FAKULTÄT FÜR
INFORMATIK

Master Thesis

Survey on Automated Analysis Techniques

1.1 Motivation and Problem Context

There is a growing need, while developing more complex software systems, that demands better support to reuse software artifacts at lower costs, in shorter development time, and with higher quality. SPL is an efficient and effective strategy by supporting large-scale reuse and dealing with many challenges of today's software development. SPL is a set of related systems that are built from a set of common software assets and share a managed, common set of features.

There are some challenges that face a developer during automated analysis operations execution on feature model: (i) some automated analysis techniques don't perform all known (public) operations (ii) many factors must be considered to support engineer in choosing the most suitable technique (iii) the enginees may need to apply technique on extended feature model; software engineers use techniques that help them to avoid these challenges and satisfy a market needs. There is a problem while choosing one of these techniques, so understanding how each technique works and making a detailed comparison with accurate evaluation helps industries applying automated analysis techniques in their own development environment.

This research to conduct a complete and detailed survey of publications, aiming at providing an overview of the existing automated analysis techniques, with their advantages, depending on provided operations of each technique, based on a wide search through papers.

1.2 Research Objectives

By the end of this study, the objectives will be the following:

- To review the existing automated analysis techniques.
- To compare how each technique was evaluated according to some important functions.

We conduct a literature review to go over relevant papers and recent researches that help identifying SPL concepts and applied techniques (how those work, their application conditions and domain conditions), to represent successful and effective method helps engineer finding most relevant product according to market needs.

1.3 Research Questions

To fulfill the purpose of this research, we represent the objectives that were mentioned in section 1.2 in the following research questions (RQ):

- **RQ1: Which analysis operations have been automated?**
  Rationale: Each automated analysis technique has some operations available in literature. The goal of this question is cataloging each technique with the related operations.

- **RQ2: What operations of analysis of on Feature Models (FM) were evaluated?**
  Rationale: Some operations have been evaluated using experiments to check in their performance. The goal of this question is to clarify which operations of each technique were evaluated.

- **RQ3: What are the experiments those were done in each paper and how have these been evaluated?**
  Researchers try to evaluate operations performance using experiments. The goal of this question is to study these experiments to clarify how these operations were evaluated and depicts the results.

1.4 Inclusion and exclusion criteria

We depend in our searching on keywords: Software Product Line (SPL), Software Product Line Engineering (SPLE), Feature Models (FM), and operations and techniques which are related automated analysis.
The qualitative component of the review considers studies that evaluate existing configuration techniques and the textual component considers publications that describe all SPL concepts and entire definitions.

The Articles which were included:

i) papers proposing any analysis operation on feature models in which the original model is not modified,
ii) Papers proposing the automation of any analysis on feature models.
iii) Papers that study performance evaluations of analysis operations.
iv) papers that study operations proposing information extraction where considered,.

Some related works should be discarded to keep the size and complexity of the review at a manageable level, are:

i) Papers presenting any application of the analysis of feature models rather than proposing new analyses,
ii) Papers dealing with the analysis of other kinds of variability models like OVM, decision models and further extensions of feature models like probabilistic feature models.
iii) Papers that contain very similar founded content.

1.5 Summary of the studies reporting performance results for analysis operations:

Firstly we introduce some studies about most important analysis techniques. Then, we summarize a comparison between some analysis operations of Feature Models:

- **[Mendonca et al., 2004]:**
  Firstly, study looks into broad category of analysis and transformation of feature models: those based on satisfiability solving (SAT) techniques, including consistency checking, finding dead and common features, model entailment and equivalence checking and interactive and batch configuration.

  The main goal of this study is to make such a hardness study for satisfiablity problems while analyzing feature model, by doing some experiments to ensure efficiency of these technique, despite very good performance of SAT solvers in performing SAT checks on large feature models, because a SAT solver may take an infeasible amount of time to check satisfiability of certain kinds of instances (large number of parameters such as number of variables and clauses).

  SAT solvers are systems used to decide satisfiability. SAT solvers usually process problems encoded in CNF. The core decision procedure behind many modern SAT solvers is the DPLL algorithm, which is a backtracking search algorithm. It successively assigns Boolean values to variables in the formula, as long as none of the clauses is violated.

  A hardness study is done depending on some experiments in order to address satisfiability problems that arise while analyze feature models. They depend on the entire hardness spectrum, from over-constrained to under-constrained, where feature models are easy to analyze for SAT solvers.

