HAL
archives-ouvertes.fr

Reverse Engineering Feature Models from Software Configurations using Formal Concept Analysis

Ra'Fat Ahmad Al-Msie'Deen, Marianne Huchard, Abdelhak-Djamel Seriai, Christelle Urtado, Sylvain Vauttier

▶ **To cite this version:**

Ra'Fat Ahmad Al-Msie'Deen, Marianne Huchard, Abdelhak-Djamel Seriai, Christelle Urtado, Sylvain Vauttier. Reverse Engineering Feature Models from Software Configurations using Formal Concept Analysis. CLA: Concept Lattices and their Applications, Ondrej Krídlo, Oct 2014, Košice, Slovakia. pp.95-106. hal-01075524

HAL Id: hal-01075524
https://hal-auf.archives-ouvertes.fr/hal-01075524

Submitted on 17 Oct 2014

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.

Reverse Engineering Feature Models from Software Configurations using Formal Concept Analysis

R. AL-msie'deen[1], M. Huchard[1], A.-D. Seriai[1], C. Urtado[2] and S. Vauttier[2]

[1]LIRMM / CNRS & Montpellier 2 University, France
Al-msiedee, huchard, Seriai@lirmm.fr
[2]LGI2P / Ecole des Mines d'Alès, Nîmes, France
Christelle.Urtado, Sylvain.Vauttier@mines-ales.fr

Abstract. Companies often develop in a non-disciplined manner a set of software variants that share some features and differ in others to meet variant-specific requirements. To exploit existing software variants and manage them coherently as a software product line, a feature model must be built as a first step. To do so, it is necessary to extract mandatory and optional features from the code of the variants in addition to associate each feature implementation with its name. In previous work, we automatically extracted a set of feature implementations as a set of source code elements of software variants and documented the mined feature implementations based on the use-case diagrams of these variants. In this paper, we propose an automatic approach to organize the mined documented features into a feature model. The feature model is a tree which highlights mandatory features, optional features and feature groups (and, or, xor groups). The feature model is completed with requirement and mutual exclusion constraints. We rely on Formal Concept Analysis and software configurations to mine a unique and consistent feature model. To validate our approach, we apply it on several case studies. The results of this evaluation validate the relevance and performance of our proposal as most of the features and their associated constraints are correctly identified.

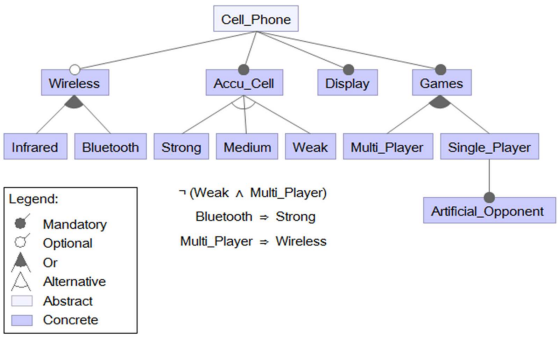
Keywords: Software Product Line, Feature Models, Software Product Variants, Formal Concept Analysis, Product-by-feature matrix.

1 Introduction

To exploit existing software variants and build a software product line (SPL), a feature model (FM) must be built as a first step. To do so, it is necessary to extract mandatory and optional features in addition to associate each feature with its name. In our previous work [1,2], we have presented an approach called REVPLINE [1] to identify and document features from the object-oriented source code of a collection of software product variants.

[1] REVPLINE stands for RE-engineering Software Variants into Software Product Line.

Dependencies between features need to be expressed via a FM which is a *de facto* standard formalism [3,4]. A FM is a tree-like hierarchy of features and constraints between them (*cf.* left side of Figure 1). FMs aim at describing the variability of a SPL in terms of features. A FM defines which feature combinations lead to valid products within the SPL (*cf.* right side of Figure 1). We illustrate our approach with the cell phone SPL FM and its 16 valid product configurations (*cf.* Figure 1) [5].



