An Optimization Strategy to Feature Models' Verification by Eliminating Verification-Irrelevant Features and Constraints

Hua Yan, Wei Zhang, Haiyan Zhao, and Hong Mei

Key Laboratory of High Confidence Software Technology, Ministry of Education of China,
Insititute of Software, School of Electronics Engineering and Computer Science,
Peking University, Beijing, 100871, China
{yanhua07,zhangw,zhhy}@sei.pku.edu.cn, meih@pku.edu.cn

Abstract. Feature models provide an effective approach to requirements reuse. One important problem related to feature models is the verification problem, which is *NP-complete* in theory. The existing approaches to feature models' verification mostly focus on how to automate the verification of feature models using third-party's tools, while these tools are usually designed to resolve general kinds of problems. However, by simply using these third-party's tools, large-sized feature models still can hardly be verified within acceptable time. We argue that, to improve the efficiency of verification, the problem itself should be at first optimized. In this paper, we propose an optimization strategy to feature models' verification, in which, verification-irrelevant features and constraints are eliminated from feature models and the problem size of verification is therefore reduced. We prove the correctness of this strategy, while experiments show the effectiveness of this strategy.

Keywords: Feature model, Verification, Problem size, Reduction.

1 Introduction

In software reuse, feature models provide an effective approach to modeling and reusing requirements in a specific software domain. The modeling responsibility of a feature model is achieved by encapsulating requirements into a set of features and clarifying possible dependencies among features. The reusing responsibility of a feature model is carried out through customization - that is, selecting a subset of features from a feature model, while maintaining those constraint dependencies among features.

One important problem related to feature models' customization is the verification problem. The verification of a feature model has two purposes. Before customization, the verification aims to find possible conflicts in constraints among features. After customization, the verification is intended to find possible conflicts in customizing-decisions on a feature model. Based on the discovery that constraints among features can be formalized as propositional formulas [4,7,1], the verification of a feature model can be formalized correspondingly as a set of propositional satisfiability (*SAT*)

S.H. Edwards and G. Kulczycki (Eds.): ICSR 2009, LNCS 5791, pp. 65–75, 2009.

problems, which are *NP-complete* in general. Because of the *NP-complete* nature, the verification of a complex feature model inevitably suffers from the state space explosion problem. For example, for a feature model with 1000 features and many complex constraints, the state-space of the verification problem is as large as 2^{1000}. Therefore, it is often infeasible to verify complex feature models without adopting any optimization strategy.

In the existing research on feature models' verification, most of them focuses on how to automate the verification of feature models using third-party's tools (such as those *SAT*-solvers, *CSP*-solvers, and Model Checkers). However, as far as to our knowledge, very little work focuses on how to optimize the verification of feature models based on the characteristics of feature models' verification (except some of our previous work [7,8]). Since those third-party's tools are often designed to resolve general kinds of problem, it is impossible for them to incorporate any optimization strategy specific to feature models' verification.

In this paper, we propose an optimization strategy to feature models' verification by eliminating verification-irrelevant features and constraints from feature models, and prove the correctness of this strategy. This strategy is developed based on the observation that most feature models contain some features and constraints that are irrelevant to feature models' verification (in the sense that these verification-irrelevant features and constraints can be safely removed without changing the results of feature models' verification), while the problem size of a feature model's verification is exponential to the number of features and constraints in the feature model. Therefore, eliminating verification-irrelevant features and constraints from feature models will reduce the problem size of verification, and alleviate the state space explosion problem. Experiments have shown that this strategy improves the efficiency of and the capability to feature models' verification.

The rest of this paper is organized as follows. Based on some preliminary knowledge introduced in Section 2, Section 3 presents the optimization strategy to feature models' verification, and gives its correctness proof. Experiments and analysis are shown in Section 4. Related works are discussed in Section 5. Finally, Section 6 concludes this paper with a short summary.

