Breathing Ontological Knowledge Into Feature Model Management

Guillaume Bécan, Mathieu Acher, Benoit Baudry, Sana Ben Nasr

Breathing Ontological Knowledge Into Feature Model Management

Guillaume Bécan, Mathieu Acher, Benoit Baudry, and Sana Ben Nasr

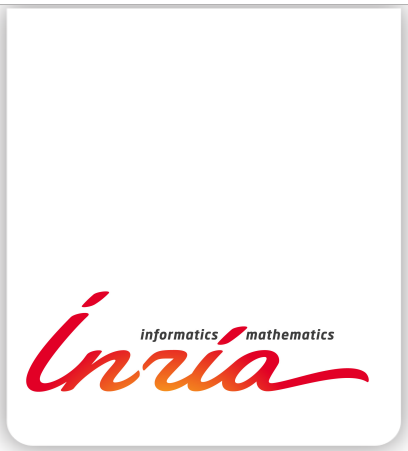
TECHNICAL REPORT

N° 441

September 2013

Project-Team Triskell

ISSN 0249-0803 ISRN INRIA/RT--441--FR+ENG



Breathing Ontological Knowledge Into Feature Model Management

Guillaume Bécan, Mathieu Acher, Benoit Baudry, and Sana Ben Nasr

Project-Team Triskell

Technical Report n° 441 — September 2013 — 16 pages

Abstract: Feature Models (FMs) are a popular formalism for modeling and reasoning about the configurations of a software product line. As the manual construction or management of an FM is time-consuming and error-prone for large software projects, recent works have focused on automated operations for reverse engineering or refactoring FMs from a set of configurations/dependencies. Without prior knowledge, meaningless ontological relations (as defined by the feature hierarchy and groups) are likely to be synthesized and cause severe difficulties when reading, maintaining or exploiting the resulting FM. In this paper we define a generic, ontological-aware synthesis procedure that guides users when identifying the likely siblings or parent candidates for a given feature. We develop and evaluate a series of heuristics for clustering/weighting the logical, syntactic and semantic relationships between features. Empirical experiments on hundreds of FMs, coming from the SPLOT repository and Wikipedia, show that an hybrid approach mixing logical and ontological techniques outperforms state-of-the-art solutions and offers the best support for reducing the number of features a user has to consider during the interactive selection of a hierarchy.

Key-words: reverse engineering, product lines, configuration, feature model, model management, slicing

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Breathing Ontological Knowledge Into Feature Model Management

Guillaume Bécan, Mathieu Acher, Benoit Baudry, and Sana Ben Nasr
Inria / IRISA, University of Rennes 1, France
firstname.lastname@inria.fr

1. INTRODUCTION

Real world success stories of *Software Product Lines* (SPLs) show that the effective management of a large set of products is possible [1, 2]. The factorization and exploitation of common features of the products as well as the handling of their variability [3–6] is an essential step for these stories. Large scale open source or industrial SPLs contain thousands of features and many logical dependencies among them [7, 8]. This complexity poses a challenge for both developers and users of SPLs.

Feature Models (FMs) are by far the most popular notation for modeling and reasoning about an SPL [8, 9]. FMs offer a simple yet expressive way to define a set of legal *configurations* (i.e., combinations of features) each corresponding to a product of an SPL [10–16]. Another important and dual aspect of an FM is the way features are conceptually related. A tree-like hierarchy and feature groups are notably used to organize features into multiple levels of increasing detail and define the *ontological semantics* [17] of an FM.

A manual elaboration of an FM is not realistic for large projects or for legacy systems. Many procedures propose to reverse engineer dependencies and features' sets from existing software artefacts – being source code, configuration files, spreadsheets or requirements [18–28] (see Figure 1, left). From these logical dependencies (typically formalized and encoded as a propositional formula in conjunctive or disjunctive normal form), numerous FMs can represent the exact same set of configurations, out of which numerous candidates are obviously not maintainable since the retained hierarchy is not adequate [20, 29]. An example of such FM is given in Figure 2a.

Both configuration and ontological aspects are important when managing FMs. First, the proper handling of configuration semantics is unquestionable. Otherwise the FM may expose to customers configurations that actually do not correspond to any existing product [30–34]. Second, a doubtful feature hierarchy may pose severe problems for a further exploitation by automated transformation tools or by stakeholders (humans) that need to understand, maintain and exploit an FM – as it is the case in many FM-based approaches [4, 9, 25, 30, 32, 35–37]. Figure 2b depicts a highly questionable user interface of a configurator that could have been generated from the FM of Figure 2a and illustrates the consequence of an inappropriate treatment of the ontological semantics.

The problem addressed in this paper can be formulated as follows: How to automate the synthesis of an FM from a set of dependencies while both addressing the configuration and ontological semantics? Is it feasible to fully synthesize an FM? Can we reduce the user effort when reviewing and selecting the likely siblings or parent candidates for a given feature? Given a set of dependencies, we challenge synthesis techniques to assist users in selecting a feature hierarchy as close as possible to a *ground truth* – without an *a priori* knowledge of the ground truth.

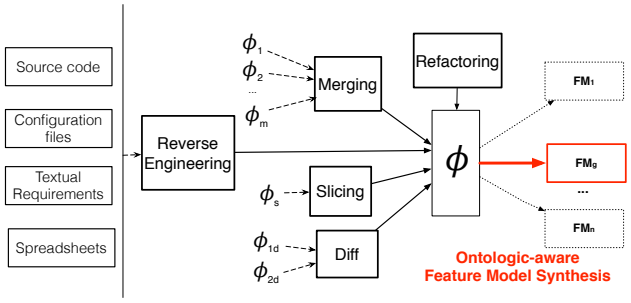
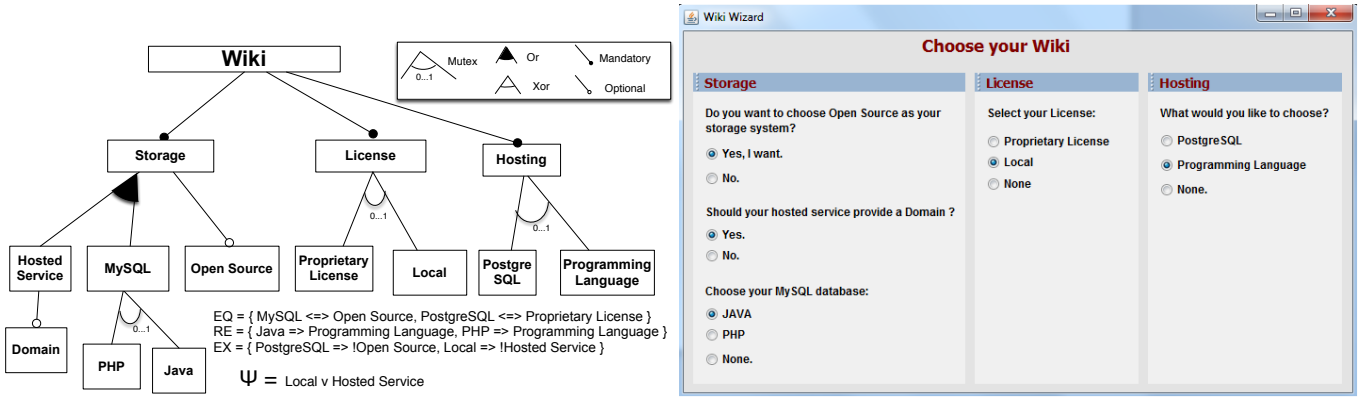


Figure 1: A key issue for automated operations: numerous FMs conformant to $[\![\phi]\!]$ exist but are likely to have an inappropriate ontological semantics

Several synthesis techniques for FMs have been proposed, mostly in the context of reverse engineering FMs, but they neglected either configuration or ontological aspects of an FM [11, 23–26, 38–43]. Other works proposed some support but users have to manually choose a relevant hierarchy among the thousands of possibilities [29, 39]. It is time-consuming and non realistic for large projects with hundreds or thousands of features. A notable exception is She *et al.* [20] who proposed a procedure to rank the correct parent features in order to reduce the task of a user. However they assume the existence of textual artefacts describing features, only consider possible parent-child relationships, and the experimented heuristics are specific to the operating system domain.

In this paper, we describe a *generic* ontologic-aware FM synthesis procedure and propose an hybrid solution combining both ontological and logical techniques. The heuristics rely on general ontologies (e.g., from Wordnet or Wikipedia), are sound and applicable without prior knowledge or artefacts, and can be used either to fully synthesize an FM or guide the users during an interactive process. We perform an empirical evaluation on 123 sets of dependencies/configurations for which we have a ground truth FM. We use two data sets: (1) the SPLOT repository [44] and (2) large FMs extracted from *product comparison matrices* (PCMs) found in Wikipedia [45]. FMs come from different domains and their complexity vary. In average, there are 18 features per FM of SPLOT, 72 features per FM of PCMs, and less than 8 parent features per features to consider. The experiments show that an hybrid approach provides the best support for decreasing the number of choices a user has to perform when interactively selecting a feature hierarchy. Specifically, we make the following contributions:

- We develop a series of heuristics for clustering and ranking the syntactic/semantic relationships between features. We



(a) fm_u (b) Configurator UI

Figure 2: Each configuration of fm_u corresponds to a product listed in Figure 3a. In the right, a non intuitive user interface of a configurator that could have been generated from fm_u due to its doubtful ontological semantics

also develop logical heuristics that can be applied before *"breathing ontological knowledge"*;

- The hybrid approach can retrieve, in average, 36.7% of parent-child relationships of the SPLOT FMs (45.5% for the PCM dataset) in one step and without any user intervention. Although the hybrid approach constitutes the state-of-the-art heuristic, the results shows that a fully automated synthesis is likely to produce FMs far from the ground truths. We provide evidence that the role of the user remains crucial and we highlight the interactive nature of the synthesis process;
- In terms of user support, the hybrid approach ranks the correct parent among the 2 first results for 49.1% of the features (for the SPLOT dataset) and 56.3% of the features (for the PCM dataset). The clusters retrieved by the hybrid approach are correct in more than half of the time while the number of clusters to review remains rather low (4.1 for the SPLOT dataset, 17.6 for the PCM dataset);
- We empirically compare our method with an existing technique [43] and logical-based heuristics we develop. In particular, we analyze the strengths, weaknesses and possible synergies between logical and ontological-based techniques, leading to the design of an hybrid approach.

The contributions not only advance the state-of-the-art of reverse engineering FMs. Important management operations of FMs (slicing, merging, diff, refactoring) also benefit from ontological capabilities since all are based on the synthesis procedure (see Fig. 1). Our procedures open avenues for practical reverse engineering or maintenance of configurable systems more and more reported in the industry or observed in the open source community.

2. BACKGROUND

Feature Models (FMs) aim at characterizing the valid combinations of features (a.k.a. configurations) of a system under study. A feature hierarchy (a tree) is used to facilitate the organization and understanding of a potentially large number of concepts (features).

