Towards Anomaly Explanation in Feature Models *

A. Felfernig[1], D. Benavides[2], J. Galindo[2], and F. Reinfrank[1]
[1]Graz University of Technology, Graz, Austria
{alexander.felfernig, florian.reinfrank}@ist.tugraz.at
[2]University of Seville, Spain
{benavides,jagalindo}@us.es

Abstract

Feature models are a wide-spread approach to variability and commonality management in software product lines. Due to the increasing size and complexity of feature models, anomalies in terms of inconsistencies and redundancies can occur which lead to increased efforts related to feature model development and maintenance. In this paper we introduce knowledge representations which serve as a basis for the explanation of anomalies in feature models. On the basis of these representations we show how explanation algorithms can be applied. The results of a performance analysis show the applicability of these algorithms for anomaly detection in feature models. We conclude the paper with a discussion of future research issues.

1 Introduction

Similar to component-oriented configuration models [Felfernig *et al.*, 2000; Felfernig, 2007], Feature Models (FM) [Kang *et al.*, 1990] are used to express variability properties of highly-variant items [Mendonca and Cowan, 2010]. Applications based on feature models help users to decide about relevant features and to learn about existing dependencies between features. Feature models can be distinguished with regard to the expressiveness of constraints defining the relationships between the different features contained in a feature model [Benavides *et al.*, 2010]. So-called *basic feature models* [Kang *et al.*, 1990] will be used as a basis for the discussions in this paper. Such models allow the definition of basic relationships between features, for example, a feature f_1 requires the inclusion of a feature f_2. *Cardinality-based feature models* [Czarnecki *et al.*, 2005] extend basic ones by also allowing cardinalities with an upper bound > 1. Finally, *extended feature models* [Batory, 2005] allow the inclusion of additional information about features in terms of feature attributes. For presentation purposes we decided to use basic feature models (see Section 2). However, the presented concepts and algorithms can be applied to advanced feature model representations as well.

Developing and maintaining large and potentially complex feature models is an error-prone activity which can be explained by the cognitive overload of software engineers and domain experts [Trinidad *et al.*, 2008; Benavides *et al.*, 2013]. In order to tackle this challenge, feature model development and maintenance processes have to be supported by intelligent techniques and tools which help to identify anomalies which become manifest in different types of inconsistencies and redundancies [Batory *et al.*, 2006; Benavides *et al.*, 2010]. An approach to the identification of *dead features* (features not part of any configuration) is presented by Trinidad et al. [Trinidad *et al.*, 2008]. The authors also introduce concepts to solve the problem of *void feature models* (no configuration exists that fulfills all the constraints in the feature model). For the identification of faulty relationships in the feature model (in these scenarios) the authors define a corresponding *diagnosis task* which is based on the concepts introduced by [Reiter, 1987]. As an alternative to the approach of [Trinidad *et al.*, 2008], White et al. [White *et al.*, 2010] show how to transform feature models into a corresponding representation of a constraint satisfaction problem (CSP) [Tsang, 1993]. On the basis of this representation, diagnoses are directly determined by the constraint solver without the support of an additional diagnostic engine. An overview of *analysis operations* (for the identification of different inconsistencies and redundancies) for feature models is provided in [Benavides *et al.*, 2010; von der Massen and Lichter, 2004].

If we are interested in *minimal* explanations (diagnoses) for feature model anomalies, the performance of the underlying algorithms becomes a challenge. An example explanation in this context would be the minimal set of constraints which have to be adapted or deleted from an inconsistent feature model (the determination of a configuration is not possible) such that the remaining constraints allow the calculation of at least one configuration. Reiter [Reiter, 1987] introduced a hitting set based approach to the determination of minimal explanations (diagnoses) – these diagnoses are also of minimal cardinality since diagnosis search is performed on the basis of breadth-first search. The idea of applying the concepts of model-based diagnosis to inconsistent constraint sets has first been introduced by Bakker et al. [Bakker *et al.*, 1993].

*This work was supported, in part, by the Austrian Research Promotion Agency under the project ICONE (827587), the European Commission (FEDER), the Spanish Government under project SETI (TIN2009-07366), and by the Andalusian Government under project THEOS (TIC-5906).

Michel Aldanondo and Andreas Falkner, Editors
Proceedings of the 15th International Configuration Workshop
August 29-30, 2013, Vienna, Austria

Felfernig et al. [Felfernig *et al.*, 2004] continued this work by introducing an approach to the automated testing and debugging of configuration knowledge bases where test cases are used to induce conflicts in the knowledge base. These conflicts are then resolved on the basis of the hitting set algorithm [Reiter, 1987]. First experiences from the application of these testing and debugging approaches in industrial scenarios are reported by Fleischanderl [Fleischanderl, 2002]. Junker [Junker, 2004] introduced the QuickXPlain (QX) algorithm. QX is an efficient divide-and-conquer based approach to the determination of minimal conflicts which can then be exploited for the determination of diagnoses. In this paper we show how diagnosis and redundancy detection algorithms can be applied to support feature model analysis operations [Benavides *et al.*, 2010]. In this context we show how to apply the diagnosis algorithm FASTDIAG [Felfernig *et al.*, 2012] (an algorithm with no need of determining conflict sets) and introduce the FMCORE algorithm which allows the detection of redundancies in feature models.

The work presented here is in the line of research dedicated to the development of intelligent quality assurance mechanisms for configuration knowledge bases [Felfernig *et al.*, 2004]. The major contributions of this paper are the following. First, we advance the state of the art in feature model anomaly detection by formalizing the anomaly types discussed in the feature modeling community on the basis of the concepts of inconsistency and redundancy. Second, we introduce the FMCORE algorithm for the detection of redundant constraints in feature models. Furthermore, we show how to apply the FASTDIAG algorithm [Felfernig *et al.*, 2012] for *explaining* different types of inconsistencies in feature models. All anomaly types will be discussed in detail in Section 3 in combination with corresponding explanation approaches.

