Configuration analysis for large scale feature models

Towards speculative-based solutions

Cristian Lorenzo Vidal Silva

Universidad de Sevilla

Tesis dirigida por:

Dr. José Á. Galindo and Dr. David Benavides

Contents

List of Figures

List of Tables

Acknowledgement

My parents always instilled in me that studying was the way to help my country. For many years, I worked to be who I am today by following that great advice. Thanks to my studies, I have grown in my thinking and goals about always assisting people; I have learned about how to form teams; and how we can also help build a better world through research works.

So far, life and my studies have allowed me to know other countries. I still remember when I was in the USA and wrote to David Benavides about my interest in being his Ph.D. student. That was just the beginning back in 2013. At the end of 2014, this path began. And many thanks to the University of Seville, for its professors, space, people, and support during these long years. I am very grateful to David and José since they taught me to study, learn, and think about new research opportunities and goals, and be patient to work for their achievement. Today, because of all the teamwork with David and José, I can say that I achieved the primary goal of delivering my Thesis with great pride.

During these years, I have had the opportunity to work as a professor and researcher at different universities, which have supported me in achieving my Ph.D. studies. I thank the Universidad de Playa Ancha, Valparaíso - Chile, which was the first to help me with time and money to be in Seville. My thanks also to the Universidad Autónoma de Chile, Talca - Chile, which also gave me the resources to be in Seville. And all thanks to the Universidad Católica del Norte, Antofagasta - Chile, my current work, which has given me all the necessary support to complete this great goal.

Many thanks to all people who assisted and supported me during these long years in the academy and daily life. I am grateful to my wife and son for giving me the time and space to accomplish this goal. They went along with me to Seville in 2019 and realized that Chile still lacks a lot, perhaps too much, to be a developed country. With research and education, we can help build that Chile, a better world for everyone.

Resumen

Los sistemas de alta variabilidad son sistemas de software en los que la gestión de la variabilidad es una actividad central. Algunos ejemplos actuales de sistemas de alta variabilidad son el sistema web de gesión de contenidos Drupal, el núcleo de Linux, y las distribuciones Debian de Linux.

La configuración en sistemas de alta variabilidad es la selección de opciones de configuración según sus restricciones de configuración y los requerimientos de usuario. Los modelos de características son un estándar "de facto" para modelar las funcionalidades comunes y variables de sistemas de alta variabilidad. No obstante, el elevado número de componentes y configuraciones que un modelo de características puede contener hacen que el análisis manual de estos modelos sea una tarea muy costosa y propensa a errores. Así nace el análisis automatizado de modelos de características con mecanismos y herramientas asistidas por computadora para extraer información de estos modelos. Las soluciones tradicionales de análisis automatizado de modelos de características siguen un enfoque de computación secuencial para utilizar una unidad central de procesamiento y memoria. Estas soluciones son adecuadas para trabajar con sistemas de baja escala. Sin embargo, dichas soluciones demandan altos costos de computación para trabajar con sistemas de gran escala y alta variabilidad. Aunque existan recusos informáticos para mejorar el rendimiento de soluciones de computación, todas las soluciones con un enfoque de computación secuencial necesitan ser adaptadas para el uso eficiente de estos recursos y optimizar su rendimiento computacional. Ejemplos de estos recursos son la tecnología de múltiples núcleos para computación paralela y la tecnología de red para computación distribuida.

Esta tesis explora la adaptación y escalabilidad de soluciones para el an+alisis automatizado de modelos de características de gran escala. En primer lugar, nosotros presentamos el uso de programación especulativa para la paralelización de soluciones. Además, nosotros apreciamos un problema de configuración desde otra perspectiva, para su solución mediante la adaptación y aplicación de una solución no tradicional. Más tarde, nosotros validamos la escalabilidad y mejoras de rendimiento computacional de estas soluciones para el análisis automatizado de modelos de características de gran escala.

Concretamente, las principales contribuciones de esta tesis son:

- **Programación especulativa para la detección de un conflicto mínimo y**

preferente. Los algoritmos de detección de conflictos mínimos determinan el conjunto mínimo de restricciones en conflicto que son responsables de comportamiento defectuoso en el modelo en análisis. Nosotros proponemos una solución para, mediante programación especulativa, ejecutar en paralelo y reducir el tiempo de ejecución de operaciones de alto costo computacional que determinan el flujo de acción en la detección de conflicto mínimo y preferente en modelos de características de gran escala.

- **Programación especulativa para un diagnóstico mínimo y preferente**. Los algoritmos de diagnóstico mínimo determinan un conjunto mínimo de restricciones que, por una adecuada adaptación de su estado, permiten conseguir un modelo consistente o libre de conflictos. Este trabajo presenta una solución para el diagnóstico mínimo y preferente en modelos de características de gran escala mediante la ejecución especulativa y paralela de operaciones de alto costo computacional que determinan el flujo de acción, y entonces disminuir el tiempo de ejecución de la solución.

- **Completar de forma mínima y preferente una configuración de modelo por diagnóstico**. Las soluciones para completar una configuración parcial determinan un conjunto no necesariamente mínimo ni preferente de opciones para obtener una completa configuración. Esta tesis soluciona el completar de forma mínima y preferente una configuración de modelo mediante técnicas previamente usadas en contexto de diagnóstico de modelos de características.

Esta tesis evalua que todas sus soluciones preservan los valores de salida esperados, y también presentan mejoras de rendimiento en el análisis automatizado de modelos de características con modelos de gran escala en las operaciones descritas.

Abstract

Variability-intensive systems are software systems in which variability management is a core activity. Some current examples of variability-intensive systems are the web content management system Drupal, the Linux kernel, and the Linux Debian distributions.

The configuration of variability-intensive systems is selecting options regarding their configuration restrictions and user requirements. Feature models are a "de facto" standard for modeling the common and variable functionalities of variability-intensive systems. However, the large number of components and configurations that a feature model can encode make the manual analysis of those models an error prone and costly task. The automated analysis of feature models then appeared with mechanisms and computer-aided tools to extract useful information from feature models. Traditional solutions of automated analysis of feature models follow a sequential computing approach to use a central processing unit and memory. These solutions are adequate to work with low-scale systems. However, these solutions require high computing costs to work with large-scale and variability-intensive systems. Although computing resources exist to improve the performance of computing solutions, all the solutions with a sequential computing approach need to be adapted to efficiently use these resources and optimize their computing performance. Examples of these resources are multi-core technology for parallel computing and network technology for distributed computing.

This thesis explores the adaptation and scalability of solutions for the automated analysis of large-scale feature models. First, we present the use of speculative programming for the parallelization of solutions. Furthermore, we appreciate a configuration problem from another perspective to solve it by adapting and applying a non-traditional solution. Later, we validate the scalability and computing performance improvements of these solutions for the automated analysis of large-scale feature models.

Specifically, the main contributions of this thesis are:

- **Speculative programming for a minimal and preferred conflict detection**. Minimal conflict detection algorithms determine the minimal conflict set or minimal set of constraints in conflict responsible for faulty behavior in the model under analysis. We propose a solution to execute in parallel and reduce

the execution time of path operations through speculative programming in the minimal and preferred conflict detection of large-scale models.

- **Speculative programming for a minimal and preferred diagnosis**. Minimal diagnosis algorithms determine a minimal set of constraints that, for adequately adapting their state, permit getting a conflict-free or consistent model. This work presents a solution for the minimum and preferred diagnosis of large-scale models by the speculative and parallel execution of high-computing cost operations that determine the action flow and then reduce the solution execution time.

- **Minimal completion of preferred configuration by diagnosis**. Minimal completion of configuration solutions determine a set of options that, for their adequate setting, convert the current configuration of the model into a complete product with preferred options. This thesis appreciates the completion of a configuration as a diagnosis task and uses as a solution an adapted minimal and preferred diagnosis algorithm.

This thesis evaluates that all its solutions preserve the expected output value and present performance improvements for the described operations in the automated analysis of large-scale feature models.

Part I

Preface

Chapter 1

Introduction

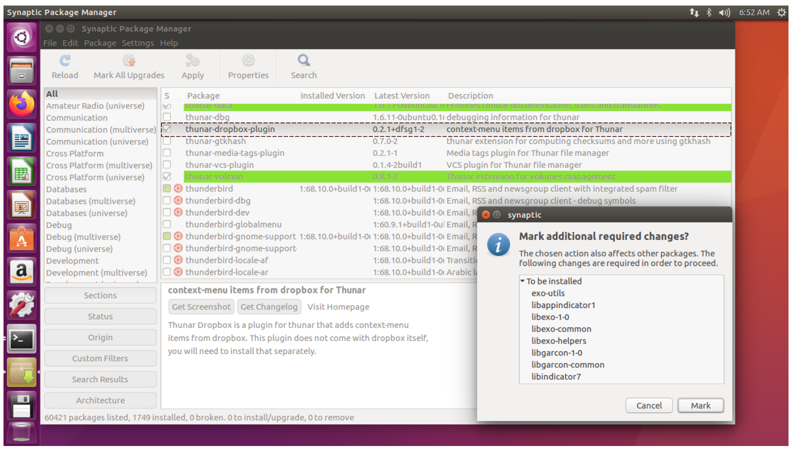
Abstract

This dissertation reports our work in the configuration analysis for large scale feature models towards speculative-based solutions. This chapter gives an overview of the contributions, the research method, and publications related to this document.

1.1 Research context

Product configuration is the activity of designing a product according to a set of requirements and configuration rules. With further details, product configuration systems need the knowledge base regarding the set of components and combination rules and the customers' requirements for selecting product components (configuration) that match their preferences [1]. A valid product configuration only depends on the selected features in a consistent knowledge base scenario; that is, each valid product results from the composition of component type instances that respect the set of defined combination rules [2]. Figure 1.1 shows a motivating scenario of product configuration for updating the set of installed packages in the Ubuntu Xenial operating system [3]. In this example, all the packages for installing and their dependencies represent the consistent knowledge base, and already installed and selected for installing packages correspond to the desired product requirements. Hence, after selecting a package for installation, such as a dropbox plugin, that package requires installing additional packages to respect dependency rules.