2.1 Syntax of Feature Models

Different syntactic constructions are offered to attach variability information to features organized in the hierarchy (see Definition 1). As an example, the FM of Figure 3b describes a family of Wiki engines. The FM states that Wiki has three *mandatory* features, Storage, Hosting and License and one *optional* feature Programming Language. There are two alternatives for Hosting: Hosted Service and Local features form an *Xor*-group (i.e., at least and at most one feature must be selected). Similarly, the features Open Source and Proprietary License form an *Xor*-group of License. Cross-tree *constraints* over features can be specified to restrict their valid combinations. Any kinds of constraints expressed in Boolean logic, including predefined forms of Boolean constraints (equals, requires, excludes), can be used. For instance, the feature PostgreSQL is logically related to Proprietary License and Domain. A similar abstract syntax is used in [11, 20, 41] while other dialects slightly differ [46].

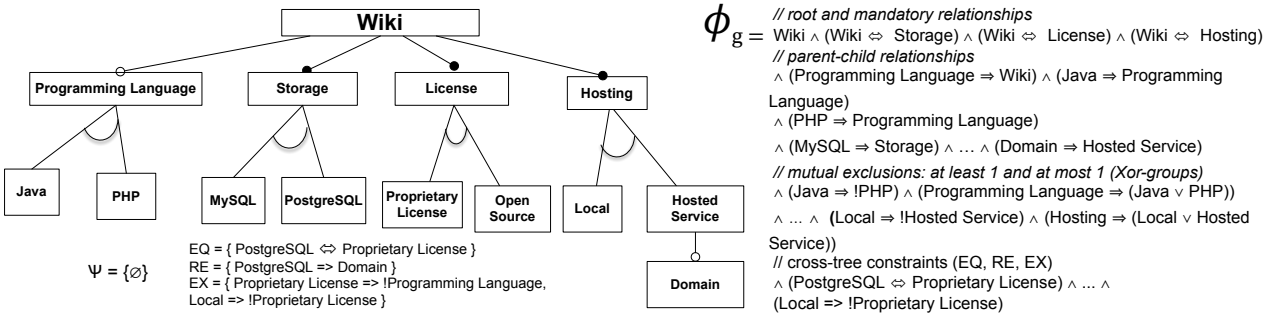
Definition 1 (Feature Model) *A feature diagram is a 8-tuple* $\langle G, E_M, G_{MTX}, G_{XOR}, G_{OR}, EQ, RE, EX \rangle$*:* $G = (\mathscr{F}, E)$ *is a rooted tree where* $\mathscr{F}$ *is a finite set of features,* $E \subseteq \mathscr{F} \times \mathscr{F}$ *is a set of directed child-parent edges ;* $E_M \subseteq E$ *is a set of edges that define mandatory features with their parents ;* $G_{MTX}, G_{XOR}, G_{OR} \subseteq 2^{\mathscr{F}}$ *are non-overlapping sets of edges participating in feature groups.* EQ *(resp.* RE*,* EX*) is a set of equals (resp. requires, excludes) constraints whose form is* $A \Leftrightarrow B$ *(resp.* $A \Rightarrow B$*,* $A \Rightarrow \neg B$*) with* $A \in \mathscr{F}$ *and* $B \in \mathscr{F}$*. The following well-formedness rule holds: a feature can have only one parent and can belong to only one feature group. A feature model is a pair* $\langle FD, \psi \rangle$ *where FD is a feature diagram, and* ψ *is a Boolean formula over* $\mathscr{F}$*.*

2.2 Configuration and Ontological Semantics

The essence of an FM is its *configuration semantics* (see Definition 2). The syntactical constructs are used to restrict the combinations of features authorised by an FM, e.g., at most one feature can be selected in a Mutex-group. The cardinality of a feature group is a pair (i, j) (with $i \leq j$) and denotes that at least i and at most j of its k arguments are true. G_{MTX} (resp. G_{XOR}, G_{OR}) are sets of *Mutex-groups* (resp. *Xor-groups*, *Or-groups*) whose cardinality is $(0,1)$ (resp. $(1,1)$, $(1,m)$: m being the number of features in the Or-group). The configuration semantics can be specified via translation to Boolean logic [11]. As an example, the formula ϕ_g (right of Figure 3b) defines the legal configurations of fm_g. In particular, the configuration semantics states that a feature cannot be selected without its parent, i.e., all features, except the root, logically imply their parent. As a consequence, the *feature hierarchy*

	PostgreSQL	MySQL	License	Domain	Proprietary License	Local	Programming Language	Java	Storage	PHP	Open Source	Wiki	Hosting	Hosted Service
P1	✓	×	✓	✓	✓	×	×	×	✓	×	×	✓	✓	✓
P2	×	✓	✓	×	×	×	✓	✓	✓	×	✓	✓	✓	✓
P3	×	✓	✓	×	×	✓	×	×	✓	×	✓	✓	✓	×
P4	×	✓	✓	×	×	×	×	×	✓	×	✓	✓	✓	✓
P5	×	✓	✓	×	×	✓	✓	×	✓	✓	✓	✓	✓	×
P6	×	✓	✓	✓	×	×	✓	✓	✓	×	✓	✓	✓	✓
P7	×	✓	✓	×	×	×	✓	×	✓	✓	✓	✓	✓	✓
P8	×	✓	✓	✓	×	×	✓	×	✓	✓	✓	✓	✓	✓
P9	×	✓	✓	✓	×	×	×	×	✓	×	✓	✓	✓	✓
P10	×	✓	✓	×	×	✓	✓	✓	✓	×	✓	✓	✓	×

(a) Product comparison matrix (✓ feature is in the product ; × feature is not in the product)



(b) In the left fm_g ; in the right the corresponding formula ϕ_g

Figure 3: Another FM with same configuration semantics than fm_u ($[\![fm_g]\!] = [\![fm_u]\!]$) and the product comparison matrix of Fig. 3a but with a more appropriate ontological semantics

also contributes to the definition of the configuration semantics.

Definition 2 (Configuration Semantics) *A configuration of a feature model g is defined as a set of selected features.* $[\![g]\!]$ *denotes the set of valid configurations of g.*

Another crucial and dual aspect of an FM is its *ontological semantics* (see Definition 3). It defines the way features are conceptually related. Obviously, the feature hierarchy is part of the ontological definition. The parent-child relationships are typically used to *decompose* a concept into sub-concepts or to *specialize* a concept. There are also other kinds of implicit semantics of the parent-child relationships, *e.g.*, to denote that a feature is *"implemented by"* another feature [10]. Looking at Fig. 3b, the concept of Wiki is composed of different properties like Hosting, License, or Programming Language ; License can be either specialized as an Open source or a Proprietary License, etc. Feature groups are part of the ontological semantics (see Definition 3) since there exists FMs with the same configuration semantics, the same hierarchy but having different groups [20, 29].

Definition 3 (Ontological Semantics) *The hierarchy* $G = (\mathcal{F}, E)$ *and feature groups* (G_{MTX}, G_{XOR}, G_{OR}) *of a feature model define the semantics of features' relationships including their structural relationships and conceptual proximity.*

3. ONTOLOGIC-AWARE SYNTHESIS

We now explain the impact of ontological semantics w.r.t. to the FM synthesis, a problem at the heart of many reverse engineering and FM management procedures (see Fig. 1).

3.1 The Importance of Ontological Semantics

Let us consider the following re-engineering scenario (see [47] for more details). After having *reverse engineered* an FM, a practitioner aims to generate a graphical interface (like in Fig. 2b) for assisting users in customizing the product that best fits his/her needs. Unfortunately, the ontological semantics of the FM is highly questionable (see Fig. 2a) and poses two kinds of problems.

Automated exploitation. Transformation tools that operate over the FM will naturally exploit its ontological semantics. Fig. 2 gives an example of a possible transformation from an FM to a user interface (UI). The generated UI (see Fig. 2a) is as unintuitive as the ontological semantics is: features PHP and Java are below MySQL, instead of being below Programming Language ; Programming Language itself is badly located below Hosting ; Open Source is below Storage whereas the intended meanings was to state that a Wiki engine has different alternatives of a License. All in all, the series of questions and the organization of the elements in the UI are clearly non exploitable for a customer willing to choose a Wiki.

Human exploitation. One could argue that an automated transformation is not adequate and a developer is more likely to write or tune the transformation. In this case, the developer faces different problems. First, there are evident readability issues when the developer has to understand and operate over the model. Second, when writing the transformation, the ontological semantics cannot be exploited as such and she has to get around the issue, complicating her task. A solution could be to *refactor* the FM, but the operation is an instance of the synthesis problem (see below). Third, default transformation rules that could reduce her effort are likely to cause problems since based on the ontological semantics.

This scenario illustrates the importance of having FMs with an appropriate ontological semantics for a further exploitation in a forward engineering process. This need is also observed in other FM-based activities such as understanding a domain or a system [25, 48], communicating with other stakeholders [37], composing or slicing FMs [9, 15, 49], relating FMs to other artefacts (e.g., models) [30,32,33,35], or simply generating other artefacts from FMs [36].

3.2 Ontologic-aware FM Synthesis Problem

The development of an ontologic-aware FM synthesis procedure raises new challenges that have been overlooked so far.

FM synthesis problem. Given a set of features' names and boolean dependencies among features, the problem is to synthesize a maximal and sound FM conforming to the configuration semantics. Formally, let ϕ be a propositional formula in conjunctive or disjunctive normal form. A synthesis function takes as input ϕ and computes an FM such that $\mathscr{F}$ is equal to the boolean variables of ϕ and $[\![FM]\!] \equiv [\![\phi]\!]$. There are cases in which the diagrammatic part of an FM is not sufficient to express $[\![\phi]\!]$ (i.e., $[\![FD]\!] \subset [\![\phi]\!]$). It is thus required that the diagrammatic part is *maximal*. Intuitively, as much logical information as possible is represented in the diagram itself, without resorting to the constraints. (A comprehensive formalization is given in [41]).

Ontologic-aware FM synthesis. The problem tackled in this paper is a generalization of the FM synthesis problem. Numerous FMs, yet maximal, can actually represent the exact same set of configurations, out of which numerous present a naive hierarchical or grouping organization that mislead the ontological semantics of features and their relationships (e.g., see Fig. 2a *versus* Fig. 3b). We seek to develop an automated procedure that computes a well-formed FM both *i)* conformant to the configuration semantics (as expressed by the logical dependencies of a formula ϕ) and *ii)* exhibiting an appropriate ontological semantics.

The ontologic-aware synthesis problem not only arises when reverse engineering FMs. It is also apparent when *refactoring* an FM, i.e., producing an FM with the same configuration semantics but with a different ontological semantics. For example, the FM of Fig. 2a could be refactored to enhance its quality and make it exploitable. The operation of *slicing* an FM has numerous practical applications (decomposition of a configuration process in multi-steps, reduction of the variability space to test an SPL, reconciliation of variability views, etc.) [15]. Despite some basic heuristics, we observed that the retained hierarchy and feature groups can be inappropriate [29] (the same observation applies for the *merge* and *diff* operators). All these automated operations are instances of the FM synthesis problem (see Fig. 1) and impacted by the problem.

