Sample Spaces and Feature Models: There and Back Again

Krzysztof Czarnecki, Steven She
University of Waterloo, Canada
{kczarnec, shshe}@swen.uwaterloo.ca

Andrzej Wąsowski
IT University of Copenhagen, Denmark
wasowski@itu.dk

Abstract

We present probabilistic feature models *(PFMs) and illustrate their use by discussing modeling, mining and interactive configuration. PFMs are formalized as a set of formulas in a certain probabilistic logic. Such formulas can express both hard and soft constraints and have a well defined semantics by denoting a set of joint probability distributions over features. We show how PFMs can be mined from a given set of feature configurations using data mining techniques. Finally, we demonstrate how PFMs can be used in configuration in order to provide automated support for choice propagation based on both hard and soft constraints. We believe that these results constitute solid foundations for the construction of reverse engineering tools for software product lines and configurators using soft constraints.*

1 Introduction

Feature models [24] represent common and varying product characteristics in a product line. Figure 1a presents a model of a simple family of vehicles. It contains cars with a manual or automatic gear, sold in a regular and North American variety. In general, a feature model consists of a feature diagram, which is a hierarchy of mandatory (gear), optional (drive by wire, for North America), and alternative (manual, automatic) features, and possibly additional constraints such as feature implications (drive by wire implies automatic) or mutual exclusions. Effectively a model defines a set of feature configurations, each representing a particular product in the family.

Basic feature models had been related to propositional logic [6, 14]. A logic formula is obtained from conjoining the constraints represented by the feature diagram and the additional constraints. It defines a set of legal feature configurations for the model. The translation of feature models into propositional logic has allowed the use of reasoning tools for automated feature model analyses, such as checking consistency and identifying dead features [7],

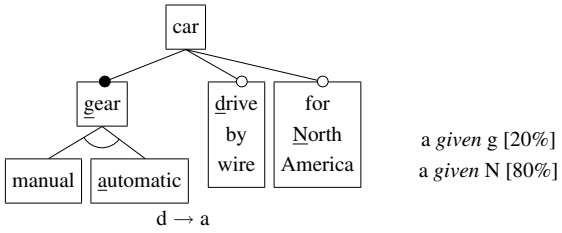


Figure 1. Feature model and soft constraints

product configuration, such as choice propagation and autocompletion [13], and feature model refactoring [2].

Although basic feature models effectively describe legal and illegal configurations of features, they cannot express preference among the legal configurations. A model can postulate that a car has an automatic or a manual gear by marking them as alternative features; creating an XOR-group. However, a basic feature model *cannot* express the fact that in North America most of the cars have an automatic gear, while only a small fraction is sold with a manual transmission. The former is an example of a *hard constraint* that is satisfied by *all* legal configurations, while the latter is an example of a *soft constraint* that should be obeyed by *most* of the legal configurations, but some may disregard it.

Interactive configuration [30, 17] uses constraints to propagate configuration choices made by the user. For example, if the user selects automatic gear, the manual gear is eliminated since both features are exclusive in our example. However, without soft constraints, selecting North America as a market has no consequence for the choice of a gear.

Product configuration can be further automated by supplementing hard constraints with soft constraints. In our example, a soft constraint from North America to automatic gear would enable preselecting automatic gear when a user selects North America. Since the gear choice is just a computed default, the user can still override it to manual gear even though North America remains selected.

Probabilistic feature models (PFMs) extend feature models with soft constraints. Consider again our example, now extended with the soft constraints in Figure 1b.

These constraints specify that 20% of cars are sold with an automatic gear, while this number gets as high as 80% in North America. Semantically, a soft constraint of the form "automatic *given* gear [20%]" specifies the conditional probability of a configuration to contain automatic given that it contains gear to be 0.2. We give a precise semantics of PFMs via translation into *probabilistic propositional logic* [19, 31, 27], which can be understood as a generalization of propositional logic. In the probabilistic context, a propositional formula representing a hard constraint can be thought as a probabilistic formula with an attached probability of one.

After defining and discussing the semantics of PFMs we shall address the problem of retrieving PFMs from a sample set of configurations, or in other words *feature model mining*—a technology aimed at reverse engineering models from a set of products. Feature model mining can be used to identify product parts that should be supported by a common platform, or to analyze how an existing platform is used by several products. It can be applied to a variety of artifacts, including source code, design models, or sales data, in order to produce a range of feature models, such as design or marketing-oriented ones. The first step of the mining process involves extracting the feature configuration of each product in the given product set. This step may involve static or dynamic code analysis or queries to a sales database. The desired feature model can then be generated from the configuration set using the procedure given in this paper.

Finally, we explain how choice propagation and autocompletion can be implemented for PFMs to support interactive product configuration with soft constraints. The PFMs used in this context can be either specified a priori by an expert or they could be obtained by mining. Choice propagation algorithms, if applied to the example of Figure 1, would first conclude that the manual gear is the default feature for an arbitrary car, but would be able to revise this choice to an automatic gear, as soon as a variant oriented for North America is selected.

We strongly believe that the present work constitutes a solid foundation for (1) modeling soft constraints in feature models, (2) constructing product analysis and reverse engineering tools for software product lines, and (3) improving interactive configuration using soft constraints. We connect feature modeling to a large and well-founded body of work in probabilistic logic, belief networks, and machine learning. Furthermore, we identifying algorithms for performing feature model mining and automated configuration in the presence of soft constraints. These algorithms are known to scale to real-world applications outside of feature modeling and our initial experiments show that they are also readily applicable in the present context.

While the presented ideas open up new directions in variability modeling for product lines, transferring these ideas into practice requires further work. We discuss several research directions and challenges, such as providing support for fusing mined and expert-provided knowledge.