Product configuration systems require systematically managing all the features and their composition rules to analyze each desired product configuration's features selection. In software engineering, Variability Models (VMs) permit describing the different relationships and configuration options for the variability management of software systems. Different VMs exist, such as Feature Models (FMs) and Orthogonal Variability Models (OVMs). FMs permit representing functional commonalities and variabilities of software systems [4], whereas OVMs permit describing the variant parts of the base model of systems [5]. Kang et al. [6] introduced FMs as part of



S	Package	Installed Version	Latest Version	Description
✓	thunar-data		1.6.11-0ubuntu0.1	Provides thunar documentation, icons and translations
	thunar-dbg		1.6.11-0ubuntu0.1	debugging information for thunar
✓	thunar-dropbox-plugin		0.2.1+dfsg1-2	context-menu items from dropbox for Thunar
	thunar-gtkhash		0.7.0-2	thunar extension for computing checksums and more using gtkhash
	thunar-media-tags-plugin		0.2.1-1	Media tags plugin for Thunar file manager
	thunar-vcs-plugin		0.1.4-2build1	VCS plugin for Thunar file manager
✓	thunar-volman		0.8.1-2	Thunar extension for volumes management
	thunderbird	1:68.10.0+build1-0	1:68.10.0+build1-0	Email, RSS and newsgroup client with integrated spam filter
	thunderbird-dbg		1:68.10.0+build1-0	Email, RSS and newsgroup client - debug symbols
	thunderbird-dev		1:68.10.0+build1-0	Email, R
	thunderbird-globalmenu		1:60.9.1+build1-0ul	Email, R
	thunderbird-gnome-support	1:68.10.0+build1-0	1:68.10.0+build1-0	Email, R
	thunderbird-gnome-support		1:68.10.0+build1-0	Email, R
	thunderbird-locale-af		1:68.10.0+build1-0	Transiti
	thunderbird-locale-ar		1:68.10.0+build1-0	Arabic l

context-menu items from dropbox for Thunar

Thunar Dropbox is a plugin for thunar that adds context-menu items from dropbox. This plugin does not come with dropbox itself, you will need to install that separately.

60421 packages listed, 1749 installed, 0 broken. 0 to install/upgrade, 0 to remove

Mark additional required changes?

The chosen action also affects other packages. The following changes are required in order to proceed.

- To be installed
  - exo-utils
  - libappindicator1
  - libexo-1-0
  - libexo-common
  - libexo-helpers
  - libgarcon-1-0
  - libgarcon-common
  - libindicator7

Figure 1.1: Example of the packages configuration in the Ubuntu Xenial OS.

the FODA (Feature-Oriented Domain Analysis) method, and they become the most used VM in the SPL community afterward.

Product Line An FM defines a set of features and their relationships for defining valid feature combinations or products, that is, sets of features that respect the FM's defined relationships. Such as Apel et al. [7] remark, FM permits representing all the products of an SPL. An FM organizes in a tree-like structure that starts at the root feature that identifies the SPL and from which tree branches of features emerge. We can then define a product in terms of the set of features that compose it, and each feature describes an increment of functionality in the products containing that feature [8]. An FM supports binary and set relationships between parent and child features and cross-tree relationships to symbolize dependencies between features. Figure 1.2 shows an FM for describing an operating system SPL and a valid configuration of it (the grey-colored features). Chapter 2 details and exemplifies common FM notations. Different tools exist for the FMs representation and support such as SPLOT [9], FeatureIDE [10], FaMa framework [11] and FAMILIAR [12]. We presents solutions for extending the FaMa framework.

Product configuration permits assisting the mass customization production [2]. Variability-Intensive Systems (VIS) follow mass customization in software engineering by addressing variability in the software development process phases. Because VIS users expect that software products can adapt to their needs, the management of the users' requirements variability in VIS represents a crucial activity for those systems [13]. Research works concerning managing the variability of VIS already exist in the literature, such as in the Linux operating system [14, 15], with the Debian-based distributions of Linux [16], with the Android mobile system [17], and with the content management framework Drupal [18] to just mention a few. Those works use variability models for representing and analyzing VIS.

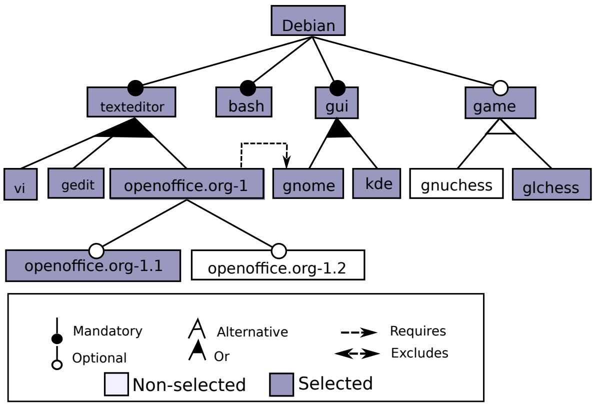


Figure 1.2: Feature model and configuration example.

Software Product Line (SPL) is a case of VIS that systematically manages commonalities and variabilities for the configuration of software products [7]. SPL defines domain engineering to analyze and develop reusable common and variable functionalities (features) in the products' domain, and application engineering to produce customized products regarding the users' features selection. Defining valid configuration in SPL is a complex task for the growing complexity of the configuration knowledge base [17]. When the users' feature selection conflicts with consistent configuration knowledge bases, it is necessary to identify those issues and solve them. The manual analysis of variability models, such as FMs, are error-prone and time-consuming tasks mainly for the increasing size of those models. For example, variability and configuration models for Debian-based distributions describe around 28000 variability points [16], and manually analyzing those models without mistakes is impractical. Mechanisms for the Automated Analysis of Feature Models (AAFM) [4] are a solution to face those issues.

1.1.1 Main issues to solve

FMs permit organizing the configuration space to facilitate the construction of software variants by describing configuration options using interdependent features or functionalities. A set of features in an FM is called a configuration, and each software variant in an FM identifies a valid configuration or product [19]. Hence, AAFM operations for assisting the obtention of conflict-free FMs and configurations are high-value tasks. Nonetheless, existing AUTOMATED ANALYSIS OF FEATURE MODEL operations usually follow a sequential computing approach and cannot scale to work on large-scale and high-variability models. Various algorithms and solutions applicable for the AAFM exist in the literature, such as QUICKXPLAIN [20] and FASTDIAG [21] to detect a minimal conflict set and a minimal-preferred diagnosis in a set of constraints in conflict, respectively, and solutions for the completion of partial products.

Even though QUICKXPLAIN and FASTDIAG are computationally efficient in theory, those algorithms are examples of solutions that take a long time to work on large-scale FM configurations. Both solutions cannot use additional computing resources for their sequential computing nature, such as multiple core or network technologies for parallel and distributed computing, respectively. Concerning the completion of products, defining and accomplishing all users' requirements for configuring large-scale systems is a tedious and complex task, and non-efficient solutions exist for assisting in the completion of partial configurations. The next lines give more details of the three problems that our dissertation faces.

- Minimal conflict set. For a consistent FM that we can define as a set of constraints, a non-consistent configuration violates those constraints. The features model of Figure 1.2 exemplifies a consistent configuration; that is, this configuration does not violate the FM constraints. For that FM and configuration, if feature $gnuchess$ were also selected, the resulting configuration would be non-consistent because a conflict exists between features $gnuchess$ and $glchess$. Chapter 2 give more details concerning FM and its constraints. In this example, $\{gnuchess, glchess\}$ is a Minimal Conflict Set (MCS) because we cannot find a subset of it that results being a conflict set. Such as [21] remark, we can solve an MCS by merely deleting one of its constraints. After finding and solving all the MCS instances, the configuration results valid (conflicts-free).

  The QUICKXPLAIN algorithm permits efficiently finding preferred MCS regarding the order of the constraints definition. The functioning of QUICKXPLAIN uses the consistency check over constraint sets, a costly action, as a primary step to achieve its main purpose, that is, to identify a preferred MCS. The application of QUICKXPLAIN for the conflict analysis of large-scale FMs such as the Android mobile operating system [17], the Linux kernel [14], and distributions of Linux like Debian [16] are examples of computationally expensive tasks mainly for the sequential nature of QUICKXPLAIN and the high demand for computing resources such as the execution time and memory space to work with large-scale models. Section 3.2 gives more details about QUICKXPLAIN and its applicability for the FM product configuration analysis.

- Minimal diagnosis. Given the set of constraints of a consistent FM and a non-consistent configuration that violates the FM constraints, a diagnosis is the set of constraints that permit getting a consistent configuration after removing those constraints. For the consistent configuration of Figure 1.2 after selecting the feature $gnuchess$, we know that $\{gnuchess, glchess\}$ is a Minimal Conflict Set (MCS). Then, either deleting $gnuchess$ or $glchess$, we can obtain a valid configuration (conflict- free); that is, $gnuchess$ and $glchess$ are examples of diagnosis.

  The FASTDIAG algorithm permits efficiently finding a preferred-minimal diagnosis regarding the order of the constraints definition. The functioning of FASTDIAG uses the consistency check over constraint sets, a costly action, as a primary step to achieve its main purpose, that is, to identify a preferred and minimal diagnosis. The application of this algorithm for the diagnosis analysis of large-scale FMs such as the Android mobile operating system [17], the Linux kernel [14], and distributions of Linux like Debian [16] result in computationally expensive tasks mainly for the sequential nature of FASTDIAG and the high demand for computing resources such as the execution time and memory space to work with large-scale models. Section 3.3 gives more details about FASTDIAG and its applicability for the FM product configuration analysis.

- Minimal completion of products. Achieving valid configurations in large-scale FMs is a time-demanding and complex task. Currently, reasoning tools that are usually part of AAFM process permit solving that issue. Section 3.4 gives more details about the minimal completion of products for the FM product configuration analysis.