  Researchers, in this study, computed consistency checks on models with up to 10,000 features and a large number of cross-tree constraints and in the worst-case scenario the SAT solver took about 0.4 seconds to complete the check. In addition, we developed an algorithm for computing valid domains and observed processing times of about 22 seconds for models with 5,000 features and a fairly large number of cross-tree constraints.

- **[Gheyi et al., 2000]**
  Researchers propose a theory for FMs in Alloy. This theory is useful for automatically checking a number of properties in the Alloy Analyzer. For instance, we show how to yield all valid configurations of a FM. Moreover, we explain how to check whether FM transformations preserve wellformedness rules of a FM. Additionally; they used the Alloy Analyzer to check general properties. For instance, they show that the refactoring relation is reflexive and symmetric using a scope of 5 atoms. Finally, they compared G-Theory and R-Theory, which is useful for checking FM refactorings.

- **[Thomas Th¨um et al., 2009]**
  Here designers should avoid arbitrary edits and restructure them in terms of a sequence of specializations, generalizations, and refactorings, to understand the evolution of a feature model.
  An edit is a refactoring, i.e., no new products are added and no existing products are removed, a specialization meaning that some existing products are removed

and no new products are added, a generalization when new products are added and no existing products removed, or an arbitrary edit otherwise.

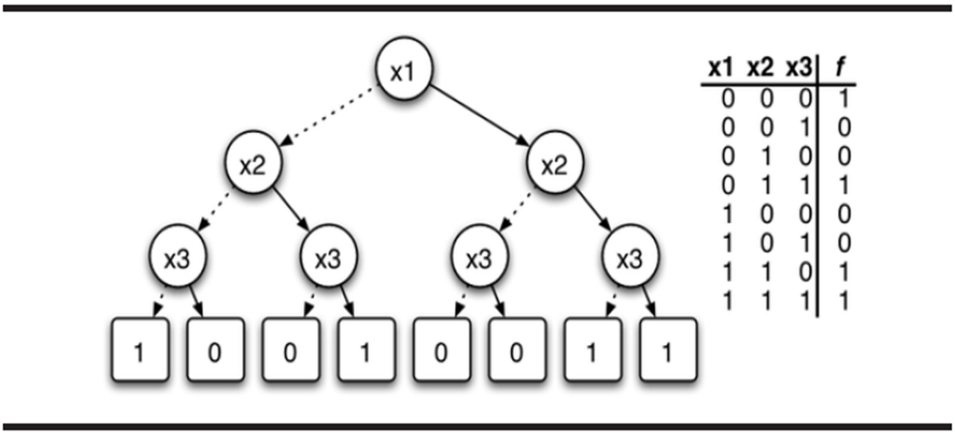
In this paper, authors present and evaluate an algorithm to determine the relationship between two feature models (i.e., specialization/refactoring/generalization/none of these) using satisfiability solvers.

They overcome the limitations of previous solutions: (a) arbitrary edits on feature models are supported, (b) the models to be compared need not have the same set of features, and (c) we empirically show that our algorithm can efficiently compare models with thousands of features.

They randomly generated feature models and performed random edits on them. In this way, we can parametrically control the size of feature models and the number and kind of edits which allows a thorough run-time evaluation.
They generated 200 random feature models with the same parameters and each performed the same number of random edits of the same kind, and for every 10 features, they create one cross-tree constraint

The parameters used in this generation (maximum number of children = 10; type of child group = (50 %,25 %, 25 %); optional child = 50 %; number of cross-tree constraints = 0:1 * n; variables in cross-tree constraints = 2–5) are fixed for in the entire experiment.

- **[Zhang et al,. 2008]**
  A Binary Decision Diagram (BDD) is a data structure used to represent boolean functions. A BDD is a rooted, directed, acyclic graph and is composed by a group of decision nodes and two types of terminal nodes called *0–terminal* and *1–terminal*. Each node in the graph represents a variable on a given formula and has two child nodes representing an assignment of the variable to 0 (dashed lines) and 1 (solid lines). All paths from the root to the 1–terminal represents the variable assignments for which the represented boolean function is true meanwhile all paths from the root to the 0–terminal represent the variable assignments for which the represented boolean function is false. Next Figure shows an example of BDD.



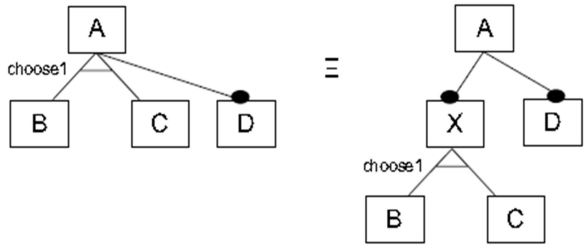
x1	x2	x3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

A BDD solver is a software package that takes as input a propositional formula and translate it into a BDD. Then, the solver can determine if the formula is satisfiable and can efficiently count the number of possible assignments. Other features are offered by BDD solvers depending on the implementation.