The remainder of this paper is organized as follows. In Section 2 we introduce a simple feature model (operating system configuration) which will be used as working example throughout the paper. Furthermore, we introduce the definitions of a *feature model configuration task* and a corresponding *feature model configuration*. In Section 3 we introduce different relevant forms of anomalies in feature models together with their formal definitions. The corresponding anomaly detection algorithms FASTDIAG and FMCORE are explained in Section 4. The performance of these algorithms is analyzed in Section 5 on the basis of selected feature models from the *S.P.L.O.T.*[1] repository. A discussion of further research issues and a conclusion is provided in Section 6.

2 Feature models

A feature model (FM) defines a set of possible products of a domain in terms of features and the relationships between them [Wang *et al.*, 2010]. Features are arranged hierarchically (tree structure with one so-called *root feature* f_r *(*$f_r = true$*)*) [Benavides *et al.*, 2010] where the nodes are the features and the edges are relationships (constraints) [Segura *et al.*, 2010]. For a more detailed overview of different feature model representations we refer the reader to [Batory, 2005; Benavides *et al.*, 2010].

[1] See www.splot-research.org.

Semantics of Feature Models. Our representation of FMs is based on the notation introduced in [Benavides *et al.*, 2010]. Relationships (constraints) in FMs are represented in terms of six different types of constraints [Batory, 2005; Benavides *et al.*, 2010; Segura *et al.*, 2010]: *mandatory*, *optional*, *alternative*, *or*, *requires*, and *excludes*. FMs are representing configurable products which can be formalized in the form of a constraint satisfaction problem (CSP) [Tsang, 1993] where each variable f_i has the assigned domain $d_i = \{true, false\}$. We define a *feature model configuration task* as follows (see Definition 1).

Definition 1 (FM Configuration Task). A feature model (FM) configuration task is defined by the triple (F,D,C) where $F = \{f_1, f_2, ..., f_n\}$ is a set of features f_i, $D = \{dom(f_1), dom(f_2), ..., dom(f_n)\}$ ($dom(f_i) = \{true, false\}$) is the set of corresponding feature domains, and $C = CR \cup CF$ is a set of constraints restricting the possible configurations which can be derived from the feature model. In this context, $CR = \{c_1, c_2, ..., c_k\}$ represents a set of requirements (of a specific user) and $CF = \{c_{k+1}, c_{k+2}, ..., c_m\}$ a set of feature model constraints.

On the basis of this definition of an feature model configuration task, we now introduce the definition of a *configuration* for a feature model (FM) configuration task (Definition 2).

Definition 2 (FM Configuration). A feature model (FM) configuration for a given FM configuration task is a *complete* assignment of the variables $f_i \in F$. Such a configuration is *consistent* iff the constraints $c_i \in C$ are not contradicting with the variable assignment. Furthermore, an FM configuration is *valid*, if it is consistent and complete.

Feature Model Constraint Types. Six basic types of constraints can be included in CF [Benavides *et al.*, 2010]. These constraint types are the following – their representation in a graphical feature model is shown in the example of Figure 1. In the following we introduce the semantics of these six types of constraints – this semantics is based on the definition given in [Benavides *et al.*, 2010].

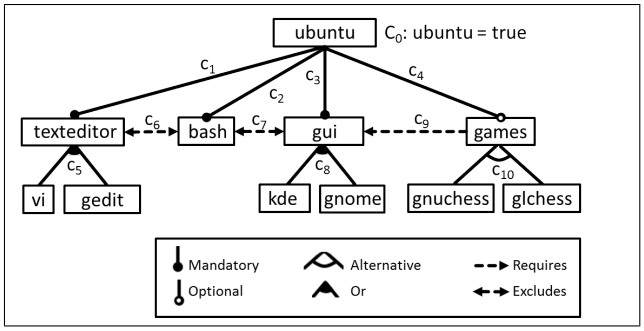

Figure 1: Feature model (FM) with faulty model elements.

Mandatory: a feature $f_2 \in F$ is *mandatory* if it is in a mandatory relationship with another feature $f_1 \in F$. This means, if f_1 is part of the configuration, f_2 must be part of the configuration as well (and vice-versa). The formalization of this constraint type (relationship) is realized on the basis of an equivalence: $f_1 \leftrightarrow f_2$. In Figure 1 the feature *gui* is a mandatory feature connected to the feature *ubuntu*.

Optional: a feature $f_2 \in F$ can (but must not) be included in the configuration in the case that feature $f_1 \in F$ is part of the configuration. This type of constraint can be formalized on the basis of an implication: $f_2 \rightarrow f_1$. In Figure 1 the feature *games* is an optional feature connected to *ubuntu*.

Alternative: only one feature $f_b \in F = \{f_1, f_2, ..., f_k\}$ can be selected if feature f_a is selected. The property can be formalized as follows: $f_1 = true \leftrightarrow (f_2 = false \wedge ... \wedge f_k = false \wedge f_a = true) \wedge ... \wedge f_k = true \leftrightarrow (f_1 = false \wedge ... \wedge f_k - 1 = false \wedge f_a = true)$. In Figure 1 an example of a feature f_a is *games*, the subfeatures are *gnuchess* and *glchess*.

Or: at least one feature $f_b \in F = \{f_1, f_2, ..., f_k\}$ must be part of the configuration if feature f_a is part of the configuration. This property can be formally defined with $f_a \leftrightarrow \{f_1, f_2, ..., f_k\}$. In Figure 1 an example of a feature f_a is *gui*, the subfeatures are *kde* and *gnome*.

