Breathing Ontological Knowledge Into Feature Model Synthesis: An Empirical Study

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

▶ **To cite this version:**

Guillaume Bécan, Mathieu Acher, Benoit Baudry, Sana Ben Nasr. Breathing Ontological Knowledge Into Feature Model Synthesis: An Empirical Study. Empirical Software Engineering, 2015, pp.51. 10.1007/s10664-014-9357-1 . hal-01096969

HAL Id: hal-01096969

https://inria.hal.science/hal-01096969

Submitted on 18 Dec 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.

Noname manuscript No.
(will be inserted by the editor)

Breathing Ontological Knowledge Into Feature Model Synthesis: An Empirical Study

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

Received: date / Accepted: date

Abstract Feature Models (FMs) are a popular formalism for modeling and reasoning about the configurations of a software product line. As the manual construction of an FM is time-consuming and error-prone, management operations have been developed for reverse engineering, merging, slicing, or refactoring FMs from a set of configurations/dependencies. Yet the synthesis of meaningless ontological relations in the FM – as defined by its feature hierarchy and feature groups – may arise and cause severe difficulties when reading, maintaining or exploiting it. Numerous synthesis techniques and tools have been proposed, but only a few consider both configuration and ontological semantics of an FM. There are also few empirical studies investigating ontological aspects when synthesizing FMs.

In this article, we define a generic, ontologic-aware synthesis procedure that computes the likely siblings or parent candidates for a given feature. We develop six heuristics for clustering and weighting the logical, syntactical and semantical relationships between feature names. We then perform an empirical evaluation on hundreds of FMs, coming from the SPLOT repository and Wikipedia. We provide evidence that a fully automated synthesis (i.e., without any user intervention) is likely to produce FMs far from the ground truths. As the role of the user is crucial, we empirically analyze the strengths and weaknesses of heuristics for computing ranking lists and different kinds of clusters. We show that a hybrid approach mixing logical and ontological techniques outperforms state-of-the-art solutions. We believe our approach, environment, and empirical results support researchers and practitioners working on reverse engineering and management of FMs.

Keywords Software Product Lines · Feature Model · Variability · Model Management · Reverse Engineering · Refactoring

Inria / IRISA, University of Rennes 1, France
E-mail: 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, 57]. The factorization and exploitation of common features of the products as well as the handling of their variability [14, 15, 67, 71] is an essential step for these stories. Large scale open source or industrial SPLs contain thousands of features and many logical dependencies among them [26, 27]. This complexity poses a challenge for both developers and users of SPLs.

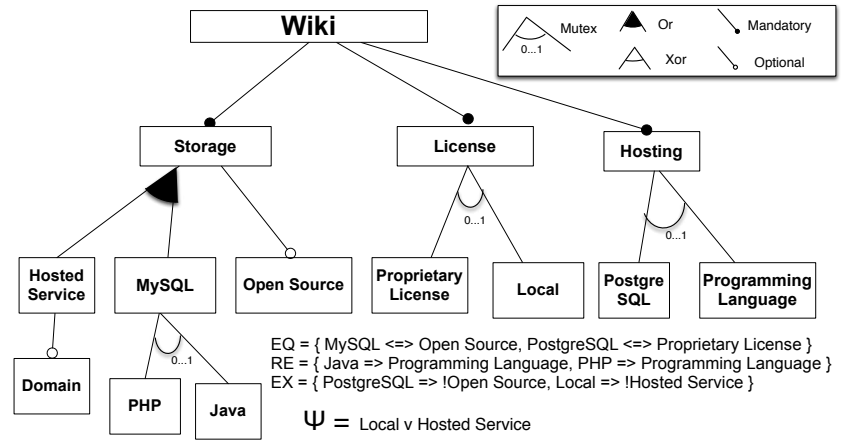
Feature Models (FMs) are by far the most popular notation for modeling and reasoning about an SPL [27, 52]. FMs offer a simple yet expressive way to define a set of legal *configurations* (i.e., combinations of features) each corresponding to a product of an SPL [6,21,39,54,72,73,80,88]. Another important 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* [36] 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 [4,5,41,44,55,68,75–78,83,90,91,94]. 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 [8, 83]. An example of such an FM is given in Figure 1a.

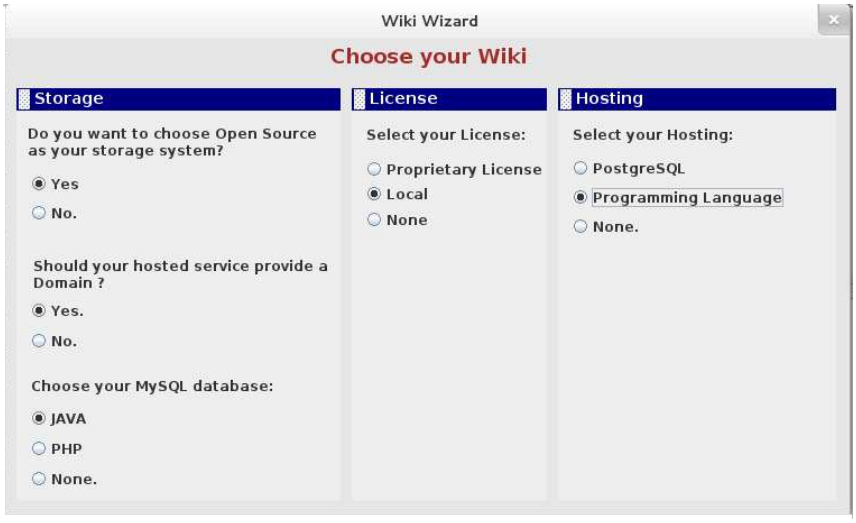
Both configuration and ontological aspects are important when managing FMs. First, the proper handling of configuration semantics is unquestionable. Otherwise the FM may expose to customers configurations that actually do not correspond to any existing product [17,32,33,37,87]. Second, a doubtful feature 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, 15,33,35,37,49,52,63]. Figure 1b depicts a highly questionable user interface of a configurator that could have been generated from the FM of Figure 1a and illustrates the consequence of an inappropriate treatment of the ontological semantics.

The problem addressed in this article 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 compute the likely siblings or parent candidates for a given feature? Given a set of dependencies, we challenge synthesis techniques to assist in selecting a feature hierarchy as close as possible to a *ground truth* – without an *a priori* knowledge of the ground truth.

Several synthesis techniques for FMs have been proposed, mostly in the context of reverse engineering FMs, but they neglected either configuration or



(a) fm_u



(b) Configurator UI

Fig. 1: What practitioners do not want. At the bottom, a non intuitive user interface of a configurator that could have been generated from fm_u due to its doubtful ontological semantics.

ontological semantics of an FM [4,5,13,39,47,48,53,58,59,78,94,97]. There are also few empirical studies investigating how the synthesis techniques behave when ontological aspects of FMs do matter [9].

In previous work [8], we proposed some basic support to guide the synthesis process, but users have to manually choose a relevant hierarchy among thousands of possibilities. It is time-consuming, error-prone, and non realistic for large projects with hundreds or thousands of features. A notable exception

is She *et al.* [83] 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 have been evaluated only in the operating system domain.

In this article we describe a *generic* ontologic-aware FM synthesis procedure. The heuristics rely on general ontologies, e.g. from Wordnet or Wikipedia. They are sound, 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 also propose a hybrid solution combining both ontological and logical techniques.

We perform an empirical evaluation on 156 sets of dependencies for which we have a ground truth FM. We use two data sets: (1) the SPLOT repository [85] and (2) large FMs extracted from *product comparison matrices* (PCMs) found in Wikipedia [79]. 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 about 7 parent candidates per features to consider. We empirically characterize the strengths and limits of state-of-the-art automated techniques in the quest of breathing ontological knowledge into FM synthesis. The experiments also show that a hybrid approach provides the best support for fully synthesizing an FM and computing ranking lists.

Specifically, we make the following contributions:

- We develop six heuristics for clustering and ranking the syntactic/semantic relationships between feature names. We also adapt logical heuristics so that they can be applied before *"breathing ontological knowledge"* into FM synthesis;
- We develop WebFML [23], an environment that integrates all the techniques and supports practitioners in synthesizing sound and meaningful FMs from various kinds of artefacts (e.g. propositional formula, dependency graph, FMs or product comparison matrices). We are unaware of such an environment despite the availability of a wide range of academic or industrial feature modeling tools [25, 56, 61, 70, 74, 89, 94];
- The empirical evaluation shows that the hybrid approach can retrieve, in average, 37.8% of parent-child relationships of the SPLOT FMs (44.4% 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 show that a fully automated synthesis is likely to produce FMs far from the ground truths. We provide empirical evidence that the role of the user remains crucial and we highlight the interactive nature of the synthesis process;
- The hybrid approach ranks the correct parent among the 2 first results for 58.9% 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);

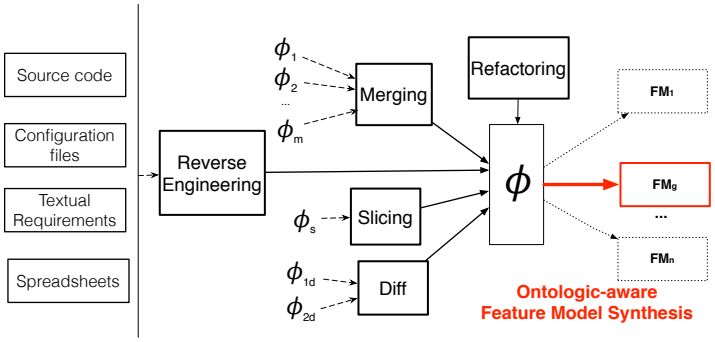


Fig. 2: A key issue for automated operations: numerous FMs conformant to the configuration semantics of ϕ exist but are likely to have an inappropriate ontological semantics.

– We compare our method with an existing technique [48] and logical-based heuristics. Based on empirical results, we analyze the strengths, weaknesses and possible synergies between logical and ontological-based techniques – leading to the design of a 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. 2). Our approach, environment and empirical results open avenues for practical reverse engineering or maintenance of software product lines more and more reported in the industry or observed in the open source community.

The remainder of the article is as follows. In Section 2, we introduce the syntax and the two kinds of semantics (configuration and ontological) of FMs. In Section 3, we motivate and define the ontologic-aware FM synthesis problem. In Section 4, we present generic, automated techniques for breathing ontological knowledge into FM synthesis. In Section 5, we describe and illustrate the integration of the synthesis techniques into the WebFML environment. In Section 6, we perform an empirical evaluation, analyze the results and discuss the key findings. In Section 7, we discuss threats to internal and external validity. In Section 8 we point out the differences and synergies between existing works and our proposal. In Section 9 we summarize the article and present future research directions.

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 [13, 39, 83] while other dialects slightly differ [81].

Definition 1 (Feature Model) *A feature diagram is a 8-tuple* $\langle G, E_M, G_{MTX}, G_{XOR}, G_{OR}, EQ, RE, EX\rangle$*:* $G = (\mathcal{F}, E)$ *is a rooted tree where* $\mathcal{F}$ *is a finite set of features,* $E \subseteq \mathcal{F} \times \mathcal{F}$ *is a set of directed child-parent edges ;* $E_M \subseteq E$ *is a set of edges that define mandatory features with their parents ;* $G_{MTX}, G_{XOR}, G_{OR} \subseteq 2^{\mathcal{F}}$ *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* $A \in \mathcal{F}$ *and* $B \in \mathcal{F}$*. 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* $\mathcal{F}$*.*

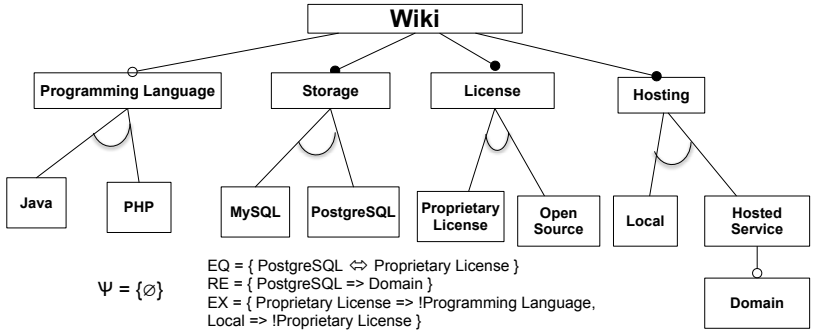
In this article, we employ the construction of *Mutex-groups*. (Techniques presented in [8, 13, 83] also rely on this construction.) Three examples are given in Figure 1a, page 3. As an illustration, PHP and Java form a Mutex-group: these features are mutually exclusive while it is not obliged to select one feature if the parent MySQL is selected. Therefore Mutex-groups differ from optional relations and Xor-groups (see next section).

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. For example, at most one feature can be selected in a Mutex-group. As such Mutex-groups semantically differ from optional relations. Mutex-groups also semantically differ from Xor-group – none of the features can be selected if the parent of a Mutex-group is selected. Formally, 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

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

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



(b) fm_g

$\phi_g =$ *// root and mandatory relationships*
Wiki ∧ (Wiki ⇔ Storage) ∧ (Wiki ⇔ License) ∧ (Wiki ⇔ Hosting)
// parent-child relationships
∧ (Programming Language ⇒ Wiki) ∧ (Java ⇒ Programming Language)
∧ (PHP ⇒ Programming Language)
∧ (MySQL ⇒ Storage) ∧ … ∧ (Domain ⇒ Hosted Service)
// mutual exclusions: at least 1 and at most 1 (Xor-groups)
∧ (Java ⇒ !PHP) ∧ (Programming Language ⇒ (Java ∨ PHP))
∧ ... ∧ (Local ⇒ !Hosted Service) ∧ (Hosting ⇒ (Local ∨ Hosted Service))
// cross-tree constraints (EQ, RE, EX)
∧ (PostgreSQL ⇔ Proprietary License) ∧ ... ∧
(Local => !Proprietary License)

(c) The corresponding formula of fm_g

Fig. 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

configuration semantics can be specified via translation to Boolean logic [39]. As an example, the formula ϕ_g (see Figure 3c) 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 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). Intuitively the ontological semantics of an FM 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 [54]. 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 [8, 83].