In [29], we empirically showed that once the feature hierarchy is defined, the variability information of the FM can be fully synthesized in the vast majority of cases. We thus focus on the challenge of selecting a hierarchy in the remainder.

4. AUTOMATED TECHNIQUES FOR FEATURE HIERARCHY SELECTION

Overview. Our ontologic-aware FM synthesis procedure essentially relies on a series of heuristics to rank parent-child relationships (see Section 4.2) and compute clusters of conceptually similar features (see Section 4.3). These two types of information can be interactively reviewed and exploited by a user, and if needs be, an optimum branching algorithm can synthesize a complete FM (see Fig. 4). The ontological heuristics do not assume any artefacts documenting features and can be combined with logical heuristics (see Section 4.5).

4.1 Selecting a Hierarchy

Sound selection. We recall that the feature hierarchy of an FM defines how features are ontologically related and also participates to the configuration semantics, since each feature logically implies its parent: $\forall (f_1, f_2) \in E, f_1 \Rightarrow f_2$. As a result, the candidate hierarchies, whose parent-child relationships violate the original constraints expressed by ϕ, can be eliminated upfront. For example, the feature Local cannot be a child feature of PostgreSQL since no configuration expressed in Fig. 3a authorizes such an implication. We rely on the *Binary Implication Graph* (BIG) of a formula (see Definition 4) to guide the selection of legal hierarchies. The BIG represents every implication between two variables (resp. features) of a formula (resp. FM), thus representing every possible parent-child relationships a legal hierarchy can have. Selecting a hierarchy now consists in selecting a set of the BIG's edges forming a rooted tree that contains all its vertices. Such a tree represents a branching of the graph (the counterpart of a spanning tree for undirected graphs). The selection over the BIG is *sound* and *complete* since every branching of the BIG is a possible hierarchy of the FM and every hierarchy is a branching of the BIG.

Definition 4 (Binary Implication Graph (BIG)) *A binary implication graph of a Boolean formula ϕ over $\mathscr{F}$ is a directed graph (V_{imp}, E_{imp}) where $V_{imp} = \mathscr{F}$ and $E_{imp} = \{(f_i, f_j) \mid \phi \wedge f_i \Rightarrow f_j\}$.*

Now that we have a convenient data structure, which captures every possible hierarchy, the whole challenge consists in selecting the most meaningful one.

A first natural strategy is to add a weight on every edge (f_1, f_2) of the BIG, defining the probability of f_2 to be the parent of f_1. The weights are computed by heuristics that directly evaluate the probability of a relationship between a feature and a parent candidate. From a user perspective, the *ranking list* can be consulted and exploited to select or ignore a parent. In addition, we perform hierarchical *clustering* based on the similarity of the features to compute groups of features. The intuitive idea is that clusters can be exploited to identify *siblings* in the tree since members of the clusters are likely to share a common parent feature. For example, clusters can help tuning the previously computed weights, thus increasing the quality of the previous ranking list. Moreover, users can operate over the clusters to define the parent of a group of features (instead of choosing a parent for all features of a group).

Once we get the two pieces of information, we select the branching that has the maximum total weight. To compute an optimum branching, we use Tarjan's algorithm [50, 51] whose complexity is $\mathscr{O}(m \log n)$ with n the number of vertices and m the number of edges. The hierarchy is then fed to syntesise a complete FM. The synthesis process is likely to be incremental and interactive ; users can validate and modify the ranking list and the set of clusters. The overall synthesis process is described in Fig. 4.

4.2 Heuristics for Parent Candidates

The design of the heuristics is guided by a simple observation: *parent-child relations in a hierarchy often represent a specialization or a composition of a concept (feature)*. For example, Java is a specialization of a Programming language while a Wiki is composed of a License, a Storage and a Programming Language. As siblings are specializations or parts of the more general concept of their parent, they share the same context. For example, Open source and Commercial are both referring to permissive rights about the use of a product. The intuitive idea is that sharing the same context tends to make a feature semantically close to its parent and its siblings.

From these observations, we developed several heuristics that exploit the feature names in order to compute the edges' weights of the BIG. We can divide the heuristics in two categories: syntactical heuristics and semantical heuristics.

Syntactical heuristics use edit distance and other metrics based on words' morphology to determine the similarity of two features. In

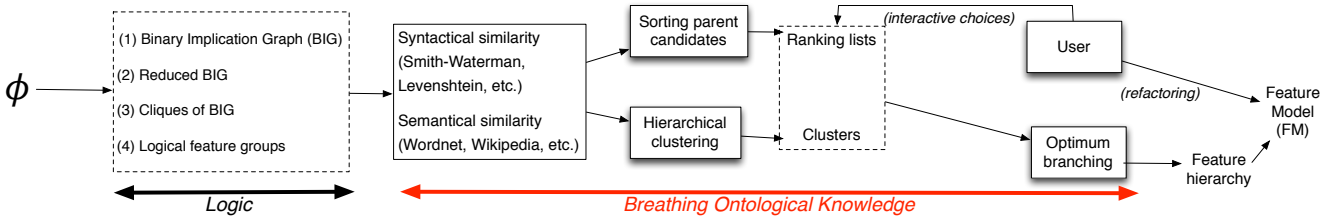


Figure 4: Ontologic-aware feature model synthesis

our example in Figure 3b, License is closer to Proprietary License than Storage because the two first features contains the same substring: *License*. We used *Smith-Waterman* [52] algorithm that looks for similar regions between two strings to determine their distance. We also used the so-called *Levenshtein* edit distance [53] that computes the minimal edit operations (renaming, deleting or inserting a symbol) required to transform the first string into the second one.

Semantical heuristics Syntactical heuristics have some limits: feature names that are not syntactically closed but semantically close (in the sense of meaning) are not identified (*e.g.* Java and PHP). Thus, we need to add some semantical knowledge to improve our technique. We explored two general ontologies out of which we built semantical metrics.

First, we explored *WordNet* [54]. This is a structured English dictionary that provides hyperonymy (specialization) and meronymy (composition) relations between word senses. As we are exclusively using the features' names, we could not use the most efficient metrics based on a text corpus [55]. Therefore, we selected two metrics named *PathLength* and *Wu&Palmer* that are only based on WordNet's structure. The *PathLength* metric gives the inverse of the length of the shortest path between two words in WordNet considering only hyperonymy relations. Wu and Palmer described a metric based on the depth of the words and their lowest common ancestor in the tree formed by hyperonymy relations [56].

These two metrics compute the similarity of two words, however features' name may contain several words. Wulf-Hadash et al. also used the *Wu&Palmer* metric in order to compute feature similarity [57]. We used the same formula for combining word similarity into sentence similarity:

$$Sim(f1,f2) = \frac{\sum_{i=1}^{m} \max_{j=1..n} wsim_{i,j} + \sum_{j=1}^{n} \max_{i=1..m} wsim_{i,j}}{m+n}$$

where m and n are respectively the number of words in $f1$ and $f2$ and $wsim_{i,j}$ is the similiarity between the i-th word of $f1$ and the j-th word of $f2$.

WordNet contains few technical words, thus we explored *Wikipedia* to increase the number of recognized words. The well known encyclopedia offers a large database containing text articles and links between them. Associating a set of articles to a feature enables the use of techniques that compute semantic similarity of texts. For example we can associate Java and Programming language to their respective articles in Wikipedia. Then we compute their similarity by comparing the links contained in these articles as proposed by Milne *et al.* [58, 59]. They created a model based on the hyperlink structure of Wikipedia in order to compute the semantic similarity of two articles. In this model, an article is represented by a vector containing the occurence of each link found in the article weighted by the probability of the link occuring in the entire Wikipedia database.

4.3 Detecting Siblings with Hierarchical Clustering

Defining weights on the BIG's edges and computing the optimum branching can be summarized as choosing the best parent for a feature. However, it is sometimes easier to detect that a group of features are siblings. To detect such siblings we reuse the previous heuristics to compute the similarity between features without considering the BIG structure. Then we perform agglomerative hierarchical clustering on these values. Agglomerative hierarchical clustering consists in putting each feature in a different cluster. Then, the closest clusters according to their similarity are merged. Finally, this operation is repeated until the similarity falls below a user specified threshold. We use the BIG to determine if the clusters belongs to one of the two categories below. In the following, C represents a cluster and *possibleParents* gives the parent candidates according to the BIG.

- $\exists f_p \in \mathscr{F}, \forall f_c \in C, f_p \in possibleParents(fc)$, *i.e.*, all the features can be siblings. It remains to find a common parent of the cluster among the other features.
- $\exists P \subset C, \forall f_p \in P, \forall f_c \in C, f_p \neq f_c, f_p \in possibleParents(fc)$, *i.e.*, some features are parent of the others. It remains to find the parent among the features within the cluster.

Once we have defined the parents of the clusters, we modify the edges' weights of the BIG to consider this new information during the optimum branching algorithm. This modification consists in setting the maximal weight to each edge between a child and its chosen parent. The rationale is that features of the clusters are likely to share the same parent, thus the idea of augmenting the weights. For example, in Fig. 3b, we could determine with a heuristic, that Java and PHP are semantically close, thus being in a same cluster. It corresponds to the first category exposed above. The key benefit is that we can now solely focus on their common parents (computed using the BIG). The best parent is chosen according to heuristics for parent candidates. (It should be noted that the heuristic is not necessarily the same one used for the clustering). We obtain Programming Language in the example. As a result, we assign the maximum weight (i.e., 1) for the BIG's edges, i.e., Java → Programming Language and PHP → Programming Language.

4.4 Implementation of Ontological Heuristics

We have developed six heuristics based on specialized libraries. The syntactical heuristics, *Smith-Waterman* and *Levenshtein*, come from the Simmetrics library [60]. *PathLength* and *Wu&Palmer* rely on extJWNL [61] which handles the communication between WordNet and our tool. Wikipedia Miner [58] offers an API to browse Wikipedia's articles offline and compute their relatedness [62]. We used this tool on the english version of *Wikipedia* and *Wiktionary* which form the last two heuristics.

In the reminder FMONTO denotes the synthesis technique that applies one of the six ontological heuristics over the BIG.

4.5 An Hybrid Solution: Logic to the Rescue

Instead of operating over the BIG, other logical structures obtained from ϕ can be considered when applying ontological heuristics (see left part of Fig. 4).

Reduced BIG. Because of the transitivity of implication, a BIG is likely to have potentially a large number of edges, out of which many candidates are not parent-child relationships. It is tempting to consider the transitive reduction of a BIG (called reduced BIG in the remainder). The counterpart is that the reduced BIG is incomplete, *i.e.*, the selection of some hierarchies is no longer possible.

Cliques. Another aggressive technique is to consider that features that always logically co-occur are likely to denote mandatory parent-child relationships. Co-occurring features are identified as cliques in the BIG and efficient techniques exist [41]. Cliques can be seen as clusters in which one of the member is possibly a parent feature of the other features.

Logical feature groups. *All* possible feature groups of ϕ can be automatically computed [11, 41] and can be seen as clusters. The computation can be performed over the BIG or the reduced BIG.