2 Probabilistic Feature Models by Example

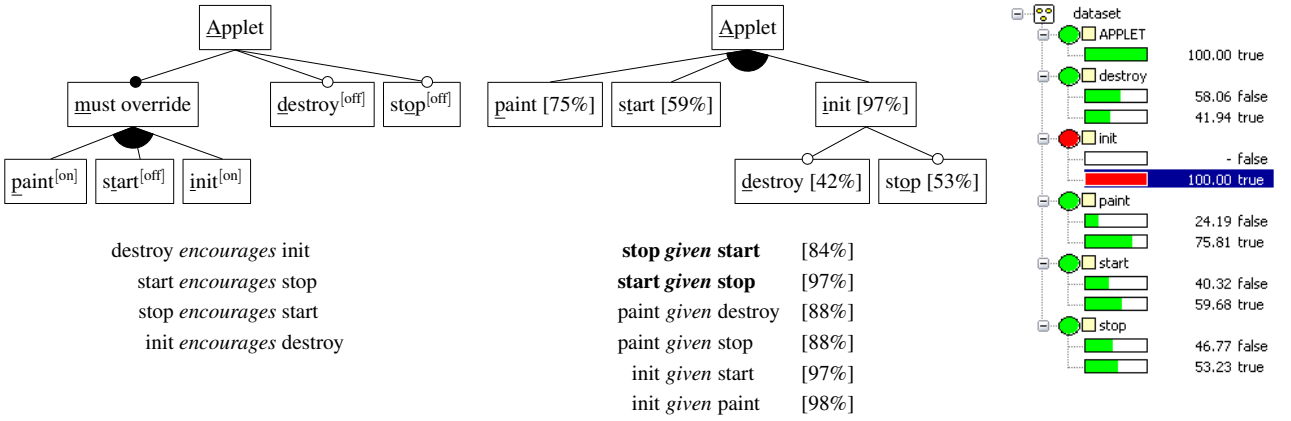
We start by giving a broad perspective on PFMs by using an example from framework-based development. We shall look at a family of Java applets [35], and in particular, on how they use the applet framework—on which methods should be overriden and in which combinations. Although the considered units of variability are very fine-grained, the example illustrates well the basic variability structures that are encountered in the context of a product family that is based on a common platform.

An applet is simply a Java class that extends the Applet class and overrides at least one of the methods: paint, start, or init. The init method is used for initialization, paint is used for redrawing the applet UI, and start/stop are used to initiate and stop the main control thread, for applets that should not be operating when not visible on screen. Finally, destroy should be provided to free any resources claimed by init, however, this is not strictly enforced. In fact, the framework does not even guarantee that destroy will be called, when the applet is terminated by its user.

Figure 2a presents a feature model capturing the above requirements for Java applets. The model has been designed by a domain expert, who has not participated directly in writing of this paper. The expert has based the design on his experience and official documentation [36, 35].

The model introduces an abstract OR-group "must override" that groups the features normally required by an applet: overloading of init, paint or start. The hierarchy also includes overloading of stop and destroy as optional. Finally, several soft constraints are added. In particular, overloading of paint and init are on-by-default, while all the other overloadings are off-by-default. The presence of destroy encourages overloading of init, as destroy expects some resources to be reserved during initialization. Similarly, init encourages destroy since if init is used, then it likely allocated some resources that should be freed. Such a model could be used in a modern development environment to guide designers of applets. See details in [5].

The soft constraints in this example have been expressed using a linguistic form of *defaults*, specifically well suited to expressing *beliefs*. Defaults can be formally understood as probability intervals for probabilistic constraints, e.g., $[0.8, 1]$. For example, "a *encourages* b" means that b is selected with high probability given that a is selected. Also, if some feature c is on-by-default then that feature is selected with high probability given that its parent is selected.



(a) A PFM specified by an expert (b) An automatically mined PFM (c) Interactive Configuration

Figure 2. A PFM of Java applets: specification, mining, and configuration

Apart from being interpreted as belief measures, probabilities can also be given a frequentist interpretation, in which a probability is viewed as the relative frequency of seeing a particular outcome of an experiment in a large number of trials. This perspective on probability naturally fits the mining applications. Figure 2b presents a PFM for the applet domain mechanically mined from a sample of 64 applets[1]. This model has been obtained by applying the procedure presented in Section 6. It is instructive to discuss the differences and similarities between these two models.

First, the mined model lacks the "must override" group as the automatic mining procedure is not able to introduce abstractions. We believe that such improvements need to be done by experts themselves. Secondly, probabilistic dependencies are specified using probabilistic logics constraints (and not linguistic conventions). Notice for example, the additional constraint stop *given* start with the conditional probability of 84% in the mined model corresponds to start *encourages* stop in the expert model. The percentages in nodes indicate the strengths of parent-child relationships. For example, the conditional probability of destroy *given* init is 42%. Our expert says that encourage and on-by-default rules should be created if the conditional probability is at least 80%. Furthermore, the off-by-default rules are should be created when the probability is less than 50%. The mining engine was tuned accordingly for this example.

Regardless of whether the PFM has been mined or designed it can support automatic derivation of configurations by means of choice-propagation, auto-completion, and default propagation. Figure 2c presents a possible sight of a configuration tool—a screen shot created using the Hugin [28] tool, which has been used as a configuration engine in our project. In the figure, a user has just selected to overload init—the highlighted bar signifies that this feature is now included with complete certainty. The other bars show the probabilities of the remaining features. This could of course be visualized differently; for example by updating the most probable defaults in a form, or by prioritizing several most likely choices at the top of a drop-down list.

The probability of stop is around 50%, which indicates almost no preference for or against it (no information). We expect, however, that as soon as the user chooses to overload start, the probability bar of stop will increase significantly towards *true*. Indeed, in such a case Hugin reports 86%.