Our thesis addresses the high computation cost of existing solutions to work on large-scale models. First, this thesis presents solutions to parallelize and reduce the execution time of path operations through speculative programming on large-scale and high variability models. Second, this thesis appreciates a problem from a different perspective by the adaptation of a non-traditional solution to reduce the execution time on large-scale models concerning traditional solutions. To validate our solutions, we applied them in the AAFM of large-scale FM and configuration instances.

1.2 Contributions

In this section, we summarize the main contributions of our research work. First, we describe our main contributions. Second, we report articles that have been published in relevant journals, conferences and workshops. Third, we describe the current availability of the developed tools.

1.2.1 Summary of contributions

This dissertation aims to provide efficient solutions in the FaMa framework to work on large-scale FM product configurations for MCS detection, minimal diagnosis, and product completion tasks. The next lines describe these contributions:

- PARALLELQUICKXPLAIN. A parallel algorithm for the minimal conflict set detection of FM configurations by using speculative programming.

  Conflict detection is applicable in many scenarios ranging from interactive decision-making to faulty hardware components or models. In these scenarios, the efficient identification of conflicts is crucial. Junker's QUICKXPLAIN [20] is a divide-and-conquer based algorithm for the determination of preferred MCS. Motivated by the increasing size and complexity of knowledge bases, we propose a speculative version of QUICKXPLAIN that improves runtime performance significantly, especially in complex knowledge bases. We propose PARALLELQUICKXPLAIN, a speculative programming version of QUICKXPLAIN, to pre-calculate in parallel consistency checkings needed by the base algorithm QUICKXPLAIN. We tested our solution with large-scale FM product configurations to validate our solution's performance improvement concerning its base solution QUICKXPLAIN.

- PARALLELFASTDIAG. A parallel algorithm for the diagnosis of FM configurations in conflict by using speculative programming.

  FASTDIAG [22] is an efficient divide-and-conquer diagnosis solution applicable in AAFM, but it does not scale for diagnosis on large-scale FMs and configurations. We propose PARALLELFASTDIAG, a speculative programming version of FASTDIAG, to pre-calculate in parallel consistency checkings needed by the base algorithm FASTDIAG.

- MINIMAL PREFERRED COMPLETION BY DIAGNOSIS. A minimal preferred completion of products by using a diagnosis task.

  The minimal preferred completion of configurations might represent an expensive computing task. Existing solutions, such as modern constraint satisfaction solvers, usually perform a complete search approach, making them unsuitable on large-scale configurations. We propose defining the completion of configuration like a diagnosis task to solving it by applying the FASTDIAG algorithm [21, 22]. FASTDIAG is an efficient solution for minimal diagnosis (updates) in the analyzed configuration. We evaluate our proposed method to complete partial configurations of random FMs and random partial products of an FM of an adapted version of the Ubuntu Xenial operating system. Our experimental analysis shows remarkable improvements in our solution regarding classical reasoner-based approaches for the same tasks.

PARALLELQUICKXPLAIN, PARALLELFASTDIAG, and MINIMAL PREFERRED COMPLETION BY DIAGNOSIS are practical tasks to any satisfiability, diagnosis, and completion of product problems in reasoner tools with automated support.

1.2.2 Publications in chronological order

Cristian Vidal contacted David Benavides in 2013 when he was finishing a Master of Science in Computer Science at Michigan State University, MI, USA. The research focus of Cristian Vidal at that time was modularization and Aspect-Oriented Software Engineering (AOSE). Davide Benavides gave advice regarding the Feature-Oriented Software Engineering (FOSE), his research area, and Cristian Vidal started being a Ph.D. student at the University of Seville at the end of 2014. Cristian Vidal, David Benavides and José A. Galindo defined a primary research focus that was to evaluate pros and cons of a mixing of programming AOSE and FOSE tools for developing modular programming solutions. After the first meeting in Seville during 2015, our primary research focus changed to applying Big Data technology such as Hadoop and Giraph for the AAFM. During 2017, after understanding that parallel computing is a base element of Big Data technology and reviewing existing solutions for the AAFM, we focus on developing solutions with parallel computation for conflict detection and diagnosis, and a diagnosis solution for the product completion. Those solutions represent the main contributions of this dissertation. Next, we present a complete list of the publications derived from our research work in chronological order.

[**2015**]. During our first year of work, we focus on proposing and developing a symbiosis of Feature-Oriented Programming (FOP) and the Aspect-Oriented Programming (AOP) methodology Join-Point Interface [23] because our motivation was applying modularization advantages of AOP with FOP. After out first meetings in Seville, we focused on Big Data solutions for the AAFM process.

- **JISBD'15**. **Cristian Vidal**, David Benavides, José A. Galindo, and Paul Leger: Exploring the Synergies between Join Point Interfaces and Feature-Oriented Programming. XX Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2015). Santander, Cantabria, Spain.
- **SCCC'15**. **Cristian Vidal**, José A. Galindo, Rodolfo Villarroel, David Benavides, Paul Leger, and Sebastián Valenzuela: JPI feature models - Exploring a JPI and FOP symbiosis for software modeling. 34th International Conference of the Chilean Computer Science Society (SCCC 2015), pages 1–6. Santiago, Chile.
- **BICT'15**. **Cristian Vidal**, David Benavides, Paul Leger, José A. Galindo, and Hiroaki Fukuda: Mixing of Join Point Interfaces and Feature-Oriented Programming for Modular Software Product Line. 9th EAI International Conference on Bio-inspired Information and Communications Technologies (BICT 2015), pages 433–437. New York City, United States.

[**2016**]. During this year, a journal published one of our papers presented in 2015. We focused on developing new solutions or adapting existing ones to analyze large-scale FMs applying Big Data computing approaches. We specialized on Hadoop and Giraph technologies and presented one of these works at the Software Product Line Conference (SPLC). Cristian Vidal also taught about Hadoop and Giraph in Chile.

- **SPLC'16**. José A. Galindo, Mathieu Acher, Juan Manuel Tirado, **Cristian Vidal**, Benoit Baudry, and David Benavides: Exploiting the enumeration of all feature model configurations: a new perspective with distributed computing. 20th International Systems and Software Product Line Conference (SPLC 2016), pages 74–78. Beijing, China.

- **EAI'16**. **Cristian Vidal**, David Benavides, Paul Leger, José A. Galindo, and Hiroaki Fukuda: Mixing of Join Point Interfaces and Feature-Oriented Programming for Modular Software Product Line. Endorsed Transaction on Scalable Information Systems, Volume 3, Number 10 (EAI 2016).
- **CIT'16**. Sebastián Valenzuela, **Cristian Vidal**, Jenny Morales, and Leopoldo López: Ejemplos de Aplicabilidad de Giraph y Hadoop para el Procesamiento de Grandes Grafos. Revista Información Tecnológica, Vol. 27 (5), September-October 2016. Universidad de La Serena, Chile. SCOPUS.

[**2017**].In this year, we obtained a publication that ended by guiding this Ph.D. dissertation. Concretely, we presented in a doctoral symposium our research goal and proposals of parallel AAFM solutions and highlighted their potential advantages. We also planned the development of those solutions.

- **SPLC'17**. **Cristian Vidal**: Exploring efficient analysis alternatives on feature models. 21th International Systems and Software Product Line Conference (SPLC 2017), volume B, pages 150–155. Seville, Spain.

[**2018**]. During this year, we continue working with the application of parallel computing for the efficient conflict detection and diagnosis of large-scale FMs. Cristian Vidal presented papers regarding his previous knowledge about Big Data and Feature Model solutions.

- **CIT'18**. **Cristian Vidal**, Miguel Bustamante, José Rubio, Luis Carter: Propuesta de Modelo de Características con Interfaz de Punto de Unión para el Modelamiento de Líneas de Productos de Software. Journal Información Tecnológica, Vol. 29 (6), November-December 2018. University of La Serena, Chile. SCOPUS.

[**2019**]. At the beginning of this year, Cristian Vidal presented a paper about applying FASTDIAG for the diagnosis for the product configuration in the AAFM process. Cristian Vidal also presented an article about the feasibility of teaching Big Data in the Chilean academy. In June of 2019, Cristian Vidal obtained a Banco Santander scholarship to be in Seville, Spain, from 23th of October of 2019 to 25 of March of 2020. Working as a team, and after Alexander Felfernig visited us at the University of Seville, Spain, in November of 2019, we matured ideas of parallelizing AAFM solutions. We started then talking about speculative programming for developing efficient AAFM solutions. The first version of PARALLELFASTDIAG was born at that time.

- **IJACSA'19**. **Cristian Vidal**: Reviewing Diagnosis Solutions for Valid Product Configurations in the Automated Analysis of Feature Models. International Journal of Advanced Computer Science and Applications (IJACSA), Volume 10 Issue 1, January 2019. SCOPUS.

[**2020**]. For the high domain level in the FASTDIAG algorithm, we developed a solution for the completion of FM product configurations. We wrote a paper about that solution and sent it to an international conference. Because FASTDIAG uses a divide-and-conquer approach like QUICKXPLAIN, we developed, tested, and validated PARALLELQUICKXPLAIN using a speculative programming approach. We wrote a research paper about PARALLELQUICKXPLAIN and sent it to the same international conference. This paper resulted in being one of the best contributions to the conference and selected for publication in the next year by a JCR journal. We also wrote an article about a test suite approach for minimal conflict and diagnosis of FM configurations. We sent that paper and presented it to another international conference. During this year, almost all the conferences were online.

- **ISMIS'20**. **Cristian Vidal-Silva**, Jesús Giraldez, José A. Galindo, and David Benavides: Automated completion of partial configurations as a diagnosis task - Using FASTDIAG to improve performance. 25th International Symposium on Methodologies for Intelligent Systems (ISMIS 2020) - Industry Session. Graz, Austria.