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.*

We choose to use *ontological semantics* in line with the terminology employed in previous papers [36,83]. In knowledge engineering, an ontology represents the semantics of concepts and their relationships using some description language (e.g., based on description logic) [19,45]. Feature modeling shares the same goal and is also a concept description technique – less rich and powerful than ontology languages[1]. We concur with the argument that *"feature models are views on ontologies"* [36], the hierarchy and feature groups being central to the conceptualization of a domain.

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 Figure 2).

[1] As feature modeling is a notational subset of ontologies, feature models can be translated into description logics [43] and ontology languages like OWL [93]. The purpose of the translation is to reuse existing description logic solvers and automatically reason about feature models (e.g., for checking consistency) [25]. It should be noted that we do not rely on these techniques (as our goal differs).

3.1 The Importance of Ontological Semantics

Let us consider the following re-engineering scenario (see [28] for more details). After having *reverse engineered* an FM, a practitioner aims to generate a graphical interface (like in Figure 1b) for assisting users in customizing the product that best fits his/her needs. Unfortunately, the ontological semantics of the FM is highly questionable (see Figure 1a) and poses two kinds of problems.

Automated exploitation. Transformation tools that operate over the FM will naturally exploit its ontological semantics. Figure 1 gives an example of a possible transformation from an FM to a user interface (UI). The generated UI (see Figure 1b) is as unintuitive as the ontological semantics of the FM 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. For example, PHP and Java are misplaced below MySQL in the FM of Figure 1a. A developer might understand that the MySQL database is implemented either in Java or PHP whereas the intended meaning was to state that the Wiki engine is developed in one of these two programming languages. This fictive example illustrates that an incorrect ontological semantics might induce a developer in error when exploiting an FM. Second, when writing the transformation, the ontological semantics cannot be exploited as such and the developer 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, a developer can hardly rely on default transformation rules in case the ontological semantics of the FM is incorrect. Default rules typically organize siblings in a same UI block. In the example of Figure 1b (see page 3) the application of transformation rules leads to a discussable layout in the UI (Programming Language and PostgreSQL, Local and Proprietary License are grouped together). As a result, a developer has to override transformation rules, increasing the effort. Another option is to correct the ontological semantics (to enable the application of default transformation rules) and thus *refactor* the original FM. The refactoring of an FM is an instance of the synthesis problem, see below.

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 [4, 16], communicating with other stakehold-

ers [63], composing or slicing FMs [6, 51, 52], relating FMs to other artefacts (e.g., models) [32,33,37,49], or simply generating other artefacts from FMs [35].

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 $\mathcal{F}$ 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* (a comprehensive formalization is given in [13]). Intuitively, as much logical information as possible is represented in the diagram itself, without resorting to the constraints.

Ontologic-aware FM synthesis. The problem tackled in this article 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 Figure 1a *versus* Figure 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 Figure 1a 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.) [6]. Despite some basic heuristics, we observed that the retained hierarchy and feature groups can be inappropriate [8] (the same observation applies for the *merge* and *diff* operators). The ontological aspect of an FM impacts all these automated operations since they are instances of the FM synthesis problem (see Figure 2, page 5).

4 Automated Techniques for Feature Hierarchy Selection

In [8], we empirically showed that once the feature hierarchy is defined, the variability information of the FM can be fully synthesized in the vast ma-

jority of cases. That is, given the knowledge of the hierarchy, synthesis techniques [13] can produce feature groups (Mutex-, Xor-, Or-groups), mandatory and optional relationships, as well as equals, implies, excludes constraints. We thus focus on the challenge of selecting a hierarchy in the remainder of this section. For this purpose, three types of techniques can be considered: logical, ontological and hybrid (a combination thereof).

Before going into details, we first introduce a central structure – the so-called binary implication graph – all kinds of techniques exploit for selecting a sound hierarchy (see Section 4.1).

We present how *logical heuristics* can further exploit the input formula and we highlight their limits (see Section 4.2).

We then describe our *ontologic-aware FM synthesis procedure* which essentially relies on a series of heuristics to rank parent-child relationships (see Section 4.3.1) and compute clusters of conceptually similar features (see Section 4.3.2). The ranking lists and clusters can be interactively reviewed and exploited by a user, and if needs be, an optimum branching algorithm can extract a complete hierarchy (see Figure 6, page 17).

The ontological heuristics can be combined with logical heuristics, resulting in an *hybrid* solution (see Section 4.4).

4.1 Sound Selection of a Hierarchy

The feature hierarchy of an FM defines how features are ontologically related and also participate 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.

Example. The feature Local cannot be a child feature of PostgreSQL since no configuration expressed in Figure 3a authorizes such an implication.

We rely on the *Binary Implication Graph*[2] (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. As an illustration, Figure 4 depicts the BIG of the formula ϕ_g of Figure 3c.

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.

[2] Definition 4 is exactly the definition of feature implication graph employed in [83]. In another context (simplification of conjunctive normal form formula), it should be noted that Heule *et al.* [50] use a different definition of binary implication graph than ours.

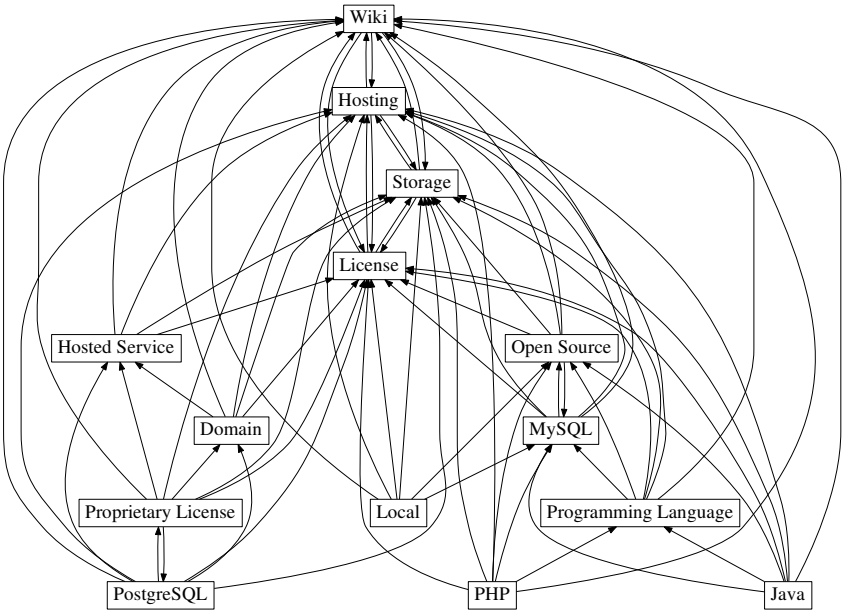


Fig. 4: Binary implicationg graph of the formula in Figure 3b

Definition 4 (Binary Implication Graph (BIG)) *A binary implication graph of a Boolean formula ϕ over $\mathcal{F}$ is a directed graph (V_{imp}, E_{imp}) where $V_{imp} = \mathcal{F}$ 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. The following sections present different types of techniques (logical, ontological and hybrid) for selecting the desired hierarchy.

4.2 Logical Heuristics

From logical constraints expressed by the input formula ϕ, we can compute three interesting logical structures (cliques, logical feature groups, reduced BIG). Heuristics can then operate over these structures for selecting a feature hierarchy.

4.2.1 Cliques

A first logical heuristic 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 can be efficiently computed [13].

Example. The BIG of Figure 4 contains 3 cliques: {Wiki, Storage, License}, {PostreSQL, Proprietary License} and {MySQL, Open Source}. The first clique contain a feature (Wiki) which is indeed the parent of Storage and License (see Figure 3b).

4.2.2 Logical feature groups

All possible feature groups of ϕ can be automatically computed [13, 39]. The computation of feature groups can be performed over the BIG or the reduced BIG that we present below.

Example. 30 MUTEX groups, 21 XOR groups and 1 OR group can be extracted from the formula in Figure 3c. Among these 52 groups, 4 represent the *Xor* groups contained in the reference FM of Figure 3b. If we promote these groups as parent-child relations in the hierarchy, it would remain only 5 out of 13 features without parents. At first glance, these groups seem promising. However, selecting these 4 groups among the 52 that are computed is challenging.

4.2.3 Reduced BIG

Because of the transitivity of implication, a BIG is likely to have a large number of edges, out of which many candidates are not parent-child relationships. It is thus tempting to remove some edges of the BIG for reducing the complexity of the problem. A so-called *reduced BIG* is a subgraph of a BIG (see Definition 4) containing at least one tree that connects all the BIG's vertices, i.e., containing at least one valid hierarchy.

Different strategies can be developed to select the edges that can be removed from the BIG to get a reduced BIG. In our synthesis procedure, we adopted the following strategy. First, we contract each clique in one node, resulting in an acyclic graph. Then, we perform a transitive reduction. Finally, we expand the cliques.

The first step results in an acyclic graph in which the nodes are sets of features. Having an acyclic graph allows to compute a unique transitive reduction that is a subgraph of the BIG [10]. The transitive reduction maintains the reachability in the graph while removing edges.

Example. In Figure 4, Java is connected to MySQL with two path: Java → MySQL and Java → Programming Language → MySQL. The edge Java → MySQL can be removed as the implication is already represented by the path through Programming Language.

The last step is necessary to get back to a graph with features as nodes. This ensures the compatibility with our algorithm on the complete BIG.

Example. After the transitive reduction, the clique {PostreSQL, Proprietary License} is connected to Domain. To expand the clique, we create a node for PostgreSQL and a node for Proprietary License. Then, we add the two following edges: PostgreSQL → Domain and Proprietary License → Domain to maintain the relations of the graph. Finally, we restore the edges PostgreSQL → Proprietary License and Proprietary License → PostgreSQL to represent the clique. Applying our

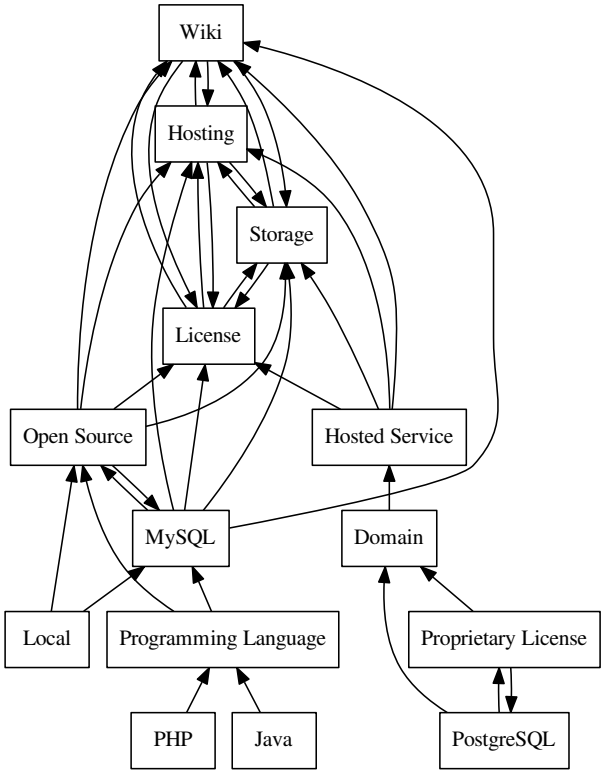


Fig. 5: Reduced binary implicationg graph of the graph in Figure 4

strategy on the BIG of Figure 4 results in the graph in Figure 5. Our strategy significantly reduces the number of edges of this BIG from 71 to 37 edges.

A reduced BIG can be used as a support for a sound synthesis, i.e., all represented hierarchies of the reduced BIG are conforming to the configuration semantics. However, a reduced BIG loses the property of completeness as it arbitrarily removes some hierarchies. Overall, it may be an efficient heuristic for reducing the complexity of the synthesis problem.

4.2.4 Potential Limits of Logical Techniques

We now present three techniques of the state-of-the-art that exploit the previous logical structures (BIG, reduced BIG, cliques and feature groups) to select

an appropriate hierarchy. Then, we highlight their potential[3] limits regarding the ontologic-aware FM synthesis problem defined in Section 3.2.

A first basic approach, denoted $\mathsf{FMRAND}_{\mathsf{BIG}}$, is to randomly select a branching of the BIG. As stated in Section 4.1, every branching of the BIG is a possible hierarchy of the FM and conversely. Therefore, $\mathsf{FMRAND}_{\mathsf{BIG}}$ randomly select one of the possible hierarchies for the FM. A second approach, denoted $\mathsf{FMRAND}_{\mathsf{RBIG}}$, is to randomly select a branching over the *reduced* BIG instead of the complete BIG.

A third approach is proposed in [48]. Haslinger et al. proposed a technique (referred as FMFASE in the remainder of the article) 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 [48], 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) feature names and ontological semantics are not considered during the synthesis.

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 [13]. We consider that the algorithm, called FMFASE_{SAT} hereafter, is representative of a pure logical-based technique for fully synthesizing an FM.

Unawareness of Ontological Meaning. All these logical techniques share the third major drawback: ontological semantics is not considered during the synthesis. $\mathsf{FMRAND}_{\mathsf{BIG}}$ and $\mathsf{FMRAND}_{\mathsf{RBIG}}$ are simply ignoring any information about the features which can results in inappropriate hierarchies such as the one in Figure 1a. FMFASE and FMFASE_{SAT} base their hierarchy selection on logical information which may not reflect actual ontological relations between features. *Example.* The features Programming Language and PostgreSQL can form a mutex group according to the formula in Figure 3c. Promoting this group as part of the hierarchy is valid from a logical point of view but it does not correspond to the desired hierarchy presented in Figure 3b.

Without exploiting ontological knowledge, it is unlikely that the synthesized FM will correspond to an appropriate ontological semantics. This third drawback is a motivation for the development of ontological heuristics which do consider the relations between the features.

[3] The empirical study of the next section is precisely here to observe and quantify the limits in practical settings.

4.3 Ontological Heuristics

As a reminder, the objective is to select the most appropriate hierarchy from the BIG which is a compact representation of all possible hierarchies of an FM. 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 ontological heuristics that directly evaluate the probability of a relationship between a feature and a parent candidate. A *ranking list* of parent candidates for each feature can be extracted from the weighted BIG. From a user perspective, the ranking lists 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 lists. Moreover, users can operate over the clusters to define the parent of a group of features (instead of choosing a parent for every feature 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 [30, 86] whose complexity is $\mathcal{O}(m \log n)$ with n the number of vertices and m the number of edges. The hierarchy is then fed to syntesize 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 Figure 6.