Requires: a feature f_2 must be included in a configuration if feature f_1 is included. This requires relationship can be defined with $f_1 \rightarrow f_2$. In Figure 1 an example of a *requires* relationship is *games* $\rightarrow$ *gui*.

Excludes: it is not allowed to combine two features f_1 and f_2 in the same configuration, i.e., feature f_1 excludes feature f_2 and vice versa: $\neg(f_1 \wedge f_2)$. In Figure 1 an example of an *excludes* relationship is $\neg(bash \wedge gui)$. Note that this is a possible faulty constraint to be detected by diagnosis.

Requires and *excludes* constraints are also denoted as *cross-tree constraints*. Finally, the set CR (customer requirements) is an additional set of constraints to be taken into account when determining configurations (solutions). The set CR specifies a set of key features which have to be included in the FM configuration for a specific user (customer).

Example Feature Model. A simple example feature model (from the domain of *operating systems*) is depicted in Figure 1. This model specifies a set of features relevant for configuring an *ubuntu* operating system installation together with constraints between the features. Note that *faulty elements* (constraints) are contained in this model – our goal in the remainder of this paper will be to introduce algorithms which help to identify and explain such faulty constraints.

The CSP-based representation [Tsang, 1993] of the feature model shown in Figure 1 is the following - a representation as FM configuration task = (F,D,C=$CR \cup CF$).

- $F = \{ubuntu, texteditor, bash, gui, games, gedit, vi, kde, gnome, gnuchess, glchess\}$
- $D = \{dom(ubuntu) = \{true, false\}, dom(texteditor) = \{true, false\}, dom(bash) = \{true, false\}, dom(gui) = \{true, false\}, dom(games) = \{true, false\}, dom(gedit) = \{true, false\}, dom(vi) = \{true, false\}, dom(kde) = \{true, false\}, dom(gnome) = \{true, false\}, dom(gnuchess) = \{true, false\}, dom(glchess) = \{true, false\}$
- $CR = \{c_0: ubuntu = true\}$
- $CF = \{ c_1 : ubuntu \leftrightarrow texteditor, c_2 : ubuntu \leftrightarrow bash, c_3: ubuntu \leftrightarrow gui, c_4: games \rightarrow ubuntu, c_5: texteditor \leftrightarrow gedit \vee vi, c_6: \neg texteditor \vee \neg bash, c_7: \neg bash \vee \neg gui, c_8: gui \leftrightarrow kde \vee gnome, c_9: games \rightarrow gui, c_{10}: (gnuchess \leftrightarrow \neg glchess \wedge games) \wedge (glchess \leftrightarrow \neg gnuchess \wedge games)\}$

3 Anomaly Patterns in Feature Models

Anomalies can be defined as patterns in data that do not conform to a well defined notion of normal behavior [Chandola *et al.*, 2009]. Trinidad et al. [Trinidad *et al.*, 2008] are using the term *error* for *incorrect definitions of relationships*, i.e., *the set of products described by a feature model does not match the SPL (software product line) it describes*. We interpret *anomalies* in the sense of [Trinidad *et al.*, 2008]: undesirable FM properties in terms of different facets of contradictory and redundant information contained in the FM.

Handling Inconsistencies. Inconsistent feature models include contradictory constraints $c_i \in C$ that can not be satisfied at the same time, leading to no valid instances derivable from FMs [Wang *et al.*, 2010]. For a given FM configuration task this means that no solution can be identified. In our working example (the FM of Figure 1) no solution can be identified due to an inconsistent constraint set C=$\{c_1, c_2, ..., c_{10}\}$.[2] Inconsistent sets of constraints can be defined on the basis of the concept of conflict sets [Junker, 2004] (see Definition 3).

Definition 3 (Conflict Set) A conflict set $CS \subseteq C$ is a set of constraints s.t. CS is inconsistent. CS is minimal iff there does not exist a conflict set CS' with ($CS' \subset CS$).

Based on Definition 3, we can identify minimal sets of constraints $CS_i \subseteq C$, such that CS_i is inconsistent. As long as there are conflicts in a given constraint set of a feature model, no solutions for the underlying FM configuration task can be identified. Our example feature model (see Figure 1) includes two minimal conflict sets which are $CS_1 = \{c_1, c_2, c_6\}$ and $CS_2 = \{c_2, c_3, c_7\}$. Each of these sets is a minimal set such that (1) no solution (configuration) can be identified and (2) none of the subsets of CS_i is inconsistent. As a consequence (due to their minimality property) conflicts (represented by conflict sets) can be resolved by simply deleting one constraint from the set.

The resolution of all conflicts (represented by conflict sets) can be based on the determination of the corresponding hitting sets (also denoted as diagnoses [Reiter, 1987]). The problem of identifying minimal sets of constraints which have to be adapted or deleted from the feature model such that the remaining constraints become consistent can be represented as an FM diagnosis task (see Definition 4).

Definition 4 (FM Diagnosis Task) A feature model diagnosis task (FM diagnosis task) is a tuple (S, AC) where $S \subseteq AC$ are constraints of the feature model. The task is to identify a minimal set of constraints which have to be deleted from S s.t. consistency can be restored in the feature model.

In this context, S helps us to focus our diagnostic activities, i.e., to focus on those model parts where we suspect faulty constraints. If no such suspects exist, S can be set to AC. An FM diagnosis, i.e., a solution to an FM diagnosis task can be defined as follows (see Definition 5).

Definition 5 (FM Diagnosis) A feature model diagnosis (FM diagnosis) is a set of constraints $\Delta \subseteq S$ with $AC - \Delta$ is consistent. Δ is minimal iff there does not exist a set Δ' with $\Delta' \subset \Delta$ and Δ' has the diagnosis property as well.