- **ISMIS'20**. **Cristian Vidal-Silva**, Alexander Felfernig, José A. Galindo, Müslüm Atas, and David Benavides: A Parallelized Variant of Junker's QUICKXPLAIN Algorithm. 25th International Symposium on Methodologies for Intelligent Systems (ISMIS 2020). Graz, Austria.

- **CONFWS'20**. **Cristian Vidal-Silva**, José A. Galindo, and David Benavides: Functional Testing of Conflict Detection and Diagnosis Tools in Feature Model Configuration: A Test Suite. 22th International Workshop on Configuration 2020 (CONFWS 2020). Vicenza, Italy.

- **JIIS'21**. **Cristian Vidal-Silva**, Alexander Felfernig, José A. Galindo, Müslüm Atas, and David Benavides: Explanations for Over-Constrained Problems with Parallelized QUICKXPLAIN. Journal of Intelligent Information Systems (JIIS). Springer-Verlag. WOS JCR Q3. Under review.

1.2.3 Tools

We developed three different tools during this Ph.D. First, we developed PARALLELQUICKXPLAIN[*], an speculative computation solution for the preferred minimal conflict detection in a set of constraints in conflict. We specialized PARALLELQUICKXPLAIN for the conflict detection of FM product configuration.Second, we developed PARALLELFASTDIAG[†], a tool for the preferred minimal diagnosis by applying speculative computation in a set of constraints in conflict. We specialized PARALLELFASTDIAG for the conflict detection of FM product configuration. Third, we developed BOLON[‡], a solution for the completion of product configuration of

[*]`https://github.com/cvidalmsu/A-Python-QX-implementation`
[†]`https://github.com/cvidalmsu/A-Python-FD-implementation`
[‡]`https://github.com/cvidalmsu/BOLON-FaMaProdConf-TestSuite`

feature models by applying a diagnosis solution.

1.2.4 Research internships and collaborations

This research work includes the collaboration of colleges from different countries. Table 1.1 shows the affiliation, name of each co-author, and co-authored articles. Figure 1.3 illustrate the research time and collaboration of colleagues during these Ph.D. studies.

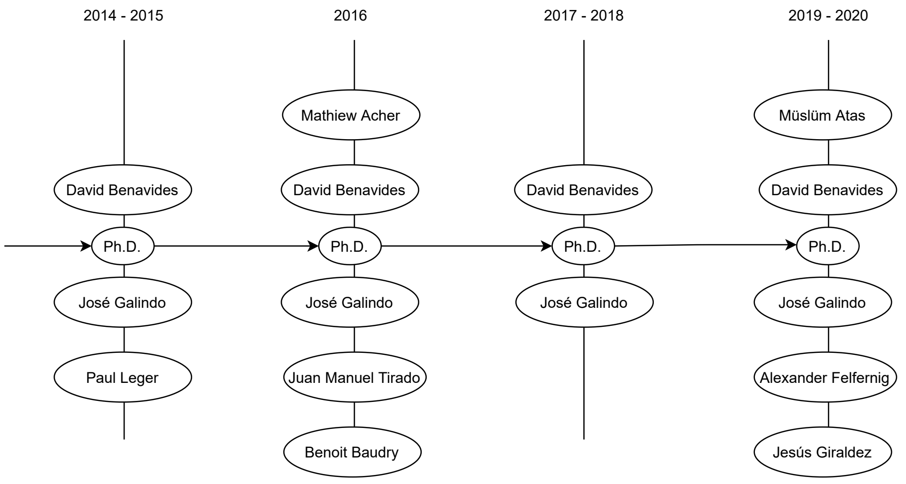


Figure 1.3: Years of research work and collaboration.

	Name	Affiliation	Paper
1.	Mathieu Acher	University of Rennes 1 and Inria, France	[24]
2.	Müslü Atas	Graz University of Technology, Austria	[25]
3.	Benoit Baudry	Inria, France	[24]
4.	David Benavides	Universidad de Sevilla, Spain	[26] [27] [28] [29] [?] [30] [25] [31] [32]
5.	Alexander Felfernig	Graz University of Technology, Austria	[25]
6.	José Á. Galindo	Universidad de Sevilla, Spain	[26] [27] [28] [29] [?] [30] [25] [31] [32]
7.	Jesús Giráldez Cru	Universidad de Granada, Spain	[31]
8.	Paul Leger	Universidad Católica del Norte, Chile	[26] [27] [29]
9.	Juan Manuel Tirado	Cambridge Computer Lab, United Kingdom	[24]

Table 1.1: List of researchers and institutions the student co-authored a work.

1.3 Structure of this dissertation

This document is organized as follows:

Part I: Preface. In the first part of this dissertation, we present the main contributions of this research as well as the background and motivating scenarios that pushed forward this work.

Part II. Background and Motivation. In the second part of the dissertation, we explore and update the basic background information required to understand the objectives of this thesis work. Chapter 2 describes variability and configuration models, Chapter 3 addresses the automated analysis of feature and product configuration models, and Chapter 4 summarises the main properties of existing research work concerning the AAFM operations of this thesis.

Part III. Contributions. This part constitutes the core contributions of our thesis that consists of three chapters organized as follows. Chapter 5 focuses in PARALLELQUICKXPLAIN, Chapter 6 presents PARALLELFASTDIAG, and Chapter 7 details our solution for the completion of products by diagnosis task. In each chapter, we motivate our research work, describe the resulting solution, and validate its functioning regarding its base one.

Part IV. Final Remarks. In this part, Chapter 8 shows the conclusions of this thesis and propose future work to address new and challenging research problems which arise from the contributions made in this dissertation.

Part V. Appendix. In Appendix A, we present PAVIA, a Big Data solution for the AAFM that we presented in SPLC 2016. In Appendix B, we describe a functional testing of a AAFM operations for conflict detection and diagnosis.

Part II

Background and Motivation

Chapter 2

Variability models

Abstract

A Variability Model (VM) permits describing common and variable components of a system family and relationships and constraints among those components for product configuration. Feature Models (FM) is an example of VM in Software Product Lines (SPLs). This chapter defines and exemplifies VMs using FMs as a standard notation for VM.

2.1 Introduction

A Variability Model (VM) represents an organized set of component types, relationships, and constraints for setting configurations (solutions) in a system family. A configuration is a particular case of design activity for defining products from the composition of instances of a well-defined and fixed set of component types concerning a group of relationships, constraints and requirements that restrict to gather some features [**?**]. This process typically relies on the product domain and problem-solving knowledge; that is, product configuration is a knowledge-based process. A configuration is an instance of a VM like an object is an instance of a class [33]. A feature model is an example of variability model. Figure 2.1 exemplifies a FM for the installation of the Debian operating system with the following constraints:

- A feature *Debian* must be composed by an instance of the feature *texteditor*, *bash* or *gui*. Moreover, an instance of the feature *Debian* can be composed by an instance of the feature *game*.
- An instance of the feature *texteditor* must be composed by at least one instance of the features *vi*, *gedit*, *openoffice.org*.
- An instance of the feature *openoffice.org-1* can be composed by one instance of the the fdeatures *openoffice.org-1.1*, *openoffice.org-1.2*, both or none of them.
- An instance of the feature *openoffice.org-1* requires an instance of the feature *gnome*.

- An instance of the feature *gui* must be composed of an instance of the features *kde*, *gnome* or both.
- An instance of the feature *game* must be composed by only one instance of the features *glchess* or *gnuchess*.

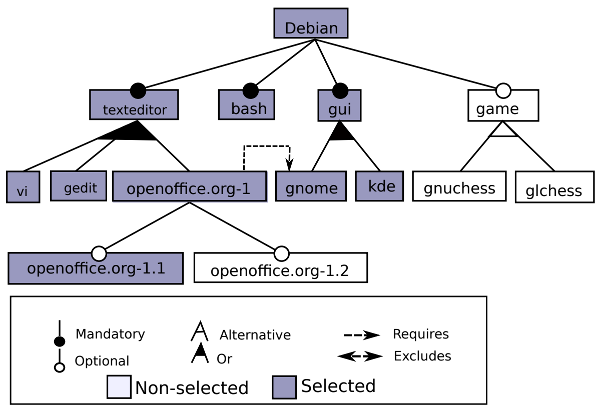


Figure 2.1: Feature model of a Debian derivative example with a valid configuration (gray features).

We can define a configuration for a VM by selecting instances of component types that respect all the restrictions and model rules. In our example, the FM constraints. Gray features of Figure 2.1 shows a valid configuration of the FM.

Feature models are information models that permit representing the variant flexibility and maintainability for systems' variability and configuration [34]. Next section describes and exemplifies different types of FMs and their use for the VM representation.

2.2 Feature models

Feature Model (FM) is a tree-like structure commonly used to represent common and variable functionalities (features) and their relationships for the configuration of products in a Software Product Line (SPL) [6]. A feature is an abstraction of a prominent or distinctive user-visible aspect, requirement, quality, or functional characteristic of a family of software systems [4, 35, 36], that is, each feature constitutes a user-visible configuration option of the problem domain [37]. Kang et al. [6] introduced FMs in the FODA (Feature-Oriented Domain Analysis) method, and they are the "de facto" standard for describing common and variable features in system families [38, 39] regardless their size because FMs facilitate the software reuse [40].

An FM organizes features and their relationships in a tree-like structure that starts with the root feature. Each successively deeper level in the FM corresponds to a more fine-grained configuration option for product-line variants. Features are nodes of that tree, and their relationships are the edges (relationships and constraints) between features [41]. The relationships among features are of two types: structural relationships between a parent and its child features, and cross-tree or cross-hierarchy constraints [2, 4].

FMs represent an effective communication medium between customers and developers of SPLs [42]. Such as the work of Benavides et al. describe [4], different FM dialects exist nowadays such as basic FMs models [8**?**], cardinality based FMs [43, 44] and extended FMs using feature attributes [45, 46] that next subsections describe.

2.2.1 Basic feature models

A basic FM support two types of relationships between features: structural relationships between parents and their child features, and cross-tree constraints [4] [2]. Thus, each non-root feature has a parent feature and is either part of a group or not [47]. Next lines describe each type of FM relationships.