3 Semantics of PFMs

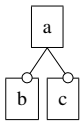
Before delving into the intricacies of PFMs, it is worth recalling the semantics of ordinary feature models as defined via translation to propositional logic [6, 14] (for more general discussion of feature model semantics see Schobbens et al. [34]. An ordinary feature model, such as the one in Figure 3a, denotes a set of *legal configurations*, as shown in Figure 3c. A *legal configuration* is a set of features selected from the feature model according to its semantics. The set of legal configurations is given by conjoining *propositional* formulas defined over a set of features. The set of propositional formulas to be conjoined is systematically constructed for a given feature model. It contains (i) the root feature,[2] (ii) implications from all subnodes to their parents, (iii) additional implications from parents to all their mandatory features, (iv) implications from parents to groups (as defined below), and (v) any additional constraints represented as propositional formulas. An implication from a parent feature p to its subfeatures $f_1, \ldots, f_k$ that form an OR-group has the following form:

$$p \rightarrow \bigvee_{i=1,\ldots,k} f_i \qquad (1)$$

Similarly, an implication from a parent feature p to its subfeatures $f_1, \ldots, f_k$ that form an XOR-group is defined as

[1] http://gsd.uwaterloo.ca/projects/fsmls/applet-fsml/applet-examples/

[2] Unlike in our previous work [14], we require that the root feature is always present, for consistency with PFMs, for which this choice is natural.



(a) Sample FM

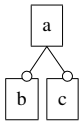
a
$b \rightarrow a$
$c \rightarrow a$

(b) Formulas

a	b	c
1	0	0
1	1	0
1	0	1
1	1	1

(c) Set of legal configurations

Figure 3. Semantics of ordinary FMs



c *given* b [80%]

(a) Sample PFM

a
$b \rightarrow a$
$c \rightarrow a$
$(c|b)[0.8, 0.8]$

(b) Formulas

a	b	c	$P(a,b,c)$
1	1	1	$p_1 \geq 0.0$
1	1	0	$p_2 = 0.25p_1$
1	0	1	$p_3 \geq 0.0$
1	0	0	$p_4 = 1 - 1.25p_1 - p_3$
0	1	1	$p_5 = 0.0$
0	1	0	$p_6 = 0.0$
0	0	1	$p_7 = 0.0$
0	0	0	$p_8 = 0.0$

where $1.25p_1 + p_3 \leq 1$

(c) Legal probability distributions

Figure 4. Semantics of PFMs

follows:

$$p \rightarrow \bigvee_{i=1,\ldots,k} \overline{f_1} \wedge \ldots \wedge \overline{f_{i-1}} \wedge f_i \wedge \overline{f_{i+1}} \wedge \ldots \wedge \overline{f_k} \quad (2)$$

The complete set of formulas for our example is given in Figure 3b. The legal configurations shown in Figure 3c are essentially all truth assignments that satisfy these formulas.

As the semantics of feature models was given by a translation to propositional logics, the semantics of PFMs is defined via translation to *probabilistic* logic [19, 31, 27].

A PFM consisting of a feature diagram and a set of additional hard and soft constraints denote a set of *legal joint probability distributions* (JPDs). A JPD assigns a probability to each *possible* configuration of features. A possible configuration is a subset of a PFM's features represented by a characteristic vector $(f_1, \ldots, f_n) \in \mathcal{F} = \{0, 1\}^n$. The set of all possible configurations, $\mathcal{F}$, represents the *sample space* over which the JPDs are defined. For the PFM in Figure 4a, the set of all legal JPDs is given in Figure 4c.[3] An example of a concrete legal JPD for the PFM is given in Table 1a. For a given JPD, legal configurations have a probability larger than 0 and *illegal configurations* have the probability of 0. A JPD determines a set of *legal samples*. A *sample* is a multiset of configurations with absolute frequencies following the corresponding probabilities specified by the JPD. Tables 1b-c show two samples consistent with the JPD in Table 1a.

Table 1. A distribution and two samples

(a) A legal distribution

a	b	c	$P(a,b,c)$
1	1	1	0.0
1	1	0	0.0
1	0	1	0.8
1	0	0	0.2
0	1	1	0.0
0	1	0	0.0
0	0	1	0.0
0	0	0	0.0

(b) Sample 1

a	b	c
1	0	0
1	0	1
1	0	1
1	0	1
1	0	1

(c) Sample 2

a	b	c
1	0	0
1	0	0
1	0	1
1	0	1
1	0	1
1	0	1
1	0	1
1	0	1
1	0	1
1	0	1

The set of legal JPDs for a PFM is specified using the following probabilistic propositional formulas (PPFs; with ordinary propositional formulas being a special case):

1. Hard constraints implied by the feature diagram; these formulas are determined the same way as for an ordinary feature model (see items (i)–(iv) above).

2. Probabilistic formula $(f_c|f_p)[x/100\%, y/100\%]$ for each child feature f_c displaying probability interval [x%,y%] in its box, as in Figure 2b, where f_p is its parent. Note that we use [x%] as a shorthand for [x%,x%].

3. Any additional soft and hard constraints supplied with the feature diagram. A soft constraint of the form "A *given* B [x%,y%]", where A and B are Boolean expressions over features, is translated to $(A|B)[x/100\%, y/100\%]$.

For example, the set of PPFs for the PFM in Figure 4a is given in Figure 4b. The PPFs for the PFMs in Figures 2a and 2b are given in Figures 5a and 5b, respectively. Note that the translation in Figure 5a assumes the thresholds of .5, .8, and .8, for off-by-default, on-by-default, and encourages relations, respectively.

Technically, a PPF induces a linear constraint on the set of legal JPDs. As a result, the set of legal JPDs of a PFM comprises of solutions for a system of linear inequalities (as in Appendix A). Similarly, reasoning procedures for a set of PPFs such as consitency checking (PSAT [31, 4]) and logical entailment represent a linear programming problem. Although the size of the linear systems grows exponentially with the number of variables if represented explicitly, efficient computation techniques exist, such as *column generation*, which avoid this explosion in practice [9]. An example of a reasoner for probabilistic logic is *nmproblog* [27]. We used this tool to verify the consistency of the formulas in Figure 5a.