	Cell_Phone	Wireless	Infrared	Bluetooth	Accu_Cell	Strong	Medium	Weak	Display	Games	Multi_Player	Single_Player	Artificial_Opponent
P-1	×	×	×		×	×			×	×	×		
P-2	×	×		×	×	×			×	×	×		
P-3	×	×	×	×	×	×			×	×	×		
P-4	×	×	×		×		×		×	×	×		
P-5	×				×			×	×	×		×	×
P-6	×				×	×			×	×		×	×
P-7	×	×	×		×	×			×	×		×	×
P-8	×	×		×	×	×			×	×		×	×
P-9	×	×	×	×	×	×			×	×		×	×
P-10	×				×		×		×	×		×	×
P-11	×	×	×		×		×		×	×		×	×
P-12	×	×	×		×			×	×	×		×	×
P-13	×	×	×		×		×		×	×	×	×	×
P-14	×	×	×		×	×			×	×	×	×	×
P-15	×	×		×	×	×			×	×	×	×	×
P-16	×	×	×	×	×	×			×	×	×	×	×

Fig. 1. Valid product configurations of cell phone SPL feature model [5].

Figure 1 shows the FM of the *cell phone* SPL [5]. The *Cell_Phone* feature is the root feature of this FM; hence it is selected in every program configuration. It has three mandatory child features (*i.e.*, the *Accu_Cell*, *Display* and *Games* features), which are also selected in every product configuration as their parent is always included. The children of the *Accu_Cell* feature form an *exclusive-or* relation, meaning that the programs of this SPL include exactly one out of the three *Strong*, *Medium* or *Weak* features. The *Multi_Player* and *Single_Player* features constitute an *inclusive-or*, which necessitates that at least one of these two features is selected in any valid program configuration. *Single_Player* has *Artificial_Opponent* as a mandatory child feature. The *Wireless* feature is an optional child feature of root; hence it may or may not be selected. Its *Infrared* and *Bluetooth* child features form an *inclusive-or* relation, meaning that if a program includes the *Wireless* feature then at least one of its two child features has to be selected as well. The cell phone SPL also introduces three cross-tree constraints. While the *Multi_Player* feature cannot be selected together with the *Weak* feature, it cannot be selected without the *Wireless* feature. Lastly, the *Bluetooth* feature requires the *Strong* feature.

Galois lattices and concept lattices [6] are core structures of a data analysis framework (Formal Concept Analysis) for extracting an ordered set of con-

cepts from a dataset, called a formal context, composed of objects described by attributes. In our approach, we consider the AOC-poset (for Attribute-Object-Concept poset) [7], which is the sub-order of the concept lattice restricted to attribute-concepts and object-concepts. Attribute-concepts (*resp.* object-concepts) are the highest (*resp.* lowest) concepts that introduce each attribute (*resp.* object). AOC-posets scale much better than lattices. For applying Formal Concept Analysis (FCA) we used the Eclipse eRCA platform[2].

Manual construction of a FM is both time-consuming and error-prone [8], even for a small set of configurations [9]. The existing approaches to extract FM from product configurations [8,10] suffer from a lot of challenges. The main challenge is that numerous candidate FMs can be extracted from the same input product configurations, yet only a few of them are meaningful and correct, while in our work we synthesize an accurate and meaningful FM using FCA. Moreover the majority of these approaches extract a basic FM without constraints between its features [11] while, in our work, we extract all kinds of FM constraints.

The remainder of this paper is structured as follows: Section 2 presents the reverse engineering FM process step-by-step. Next, Section 3 presents the way that we propose to evaluate the obtained FMs. Section **??** describes the experimentation and threats to the validity. Section 4 discusses the related work. Finally, in Section 5, we conclude this paper.

2 Step-by-Step FM Reverse Engineering

This section presents step-by-step the FM reverse engineering process. According to our approach, we identify the FM in seven steps as detailed in the following, using strong properties of FCA to group features among product configurations. The AOC-poset is built from a set of known products, and thus does not represent all possible products. Thus, the FM structure has to be considered only as a candidate feature organization that can be proposed to an expert. The algorithm is designed such that all existing products (used for construction of candidate FM) are covered by the FM. Besides, it allows to define possible unused close variants.