2 Preliminary

In this section, we first give a notation for feature models with formal semantics of constraints among features, and clarify two kinds of constraint in feature models according to the source of constraints, and then introduce the criteria for feature models' verification, which are proposed in our previous research [7].

2.1 A Notation for Feature Models

The symbols in this notation are listed in Table 1 and explained from the viewpoint of customization. The formal definitions of constraints among features are given in Table 2 and Table 3.

Table 1. Symbols in the notation for feature models

Symbol	Name	Explanation
	A *mandatory feature* named "*X*".	A mandatory feature must be selected in a customizing result, if its parent feature is selected or it hasn't a parent feature. If its parent is removed, it must be removed.
	An *optional feature* named "*Y*".	An optional feature can either be selected in or be removed from a customizing result, if its parent feature is selected or it hasn't a parent. If its parent is removed, it must be removed.
	A *refinement* relation between two features.	A refinement connects two features. The feature connecting to the non-arrow end is called the *parent* of the other feature. A feature can only have one parent feature at most.
	A *requires* constraint between two features.	A *requires* constraint connects two features. The feature connecting to the non-arrow end is called the *requirer*, and the other the *requiree*. This constraint means that if the *requirer* is selected, the *requiree* must be selected.
	An *excludes* constraint between two features.	An *excludes* constraint connects two features. This constraint means that the two features should not be both selected in a same customizing result.
	A *binding predicate* among a set of features and binding predicates	The left end connects to a *composite* constraint or to one of the right ends of a binding predicate. The right ends connect to a set of features or binding predicates. We define three *types* of binding predicate: *and* (denoted by ∧); *or* (denoted by ∨); *xor* (denoted by *1*). See **Table 2** for the formal definition of binding predicates.
	A *composite* constraint between two binding predicate	We define two *types* of composite constraint: *requires* (denoted by ⟶); *excludes* (denoted by ⤫). See Table 3 for their formal definitions.

Table 2. The formal definitions of binding predicates. In this table, *A* and *B* denotes features, and *p* and *q* denotes binding predicates. For a feature *F*, *bind*(*F*) is a predicate; it is *true* if *F* is selected, *false* otherwise. In our notation, we only use binding predicates as constituent parts of the composite constraints, but not as individual constraints.

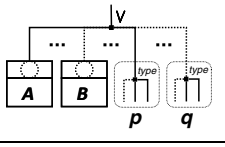
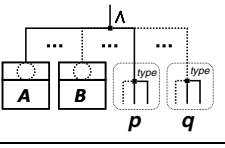
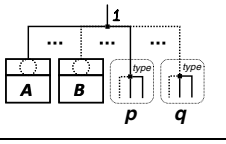
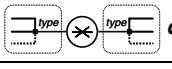
	or(*A*, …, *B*, …, *p*, …, *q*)	*and*(*A*, …, *B*, …, *p*, …, *q*)	*xor*(*A*, …, *B*, …, *p*, …, *q*)
Binding Predicate			
Formal Definition	$bind(A)\vee...\vee\neg bind(B)\vee...\vee p\vee...\vee\neg q$	$bind(A)\wedge... \wedge\neg bind(B)\wedge...\wedge p\wedge...\wedge\neg q$	$bind(A)\otimes... \otimes\neg bind(B)\otimes...\otimes p\otimes...\otimes\neg q$

Table 3. The formal definitions of composite constraints. In this table, *p* and *q* denotes binding predicates. In the situation that *p* and *q* only contains one feature, the two types of composite constraints becomes the *requires* and the *excludes* constraints between two features.

Composite Constraint	*requires*(*p*, *q*)	*excludes*(*p*, *q*)
		
Formal Definition	$p \rightarrow q$	$p \rightarrow \neg q$

2.2 Implicit Constraints and Explicit Constraints

Constraints in feature models can be classified into two kinds according to their source. The first kind is the implicit constraints that are imposed by refinements relations between features. In feature models, each refinement relation implicitly imposes a *requires* constraint between the involved two features – that is, the child feature *requires* the parent feature. The second kind is the explicit constraints that are explicitly added into feature models by model constructors.