In this paper, researchers present a kind of semantics for constraints in clone enabled feature models, which resolves the problem of what kinds of constraint should be added to a feature model after some features are cloned. The semantics is composed of two patterns: the generating pattern and the adapting pattern, to address the two problems of what kind of constraints should be imposed on a clonable feature and its clones, and how an existing constraint should be transformed in the context that features involved in the constraint are cloned, respectively.
After that, they propose a BDD-based approach to verifying clone-enabled feature models, an approach that makes efficient use of the BDD (binary decision diagram) data structures, by considering the specific characteristics of feature models' verification. Experiments show that this BDD-based approach is more efficient and can verify more complex feature models than our previous method.

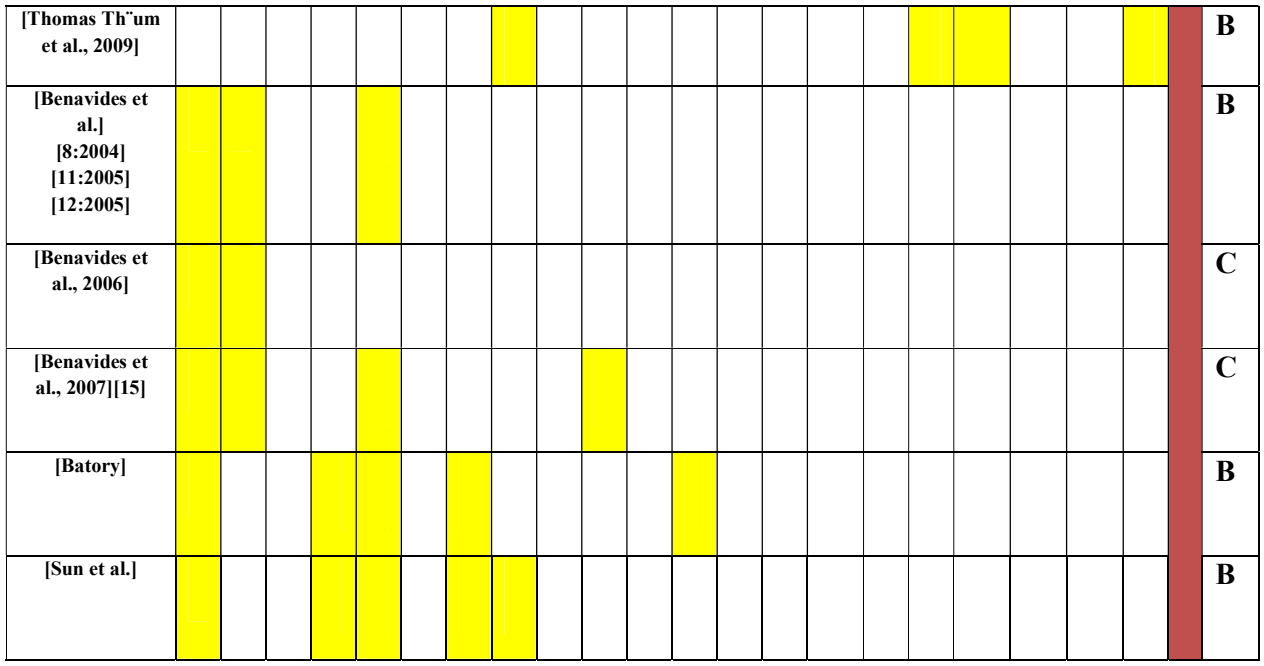
To examine the approach's efficiency and capability, the researchers apply it to verify two sets of designed feature models. One set contains 20 feature models only with binary constraints, and the number of features in them varies from 10, 20 to 90, and then from 100, 200, to 1000. The other set contains 20 feature models with both binary and composite constraints, and the number of features also varies from 10 to 1000.
The BDD-based approach can verify complex feature models with 500 features using 66.7 seconds, and verify simple feature models with 1000 features using 37.2 seconds.

- **[Zhang et al,. 2004]**
  The SVM system is a system "for checking finite systems against specification in temporal logic", however the proposal uses propositional logic. One of the main contributions of this work is the simplification operation. Moreover, a systematic way to detect dead features was provided as well.
  Next figure depicts example of simplified FM:



- **[Benavides, 2007]**
  FeAture Model Analyser (FAMA) provides engineering support for software product line analysts to automate the analysis of feature models. Furthermore, FAMA is extensible because it can be complemented with i) further models other than feature models, ii)other operations and iii) other solvers that will certainly appear in the future.
  It supports abstraction, basic and extended FM, and multiple solvers.