- Structural relationships between parents and their child features:
  - Mandatory: A mandatory relationship states that a parent feature requires its child. The top-left figure of Table 2.1 shows the graphic representation of a mandatory relationship between a parent feature and a child feature.
  - Optional: An optional relationship states that a child feature may be or not present (it is does not required by its parent feature). The top-right figure of Table 2.1 illustrates an optional relationship between a parent feature and a child feature.
  - Set: A defined number of features of a set of children features (sub-features) are selectable for products when their parent is selected. This number of features is given by a cardinality relation [x, y], for x <= y and y <= number of child features in the set. Two cases are XOR (alternative) and Or (inclusive) sets.
    - ∗ Inclusive Or: At least one child features must be present. The cardinality relation is [1, n] in this case (n corresponds to the number of child features). The middle-left figure of Table 2.1 illustrates an inclusive relationship between a parent feature and a set of children features.
    - ∗ Alternative XOR: Only one child feature must be present. The associated cardinality relation is [1, 1] in this case. The middle-right figure of Table 2.1 illustrates an alternative relationship between a parent feature and a set of children features.
- Cross-Tree Constraints.

– Requires: For two features A and B, if A requires B then the presence of A implies the presence of B in a product. The bottom-left figure of Table 2.1 illustrates a requires cross-tree constraint relationship between a source feature A and a target feature B.

– Excludes: For two features A and B, if A excludes B then A and B cannot be present in the same product. The bottom-right figure of Table 2.1 illustrates an excludes cross-tree constraint relationship between features A and B.

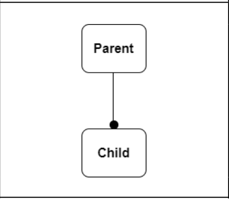
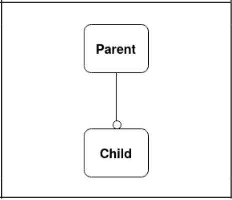
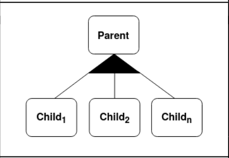
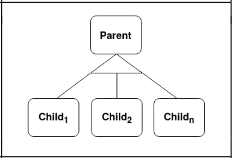
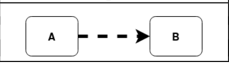
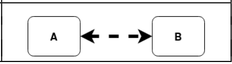
Unary relations	
Mandatory	Optional
	
Set relations	
Inclusive (Or)	Alternative (XOR)
	
Cross-tree constraints	
Requires	Excludes
	

Table 2.1: Feature model relations

Such as Benavides et al. [4] indicate, more complex cross-tree relationships exist in the literature to define constraints in the form of generic propositional formulas such as “A and not B implies C”.

Figure 2.1 illustrates a valid configuration for the FM of the Debian operating system: nodes represent the features of the model (i.e., selectable packages to install) and edges are the constraints between features (e.g., packages that require the installation of other packages). In this example, we can observe that packages *texteditor*, *bash* and *gui* are mandatory (i.e., they must be always included in any Debian configuration) whereas the package *games* is optional. We can also observed that *textditor* requires at least one of the packages *vi*, *gedit* or *openoffice*. Likewise, feature *gui* requires at least one of *gnome* or *kde*. In the case of the package *games*, we observe that it requires either *gnuchess* or *glchess*, but only one, non both. Similarly, the package *openoffice.org-1* requires either version *openoffice.org-1.1* or *openoffice.org-1.2*. Finally, we observe that *openoffice.org-1* strictly requires the installation of *gnome*.

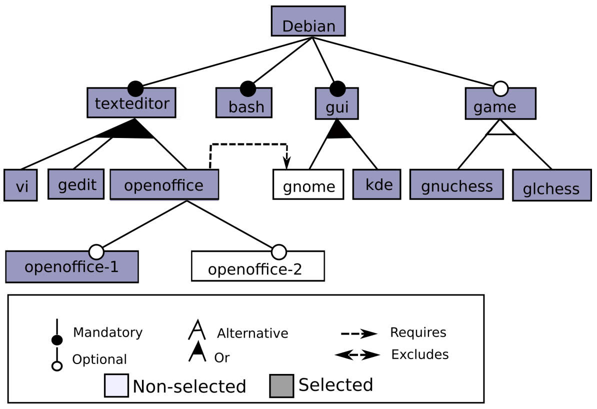


Figure 2.2: Feature model of a Debian derivative example with a non-valid configuration (gray features).

Hence, the selection of features {*Debian*, *texteditor*, *vi*, *gedit*, *openoffice*, *openoffice-1*, *bash*, *gui*, *gnome*, *kde*, *game*, *glchess*} exemplifies a valid product. Figure 2.2 illustrates a non-valid configuration for the FM of the Debian operating system: the selection of features {*Debian*, *texteditor*, *vi*, *gedit*, *openoffice*, *openoffice-1*, *bash*, *gui*, *kde*, *game*, *gnuchess*, *glchess*} exemplifies a non-valid configuration that does not respect the requires cross-tree constraint between the option *openoffice.org-1* and *gnome* (only the first option is selected), and *gnuchess* and *glchess* are selected for the option *game* when only one feature must be selected (*game* represents an alternative set of features).

2.2.2 Cardinality-based feature models

Cardinality-based feature models introduce cardinality annotations in solitaire features and on subtrees (group of features) of traditional (FODA) FMs to specify how many instances (clones) can be included in a product configuration [43] [48]:

- Feature cardinality: A sequence of intervals in the form $[n_i, n_{i'}]$ with the lower bound n_i and the upper bounds $n_{i'}$. Such as Benavides et al. [4] mention, cardinalities for mandatory and optional features are [1, 1] and [0, 1], respectively.

- Group cardinality: For a group of k features, an interval [n, n'] with n as the lower bound and n' as the upper bound, and n' <= k. For basic FMs, an interval [1, k] for k sub-features in optional sets, and [1, 1] for alternative sets.

Cardinality-based feature models can include inconsistencies concerning the intervals definition and their components [49] such as false unbounded intervals and gap intervals. Figure 2.3 presents the equivalent cardinality-based FM of the feature

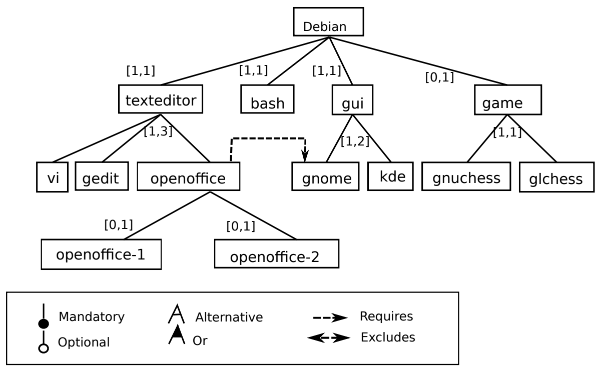


Figure 2.3: Feature model example of a cardinality-based FM of the base FM of Figures 2.1 and 2.2.

model of Figures 2.1 and 2.2. The main motivation of cardinality-based FMs is their practical application by the inclusion of UML-like feature multiplicities [4].

2.2.3 Extended feature models

Extended feature models include additional details about features in the form of attributes to add measurable information about their features [4, 46]. Feature attributes provide extra information required to support features in terms of measurable characteristics and complex cross-tree relations [45].

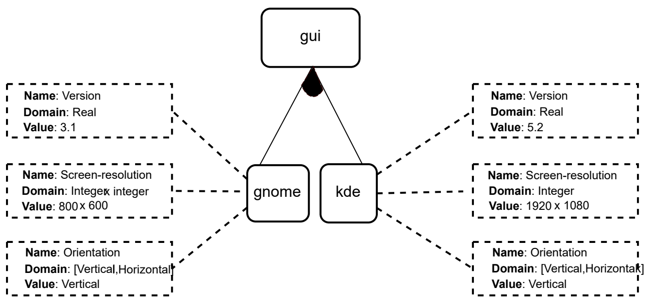


Figure 2.4: Extended feature model example of part of the base FM of Figures 2.1 and 2.2.

For instance, Figure 2.4 depicts a partial extended feature model using the notation proposed in [50]. As illustrated in the figure, an attribute mainly consists of:

- Attribute name. Name or id of the attribute, such as *Version*, *Screen-Resolution*, and *Orientation*.

- Attribute domain. It defines the range of possible values for the attribute. For example, *Real*, *(Integer, Integer)*, and *[Horizontal, Vertical]*.
- Attribute value. The value of the attribute. This could be an specific or default value within the domain, or an expression depending on the value of other attributes of the same or other features.

Extended feature models also considered relations between attributes. Hence, features could require or excludes other features depending on the value of any of their attributes. For instance, in the base FM of Figures 2.1 and 2.2, openoffice.org-1 could require an specific version of gnome, such as *gnome.version* >= 8.1.1. FeatureIDE [51] considers the use of attributes in FMs from version 3.5. Current version of pure::variants [52] also support FMs with attributes.

2.3 Other variability modeling approaches

2.3.1 Orthogonal Variability Model (OVM)

Orthogonal Variability Model (OVM) is a modeling language for defining a cross-sectional view of the variability of features in an SPL across all software development artifacts [53]. OVM permits define the software variability separately without updating the requirements, design, and other software base models. Base models carry out the variability defined by the variability elements in the OVM model. The main classes in OVM are variation points and variants:

- A variation point documents the aspects that can vary in the software development artifacts. Customers or development staff can choose those aspects.
- A variant is related to a variation point and documents how this variation point can vary.

An OVM traditionally is used for documenting SPL variability [53], and it does not consider commonalities in the software artifacts explicitly [54]. Nevertheless, an OVM expresses common variability in all the software artifacts. Regarding automated support, OVM instances like FMs support a translation into logical approaches for the automated analysis [55]. Figure 2.5 shows an OVM for the FM of Figures 2.1 and 2.2.