The first step of our FM extraction process is the identification of the AOC-poset. First, a *formal context*, where objects are software product variants and attributes are features (*cf.* Figure 1), is defined. The corresponding *AOC-poset* is then calculated. The intent of each concept represents features common to two or more products or unique to one product. As AOC-posets are ordered, the intent of the most general (*i.e.*, top) concept gathers mandatory features that are common to all products. The intents of all the remaining concepts represent the optional features. The extent of each of these concepts is the set of products sharing these features (*cf.* Figure 2). In the following algorithms, for a Concept C, we call $intent(C)$, $extent(C)$, $simplified\ intent(C)$, and $simplified\ extent(C)$ its associated sets. Efficient algorithms can be found in [7].

The other steps are presented in the next sections.

[2] The eRCA : `http://code.google.com/p/erca/`

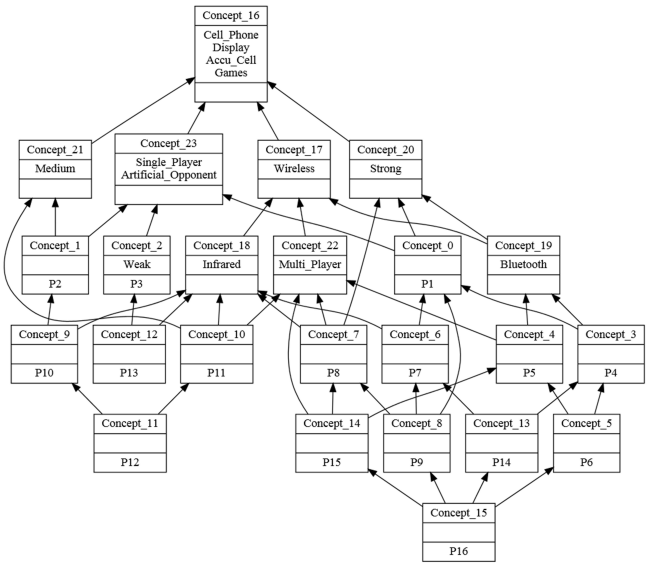


Fig. 2. The AOC-poset for the formal context of Figure 1.

2.1 Extracting root feature and mandatory features

Algorithm 1 is a simple algorithm for building the Base node (*cf.* Figure 3). Features in the top concept of the AOC-poset (Concept_16) are used in every product configuration. The *Cell_Phone* feature is the root feature of the cell phone FM (line 5). Then a mandatory Base node is created (lines 8,9). It is linked to nodes created to represent all the other features in the top concept, *i.e.*, Accu_Cell, Display and Games (lines 12-16).

2.2 Extracting atomic set of features (AND-group)

Algorithm 2 is a simple algorithm for building AND-groups of features (excluding all the mandatory features, line 3). An AND-group of features is created (line 8) to group optional features that appear in the same simplified intent (test line 6), meaning that these features are always used together in all the product configurations where they appear. Lines 12-16, nodes are created for every feature of the AND-group and they are attached to an *And* node. For instance, *Concept_23* in Figure 2 has a simplified intent with two features, *Single_Player* and *Artificial_Opponent*, leading to the *And* node of Figure 3.

2.3 Extracting exclusive-or relation

Features that form exclusive-or relation can be identified in the concept lattice using the meet (denoted by $\sqcap$) lattice operation [12], which amounts to compute

Algorithm 1: ComputeRootAndMandatoryFeature

// Top concept $\top$
$\exists$ F $\in$ A, which represents the name of the soft. family with F in feature set of $\top$
Data: AOC_K, $\leq_s$: the AOC-poset associated with K
Result: part of the FM containing root and mandatory features
// Compute the root Feature
CFS $\leftarrow$ intent($\top$)
Create node *root*, label (root) $\leftarrow$ F, type (*root*) $\leftarrow$ abstract
CFS$'$ $\leftarrow$ CFS $\setminus$ {F}
if *CFS*$'$ $\neq \emptyset$ **then**
    Create node *base* with label (*base*) $\leftarrow$ "Base"
    type (*base*) $\leftarrow$ abstract
    Create edge e = (*root*, *base*)
    type (e) $\leftarrow$ mandatory
    for *each* F_e *in CFS*$'$ **do**
        Create node *feature*, with label (*feature*) $\leftarrow$ F_e
        type (*feature*) $\leftarrow$ concrete
        create edge e = (*base*, *feature*)
        type (e) $\leftarrow$ mandatory