[2] Note that we interpret the constraint $c_0 : ubuntu = true$ as element of the (customer) requirements CR.

The diagnoses for our example FM diagnosis task are $\Delta_1 = \{c_1, c_3\}$, $\Delta_2 = \{c_1, c_7\}$, $\Delta_3 = \{c_2\}$, $\Delta_4 = \{c_3, c_6\}$, $\Delta_5 = \{c_6, c_7\}$. These represent five ways to delete (adapt) constraints from (in) the feature model such that at least one configuration can be determined. The calculation of all Δ_i is sketched in Figure 2. The underlying assumption in this example is that – conform to the algorithm introduced by Reiter [Reiter, 1987] – the search tree (hitting set directed acyclic graph – HSDAG) is expanded in breadth-first manner.

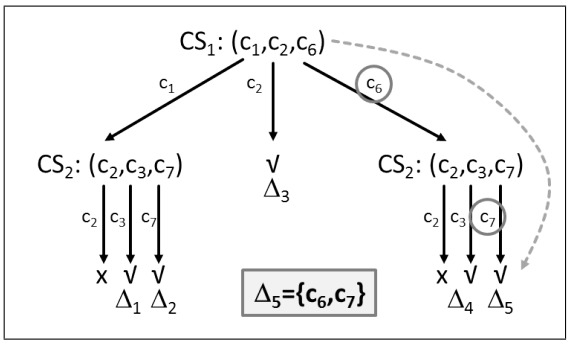

Figure 2: Determination of diagnoses for a given inconsistent feature model (FM). The following discussion of anomaly types assumes that $\Delta_5 = \{c_6, c_7\}$ was chosen.

One possible approach to determine the complete set of diagnoses is based on the *hitting set directed acyclic graph* (HSDAG) algorithm introduced by Reiter [Reiter, 1987]. The basic idea of this algorithm is to determine a conflict (in the example $CS_1 : \{c_1, c_2, c_6\}$) and then to resolve this conflict. If this conflict is resolved (e.g., by deleting the constraint c_1) the algorithm checks whether further conflicts exist in the feature model. In our example this is the case and the next determined conflict set is $CS_2 : \{c_2, c_3, c_7\}$. If we delete, for example c_3 from CS_2, we receive the diagnosis $\Delta_1 = \{c_1, c_3\}$. In a similar fashion all other diagnoses can be determined. Note that $\{c_1, c_2\}$ is not a (minimal) diagnosis since $\{c_2\}$ is already a diagnosis. The HSDAG algorithm is a traditional way of determining diagnoses – more efficient approaches will be presented in Section 4.

Feature Model Anomaly Patterns. We can now discuss in more detail different basic types of feature model *anomalies*. Ways to explain these anomalies and related algorithms will then be discussed in detail in Section 4. An overview of these anomalies and related property checks is shown in Table 1. The following types of anomalies are taken from Benavides et al. [Benavides *et al.*, 2010].

Void feature model. If model constraints in CF are inconsistent (inconsistent(CF $\cup$ c_0)), we are interested in solutions to the FM diagnosis task (S=CF, AC = CF $\cup$ c_0). In this case we want to figure out which are the minimal sets of constraints that are responsible for the given inconsistency in the feature model. We do not include c_0 (e.g., $c_0 : ubuntu = true$) in the set S since we are not interested in changing this constraint. The feature model of our example (see Figure 1) is an example of a void feature model.

Note that for the following discussions we assume that $\Delta_5 = \{c_6, c_7\}$ (see Figure 2) has been chosen by the engineer and $\{c_6, c_7\}$ have been deleted from the feature model.

Dead feature f_i. If a feature f_i is not included in any of the possible configurations (i.e., inconsistent(CF $\cup$ $f_i = true$)), we are interested in solutions to the FM diagnosis task (S = CF, AC = CF $\cup$ $\{c_0\}$ $\cup$ $\{f_i = true\}$). This way we are able to figure out the minimal sets of constraints that are responsible for the non-acceptance of f_i. In our working example, there is no such dead feature (assuming that the constraints in Δ_5 have been deleted from the feature model). If we would substitute the constraint $c_9 : games \rightarrow gui$ with $c_9 : \neg gui \vee \neg games$, the feature *games* would be a dead feature. If we then want to make *games* a feature which is included in at least one configuration, the diagnoses for (S = CF, AC = CF $\cup$ $\{c_0\}$ $\cup$ $\{games = true\}$) are $\Delta_1 = \{c_3\}$ and $\Delta_2 = \{c_9\}$.

Conditionally dead feature f_i. Such a feature f_i is not included in all of the possible configurations, i.e., consistent (CF $\cup$ $\{c_0\}$ $\cup$ $\{f_i$=false$\}$) and consistent (CF $\cup$ $\{c_0\}$ $\cup$ $\{f_i$=true$\}$). If we want to have f_i in each configuration, we have to add $\{f_i = true\}$ to the set CF. In our working example, *games* is a conditionally dead feature since there are also solutions with no inclusion of this feature. In order to make *games* part of every possible feature model configuration, we have to make this clear in the feature model. One way to achieve this would be to convert constraint c_4 into a mandatory constraint – this would have the same effect as adding $games = true$ as an additional constraint to CF.