2.3.2 Debian variability model

The work of Galindo et al. [16] describe that a Debian–based installation is linked with a set of package repositories, and each package repository is associated with a configuration file written in the Debian package dependency language. The Debian pacakge dependency language is a textual language that support information regarding the software packages along with their relationships and dependencies. For each

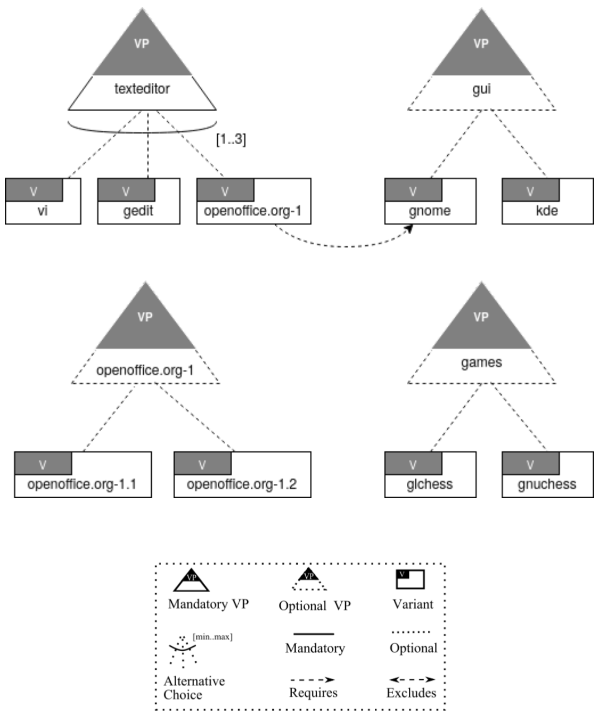


Figure 2.5: OVM of the Debian distribution example.

package in the file, a set of properties exists. Relationships between packages are *Depends*, *Suggests*, *Conflicts* and *Replaces*. We recommend to review the work of [16] for more details.

2.3.3 CVL

Common Variability Language (CVL) is a domain-independent language for specifying and resolving variability [56]. CVL distinguishes between the domain and variability models: domain model using a meta-object based model without variability concerns, and variability model using the CVL metamodel. CVL provides an executable engine to generate fully configured products considering the variability constraints. That engine seems promising for integrating CVL with development tools.

2.3.4 Clafer

Clafer (class, feature, reference) is a class modeling language with first-class support for feature modeling [57]. Clafer mixes constructs from the UML and features modeling worlds. Clafer seems promising for the existing automated reasoning support for the language.

2.4 Summary

This chapter defined and exemplified a variability and configuration model as a feature model and its corresponding instantiation. Next, we presented and illustrated FM notations for variability and configuration modeling. We also summarize other variability and configuration modeling techniques, some of them including validation tool support.

Chapter 3

Automated analysis of variability models

Abstract

The manual analysis of large-scale FMs and their configurations are complex and error-prone tasks. A solution approach is to translate them into a logic representation, apply analysis operations using off-the-shelf reasoning solver or programming tools, and obtain analysis results. This chapter describes operations for assisting in that Automated Analysis of Feature Model (AAFM) process. Specifically, this chapter describes the QUICKXPLAIN algorithm for the detection of preferred minimal conflict, the FASTDIAG algorithm for the detection of preferred minimal diagnosis, and off-the-shelf solvers to complete partial product configuration.

3.1 Introduction

The development process of a Variability Intensive System (VIS) considers identifying and representing the system's components and relationships among those components as two core activities. The application and analysis of FMs is a common approach to perform those analysis tasks. Benavides et al. [4] mention that the manual analysis of FMs is a time-demanding and error-prone activity and the Automated Analysis of Feature Models (AAFM) process permits solving those issues. The AAFM process starts by translating the FM and additional information, such as global restrictions, into a logical set of constraints. Afterward, queries can proceed with the translated model using off-the-shelf solver and other tools such as programming solutions, and thus obtaining analysis results [**?**]. Figure 3.1 illustrates the AAFM process.

Such as Galindo et al. [**?**] summarize, six different variability facets exist where the AAFM is currently applied: i) product configuration and derivation; ii) testing and evolution; iii) reverse engineering; iv) multi-model variability-analysis; v) variability modelling, and; vi) variability-intensive systems. The first AAFM application

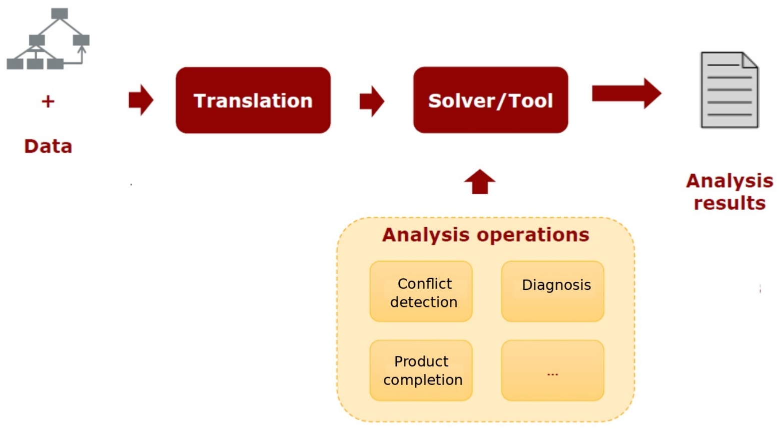


Figure 3.1: Automated Analysis of Feature Models (AAFM) process.

results in the most traditional usage of automated analysis mechanisms. This thesis aims to contribute to it.

Developing FM and product configurations without errors or conflicts requires identifying each conflict and the necessary steps to solve or diagnose them one by one. Hence, conflict detection and diagnosis are operations needed for getting conflicts-free models. The completion of a product configuration of FM by hand also represents an error-prone and time-consuming task. Solutions for those tasks to work efficiently on large-scale models represent high-value tasks nowadays. AAFM solutions for the conflict detection, diagnosis, and completion of products already exist. Next sections describe an existing algorithm for detecting Minimal Conflict Sets (MCS), an existing algorithm for detecting Minimal Diagnosis (MD), and traditional approaches to complete product configurations.

3.2 Minimal Conflict Sets (MCS) detection

An MCS of a system represents a minimal set of constraints in conflict. For definition 1 [**?**], it is necessary to identify the set of constraints B that represents a consistent background knowledge, and the set of constraints C that is the suspected subject of a conflict search.

Definition 1 *A set* $AC = B \cup C = \{c_1, c_2, ..., c_n\}$ *represents the set of all constraints in the knowledge base; that is,* AC *is the union of the consistent knowledge base* B *and the suspicious set of constraints subject of conflict search* C*. Then, a conflict* $CS = \{c_a, c_b, ..., c_z\}$ *is a non-empty and non-consistent subset of* C*.* CS *is minimal if*

$\neg\exists$ CS' such that $CS' \subset CS$ CS is preferred if the order of its constraints follow a defined ranking of preferences.

For the FM of Figure 2.2, concerning definition 1, the consistent base knowledge B is a formal definition of the FM, that is, a logic representation of the set of features and their relationships. We can detect conflict in the configuration of products for that model. For the product configuration C = {*Debian*, *texteditor*, *bash*, *gui*, *game*, *vi*, *gedit*, *openoffice.org-1*, *gnome*, *kde*, *glchess*, *openoffice.org-1*}, the resulting minimal conflict set is {} because C represents a consistent configuration. For the product configuration C = {*Debian*, *texteditor*, *bash*, *gui*, *game*, *vi*, *gedit*, *openoffice.org-1*, *kde*, *gnuchess*, *glchess*, *openoffice.org-1*}, the resulting preferred minimal conflict set is {*openoffice.org-1*, $\neg gnome$}}. The next lines describe the QUICKXPLAIN algorithm for efficiently detecting preferred MCS.

3.2.1 QUICKXPLAIN algorithm

QUICKXPLAIN [20] is an efficient approach to determine a minimal conflict set. QUICKXPLAIN receives C as the set of suspicious constraints with conflict and B as the set of consistent constraints of the background knowledge. Then, a conflict does not exist if $B \cup C$ is consistent or C is empty. On the other hand, QUICKXPLAIN proceeds by returning the results of the function QX. QX receives the parameters C (initially the complete set of constraints with conflict), B (initially the knowledge base), and $B\delta$ (initially empty) that represents the last items added to B. Function QX follows a divide-and-conquer approach for conflict detection. Hence, $B\delta$ corresponds to the set of constraints added for reviewing the consistency of the knowledge base, and C is the set of constraints to continue analyzing if the current B is consistent. Algorithms 3.1 and 3.2 show the pseudo-code of the functions of QUICKXPLAIN.

Algorithm 3.1 QUICKXPLAIN$(C, B) : CS$

```
1: if CONSISTENT($B \cup C$) then
2:     $return$('no conflict')
3: else if $C = \emptyset$ then
4:     $return(\emptyset)$
5: else
6:     $return$(QX($C, B, \emptyset$))
7: end if
```

Algorithm 3.2 QX($C = \{c_1..c_m\}, B, B\delta) : CS$

```
1: if $B\delta \neq \emptyset$ and InConsistent($B$) then
2:     return($\emptyset$)
3: end if
4: if $C = \{c_\alpha\}$ then
5:     return($\{c_\alpha\}$)
6: end if
7: $k = \lfloor \frac{m}{2} \rfloor$
8: $C_a \leftarrow c_1...c_k; C_b \leftarrow c_{k+1}...c_m;$
9: $\Delta_2 \leftarrow QX(C_a, B \cup C_b, C_b);$
10: $\Delta_1 \leftarrow QX(C_b, B \cup \Delta_2, \Delta_2);$
11: return($\Delta_1 \cup \Delta_2$)
```

QuickXPlain permits determining one MCS per computation. Felfernig et al. [2] indicate that we need to update adequately or delete one of the constraints of an MCS for solving it, and, if the model is non-consistent yet, to apply QuickXPlain and repeat the process. When the resulting model is consistent, the updated constraints represent a diagnosis or solution for the model. Table 3.1 shows the tracking steps of a QuickXPlain application for the FM configuration example of Figure 2.2. Column B represents the set of consistent constraints of the base knowledge for the Feature Model (FM) definition and column C is the set of selected features. The final result represents a minimal conflict that we can solve by updating one of its constraints adequately. In this case, a solution is to update the state of $\neg gnome$.