Operations / Papers	Void Feature Model	#products	Dead features	Valid product	All products	Equivalent FMs	Explanations	Refactoring	Optimization	Commonality	Filter	Valid partial configuration	Atomic sets	False optional features	Corrective Explanations	Dependency Analysis	Generalization	Arbitary edits	Conditional dead features	Multi-step configurations	Specialization		Feature Model Notation
[Gheyi et al., 2000]																							B
[Zhang et al,. 2004]																							B
[Mendonca et al., 2004]																							B
[Benavides, 2007] [FAMA]																							C
[Zhang et al,. 2008]																							C



Next table depicts Summary of the proposals for the analysis of basic feature models:

Papers / Operations	SAT-based [Mendonca et al., 2004]	BDD-based [Zhang et al,. 2008]	Algorithm reasoning [Thomas Th¨um et al., 2009]	Alloy Analyzer [Gheyi et al., 2000]	SVM [Zhang et al,. 2004]	FAMA [Benavides, 2007]
	CSP+PL		DL	PL		hybrid
Void feature model	✓	✓	✓	✓	✓	✓
#products	∼ +	✓ +	✓	∼	∼	✓
Dead features	–	✓	✓	–	✓ +	✓
Valid product	✓	✓	✓	✓	∼	✓
All products	✓	✓	✓	✓	∼	✓

Equivalent FMs	~		–		✓		✓		~	✓
Explanations	✓		✓		–		✓		–	~
Refactoring	–		✓		✓ +		✓ +		✓	✓
Optimization	–		✓		–		–		–	✓
Atomic sets	–		–		–		–			–
Corrective explanations	–		✓		–		–		–	~
Dependency analysis	✓		–		✓		–		–	–
Generalization	–		✓		✓		–		–	✓
Arbitary edit	–		✓		–		✓		–	–
Conditional dead features	–		✓		✓		✓		–	✓
Multi-step configuration	✓		✓		✓		–		–	–
Specialization	–		–		✓		–		–	–
Simplification									✓	

Papers / Operations	[Benavides et al.] [8:2004][11:2005] [12:2005]	[Benavides et al., 2006] [13]	[Benavides et al., 2006] [14]	[Benavides et al., 2007] [15]
	CP		**Multi**	
Void feature model	✓	✓	✓	✓
#products	✓	✓	✓	✓
Dead features	–	-	-	-

Valid product	-	-	-	-
All products	✓	-	-	✓
Commonality	○	–	-	✓
Refactoring	–	-	-	-
Optimization	○	-	–	-
Atomic sets	–	–	–	–
Corrective explanations	–	–	–	–
Dependency analysis	–	–	–	–
Generalization	–	–	–	–
Arbitary edit	–	–	–	–
Conditional dead features	–	–	–	–
Multi-step configuration	–	–	–	–
Specialization	–	–	–	–
Simplification	–	–	–	–

Where a (√) symbol means that the operation has been proposed, the (∼) symbol means that although the operation has not been proposed, we envisage that the operation can be performed, and a (-) symbol means that the operation has not been contemplated and we do not envisage a way to include it in the proposal, finally a (+) symbol means that the operation has been evaluated.

All these proposals are suitable at implementation level and can be considered of a way of performing some operations of analysis over feature models. Nevertheless, we are not aware as a way of using these solvers to analyse advanced feature models and this is the main drawback that we see in these proposals.

FAMA provides engineering support for software product line analysts to automate the analysis of feature models. Furthermore, FAMA is extensible in the sense that it can be complemented with i) further models other than feature models, ii)other operations and iii) other solvers that will certainly appear in the future.

1.6 Experimental Results [9]

For here we can put the results in one table designed like follows

Paper name	**Number of features**	**Number of constraints**	**time**	**Type of featuer model**	**Number of instances**	**anotations**
[Mendonca et al., 2004]	10000	0.1 to 3.5	30 sec	Simple, complex	52500	
[Zhang et al,. 2008]	10- 1000.		37.2 sec	Simple, complex	20	
[Gheyi et al., 2000]	300	-	2 hours			
[Thomas Th¨um et al., 2009]	10-10000	2-5	11 sec		2000	
[Benavides et al. 2005][11]	0-25	-	1.8 sec	basic	6	
[Benavides et al. 2005][12]	0-25		5	basic	3	
[Benavides et al. 2006][13]	15-52		10	basic	5	
[Benavides et al. 2006][14]	50-300	0-25%	5.3	basic	200	

We make a study about experiments done in this domain in order to evaluate the performance of automated analysis techniques.