Algorithm 2: Compute*AtomicSetOfFeatures* (and groups)

Data: AOC_K, $\leq_s$: the AOC-poset associated with K
Result: part of the FM with *and* groups of features
// Compute atomic set of features
// Feature List (FL) is the list of all features (FL = A in K=(O, A, R)).
FL$'$ $\leftarrow$ FL $\setminus$ CFS // FL $\setminus$ intent($\top$)
AsF $\leftarrow \emptyset$
int count $\leftarrow$ 1
for *each concept* $C \neq \top$ *such that* | *simplified intent*($\mathcal{C}$) | ≥ 2 **do**
    AsF $\leftarrow$ AsF $\cup$ simplified intent($\mathcal{C}$)
    Create node *and* with label (*and*) $\leftarrow$ "AND"+ count
    type (*and*) $\leftarrow$ abstract
    create edge e = (*root*, *and*)
    type (e) $\leftarrow$ optional
    for *each F in simplified intent*($\mathcal{C}$) **do**
        create node *feature*, with label (*feature*) $\leftarrow$ F
        type (*feature*) $\leftarrow$ concrete
        create edge e =(*and*, *feature*)
        type (e) $\leftarrow$ mandatory

the greatest lower bounds in the AOC-poset. If a feature A is introduced in concept C_1, a feature B is introduced in concept C_2 and $C_1 \sqcap C_2 = \bot$ (and extent($\bot$) = $\emptyset$), that is, if the bottom of the lattice is the greatest lower bound of C_1 and C_2, the two features never occur together in a product. In our current

approach, we only build a single Xor group of features, when any group of mutually exclusive features exists. Computing exclude constraints (*see* Section 2.6) will deal with the many cases where several Xor group of features exist (a set of exclude constraints defining mutual exclusion is equivalent to a Xor group).

Algorithm 3 is a simple algorithm for building the single Xor group of features. The principle is to traverse the set of super-concepts of each minimum elements of the AOC-poset and to keep the concepts that are the super-concepts of only one minimum concept. Only features that are not used in the previous steps are considered in FL" (line 2). Lines 6-10, in our example, we consider the three minimum concepts Concept_11, Concept_12 and Concept_15. The many SSC sets are the sets of super-concepts for Concept_11, Concept_12 and Concept_15. $\mathcal{C}xor$ is the set of all concepts, except Concept_11, Concept_12 and Concept_15. Lines 11-15 only keep in $\mathcal{C}xor$ concepts that do not appear in two SSC sets. $\mathcal{C}xor$ contains concepts number 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 13, 14, 19, 20 and 21. Line 16 eliminates Concept_19 which is not a maximum. As there are three features (Medium, Strong, Weak, from Concept_21, Concept_20, and Concept_2 respectively) that are in FL" and in the simplified intent of concepts of $\mathcal{C}xor$ (line 18), an Xor node is created and linked to the root (lines 19-26). Then, on lines 27-33, nodes are created for the features and linked to the Xor node. Figure 3 shows this Xor node.

2.4 Extracting inclusive-or relation

Optional features are features that are used in some (but not all) product configurations. There are many ways of finding and organizing them. Algorithm 4 is a simple algorithm for building the Or group of features. In our approach, we pruned the AOC-poset by removing the top concept, concepts that correspond to AND groups of features, and concepts that correspond to features that form an exclusive-or relation. The remaining concepts define features that are grouped (lines 8-12) into an Or node (created and linked to the root on lines 4-7). In the AOC-poset of Figure 2, the *Wireless*, *Infrared*, *Bluetooth*, and *Multi_Player* features form an inclusive-or relation (*cf.* Figure 3).