Figure 1 shows a feature model example, in which, (a) is the refinement view, consisting of features and refinements between features, and (b) is the constraint view, consisting of those explicit constraints added by model constructors. Figure 1(c) shows all the constraints in the form of propositional formulas. The constraints from 1 to 6 are implicit constraints derived from the refinement view, and the constraints from 7 to 8 are explicit constraints.

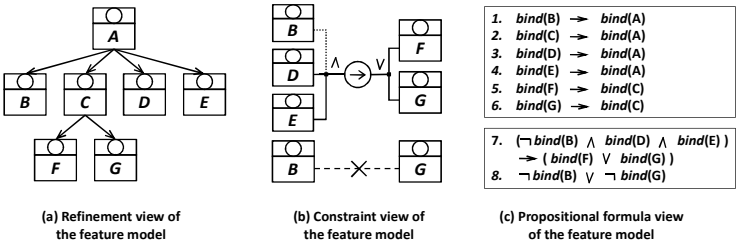


Fig. 1. A feature model example with explicit and implicit constraints

2.3 Three Criteria for Feature Models' Verification

In our previous research [7], we proposed three criteria for feature models' verification. According to the deficiency framework for feature models [10], the three criteria can detect all the *anomaly* and *inconsistency* deficiencies. Due to space limitation, we just list the three criteria as follows without further explanations.

1. There exists at least a set of customizing decisions to all undecided features in a feature model that will not violate any constraints in the feature model.
2. Each undecided feature in a feature model has the chance to be selected without violating any constraints in the feature model.
3. Each undecided feature in a feature model has the chance to be removed without violating any constraints in the feature model.

More information about what kinds of deficiencies exist in feature models and how the three criteria detect these deficiencies can be referred to literature [10] and [7].

3 Eliminating Verification-Irrelevant Features and Constraints

In this section, we present the optimization strategy to feature model's verification, and give the correctness proof of this strategy.

3.1 The Optimization Strategy

During feature models' verification, one kind of errors (called *implicit constraint violations*) can be easily detected and fixed at first. This kind of errors occurs when a child feature is selected while its parent is not. By checking the binding states of two features in each refinement, implicit constraint violations can be detected. Then, these violations can be fixed by removing all the offspring features of the removed features and selecting all the ancestor features of the selected features.

We have observed that certain features and constraints are irrelevant to the verification after handling *implicit constraint violations*. That is, these features and constraints can be safely eliminated without changing the verification results. Based on this observation, we develop an optimization strategy to feature models' verification. This strategy is based on the following two concepts.

Definition 1. Verification-irrelevant features.

A feature is verification-irrelevant, iff this feature does not appear in any explicit constraint, and none of its offspring features appears in any explicit constraint.

Definition 2. Verification-irrelevant constraints.

A constraint is verification-irrelevant, iff at least one feature involved in this constraint is verification-irrelevant.

Figure 2 shows the verification-irrelevant features and constraints in a feature model example. After eliminating these verification-irrelevant features and constraints, this feature model is reduced to the feature model in Figure 1.

