.e., not necessarily boolean).

In the second part, we focused on multi-valued input descriptions, i.e., representing software products by both boolean features and multi-valued characteristics. As traditional feature relationships are not sufficient to capture the intrinsic variability of this type of multi-valued descriptions, we identified new variability relationships which take into account multi-valued characteristics. For that, we analyzed the three FD extensions that were proposed to enhance the expressiveness of the feature-based variability representations. Then, we discussed the few existing methods that tackle this type of extraction, and we show how Formal Concept Analysis may be extended to be applied in the multi-valued case. We relied on the definition of value taxonomies (i.e., grouping values under more abstract values) for non-boolean attributes, which enable the extraction of more precise variability relationships and the definition of redundancy elimination techniques. We mentioned other Formal Concept Analysis

extensions that may be useful to go further in the problem of complex variability extraction, e.g., by relying on more complex descriptions or even interconnected ones.

## References

1. Acher, M., Baudry, B., Heymans, P., Cleve, A., Hainaut, J.: Support for reverse engineering and maintaining feature models. In: The Seventh International Workshop on Variability Modelling of Software-intensive Systems, VaMoS '13, Pisa, Italy, January 23 - 25, 2013, pp. 20:1–20:8 (2013)
2. Acher, M., Cleve, A., Perrouin, G., Heymans, P., Vanbeneden, C., Collet, P., Lahire, P.: On extracting feature models from product descriptions. In: Sixth International Workshop on Variability Modelling of Software-Intensive Systems, Leipzig, Germany, January 25-27, 2012. Proceedings, pp. 45–54 (2012)
3. Al-Msie'deen, R., Huchard, M., Seriai, A., Urtado, C., Vauttier, S.: Reverse engineering feature models from software configurations using formal concept analysis. In: Proceedings of the Eleventh International Conference on Concept Lattices and Their Applications, Košice, Slovakia, October 7-10, 2014., pp. 95–106 (2014)
4. Apel, S., Batory, D.S., Kästner, C., Saake, G.: Feature-Oriented Software Product Lines - Concepts and Implementation. Springer (2013)
5. Assunção, W.K.G., Lopez-Herrejon, R.E., Linsbauer, L., Vergilio, S.R., Egyed, A.: Multi-objective reverse engineering of variability-safe feature models based on code dependencies of system variants. *Empirical Software Engineering* **22**(4), 1763–1794 (2017)
6. Assunção, W.K.G., Lopez-Herrejon, R.E., Linsbauer, L., Vergilio, S.R., Egyed, A.: Reengineering legacy applications into software product lines: a systematic mapping. *Empirical Software Engineering* **22**(6), 2972–3016 (2017)
7. Batory, D.S.: Feature models, grammars, and propositional formulas. In: Software Product Lines, 9th International Conference, SPLC 2005, Rennes, France, September 26-29, 2005, Proceedings, pp. 7–20 (2005)
8. Bécan, G., Acher, M., Baudry, B., Nasr, S.B.: Breathing ontological knowledge into feature model synthesis: an empirical study. *Empirical Software Engineering* **21**(4), 1794–1841 (2016)
9. Bécan, G., Behjati, R., Gotlieb, A., Acher, M.: Synthesis of attributed feature models from product descriptions. In: Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20-24, 2015, pp. 1–10 (2015)
10. Benavides, D., Segura, S., Cortés, A.R.: Automated analysis of feature models 20 years later: A literature review. *Inf. Syst.* **35**(6), 615–636 (2010)
11. Berger, T., Rublack, R., Nair, D., Atlee, J.M., Becker, M., Czarnecki, K., Wasowski, A.: A survey of variability modeling in industrial practice. In: The Seventh International Workshop on Variability Modelling of Software-intensive Systems, VaMoS '13, Pisa, Italy, January 23 - 25, 2013, pp. 7:1–7:8 (2013)
12. Bessiere, C., Daoudi, A., Hebrard, E., Katsirelos, G., Lazaar, N., Mechqrane, Y., Narodytska, N., Quimper, C., Walsh, T.: New approaches to constraint acquisition. In: Data Mining and Constraint Programming - Foundations of a Cross-Disciplinary Approach, pp. 51–76 (2016)
13. Carbonnel, J., Huchard, M., Nebut, C.: Exploring the Variability of Interconnected Product Families with Relational Concept Analysis. In: Proceedings of 7th International Workshop on Reverse Variability Engineering (REVE 2019) @ SPLC (2019)
14. Carbonnel, J., Huchard, M., Nebut, C.: Modelling equivalence classes of feature models with concept lattices to assist their extraction from product descriptions. *Journal of Systems and Software* **152**, 1–23 (2019)
15. Carbonnel, J., Huchard, M., Nebut, C.: Towards Complex Product Line Variability Modelling: Mining Relationships from Non-Boolean Descriptions. *Journal of Systems and Software* (2019)