3.3 Minimal diagnosis detection

Identifying and solving conflicts one by one is necessary to obtain a conflict-free model: we need to identify a conflict first, adapt (update or eliminate) constraints of that conflict for its solution, and repeat this process until no more conflict exists, that is, until reaching a consistent model. The set of all the adapted constraints for getting a conflict-free model represents a diagnosis. Definition 2 formally defines the term diagnosis [**?**].

Definition 2 *A set $AC = \{c_1, c_2, ..., c_n\}$ represents the set of all constraints in the problem for diagnosis; that is, AC is the union of the consistent base knowledge B and the set of constraints subject of conflict search C: $AC = B \cup C$. Then, a diagnosis is a set of constraints $\Delta \subseteq C$ such that $(B \cup C - \Delta)$ results in a consistent or conflict-free set. Δ is minimal if $\neg\exists$ Δ' such that $\Delta' \subset \Delta$. A minimal diagnosis is of minimal cardinality if there does not exist a minimal diagnosis Δ' such as $|\Delta'| < |\Delta|$.*

Step	**C**	**B**	**B**δ	C_a	C_b	**Return**
1	{Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	B	$\emptyset$	{Debian, texteditor, bash, gui, game, vi, gedit}	{openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{openoffice-1, ¬ gnome}
2	{Debian, texteditor bash, gui, game, vi, gedit}	$B \cup$ {kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome},	{openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}			$\emptyset$
3	{openoffice, kde, gnuchess glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	B	$\emptyset$	{openoffice kde, gnuchess}	{glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{openoffice-1, ¬ gnome}
4	{openoffice, kde, gnuchess}	$B \cup$ {glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}			$\emptyset$
5	{glchess, openoffice-1, ¬openoffice-2, ¬gnome}	B	$\emptyset$	{glchess, openoffice-1}	{¬openoffice-2, ¬gnome}	{openoffice-1, ¬gnome}
6	{glchess, openoffice-1}	$B \cup$ {¬openoffice-2, ¬gnome}	{¬openoffice-2, ¬gnome}	{glchess}	{openoffice-1}	{openoffice-1}
7	{glchess}	$B \cup$ {¬openoffice-2, ¬gnome, openoffice-1}	{openoffice-1}			$\emptyset$
8	{openoffice-1}	$B \cup$ {¬openoffice-2, ¬gnome}	$\emptyset$			{openoffice-1}
9	{¬openoffice-2, ¬ gnome}	$B \cup$ {¬gnome}	{openoffice-1}	{¬openoffice-2}	{gnome}	{openoffice-1}
10	{¬openoffice-2}	$B \cup$ {openoffice-1} gnome}	{¬gnome}			$\emptyset$
11	{¬gnome}	$B \cup$ {openoffice-1}	$\emptyset$			{¬gnome}

Table 3.1: QUICKXPLAIN execution tracking on configuration example of Figure 2.2

A minimal diagnosis for the FM configuration of Figure 2.2 has to consider solutions for each conflict. Hence, this example contains two diagnosis options. For getting a conflict-free model, the user has to solve each diagnosis. Cases with multiple diagnosis instances can exist, and determining all the diagnosis can be computationally an expensive task. Model constraints can be in relevance order for obtaining preferred diagnosis, then obtaining all the diagnosis to look for the preferred one is a time-demanding and lost time activity since solving one diagnosis is enough for a conflict-free model. The next lines describe the FASTDIAG algorithm to determine a minimal preferred diagnosis.

3.3.0.1 FASTDIAG algorithm

FASTDIAG algorithm permits determining a preferred or leading diagnosis concerning a previously defined relevance order of constraints in the knowledge base. FASTDIAG follows the algorithmic structure and reasoning of QUICKXPLAIN for a different purpose, that is, the diagnosis detection without the calculation of MCS instances. Hence, FASTDIAG is based on conflict-independent search strategies [22]. Algorithms 3.3 and 3.4 give the pseudo-code of FASTDIAG functions.

Algorithm 3.3 FASTDIAG(C, AC) : *diagnosis* Δ

```
1: if $C = \emptyset$ or INCONSISTENT$(AC - C)$ then
2:     $return(\emptyset)$
3: else
4:     $return(\text{FD}(\emptyset, C, AC))$
5: end if
```

Algorithm 3.4 FD$(D, C = \{c_1..c_q\}, AC)$: *diagnosis* Δ

```
1: if $D \neq \emptyset$ and CONSISTENT$(AC)$ then
2:     return$(\emptyset)$
3: end if
4: if $|C| = 1$ then
5:     return$(C)$
6: end if
7: $k = \lfloor \frac{q}{2} \rfloor$
8: $C_a \leftarrow c_1...c_k$; $C_b \leftarrow c_{k+1}...c_q$;
9: $\Delta_1 \leftarrow \text{FD}(C_b, C_a, AC - C_b)$;
10: $\Delta_2 \leftarrow \text{FD}(\Delta_1, C_b, AC - \Delta_1)$;
11: return$(\Delta_1 \cup \Delta_2)$
```

Step	D	C	AC	C_a	C_b	Return
1	$\emptyset$	{Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{Debian, texteditor, bash, gui, game, vi, gedit}	{openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{glchess, ¬ gnome}
2	{openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{Debian, texteditor, bash, gui, game, vi, gedit}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit}			$\emptyset$
3	$\emptyset$	{openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{openoffice, kde, gnuchess}	{glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{glchess, ¬ gnome}
4	{glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{openoffice, kde, gnuchess}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess}			{glchess, ¬ gnome}
5	$\emptyset$	{glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, glchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{glchess, openoffice-1}	{¬ openoffice-2, ¬ gnome}	{glchess, ¬ gnome}
6	{¬ openoffice-2, ¬ gnome}	{glchess, openoffice-1}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, glchess, openoffice-1}	{glchess}	{openoffice-1}	{glchess, ¬ gnome}
7	{openoffice-1}	{glchess}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, glchess}			{glchess}
8	{glchess}	{openoffice-1}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, openoffice-1}			$\emptyset$

Table 3.2: Tracking of the FASTDIAG execution of configuration example of Figure 2.2 (part 1)

Step	D	C	AC	C_a	C_b	Return
8	{glchess}	{openoffice-1}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, openoffice-1}			$\emptyset$
9	{glchess}	{¬ openoffice-2, ¬ gnome}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, openoffice-1, ¬ openoffice-2, ¬ gnome}	{¬ openoffice-2}	{¬ gnome}	{¬ gnome}
10	{¬ gnome}	{¬ openoffice-2}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, openoffice-1, ¬ openoffice-2}			$\emptyset$
11	$\emptyset$	{¬ gnome}	$B \cup$ {Debian, texteditor, bash, gui, game, vi, gedit, openoffice, kde, gnuchess, openoffice-1, ¬ openoffice-2, ¬gnome}			{¬ gnome}

Table 3.3: Tracking of the FastDiag execution of configuration example of Figure 2.2 (part 2)

Assuming that conflicts to diagnosis exist, If the conflict set C is non-empty, and AC without C is consistent, algorithm FastDiag calls and waits for the recursive results algorithm FD. FD first reviews the consistency of AC as a source of diagnosis. Because always AC contains C and does not contain D, initially S is the constraints set with conflicts and D is empty, when D is not empty, and AC is consistent D is the source of conflict. When that base case is not accomplished, either because D is empty (such as at the beginning) or AC is consistent (this is only possible after removing elements from $AC - D$ represents the last removed elements from AC), then AC is still in conflict, and C is a source of conflict. Then, FD reviews the size of C since if it were minimal (size 1), then C is the diagnosis. If C is not of minimal size, FD proceeds to partition C in the sets C_1 and C_2, of which the last one corresponds to the most preferred partition. Afterwards, FD calls FD over C_2, C_1 and $AC - C_2$ to review if C_2 is the diagnosis source, and if not so, to continue reviewing C_1 with that goal. Tables 3.2 and 3.3 show the tracking

3.4 Product completion

The completion of partial configurations consist of finding the non-selected components necessary to update that permit evolving the partial setting into a complete product configuration. In FM configurations, each feature is decided to be either present or absent in the resulting products, whereas in partial configurations, some features are undecided. The completion of partial configurations is a non-trivial and computationally expensive task mainly for the FM constraints [58], a process that is usually computationally more expensive in large-scale FMs. Product configurations can result in misconfigurations (i.e., non-valid configurations) which can impact on the system availability [59]. Unavailability of the Facebook platform [60], service-level problems of Google [61], and invalid operation of Hadoop clusters [62] are known misconfiguration examples.

An usual and efficient solution for the completion of partial products is the application of reasoning tools such as CSP and SAT solvers for obtaining a set of necessary features for the completion of the partial configuration. Those solutions can be minimal, but they not always represent the preferred configuration [63].

Figure 3.2 illustrates a conflict-free partial product and features for a minimal completion of the FM of Figure 2.2. The partial configuration presents the selection of four features, $\{Debian, texteditor, bash, gui\}$ (features in background color grey). Given the rest of the model's features (features in background color white, and features in background color green), features in background color green represent a preferred minimal completion; that is, the set of features necessary for obtaining a preferred and minimal completion of the partial configuration. That completion per default takes the typographic order of each non-selected feature as the order of preference.