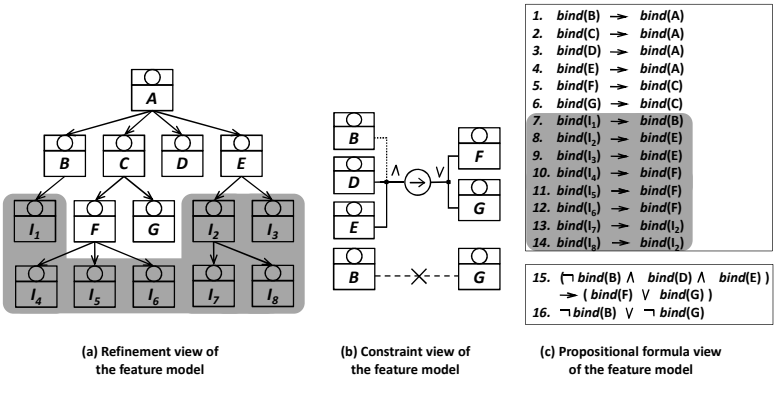


Fig. 2. A feature model example. The verification-irrelevant features and constraints are shown in the grey area. There are 8 verification-irrelevant features (I1 to I8 in (a)) and 8 verification-irrelevant constraints (7 to 14 in (c))

Based on the above two definitions, the optimization strategy can be expressed as follows. *Given a feature model without implicit constraint violations, eliminating those verification-irrelevant features and constraints from this feature model will not influence the verification result of this feature model.*

Following this strategy, we can deduce that the feature model in Figure 2 and the one in Figure 1 are equivalent from the verification point of view if the feature model in Figure 2 contains no implicit constraint violations.

3.2 The Correctness Proof

We prove the correctness of this strategy by two theorems. In the first theorem, we show that, after handling *implicit constraint violations*, if the reduced feature model satisfies the three verification criteria, the original feature model will also satisfy these criteria. To prove this theorem, three lemmas are first introduced. In the second theorem, we show that, if the original feature model satisfies the three verification criteria, the reduced feature model will also satisfy them. Based on the two theorems, the correctness of this strategy can be naturally deduced.

In the following, we use two pairs $<F, C>$ and $<F', C'>$ to denote the original feature model and the reduced feature model, respectively. In the two pairs, F and F' denote the feature sets of the two feature models, and C and C' denote the constraint sets. In addition, we use $<F, C-C'>$ to denote the feature model consisting of the feature set F and the set of verification-irrelevant constraints $C-C'$ being removed from the original feature model. The three verification criteria (see Section 2.3) on a feature model x are denoted by $VerifyC1(x)$, $VerifyC2(x)$ and $VerifyC3(x)$, respectively. The conjunction of the three criteria is denoted by $Verify(x)$.

Lemma 1. $\vdash (VerifyC1(<F, C-C'>) \wedge VerifyC1(<F', C'>)) \rightarrow VerifyC1(<F, C>)$

Proof: After handling *implicit constraint violations,* $VerifyC1(<F, C-C'>)$ is true, then, the assignment of F' can be expanded to the assignment of F by giving each undecided eliminated feature a removing customizing decision, which will not cause any conflict.

Lemma 2. $\vdash (VerifyC2(<F, C-C'>) \wedge VerifyC2(<F', C'>)) \rightarrow VerifyC2(<F, C>)$
Proof: Model reusers can select all the undecided eliminated features without violating any constraint.

Lemma 3. $\vdash (VerifyC2(<F, C-C'>) \wedge VerifyC2(<F', C'>)) \rightarrow VerifyC2(<F, C>)$
Proof: Model reusers can remove all the undecided eliminated features without violating any constraint.

Theorem 1. $Verify(<F, C-C'>) \vdash Verify(<F', C'>) \rightarrow Verify(<F, C>)$
Proof: We can deduce from Lemma 1, Lemma2 and Lemma 3 that

$\vdash (Verify(<F, C-C'>) \wedge Verify(<F, C'>)) \rightarrow Verify(<F, C'>))$

Thus, $\vdash (Verify(<F, C-C'>) \wedge Verify(<F, C'>)) \rightarrow Verify(<F, C'>))$

$\Rightarrow$ $Verify(<F, C-C'>) \wedge Verify\ (<F, C'>) \vdash Verify\ (<F, C>)$

$\Rightarrow$ $Verify(<F, C-C'>) \vdash Verify(<F', C'>) \rightarrow Verify(<F, C>)$

Theorem 2. $\vdash Verify(<F, C>) \rightarrow Verify(<F', C'>)$
Proof: $F' \subseteq F, C' \subseteq C \quad \Rightarrow \vdash Verify(<F, C>) \rightarrow Verify(<F', C'>)$

Corollary. $Verify(<F, C-C'>) \vdash Verify(<F', C'>) \leftrightarrow Verify(<F, C>)$

4 Experiments

In this section, we first introduce an algorithm of generating random feature models, and then apply our optimization strategy to the verification of a set of randomly generated feature models to demonstrate the effectiveness of this strategy.