Full mandatory feature f_i. A feature f_i is fully mandatory if it is included in every possible solution (configuration), i.e., inconsistent(CF $\cup$ $\{c_0\}$ $\cup$ $\{f_i = false\}$). If we want to adapt the feature model in such a way that it also allows f_i to be not included, we can determine the corresponding (minimal) sets of responsible constraints by solving the FM diagnosis task (S=CF, AC= CF $\cup$ $\{c_0\}$ $\cup$ $\{f_i = false\}$). In our working example, the feature *gui* is a full mandatory feature since it is part of every possible configuration. If we want to allow configurations where *gui* is not included, the only diagnosis for (S=CF, AC= CF $\cup$ $\{c_0\}$ $\cup$ $\{gui = false\}$) is $\Delta_1 = \{c_3\}$.

False optional feature f_i. A *false optional feature* f_i is included in all configurations (e.g., products of a product line) although it has not been modeled as mandatory. If we replace the constraint $c_9 : games \rightarrow gui$ with $c_9 : gui \rightarrow games$, the feature *games* becomes a false optional feature since it is included in every possible configuration. An alternative interpretation of a false optional feature focuses on the *optional* relationship between a feature f_{par} and f_{opt}. If the consistency check of (CF $\cup$ $\{c_0\}$ $\cup$ $\{f_{par} = true \wedge f_{opt} = false\}$) returns *false* (and f_{par} = $true$), the feature f_{opt} is not an option. In our example (under the assumption that c_9 is adapted as mentioned), the diagnosis for (S = CF, AC = CF $\cup$ $\{c_0\}$ $\cup$ $\{ubuntu = true \wedge games = false\}$) is $\Delta_1 = \{c_3\}$.

Redundant constraint c_i. In our working example the constraint $c_9 : games \rightarrow gui$ is redundant since *gui* is a full mandatory feature. If we check the consistency of {CF - $\{c_9\}$ $\cup$ $\neg$CF} we see that c_9 is redundant since the expression is inconsistent. In other words, CF - $\{c_9\}$ $\models$ c_9, i.e., c_9 logically follows from CF - $\{c_9\}$ – therefore it is redundant. The second redundant constraint in our working example is c_4 since the feature *ubuntu* is a full mandatory feature as well. Con-

Analysis operation	Property Check	Explanation (Diagnosis Task)
Void feature model	inconsistent(CF $\cup \{c_0\}$)?	FASTDIAG(CF,CF $\cup \{c_0\}$)
Dead (f_i)	inconsistent(CF $\cup \{c_0\} \cup \{f_i$=true$\}$)?	FASTDIAG(CF,CF $\cup \{c_0\} \cup \{f_i$ = true$\}$)
Conditionally dead (f_i)	consistent(CF $\cup \{c_0\} \cup \{f_i$=false$\}$) and consistent(CF $\cup \{c_0\} \cup \{f_i$=true$\}$)?	CF $\leftarrow$ CF $\cup \{f_i$=true$\}$
Full mandatory (f_i)	inconsistent(CF $\cup \{c_0\} \cup \{f_i$=false$\}$)?	FASTDIAG(CF,CF $\cup \{c_0\} \cup \{f_i = false\}$)
False optional (f_{opt})	inconsistent(CF $\cup \{c_0\} \cup$ $\{f_{par}$=true $\wedge f_{opt}$=false$\}$)?	FASTDIAG(CF, CF $\cup \{c_0\} \cup$ $\{f_{par} = true \wedge f_{opt} = false\}$)
Redundant (c_i)	inconsistent((CF $\cup \{c_0\}$ - $\{c_i\}$) $\cup \neg$(CF $\cup c_0$))?	$c_i \notin$ FMCORE(CF $\cup \{c_0\}$)

Table 1: Feature model analysis operations, property checks, and related explanations. For example, figuring out whether a feature model is void (no solution can be found) can be determined on the basis of a consistency check *inconsistent* (CF $\cup \{c_0\}$). A related explanation can be determined by solving the FM diagnosis task (CF, CF $\cup \{c_0\}$). The related diagnosis (FASTDIAG) and redundancy detection algorithm (FMCORE) are discussed in Section 4.

sequently, the constraints $\{c_4, c_9\}$ can be deleted from the feature model without changing the underlying semantics.[3]

In the following section we focus on the presentation of two algorithms which help to determine explanations for the different feature model anomaly patterns.

4 Explaining Anomalies

The two basic algorithms for determining diagnoses and redundancies are FASTDIAG and FMCORE. FASTDIAG [Felfernig *et al.*, 2012] is a divide-and-conquer algorithm that supports the efficient determination of minimal diagnoses without the need of having conflict sets available. FMCORE is an algorithm which focuses on the determination of *minimal cores*, i.e., redundancy-free subsets of a constraint set.

Determination of Diagnoses. In FASTDIAG (see Algorithm 1), the set S represents the set of constraints where a diagnosis should be searched, The set AC contains all constraints of the feature model. For example, if we want to diagnose a *void feature model* (CF $\cup \{c_0\}$ is inconsistent – see Table 1), we would activate the algorithm with FASTDIAG(CF,CF $\cup \{c_0\}$), i.e., S = CF and AC = CF $\cup \{c_0\}$. We do not include c_0 in the set of diagnosable constraints since c_0 (the root constraint) is assumed to be correct (e.g., c_0 : $ubuntu = true$). First, the algorithm (see Algorithm 1) checks whether the considered constraint set can be diagnosed (if the set S is empty, no diagnosis will be found) and whether the constraints in AC-S are inconsistent (in this case no diagnosis can be determined).