4 Underspecification in PFMs

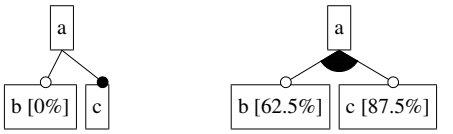
As we have just said, a consistent set of PPFs, and thereby a consistent PFM, is usually an underspecification

[3] The derivation of the JPDs for the example is given in Appendix A.

root: A	**root:** A
child $\rightarrow$ **parent** :	**child** $\rightarrow$ **parent** :
$m \rightarrow a$	$p \rightarrow a$
$d \rightarrow a$	$t \rightarrow a$
$o \rightarrow a$	$i \rightarrow a$
$p \rightarrow m$	$d \rightarrow i$
$t \rightarrow m$	$o \rightarrow i$
$i \rightarrow m$	**OR-group:**
OR-group:	$a \rightarrow p \vee t \vee i$
$m \rightarrow p \vee t \vee i$	**parent** $\rightarrow$ **child:**
parent $\rightarrow$ **child:**	$(p\|a)[0.75, 0.75]$
$a \rightarrow m$	$(t\|a)[0.59, 0.59]$
$(d\|a)[0, 0.5)$	$(i\|a)[0.97, 0.97]$
$(o\|a)[0, 0.5)$	$(d\|i)[0.42, 0.42]$
$(p\|m)[0.8, 1]$	$(o\|i)[0.53, 0.53]$
$(t\|m)[0, 0.5)$	**additional:**
$(i\|m)[0.8, 1]$	$(o\|t)[0.84, 0.84]$
additional:	$(t\|o)[0.97, 0.97]$
$(d\|i)[0.8, 1]$	$(p\|d)[0.88, 0.88]$
$(i\|d)[0.8, 1]$	$(p\|o)[0.88, 0.88]$
$(o\|t)[0.8, 1]$	$(i\|t)[0.97, 0.97]$
$(t\|o)[0.8, 1]$	$(i\|p)[0.98, 0.98]$

(a) Formulas for Figure 2a (b) Formulas for Figure 2b

Figure 5. Mapping PFMs to formulas



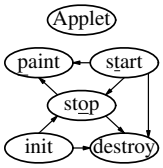
(a) $p_1 = 0,\ p_3 = 1$ (b) $p_1 = 0.5,\ p_2 = 0.375$

Figure 6. PFMs with only a single distribution consistent with Figure 3a

representing more than one probability distribution. For example, the PFMs in Figures 2a, 2b, and 4a are underspecifications of JPDs. This is in stark contrast to propositional logics, and ordinary feature models, where a consistent set of formulas is a *complete and precise* specification of the legal configurations of the "hard" structure.

To further illustrate this point, let us analyze the range of legal JPDs for the PFM in Figure 4. To this end, consider the PFMs in Figure 6. Each of these PFMs happens to represent a single JPD that is also consistent with the PFM in Figure 4a. The corresponding JPD can be obtained by substituting the specified values for p_1 and p_2 in Figure 4c.

In general, it is difficult to assess whether a PFM represents a single JPD or not just by looking at it. A naïve way of deciding this would be to supply the probability of every possible configuration as an additional constraint, exponential in the number of features (as in Figure 4c), and checking whether the solution is unique (practical only for the smallest PFMs).



(a) DAG

t	o	$P(p \mid t, o)$	$P(\bar{p} \mid t, o)$
1	1	0.91	0.09
0	1	0.00	1.00
1	0	0.17	0.83
1	1	0.72	0.28

(b) CPT for paint

Figure 7. Automatically learned BN for Applet

A complete and compact representation of a single JPD can be represented by a Bayesian Network (BN) [32], as shown in Figure 7. A BN consists of a directed acyclic graph (DAG) (Figure 7a) and a *conditional probability table* (CPT) (Figure 7b) for every node. Each node in the DAG represents a variable and the edges represent direct dependencies. The DAG encodes all the dependencies and independencies among the variables. The rules for how dependencies and independencies are inferred from a DAG are somewhat technical, and will not be further explained here; see [32] for details. For example, the DAG in Figure 7a states that Applet is independent of the other variables, which is consistent with the fact that every legal sample has to include it. To give another example, paint is dependent on all the other variables except Applet. Because of space constraints, only the CPT for paint is given (Figure 7b) and the CPTs for stop and destroy are elided. Each of the nodes without ancestors have a *marginal* (i.e., "unconditional") probability attached. For example, the marginal probabilities for Applet and init are 1 and .97, respectively. It is important to note that there are many different DAGs, even with the same underlying undirected graph, that can represent the same JPD [22].

As a BN represents a single JPD, we can use it to infer precise probabilities (as opposed to probability intervals) for events defined as propositional formulas. Well-studied algorithms exist for both inference in BNs and for learning BNs from concrete configuration samples [22]. In fact, the BN shown in Figure 7 was automatically learned[4] from the same sample from which the PFM in Fig. 2b was extracted.

Our analysis of underspecification in PFMs would be incomplete without briefly discussing the connection between PFMs and BNs. On the one hand, BNs offer several advantages. For one, reasoning in BNs, while still computationally hard, is typically more efficient than reasoning in general probabilistic logic. Furthermore, BNs offer a systematic way to elicit all dependencies and independencies among variables and to specify just a single JPD. Also, BNs are supported by several industry-strength tools, which are used in various domains, such as economic modeling and medical diagnosis. On the other hand, PFMs have strengths that BNs lack. Most importantly, a PFM provides a specification of variability that is easier to understand just by

[4] Using Hugin [28], ver. 6.9, with significance factor of 0.4.

looking at the model. As an example, consider the PFM in Figure 2b and the BN in Figure 7, which were extracted from the same configuration sample. In general, the hierarchy that is typically present in modular software (even if the software is not entirely modular) is easily reflected in a feature model. However, a BN does not make this hierarchy readily visible. While a feature model with no additional constraints can be mapped to a hierarchical BN, supplying additional constraints tends to destroy the hierarchy in the corresponding BN. Nevertheless, as will be shown in the following sections, PFMs can be meaningfully combined with BNs for some applications.

The underspecification of PFMs is a very powerful feature which enables abstract modeling. For example, the PFMs in Figures 2a and 2b are both abstract in the sense of denoting sets of legal configurations. The first PFM is even more abstract than the second since it specifies probability intervals for some of the feature dependencies that are given as concrete probability values in the other PFM. Yet, both PFMs are still more abstract than the BN in Figure 7. In a sense, the BN represent a possible "ground model" containing all the details about the JPD it represents.