2.5 Extracting require constraints

Algorithm 5 is a simple algorithm for identifying require constraints. A require constraint, *e.g.*, saying "variable feature A always requires variable feature B", can be extracted from the lattice via implications. We say that A implies B (written $A \rightarrow B$). The require constraints can be identified in the AOC-poset: when a feature F_1 is introduced in a subconcept of the concept that introduces another feature F_2, there is an implication $F_1 \rightarrow F_2$. We only consider the transitive reduction of the AOC-poset limited to Attribute-concepts (line 2) and features that are in simplified intents (line 3-4). In the AOC-poset of Figure 2, we find 6 require constraints from the transitive reduction of the AOC-poset to

Algorithm 3: Compute*Exclusive-or*Relation (Xor)

Data: AOC_K, $\leq_s$: the AOC-poset associated with K
Result: part of the FM with *XOR* group of features

// Compute exclusive-or relation
FL$''$ $\leftarrow$ FL$'$ $\setminus$ AsFs
$\mathcal{C}xor \leftarrow \emptyset$
SSCS $\leftarrow \emptyset$ // set of super-concept sets
Minimum-set $\leftarrow \emptyset$
for *each minimum of* AOC_K *denoted by* m **do**
    Let *SSC* the set of super-concepts of m (except $\top$)
    SSCS $\leftarrow$ SSCS $\cup$ {SSC}
    Minimum-set $\leftarrow$ Minimum-set $\cup$ {m}
    $\mathcal{C}xor \leftarrow \mathcal{C}xor \cup$ SSC
while *SSCS* $\neq \emptyset$ **do**
    SSC-1 $\leftarrow$ any element in (SSCS)
    SSCS $\leftarrow$ SSCS $\setminus$ SSC-1
    for *each SSC-2 in SSCS* **do**
        $\mathcal{C}xor \leftarrow \mathcal{C}xor \setminus$ (SSC-1 $\cap$ SSC-2)
$\mathcal{C}xor \leftarrow$ Max($\mathcal{C}xor$)
XFS $\leftarrow \emptyset$
if $|\mathcal{C}xor| > 1$ *and* $|FL'' \cap \cup_{C \in \mathcal{C}xor} simplified\ intent(C)| > 1$ **then**
    Create node *xor* with label (*xor*) $\leftarrow$ "XOR"
    type (*xor*) $\leftarrow$ abstract
    create edge e = (*root*, *xor*)
    // if all products are covered by $\mathcal{C}xor$
    if $\cup_{C \in \mathcal{C}xor} extent(C) = O$ **then**
        type (e) $\leftarrow$ mandatory
    else
        type (e) $\leftarrow$ optional
    for *each concept* $C \in \mathcal{C}xor$ **do**
        for *each F in simplified intent*($\mathcal{C}$) $\cap\ FL''$ **do**
            create node *feature*, with label (*feature*) $\leftarrow F$
            type (*feature*) $\leftarrow$ concrete
            create edge e = (*xor*, *feature*)
            type (e) $\leftarrow$ alternative
            XFS $\leftarrow$ XFS $\cup$ F

attribute-concepts (*cf.* Figure 3). Remark that implications ending to mandatory features are useless because they are represented in the FM by the Base node.

2.6 Extracting exclude constraints

In our current proposal, we compute binary exclude constraints $\neg(A \wedge B)$ under the condition that A and B are not both linked to the *Or* group. To mine

Algorithm 4: ComputeInclusive-orRelation (Or)

Data: AOC_K, $\leq_s$: the AOC-poset associated with K
Result: part of the FM with OR group of features

```
1  // Compute inclusive-or relation
2  FL''' ← FL'' \ XFS
3  if FL''' ≠ ∅ then
4      Create node or with label (or) ← "OR"
5      type (or) ← abstract
6      create edge e = (root, or)
7      type (e) ← optional
8      for each F in FL''' do
9          create node feature, with label (feature) ← F
10         type (feature) ← concrete
11         create edge e = (or, feature)
12         type (e) ← Or
```

Algorithm 5: Compute*Require*Constraint ($Requires$)