Algorithm 1 FASTDIAG(S, AC): Δ

if $isEmpty(S)$ *or* $inconsistent(AC - S)$ **then**
  $return\ \emptyset$;
else
  $return\ DIAG(\emptyset, S, AC)$
end if

The major idea of FASTDIAG (and its subfunction DIAG – see Algorithm 2) is to divide a set S of inconsistent constraints into two subsets S_1 and S_2. If the first part becomes consistent, the diagnosis is searched in the other part and the first part can be omitted (no constraints part of the diagnosis will be found there). If a *singleton* constraint of S triggers an inconsistency, this constraint is considered a part of the diagnosis. FASTDIAG determines exactly one diagnosis at a time. If we want to determine more than one or even the complete set of diagnoses, we need to combine FASTDIAG with a corresponding algorithm that supports the construction of HSDAGs. The discussion of this approach is outside the scope of this paper. We want to refer the reader to the work of Felfernig et al. [Felfernig *et al.*, 2012]. Compared to traditional diagnosis approaches, FASTDIAG needs in the worst case $2d \times log_2(\frac{n}{d}) + 2d$ consistency checks where d is the number of constraints in the minimal diagnosis and n is the number of constraints in S [Felfernig *et al.*, 2012]. The corresponding best case complexity in terms of the number of consistency checks is $log_2(\frac{n}{d}+2d)$. A similar worst case (and best case) complexity in traditional diagnosis approaches can be expected for each determination of a conflict set (see, e.g., Figure 2) [Felfernig *et al.*, 2012].

Algorithm 2 DIAG(D, $S = \{s_1, ..., s_r\}$, AC): Δ

if $D \neq \emptyset$ *and* $consistent(AC)$ **then**
  $return\ \emptyset$;
end if
if $singleton(S)$ **then**
  $return\ S$;
end if
$k \leftarrow \lceil \frac{r}{2} \rceil$;
$S_1 \leftarrow \{s_1, ..., s_k\}; S_2 \leftarrow \{s_{k+1}, ..., s_r\}$;
$\Delta_1 \leftarrow DIAG(S_2, S_1, AC - S_2)$;
$\Delta_2 \leftarrow DIAG(\Delta_1, S_2, AC - \Delta_1)$;
$return(\Delta_1 \cup \Delta_2)$;

Determination of Redundancies. A constraint f_i of a feature model (represented by the constraint set CF) is redundant if its deletion from the model does not change the set of possible solutions. More formally, CF - $\{f_i\} \models f_i$ which means that f_i logically follows from CF - $\{f_i\}$ and therefore is redundant. An algorithm for redundancy detection should definitely not check redundancy properties on the basis of concrete configurations since such an approach becomes com-

[3]Note that redundancies can also be intended to achieve goals such as improving understandability or increasing efficiency – a discussion of related issues is outside the scope of this paper.

Feature Model: Car Selection	#Variables: 72		#Constraints:96			
# Diagnoses	Inconsistency Rate					
	2% (8 diagnoses)		5% (64 diagnoses)		7% (182 diagnoses)	
	FASTDIAG	HSDAG	FASTDIAG	HSDAG	FASTDIAG	HSDAG
1	452	874	561	1888	858	5366
2	749	890	920	1891	1638	5382
3	1045	921	1294	2138	2059	5506
4	1373	936	1653	2143	2324	5522
5	1529	968	1872	2262	2464	5544
10	–	–	2511	2418	2932	5709
20	–	–	2964	2450	3806	6162
all	1632	1027	4383	3339	11856	8860

Table 2: Evaluation of FASTDIAG and HSDAG with the *Car Selection* feature model from S.P.L.O.T.

Feature Model: SmartHome V. 2.2	#Variables: 61		#Constraints:63			
# Diagnoses	Inconsistency Rate					
	2% (8 diagnoses)		5% (12 diagnoses)		7% (77 diagnoses)	
	FASTDIAG	HSDAG	FASTDIAG	HSDAG	FASTDIAG	HSDAG
1	297	920	312	952	577	2683
2	437	967	452	968	951	2684
3	609	983	592	983	1341	2686
4	734	998	733	1139	1762	2699
5	843	1014	842	1155	2090	2671
10	–	–	967	1529	2792	2715
20	–	–	–	–	3369	2746
all	1155	1061	1606	1545	6224	3151

Table 3: Evaluation of FASTDIAG and HSDAG with the *SmartHome V 2.2* feature model from S.P.L.O.T.

pletely inefficient even in the case of simple feature models. The basic idea of the FMCORE algorithm is to iterate over the given set of constraints (S) and for each constraint $c_i \in S$ to check whether the deletion of c_i changes the semantics of S. The assumption is that if c_i is non-redundant, its deletion from S will change the semantics of S, i.e., $S - \{c_i\} \cup \overline{S}$ becomes consistent. All these individual redundant constraints are deleted from S_{temp} (a temporal copy of S). Finally, the algorithm returns the set S_{temp} which represents a minimal core, i.e., the original set S without redundant constraints.

Note that – instead of checking the inconsistency of $C_S - \{c_i\} \cup \overline{S}$ (see, e.g., [Felfernig *et al.*, 2011]) – FMCORE systematically reduces the number of constraints to be checked in $\overline{S}$. Given a configuration knowledge base S and its complement $\overline{S}$, the (in)consistency check of $S - \{c_i\} \cup \overline{S}$ can be reduced to the inconsistency check of $S - \{c_i\} \cup \overline{S'}$ where $\overline{S'} = \{\neg c_i\}$. If we assume that $S = \{c_1 \wedge c_2 \wedge .. \wedge c_m \wedge c_{m+1} \wedge .. \wedge c_n\}$, $\overline{S} = \{\neg c_1 \vee \neg c_2 \vee .. \vee \neg c_m \vee \neg c_{m+1} \vee .. \vee \neg c_n\}$, and $\gamma = \{c_{m+1} \wedge .. \wedge c_n\}$ then the consistency check of $S - \gamma \cup \overline{S}$ can be reduced to $\{c_1 \wedge c_2 \wedge .. \wedge c_m\} \cup \{\neg c_{m+1} \vee .. \vee \neg c_n\}$. In FMCORE (Algorithm 3) this property is taken into account.