5 Application-Specific Semantic Refinements

The abstraction afforded by PFMs is particularly useful in model specification and mining. Just as a UML model of a system design does not show all the implementation detail present in the system, a PFM can choose to abstract the details of a particular JPD that it models.

However, the underspecification present in the standard semantics of PFMs may sometimes need to be restricted depending on usage context. For example, a reasonable restriction could be that any hard constraints that the legal JPDs for a PFM satisfy and that can be represented in a feature diagram (e.g, as subfeature relations or feature groups) must be shown in the PFM's feature diagram. Under this restriction, the set of legal JPDs for the PFM in Figure 4a could not include any of the special cases shown in Figure 6. Furthermore, in PFM mining, a particular mining technique might guarantee a complete recovery of certain structures (e.g., the feature diagram) from a given sample. For example, the mining procedure given in the following section guarantees this; however, additional constraints are recovered only if they have a certain form. The mined PFM needs to be interpreted with this restriction in mind.

Semantics restrictions are not uncommon in other modeling languages. For example, just by looking at a UML class diagram that shows some associations but without any multiplicities, it is difficult to say whether the multiplicities are just elided or some default values should be assumed. Each choice represents a refinement of the standard UML class modeling notation, and the particular refinement used in a model needs to be indicated in order to be able to correctly interpret the model.

FEATURE-MODEL($\mathcal{S}$: dataset; $\check{s}, \hat{s}, m : [0,1]$)

1 $\triangleright$ $\check{s}$ is min. support and $\hat{s}$ is max. support;
2 $\triangleright$ m is min. confidence

3 $\triangleright$ Mine strong conjunctive association rules [1]
4 $C \leftarrow$ CONJ-RULE-MINER($\mathbf{e}(\mathcal{S}), \check{s}, m$)

5 $\triangleright$ Mine frequent minimal OR-clauses [40]
6 $O \leftarrow$ BLOSOM-MO($\mathcal{S}, \check{s}, \hat{s}$)

7 $\triangleright$ Select strong disjunctive association rules
8 $D \leftarrow \{r = p \Rightarrow f_1 \vee \ldots \vee f_k \mid$
9 $\quad f_1 \vee \ldots \vee f_k \in O$
10 $\quad \wedge\, \mathbf{c}(f_1 \vee \ldots \vee f_k) \subseteq \mathbf{c}(p)$
11 $\quad \wedge\, \mathrm{conf}(r) \geq m \wedge \mathrm{supp}(r) \geq \check{s}\}$

12 $\triangleright$ Select rules for feature graph generation
13 $H \leftarrow \{r \in C \cup D \mid$
14 $\quad (\ r$ has one of the forms:
15 $\quad f_i \Rightarrow f_j,\ f_i \Rightarrow \bar{f}_j,\ f_i \Rightarrow f_j \vee \ldots \vee f_k\ )$
16 $\quad \wedge\, \mathrm{conf}(r) = 1\}$

17 $\triangleright$ Generate feature graph [14]
18 $G \leftarrow$ FEATURE-GRAPH $\left(\bigwedge_{r \in H} r\right)$

19 $\triangleright$ Collect soft constraints
20 $E \leftarrow C \cup D \setminus H$

21 **return** G, E

Figure 8. Mining of a feature model

Configuration is a special case as it typically requires the most restricted semantics. In configuration, we want to efficiently propagate configuration choices, which is best achieved only if a single JPD is given. Thus, an underspecified PFM needs to "completed" to a concrete JPD for configuration. The minimum requirement for such completion is that it does not introduce new hard constraints. We will give two techniques for achieving this goal in Section 7.

6 Mining (Probabilistic) Feature Models

We present an automatic mining procedure (Figure 8) that retrieves a feature model from a multiset of sample configurations of the kind shown in Tables 1b-c. The procedure first applies existing data mining algorithms [1, 40] in order to detect reoccurring feature patterns and then generates the feature diagram using an algorithm described in [14]. In consistence with data mining literature, we refer to the multiset of configurations as a *dataset*, however we still prefer "configurations" over "transactions"—more typically seen in the data mining literature.

The first part of the procedure involves mining three kinds of implications, needed to construct the feature diagram [14]:

1. *Binary feature implications* $f_i \rightarrow f_j$: These rules are used to construct the feature hierarchy.
2. *Group implications* $f_i \rightarrow f_j \vee \ldots \vee f_k$: These rules are needed to identify OR- and XOR-groups.
3. *Exclusion clauses* $f_i \rightarrow \bar{f}_j$: These rules are needed to identify XOR-groups.

The rule mining is achieved using two algorithms:

1. *Conjunctive association rule mining (line 3)*: Several well-known data-mining algorithms extract *conjunctive association rules*, i.e., rules of the form $A \Rightarrow B$, where A and B are conjunctions of features. The conjunctive rule miner is used to retrieve both binary feature implications and exclusion clauses. To mine for exclusion clauses, the feature space is extended to include representations of unselected features. We define $\mathbf{e}(\mathcal{S})$, which extend each transaction in $\mathcal{S}$ to include the representations of absent features, where $f^{\text{-}}$ is a new feature which represents $\bar{f}$.

$$\mathbf{e}(\mathcal{S}) = \{t \cup \{f^{\text{-}} \mid f \notin t\} \mid t \in \mathcal{S}\} \tag{3}$$

   The mining of such rules involves finding appropriate AND-clauses, known as *frequent itemsets* [18]. In our tests, we used LCM [37] to find frequent itemsets and generated association rules using the procedure described by Agrawal et al. [1].

2. *Disjunctive association rule mining (lines 5–11)*: Mining disjunctive rules has not been as extensively studied as conjunctive rule mining. Simple *disjunctive association rules* have the form $a \Rightarrow B$, where a is a single feature and B is a disjunction. These rules are constructed from so-called *minimum* OR*-clauses* (lines 5–6) which are computed using an algorithm by Zhao et al. [40]. The $\mathbf{c}(B)$ operator used in the algorithm returns the configurations that satisfy the expression B.