Data: AC_K, $\leq_s$: the AC-poset associated with K
Result: Require - the set of require constraints

```
1  Require ← ∅
2  for each edge (C1, C2) = e in transitive reduction of AC-poset do
3  for all f1, f2 with f1 ∈ simplified intent(C1) and f2 ∈ simplified intent(C2) do
4      Require ← Require ∪ {f1 ⟶ f2}
```

exclude constraints from an AOC-poset, we use the meet[3] of the introducers of the two involved features. For example, the meet of Concept_2 which introduces *Weak* and Concept_22 which introduces *Multi_Player* is the bottom (in the whole lattice). In the AOC-poset they don't have a common lower bound. We can thus deduce $\neg(Weak \wedge Multi\_Player)$. In the AOC-poset of Figure 2, there are three exclude constraints (*cf.* Figure 3). Algorithm 6 is a simple algorithm for identifying exclude constraints. It compares features that are below the OR group with each set of features in the intent of a minimum (line 4), in order to determine which are incompatible: this is the case for a pair (f1, f2) where f1 is in the OR group and not in the minimum intent, and f2 is in the minimum intent but not in the OR group (lines 6-10). Figure 3 shows the resulting FM based on the product configurations of Figure 1.

[3] in the lattice

Algorithm 6: Compute*Exclude*Constraint (*Excludes*)

Data: AOC_K, $\leq_s$: the AOC-poset associated with K
Result: Exclude - the set of exclude constraints.

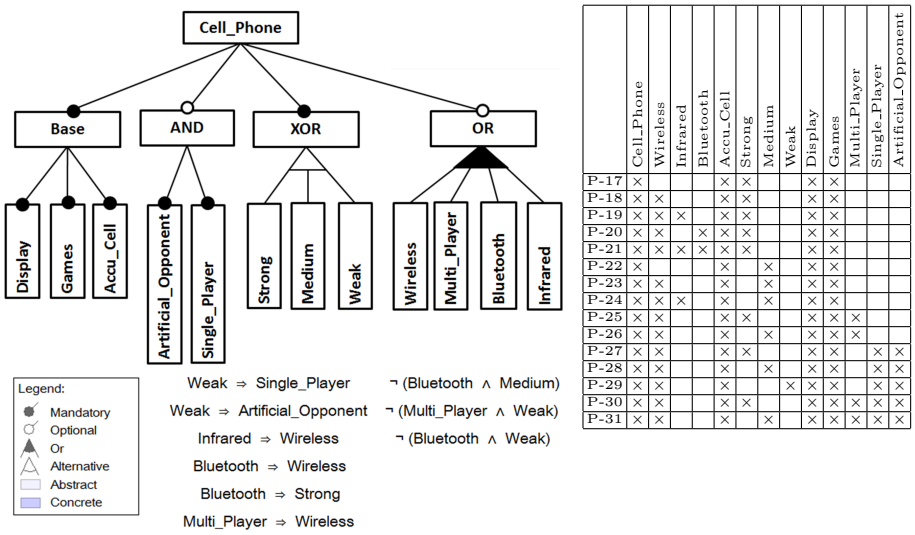
```
// Minimum-set from Algorithm 3
// FL''' from Algorithm 4
Exclude ← ∅
for each P ∈ Minimum-set do
    Pintent ← intent(P) \ intent(⊤)
    Opt-feat-set ← FL''' \ (FL''' ∩ Pintent)
    Super-feat-set ← Pintent \ (FL''' ∩ Pintent)
    if Opt-feat-set ≠ ∅ and Super-feat-set ≠ ∅ then
        for each f1 ∈ Opt-feat-set, f2 ∈ Super-feat-set do
            Exclude ← Exclude ∪ {¬(f1 ∧ f2)}
```

3 Experimentation

In order to evaluate the mined FM we rely on the SPLOT homepage[4] and the FAMA Tool[5]. Our implementation[6] converts the FM that has been drawn using SPLOT homepage into the format of FAMA. Then, we can easily generate a file containing all valid product configurations [13]. Figure 3 shows all valid product configurations for the mined FM by our approach (the first 16 product configurations are the same as in Figure 1). We compare the sets of configurations defined by the two FMs (*i.e.*, the initial FM compared to the mined FM). The mined FM introduces 15 extra product configurations which correspond to feature selection constraints that have not been detected by our algorithm.

Evaluation Metrics: In our work, we rely on *precision*, *recall* and *F-measure* metrics to evaluate the mined FM. All measures have values in [0, 1]. If recall equals 1, all relevant product configurations are retrieved. However, some retrieved product configurations might not be relevant. If precision equals 1, all retrieved product configurations are relevant. Nevertheless, relevant product configurations might not be retrieved. If F-Measure equals 1, all relevant product configurations are retrieved. However, some retrieved product configurations might not be relevant. F-Measure defines a trade-off between precision and recall, so that it gives a high value only in cases where both recall and precision are high. The result of the product configurations that are identified by the mined cell phone FM is as follow: (*precision*: 0.51), (*recall*: 1.00) and (*F-Measure*: 0.68). The recall measure is 1 by construction, due to the fact that the algorithm was designed to cover existing products.