4.3.1 Heuristics for Parent Candidates

The design of the ontological heuristics is guided by a simple observation: *parent-child relations in a hierarchy often represent a specialization or a composition of a concept (feature).*

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. Different kinds of relations (e.g., extension of the behavior of a parent feature) can be considered. The intuitive idea is that sharing the same context tends to make a feature semantically close to its parent and its siblings.

Example. Open source and Proprietary License are both referring to permissive rights about the use of a product.

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.

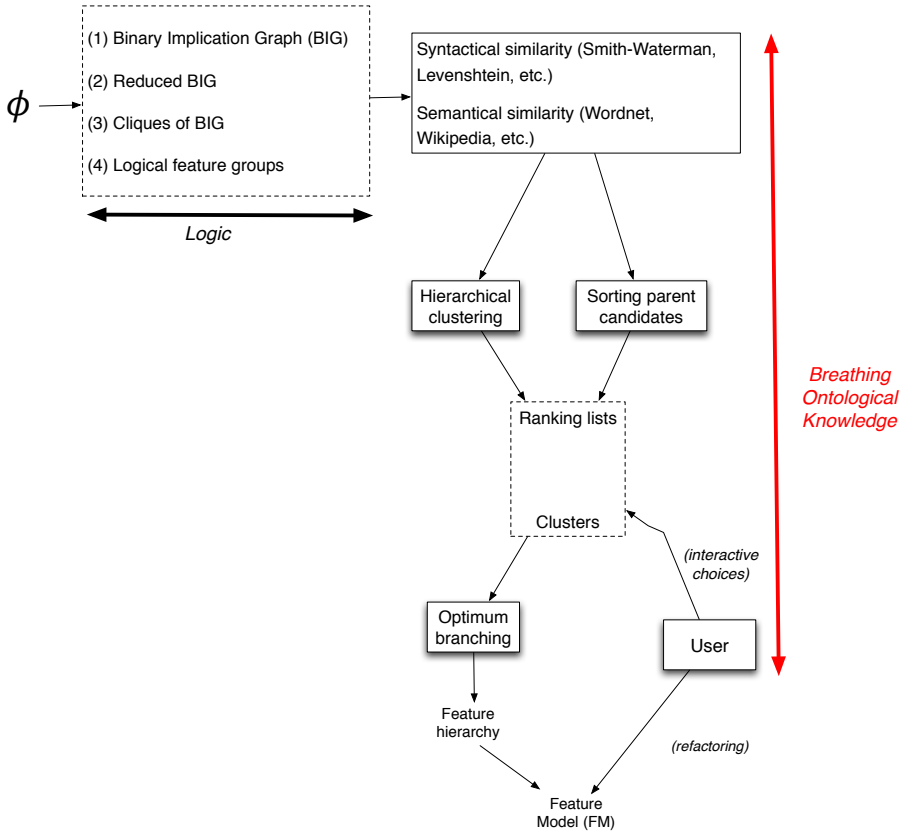


Fig. 6: Ontologic-aware feature model synthesis

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

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* [84] algorithm that looks for similar regions between two strings to determine their distance. We also used the so-called *Levenshtein* edit distance [92] 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 close but semantically close (in the sense of meaning) are not identified, for example, 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* [64]. 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 [29]. Such metrics would require more than the few words contained in the features' names. 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 [95].

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 [96]. 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.

Example. We can associate the features Java and Programming language of the FM in Figure 3b to their respective articles in Wikipedia.

To search articles on Wikipedia and compare them, we use the work of Milne *et al.* [60, 66]. The authors provide a tool, called WikipediaMiner, that can extract a database from an offline dump of Wikipedia, search Wikipedia articles from the database and compare articles based on their hyperlinks[4].

WikipediaMiner provides a model based on the hyperlink structure of Wikipedia that serves as a basis to compute the semantic similarity of two articles. In this model, an article is represented by a vector containing the occurrence of each link found in the article weighted by the probability of the link occurring in the entire Wikipedia database.

Our heuristics exploit WikipediaMiner API and databases to search articles with the closest title to a feature's name[5]. Then, it computes the semantic

[4] The extraction process is time-consuming (about 2 days on a single machine) and the extracted database contains approximatively 40GB of data. WikipediaMiner also provides an API to search and compare articles in the database.

[5] The heuristics based on WikipediaMiner do not guarantee that the most relevant article is retrieved. For instance, searching Storage in Wikipedia leads to a disambiguation page linking to different types of storage. Our current strategy is to arbitrarily choose a given definition. Asking the user to choose the most appropriate article can raise this limitation and is an interesting perspective to further improve the effectiveness of our heuristic.

similarity of the features based on the similarity of the articles. The scores computed by WikipediaMiner are directly assigned to the relevant edges in the BIG. If no article is found, the heuristics simply assigns the minimal weight (i.e., 0) to the edges connected to the feature. It is important to note that the process is fully automated: users of such heuristic never have to manually search or compare Wikipedia articles.

4.3.2 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 heuristics of the previous section (*Smith-Waterman*, *Levenshtein*, *PathLength*, *Wu&Palmer*, *Wikipedia* and *Wiktionary*).

We first 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 and merge the closest clusters according to their similarity. The merge operation is repeated until the distance between two clusters falls below a user specified threshold.

Finally, 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.

- $\exists f_p \in \mathcal{F}, \forall f_c \in C, f_p \in possibleParents(fc)$, *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.

In the two cases, the best parent is chosen according to heuristics for parent candidates.

Example. In Figure 3b, we determine with the Wikipedia 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 – we can automatically compute them using the BIG. The best parent among the common parents corresponds here to Programming Language.

Clusters and Optimum Branching. 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.

Example. We assign the maximum weight (i.e., 1) for the following BIG's edges: Java → Programming Language and PHP → Programming Language.

4.4 Logic + Ontologic = Hybrid

Ontological heuristics allows to synthesize a FM that have both a correct configuration semantics and an appropriate ontological semantics. However, the complexity of the BIG may disturb this type of heuristics and limit their benefits. Instead of operating over the BIG, other logical structures obtained from ϕ can be considered when applying ontological heuristics (see left part of Figure 6). This mix of ontological and logical heuristics is what we call a hybrid heuristic. It exploits logical information to reduce the complexity of the problem and then select an appropriate hierarchy based on ontological information.

We propose a hybrid approach called $\mathsf{FMONTO}_{\mathsf{LOGIC}}$ in the remainder. The ontological heuristics we develop in Section 5.1 operate this time over the reduced BIG (instead of the BIG itself). As explained in Section 4.2, this structure significantly reduces the number of parent candidates and clusters; the drawback is that some hierarchies are no longer represented.

Example. In the complete BIG (see Figure 4), the feature Java has 7 parent candidates. In the reduced BIG (see Figure 5), the same feature Java is this time directly in relation with the correct parent Programming Language. On the other hand, the loss of completeness can prevent the selection of some desired hierarchies, e.g., PostgreSQL is not anymore in relation with its parent Storage after the reduction of the BIG (see Figure 5).

5 Tool Support

We implement and integrate all previous automated techniques into WebFML, a Web-based environment for managing FMs [23]. A unique feature of WebFML is its interactive support for choosing a *sound* and *meaningful* hierarchy. Ranking lists, clusters, cliques, and other facilities (graphical preview, undo/redo, etc.) are all integrated into WebFML to guide users in the choice of a relevant hierarchy among the numerous possibilities.

WebFML is built on top of FAMILIAR [42] and provides FM management operations all based on the synthesis procedure. Likewise practitioners can envision the use of WebFML in practical scenarios: the merging of multiple product lines [7]; the slicing of a configuration process into different steps or tasks [51]; the sound refactoring of FMs [8], especially when fully automated techniques produce incorrect FMs; the reverse engineering of configurable systems through the combined use of composition, decomposition and refactoring operators [2, 3].

WebFML also aims to ease the mechanical translation of artefacts with variability into FMs. It enables practitioners (e.g., product line managers, software developers) to reduce their effort and avoid accidental complexity when (re)-engineering or maintaining their variability models. In a nutshell, WebFML is a comprehensive environment for synthesizing FMs from various kinds of artefacts:

- *product comparison matrices*: specific form of spreadsheets documenting the features of products under comparison. By giving a specific interpretation of their semantics, these matrices can be interpreted as FMs [79];
- *dependency graph* which can be extracted from configuration files of build systems (e.g. *pom.xml* files in Maven) [2, 3];
- *compilation directives and source code* in which features and their dependencies can be mined in order to synthesize an FM [13, 83].
- *propositional formula* in conjunctive or disjunctive normal form [8, 13, 82];
- *a (set of) FMs*: slicing, merging or refactoring existing FMs are instances of the synthesis problem we are addressing with WebFML [7, 8, 51].

5.1 Implementation of Ontological Heuristics

As part of WebFML we implemented six ontological heuristics (see Section 4) based on specialized libraries:

- The syntactical heuristics, *Smith-Waterman* (SW) and *Levenshtein* (L), come from the Simmetrics library[6];
- *PathLength* (PL) and *Wu&Palmer* (WP) rely on extJWNL[7] which handles the communication between WordNet and our tool.
- Wikipedia Miner [66] offers an API to browse Wikipedia's articles offline and compute their relatedness [65]. We used this tool on the english version of *Wikipedia* (Wiki) and *Wiktionary* (Wikt) which form the last two heuristics.

5.2 Features of the environment

WebFML offers an interactive mode where the user can import a formula (e.g., in DIMACS format), synthesizes a complete FM and export the result in different formats. During the FM synthesis, the graphical user interface displays a ranking list of parent candidates for every feature, a list of clusters, a list of cliques and a graphical preview of the FM under construction (see Figure 7, Ⓔ, Ⓕ, Ⓖ and Ⓗ).

During the interactive process, users can:

- select or ignore a parent candidate in the ranking lists (see Figure 7, Ⓕ);
- select a parent for a cluster within the cluster's features or any potential parent feature outside the cluster (see Figure 7, Ⓖ). The user can also consider a subset of a cluster when selecting the parent;
- select a parent for a clique with the same interaction as clusters (see Figure 7, Ⓗ).
- undo a previous choice (see Figure 7, Ⓒ);

[6] `http://sourceforge.net/projects/simmetrics`
[7] `http://extjwnl.sourceforge.net`

- define the different heuristics and parameters of the synthesis (see Figure 7, Ⓑ);
- automatically generate a complete FM according to previous choices and selected heuristics (see Figure 7, Ⓒ);
- use FAMILIAR capabilities within an integrated console (see Figure 7, Ⓘ);
- manage FAMILIAR script files and previously computed variables (see Figure 7, Ⓐ and Ⓓ).

A typical usage is to perform some choices, generate a complete FM with the heuristics and reiterate until having a satisfactory model. At any moment, the optimum branching algorithm can synthesize a complete FM.

5.3 Interactive edits of parent candidates and clusters

The ranking lists of parent candidates and the clusters form the main information during the synthesis. As the amount of information can be overwhelming, we propose a tree explorer view for manipulating and visualizing both parent candidates and clusters (see Figure 7, Ⓕ, Ⓖ and Ⓗ). A tree explorer view is scalable: it allows to focus on specific features or clusters while the other information in the explorer can be hidden. During the synthesis, the user interacts almost exclusively by clicking on the elements of these explorers. Clicking on a feature in parent candidate lists allows to select or ignore a parent candidate. Users can also click on a cluster to choose a parent among the common parent candidates of the sibling features (see Figure 7, Ⓖ). If the cluster contains the desired parent, the user can click on the feature and select the children within the cluster. In both cases, the user can deselect some features to consider only a subset of the cluster. The same behaviour applies to cliques. We also propose a *graphical preview* of the FM based on Dagre[8] and D3[9] javascript libraries. It allows to summarize the previous choices and have a global view of the current result (see Figure 7, Ⓔ).

6 Evaluation

So far, we have presented a sound procedure and various automated techniques, integrated into the WebFML environment, for synthesizing FMs. In this section we empirically investigate which heuristics presented in Section 4 are the best for breathing ontological knowledge. We conduct a series of experiments on 156 realistic FMs (see Section 6.1 for more details). Given an encoding of a FM fm_t as a formula, we evaluate the ability of the different synthesis techniques at selecting a feature hierarchy as close as possible to the original hierarchy of fm_t. Specifically we aim at answering the following research questions:

[8] `https://github.com/cpettitt/dagre`
[9] `http://d3js.org/`

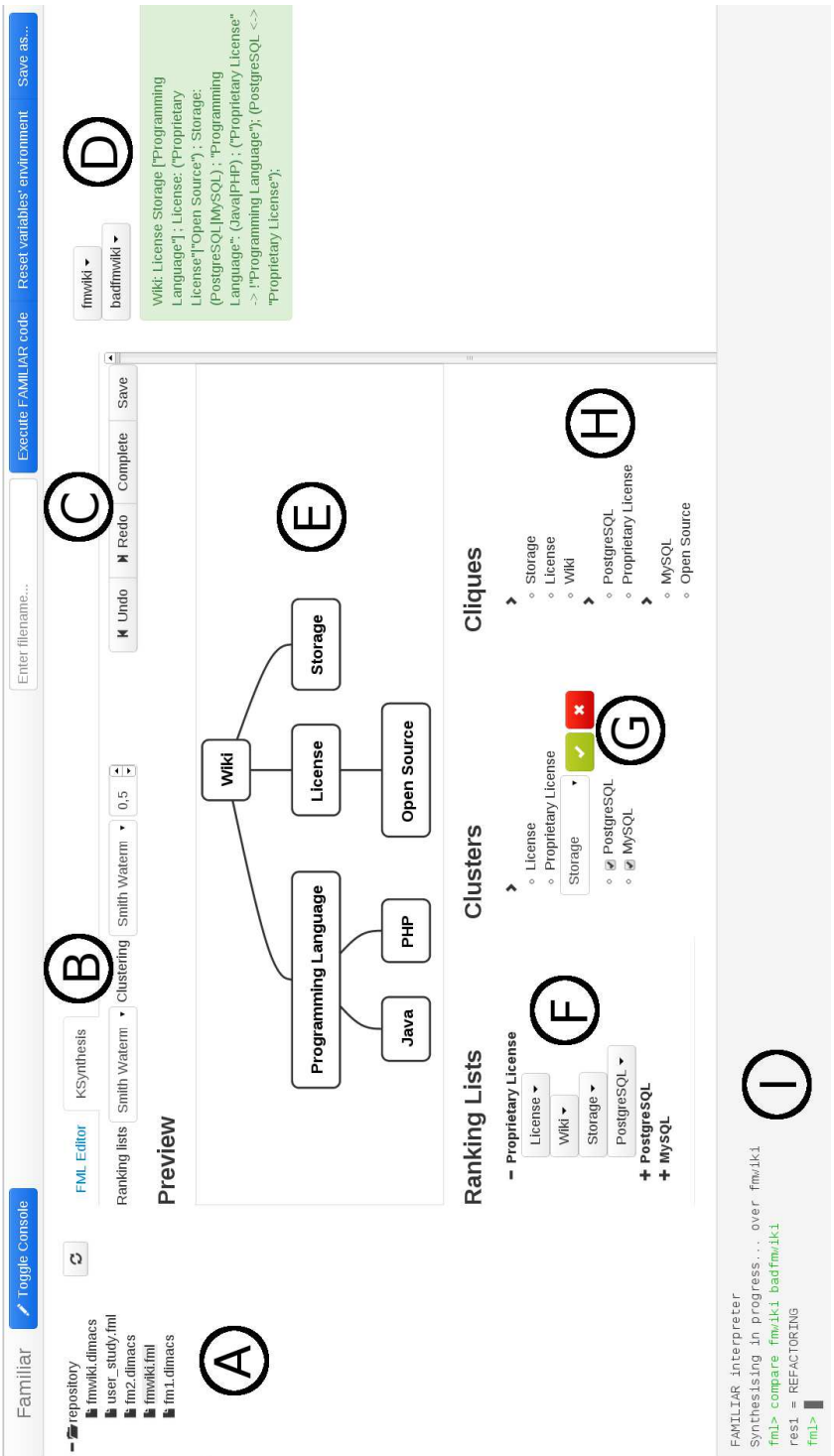


Fig. 7: Interface of the environment during synthesis

- **RQ1:** How effective are the techniques to fully synthesize an FM? Is a fully automated synthesis feasible? The effectiveness corresponds to the percentage of parent-child relations that are correctly synthesized with respect to the ground truth. When the whole synthesis process is performed in a single step, without any user intervention, the resulting FM may be far from the ground truth. This question aims at quantifying this potential effect.
- **RQ2:** How effective are techniques to compute ranking lists and clusters? For ranking lists, the effectiveness corresponds to the percentage of parents from the ground truth that are in the top 2 positions of the lists. For clusters, it corresponds to the percentage of correct clusters (*i.e.* clusters that exist in the ground truth) and the percentage of features that correct clusters represent in the ground truth FM. Selecting 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. Is the reduction of the BIG a good heuristic? Are logical structures like cliques and feature groups good clusters? Do ontological heuristics help?

For each of the research question, we compare logical, ontological and hybrid-based techniques exposed in Section 4. We 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.

6.1 Experimental Settings

6.1.1 FM synthesis techniques under evaluation

Table 1 summarizes the available techniques and when they can be applied (e.g., FMFASE does not compute ranking lists or clusters and thus cannot address **RQ2**). FMONTO denotes the synthesis technique that relies on one of the six ontological heuristics over the BIG for computing the full FM, the ranking lists and clusters. As a reminder, the abbreviations used in the evaluation for the different ontological heuristics of FMONTO and $\text{FMONTO}_{\text{LOGIC}}$ are mentioned in Section 5.1.

For FMFASE, we use the binary provided by the authors [11]. To reduce fluctuations caused by random generation [18], we run 1000 iterations for $\text{FMRAND}_{\text{BIG}}$ and $\text{FMRAND}_{\text{RBIG}}$ each time these heuristics are involved. The results are reported as the mean value over 1000 iterations.

6.1.2 Data

Our experiments operate over two data sets. We translate the configuration semantics of each FM of the data sets into a Boolean formula ϕ. The formula serves as input for our empirical evaluation procedure. The original hierarchy and feature groups of the FMs constitute the *ground truth* to evaluate the ontological quality of the different algorithms and heuristics.

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

SPLOT dataset. The SPLOT [85] 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[10]. This resulted in 201 FMs with a total of 5023 features. Due to memory consumption issues [48] and technical issues, 75 FMs from the SPLOT data set could not be handled by FMFASE. Therefore, we performed the experiment on the remaining 126 FMs that represent 2214 features.

Overall, the dataset is similar to the one used in [48] which was also extracted from SPLOT, 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 [24, 79]. We gathered 30 FMs with an automated extraction process – in the same vein as the one described in [5]. Each configuration of 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.

We exploit the structure of the matrices as follows. In a PCM, there are values in each column and headers for characterizing a column. For example, the values MacOS X, Windows 7, or Ubuntu are values of a column whose header name is Operating System. We use the following strategy for deriving a hierarchy: values and headers are features; values are located below the header. In the example, MacOS X, Windows 7, or Ubuntu would be features of Operating System in a feature hierarchy. The overall structure of a Wikipedia page was created by domain experts (Wikipedia contributors) for organizing features of a set of product. 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 the depth of the hierarchies is 4 in average.

Table 2 presents some statistics about the FMs. As an indication of the complexity of the synthesis process, we compute the number of parent candidates for each feature from the BIG of an FM. We have an average of 6.3 (from 0 to 36) parent candidates in SPLOT FMs and 8.3 (from 1 to 86) for PCM FMs. We also compute the average number of character contained in

[10] 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.

Dataset	# features	# edges in the BIG	# edges in the reduced BIG	Depth of hierarchy
SPLOT (126 FMs)	17.6 (min 10, max 56)	131.7	54.8	3.8
PCM (30 FMs)	72.4 (min 23, max 177)	580.5	316.6	4.0

Table 2: Properties of FMs (average per FM)

feature names of the input formula. The average is 11 characters per name for SPLOT FMs and 18 for PCM FMs. With so little information for the ontological heuristics used in FMONTO and $\text{FMONTO}_{\text{LOGIC}}$, the datasets are challenging and longer feature descriptions may improve the following results.

6.1.3 Method for Comparing the Evaluated Techniques

To answer our research questions, we compare the presented techniques according to a set of metrics that will be defined throughout the evaluation. To perform this comparison, we compute the *average* and the *median* of the metrics over the datasets.

We also run statistical tests to assess the significance of a difference between two techniques. When comparing two techniques, we measure the same metric on the same FMs, thus having paired samples. In this case, the most suitable tests are paired tests. A classic paired test is the t-test. However, this particular test requires that the data follows a normal distribution. To verify this assumption, we used the Shapiro-Wilk test. Most of these tests rejected (according to a threshold of 0.05 for the p-value) the hypothesis that the data is normal. Therefore, our experiments rely on the Wilcoxon rank-sum test, which does not assume a normal distribution of data. In our evaluation, the null hypothesis states that the two compared techniques have the same mean. The alternate hypothesis states that the means are different. Therefore, if we reject the null hypothesis, the difference between two techniques is significant but the direction of this difference is still unknown.

To complete the results of our tests, we also compute the effect size (*i.e.* how different are the results). In our case, we measure the effect size as the difference of the means of the two compared techniques. Using this definition allows an easy interpretation of the effect size in terms of the metric used. It also allows to know the direction of the difference between two heuristics.

In the following sections, the results of the statistical tests are reported according to a threshold of 0.05 for the p-value. For further details, all the p-values and effect sizes are reported in Appendix A.

6.2 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 (see Definition 5).

Definition 5 (Effectiveness for a fully automated synthesis) *The effectiveness of an FM synthesis algorithm in a fully automated process is computed as follows:*

$$effectiveness = \frac{commonEdges(synthesized\ FM,\ ground\ truth)}{|F| - 1}$$

commonEdges(synthesized FM, ground truth) represents the number of edges that are common between the hierarchy synthesized by the algorithm and the hierarchy from the ground truth. It corresponds to the number of correct parent-child relations according to the ground truth. $|F|$ *is the number of features in the ground truth FM. As an FM hierarchy is a tree,* $|F|-1$ *is the total number of parent-child relations.*

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. In this experiment, we only use the ranking lists computed by the different techniques and we do not consider ontological or logical clusters.

Table 3 reports the percentage of common edges with the ground truth. We split the table in two, clearly separating (1) techniques that operate over the BIG (see Table 3a) from (2) techniques that apply a logical heuristic and operate over the reduced BIG (see Table 3b). There are two hypotheses behind this separation.

(**H1**) 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 $\mathsf{FMRAND}_{\mathsf{BIG}}$. For SPLOT, the best heuristic PathLength improves by 9.5% (average) and 10.6% (median) the effectiveness of $\mathsf{FMRAND}_{\mathsf{BIG}}$. Similar observations are made for the PCM dataset: the best heuristic Wikipedia improves by 10.4% (average) and 13.1% (median) the effectiveness of $\mathsf{FMRAND}_{\mathsf{BIG}}$.

Table 3b shows that almost all ontological heuristics (FMONTO) outperform FMFASE, FMFASE_{SAT} or $\mathsf{FMRAND}_{\mathsf{RBIG}}$ being on SPLOT or PCM dataset. Only the Levenshtein heuristic is outperformed by FMFASE on SPLOT dataset and by $\mathsf{FMRAND}_{\mathsf{RBIG}}$ on 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).

Statistical tests confirm that FMONTO significantly outperforms $\mathsf{FMRAND}_{\mathsf{BIG}}$, except for the Wu&Palmer heuristic on the PCM dataset and the Levenshtein heuristic on both datasets. For the results of $\mathsf{FMONTO}_{\mathsf{LOGIC}}$, 6 out of 30 statistical tests shows that it outperforms pure logical techniques ($\mathsf{FMRAND}_{\mathsf{RBIG}}$, FMFASE and FMFASE_{SAT}). The other tests cannot conclude that there is a significant difference with the results of the pure logical techniques.

Overall, we conclude that the hypothesis **H1** is verified for the ontological techniques of FMONTO. We also note an improvement of hybrid techniques of

FMONTO$_{\text{LOGIC}}$ compared to pure logical techniques (FMRAND$_{\text{RBIG}}$, FMFASE and FMFASE$_{SAT}$). However, the tests cannot confirm that this difference is significant. This may be explained by the use of the reduced BIG. The next hypothesis address this question.

(**H2**) The reduced BIG can improve the effectiveness.

H2 Results. Results in Table 3a and Table 3b indicate that all techniques benefit from the reduction of the BIG. However the improvement is more apparent for FMRAND$_{\text{BIG}}$. Specifically FMRAND$_{\text{RBIG}}$ increases by 24.1% (resp. 10.4%) the effectiveness of FMRAND$_{\text{BIG}}$ in PCM dataset (resp. SPLOT dataset). Comparatively, the best improvement of FMONTO$_{\text{LOGIC}}$ w.r.t. FMONTO is 21.8% (resp. 8.2%) in PCM dataset (resp. SPLOT dataset).

The reduction of the BIG significantly decreases the number of edges, thus favouring a random selection. For the SPLOT dataset, 56.1% (in average, 59.1% for the median) of the parent-child relations from the ground truth are kept after reduction of the BIG while more than half of the edges are removed. For PCM dataset, 83.8% (in average, 88.1% for the median) of the parent-child relations from the ground truth are kept after reduction of the BIG while almost half of the edges are removed. In practice the tradeoff between a reduction of the problem space and a less accurate representation clearly favours both FMONTO$_{\text{LOGIC}}$ and FMRAND$_{\text{RBIG}}$.

If we compare the score of FMONTO in Table 3a with the scores of FMRAND$_{\text{RBIG}}$, FMFASE and FMFASE$_{SAT}$ in Table 3b, we note that the purely logical techniques on the reduced BIG outperform ontological techniques on the complete BIG. This is another important observation in favour of **H2**. 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. Finally, all the statistical tests confirm that **H2** is verified.

Key findings for RQ1.

- A fully automated synthesis produces FMs far from the ground truth. At best, the percentage of correct parent-child relationships in the synthesized FM is 37.8% (for SPLOT) and 44.4% (for PCM); 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.
- The experiment demonstrates that a 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.

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

(a) Full synthesis with BIG

Data set		Pure logical techniques	Ontological techniques (FMONTO)					
		$\text{FMRAND}_{\text{BIG}}$	SW	L	WP	PL	Wiki	Wikt
SPLOT	average	20.9	28.8	26.9	29.1	**30.4**	29.4	28.5
	median	16.7	24.0	21.8	**27.3**	**27.3**	26.5	**27.3**
PCM	average	15.9	24.0	19.4	20.8	23.2	**26.3**	22.7
	median	14.0	19.9	18.0	19.0	22.4	**27.1**	19.0

(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 ($\text{FMONTO}_{\text{LOGIC}}$)					
		FMFASE	FMFASE_{SAT}	$\text{FMRAND}_{\text{RBIG}}$	SW	L	WP	PL	Wiki	Wikt
SPLOT	average	36.1	31.9	31.3	36.9	34.9	36.5	**37.8**	37.4	36.7
	median	35.8	25.0	27.3	32.7	30.8	32.2	**36.2**	33.9	33.3
PCM	average	●	39.9	40.0	43.1	39.8	42.6	43.6	**44.4**	42.2
	median	●	32.9	32.8	35.9	34.2	34.1	35.2	**37.4**	35.5

6.3 RQ2: Quality of ranking lists and clusters

Our goal is to evaluate the quality of the ranking lists and the clusters. We also evaluate the use of cliques and feature groups during the synthesis for selecting the hierarchy of an FM. In this experiment we do not consider FMFASE and FMFASE_{SAT} techniques as they do not provide such information.

6.3.1 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 look at the *Top 2* positions of the lists and evaluate the effectiveness of a technique at computing ranking lists as defined in Definition 6. With an average of 6.3 parent candidates per feature in the SPLOT dataset, 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 32% (in average) of having a correct parent for randomized approaches.

Definition 6 (Effectiveness for computing ranking lists) *The effectiveness of an FM synthesis algorithm for computing ranking lists is computed as follows:*

$$effectiveness = \frac{\#\ parents\ in\ top\ 2\ positions}{|F| - 1}$$

To compute the # parents in top 2 positions, we check that the parent of each feature (as it appears in the ground thuth) appears in the top 2 positions of the ranking list. $|F| - 1$ *represents the number of features that have a parent in the ground truth (*i.e. *all the features except the root).*

Table 4 reports the percentage of correct parents in the Top 2 positions of the ranking lists. As in the previous experiment, we separate the techniques

that operate over the BIG from the techniques that operate over the reduced BIG and pose the same two hypotheses as in RQ1.