The mining algorithms use two standard relevance measures for the mined rules: *support* and *confidence* [18]. The *support of a Boolean expression*, e.g., an OR-clause, in a dataset is defined as the relative frequency of the expression being *true* for the configurations in the dataset $\mathcal{S}$:

$$\text{supp}(B) = \frac{|\mathbf{c}(B)|}{|\mathcal{S}|} = P(B) \tag{4}$$

In the above definition, $|\mathbf{c}(B)|$ is the number of configurations that satisfy B and $|\mathcal{S}|$ is the dataset size.

The *rule support* for (conjunctive or disjunctive) association rules is defined as follows:

$$\text{supp}(A \Rightarrow B) = \text{supp}(A \wedge B) = P(A \wedge B) \tag{5}$$

The *confidence* measure for association rules is defined as follows:

$$\text{conf}(A \Rightarrow B) = \frac{\text{supp}(A \wedge B)}{\text{supp}(A)} = P(B|A) \tag{6}$$

Thus, expression support and rule confidence provide a clear connection between the *data mining* and the *probabilistic* interpretation of expressions and rules with respect to $\mathcal{S}$ (cf. Section 3): $\text{supp}(B) = P(B)$ and $\text{conf}(A \Rightarrow B) = P(B|A)$. An association rule $A \Rightarrow B$ with confidence $c\%$ is equivalent to the PPF $(B|A)[c, c]$, and consequently the soft constraint B given A $[c\%]$. A rule with 100% confidence is equivalent to the implication, $A \rightarrow B$. The rule mining algorithms used in Figure 8 return the confidence and support for mined rules.

Note that the support of a rule is a measure of the statistical significance of the rule in $\mathcal{S}$. For example, the rule $A \Rightarrow B$ may almost trivially achieve the confidence of 1 if $\text{supp}(B) \approx 1$ and $\text{supp}(A)$ is some very small number, e.g., $.0001$ in $\mathcal{S}$. In other words, the satisfaction of the rule by most of the configurations in $\mathcal{S}$, as reflected by the high confidence, stems mostly from the fact that $\mathcal{S}$ exhibits only very few cases of A. However, the small number of cases is reflected by the low rule support of $.0001$. The generation of such weakly supported rules can be avoided by requiring some sufficiently minimal support. Rules that satisfy both minimal support and confidence thresholds are referred to as being *strong*.

The final step of the procedure involves the generation of the feature diagram (line 17). This step uses the algorithm described in [14]. The input to that algorithm is a formula obtained by conjuncting the mined binary feature implications, group implications, and exclusion clauses with confidence of 1 (lines 12–16). The remaining rules are collected as soft constraints (line 19). For simplicity, we do not recover all hard additional constraints. This step can be achieved by computing the disjunction of all transactions and conjoining it with the negation of the formula corresponding to the feature diagram.

The computational cost of the procedure consists of the cost of rule mining and the cost of feature diagram generation. Conjunctive association rule mining has been extensively studied and a wealth of efficient algorithms exist that were shown to scale to large problems (e.g., rule mining in market basket analysis or genomics) [18]. The minimum OR-clause miner [40] is exponential in the number of features; however, several search space pruning techniques used in the algorithm enabled scaling it to practical problems. For example, mining minimum OR-clauses from 300 configurations with 50 features takes less than 20 seconds on a 1.4GHz Pentium-M with 448MB running Windows XP through VMware [40]. We have argued in [14] that the cost of feature diagram construction is polynomial in the size of the BDD representing the input propositional formula. The

BDD itself can be exponential in the number of features, but rarely is for typical feature models.

One can consider useful extensions to our basic miner:

- *Rule pruning*: Several rules generated by the basic miners may be redundant. For example, the rules "init *given* start" and "init *given* paint" in Figure 2b add little new information to the already shown $P(\text{init} \mid \text{Applet}) = 0.97$. Standard techniques for rule pruning are available [29, 8].
- *Using a priori structures*: Some structures may be already known from the problem domain and do not need to be mined. For example, it is clear that method overrides and method calls imply their containing classes and there is no need to rediscover such structures through data mining. The standard mining algorithms can be optimized to take such a priori constraints into account through *constraint-based association mining* [18]. Finally, some structures may not be present in the mined dataset, but they may be known to the domain expert. Such structures can be simply added as constraints to $C \cup D$. This kind of model refinement may need dedicated tool support in order to discover inconsistencies and propose fixes.
- *Treating bias and outliers*: The dataset might not be representative of a population, e.g., legal configurations might be absent from the sample, or it may contain errors or outliers. In particular, rules with confidence 1 need to be reviewed by an expert to determine whether they should be confirmed as hard rules or their confidence needs to be lowered because of bias in the dataset. Similarly, strong rules with confidence lower than 1 should be reviewed whether they should be confirmed as soft rules or need to be "hardened" because of outliers or errors in the dataset. For example, during mining of the Applet model in Figure 2b, the initial sample of applets contained two applets that did not override any of init, start, or paint. As a result, the corresponding OR-group was lost from the feature diagram and the simple disjunctive rule for the three methods only appeared in the soft constraints. Upon closer inspection, we determined that these two applets were outliers; they were examples demonstrating the implementation of applet constructors rather than complete applets. Following that discovery, the two applets were removed from the sample and the model was regenerated.

7 Configuration Under Soft Constraints

As discussed in Section 4, PFMs in general, do not uniquely determine a probability distribution on configurations. Instead, a range of legal JPDs is possible. However, given a concrete legal JPD, a PFM can be used to guide users in their selections of desired configurations. We shall now discuss the advantages of such an approach in contrast to discrete interactive configuration, and mention briefly the algorithms that would support it.