[4] SPLOT homepage : `http://gsd.uwaterloo.ca:8088/SPLOT/`
[5] FAMA Tool Suite : `http://www.isa.us.es/fama/`
[6] Source Code : `https://code.google.com/p/sxfmtofama/`



	Cell_Phone	Wireless	Infrared	Bluetooth	Accu_Cell	Strong	Medium	Weak	Display	Games	Multi_Player	Single_Player	Artificial_Opponent
P-17	×				×	×			×	×			
P-18	×	×			×	×			×	×			
P-19	×	×	×		×	×			×	×			
P-20	×	×		×	×	×			×	×			
P-21	×	×	×	×	×	×			×	×			
P-22	×				×		×		×	×			
P-23	×	×			×		×		×	×			
P-24	×	×	×		×		×		×	×			
P-25	×	×			×	×			×	×	×		
P-26	×	×			×		×		×	×	×		
P-27	×	×			×	×			×	×		×	×
P-28	×	×			×		×		×	×		×	×
P-29	×	×			×			×	×	×		×	×
P-30	×	×			×	×			×	×	×	×	×
P-31	×	×			×		×		×	×	×	×	×

Fig. 3. The mined FM and its extra product configurations.

To validate our approach[7], we ran experiments on 7 case studies: ArgoUML-SPL [1], mobile media software variants [2], public health complaint-SPL[8], video on demand-SPL [8,3,14], wiki engines [10], DC motor [11] and cell phone-SPL [5]. Table 1 summarizes the obtained results.

Results show that precision appears to be not very high for all case studies. This means that many of the identified product configurations of the mined FM are extra configurations (not in the initial set that is defined by the original FM). Considering the recall metric, its value is 1 for all case studies. This means that product configurations defined by the initial FM are included in the product configurations derived from the mined FM. Experiments show that if the generated AOC-poset has only one bottom concept there is no exclusive-or relation or exclude constraints from the given product configurations. In our work, the mined FM defines more configurations than the initial FM. The reason behind this limitation is that some feature selection constraints are not detected. Nevertheless, the AOC-poset contains information for going beyond this limitation. We plan to enhance our algorithm to deal with that issue, at the price of an increase of complexity.

4 Related Work

For the sake of brevity, we describe only the work that most closely relates to ours. The majority of existing approaches are designed to reverse engineer FM

[7] Source code: `https://code.google.com/p/refmfpc/`
[8] `http://www.ic.unicamp.br/~tizzei/phc/`

Table 1. The results of configurations that are identified by the mined FMs.

#	case study	Number of Products	Number of Features	Group of Features				CTCs		Execution times (in ms)	Evaluation Metrics		
				Base	Atomic Set of Features	Inclusive-or	Exclusive-or	Requires	Excludes		Precision	Recall	F-Measure
1	ArgoUML-SPL	20	11	×		×		×		509	0.60	1.00	0.75
2	Mobile media	8	18	×	×	×				441	0.68	1.00	0.80
3	Health complaint-SPL	10	16	×	×	×		×		439	0.57	1.00	0.72
4	Video on demand	16	12	×	×	×		×		572	0.66	1.00	0.80
5	Wiki engines	8	21	×	×	×	×	×	×	555	0.54	1.00	0.70
6	DC motor	10	15			×		×		444	0.83	1.00	0.90
7	Cell phone-SPL	16	13	×	×	×	×	×	×	486	0.51	1.00	0.68

from high level models (*e.g.*, product descriptions) [10,14]. Some approaches offer an acceptable solution but are not able to identify important parts of FM such as cross-tree constraints, and-group, or-group, xor-group [11]. The main challenge of works that reverse engineer FMs from product configurations ([8,3]) is that numerous candidate FMs can be extracted from the same input configurations, yet only a few of them are meaningful and correct. The majority of existing approaches are designed to identify the dependencies between features regardless of FM hierarchy [8]. Work that relies on FCA to extract a FM does not fully exploit resulting lattices. In [11], authors rely on FCA to extract a basic FM without cross-tree constraints, while in [12], authors use FCA as a tool to understand the variability of existing SPL based on product configurations. Their work does not produce FMs. In our work, we rely on FCA to extract FMs from the software configurations. The resulting FMs exactly describe the given product configuration set. The proposed approach is able to identify all parts of FMs.