4.5.1 Pure logical techniques

Before presenting our hybrid solution (combining ontological and logical techniques), we describe possible techniques that only rely on logic.

A first basic approach, denoted $\mathsf{FMRAND}_{\mathsf{BIG}}$, is to weight the BIG with random values. The optimum branching can be applied afterwards to generate a complete FM. Similarly, it can be used to compute ranking lists or clusters. A second approach, denoted $\mathsf{FMRAND}_{\mathsf{RBIG}}$, is to assign random values over the reduced BIG. A third approach is proposed in [43]. Haslinger et al. proposed a technique (referred as FMFASE in the remainder of the paper) that takes as input a set of products and fully generates an FM. The algorithm is iterative and essentially relies on logical structures exposed above: cliques are randomly unfolded, a reduced BIG is used to select parents, and feature groups (if any) are promoted.

FMFASE has three major drawbacks: (1) as reported in [43], the generated FM may not conform to the input configurations in the case of arbitrary constraints ; (2) the operations for computing logical structures assume the enumeration of the complete set of product. It leads to an exponential blowup ; (3) the feature names and the ontological semantics are not considered during the synthesis.

4.5.2 An hybrid solution

The two first drawbacks – the lack of soundness and the performance issues – prompted us to re-develop their algorithm based this time on state-of-the-art satisfiability techniques [41]. We consider that the algorithm, called FMFASE_{SAT} hereafter, is representative of a pure logical-based technique for fully synthesizing an FM. Yet the third drawback – unawareness of ontological semantics – is not addressed by FMFASE_{SAT}.

We propose an hybrid approach based on both ontological and logical techniques, called $\mathsf{FMONTO}_{\mathsf{LOGIC}}$ in the remainder. The ontological heuristics we develop in Section 4.4 operate this time over the transitive reduction of the BIG (instead of the BIG itself).

5. EVALUATION

We conducted a series of experiments to empirically evaluate our techniques on 123 realistic FMs (see Section 5.1). We aim at answering the following research questions:

RQ1: How effective are our techniques to fully synthesize an FM? Is a fully automated synthesis feasible? The whole synthesis process is performed in only one step without any intervention of the user. It is particularly challenging and the resulting FM may be far from the ground truth.

RQ2: How effective are our techniques to compute ranking lists and clusters and thus assist users? The most likely parents and siblings of a given feature can significantly reduce the number of choices a user has to perform during the interactive selection of a feature hierarchy.

For each of the research question, we compare logical, ontological and hybrid-based techniques exposed in Section 4. We notably explore the possible role of the reduced BIG, cliques and logical feature groups for improving the synthesis – "as such" or as a complement to ontological techniques.

5.1 Experimental Settings

5.1.1 Techniques

Table 1 summarizes the available techniques and when they can be applied (e.g., FMFASE does not provide user support and thus cannot address **RQ2**).

For FMFASE we use the binary provided by the authors [63]. To reduce fluctuations caused by random generation, we performed 100 iterations for $\mathsf{FMRAND}_{\mathsf{BIG}}$ and $\mathsf{FMRAND}_{\mathsf{RBIG}}$ in each of the experiments they are involved. The results are reported as the mean value over 100 iterations.

5.1.2 Data

Our experiments operate over two data sets. From each FM of the data sets, we translate its configuration semantics into a Boolean formula ϕ. The formula serves as input of our procedure. The original hierarchy and feature groups of the FMs constitute a *ground truth* on which we can rely to evaluate the ontological quality of the different algorithms and heuristics.

SPLOT dataset. The SPLOT [44] public repository offers a large set of FMs from different domains created by academics or practitioners. From this repository, we manually selected FMs that are written in English and contain meaningful feature names[1]. It results in 108 FMs with a total of 2374 features. Due to memory consumption issues [43], 15 FMs from the SPLOT data set could not be handled by FMFASE. Therefore, we perform the experiment on the remaining 93 FMs. Overall the experiments utilize a similar dataset than the one used in [43], authorizing a fair comparison with FMFASE.

PCM dataset. *Product Comparison Matrices* (PCMs) compare features of domain specific products and now abound on the internet and in particular in Wikipedia [45]. We gathered 30 FMs with an automated extraction process – in the same vein as the one described in [24]. Each configuration authorized by the extracted FM corresponds to at least one product of the PCM. The structure of the matrix and the Wikipedia pages (sections, headers, etc.) is exploited to produce hierarchies. Cell values (plain text values, *"yes"* value, *"no"* value, etc.) are interpreted in terms of variability. The dataset is challenging for the synthesis procedures since the number of cross-tree constraints and the number of features are rather important, feature names are disparate, and last but not least the depth of the hierarchies is 4 in average.

[1]Essentially we remove FMs with nonsense feature names like F1 or FT22 or written in Spanish. We did not discard FMs containing feature names not recognized by our ontologies

Technique	Logical/ontological	Research questions
FMFASE	Logical over reduced BIG, cliques, feature groups	RQ1
FMFASE_{SAT}	Logical over reduced BIG, cliques, feature groups	RQ1
$\text{FMRAND}_{\text{BIG}}$	Randomization over BIG	RQ1 and RQ2
$\text{FMRAND}_{\text{RBIG}}$	Randomization over reduced BIG	RQ1 and RQ2
FMONTO	Ontological over BIG	RQ1 and RQ2
$\text{FMONTO}_{\text{LOGIC}}$	Hybrid (Ontological over reduced BIG)	RQ1 and RQ2

Table 1: Synthesis techniques used for the experiments

Dataset	# features	# edges in the BIG	# edges in the reduced BIG	Depth of hierarchy
SPLOT (93 FMs)	17.6 (min 9, max 77)	138.9	56.6	3.8
PCM (30 FMs)	72.4 (min 23, max 177)	567.4	318.2	4

Table 2: Properties of FMs (average per FM)

Table 2 presents some statistics of the FMs. To give an overview of the complexity of the synthesis process, we also compute the number of parent candidates for each feature from the BIG of an FM. We have an average of 7.9 (from 0 to 36) parent candidates in SPLOT FMs and 7.8 (from 1 to 86) for PCM-based FMs.

5.2 RQ1: Fully Automated Synthesis

Our goal is to evaluate the effectiveness of a *fully* automated synthesis technique. The resulting synthesized FM should exhibit a feature hierarchy as close as possible to the original hierarchy of the ground truth. We consider that the more the number of common edges between the two hierarchies is, the more effective the technique is. For each input formula/set of configurations of the two data sets, we challenge all the techniques of Table 1 to fully synthesize an FM.

Table 3 reports the percentage of common edges with the ground truth. For each dataset and technique, we compute the *average* (i.e., the sum of the values divided by the number of values) and the *median* (i.e., the middle number in a sorted list of numbers). We split the table in two, clearly separating (1) techniques that operate over the BIG (see Table 3a) from (2) techniques that applies a logical heuristic and operate over the reduced BIG (see Table 3b). There are two hypotheses behind this separation.

(**H1**) Our first hypothesis is that ontological techniques are superior to random or logical heuristics when operating over the *same* logical structure.

H1 Results. In Table 3a all ontological heuristics (FMONTO) outperform $\text{FMRAND}_{\text{BIG}}$. For SPLOT, the best heuristic PathLength improves by 10.7% (average) and 12.3% (median) the effectiveness of $\text{FMRAND}_{\text{BIG}}$. Similar observations are made for the PCM dataset: the best heuristic Wikipedia improves by 10.7% (average) and 13.3% (median) the effectiveness of $\text{FMRAND}_{\text{BIG}}$.

Table 3b also shows that all ontological heuristics (FMONTO) outperform FMFASE, FMFASE_{SAT} or $\text{FMRAND}_{\text{BIG}}$ being on SPLOT or PCM dataset. However the improvement gained by ontological heuristics is less significant than in Table 3a. A possible reason is the use of the reduced BIG (see **H2** below).

(**H2**) Our second hypothesis is that the reduced BIG can improve the effectiveness.

H2 Results. Looking at Table 3a and Table 3b, we can observe that all techniques benefit from the reduction of the BIG. However the improvement is more apparent for $\text{FMRAND}_{\text{BIG}}$. Specifically $\text{FMRAND}_{\text{RBIG}}$ increases by 24.5% (resp. 9.2%) the effectiveness of $\text{FMRAND}_{\text{BIG}}$ in PCM dataset (resp. SPLOT dataset). Comparatively, the improvement of $\text{FMONTO}_{\text{LOGIC}}$ w.r.t. FMONTO is 18.2% (resp. 4.8%) in PCM dataset (resp. SPLOT dataset).

The reduction of the BIG significantly decreases the number of edges, thus favouring a random selection. For SPLOT dataset, 60.4% (in average, 62.5% for the median) of edges are removed from the BIG while being actually parent-child relationships in the ground truths. For PCM dataset, 87.9% (in average, 89.7% for the median) of edges are removed from the BIG while being actually parent-child relationships in the ground truths. In practice the tradeoff between a reduction of the problem space and a less accurate representation clearly favours both approaches.

Another important observation in favour of **H2** is that $\text{FMRAND}_{\text{RBIG}}$, FMFASE and FMFASE_{SAT} outperform FMONTO. Without the use of the reduced BIG, ontological heuristics are beat by randomized or logical-based heuristics, highlighting the prior importance of the logical structure.

Summary for RQ1. The experiment demonstrates that an hybrid approach provides the best support for fully synthesizing an FM. A key aspect is that the reduced BIG significantly improves the effectiveness of all techniques. At best, the percentage of good parent-child relationships in the synthesized FM is 36.7% (for SPLOT) and 45.5% (for PCM). A fully automated synthesis produces FMs far from the ground truth. In practice the hybrid approach could provide a *"by-default"* visualisation of the FM but numerous faulty parent-child relationships (more than a half) need to be reviewed and corrected. Therefore it remains crucial to guide the users when refactoring the faulty FM or interactively selecting a feature hierarchy during the synthesis process.

5.3 RQ2: User Support

Our goal is to evaluate the quality of the ranking lists and the clusters. In this experiment we do not consider FMFASE and FMFASE_{SAT} techniques as they do not provide user support.

5.3.1 Ranking lists

We consider that the ranking lists should place the original parent of the ground truth as close as possible to the top of the list. Specifically we check for each feature that its correct parent from the ground truth appears in the *Top 2* positions of the list. With an average of less than 8 parent candidates per feature for both datasets, we chose to restrict our evaluation to the top 2 positions in order to reduce the impact of random choices. Indeed, it already allows a probability of almost 25% (in average) of having a correct parent for randomized approaches.

Table 4 reports the percentage of correct parents in the Top 2 positions of the ranking lists. As in the previous experiment we compute the average and the median for each dataset and technique. We also separate techniques that operate over the BIG from techniques that operate over the reduced BIG and pose the same two hypothesises.

(**H3**) We hypothesize that ontological techniques are superior to random heuristics when operating over the *same* logical structure.