First, introduction of probabilities into the modeling language has allowed us to configure soft preferences, among many combinatorically related selections of features. Second, these preferences can change dynamically throughout the configuration process in a *non-monotonic* way. Recall the car example. The manual gear is a preferred feature initially, as nothing is known about the variant of the car. However, the order of preference (probabilities) will be *reversed*, from manual to automatic, as soon as the North American variant is selected. A user of such a configuration system will experience a more *adaptive* form of guidance than just being presented with valid choices, as the guiding engine will not only propagate hard constraints *about the product*, but also reason about the *user's* preferences.

In order to provide this functionality one needs address two problems: (1) obtain a unique and useful probability distribution consistent with a PFM; (2) provide interactive configuration algorithms that can compute posterior marginal probabilities for feature variables and groups, given the current state of the system (i.e., the features that were selected or eliminated so far), and that can support model auto-completion with a most-likely configuration. Two methods relying on existing technology meet these requirements and will be discussed next.

Configuration with a BN. As mentioned, if the model is mined from sample data, one could use the same sample to automatically learn a BN reflecting the JPD of the configurations. There exist numerous algorithms for learning BNs from a sample [22]. A BN constructed in this way would be automatically consistent with its PFM, however, this method requires an unbiased sample that is representative of the entire configuration space. Once the JPD is represented as a BN, one can use inference and evidence propagation algorithms to compute marginal probabilities for features in a given system state. Also, a *most probable explanation* algorithm [22, 15, 20, 32] can be applied to find a maximum probability assignment to all unobserved variables (i.e., undecided features), given the observed state of the system.

Maximum Entropy Configuration. *Entropy* is a measure of uncertainty used in information theory. A high entropy means a lack of information or low predictability, while low entropy means the presence of precise information or high predictability. There exist methods [27, 21, 19, 31] for finding the JPD satisfying an underspecified system of constraints that maximizes entropy. Searching for a maximum entropy JPD generalizes the principle of indifference; entropy maximization distributes the prob-

abilities among each possible configuration as uniformly as possible, such that the distribution still satisfies all the constraints imposed by the PPFs of a PFM. Entropy maximization is guaranteed not to introduce new hard constraints that are not already logically entailed by the hard PPFs of the PFM. In fact, the JPD inferred by entropy maximization from a set of PPFs is the only "honest" conclusion from these PPFs, i.e., one that does not take any other evidence into account [21]. Entropy maximization can be seen as a specialized reasoning method for probabilistic logic [27].

Even though we can reduce the configuration algorithms on PFMs to existing solutions, more practical evaluation is required in future. In particular, we would like to implement prototype configurators and investigate solutions that improve the running times of the above algorithms, such as direct encodings of hard constraints in Bayesian networks [38], compilation based inference in BNs [10, 11], and column generation methods [3].

8 Related Work

Throughout the paper, we have extensively referred to the body of foundational work that has been used to building the PFM infrastructure. Here we should briefly mention related contributions that have not been referred so far.

Encourages and discourages constraints have been suggested as an extension to FM several times, for example in [12, 39]. However, no precise semantics have been given to these extensions in the previous work. The work that comes closest to PFMs is Robak's extension of feature models with fuzzy logic [33]. Fuzzy logic is an alternative way of reasoning under uncertainty, different from probabilistic logic. The advantage of probabilistic logics are that it gives in easily to both interpretations as degree of belief and as relative frequencies, which allows us to use the same modeling language for specification, mining and configuration. Related to PFMs are also *i* goal models*, which can be thought of as AND-OR trees augmented with encourages and discourages relationships [16]. Such models have been used in the configuration of software [25].

Reverse engineering of variability models was addressed before. Loesch [26] has discussed a variability extraction and visualization method based on concept analysis. He does not directly use feature models. Interestingly, concept analysis corresponds to mining closed AND-clauses [40]. Our work also considers OR- and XOR-groups and soft constraints. Jepsen [23] and colleagues describe a manual process of identifying code differences between variants and structuring them in a feature model. Our ultimate aim is to automate these kind of procedures. The present paper is just a step towards this goal.

9 Conclusion

We have presented probabilistic feature models, which are basic feature models extended with soft constraints. We have argued for the need of soft constraints in modeling, reverse-engineering and configuration applications. We then presented a mining procedure that retrieves PFMs from representative samples of configurations and discussed the algorithms and data structures that can be used to perform interactive configuration with soft constraints. PFMs have been illustrated with examples from automotive and software engineering domains.

To the best of our knowledge, PFMs are a substantial extension of feature model that will enable a new generation of tools for feature modeling, reverse engineering and configuration. We intend to address several challenges posed by such tools in the near future.

Acknowledgements. Authors would like to thank Michał Antkiewicz for creating the feature model in Figure 2a and the anonymous reviewers for their valuable feedback.

A Derivation of JPDs denoted by a PFM

The sample probabilistic feature model (PFM) in Figure 4a denotes a set of joint probability distributions (JPDs), which are defined in a closed form in Figure 4c. This definition can be derived as follows.

First, $p_1, \dots p_8$, Figure 4c, need to satisfy the axioms:

$$0 \leq p_i \leq 1 \text{ for } i = 1 \dots 8 \tag{7}$$

$$\sum_{i=1\dots8} p_i = 1 \tag{8}$$

Furthermore, each formula in Figure 4b places a constraint on some of the probabilities p_i. In our case, we only have to consider a and $(c|b)[0.8, 0.8]$ since $b \rightarrow a$ and $c \rightarrow a$ are logically entailed by the first formula. The probabilistic meaning of the propositional formula a is

$$P(a) = 1 \tag{9}$$

By decomposing a into atomic events, we obtain

$$\begin{aligned} P(a) &= P(abc \vee ab\bar{c} \vee a\bar{b}c \vee a\bar{b}\bar{c}) \\ &= P(abc) + P(ab\bar{c}) + PP(a\bar{b}c) + P(a\bar{b}\bar{c}) \\ &= p_1 + p_2 + p_3 + p_4 \end{aligned} \tag{10}$$

From 9 and 10 we obtain

$$p_1 + p_2 + p_3 + p_4 = 1 \tag{11}$$

From 8 and 11 immediately follows:

$$p_i = 0 \text{ for } i = 5 \dots 8 \tag{12}$$

The meaning of the probabilistic formula $(c|b)[0.8, 0.8]$ is

$$0.8 \leq P(c|b) \leq 0.8 \tag{13}$$

By definition of conditional probability and 13 we have $P(cb) = 0.8P(b)$ which, by decomposing cb and b, gives

$$p_1 + p_5 = 0.8(p_1 + p_2 + p_5 + p_6) \tag{14}$$

From 14 and 12 we obtain

$$p_2 = 0.25p_1 \tag{15}$$

From 11 and 15 we obtain $p_4 = 1 - 1.25p_1 - p_3$. Finally, since $p_4 \geq 0$ (from 7), we also have $1.25p_1 + p_3 \leq 1$.