These were done depending on randomly generated FMs (different numbers of features).The experiments focused on a performance comparison of three solvers (CSP, SAT and BDD) in order to test how these representations can influence in the automatic analysis of FMs. The comparison results were obtained from the execution of a number of FMs mapped as CSP, BDD and SAT.

It was used four groups of 50 randomly generated FMs. Each group included FMs with a number of features in an specific range ([50-100),[100-150),[150-200) and [200-300)) with a double aim: test the performance of small, medium and large instances and working out averages from the results in order to avoid as much exogenous interferences as possible. Each FM was executed several times increasing the number of cross–tree constraints from one until the 25% of the number of the features in the FM in order to find out how dependencies influence in the performance. The dependencies were added randomly as well, but cheking that the same feature can not appear in more than one cross– tree constraint and that a feature can not have a cross tree constraint with any of its ancestors. Averages were obtained from all the FMs in each range with the same percentage of cross–tree constraints. Table above summarizes the characteristics of the experiments.

The experimental comparison revealed some interesting results. The first evidence was that BDD is on average 96% faster than CSP and 75% faster than SAT finding one solution. However, BDD revealed a memory usage on average 928% higher than CSP and 1672% higher than SAT. On the other hand, although CSP and SAT showed a similar memory usage, SAT representation showed better results in both aspects, memory and especially in time.

In the range of (200-300) features, it was found some cases where the memory used by the solver was around 300 Mb. But with bigger FMs (e.g. 1000 features) this can be even a bigger problem. The results obtained from finding the total number of configurations of a given FM showed a great superiority of BDD. While CSP and SAT were computationally incapable of performing that operation in a reasonable time in most of the cases, BDD lasted 5312 ms to work out the 7.77 × 1034 solutions of the worst case.

Discussion this part we can keep it for later

The great superiority of BDD on finding the total number of solutions is because for calculating the number of solutions, in general, CSP and SAT solvers have to retrieve all the solutions (which is a #P-complete problem) meanwhile BDD solvers use

efficient graph algorithms to calculate the total number of solutions without the need of calculating all the solutions.

The huge memory usage of BDD solvers depends on the variable ordering for representing the BDD. As stated earlier, the size of BDDs can be reduced with a good variable ordering, however, calculating the best variable ordering is a NP-hard problem. To the best of our knowledge, there is only a proposal to include feature attributes in the automated analysis of feature models.

As a result of the test, there is not an optimum representation for all the possible operations that can be performed on FMs. Therefore, a better solution is a framework for the automated analysis of feature models. The framework will be designed to be open to other solvers where formal semantics of feature models will play a fundamental role.

Next table depicts summary of experimental results for No. of products operation:

	# products	speed	Memory
CSP	○	○	○
SAT	○	○	○
BDD	✓	✓	✓

Bibliography

[1] Reitz, John M. *Dictionary for Library and Information Science.* Library of Congress Cataloging in Publication Data, 2004.

[2] K. Pohl, G. B¨ockle, and F. van der Linden. Software Product Line Engineering: Foundations, Principles, and Techniques. Springer–Verlag, 2005.

[3] Marcilio Mendonca, Krzysztof Czarnecki, Andrzej Wa¸sowski. SAT-based Analysis of Feature Models is Easy.

[4] Rohit Gheyi, Tiago Massoni, Paulo Borba. A Theory for Feature Models in Alloy.

[5] Thomas Th¨um, Don Batory, Christian K¨astner, Reasoning about Edits to Feature Models, 2009.

[6] Wei Zhang, Hua Yan1, Haiyan Zhao1, and Zhi Jin. A BDD-Based Approach to Verifying Clone-Enabled Feature Models' Constraints and Customization, 2008..

[7] W. Zhang, H. Zhao, and H. Mei. A propositional logic-based method for verification of feature models. In J. Davies, editor, *ICFEM 2004*, volume 3308, pages 115–130. Springer–Verlag, 2004.

[8] D. Benavides, A. Ruiz-Cortés, and P. Trinidad. Coping with automatic reasoning on software product lines. In Proceedings of the 2nd Groningen Workshop on Software Variability Management, November 2004.

[9] D. Benavides. ON THE AUTOMATED ANALYSIS OF SOFTWARE PRODUCT LINES USING FEATURE MODELS. 2007.

[10] David Benavides, Sergio Segura, Pablo Trinidad, Antonio Ruiz-Cort´es, A first step towards a framework for the automated analysis of feature models