The number of consistency checks of FMCORE in the best case equals the number of consistency checks in the worst case – in both cases the number of consistency checks needed is exactly n (the number of constraints in S).

In order to analyze the performance of FASTDIAG and FMCORE we conducted a performance analysis for both algorithms on the basis of different feature models provided by the *S.P.L.O.T.* repository. The results of this analysis are presented in the following section.

Algorithm 3 FMCORE(S): Δ

```
{S: the (redundant constraint set)}
{S̄: the complement of S}
{Δ: set of redundant constraints}
S_temp ← S;
for all c_i in S_temp do
  if isInconsistent((S_temp − {c_i}) ∪ {¬c_j}) then
    S_temp ← S_temp − {c_i};
  end if
end for
return S_temp;
```

5 Performance Evaluation

For evaluation purposes we selected different feature models offered by the *S.P.L.O.T.* repository: *Car Selection* (Table 2), *SmartHome V. 2.2.* (Table 3), and *Xerox* (Table 4). In order to evaluate the performance of FASTDIAG, we randomly inserted additional cross-tree constraints in the feature models for inducing inconsistencies which could then be exploited for determining minimal diagnoses. For a systematic evaluation we generated different versions of the (inconsistent) feature models which differed in terms of their inconsistency

Feature Model: Xerox	#Variables: 172		#Constraints:205			
# Diagnoses	Inconsistency Rate					
	2% (140 diagnoses)		5% (84 diagnoses)		7% (55 diagnoses)	
	FASTDIAG	HSDAG	FASTDIAG	HSDAG	FASTDIAG	HSDAG
1	1638	3354	1260	2996	1740	3023
2	2013	6646	1710	3167	2050	3203
3	2262	12106	1970	9454	2330	9544
4	2434	12355	2180	9536	2580	9654
5	2637	28111	2341	12044	2790	12165
10	3417	69950	2921	64631	3330	65240
20	4758	75317	3911	90715	5010	91726
all	46785	>100000	17301	>100000	10541	>100000

Table 4: Evaluation of FASTDIAG and HSDAG with the *Xerox* feature model from S.P.L.O.T.

Feature Model	#Variables	#Constraints	Redundancy Rate	Runtime (ms)
Car Selection	72	96	0.64	5070
SmartHome V. 2.2	61	63	0.29	1907
Xerox	172	205	0.71	3261

Table 5: Evaluation of FMCORE with selected S.P.L.O.T. feature models.

rate (see Formula 1) which was categorized in $\{2\%, 5\%, 7\%\}$. We used a random variable to control the degree of generated inconsistencies (the number of conflicts) in a feature model. As reasoning engine we used the CHOCO constraint solving library.[4] In order to import feature models to our environment we implemented a parser that generated CHOCO knowledge bases from S.P.L.O.T. SXFM based feature models.

$$Inconsistency\ Rate = \frac{\#conflicts\ in\ FM}{\#constraints\ in\ FM} \quad (1)$$

The performance tests were executed within a Java application running on a 64bit Windows 7 desktop PC using 8GB RAM and an Intel(R) Core(TM) i5-2320 CPU with 3.0GHz. Each run of the diagnosis algorithm for a specific setting has been repeated 10 times were in each run the ordering of the constraints was randomized. For each setting we evaluated the runtime (in ms) of both, the standard hitting set based approach to the termination of diagnoses [Reiter, 1987] (HSDAG) and FASTDIAG. As scenario we choose the diagnosis of *void feature models* where we induced different degrees of inconsistency (based on the *inconsistency rate measure* – see Formula 1). The upper bound for the evaluation time was set to 100.000 ms – in the case that this upper limit was exceeded, the search was stopped.

If one or a few diagnoses are required (which is typical for interactive settings) then FASTDIAG outperforms the standard HSDAG approach in most of the cases. If all diagnoses are required, for example, in situations where diagnoses are computed offline, the standard HSDAG approach seems to be the better choice. We want to emphasize that the presented diagnosis algorithms are independent of the underlying reasoning mechanisms, i.e., beside using a basic constraint-based approach for supporting the reasoning tasks (mainly consistency checking), description logics or SAT-based approaches can be applied as well. Finally, we also evaluated the performance of the redundancy detection algorithm FMCORE (see Table 5). Our goal was to figure out for the selected feature models to which extent the constraints in the feature models are redundant. We measured redundancy in the terms of the *redundancy rate* (see Formula 2).

$$Redundancy\ Rate = \frac{\#redundant\ constraints\ in\ FM}{\#constraints\ in\ FM} \quad (2)$$

The outcome of this analysis was that all the investigated feature models showed quite different degrees of redundancy (see Table 5). However, we consider these as preliminary results and further analyses have to be conducted, for example, we are interested in intra-constraint redundancies and the share of redundancy in cross-tree constraints with regard to the overall number of constraints in the feature model.

Note that the FMCORE algorithm is especially useful in situations where models are developed by one or a few engineers. In this case the degree of redundant constraints in the model is low. For scenarios with high redundancy rate, alternative algorithms have already been developed (see, e.g., [Felfernig *et al.*, 2011]).

6 Conclusions

In this paper we presented a consistency-based approach to explaining anomalies in feature models. We introduced definitions which are useful for the explanation of anomalies and discussed the corresponding algorithms which help to determine minimal diagnoses (FASTDIAG) and minimal sets of non-redundant constraints (FMCORE). Our future work will focus on: (1) The definition of further anomaly patterns in alternative knowledge representations such as *advanced feature models* [Batory, 2005] and UML models [Felfernig *et al.*, 2000]. Due to higher expressiveness, these representations include further anomaly patterns such as multiplicity

[4] www.emn.fr/z-info/choco-solver.

bounds which can not represented by configurations, unsatisfiable preconditions in constraints, and unexplained incompatibilities. (2) The development of mechanisms for the automated generation of test cases for feature models. (3) Further algorithms that enable the determination of diagnoses and redundancies on an intra-constraint level. (4) Evaluation of the developed algorithms with further benchmarks.