References

[1] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD '93*. ACM, 1993.

[2] V. Alves, R. Gheyi, T. Massoni, U. Kulesza, P. Borba, and C. Lucena. Refactoring product lines. In *GPCE'06*. ACM, 2006.

[3] K. A. Anderson and J. N. Hooker. A linear programming framework for logics of uncertainty. *Decis. Support Syst.*, 1996.

[4] P. S. S. Andrade, J. Rocha, D. P. Couto, A. da Costa Teves, and F. Cozmans. A toolset for propositional probabilistic logic. In *BCSC '07*. BCS, 2007.

[5] M. Antkiewicz and K. Czarnecki. Framework-specific modeling languages with round-trip engineering. In *MoDELS '06*. Springer, 2006.

[6] D. S. Batory. Feature models, grammars, and propositional formulas. In *SPLC '05*. Springer, 2005.

[7] D. S. Batory, D. Benavides, and A. R. Cortés. Automated analysis of feature models: challenges ahead. *Commun. ACM*, 2006.

[8] M. Bruch, T. Schäfer, and M. Mezini. FrUiT: IDE support for framework understanding. In *eclipse '06*. ACM, 2006.

[9] V. Chandru and J. N. Hooker. *Optimization Methods for Logical Inference*. Wiley-Interscience, 1999.

[10] M. Chavira and A. Darwiche. Compiling Bayesian networks using variable elimination. In *IJCAI '07*, 2007.

[11] M. Chavira, A. Darwiche, and M. Jaeger. Compiling relational Bayesian networks for exact inference. *Int. J. Approx. Reasoning*, 2006.

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

[13] K. Czarnecki and C. H. P. Kim. Cardinality-based feature modeling and constraints: a progress report. In *International Workshop on Software Factories*, 2005.

[14] K. Czarnecki and A. Wąsowski. Feature models and logics: There and back again. In *SPLC '07*. IEEE, 2007.

[15] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[16] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani. Reasoning with goal models. In *ER'02*, 2002.

[17] T. Hadzic, S. Subbarayan, R. M. Jensen, H. R. Andersen, J. Møller, and H. Hulgaard. Fast backtrack-free product configuration using a precompiled solution space representation. In *PETO '04*. DTU-tryk, 2004.

[18] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000.

[19] J. N. Hooker. Mathematical programming models for reasoning under uncertainty (plenary address). In *Operations Research Proceedings '91*. Springer-Verlag, 1992.

[20] C. Huang and A. Darwiche. Inference in belief networks: A procedural guide. *Int. J. Approx. Reasoning*, 1996.

[21] E. T. Jaynes. *Probability Theory: The Logic of Science*. Cambridge University, 2003.

[22] F. V. Jensen and T. D. Nielsen. *Bayesian Networks and Decision Graphs*. Springer, 2007.

[23] H. P. Jepsen, J. G. Dall, and D. Beuche. Minimally invasive migration to software product lines. In *SPLC '07*. IEEE Computer Society, 2007.

[24] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, 1990.

[25] S. Liaskos, A. Lapouchnian, Y. Wang, Y. Yu, and S. Easterbrook. Configuring common personal software: a requirements-driven approach. In *RE'05*, 2005.

[26] F. Loesch and E. Ploedereder. Optimization of variability in software product lines. In *SPLC '07*. IEEE Computer Society, 2007.

[27] T. Lukasiewicz and G. Kern-Isberner. Probabilistic logic programming under maximum entropy. *Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, 1999.

[28] A. L. Madsen, F. Jensen, U. Kjærulff, and M. Lang. The Hugin tool for probabilistic graphical models. *International Journal on Artificial Intelligence Tools*, 2005.

[29] A. Michail. Data mining library reuse patterns using generalized association rules. In *ICSE '00*. ACM, 2000.

[30] J. Møller, H. R. Andersen, and H. Hulgaard. Product configuration over the Internet. In *INFORMS '01*, 2001.

[31] N. Nilsson. Probabilistic logic. *Artificial Intelligence*, 1986.

[32] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.

[33] S. Robak and A. Pieczyński. Employment of fuzzy logic in feature diagrams to model variability in software families. *J. Integr. Des. Process Sci.*, 2003.

[34] P.-Y. Schobbens, P. Heymans, and J.-C. Trigaux. Feature diagrams: A survey and a formal semantics. In *RE'06*, 2006.

[35] Sun Microsystems. Java tutorials, lesson: Applets. http://java.sun.com/docs/books/tutorial/deployment/applet/.

[36] Sun Microsystems. Taking advantage of the Applet API. Available from http://java.sun.com/docs/books/tutorial/deployment/applet/appletsonlyindex.html.

[37] T. Uno, M. Kiyomi, and H. Arimura. Lcm ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *FIMI*, 2004.

[38] M. Valtorta, J. Byrnes, and M. Huhns. Logical and probabilistic reasoning to support information analysis in uncertain domains. In *Progic '07*, 2007. To appear.

[39] H. Wada, J. Suzuki, and K. Oba. A feature modeling support for non-functional constraints in service oriented architecture. In *IEEE Computer Society*, 2007.

[40] L. Zhao, M. J. Zaki, and N. Ramakrishnan. BLOSOM: A framework for mining arbitrary boolean expressions over attribute sets. Technical Report 06-05, 2006. Available from http://www.cs.rpi.edu/research/pdf/06-05.pdf.