(**H3**) 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 10.2% (average) and 11.7% (median) the effectiveness of $\text{FMRAND}_{\text{BIG}}$. For the other dataset, the Wikipedia heuristic improves by 9.4% (average) and 10.6% (median) the effectivness of $\text{FMRAND}_{\text{BIG}}$. The statistical tests confirm these results for the SPLOT dataset and for the Wikipedia heuristic on the PCM dataset. For the other heuristics, the tests cannot confirm that there is a significant difference between FMONTO and $\text{FMRAND}_{\text{BIG}}$.

Table 4b also shows that $\text{FMONTO}_{\text{LOGIC}}$ outperforms $\text{FMRAND}_{\text{RBIG}}$ on both data sets but the scores are significantly closer. The tests reflect this observation as they cannot confirm that a difference exists between $\text{FMONTO}_{\text{LOGIC}}$ and $\text{FMRAND}_{\text{RBIG}}$, except for the PathLength heuristic on the SPLOT dataset.

Overall, we cannot conclude that the hypothesis **H3** is verified in all cases but the average and median scores indicate at least a slight improvement of the effectiveness of FMONTO (resp. $\text{FMONTO}_{\text{LOGIC}}$) compared to $\text{FMRAND}_{\text{BIG}}$ (resp. $\text{FMRAND}_{\text{RBIG}}$).

(**H4**) The reduced BIG improves the quality of ranking lists.

H4 Results. The SPLOT dataset (resp. PCM dataset) results in a 5.7% (average) and 6.3% (median) (resp. 16.5% and 15%) improvement of the effectiveness of our best heuristic PathLength. $\text{FMRAND}_{\text{RBIG}}$ also clearly benefits from the reduction of the BIG as it improves the effectiveness of $\text{FMRAND}_{\text{BIG}}$ by 8.6% for SPLOT and 19.9% for PCM.

The statistical test confirms the results on the PCM dataset. However, on the SPLOT dataset, the difference is not significant for most of the heuristics. This may be explained by the 43.9% of correct parents brutally removed by the reduction in the SPLOT dataset. For the PCM dataset, the reduction of the BIG removes only 16.2% of correct parents.

As a result, the hypothesis **H4** is verified for the PCM dataset but the results for SPLOT shows that the reduced BIG may have a limited effect.

6.3.2 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 4.3.2, page 19. For example in Figure 3b, page 7, {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. The effectiveness of a technique corresponds to the number of correct clusters and the percentage of features that these clusters represent (see Definition 7 for further details).

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

(a) With BIG

Data set		Pure logical technique	Ontological techniques (FMONTO)					
		$\mathsf{FMRAND}_{\mathsf{BIG}}$	SW	L	WP	PL	Wiki	Wikt
SPLOT	average	43.0	50.0	49.5	51.6	**53.2**	51.4	50.2
	median	38.3	48.8	46.3	50.0	50.0	**55.4**	50.0
PCM	average	32.3	39.9	37.3	37.0	39.8	**41.7**	38.0
	median	28.9	37.2	35.8	34.5	**40.0**	39.5	34.0

(b) With reduced BIG

Data set		Pure logical technique	Hybrid techniques ($\mathsf{FMONTO}_{\mathsf{LOGIC}}$)					
		$\mathsf{FMRAND}_{\mathsf{RBIG}}$	SW	L	WP	PL	Wiki	Wikt
SPLOT	average	51.6	56.4	55.6	57.1	**58.9**	57.1	56.5
	median	50.5	55.6	55.1	54.9	56.3	**58.3**	57.1
PCM	average	52.2	54.9	53.0	55.0	**56.3**	55.7	54.3
	median	46.5	53.5	47.7	52.9	55.0	**55.9**	53.6

Definition 7 *The effectiveness of an FM synthesis algorithm for computing clusters is composed of two aspects:*

$$\textit{Percentage of correct clusters} = \frac{\textit{\# correct clusters}}{\textit{\# clusters}}$$

$$\textit{Percentage of features in a correct cluster} = \frac{\textit{\# features in correct clusters}}{|F|}$$

 # clusters represents the number of clusters that are generated by the algorithm. # correct clusters is the number of clusters that contain either sibling features from the ground truth or siblings with their corresponding parent. # features in correct clusters is the sum of the correct clusters' sizes. Finally, $|F|$ represents the number of features in the ground truth.

Table 5 reports for each dataset, the number of clusters, the clusters' sizes, the percentage of correct clusters computed by the techniques, and the number of features in a correct cluster (in average and for the median). For each heuristic, we optimized[11] the thresholds for the clustering in order to produce the largest number of features in a correct cluster. Intuitively, clusters allow the user to choose only one parent for a set of features. Maximizing the number of features in correct clusters potentially reduces the user effort. As previously, we separate our evaluation in two hypotheses.

(**H5**) Ontological techniques are superior to random heuristics when computing clusters over the *same* logical structure.

H5 Results. Table 5a shows that FMONTO generates less clusters per FM than $\mathsf{FMRAND}_{\mathsf{BIG}}$. However, FMONTO produces clusters that are slightly bigger and more accurate. For SPLOT, our best heuristic PathLength generates

[11] The threshold values were manually determined. For each heuristic, we changed the threshold value by steps of 0.1 (all our heuristics return a value between 0 and 1) to maximize the average number of features in a correct cluster.

67.6% (average) and 75% (median) of correct clusters while $\mathsf{FMRAND}_{\mathsf{BIG}}$ reaches 27.7% (average) and 25% (median). The percentage of features in correct clusters is also significantly better than $\mathsf{FMRAND}_{\mathsf{BIG}}$. For the PCM dataset, the difference is greater with 79.2% (average) and 77.7% (median) of correct clusters for PathLength comparted to only 14.9% (average) and 12.8% (median) for $\mathsf{FMRAND}_{\mathsf{BIG}}$. Table 5b also shows that $\mathsf{FMONTO}_{\mathsf{LOGIC}}$ outperforms $\mathsf{FMRAND}_{\mathsf{RBIG}}$ for all the results.

We observe that PathLength is the best heuristic according to the percentage of correct clusters whereas Levenshtein is the best one according to the percentage of features in a correct cluster. We also note that the Levenshtein heuristic produces more and bigger clusters than PathLength. These differences show that there is a tradeoff between the number of features contained in clusters and their accuracy. It corresponds to the classical tradeoff between precision and recall that will be further illustrated with the use of the reduced BIG in **H6**.

Statistical tests that FMONTO (resp. $\mathsf{FMONTO}_{\mathsf{LOGIC}}$) outperforms $\mathsf{FMRAND}_{\mathsf{BIG}}$ (reps. $\mathsf{FMRAND}_{\mathsf{RBIG}}$) on both datasets, except for Wu&Palmer and PathLength heuristics. It confirms the usefulness of ontological-based heuristics, whether they operate over a reduced BIG or a BIG and validates the hypothesis **H5**.

(**H6**) The reduced BIG improves the quality of the clusters.

H6 Results. Table 5b shows that the reduced BIG allows to produce less clusters per FM compared to the techniques operating over the complete BIG. The clusters are also slightly smaller. For SPLOT, 74.8% (average) and 100% (median) of clusters are correct with our best heuristic Wiktionary. For PCM, the PathLength heuristic produces 89.2% (average) and 89.6% (median) of correct clusters. Therefore, and for every ontological-based heuristics, the accuracy of the generated clusters increases when we reduce the BIG. The statistical tests confirm these results except for Wu&Palmer and PathLength heuristics on the SPLOT dataset despite their improvements.

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 5a), our best heuristic Levenshtein produces 31% (average) and 30% (median) of correct clusters for SPLOT (resp. 41.4% and 41.2% for PCM). On a reduced BIG, Levenshtein produces 29.2% (average) and 29% (median) for SPLOT (reps. 41% and 41.2% for PCM). All the statistical tests are consistent with these observations as they cannot state that the difference is significant.

The results show that there is no clear superiority of an approach. The use of the reduced BIG or the use of the complete BIG when computing clusters have both pros and cons. On the one hand, the reduced BIG supports the identification of more accurate but less clusters with less features. On the other hand, 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.

Table 5: Clusters generated by heuristics (RQ2)

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

Metric	Data set		Pure logical technique	Ontological techniques (FMONTO)					
			FMRAND$_{\text{BIG}}$	SW	L	WP	PL	Wiki	Wikt
Number of clusters	SPLOT	average	**6.2**	4.1	4.0	2.7	2.5	3.3	2.9
		median	**5.0**	4.0	3.5	2.0	2.0	3.0	3.0
	PCM	average	**29.7**	16.8	17.6	5.8	8.7	9.8	12.3
		median	**25.0**	15.5	14.5	5.0	7.0	8.0	9.5
Clusters'size	SPLOT	average	2.2	**2.9**	2.8	2.5	2.3	**2.9**	2.3
		median	2.1	**2.7**	2.5	2.3	2.2	**2.7**	2.0
	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	**5.1**	2.8
Percentage of correct clusters	SPLOT	average	27.7	50.4	55.3	57.3	**67.6**	54.7	66.8
		median	25.0	50.0	50.0	50.0	**75.0**	50.0	69.0
	PCM	average	14.9	49.6	62.6	60.8	**79.2**	53.7	71.0
		median	12.8	47.3	65.5	64.6	**77.7**	57.1	66.7
Percentage of features in a correct cluster	SPLOT	average	18.9	29.0	**31.0**	19.5	21.8	25.7	24.5
		median	16.7	26.5	**30.0**	17.9	20.0	22.2	23.3
	PCM	average	12.5	35.0	**41.4**	12.9	22.8	24.5	32.2
		median	10.5	32.3	**41.2**	12.4	23.2	21.9	31.9

(b) Clusters generated by FMRAND$_{\text{RBIG}}$ and FMONTO$_{\text{LOGIC}}$

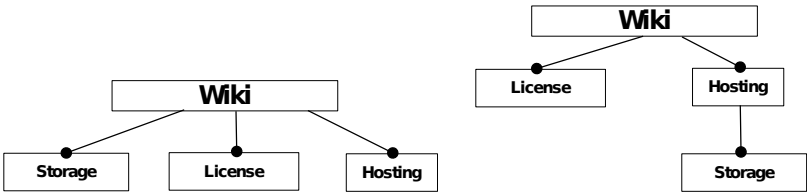
Metric	Data set		Pure logical technique	Ontological techniques (FMONTO$_{\text{LOGIC}}$)					
			FMRAND$_{\text{RBIG}}$	SW	L	WP	PL	Wiki	Wikt
Number of clusters	SPLOT	average	**4.0**	3.0	3.1	2.3	2.0	2.5	2.3
		median	**4.0**	3.0	3.0	2.0	2.0	2.0	2.0
	PCM	average	12.9	11.2	**14.0**	4.8	7.6	7.1	9.8
		median	12.0	10.0	**13.0**	5.0	6.5	6.0	9.0
Clusters'size	SPLOT	average	2.1	**2.5**	**2.5**	2.2	2.1	**2.5**	2.1
		median	2.0	2.3	2.4	2.0	2.0	**2.5**	2.0
	PCM	average	2.2	3.1	2.7	2.5	2.5	**4.4**	2.9
		median	2.2	2.7	2.8	2.4	2.3	**3.8**	2.7
Percentage of correct clusters	SPLOT	average	40.4	66.2	67.3	64.7	73.2	67.5	**74.8**
		median	33.3	66.7	75.0	66.7	**100.0**	75.0	**100.0**
	PCM	average	36.7	68.2	75.9	73.3	**89.2**	72.4	85.7
		median	28.6	71.4	82.7	76.4	**89.6**	80.0	89.4
Percentage of features in a correct cluster	SPLOT	average	17.9	26.6	**29.2**	18.7	19.8	23.8	22.4
		median	15.4	23.1	**29.0**	16.7	17.0	20.0	20.0
	PCM	average	12.3	34.6	**41.0**	12.8	22.7	24.2	32.2
		median	10.5	32.3	**41.2**	12.4	23.2	21.4	31.9

6.3.3 Using Cliques in FM Hierarchy Selection

Features that co-occur in configurations (i.e., cliques) can be efficiently computed using standard logical techniques [13]. We want to further understand how an FM synthesis algorithm can benefit from the use of cliques as a logical heuristic.

A first step is to identify the type of relations contained in cliques. For the SPLOT dataset (resp. the PCM dataset), we report that 93.8% (average) and 100% (median) of the cliques (resp. 98.9% and 100%) contain features that are connected to others by parent-child relationships. The remaining cliques are not subtrees of the hierarchy and contain at least one bi-implication cross-tree constraint. Therefore, cliques almost always represent parent-child relations between features.

From this observation, we consider cliques as a special kind of cluster. For example, FMFASE uses cliques as a cluster. It chooses one feature of a clique and places it as the parent of the others. This pattern is what we call a *simple*



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

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

Fig. 8: Clique unfolding: simple versus complex

unfolding: users just have to select one feature in the clique that will play the role of parent feature of the others. For instance, the clique {Wiki, Storage, License, Hosting} is transformed through a simple unfolding in Figure 8a: Wiki is the parent of Storage, License and Hosting.

However, more *complex unfolding* may arise. Let us take a fictive example and consider that the feature Storage is below Hosting (and not Wiki as in the running example). In that case, the clique requires a complex unfolding that consists in defining several levels of features in the hierarchy (see Figure 8b for an example).

We evaluate the use of cliques as clusters with the same metrics used in Section 6.3.2. Table 6a presents the results. We observe that simple unfolding (*i.e.* a correct cluster) is required in 50% (average) and 50% (median) of the cliques of SPLOT FMs. For PCM FMs, this pattern represents 31.1% (average) and 0% (median) of the cliques. In at least half of the cases, cliques require more user effort than a traditional cluster in which only one parent should be selected (*i.e.* it requires a complex unfolding). Therefore, a simple unfolding can be used as a default heuristic for selecting parent-child relations in a clique but not as a reliable heuristic in a fully automated synthesis.