References

[Bakker *et al.*, 1993] R. Bakker, F. Dikker, F. Tempelman, and P. Wogmim. Diagnosing and solving over-determined constraint satisfaction problems. In *Proceedings of IJCAI-93*, pages 276–281. Morgan Kaufmann, 1993.

[Batory *et al.*, 2006] D. Batory, D. Benavides, and A. Ruiz-Cortes. Automated analysis of feature models: challenges ahead. *Comm. of the ACM*, 49:45–47, 2006.

[Batory, 2005] D. Batory. Feature Models, Grammars, and Propositional Formulas. In H. Obbink and K. Pohl, editors, *Software Product Lines Conference*, volume 3714 of *LNCS*, pages 7–20. Springer, 2005.

[Benavides *et al.*, 2010] D. Benavides, S. Segura, and A. Ruiz-Cortes. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35:615–636, 2010.

[Benavides *et al.*, 2013] D. Benavides, A. Felfernig, J. Galindo, and F. Reinfrank. Automated Analysis in Feature Modelling and Product Configuration. In *13th International Conference on Software Reuse (ICSR 2013)*, number 7925 in LNCS, pages 160–175, Pisa, Italy, 2013.

[Chandola *et al.*, 2009] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41:15:1–15:58, July 2009.

[Czarnecki *et al.*, 2005] K. Czarnecki, S.Helsen, and U.Eisenecker. Formalizing Cardinality-based Feature Models and their Specialization. *SoftwareProcess: Improvement and Practice*, 10(1):7–29, 2005.

[Felfernig *et al.*, 2000] A. Felfernig, G. E. Friedrich, and D. Jannach. UML as Domain Specific Language for the Construction of Knowledge-based Configuration Systems. *International Journal of Software Engineering and Knowledge Engineering*, 10(4):449–469, 2000.

[Felfernig *et al.*, 2004] A. Felfernig, G. Friedrich, D. Jannach, and M. Stumptner. Consistency-based diagnosis of configuration knowledge bases. *Artificial Intelligence*, 152(2):213 – 234, 2004.

[Felfernig *et al.*, 2011] A. Felfernig, C. Zehentner, and P. Blazek. Corediag: Eliminating redundancy in constraint sets. In *22nd International Workshop on Principles of Diagnosis*, pages 219–224, Murnau, Germany, 2011.

[Felfernig *et al.*, 2012] A. Felfernig, M. Schubert, and C. Zehentner. An efficient diagnosis algorithm for inconsistent constraint sets. *AI for Engineering Design, Analysis, and Manufacturing (AIEDAM)*, 26(1):53–62, 2012.

[Felfernig, 2007] A. Felfernig. Standardized configuration knowledge representations as technological foundation for mass customization. *IEEE Transactions on Engineering Management*, 54:41–56, 2007.

[Fleischanderl, 2002] G. Fleischanderl. Suggestions from the software engineering practice for applying consistency-based diagnosis to configuration knowledge bases. In *13th Intl. Workshop on Principles of Diagnosis (DX-02)*, pages 33–35, Semmering, Austria, 2002.

[Junker, 2004] U. Junker. QuickXPlain: preferred explanations and relaxations for over-constrained problems. In *Proceedings of the 19th National Conference on Artifical Intelligence*, AAAI 2004, pages 167–172. AAAI, 2004.

[Kang *et al.*, 1990] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature-oriented Domain Analysis (FODA) – Feasibility Study. *TechnicalReport CMU – SEI-90-TR-21*, 1990.

[Mendonca and Cowan, 2010] M. Mendonca and D. Cowan. Decision-making coordination and efficient reasoning techniques for feature-based configuration. *Science of Computer Programming*, 75(5):311 – 332, 2010. Coordination Models, Languages and Applications(SAC 2008).

[Reiter, 1987] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[Segura *et al.*, 2010] S. Segura, R. Hierons, D. Benavides, and A. Ruiz-Cortes. Automated test data generation on the analyses of feature models: A metamorphic testing approach. In *3rd Intl. Conference on Software Testing, Verification and Validation (ICST)*, pages 35–44, 2010.

[Trinidad *et al.*, 2008] P. Trinidad, D. Benavides, A. Duran, A. Ruiz-Cortez, and M. Toro. Automated error analysis for the agilization of feature modeling. *Journal of Systems and Software*, 81:883–896, 2008.

[Tsang, 1993] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, London, 1993.

[von der Massen and Lichter, 2004] T. von der Massen and H. Lichter. Deficiencies in Feature Models. In T. Mannisto and J. Bosch, editors, *Workshop on Software Variability Management for Product Derivation*, 2004.

[Wang *et al.*, 2010] B. Wang, Y. Xiong, Z. Hu, H. Zhao, W. Zhang, and H. Mei. A dynamic-priority based approach to fixing inconsistent feature models. In D. Petriu, N. Rouquette, and O. Haugen, editors, *Model Driven Engineering Languages and Systems*, volume 6394 of *LNCS*, pages 181–195. Springer Berlin, 2010.

[White *et al.*, 2010] J. White, D. Benavides, D. Schmidt, P. Trinidad, B. Dougherty, and A. Ruiz-Cortes. Automated diagnosis of feature model configurations. *Journal of Systems and Software*, 83(7):1094–1107, 2010.