4.1 An Algorithm of Generating Random Feature Models as Test Cases

To provide test cases for our experiments, we design an algorithm of generating random feature models (see Algorithm 1).

Algorithm 1. An algorithm of generating random feature models. In this algorithm, *GetRandomInt(n)* is a function that returns a random integer from 0 to *n*.

```
Input:
    fN  :  The number of features of the feature model.
    cN  :  The number of explicit constraints of the feature model.
   mNC  :  The maximum number of features in the constraints.
    mW  :  The maximum number of children of a feature. Its default value
           is 10.
     p  :  The percentage of verification-irrelevant features in the
           feature model. Its default value is -1, which means that the
           percentage of verification-irrelevant features is random.
Output:
   The random generated feature model
generate_random_fm(int fN,int cN,int mNC,int mW=10,int p=-1){
   FeatureModel fm = new FeatureModel();
   Queue queue = new Queue();
   Feature f = new Feature();
   queue.push(f);
   int counter = 1;
   while(!queue.isEmpty()){
     Feature parent = queue.pop();
     for(int i = 0; i < GetRandomInt(mW); i++){
        Feature child = new Feature();
        parent.addChild(child);
        fm.featureSet.add(child);
        counter++;
        if (p != -1 && counter <= (1-p)*fN){
         Constraint constraint = new constraint(child,parent,"requires");
         fm.addExplicitConstraint(constraint);
        }
        if (counter == fN) break L;
    }
   }
L:for(int i = 0; i < cN; i++){
    Set source = new Set(), target = new Set();
    for(int i = 0; i <GetRandomInt(mNC-2)+2; i++){
      if(GetRandomInt(1) == 0)
         source.add(fm.getFeature(GetRandom((1-p)*fN)));
       else// GetRandomInt(1) == 1
        target.add(fm.getFeature(GetRandom((1-p)*fN)));
    }
    String type;// the type of the constraint, see Table 3
    if(GetRandomInt(1) == 0) type = "requires";
    else type = "excludes";
    Constraint constraint = new constraint(source, target, type);
    fm.explicitConstraints.add(constraint);
   }
   return fm;
}
```

It should be noticed that in a feature model generated by Algorithm 1, all features are optional. This is because that, in our experiments, we assume feature models have been optimized by the atomic-set technique proposed in our previous research [7]. By applying this technique, mandatory features can be eliminated from feature models.

4.2 Experiments and Analysis

To make the experiments reflect the effectiveness of our strategy for feature models with different complexity, we generate three groups of feature model by varying the parameters of Algorithm 1. We use a BDD-based feature models' verifier [8] to verify the optimized feature models. The environment for our experiments is a computer with an Intel Core DUO 2.66GHz CPU, 2GB of memory and a Windows XP OS.

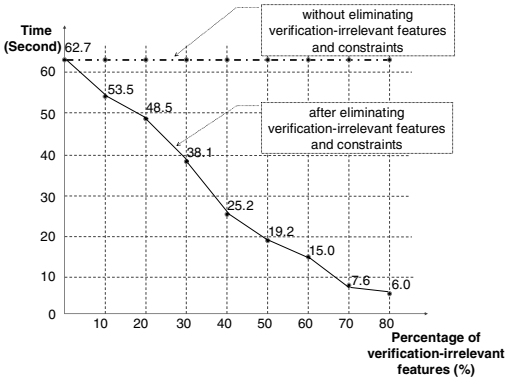


Fig. 3. Experiment results of the first group of test cases. This group has 9 feature models. All of them contain 500 features and 50 explicit constraints, and the percentage of verification-irrelevant features varies from 0% to 80% with an increment of 10%.