6.3.4 Using Logical Feature Groups in FM Hierarchy Selection

In addition to cliques, logical feature groups can be exploited during the selection of the FM hierarchy. All *possible* logical feature groups of a formula can be computed either using the reduced BIG [13] or the BIG. A logical feature group consists in a set of sibling features and a parent feature. It is in line with the definition of cluster given in Section 4.3.2. We study here how the so-called *logical feature groups* are interesting structures for representing clusters.

Computing feature groups from a formula may lead to numerous groups containing the same set of sibling features but with different parents. This is due to feature groups in which the parent is logically bi-implied by another feature (i.e., the parent feature belongs to a clique). An example is given in Figure 9. To avoid this combinatorial explosion, we omit the parents and consider only the sibling features. We introduce a place-holder that can be one of the parent of the siblings (see right of Figure 9). This factoring strategy has

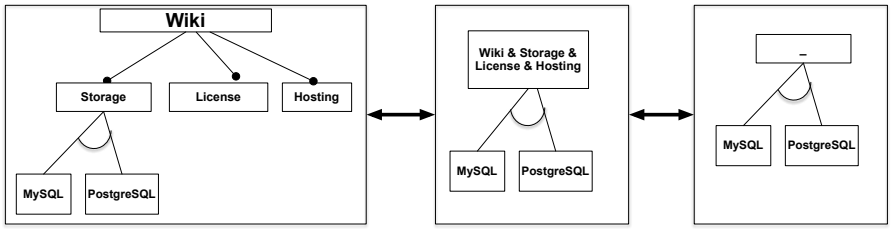


Fig. 9: Logical feature groups: bi-implied features, clique contraction, parent place-holder

the merit of presenting to users only one cluster, abstracting numerous other clusters.

For the experiment, we verified whether logical feature groups are correct clusters – in line with the definition of Section 6.3.2, page 30.

Table 6a shows the results. For SPLOT FMs, 69.3% (average) and 75% (median) of logical feature groups are correct clusters. These results are slightly superior to the 67.6% (average) and 75% (median) of correct clusters generated by our best heuristic PathLength (see Table 5a).

For PCM FMs, 92.2% (average) and 100% (median) of feature groups are correct clusters. It outperforms the 79.2% (average) and 77.7% (median) of correct clusters from our best 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 challenge an automated technique or overwhelm a user with numerous incorrect clusters.

(**H7**) The reduced BIG can improve the quality of logical feature groups.

H7 Results. Table 6b shows the same metrics than Table 6a but for feature groups computed from the reduced BIG of the FM. We note that reducing the BIG significantly reduces the number of computed feature groups. Also, their accuracy increases compared to the groups in Table 6a but the correct groups often involve less features. Statistical tests show that the percentage of correct clusters is significantly better when using a reduced BIG. However, the difference in the percentage of features in a correct cluster is not stated as significant. Moreover, the test for the Mutex groups on the SPLOT dataset shows a significant decrease of the metric. Therefore, the test cannot verify **H7** but they illustrate the tradeoff between precision and recall when computing clusters which is consistent with the results of **H6**.

Table 6: Cliques and logical feature groups as clusters

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

Metric	Data set		Cliques	Feature groups			
				All groups	Mutex	Xor	Or
Number of clusters	SPLOT	average	1.6	9.5	2.8	5.9	1.3
		median	1.0	3.0	0.0	1.0	1.0
	PCM	average	1.4	55.7	45.2	17.8	●
		median	1.0	8.0	7.5	1.0	●
Clusters'size	SPLOT	average	5.0	2.2	1.2	1.6	1.8
		median	4.0	2.3	0.0	2.0	2.0
	PCM	average	7.0	5.7	5.6	5.4	●
		median	6.0	5.0	4.5	5.0	●
Percentage of correct clusters	SPLOT	average	50.0	69.3	41.7	89.1	72.9
		median	50.0	75.0	33.3	100.0	100.0
	PCM	average	31.1	92.2	91.5	91.1	●
		median	0.0	100.0	100.0	100.0	●
Percentage of features in a correct cluster	SPLOT	average	17.5	36.3	5.8	21.9	13.2
		median	15.4	35.0	0.0	16.0	8.9
	PCM	average	3.3	60.7	53.2	15.6	●
		median	0.0	60.7	53.1	12.2	●

(b) With the feature graph (reduced BIG)

Metric	Data set		Feature groups			
			All groups	Mutex	Xor	Or
Number of clusters	SPLOT	average	3.7	0.4	2.0	1.3
		median	2.0	0.0	1.0	1.0
	PCM	average	10.4	8.4	2.0	●
		median	8.0	5.0	1.0	●
Clusters'size	SPLOT	average	2.2	0.5	1.4	1.8
		median	2.3	0.0	2.0	2.0
	PCM	average	5.9	5.3	5.8	●
		median	5.0	3.9	5.0	●
Percentage of correct clusters	SPLOT	average	80.0	69.0	92.1	72.9
		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	32.6	2.6	16.7	13.2
		median	33.3	0.0	13.6	8.9
	PCM	average	64.3	46.0	18.3	●
		median	60.7	40.7	14.0	●

Key findings 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 58.9% 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 or clusters;
- Ontological heuristics are beneficial to ranking lists. Ontological heuristics also generate less clusters than $\mathsf{FMRAND}_{\mathsf{BIG}}$ but they are far more accurate;
- Among the ontological heuristics, there is no clear winner – even if all are beneficial for breathing ontological knowledge;
- In 93.8% (resp. 98.9%) of the cases, all features of the cliques are connected by parent-child relationships to at least one feature. Nevertheless cliques require complex unfolding in more than 50% of the cases;
- Logical feature groups form accurate clusters especially when they are computed over the reduced BIG. The place-holder for factoring out similar feature groups makes the difference. Yet there are extreme cases in which lots of logical feature groups do not correspond to clusters.

6.4 Summary and Discussion

Eventually a combined use of techniques, that is, a hybrid approach empirically emerges:

- Ontological heuristics with reduced BIG are the best suited for a one-step full synthesis of FM (see **RQ1**);
- Ontological heuristics over the reduced BIG are the best suited for computing ranking lists (see **RQ2**);
- Logical feature groups with place-holders and computed over the reduced BIG are the most accurate clusters (see **RQ2**);
- Cliques and clusters obtained with ontological heuristics can complete the information presented to users (see **RQ2**).

The empirical results impact feature modeling tools (e.g., WebFML) and call for more investigations in terms of user experience.

As part of WebFML we have integrated *default* synthesis heuristics for (1) one-step full synthesis, (2) ranking lists and (3) clusters. We choose the most suited heuristics according to the empirical results. We now discuss our choices, some tradeoffs, and practical implications of the experiments.

For *one-step full synthesis*, we select *PathLength* with the reduced BIG as the default heuristic (*PathLength* gives the best results for the SPLOT dataset). All heuristics operating over the BIG are inferior to *PathLength* and are not included in WebFML. Statistical tests show no clear superiority of *PathLength* over other ontological heuristics with the reduced BIG or FMFASE. The choice of *PathLength* can thus be replaced by another heuristic (WebFML allows users to choose a specific heuristic).

More importantly, the empirical results show that one-step full synthesis is far from the ground truth. Two attitudes are possible for a user. The first is to *refactor* the synthesized FM and fix the erroneous relations between features. The second is to *not* use a one-step full synthesis. In any case, a crucial feature of WebFML is the ability to provide an interactive support for users (see hereafter).

For the computation of *ranking lists*, *PathLength* and *Wikipedia* heuristics outperform the other heuristics of FMONTO and $\text{FMONTO}_{\text{LOGIC}}$. The two heuristics have very similar results. In WebFML, we choose *PathLength* as the default heuristic. The reason of this choice instead of *Wikipedia* is only technical. The *Wikipedia* heuristic requires a large database (40GB) which makes the deployment harder than with a WordNet based heuristic.

For the computation of *clusters*, two ontological heuristics stand out and have interesting properties: *PathLength* and *Levenshtein*. Yet the choice of a default heuristic requires to consider some tradeoffs. On the one hand, *PathLength* is the most accurate heuristic. On the other hand, the correct clusters produced by *Levenshtein* cover more features. In practice, a user of WebFML needs to review less clusters with *PathLength* but the part of the hierarchy that could be synthesized from these clusters is smaller. Conversely, using *Lev-*

enshtein forces the user to review more clusters but the induced benefit can be more important.

The tradeoff between the number of clusters to review and the accuracy of the heuristics needs to be addressed in a usability study. Currently, *Levenshtein* is the default heuristic retained for computing clusters in WebFML. WebFML does not present logical feature groups by *default*, considering they may overwhelm users with lots of false clusters. Nevertheless users can depict them on demand, typically when choices have already been made and the number of clusters is becoming lower. More generally, users can change heuristics for clusters if needs be and WebFML provides facilities to manage clusters (e.g., non relevant clusters can be removed).

Summary. Ranking lists, logical feature groups, cliques, clusters with ontological heuristics: all are potentially useful for the synthesis; and the better they are, the better it is for users. Based on empirical results, WebFML integrates state-of-the-art techniques and provides advanced facilities for manipulating the information (e.g., unfolding of cliques). Now the overall challenge is to make all these structures visible into an environment – without overwhelming and distracting the user with extraneous or redundant information. The *usability* aspects of an FM synthesis environment like WebFML deserve a focused and careful attention. We leave it as future work.

7 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 [85]. SPLOT is a common benchmark for the FM community (see, e.g., [8,13,47,48,62,72,73,80,97]) 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, a 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 semantics 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 researchers.

Another external threat is that we hypothesize that the user effort is reduced thanks to the branching algorithm, the computation of clusters and ranking lists, and the interactive support. We have not run user experiments

to validate this claim. Such experiments involve usability of WebFML while we focus on the automated techniques. This evaluation is future work.

A first *internal threat* is that the manual selection of the SPLOT FMs was done by the authors. To avoid subjective choices, we clearly defined criteria for rejecting a FM before doing the selection. Moreover, the selection was performed separately by two of the authors and their results were cross-checked.

Another 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). Importantly, 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.

The statistical tests performed in the evaluation form another internal threat. Performing a different test or interpreting differently the results could change the conclusions about the effectiveness of our algorithms and heuristics. However, the Wilcoxon test and the 0.05 threshold for the p-values are classical setups for this kind of evaluation. Moreover, we provide all the p-values in Appendix A so that the reader can make her own interpretation of the results.

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 [6, 13, 39]. We replicated numerous times the empirical study, on different datasets (see, *e.g.*, our technical report using an older SPLOT dataset [22]). Besides the collection of results and statistics is fully automated with R scripts.

8 Related Work

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

8.1 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 [8, 13, 38, 39, 47, 48, 53, 58, 59, 83]. An important limitation of prior works is the identification of the feature hierarchy when synthesizing the FM, that is, the user support is either absent or limited.

In [13,39], the authors calculate a diagrammatic representation of *all possible* FMs, leaving open the selection of the hierarchy and feature groups. Janota *et al.* [53] developed an interactive editor, based on logical techniques, to guide users in synthesizing an FM. The algorithms proposed in [47, 48, 58] do not control the way the feature hierarchy is synthesized either. We demonstrated in Section 6.2 that the ontological semantics of the resulting FMs significantly deviates from the ground truths while a hybrid approach provides better results. In addition no user support is provided in [47, 48, 58, 59] to interactively synthesize or refactor the resulting FM. In [8], we proposed a synthesis procedure that processes user-specified knowledge for organizing the hierarchy of features. The effort may be substantial since users have to review numerous potential parent features (6.3 in average for the FMs of the SPLOT repository and 8.3 for the PCM dataset, see Section 6).

Our study provides empirical evidence that a one step synthesis is unlikely, highlighting the need to interactively support users in synthesizing FMs. The results also further the understanding of strengths and weaknesses of logical components (cliques, feature groups, BIG) computed by some of these approaches.

She *et al.* [83] 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, they assume the existence of feature descriptions in the software projects Linux, eCos, and FreeBSD. In this article we develop a series of alternative techniques for computing ranking lists and also clusters, applicable to any domain. We also perform an empirical study in other domains than the operating system domain.

She *et al.* reported in Section 7 of [83] 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 empirical study confirms that a full synthesis is not adequate either and that the user support is highly needed.*

Davril *et al.* [40] presented a fully automated approach, based on prior work [46], 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 [40].

8.2 FM extraction

Nadi *et al.* [68] developed a comprehensive infrastructure to automatically extract configuration constraints from C code. Their mining procedure can serve as input of our techniques to eventually synthesize a feature model.

In [5], a semi-automated procedure to support the transition from product descriptions (expressed in a tabular format) to FMs is proposed. In [4], 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 [78].

The procedures exposed in [4, 5, 40, 78, 83] 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 it is the case in the matrix of Fig. 3a (cf page 7). Ontological knowledge is also missing when extracting conditional compilation directives from source code and build files [41]. 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.

Yi *et al.* [97] proposed to apply support vector machine and genetic techniques to mine binary constraints (requires and excludes) from Wikipedia. This scenario is particularly relevant when dealing with *incomplete* dependencies. They evaluated their approach on two FMs of SPLOT. Our techniques might be used to *suggest* logical relationships in case strong ontological relations between features are inferred. Yet the idea of passing "from ontology to logic" (rather than "from logic to ontology" as we do in this article) needs to be empirically evaluated.

Vacchi *et al.* [90] proposed an approach to automatically infer a feature model from a collection of already implemented language components. The goal is to automate the configuration of families of (domain-specific) programming languages. The evaluation showed that a substantial effort is needed to incorporate domain (or ontological) knowledge. Given the specific application domain, there are three notable differences: (1) the mined implication graph is a rough over-approximation of the configuration set; (2) the complete list

of features is not a priori known; (3) feature names are quite technical and specific, so that users have to build their own semantic network. Bagheri *et al.* [20] proposed a collaborative process to mine and organize features using a combination of natural language processing techniques and Wordnet. Ferrari *et al.* [44] applied natural language processing techniques to mine commonalities and variabilities from brochures.

Alves *et al.* [12], Niu *et al.* [69], Weston *et al.* [94], and Chen *et al.* [31] 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.

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.