H3 Results. In Table 4a, FMONTO outperforms $\text{FMRAND}_{\text{BIG}}$. For SPLOT FMs, the best heuristic PathLength improves by 9% (average) and 11.7% (median) the effectiveness of $\text{FMRAND}_{\text{BIG}}$. For the other dataset, the Wikipedia heuristic improves by 10.2% (average) and 12.3% (median) the effectivness of $\text{FMRAND}_{\text{BIG}}$. Table 4b also shows that FMONTO outperforms $\text{FMRAND}_{\text{BIG}}$ on both data sets but the scores are significantly closer. Overall we can conclude the hypothesis **H3** is verified.

(**H4**) We hypothesize that the reduced BIG can improve the quality of ranking lists.

H4 Results. The PCM dataset results in a 15.6% (average) and

Table 3: Effectiveness of full synthesis (percentage of common edges)

(a) Full synthesis with BIG

Data set		Pure logical techniques	Ontological techniques (FMONTO)					
		$FMRAND_{BIG}$	Smith Waterman	Levenshtein	Wu & Palmer	PathLength	Wikipedia	Wiktionary
SPLOT	average	21.2	29.5	27.7	31.4	**31.9**	30.9	29.4
	median	16.7	25.0	22.2	28.6	**30.0**	28.6	28.6
PCM	average	16.6	24.7	19.8	21.2	23.4	**27.3**	23.2
	median	14.9	20.5	18.6	19.1	23.7	**28.2**	20.5

(b) Full synthesis with a reduced BIG (● means that the approach cannot be applied due to performance issues)

Data set		Pure logical techniques			Hybrid techniques ($FMONTO_{LOGIC}$)					
		FMFASE	$FMFASE_{SAT}$	$FMRAND_{RBIG}$	Smith Waterman	Levenshtein	Wu & Palmer	PathLength	Wikipedia	Wiktionary
SPLOT	average	33.8	31.4	30.4	34.9	34.0	36.2	**36.7**	35.7	35.4
	median	27.8	27.3	27.8	30.0	31.8	31.8	34.5	33.3	**34.8**
PCM	average	●	40.8	41.1	44.1	41.2	44.8	44.9	**45.5**	42.5
	median	●	37.5	35.1	37.7	33.3	35.9	**40.5**	39.0	35.1

12.4% improvement of the effectiveness of our best heuristic PathLength. However, for SPLOT, we note that reducing the BIG has negative impact on FMONTO: all ontological heuristics have a lower score. For SPLOT, we also observe that $FMRAND_{RBIG}$ is only slightly better (+0.4% in average) than $FMRAND_{BIG}$.

As a result, the hypothesis **H4** is not verified and the reduced BIG may have the opposite effect. This may be explained by the 39.6% of correct parents brutally removed by the reduction in the SPLOT dataset. However, for the PCM dataset, the transitive reduction removes only 12.1% of correct parents.

Our results are also in contradiction with hypothesis **H2**. In other words, the reduction of the BIG is more effective for the *Top 1*.

5.3.2 Clusters

We consider that the computed clusters should contain either sibling features from the ground truth or siblings with their corresponding parent – in line with the two cases identified in Section 4.3. For example in Figure 3b, {Hosted Service, Local} is a correct cluster because the two features are siblings. {Hosted Service, Local, Hosting} is also a correct cluster because Hosting is the parent of Hosted Service and Local.

For each dataset, we report on the number of clusters, the cluster size, the percentage of correct clusters computed by the techniques, and the number of features in a correct cluster (in average and for the median).

Table 4c shows that FMONTO generates less clusters per FM than $FMRAND_{BIG}$. However, FMONTO produces clusters that are slightly bigger and more accurate. For SPLOT, our best heuristic PathLength FMONTO generates 70.1% (average) and 80% (median) of correct clusters while $FMRAND_{BIG}$ reaches 26.7% (average) and 25% (median). The percentage of features in correct clusters is also significantly better than $FMRAND_{BIG}$. For the PCM dataset, the difference is greater with 79.2% (average) and 78.6% (median) of correct clusters for PathLength and 14.9% (average) and 13% (median) for $FMRAND_{BIG}$.

We also experimented with the reduced BIG (instead of the BIG) for computing clusters. We did not observe significant differences. Details can be found in [64].

Clusters and logical feature groups.[2] Logical heuristics can exploit cliques and feature groups as clusters.

For example FMFASE chooses one feature of a clique and places it as the parent of the others. We observe this pattern is respected in 49% (average) and 50% (median) of the cliques of SPLOT FMs. For PCM FMs, this pattern represents 31.7% (average) and 0% (median) of the cliques. For SPLOT (resp. PCM), we note that only 5.4% (average) and 0% (median) of the cliques (resp. 1.1% and 0%) do not only contain parent-child relations. As a result, cliques almost always contain parent-child relationships but require more user effort than a traditional clique in which only one parent should be selected.

For feature groups, we check that they are correct clusters as defined previously. For SPLOT, they represent 69.6% (average) and 75% (median) of correct clusters. For PCM, they represent 92.5% (average) and 100% (median) of correct clusters. If we compute the feature groups over the reduced BIG, it significantly reduces the number of groups but increases their accuracy. Features groups are thus good candidates for clusters, but induce a lot of false positives.

Summary for RQ2. The experiment demonstrates that an approach based on ontological heuristics provides the best support for producing ranking lists. At best, the user has to review only 2 parent candidates for 52.4% of the features (for SPLOT) and 56.3% (for PCM). An approach based on the reduced BIG does not necessarily improve the quality of the ranking lists. The experiment also demonstrates that an ontological approach produces less clusters than $FMRAND_{BIG}$ but they are more accurate. Logical clusters such as cliques and feature groups can also be used to reduce the user effort. Feature groups form accurate clusters especially when they are computed over the reduced BIG. Cliques require complex unfolding in more than half of the cases.

5.4 Threats to Validity

Threats to *external validity* are conditions that limit our ability to generalize the synthesis results to other forms of dependencies or feature names. A first concern is whether FMs are representative of practice. We use the SPLOT public repository [44]. SPLOT is a common benchmark for the FM community (see, e.g., [13, 14, 16, 29, 38, 40, 41, 43, 65]) and is considered to manage *"realistic"* examples by several authors. We also diversify the dataset with the use of PCMs.

Our major concern is whether FMs (from SPLOT or PCMs) are good ground truths w.r.t ontological semantics. Indeed, an unique characteristic of our work is that we do consider the ontological semantics of the FMs. From this respect, it is hard to certify that the chosen FMs are of good quality. The fact that FMs of SPLOT come from academic publications and practitioners is certainly a good point, but not a guarantee. A possible improvement is a manual review of the FMs, at least to discard FMs with nonsense feature names, at best to possibly improve FMs. The ontological seman-

[2] Details of the results about cliques and logical feature groups can be found in [64]

Table 4: Percentage of correct parents in the top 2 positions of the ranking lists (RQ2)

(a) With BIG

Data set		Pure logical technique	Hybrid techniques (FMONTO)					
		$\text{FMRAND}_{\text{BIG}}$	Smith Waterman	Levenshtein	Wu & Palmer	PathLength	Wikipedia	Wiktionary
SPLOT	average	43.4	51.6	48.7	50.7	**52.4**	52.3	50.9
	median	38.3	50.0	50.0	50.0	50.0	**55.6**	51.2
PCM	average	33.3	40.9	38.4	37.9	40.7	**43.5**	38.4
	median	32.1	38.8	36.2	37.5	42.1	**44.4**	36.4

(b) With reduced BIG

Data set		Pure logical technique	Hybrid techniques ($\text{FMONTO}_{\text{LOGIC}}$)					
		$\text{FMRAND}_{\text{RBIG}}$	Smith Waterman	Levenshtein	Wu & Palmer	PathLength	Wikipedia	Wiktionary
SPLOT	average	43.8	48.0	47.9	47.9	48.1	**49.1**	48.2
	median	43.0	44.4	45.5	45.5	44.4	**47.7**	44.4
PCM	average	52.0	54.9	53.6	55.6	**56.3**	56.0	53.2
	median	47.8	54.4	48.3	56.1	54.5	55.3	**57.9**

(c) Clusters generated by $\text{FMRAND}_{\text{BIG}}$ and FMONTO

Metric	Data set		Pure logical technique	Ontological techniques (FMONTO)					
			$\text{FMRAND}_{\text{BIG}}$	Smith Waterman	Levenshtein	Wu & Palmer	PathLength	Wikipedia	Wiktionary
Number of clusters	SPLOT	average	**6.2**	4.1	4.1	2.7	2.4	3.3	2.8
		median	**5**	4	4	2	2	3	3
	PCM	average	**29.7**	16.8	17.6	5.8	8.7	9.8	12.3
		median	**25**	16	15	5	7	8	10
Clusters'size	SPLOT	average	2.2	2.8	2.8	2.3	2.2	**2.9**	2.3
		median	2.1	2.6	2.5	2.0	2.0	**2.7**	2.2
	PCM	average	2.3	3.3	2.8	2.6	2.6	**5.8**	3.0
		median	2.3	3.1	2.8	2.5	2.4	**5.2**	2.9
Percentage of Correct clusters	SPLOT	average	26.7	50.8	52.0	57.9	**70.1**	54.6	64.2
		median	25.0	50.0	50.0	50.0	**80.0**	50.0	66.7
	PCM	average	14.9	49.6	62.6	60.8	**79.2**	53.7	71.0
		median	13.0	48.0	66.7	66.7	**78.6**	57.1	66.7
Percentage of features in a correct cluster	SPLOT	average	18.2	29.2	**30.0**	18.8	22.0	26.6	23.5
		median	15.4	23.5	**29.4**	18.2	18.2	20.0	20.0
	PCM	average	12.5	35.0	**41.4**	12.9	22.8	24.5	32.2
		median	10.8	32.4	**41.5**	12.5	23.3	22.2	32.6

tics of FMs we extract from PCMs is aligned with the structure of Wikipedia pages and the matrix itself but would also benefit from an external review, *e.g.*, by another pool of researcher.

Another external threat is that we hypothesize that the user effort is reduced thanks to clusters computation, ranking lists and the branching algorithm. Yet we have not run user experiments to validate this claim. This evaluation is our immediate concern.

A first *internal threat* is that the extraction of FMs from PCMs is a mix between automated techniques and manual directives. This creates a threat of potential bias, since the author knew the procedures that were to be evaluated against this model. We take care of retaining the original structure of the PCMs. The manual intervention essentially consists in removing unnecessary columns (like the version number, website or developer name of a product). Our interpretation of variability remains fixed for all the PCMs (e.g., we interpret a *"No"* value in a cell as an absence of a feature). Another interpretation of variability can lead to a different set of dependencies and may disturb some heuristics (e.g., the use of the reduced BIG). We plan to further investigate this hypothesis in the future. Moreover, as we apply part of our procedures to Wikipedia PCMs dataset, one might perceive that some of the heuristics, based also on Wikipedia, are biased. However the heuristics do not operate over Wikipedia pages where we extracted the PCM. We exploit Wikipedia as a general ontology.

Another internal threat comes from the manual optimization of the clustering thresholds for the evaluation of the heuristics. Another set of thresholds could generate less favourable results. It is unclear whether this difference would be significant.