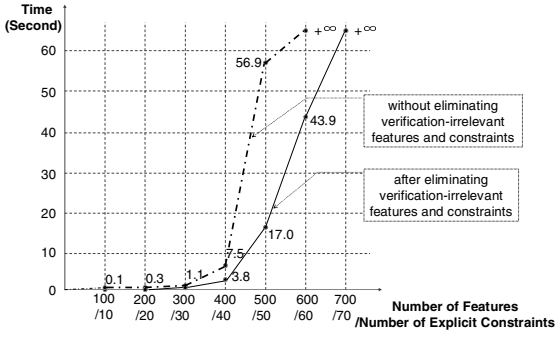


Fig. 4. Experiment results of the second group of test cases. This group has 7 feature models. The number of features in each test case varies from 100 to 700 with an increment of 100, the number of explicit constraints in each test case varies from 10 to 70 with an increment of 10, and the percentage of verification-irrelevant features is random.

Figure 3 shows the experiment results of the first group, from which we can see that our strategy improves the efficiency of verification for this group of test cases. Moreover, our strategy becomes more effective as the percentage of verification-irrelevant features increases.

Figure 4 shows the experiment results of the second group. We can see that although both of the solid curve and the dashed curve rise as the number of features and explicit constraints increases, the growth rate of solid curve is lower than that of the dashed curve, which means that our strategy decreases the time for feature models' verification. The experiment results also show that, for this group of test case, our strategy increases the capability to feature models' verification.

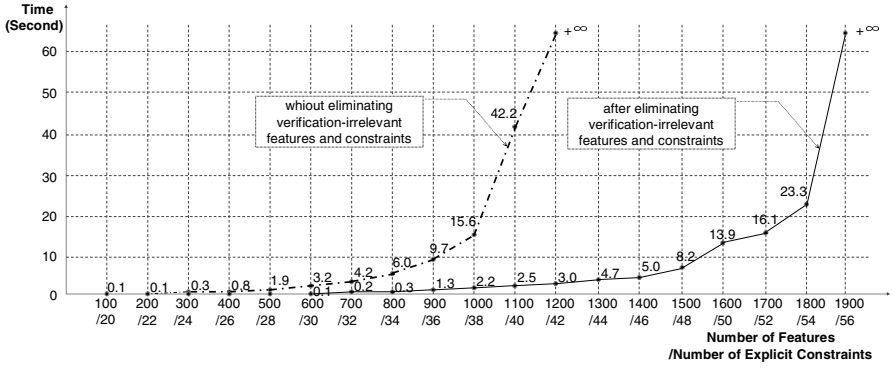


Fig. 5. Experiment results of the third group of test cases. This group has 19 feature models. The number of features in each test case varies from 100 to 1900 with an increment of 100. The number of explicit constraints in each test case varies from 20 to 56 with an incremental change of 2. The percentage of verification-irrelevant features is random.

Figure 5 shows the experiment results of the third group. We can see that, by applying our strategy, a feature model that contains 1500 features can be verified within 10 seconds. For this group of test cases, this strategy improves both the capability to and the efficiency of feature models' verification.