5 Conclusion

In this paper, we proposed an automatic approach to extract FMs from software variants configurations. We rely on FCA to extract FMs including configuration constraints. We have implemented our approach and evaluated its produced results on several case studies. The results of this evaluation showed that the resulting FMs exactly describe the given product configuration set. The FMs are generated in very short time, because our FCA tool (based on traversals of the AOC-poset) scales significantly better than the standard FCA approaches to calculate and traverse the lattices. The current work extracts a FM with two levels of hierarchy. As a perspective of this work, we plan to enhance the extracted FM by increasing the levels of hierarchy based on AOC-poset structure and to avoid allowing the FM to represent extra configurations.

Acknowledgment The authors would like to thank the reviewers for their valuable remarks that helped improve the paper. This work has been supported by the CUTTER ANR-10-BLAN-0219 project.

References

1. Al-Msie'deen, R., Seriai, A., Huchard, M., Urtado, C., Vauttier, S., Salman, H.E.: Mining features from the object-oriented source code of a collection of software variants using formal concept analysis and latent semantic indexing. In: SEKE '13. (2013) 244–249
2. Al-Msie'deen, R., Seriai, A., Huchard, M., Urtado, C., Vauttier, S.: Documenting the mined feature implementations from the object-oriented source code of a collection of software product variants. In: SEKE '14. (2014) 264–269
3. Acher, M., Baudry, B., Heymans, P., Cleve, A., Hainaut, J.L.: Support for reverse engineering and maintaining feature models. In: VaMoS '13, New York, NY, USA, ACM (2013) 20:1–20:8
4. She, S., Lotufo, R., Berger, T., Wasowski, A., Czarnecki, K.: Reverse engineering feature models. In: ICSE '11, New York, NY, USA, ACM (2011) 461–470
5. Haslinger, E.N.: Reverse engineering feature models from program configurations. Master's thesis, Johannes Kepler University Linz, Linz, Austria (September 2012)
6. Ganter, B., Wille, R.: Formal concept analysis - mathematical foundations. Springer (1999)
7. Berry, A., Gutierrez, A., Huchard, M., Napoli, A., Sigayret, A.: Hermes: a simple and efficient algorithm for building the AOC-poset of a binary relation, Annals of Mathematics and Artificial Intelligence (may 2014)
8. Haslinger, E.N., Lopez-Herrejon, R.E., Egyed, A.: Reverse engineering feature models from programs' feature sets. In: WCRE '11, IEEE (2011) 308–312
9. Andersen, N., Czarnecki, K., She, S., Wasowski, A.: Efficient synthesis of feature models. In: SPLC (1), ACM (2012) 106–115
10. Acher, M., Cleve, A., Perrouin, G., Heymans, P., Vanbeneden, C., Collet, P., Lahire, P.: On extracting feature models from product descriptions. In: VaMoS '12, New York, NY, USA, ACM (2012) 45–54
11. Ryssel, U., Ploennigs, J., Kabitzsch, K.: Extraction of feature models from formal contexts. In: SPLC '11, New York, NY, USA, ACM (2011) 4:1–4:8
12. Loesch, F., Ploedereder, E.: Optimization of variability in software product lines. In: SPLC '07, Washington, DC, USA, IEEE Computer Society (2007) 151–162
13. Benavides, D., Segura, S., Ruiz-Cortés, A.: Automated analysis of feature models 20 years later: A literature review. Inf. Syst. **35**(6) (September 2010) 615–636
14. Lopez-Herrejon, R.E., Galindo, J.A., Benavides, D., Segura, S., Egyed, A.: Reverse engineering feature models with evolutionary algorithms: An exploratory study. In: SSBSE, Springer (2012) 168–182