Finally we implement various heuristics and procedures for synthesizing FMs or collecting statistics. Their implementation may be incorrect. We thoroughly tested our infrastructure using test cases and reuse as much as possible existing codes [11, 15, 41].

6. RELATED WORK

We discuss the differences and synergies between existing works and our proposal.

6.1 FM synthesis

Techniques for synthesising an FM from a set of dependencies (e.g., encoded as a propositional formula) or from a set of configurations (e.g., encoded in a product comparison matrix) have been proposed [11, 20, 29, 38, 39, 41–43, 66]. An important limitation of prior works is the identification of the feature hierarchy when synthesizing the FM (i.e., the user support is either absent or limited). In [11, 41], the authors calculate a diagrammatic representation of *all possible* FMs, leaving open the selection of the hierarchy and feature groups. The procedure exposed in [66] for probabilistic FMs does not offer ontological support either. In [29], we proposed a synthesis procedure that processes user-specified knowledge for organizing the hierarchy of features. Janota *et al.* [39] propose an interactive environment to guide user in synthesizing an FM. In both cases [29, 39], the effort may be substantial since users have to review numerous potential parent features (7.9 in average for the FMs of the SPLOT repository, see Section 5).

Our experiments further the understanding of strengths and weaknesses of logical components computed by these approaches. From this perspective, the feature graph structure presented in [41] has interesting properties.

The algorithms proposed in [38, 42, 43] do not control the way the feature hierarchy is synthesized in the resulting FM. We demonstrated in Section 5.2 that the ontological semantics of the resulting FMs significantly deviates from the ground truths while an hybrid

approach provides better results. In addition no user support is provided to interactively synthesize or refactor the resulting FM.

Davril *et al.* [67] presented a fully automated approach, based on prior work [68], for constructing FMs from publicly available product descriptions found in online product repositories and marketing websites such as SoftPedia and CNET. The proposal is evaluated in the anti-virus domain.

A key difference is the presence of textual documents to mine and organize features. Our techniques operate over a predefined set of features and dependencies. We do not assume any additional inputs for selecting the feature hierarchy (see next section for a discussion). Moreover no user support is provided to refactor the resulting FM or breath ontological knowledge during the synthesis procedure. It could help users to improve the quality of FMs, i.e., closer to the quality of reference FMs manually specified by participants of the experiment [67].

She *et al.* [20] proposed an heuristic to rank the correct parent features in order to reduce the task of a user. Though the synthesis procedure is generic and similar to ours, the ontological heuristics are specific to the operating system domain. We develop a series of alternative techniques for computing ranking lists and also clusters, applicable to any domain. Another difference is the existence of feature descriptions in the software projects Linux, eCos, and FreeBSD. She *et al.* reported in Section 7 of [20] that their attempts to fully synthesize an FM did not lead to a desirable hierarchy – such as the one from reference FMs used in their evaluation – coming to the conclusion that an additional expert input is needed. *Our evaluation confirms that a full synthesis is not adequate and that the user support is highly needed.*

6.2 FM extraction

In [24], a semi-automated procedure to support the transition from product descriptions (expressed in a tabular format) to FMs is proposed. In [25], architectural knowledge, plugins dependencies and the slicing operator are combined to obtain an exploitable and maintainable FM ; in particular the feature hierarchy reflects the hierarchical structure of the system. Ryssel et al. developed methods based on Formal Concept Analysis and analyzed incidence matrices containing matching relations [26].

The procedures exposed in [20,24–26,67] are specific to a project or a domain and assume the existence of a certain structure or artefacts (e.g., textual corpus, hierarchical model of the architecture) to organize the features. We cannot make similar assumptions in the general case. First, there are sometimes no opportunity to exploit artefacts or knowledge. In the case of the SPLOT repository, there are no artefacts (e.g., feature descriptions) associated to FMs. Another example is when the list of features is flattened and no prior ontological knowledge can be inferred as is the case in the matrix of Fig. 3a (cf page 4). Ontological knowledge is also missing when extracting conditional compilation directives from source code and build files [27]. Second, procedures to extract ontological knowledge are specifically developed or customized to a project, requiring a substantial effort.

Our solution can be seen as an agnostic, lightweight method to breath ontological knowledge into FM synthesis, reusable in every projects or domains.

Alves *et al.* [69], Niu *et al.* [70], Weston *et al.* [23] and Chen *et al.* [71] applied information retrieval techniques to abstract requirements from existing specifications, typically expressed in natural language. These works do not consider precise logical dependencies and solely focus on ontological semantics. As a result users have to manually set the variability information. Moreover a risk is to build an FM in contradiction with the actual dependencies of a system. Bagheri *et al.* [72] proposed a collaborative process to mine and organize features using a combination of natural language processing techniques and Wordnet.

The techniques exposed in this section are complementary to our proposals, since they compute ontological knowledge that can be incorporated into our FM synthesis support.

7. CONCLUSION

In this paper, we addressed the problem of synthesising a feature model (FM) conformant to a set of dependencies and also exhibiting an appropriate ontological semantics as defined by its hierarchy and feature groups. This problem is crucial for software product line (re-)engineering scenarios involving reverse engineering, slicing, or refactoring of FMs.

We developed and evaluated a series of automated techniques, applicable without prior knowledge or artefacts, to breath ontological knowledge into FM synthesis. Our empirical evaluation on 123 FMs, coming from various domains, demonstrated that an hybrid approach, mixing ontological and logical techniques, provides the best support – either for fully synthesizing FMs or for assisting users through ranking lists and clusters.

The data, code, and instructions for reproducing the results are available online `http://tinyurl.com/OntoFMExperiments` and act as a baseline for comparison. Based on our findings, we developed an ontologic-aware synthesis environment that allows important automated operations for FMs with ontological capabilities and guides the users throughout the interactive process. More details can be found in `http://tinyurl.com/OntoFMEnv`.

Our immediate concern is to conduct user experiments to evaluate the effort saved by our procedures in practical reverse engineering or maintenance settings. Another research direction is to exploit *specific* information sources and artefacts that may be present in software projects to automatically capture and breath ontological knowledge. Meanwhile, we can hope that *generic* ontologies (like Wordnet) and open, collaborative-based initiatives (like Wikipedia) will be enriched to cover more and more technical domains. Future work could also generalize our ontological-aware synthesis to FMs with attributes and multi-features [73].

8. REFERENCES

[1] F. J. v. d. Linden, K. Schmid, and E. Rommes, *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.

[2] "Sei's software product line hall of fame."

[3] K. Pohl, G. Böckle, and F. J. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag, 2005.

[4] S. Apel and C. Kästner, "An overview of feature-oriented software development," *Journal of Object Technology (JOT)*, vol. 8, no. 5, pp. 49–84, July/August 2009.

[5] S. Apel, D. Batory, C. Kästner, and G. Saake, *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer-Verlag, 2013.

[6] G. Mussbacher, J. Araújo, A. Moreira, and D. Amyot, "Aourn-based modeling and analysis of software product lines," *Software Quality Journal*, vol. 20, no. 3-4, pp. 645–687, 2012.

[7] T. Berger, S. She, R. Lotufo, A. Wasowski, and K. Czarnecki, "A study of variability models and languages in the systems software domain," *IEEE Transactions on Software Engineering*, vol. 99, no. PrePrints, p. 1, 2013.

[8] Berger, Thorsten and Rublack, Ralf and Nair, Divya and Atlee, Joanne M. and Becker, Martin and Czarnecki, Krzysztof and Wasowski, Andrzej, "A survey of variability modeling in industrial practice," in *VaMoS'13*. ACM, 2013.

[9] A. Hubaux, T. T. Tun, and P. Heymans, "Separation of concerns in feature diagram languages: A systematic survey (to appear)," *ACM Computing Surveys*, 2013.

[10] K. Kang, J. Lee, and P. Donohoe, "Feature-oriented product line engineering," *Software, IEEE*, vol. 19, no. 4, pp. 58–65, 2002.

[11] K. Czarnecki and A. Wasowski, "Feature diagrams and logics: There and back again," in *SPLC'07*. IEEE, 2007, pp. 23–34.

[12] T. Thüm, D. Batory, and C. Kästner, "Reasoning about edits to feature models," in *ICSE'09*. ACM, 2009, pp. 254–264.

[13] R. Pohl, K. Lauenroth, and K. Pohl, "A performance comparison of contemporary algorithmic approaches for automated analysis operations on feature models," in *ASE'11*, 2011, pp. 313–322.

[14] R. Pohl, V. Stricker, and K. Pohl, "Measuring the structural complexity of feature models," in *ASE'13*, 2013.

[15] M. Acher, P. Collet, P. Lahire, and R. France, "Familiar: A domain-specific language for large scale management of feature models," *Science of Computer Programming (SCP)*, vol. 78, no. 6, pp. 657–681, 2013.

[16] A. S. Sayyad, T. Menzies, and H. Ammar, "On the value of user preferences in search-based software engineering: a case study in software product lines," in *ICSE'13*, 2013, pp. 492–501.

[17] K. Czarnecki, C. H. P. Kim, and K. T. Kalleberg, "Feature models are views on ontologies," in *SPLC '06*. IEEE, 2006, pp. 41–51.

[18] M. T. Valente, V. Borges, and L. Passos, "A semi-automatic approach for extracting software product lines," *IEEE Transactions on Software Engineering*, vol. 38, no. 4, pp. 737–754, 2012.

[19] C. Kästner, A. Dreiling, and K. Ostermann, "Variability mining: Consistent semiautomatic detection of product-line features," *IEEE Transactions on Software Engineering*, 2013, (to appear).

[20] S. She, R. Lotufo, T. Berger, A. Wasowski, and K. Czarnecki, "Reverse engineering feature models," in *ICSE'11*. ACM, 2011, pp. 461–470.

[21] J. Rubin and M. Chechik, "Locating distinguishing features using diff sets," in *ASE'12*. ACM, 2012, pp. 242–245.

[22] ——, *Domain Engineering: Product Lines, Conceptual Models, and Languages*. Springer, 2013, ch. A Survey of Feature Location Techniques.

[23] N. Weston, R. Chitchyan, and A. Rashid, "A framework for constructing semantically composable feature models from natural language requirements," in *SPLC'09*. ACM, 2009, pp. 211–220.

[24] M. Acher, A. Cleve, G. Perrouin, P. Heymans, C. Vanbeneden, P. Collet, and P. Lahire, "On extracting feature models from product descriptions," in *VaMoS'12*. ACM, 2012, pp. 45–54.

[25] M. Acher, A. Cleve, P. Collet, P. Merle, L. Duchien, and P. Lahire, "Extraction and evolution of architectural variability models in plugin-based systems," *Software and Systems Modeling (SoSyM)*, 2013.

[26] U. Ryssel, J. Ploennigs, and K. Kabitzsch, "Extraction of feature models from formal contexts," in *FOSD'11*, 2011, pp. 1–8.