16. Czarnecki, K., Eisenecker, U.W.: Generative programming - methods, tools and applications. Addison-Wesley (2000)
17. Czarnecki, K., Grünbacher, P., Rabiser, R., Schmid, K., Wasowski, A.: Cool features and tough decisions: a comparison of variability modeling approaches. In: Sixth International Workshop on Variability Modelling of Software-Intensive Systems, Leipzig, Germany, January 25-27, 2012. Proceedings, pp. 173–182 (2012)
18. Czarnecki, K., Helsen, S., Eisenecker, U.W.: Staged configuration using feature models. In: Software Product Lines, Third International Conference, SPLC 2004, Boston, MA, USA, August 30-September 2, 2004, Proceedings, pp. 266–283 (2004)
19. Czarnecki, K., She, S., Wasowski, A.: Sample spaces and feature models: There and back again. In: Proceedings of the 12th International Conference on Software Product Lines (SPLC'08), pp. 22–31 (2008)
20. Czarnecki, K., Wasowski, A.: Feature diagrams and logics: There and back again. In: Software Product Lines, 11th International Conference, SPLC 2007, Kyoto, Japan, September 10-14, 2007, Proceedings, pp. 23–34 (2007)
21. Davril, J., Delfosse, E., Hariri, N., Acher, M., Cleland-Huang, J., Heymans, P.: 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, Saint Petersburg, Russian Federation, August 18-26, 2013, pp. 290–300 (2013)
22. Dolques, X., Braud, A., Huchard, M., Ber, F.L.: Rcaexplore, a FCA based tool to explore relational data. In: Supplementary Proceedings of ICFA 2019 Conference and Workshops, Frankfurt, Germany, June 25-28, 2019., pp. 55–59 (2019)
23. Ferrari, A., Spagnolo, G.O., Gnesi, S., Dell'Orletta, F.: CMT and FDE: tools to bridge the gap between natural language documents and feature diagrams. In: Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20-24, 2015, pp. 402–410 (2015)
24. Ganter, B., Kuznetsov, S.O.: Pattern structures and their projections. In: Conceptual Structures: Broadening the Base, 9th International Conference on Conceptual Structures, ICCS 2001, Stanford, CA, USA, July 30-August 3, 2001, Proceedings, pp. 129–142 (2001)
25. Ganter, B., Wille, R.: Formal Concept Analysis - Mathematical Foundations. Springer (1999)
26. Godin, R., Mili, H.: Building and maintaining analysis-level class hierarchies using Galois lattices. In: Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), Eighth Annual Conference, Washington, DC, USA, September 26 - October 1, 1993, Proceedings., pp. 394–410 (1993)
27. Hacene, M.R., Huchard, M., Napoli, A., Valtchev, P.: Relational concept analysis: mining concept lattices from multi-relational data. *Ann. Math. Artif. Intell.* **67**(1), 81–108 (2013)
28. Halin, A., Nuttinck, A., Acher, M., Devroey, X., Perrouin, G., Heymans, P.: Yo variability! jhipster: a playground for web-apps analyses. In: Proceedings of the Eleventh International Workshop on Variability Modelling of Software-intensive Systems, VaMoS 2017, Eindhoven, Netherlands, February 1-3, 2017, pp. 44–51 (2017)
29. Haslinger, E.N., Lopez-Herrejon, R.E., Egyed, A.: Reverse engineering feature models from programs' feature sets. In: 18th Working Conference on Reverse Engineering, WCRE 2011, Limerick, Ireland, October 17-20, 2011, pp. 308–312 (2011)
30. Haslinger, E.N., Lopez-Herrejon, R.E., Egyed, A.: On extracting feature models from sets of valid feature combinations. In: Fundamental Approaches to Software Engineering - 16th International Conference, FASE 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings, pp. 53–67 (2013)
31. Haugen, Ø., Wasowski, A., Czarnecki, K.: CVL: common variability language. In: 17th International Software Product Line Conference, SPLC 2013, Tokyo, Japan - August 26 - 30, 2013, p. 277 (2013)
32. Holl, G., Grünbacher, P., Rabiser, R.: A systematic review and an expert survey on capabilities supporting multi product lines. *Information & Software Technology* **54**(8), 828–852 (2012)

33. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Tech. Rep. CMU/SEI-90-TR-021 (1990)
34. Kang, K.C., Lee, J., Donohoe, P.: Feature-oriented project line engineering. *IEEE Software* **19**(4), 58–65 (2002)
35. Karatas, A.S., Oguztüziin, H., Dogru, A.H.: From extended feature models to constraint logic programming. *Sci. Comput. Program.* **78**(12), 2295–2312 (2013)
36. Kaytoue, M., Kuznetsov, S.O., Napoli, A.: Revisiting numerical pattern mining with formal concept analysis. In: *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, pp. 1342–1347 (2011)
37. Krueger, C.W.: Easing the transition to software mass customization. In: *Software Product-Family Engineering, 4th International Workshop, (PFE'01) Revised Papers*, pp. 282–293 (2001)
38. Linsbauer, L., Lopez-Herrejon, R.E., Egyed, A.: Feature model synthesis with genetic programming. In: *Search-Based Software Engineering - 6th International Symposium, SSBSE 2014, Fortaleza, Brazil, August 26-29, 2014. Proceedings*, pp. 153–167 (2014)
39. Loesch, F., Ploedereder, E.: Restructuring variability in software product lines using concept analysis of product configurations. In: *11th European Conference on Software Maintenance and Reengineering, Software Evolution in Complex Software Intensive Systems, CSMR 2007, 21-23 March 2007, Amsterdam, The Netherlands*, pp. 159–170 (2007)
40. Loh, W.: Classification and regression trees. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* **1**(1), 14–23 (2011)
41. Lopez-Herrejon, R.E., Linsbauer, L., Galindo, J.A., Parejo, J.A., Benavides, D., Segura, S., Egyed, A.: An assessment of search-based techniques for reverse engineering feature models. *Journal of Systems and Software* **103**, 353–369 (2015)
42. Mannion, M.: Using first-order logic for product line model validation. In: *Software Product Lines, Second International Conference, SPLC 2, San Diego, CA, USA, August 19-22, 2002, Proceedings*, pp. 176–187 (2002)
43. Martinez, J., Tërnavá, X., Ziadi, T.: Software product line extraction from variability-rich systems: the Robocode case study. In: *Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 1, SPLC 2018, Gothenburg, Sweden, September 10-14, 2018*, pp. 132–142 (2018)
44. Martinez, J., Ziadi, T., Bissyandé, T.F., Klein, J., Traon, Y.L.: Automating the extraction of model-based software product lines from model variants (T). In: *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE'15)*, pp. 396–406. IEEE Computer Society (2015)
45. Martinez, J., Ziadi, T., Bissyandé, T.F., Klein, J., Traon, Y.L.: Bottom-up technologies for reuse: automated extractive adoption of software product lines. In: *Proceedings of the 39th International Conference on Software Engineering, (ICSE'17), Companion Volume*, pp. 67–70. IEEE Computer Society (2017)
46. Nasr, S.B., Bécan, G., Acher, M., Filho, J.B.F., Sannier, N., Baudry, B., Davril, J.: Automated extraction of product comparison matrices from informal product descriptions. *Journal of Systems and Software* **124**, 82–103 (2017)
47. Poelmans, J., Ignatov, D.I., Kuznetsov, S.O., Dedene, G.: Formal concept analysis in knowledge processing: A survey on applications. *Expert Syst. Appl.* **40**(16), 6538–6560 (2013)
48. Pohl, K., Böckle, G., van der Linden, F.: *Software Product Line Engineering - Foundations, Principles, and Techniques*. Springer (2005)
49. Priss, U.: Formal concept analysis in information science. *ARIST* **40**(1), 521–543 (2006)
50. Riebisch, M., Böllert, K., Streitferdt, D., Philippow, I.: Extending feature diagrams with uml multiplicities. In: *Proceedings of the 6th World Conference on Integrated Design & Process Technology (IDPT'02)* (2002)
51. Rysse, U., Ploennigs, J., Kabitzsch, K.: Extraction of feature models from formal contexts. In: *Software Product Lines - 15th International Conference, SPLC 2011, Munich, Germany, August 22-26, 2011. Workshop Proceedings (Volume 2)*, p. 4 (2011)

52. Safdar, S.A., Yue, T., Ali, S., Lu, H.: Evaluating variability modeling techniques for supporting cyber-physical system product line engineering. In: Proceedings of the 9th International Conference on System Analysis and Modeling. Technology-Specific Aspects of Models (SAM'16), pp. 1–19 (2016)
53. Sannier, N., Acher, M., Baudry, B.: From comparison matrix to variability model: The wikipedia case study. In: 2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013, pp. 580–585 (2013)
54. Sawyer, P., Mazo, R., Diaz, D., Salinesi, C., Hughes, D.: Using constraint programming to manage configurations in self-adaptive systems. *IEEE Computer* **45**(10), 56–63 (2012)
55. Schmid, K., John, I.: A customizable approach to full lifecycle variability management. *Sci. Comput. Program.* **53**(3), 259–284 (2004)
56. She, S., Lotufo, R., Berger, T., Wasowski, A., Czarnecki, K.: Reverse engineering feature models. In: Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011, pp. 461–470 (2011)
57. She, S., Ryssel, U., Andersen, N., Wasowski, A., Czarnecki, K.: Efficient synthesis of feature models. *Information & Software Technology* **56**(9), 1122–1143 (2014)
58. Temple, P., Acher, M., Jézéquel, J., Barais, O.: Learning contextual-variability models. *IEEE Software* **34**(6), 64–70 (2017)
59. Tilley, T., Cole, R., Becker, P., Eklund, P.W.: A survey of formal concept analysis support for software engineering activities. In: Formal Concept Analysis, Foundations and Applications, pp. 250–271 (2005)