8.3 Tool Support

There are numerous existing academic or industrial tools for specifying and reasoning about FMs. *FeatureIDE* [88, 89] is an Eclipse-based IDE that supports all phases of feature-oriented software development for the development of product lines (domain analysis, domain implementation, requirements analysis and software generation). *FAMA (Feature Model Analyser)* [25] is a framework for the automated analysis of FMs integrating some of the most commonly used logic representations and solvers. SPLOT [61] provides a Web-based environment for editing and configuring FMs. S2T2 [70] is a tool for the configuration of large FMs. Weston et al. [94] provided a tool framework *ArborCraft* which automatically processes natural-language requirements documents into a candidate FM, which can be refined by the requirements engineers. Commercial solutions (*pure::variants* [74] and *Gears* [56]) also provide a comprehensive support for product lines (from FMs to model/source code derivation). Our environment is compatible with some of these tools. However none of the existing tools propose support for synthesizing, merging, slicing, or refactoring FMs.

9 Conclusion

In this article, 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 156 FMs, coming from various domains, demonstrated that a 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, called WebFML, that equips important automated operations for FMs with ontological capabilities. More details can be found in http://tinyurl.com/WebFMLDemo.

As future work, we plan to investigate usability aspects of WebFML 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 [34] or formalisms close to FMs [26].

References

1. SEI's Software Product Line Hall of Fame
2. Abbasi, E.K., Acher, M., Heymans, P., Cleve, A.: Reverse engineering web configurators. In: CSMR/WRCE'14 (2014)
3. Acher, M., Cleve, A., Collet, P., Merle, P., Duchien, L., Lahire, P.: Reverse engineering architectural feature models. In: ECSA'11, *LNCS*, vol. 6903, pp. 220–235 (2011)
4. Acher, M., Cleve, A., Collet, P., Merle, P., Duchien, L., Lahire, P.: Extraction and evolution of architectural variability models in plugin-based systems. Software and Systems Modeling (SoSyM) (2014)
5. Acher, M., Cleve, A., Perrouin, G., Heymans, P., Vanbeneden, C., Collet, P., Lahire, P.: On extracting feature models from product descriptions. In: VaMoS'12, pp. 45–54. ACM (2012)
6. Acher, M., Collet, P., Lahire, P., France, R.: Familiar: A domain-specific language for large scale management of feature models. Science of Computer Programming (SCP) **78**(6), 657–681 (2013)
7. Acher, M., Combemale, B., Collet, P., Barais, O., Lahire, P., France, R.B.: Composing your compositions of variability models. In: MoDELS'13, pp. 352–369 (2013)
8. Acher, M., Heymans, P., Cleve, A., Hainaut, J.L., Baudry, B.: Support for reverse engineering and maintaining feature models. In: VaMoS'13. ACM (2013)
9. Ahnassay, A., Bagheri, E., Gasevic, D.: Empirical evaluation in software product line engineering. Tech. Rep. TR-LS3-130084R4T, Laboratory for Systems, Software and Semantics, Ryerson University (2013)
10. Aho, A.V., Garey, M.R., Ullman, J.D.: The transitive reduction of a directed graph. SIAM Journal on Computing **1**(2), 131–137 (1972)
11. Algorithm of Haslinger et al. [48]: http://www.jku.at/sea/content/e139529/e126342/e188736/
12. Alves, V., Schwanninger, C., Barbosa, L., Rashid, A., Sawyer, P., Rayson, P., Pohl, C., Rummler, A.: An exploratory study of information retrieval techniques in domain analysis. In: SPLC'08, pp. 67–76. IEEE (2008)
13. Andersen, N., Czarnecki, K., She, S., Wasowski, A.: Efficient synthesis of feature models. In: Proceedings of SPLC'12, pp. 97–106. ACM (2012)

14. Apel, S., Batory, D., Kästner, C., Saake, G.: Feature-Oriented Software Product Lines: Concepts and Implementation. Springer-Verlag (2013)
15. Apel, S., Kästner, C.: An overview of feature-oriented software development. Journal of Object Technology (JOT) **8**(5), 49–84 (2009)
16. Apel, S., Kästner, C., Lengauer, C.: Language-independent and automated software composition: The featurehouse experience. IEEE Transactions on Software Engineering (TSE) **39**, 63–79 (2013)
17. Apel, S., von Rhein, A., Wendler, P., Größlinger, A., Beyer, D.: Strategies for product-line verification: Case studies and experiments. In: ICSE'13. IEEE (2013)
18. Arcuri, A., Briand, L.: A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: Proceedings of the 33rd International Conference on Software Engineering, ICSE '11, pp. 1–10. ACM, New York, NY, USA (2011)
19. Baader, F., Nutt, W.: The description logic handbook. chap. Basic Description Logics, pp. 43–95. Cambridge University Press, New York, NY, USA (2003)
20. Bagheri, E., Ensan, F., Gasevic, D.: Decision support for the software product line domain engineering lifecycle. Automated Software Engineering **19**(3), 335–377 (2012)
21. Bagheri, E., Gasevic, D.: Assessing the maintainability of software product line feature models using structural metrics. Software Quality Journal **19**(3), 579–612 (2011)
22. Bécan, G., Acher, M., Baudry, B., Ben Nasr, S.: Breathing Ontological Knowledge Into Feature Model Management. Rapport Technique RT-0441, INRIA (2013). URL `http://hal.inria.fr/hal-00874867`
23. Bécan, G., Nasr, S.B., Acher, M., Baudry, B.: WebFML: Synthesizing Feature Models Everywhere. In: SPLC'14 (2014)
24. Bécan, G., Sannier, N., Acher, M., Barais, O., Blouin, A., Baudry, B.: Automating the formalization of product comparison matrices. In: Proceedings of the 29th ACM/IEEE international conference on Automated software engineering, pp. 433–444. ACM (2014)
25. Benavides, D., Segura, S., Ruiz-Cortes, A.: Automated analysis of feature models 20 years later: a literature review. Information Systems **35**(6) (2010)
26. Berger, T., She, S., Lotufo, R., Wasowski, A., Czarnecki, K.: A study of variability models and languages in the systems software domain. IEEE Transactions on Software Engineering (2013)
27. 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)
28. Boucher, Q., Abbasi, E., Hubaux, A., Perrouin, G., Acher, M., Heymans, P.: Towards more reliable configurators: A re-engineering perspective. In: PLEASE'12 Int'l workshop at ICSE'12, (2012)
29. Budanitsky, A., Hirst, G.: Evaluating wordnet-based measures of lexical semantic relatedness. Computational Linguistics **32**(1), 13–47 (2006)
30. Camerini, P., Fratta, L., Maffioli, F.: A note on finding optimum branchings. Networks **9**(4), 309–312 (1979)
31. Chen, K., Zhang, W., Zhao, H., Mei, H.: An approach to constructing feature models based on requirements clustering. In: RE'05, pp. 31–40 (2005)
32. Classen, A., Heymans, P., Schobbens, P.Y., Legay, A.: Symbolic model checking of software product lines. In: ICSE'11, pp. 321–330. ACM (2011)
33. Classen, A., Heymans, P., Schobbens, P.Y., Legay, A., Raskin, J.F.: Model checking lots of systems: efficient verification of temporal properties in software product lines. In: ICSE'10, pp. 335–344. ACM (2010)
34. Cordy, M., Schobbens, P.Y., Heymans, P., Legay, A.: Beyond boolean product-line model checking: dealing with feature attributes and multi-features. In: ICSE'13, pp. 472–481 (2013)
35. Czarnecki, K., Eisenecker, U.: Generative Programming: Methods, Tools, and Applications. Addison-Wesley (2000)
36. Czarnecki, K., Kim, C.H.P., Kalleberg, K.T.: Feature models are views on ontologies. In: SPLC '06, pp. 41–51. IEEE (2006)
37. Czarnecki, K., Pietroszek, K.: Verifying feature-based model templates against well-formedness ocl constraints. In: GPCE'06, pp. 211–220. ACM (2006)
38. Czarnecki, K., She, S., Wasowski, A.: Sample spaces and feature models: There and back again. In: SPLC'08, pp. 22–31 (2008)

39. Czarnecki, K., Wasowski, A.: Feature diagrams and logics: There and back again. In: SPLC'07, pp. 23–34. IEEE (2007)
40. Davril, J.M., Delfosse, E., Hariri, N., Acher, M., Cleland-Huang, J., Heymans, P.: Feature model extraction from large collections of informal product descriptions. In: ESEC/FSE'13 (2013)
41. Dietrich, C., Tartler, R., Schröder-Preikschat, W., Lohmann, D.: A robust approach for variability extraction from the linux build system. In: SPLC'12, pp. 21–30 (2012)
42. FAMILIAR: FeAture Model scrIpt Language for manIpulation and Automatic Reasoning: `http://nyx.unice.fr/projects/familiar/`
43. Fan, S., Zhang, N.: Feature model based on description logics. In: B. Gabrys, R. Howlett, L. Jain (eds.) Knowledge-Based Intelligent Information and Engineering Systems, *Lecture Notes in Computer Science*, vol. 4252, pp. 1144–1151. Springer Berlin Heidelberg (2006)
44. Ferrari, A., Spagnolo, G.O., dell'Orletta, F.: Mining commonalities and variabilities from natural language documents. In: T. Kishi, S. Jarzabek, S. Gnesi (eds.) SPLC, pp. 116–120. ACM (2013)
45. Gruber, T.R.: A translation approach to portable ontology specifications. Knowl. Acquis. **5**(2), 199–220 (1993)
46. Hariri, N., Castro-Herrera, C., Mirakhorli, M., Cleland-Huang, J., Mobasher, B.: Supporting domain analysis through mining and recommending features from online product listings. IEEE Transactions on Software Engineering (2013)
47. Haslinger, E.N., Lopez-Herrejon, R.E., Egyed, A.: Reverse engineering feature models from programs' feature sets. In: WCRE'11, pp. 308–312. IEEE (2011)
48. Haslinger, E.N., Lopez-Herrejon, R.E., Egyed, A.: On extracting feature models from sets of valid feature combinations. In: FASE'13, *LNCS*, vol. 7793, pp. 53–67 (2013)
49. Heidenreich, F., Sanchez, P., Santos, J., Zschaler, S., Alferez, M., Araujo, J., Fuentes, L., amd Ana Moreira, U.K., Rashid, A.: 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 **6210**, 69–114 (2010)
50. Heule, M.J.H., Järvisalo, M., Biere, A.: Efficient cnf simplification based on binary implication graphs. In: Proceedings of the 14th International Conference on Theory and Application of Satisfiability Testing, SAT'11, pp. 201–215. Springer-Verlag, Berlin, Heidelberg (2011)
51. Hubaux, A., Acher, M., Tun, T.T., Heymans, P., Collet, P., Lahire, P.: Domain Engineering: Product Lines, Conceptual Models, and Languages, chap. Separating Concerns in Feature Models: Retrospective and Multi-View Support. Springer (2013)
52. Hubaux, A., Tun, T.T., Heymans, P.: Separation of concerns in feature diagram languages: A systematic survey. ACM Computing Surveys (2013)
53. Janota, M., Kuzina, V., Wasowski, A.: Model construction with external constraints: An interactive journey from semantics to syntax. In: MODELS'08, *LNCS*, vol. 5301, pp. 431–445 (2008)
54. Kang, K., Lee, J., Donohoe, P.: Feature-oriented product line engineering. Software, IEEE **19**(4), 58–65 (2002)
55. Kästner, C., Dreiling, A., Ostermann, K.: Variability mining: Consistent semiautomatic detection of product-line features. IEEE Transactions on Software Engineering (2013)
56. Krueger, C.W.: Biglever software Gears and the 3-tiered spl methodology. In: OOPSLA'07, pp. 844–845. ACM (2007)
57. Linden, F.J.v.d., Schmid, K., Rommes, E.: Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2007)
58. 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'12, *LNCS*, vol. 7515, pp. 168–182. Springer (2012)
59. Lopez-Herrejon, R.E., Linsbauer, L., Galindo, J.A., Parejo, J.A., Benavides, D., Segura, S., Egyed, A.: An assessment of search-based techniques for reverse engineering feature models. Journal of Systems and Software (2014)
60. Medelyan, O., Milne, D.N., Legg, C., Witten, I.H.: Mining meaning from wikipedia. Int. J. Hum.-Comput. Stud. **67**(9), 716–754 (2009)

61. Mendonca, M., Branco, M., Cowan, D.: S.p.l.o.t.: software product lines online tools. In: OOPSLA'09 (companion). ACM (2009)
62. Mendonca, M., Wasowski, A., Czarnecki, K.: SAT-based analysis of feature models is easy. In: SPLC'09, pp. 231–240. IEEE (2009)
63. Metzger, A., Pohl, K., Heymans, P., Schobbens, P.Y., Saval, G.: Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis. In: RE'07, pp. 243–253 (2007)
64. Miller, G.A.: Wordnet: a lexical database for english. Communications of the ACM **38**(11), 39–41 (1995)
65. Milne, D.: Computing semantic relatedness using wikipedia link structure. In: the new zealand computer science research student conference. Citeseer (2007)
66. Milne, D.N., Witten, I.H.: An open-source toolkit for mining wikipedia. Artif. Intell. **194**, 222–239 (2013)
67. Mussbacher, G., Araújo, J., Moreira, A., Amyot, D.: Aourn-based modeling and analysis of software product lines. Software Quality Journal **20**(3-4), 645–687 (2012)
68. Nadi, S., Berger, T., Kästner, C., Czarnecki, K.: Mining configuration constraints: Static analyses and empirical results. In: Proceedings of the 36th International Conference on Software Engineering (ICSE) (2014)
69. Niu, N., Easterbrook, S.M.: Concept analysis for product line requirements. In: K.J. Sullivan, A. Moreira, C. Schwanninger, J. Gray (eds.) AOSD'09, pp. 137–148. ACM (2009)
70. Pleuss, A., Botterweck, G.: Visualization of variability and configuration options. International Journal on Software Tools for Technology Transfer **14**(5), 497–510 (2012)
71. Pohl, K., Böckle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag (2005)
72. Pohl, R., Lauenroth, K., Pohl, K.: A performance comparison of contemporary algorithmic approaches for automated analysis operations on feature models. In: ASE'11, pp. 313–322 (2011)
73. Pohl, R., Stricker, V., Pohl, K.: Measuring the structural complexity of feature models. In: ASE'13 (2013)
74. pure::variants: `http://www.pure-systems.com/pure_variants.49.0.html`
75. Rabkin, A., Katz, R.: Static extraction of program configuration options. In: ICSE'11, pp. 131–140. ACM (2011)
76. Rubin, J., Chechik, M.: Locating distinguishing features using diff sets. In: ASE'12, pp. 242–245. ACM (2012)
77. Rubin, J., Chechik, M.: Domain Engineering: Product Lines, Conceptual Models, and Languages, chap. A Survey of Feature Location Techniques. Springer (2013)
78. Ryssel, U., Ploennigs, J., Kabitzsch, K.: Extraction of feature models from formal contexts. In: FOSD'11, pp. 1–8 (2011)
79. Sannier, N., Acher, M., Baudry, B.: From Comparison Matrix to Variability Model: The Wikipedia Case Study. In: ASE'13. IEEE (2013)
80. Sayyad, A.S., Menzies, T., Ammar, H.: On the value of user preferences in search-based software engineering: a case study in software product lines. In: ICSE'13, pp. 492–501 (2013)
81. Schobbens, P.Y., Heymans, P., Trigaux, J.C., Bontemps, Y.: Generic semantics of feature diagrams. Comput. Netw. **51**(2), 456–479 (2007)
82. She, S.: Feature Model Synthesis. Ph.D. thesis, University of Waterloo (2013)
83. She, S., Lotufo, R., Berger, T., Wasowski, A., Czarnecki, K.: Reverse engineering feature models. In: ICSE'11, pp. 461–470. ACM (2011)
84. Smith, T., Waterman, M.: Identification of common molecular subsequences. Molecular Biology **147**, 195–197 (1981)
85. SPLOT: Software Product Line Online Tools: `http://www.splot-research.org/`
86. Tarjan, R.E.: Finding optimum branchings. Networks **7**(1), 25–35 (1977)
87. Thaker, S., Batory, D., Kitchin, D., Cook, W.: Safe composition of product lines. In: GPCE '07, pp. 95–104. ACM, New York, NY, USA (2007)
88. Thüm, T., Batory, D., Kästner, C.: Reasoning about edits to feature models. In: ICSE'09, pp. 254–264. ACM (2009)