[27] C. Dietrich, R. Tartler, W. Schröder-Preikschat, and D. Lohmann, "A robust approach for variability extraction from the linux build system," in *SPLC'12*, 2012, pp. 21–30.

[28] A. Rabkin and R. Katz, "Static extraction of program configuration options," in *ICSE'11*. ACM, 2011, pp. 131–140.

[29] M. Acher, P. Heymans, A. Cleve, J.-L. Hainaut, and B. Baudry, "Support for reverse engineering and maintaining feature models," in *VaMoS'13*. ACM, 2013.

[30] K. Czarnecki and K. Pietroszek, "Verifying feature-based model templates against well-formedness ocl constraints," in *GPCE'06*. ACM, 2006, pp. 211–220.

[31] S. Thaker, D. Batory, D. Kitchin, and W. Cook, "Safe composition of product lines," in *GPCE '07*. New York, NY, USA: ACM, 2007, pp. 95–104.

[32] A. Classen, P. Heymans, P.-Y. Schobbens, A. Legay, and J.-F. Raskin, "Model checking lots of systems: efficient verification of temporal properties in software product lines," in *ICSE'10*. ACM, 2010, pp. 335–344.

[33] A. Classen, P. Heymans, P.-Y. Schobbens, and A. Legay, "Symbolic model checking of software product lines," in *ICSE'11*. ACM, 2011, pp. 321–330.

[34] S. Apel, A. von Rhein, P. Wendler, A. Größlinger, and D. Beyer, "Strategies for product-line verification: Case studies and experiments," in *ICSE'13*. IEEE, 2013.

[35] F. Heidenreich, P. Sanchez, J. Santos, S. Zschaler, M. Alferez, J. Araujo, L. Fuentes, U. K. amd Ana Moreira, and A. Rashid, "Relating feature models to other models of a software product line: A comparative study of featuremapper and vml*," *Transactions on Aspect-Oriented Software Development VII, Special Issue on A Common Case Study for Aspect-Oriented Modeling*, vol. 6210, pp. 69–114, 2010.

[36] K. Czarnecki and U. Eisenecker, *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.

[37] A. Metzger, K. Pohl, P. Heymans, P.-Y. Schobbens, and G. Saval, "Disambiguating the documentation of variability

in software product lines: A separation of concerns, formalization and automated analysis," in *RE'07*, 2007, pp. 243–253.

[38] E. N. Haslinger, R. E. Lopez-Herrejon, and A. Egyed, "Reverse engineering feature models from programs' feature sets," in *WCRE'11*. IEEE, 2011, pp. 308–312.

[39] M. Janota, V. Kuzina, and A. Wasowski, "Model construction with external constraints: An interactive journey from semantics to syntax," in *MODELS'08*, ser. LNCS, vol. 5301, 2008, pp. 431–445.

[40] L. Yi, W. Zhang, H. Zhao, Z. Jin, and H. Mei, "Mining binary constraints in the construction of feature models," in *RE'12*. IEEE, 2012, pp. 141–150.

[41] N. Andersen, K. Czarnecki, S. She, and A. Wasowski, "Efficient synthesis of feature models," in *Proceedings of SPLC'12*. ACM, 2012, pp. 97–106.

[42] R. E. Lopez-Herrejon, J. A. Galindo, D. Benavides, S. Segura, and A. Egyed, "Reverse engineering feature models with evolutionary algorithms: An exploratory study," in *SSBSE'12*, ser. LNCS, vol. 7515. Springer, 2012, pp. 168–182.

[43] E. N. Haslinger, R. E. Lopez-Herrejon, and A. Egyed, "On extracting feature models from sets of valid feature combinations," in *FASE'13*, ser. LNCS, vol. 7793, 2013, pp. 53–67.

[44] SPLOT: Software Product Line Online Tools, "http://www.splot-research.org/."

[45] N. Sannier, M. Acher, and B. Baudry, "From Comparison Matrix to Variability Model: The Wikipedia Case Study," in *ASE'13*. IEEE, 2013.

[46] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, and Y. Bontemps, "Generic semantics of feature diagrams," *Comput. Netw.*, vol. 51, no. 2, pp. 456–479, 2007.

[47] Q. Boucher, E. Abbasi, A. Hubaux, G. Perrouin, M. Acher, and P. Heymans, "Towards more reliable configurators: A re-engineering perspective," in *PLEASE'12 Int'l workshop at ICSE'12*, ser. , 2012.

[48] S. Apel, C. Kästner, and C. Lengauer, "Language-independent and automated software composition: The featurehouse experience," *IEEE Transactions on Software Engineering (TSE)*, vol. 39, pp. 63–79, 2013.

[49] A. Hubaux, M. Acher, T. T. Tun, P. Heymans, P. Collet, and P. Lahire, *Domain Engineering: Product Lines, Conceptual Models, and Languages*. Springer, 2013, ch. Separating Concerns in Feature Models: Retrospective and Multi-View Support.

[50] R. E. Tarjan, "Finding optimum branchings," *Networks*, vol. 7, no. 1, pp. 25–35, 1977.

[51] P. Camerini, L. Fratta, and F. Maffioli, "A note on finding optimum branchings," *Networks*, vol. 9, no. 4, pp. 309–312, 1979.

[52] T. Smith and M. Waterman, "Identification of common molecular subsequences," *Molecular Biology*, vol. 147, pp. 195–197, 1981.

[53] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *Journal of the ACM (JACM)*, vol. 21, no. 1, pp. 168–173, 1974.

[54] G. A. Miller, "Wordnet: a lexical database for english," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[55] A. Budanitsky and G. Hirst, "Evaluating wordnet-based measures of lexical semantic relatedness," *Computational Linguistics*, vol. 32, no. 1, pp. 13–47, 2006.

[56] Z. Wu and M. Palmer, "Verbs semantics and lexical selection," in *the 32nd annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 1994, pp. 133–138.

[57] O. Wulf-Hadash and I. Reinhartz-Berger, "Cross product line analysis," in *VaMoS'13*. ACM, 2013.

[58] D. N. Milne and I. H. Witten, "An open-source toolkit for mining wikipedia," *Artif. Intell.*, vol. 194, pp. 222–239, 2013.

[59] O. Medelyan, D. N. Milne, C. Legg, and I. H. Witten, "Mining meaning from wikipedia," *Int. J. Hum.-Comput. Stud.*, vol. 67, no. 9, pp. 716–754, 2009.

[60] SimMetrics, "http://sourceforge.net/projects/simmetrics."

[61] extJWNL, "http://extjwnl.sourceforge.net."

[62] D. Milne, "Computing semantic relatedness using wikipedia link structure," in *the new zealand computer science research student conference*. Citeseer, 2007.

[63] Algorithm of Haslinger et al. [43], "http://www.jku.at/sea/content/e139529/e126342/e188736/."

[64] Companion web page with all the results, data, code, and instructions to reproduce results, "http://tinyurl.com/OntoFMExperiments."

[65] M. Mendonca, A. Wasowski, and K. Czarnecki, "SAT-based analysis of feature models is easy," in *SPLC'09*. IEEE, 2009, pp. 231–240.

[66] K. Czarnecki, S. She, and A. Wasowski, "Sample spaces and feature models: There and back again," in *SPLC'08*, 2008, pp. 22–31.

[67] J.-M. Davril, E. Delfosse, N. Hariri, M. Acher, J. Cleland-Huang, and P. Heymans, "Feature model extraction from large collections of informal product descriptions," in *ESEC/FSE'13*, 2013.

[68] N. Hariri, C. Castro-Herrera, M. Mirakhorli, J. Cleland-Huang, and B. Mobasher, "Supporting domain analysis through mining and recommending features from online product listings," *IEEE Transactions on Software Engineering*, vol. 99, no. PrePrints, p. 1, 2013.

[69] V. Alves, C. Schwanninger, L. Barbosa, A. Rashid, P. Sawyer, P. Rayson, C. Pohl, and A. Rummler, "An exploratory study of information retrieval techniques in domain analysis," in *SPLC'08*. IEEE, 2008, pp. 67–76.

[70] N. Niu and S. M. Easterbrook, "Concept analysis for product line requirements," in *AOSD'09*, K. J. Sullivan, A. Moreira, C. Schwanninger, and J. Gray, Eds. ACM, 2009, pp. 137–148.

[71] K. Chen, W. Zhang, H. Zhao, and H. Mei, "An approach to constructing feature models based on requirements clustering," in *RE'05*, 2005, pp. 31–40.

[72] E. Bagheri, F. Ensan, and D. Gasevic, "Decision support for the software product line domain engineering lifecycle," *Automated Software Engineering*, vol. 19, no. 3, pp. 335–377, 2012.

[73] M. Cordy, P.-Y. Schobbens, P. Heymans, and A. Legay, "Beyond boolean product-line model checking: dealing with feature attributes and multi-features," in *ICSE'13*, 2013, pp. 472–481.

APPENDIX

The appendix aims at complementing Section 5.3.2 where we discuss the effectiveness of ontological-based and logical-based techniques for computing clusters. We report on further empirical results. We first analyze the effect of the transitive reduction of the BIG when computing clusters (see Appendix A). We also present results about cliques (see Appendix B) and logical feature groups (see Appendix C) – two logical structures that can be considered as clusters.

A. CLUSTERS ON THE REDUCED BIG

Instead of operating over the complete BIG, ontological-based or randomized heuristics can operate over a reduced BIG in order to compute clusters.

Results

Table 5 (see page 16) shows that the reduced BIG allows to produce less clusters per FM compared to the results of Section 5.3.2 with the complete BIG. The clusters are also slightly smaller. For SPLOT, 76.2% (average) and 100% (median) of clusters are correct with our best heuristic Wiktionary. For PCM, the PathLength heuristic produces 89.1% (average) and 91.7% (median) of correct clusters.

Therefore, and for every ontological-based heuristics, the accuracy of the generated clusters increases when we reduce the BIG. However, the percentage of features in a correct cluster is slightly inferior compared to the results with a complete BIG. On a complete BIG (see Table 4c), our best heuristic Levenshtein produces 30% (average) and 29.4% (median) of correct clusters for SPLOT (resp. 41.4% and 41.5% for PCM). On a reduced BIG, Levenshtein produces 27.1% (average) and 24.1% (median) for SPLOT (reps. 40.5% and 41.5% for PCM).

Conclusion

The results show that there is no clear superiority of an approach. The use of the reduced BIG or the use of the BIG when computing clusters have both pros and cons. On the one hand, the reduced BIG has the advantage of being more accurate but less clusters with less features are computed. On the other hand, the use of the complete BIG provides more false positives but a user can consult and manipulate larger clusters. From a practical and tooling point of view, there is a classical tradeoff to find between precision and recall.

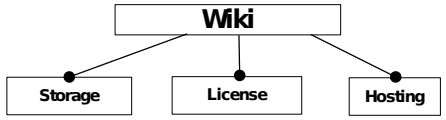
Finally, we note that $\mathsf{FMONTO}_{\mathsf{LOGIC}}$ outperforms $\mathsf{FMRAND}_{\mathsf{RBIG}}$ for all the results of Table 5. It confirms the usefulness of ontological-based heuristics, whether they operate over a reduced BIG or a BIG.