5 Related Work

The verification problem of feature models has been noticed since the first time feature models are proposed [3]. Existing research on feature models' verification can be classified into three categories: specification and formalization of verification criteria, formalization of feature models, and automation of verification. In the research on specification and formalization of verification criteria, der Maßen and Lichter [10] proposed a deficiency framework of feature models. In our previous work [7], we proposed three formalized verification criteria. Our investigation [9] shows that the three criteria can detect all kinds of anomaly and inconsistency deficiency in der Maßen and Lichter's deficiency framework. In the research on formalization of feature models, Mannion [4] proposed a first-order logic based method for the

formalization of feature models' constraints. In [7], we classified constraints in feature models into several types, and clarified their formal semantics based on propositional logic. In the research on automation of verification, several verification methods using third-party's tools are proposed. For example, Batory proposed a LTMS-based method [1]. In Czarnecki's research [2], a commercial BDD package is used. White et al. proposed a CSP-Solver-based approach [6].

However, little existing research has addressed how to optimize feature models' verification at the problem level. One exception is the atomic-set technique proposed in our previous work [7]. Segura gave a quantitative analysis to the effectiveness of the atomic-set technique [5]. The strategy proposed in this paper can be integrated with the atomic-set technique through sequential composition. That is, a feature model can be first reduced by the atomic-set technique, and then be further reduced through the strategy in this paper. Furthermore, these two optimization techniques can also be seamlessly integrated with the existing approaches to feature models' verification by equipping these approaches with a preprocessing of optimization.

6 Conclusions

In this paper, we proposed an optimization strategy to feature models' verification, and proved the correctness of this strategy. This strategy provides a way to eliminate features and constraints that are irrelevant to feature models' verification. Experiment results demonstrate that by applying this strategy, the verification efficiency of feature models is improved, and the verification capability is enhanced as well.

Acknowledgments. This work is supported by the National Grand Fundamental Research 973 Program of China under Grant No. 2009CB320701, the Science Fund for Creative Research Groups of China under Grant No. 60821003, the Hi-Tech Research and Development Program of China under Grant No. 2006AA01Z156, and the Natural Science Foundation of China under Grant No. 60703065 and 60873059.

References

1. Batory, D.: Feature Models, Grammars, and Propositional Formulas. In: Obbink, H., Pohl, K. (eds.) SPLC 2005. LNCS, vol. 3714, pp. 7–20. Springer, Heidelberg (2005)
2. Czarnecki, K., Kim, C.H.P.: Cardinality-Based Feature Modeling and Constraints: A Progress Report. In: OOPSLA 2005 International Workshop on Software Factories (2005)
3. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-Oriented Domain Analysis Feasibility Study. Technical reports, Software Engineering Institute, Carnegie Mellon University (1990)
4. Mannion, M.: Using First-Order Logic for Product Line Model Validation. In: Chastek, G.J. (ed.) SPLC 2002. LNCS, vol. 2379, pp. 176–187. Springer, Heidelberg (2002)
5. Segura, S.: Automated Analysis of Feature Models using Atomic Sets. In: Workshop on Analyses of Software Product Lines, in Conjunction with the 12th Software Product Line Conference (2008)
6. White, J., Benavides, D., Schmidt, D.C., Trinidad, P., Ruiz-Cortés, A.: Automated diagnosis of product-line configuration errors in feature models. In: Proceedings of the 12th Software Product Line Conference (2008)

7. Zhang, W., Zhao, H., Mei, H.: A Propositional Logic-Based Method for Verification of Feature Models. In: Proceedings of 6th International Conference on Formal Engineering Methods, pp. 115–130 (2004)
8. Zhang, W., Yan, H., Zhao, H., Jin, Z.: A BDD-Based Approach to Verifying Clone-Enabled Feature Models' Constraints and Customization. In: Mei, H. (ed.) ICSR 2008. LNCS, vol. 5030, pp. 186–199. Springer, Heidelberg (2008)
9. Zhang, W., Mei, H., Zhao, H.: Feature-Driven Requirements Dependency Analysis and High-Level Software Design. Requirements Engineering Journal 11(3), 205–220 (2006)
10. von der Maßen, T., Lichter, H.: Deficiencies in feature models. In: Workshop on Software Variability Management for Product Derivation, in Conjunction with the 3rd Software Product Line Conference (2004)