89. Thüm, T., Kstner, C., Benduhn, F., Meinicke, J., Saake, G., Leich, T.: Featureide: An extensible framework for feature-oriented software development. Science of Computer Programming (2012)
90. Vacchi, E., Combemale, B., Cazzola, W., Acher, M.: Automating Variability Model Inference for Component-Based Language Implementations. In: 18th International Software Product Line Conference (SPLC'14) (2014)
91. Valente, M.T., Borges, V., Passos, L.: A semi-automatic approach for extracting software product lines. IEEE Transactions on Software Engineering **38**(4), 737–754 (2012)
92. Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. Journal of the ACM (JACM) **21**(1), 168–173 (1974)
93. Wang, H.H., Li, Y.F., Sun, J., Zhang, H., Pan, J.: Verifying feature models using owl. Web Semant. **5**(2), 117–129 (2007)
94. Weston, N., Chitchyan, R., Rashid, A.: A framework for constructing semantically composable feature models from natural language requirements. In: SPLC'09, pp. 211–220. ACM (2009)
95. Wu, Z., Palmer, M.: Verbs semantics and lexical selection. In: the 32nd annual meeting on Association for Computational Linguistics, pp. 133–138. Association for Computational Linguistics (1994)
96. Wulf-Hadash, O., Reinhartz-Berger, I.: Cross product line analysis. In: VaMoS'13. ACM (2013)
97. Yi, L., Zhang, W., Zhao, H., Jin, Z., Mei, H.: Mining binary constraints in the construction of feature models. In: RE'12, pp. 141–150. IEEE (2012)

A Detailed Results of Statistical Tests

Numerous statistical tests were performed to evaluate our FM synthesis algorithm and heuristics (see Section 6 for further details). In this appendix, we present the comprehensive results of the tests. In particular, we report all the p-values and effect sizes that were computed.

There are two kinds of tables in this appendix. The first kind compares ontological techniques (displayed on top of the table) to purely logical techniques (displayed on the left of the table). The p-value corresponds to the comparison of an ontological technique with a logical one. The effect size is the difference between the mean score of the ontological technique and the mean score of the logical one. It means that if the effect size is positive, the ontological technique outperforms the logical one whereas if the effect size is negative, it is the opposite relation.

The second kind of tables compares each heuristic performing on the complete BIG with the same heuristic on the reduced BIG. The p-value corresponds to this comparison and the effect size is the difference between the mean score of the heuristic on the reduced BIG and the mean score on the complete BIG. It means that if the effect size is positive, the same heuristic performs better on the reduced BIG than on the complete BIG. If the effect size is negative, the reduced BIG has a negative impact on the heuristic.

Each table is related as follows to an hypothesis in Section 6:

H1 Table 7a compares FMONTO with $\text{FMRAND}_{\text{BIG}}$ for automatically synthesizing an FM while Table 7b compares $\text{FMONTO}_{\text{LOGIC}}$ with $\text{FMRAND}_{\text{RBIG}}$.

H2 Table 8 compares $\text{FMRAND}_{\text{BIG}}$ and FMONTO with respectively $\text{FMRAND}_{\text{RBIG}}$ and $\text{FMONTO}_{\text{LOGIC}}$ on the fully automated synthesis.

H3 Table 9a compares FMONTO with $\text{FMRAND}_{\text{BIG}}$ for computing ranking lists while Table 9b compares $\text{FMONTO}_{\text{LOGIC}}$ with $\text{FMRAND}_{\text{RBIG}}$.

H4 Table 10 compares $\text{FMRAND}_{\text{BIG}}$ and FMONTO with respectively $\text{FMRAND}_{\text{RBIG}}$ and $\text{FMONTO}_{\text{LOGIC}}$ on the computation of ranking lists.

H5 Table 11a (resp. Table 11c) compares FMONTO with $\text{FMRAND}_{\text{BIG}}$ on the percentage of correct clusters (resp. percentage of features in a correct cluster) generated by the heuristics. Table 11b and 11d present the same results but for $\text{FMONTO}_{\text{LOGIC}}$ and $\text{FMRAND}_{\text{RBIG}}$.

H6 Table 12a compares $\text{FMRAND}_{\text{BIG}}$ and FMONTO with respectively $\text{FMRAND}_{\text{RBIG}}$ and $\text{FMONTO}_{\text{LOGIC}}$ on the percentage of generated correct clusters. Table 12b presents the same comparison but for the percentage of features in a correct cluster.

H7 Table 13a compares the percentage of correct feature groups generated from the BIG with feature groups generated from the reduced BIG. Table 13b presents the same comparison but for the percentage of features in a correct group.

Table 7: H1 - Full synthesis

(a) Full synthesis with BIG

Reference	Data set	Result	Ontological techniques (FMONTO)					
			SW	L	WP	PL	Wiki	Wikt
$\text{FMRAND}_{\text{BIG}}$	SPLOT	p-value	0.005	0.226	0.003	0.000	0.000	0.002
		effect size	7.9	6.0	8.2	9.5	8.5	7.6
	PCM	p-value	0.005	0.341	0.192	0.046	0.002	0.043
		effect size	8.1	3.5	4.9	7.4	10.4	6.9

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

Reference	Data set	Result	Ontological techniques ($\text{FMONTO}_{\text{LOGIC}}$)					
			SW	L	WP	PL	Wiki	Wikt
$\text{FMRAND}_{\text{RBIG}}$	SPLOT	p-value	0.041	0.409	0.066	0.010	0.012	0.045
		effect size	5.7	3.6	5.2	6.5	6.2	5.4
	PCM	p-value	0.511	0.909	0.700	0.483	0.350	0.641
		effect size	3.1	-0.1	2.6	3.6	4.4	2.2
FMFASE	SPLOT	p-value	0.840	0.474	0.939	0.544	0.644	0.876
		effect size	0.8	-1.2	0.4	1.6	1.3	0.6
	PCM	p-value	●					
		effect size						
FMFASE_{SAT}	SPLOT	p-value	0.054	0.393	0.086	0.021	0.028	0.055
		effect size	5.0	3.0	4.5	5.8	5.5	4.7
	PCM	p-value	0.633	0.812	0.829	0.620	0.438	0.761
		effect size	3.3	0.0	2.7	3.7	4.5	2.3

Table 8: H2 - Full synthesis (BIG vs reduced BIG)

Data set	Result	Pure logical techniques	Ontological techniques (FMONTO)					
		$\text{FMRAND}_{\text{BIG}}$	SW	L	WP	PL	Wiki	Wikt
SPLOT	p-value	0.000	0.004	0.009	0.013	0.010	0.003	0.004
	effect size	10.4	8.2	8.0	7.4	7.4	8.0	8.2
PCM	p-value	0.000	0.000	0.001	0.000	0.000	0.002	0.000
	effect size	24.1	19.1	20.5	21.8	20.3	18.1	19.5

Table 9: H3 - Top 2

(a) Top 2 with BIG

Reference	Data set	Result	Ontological techniques (FMONTO)					
			SW	L	WP	PL	Wiki	Wikt
$\text{FMRAND}_{\text{BIG}}$	SPLOT	p-value	0.021	0.022	0.007	0.000	0.007	0.009
		effect size	7.0	6.5	8.7	10.3	8.4	7.2
	PCM	p-value	0.103	0.213	0.343	0.065	0.021	0.246
		effect size	7.5	5.0	4.7	7.4	9.4	5.7

(b) Top 2 with a reduced BIG

Reference	Data set	Result	Ontological techniques ($\text{FMONTO}_{\text{LOGIC}}$)					
			SW	L	WP	PL	Wiki	Wikt
$\text{FMRAND}_{\text{RBIG}}$	SPLOT	p-value	0.126	0.161	0.076	0.016	0.073	0.105
		effect size	4.8	4.0	5.5	7.4	5.6	4.9
	PCM	p-value	0.569	0.980	0.695	0.417	0.483	0.783
		effect size	2.7	0.8	2.8	4.1	3.4	2.0

Table 10: H4 - Top 2 (BIG vs reduced BIG)

Data set	Result	Pure logical techniques	Ontological techniques (FMONTO)					
		$FMRAND_{BIG}$	SW	L	WP	PL	Wiki	Wikt
SPLOT	p-value	0.004	0.048	0.059	0.082	0.060	0.109	0.045
	effect size	8.6	6.4	6.1	5.4	5.7	5.7	6.3
PCM	p-value	0.000	0.015	0.016	0.004	0.008	0.031	0.006
	effect size	19.9	15.1	15.7	18.0	16.5	14.0	16.3

Table 11: H5 - Clusters generated by heuristics

(a) Percentage of correct clusters with BIG

Reference	Data set	Result	Ontological techniques (FMONTO)					
			SW	L	WP	PL	Wiki	Wikt
$FMRAND_{BIG}$	SPLOT	p-value	0.000	0.000	0.000	0.000	0.000	0.000
		effect size	22.7	27.6	29.6	39.9	26.9	39.1
	PCM	p-value	0.000	0.000	0.000	0.000	0.000	0.000
		effect size	34.7	47.7	45.9	64.3	38.9	56.1

(b) Percentage of correct clusters with a reduced BIG

Reference	Data set	Result	Ontological techniques ($FMONTO_{LOGIC}$)					
			SW	L	WP	PL	Wiki	Wikt
$FMRAND_{RBIG}$	SPLOT	p-value	0.000	0.000	0.000	0.000	0.000	0.000
		effect size	25.1	26.1	23.6	32.0	26.3	33.6
	PCM	p-value	0.000	0.000	0.000	0.000	0.000	0.000
		effect size	31.5	39.2	36.6	52.5	35.6	48.9

(c) Percentage of features in a correct cluster with BIG

Reference	Data set	Result	Ontological techniques (FMONTO)					
			SW	L	WP	PL	Wiki	Wikt
$FMRAND_{BIG}$	SPLOT	p-value	0.000	0.000	0.678	0.545	0.008	0.036
		effect size	10.0	12.1	0.6	2.9	6.8	5.6
	PCM	p-value	0.000	0.000	0.944	0.000	0.000	0.000
		effect size	22.5	28.9	0.4	10.3	12.0	19.7

(d) Percentage of features in a correct cluster with a reduced BIG

Reference	Data set	Result	Ontological techniques ($FMONTO_{LOGIC}$)					
			SW	L	WP	PL	Wiki	Wikt
$FMRAND_{RBIG}$	SPLOT	p-value	0.000	0.000	0.793	0.812	0.033	0.111
		effect size	8.8	11.3	0.9	1.9	6.0	4.6
	PCM	p-value	0.000	0.000	0.944	0.000	0.000	0.000
		effect size	22.3	28.7	0.5	10.4	11.9	19.8

Table 12: H6 - Clusters generated by heuristics (BIG vs reduced BIG)

(a) Percentage of correct clusters

Data set	Result	Pure logical techniques	Ontological techniques (FMONTO)					
		$\text{FMRAND}_{\text{BIG}}$	SW	L	WP	PL	Wiki	Wikt
SPLOT	p-value	0.000	0.000	0.002	0.111	0.191	0.001	0.040
	effect size	13.4	15.8	12.0	7.4	5.5	12.8	8.0
PCM	p-value	0.000	0.001	0.002	0.045	0.007	0.001	0.001
	effect size	21.9	18.6	13.3	12.5	10.1	18.6	14.7

(b) Percentage of features in a correct cluster

Data set	Result	Pure logical techniques	Ontological techniques (FMONTO)					
		$\text{FMRAND}_{\text{BIG}}$	SW	L	WP	PL	Wiki	Wikt
SPLOT	p-value	0.220	0.294	0.506	0.694	0.348	0.403	0.375
	effect size	-1.1	-2.3	-1.8	-0.8	-2.1	-1.9	-2.1
PCM	p-value	0.866	0.956	0.962	0.950	0.997	0.915	1.000
	effect size	-0.1	-0.4	-0.3	-0.1	-0.1	-0.3	0.0

Table 13: H7 - Feature groups (BIG vs reduced BIG)

(a) Percentage of correct clusters

Data set	Result	All groups	Mutex	Xor
SPLOT	p-value	0.008	0.010	0.257
	effect size	10.7	27.4	3.0
PCM	p-value	0.024	0.051	0.233
	effect size	7.8	8.5	8.9

(b) Percentage of features in a correct cluster

Data set	Result	All groups	Mutex	Xor
SPLOT	p-value	0.398	0.013	0.143
	effect size	-3.7	-3.2	-5.2
PCM	p-value	0.761	0.395	0.670
	effect size	3.6	-7.3	2.7