B. CLIQUES

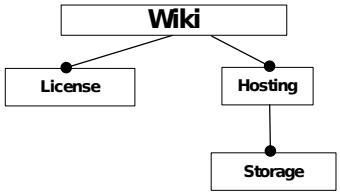
Features that co-occur in configurations (i.e., cliques) can be efficiently computed using standard logical techniques [41]. As previously reported in Section 5.3.2, cliques almost always represent parent-child relations between features[3]. Therefore cliques can be seen as special kinds of clusters.

Unfolding of cliques

In the most favourable case, users just have to select one feature in the clique that will play the role of parent feature of the others.



(a) Simple unfolding : one parent and one level of decendants



(b) Complex unfolding : one parent and several levels of descendants

Figure 5: Clique unfolding: simple versus complex

For instance, the clique {Wiki, Storage, License, Hosting} is transformed through a *simple unfolding* in Figure 5a: Wiki is the parent of Storage, License and Hosting. However, more *complex unfolding* may arise. We could have decided that the Storage is part of the Hosting concern. Thus, Storage would be below Hosting (and not Wiki). In that case, the clique requires a complex unfolding that consists in defining several levels of features in the hierarchy (see Figure 5b for an example).

User effort

Our empirical experience shows that complex unfolding is required in at least half of the cases (49% in average for SPLOT, 31.7% in average for PCMs, see Table 6a page 16). Therefore, using cliques requires more user effort than a traditional cluster in which only one parent should be selected.

Theoretical benefits

Table 6a, column "Cliques" (see page 16) shows the metrics (number of clusters, clusters' size, percentage of correct clusters, percentage of features in a correct cluster) applied on the cliques. The goal is to understand the possible benefits of using cliques as clusters[4].

We first observe that the number of cliques is low: 1.6 (resp. 1.4) in average for SPLOT (resp. PCM). As a comparison, the number of clusters we obtain with ontological heuristics can be superior to 4 in average for SPLOT, to 17 in average for PCMs. A second related observation is that features involved in cliques are not sufficient per se. For SPLOT, features involved in cliques represent only 16.6% (average) and 15% (median) of the total features in the original FM. Comparatively, our clusters reaches 30% (average) and 29.4% (median) (see Table 4c, page 10). For PCM, the correct cliques represent only 2.9% (average) and 0% (median) of the FMs' features while our clusters reaches 41.4% (average) and 41.5% (median).

Conclusion

We can conclude that cliques represent an interesting logical structure but *i)* their numbers are typically low, thus are far for being sufficient for fully completing an interactive synthesis and *ii)* users have to perform complex unfolding in a significant number of cases. On the other hand, clusters computed with ontological-based

[3]We recall that for SPLOT (resp. PCM), only 5.4% (average) and 0% (median) of the cliques (resp. 1.1% and 0%) do not only contain parent-child relations and are represented as bi-implication cross-tree constraints.

[4]As a reminder and for the sake of comparison, Table 4c, page 10 shows the results of our ontological heuristics when computing clusters.

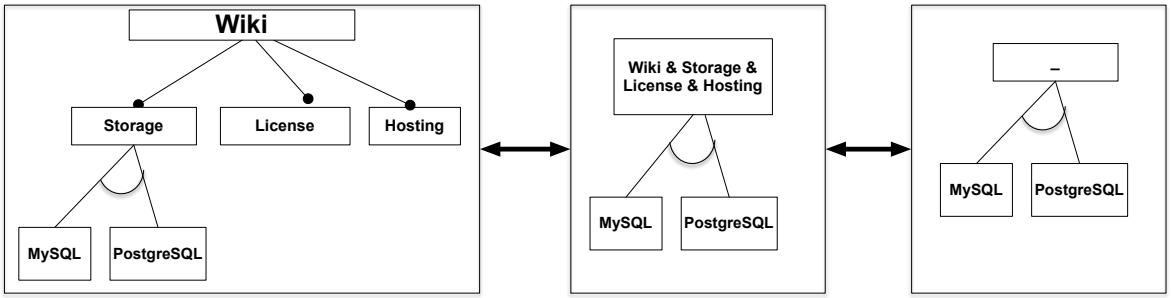


Figure 6: Logical feature groups: bi-implied features, clique contraction, parent place-holder

heuristics or logical feature groups (see next Appendix) require a more simple unfolding and represent a larger source of information.

C. LOGICAL FEATURE GROUPS

There exists different kinds of clusters: *i)* clusters computed with ontological-based heuristics either with the BIG or the reduced BIG, *ii)* cliques, and *iii)* feature groups. Indeed, a feature group consists in a set of sibling features and a parent feature – in line with the definition of cluster given in the paper.

All *possible* feature groups of a formula can be computed either using the reduced BIG [41] or the BIG. We study here how the so-called *logical feature groups* are interesting structures for representing clusters. The expected benefit is that users can review logical feature groups and decide to integrate some of the them in the feature diagram.

A place-holder for logical feature groups

There are cases in which the group's parent is logically bi-implied by another feature (i.e., the parent feature belongs to a clique). An example is given in Figure 6. Therefore, it leads to numerous feature groups containing the same set of sibling features but different parents. To avoid this combinatorial explosion, we omit the parent and consider only the sibling features. We introduce a place-holder that can be one of the parent of the clique (see right of Figure 6).

Results and observations

Table 6a shows the metrics for clusters (see Section 5.3.2) applied on these groups. For SPLOT FMs, 69.6% (average) and 75% (median) of feature groups are correct clusters. These results are similar to the 70.1% (average) and 80% (median) of correct clusters generated by our best heuristic PathLength (see Table 4c). For PCM FMs, 92.5% (average) and 100% (median) of feature groups are correct clusters. It outperforms the 79.2% (average) and 78.6% (median) of correct clusters from our heuristic PathLength. However, we note that 473 Xor groups were generated from a unique FM and 36 only were correct clusters. This extreme example shows that, in some cases, logical feature groups may overwhelm the user with incorrect clusters.

Table 6b shows the same metrics for feature groups computed from the reduced BIG of the FM. It significantly reduces the number of computed feature groups. Their accuracy increases compared to the groups in Table 6a but the correct groups involve less features.

Summary

Logical feature groups are good candidates for clusters. They allow to reduce the user effort assuming that he/she correctly replaces the "place-holder" with the right parent. However, in some cases, logical feature groups may introduce a lot of false positives.

Metric	Data set		Pure logical technique	Ontological techniques (FMONTO$_{\text{LOGIC}}$)					
			FMRAND$_{\text{RBIG}}$	Smith Waterman	Levenshtein	Wu & Palmer	PathLength	Wikipedia	Wiktionary
Number of clusters	SPLOT	average	**3.9**	2.8	2.9	2.1	1.8	2.4	2.1
		median	**4**	3	3	2	2	2	2
	PCM	average	12.7	11.1	**13.9**	4.7	7.5	7.1	9.7
		median	12	10	**13**	5	7	6	9
Clusters'size	SPLOT	average	2.0	**2.4**	**2.4**	2.1	1.9	2.3	2.0
		median	2.0	2.3	**2.4**	2.0	2.0	2.3	2.0
	PCM	average	2.2	3.0	2.7	2.4	2.5	**4.1**	2.9
		median	2.2	2.8	2.8	2.4	2.3	**3.7**	2.7
Percentage of Correct clusters	SPLOT	average	39.9	69.1	65.4	66.7	75.7	67.2	**76.2**
		median	33.3	75.0	66.7	66.7	**100.0**	75.0	**100.0**
	PCM	average	37.8	68.1	76.6	75.3	**89.1**	74.9	85.2
		median	29.4	75.0	83.3	77.8	**91.7**	80.0	88.9
Percentage of features in a correct cluster	SPLOT	average	17.1	26.1	**27.1**	17.5	19.2	23.9	20.5
		median	15.4	21.4	**24.1**	16.7	15.4	20.0	16.7
	PCM	average	12.2	34.1	**40.5**	12.5	22.4	24.1	31.7
		median	10.5	32.3	**41.5**	12.3	23.1	21.6	32.6

Table 5: Clusters generated by FMRAND$_{\text{RBIG}}$ and FMONTO$_{\text{LOGIC}}$

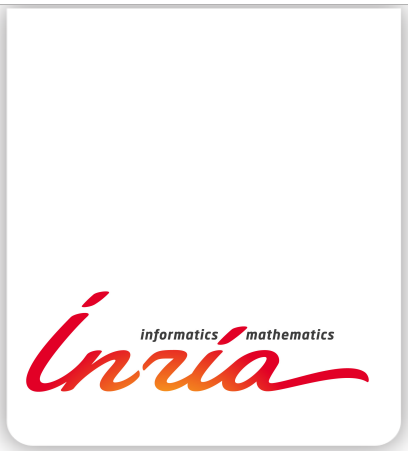
Metric	Data set		Cliques	Feature groups			
				All groups	Mutex	Xor	Or
Number of clusters	SPLOT	average	1.6	11.8	3.5	7.2	1.1
		median	1	3	1	1	1
	PCM	average	1.4	63.2	45.4	17.9	●
		median	1.0	9	8	1	●
Clusters'size	SPLOT	average	5.1	2.4	1.3	1.7	1.8
		median	4.0	2.3	2.0	2.0	2.0
	PCM	average	6.3	6.0	5.9	5.5	●
		median	6.0	5.0	5.3	5.0	●
Percentage of correct clusters	SPLOT	average	49.0	69.6	34.5	89.1	81.0
		median	50.0	75.0	14.3	100	100
	PCM	average	31.7	92.5	92.1	91.1	●
		median	0.0	100.0	100.0	100.0	●
Percentage of features in a correct cluster	SPLOT	average	16.6	36.6	5.4	19.4	11.8
		median	15.0	33.3	0.0	15.0	10.5
	PCM	average	2.9	70.1	54.0	16.1	●
		median	0.0	71.7	56.6	14.9	●

(a) With the BIG (● means that the clusters cannot be computed due to performance issues)

Metric	Data set		Feature groups			
			All groups	Mutex	Xor	Or
Number of clusters	SPLOT	average	3.7	0.5	2.1	1.1
		median	2	0	1	1
	PCM	average	10.4	8.4	2.0	●
		median	8	5	1	●
Clusters'size	SPLOT	average	2.3	0.6	1.5	1.8
		median	2.3	0.0	2.0	2.0
	PCM	average	5.9	5.3	5.8	●
		median	5.0	4.1	5.0	●
Percentage of correct clusters	SPLOT	average	82.5	65.6	92.3	81.0
		median	100.0	100.0	100.0	100.0
	PCM	average	100.0	100.0	100.0	●
		median	100.0	100.0	100.0	●
Percentage of features in a correct cluster	SPLOT	average	28.8	2.8	14.2	11.8
		median	28.6	0.0	12.5	10.5
	PCM	average	62.1	44.5	17.6	●
		median	59.5	41.5	14.9	●

(b) With the feature graph (reduced BIG)

Table 6: Cliques and logical feature groups as clusters



RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-0803