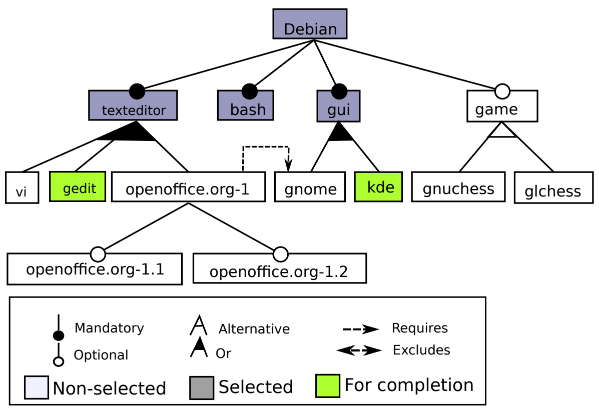


Figure 3.2: Example of a partial product and features for completion in a Debian derivative example.

3.5 Summary

This chapter described AAFM solutions for assisting the production process of conflict-free models. QUICKXPLAIN, FASTDIAG, and the used of solvers for the completion of partial product are efficient algorithm and tool solutions for identifying MCS, minimal diagnosis and the completion of partial products. The computing efficiency of the first two solution are the base for the contributions of this thesis.

Part III

Our contributions

Chapter 4

A review of current AAFM solutions for minimal conflict, diagnosis, and product completion

Abstract

Kang et al. defined Feature Models (FMs) 30 years ago that are useful tools for modeling Variability Intensive Systems (VIS). The Automated Analysis of Feature Model (AAFM) is a thriving, motivating, and active research area. In 2010, Benavides et al. published a survey about the first 20 years of AAFM. This chapter reviews the AAFM solutions for conflict detection, diagnosis, and product completion in FMs from 2010 to 2019. We highlight that the computing performance and approaches of existing solutions to work on large-scale and high-variability FM and configurations are real research opportunities for developing new and more efficient solutions for that purpose.

4.1 Introduction

Variability is the ability of a system for being extended, changed, customized, or configured for using it in specific contexts [64]. Variability Intensive Systems (VIS) can tailor or adapt to particular needs in different domains [65]. For example, VIS can easily update or expand their features, be aware of mobile applications' location and resource, and fault tolerance and ease to recover of critical embedded systems [66]. Variability has become a key in software systems for the steadily increasing demand for highly customizable products [67]. In Software Product Line Engineering (SPLE), variability models such as FMs are primary artifacts for the SPLs success [5, 68]. Variability modeling specifies all significant and legal combinations of features in a software product line [69].

The manual analysis of variability models is a complicated, tedious, and error-prone task, and the Automated Analysis of Variability Models (AAVM) appears to face that issue. For example, Ross et al. [68] present the tool FAMA-OVM for the automated analysis of Orthogonal Variability Models (OVM). Likewise, the works of [70] and [71] describe the structure, components, and results improvements in applying a process for the AAVM of OVM. In the FM context, the Automated Analysis of Feature Models (AAFM) is a current and active research area. Such as Benavides et al. [72] indicated, the AAFM and product configurations have a lot of potential synergies because, in the product reconfiguration in dynamic SPLs, both concepts overlap. An FM configuration describes a valid product if the selection of features satisfies all the FM constraints [40].

Inconsistencies in models imply the presence of errors [73] [74]. The main challenge of fixing inconsistencies in variability models is to fix them for all configurations [69]. The consistency of an FM means that it remains well-formed (syntactic consistency) and defines at least one valid product (semantic consistency) [75]. Checking and resolving FM inconsistencies are essential tasks during the product line evolution [76]. Automated tools for inconsistency checking of FM and configuration are critical activities for successful SPLs [77]. A traditional AAFM approach is to convert the FM into logic formulas supportable by reasoning solvers such as CSP and SAT solvers, to define and execute analysis tasks regarding the model information and reasoning analysis on those solvers [78]. This chapter focuses on AAFM operations for the product configuration.

Benavides et al. [4] in 2010 presented a Systematic Literature Review (SLR) about FM and AAFM operations. Different AAFM proposals exist with the use of various formal approaches such as CSP, SAT, and BDD solvers [5]. The next lines synthesize state of the art regarding AAFM operations for minimal conflict detection, minimal diagnosis, and the minimal completion of partial product configuration from 2010 to August of 2019. This chapter has the following organization: first, we present the primary method and objectives of this review. Then, we summarize the main discovered results.

4.2 Systematic Literature Review (SLR)

A Systematic Literature Review (SLR) is a secondary research study based on previous (primary) studies to answer defined Research Questions (RQs) [79]. An SLR identifies, evaluates, integrates, and synthesizes evidence and results of relevant primary studies to answer a defined set of RQs, and to analyze possible new results [80, 81]. An SLR traditionally consists of the following phases (Figure 4.1) [82].

i) Research Questions: After defining a topic of research, the first step of an SLR is to determine the Research Questions (RQs), the main questions to answer in the

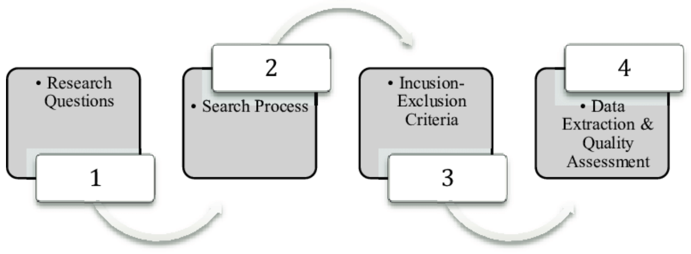

• Research Questions
1
2
• Search Process
• Incusion-Exclusion Criteria
3
4
• Data Extraction & Quality Assessment

Figure 4.1: Steps of a traditional SLR process

SLR process.

$ii)$ Search Process: The search process presents the sources of search and keywords to select primary studies.

$iii)$ Inclusion/Exclusion Criteria: The next step is to establish inclusion/exclusion criteria in the search process and quality criteria for the primary articles.

$iv)$ Data Extraction & Quality Assessment: During data extraction, the information is extracted, collected, and organized for its evaluation and final results.

4.3 Review process

This SLR aims to know details about existing AAFM solutions for conflict detection, diagnosis detection, and product completion in FMs from 2010 to December of 2019. We want to know the main focus, implementation approaches, execution requirements, and computing performance of existing AAFM solutions for those purposes. Then, we will classify existing options, their scenarios of use, along with their benefits and issues.

4.3.1 Definition of research questions

The RQs are built based on the PICO (Population, Intervention, Comparison, Outcome) strategy [83] to derive search strings applicable to digital libraries. In that context, Population corresponds to FMs; Intervention represents a solution for the conflict detection, diagnosis, or product configuration; the Comparison is blank; the Outcomes correspond to the found research from 2010 to December of 2019. The next lines define RQs and search strings based on the PICO approach.

- RQ1: What are the automated solutions for conflict detection in FMs from 2010 to December of 2019? Following the process described by [81], we defined the next sub-questions.

  i. What conflicts do these solutions focus on?

  ii. Are those solutions new or extensions of existing operations?

iii. What are the computing approaches and functioning requirements of those solutions? Do they differ in their base solutions?

iv. Are those solutions able to work on large-scale FM instances?

v. What is the computing performance and degree of improvement or deterioration of those solutions regarding their base solutions?

- RQ2: What are the automated solutions for diagnosis in FM instances from 2010 to December of 2019? Following the process described by [81], this question inspires the next sub-questions.

  i. What conflict do these solutions focus on?

  ii. Are those solutions new or extensions of existing operations?

  iii. What are the computing approaches and functioning requirements of those solutions? Do they differ in their base solutions?

  iv. Are those solutions able to work on large-scale FM instances?

  v. What is the computing performance and degree of improvement/deterioration of those solutions regarding their base ones?

- RQ3: What are the automated solutions for the completion of product configuration in FM instances from 2010 to December of 2019? Following the process described by [81], this question inspires the next sub-questions.

  i. Are those solutions new or extensions of existing operations?

  ii. What are the computing approaches and functioning requirements of those solutions? Do they differ in their base solutions?

  iii. Are those solutions able to work on large-scale FM instances?

  iv. What is the computing performance and degree of improvement/deterioration of those solutions regarding their base ones?

4.3.2 Source material

We follow an automated selection method to look for papers in Google scholar [84], Scopus [85], and WebOfKnowledge [86]. We used the next key-phrase to look for research publications in those databases.

- $(\{[f \mid F]eature[s \mid \text{␣}]\}\ \{[m \mid M]odel[s \mid \text{␣}]\}\ AND\ \Big[\{[a \mid A]utomated\}\ OR\ \{[a \mid A]utomatic\}\Big]\ AND\ \Big(\{[a \mid A]nomal[y \mid ies]\}\ OR\ [c \mid C]onflict[s \mid \text{␣}]\}\ OR\ (\{[d \mid D]iagnosis\})\ OR\ (\{[c \mid C]ompletion\}\Big)\ AND\ \{PubYear >= 2010\}\ AND\ \{PubYear <= 2019\}\ AND\ \{Language\ in(Spanish \mid English)\}$

Examples of valid search string for our SLR are:

- Feature Models Automated Anomaly

- Feature Models automated conflicts
- Feature models Automated Diagnosis
- Feature Models automatic diagnosis
- Feature models Automated Completion
- Feature Models automatic completion

We accomplish a systematic review using a valid search string in the title, abstract, or keywords set for each analyzed paper. PubYear refers to the publication date of the article. Specifically, we looked for papers from 2010 to December of 2019, the search date, both years inclusive. Scopus and WebOfKnowlede libraries support search string accordingly to our definition. Regarding the language, we look for papers written in English only. Google scholar permits looking for defined terms either in the title or somewhere in the document. In Google Scholar, we looked for results year-per-year from 2010. We used the primary 20 results per-year for the high-quantity of results. We obtained 200 results from Google Scholar, 43 results in Scopus, and 23 results in WebOfKnowledge. Hence, we received 266 results in our automated search. As a first filter step, we discarded duplicated work for obtaining 223 results. Second, after rejecting works non-available and non-related with the research topic, we obtained 120 works. According to the Google Scholar database data, there was only one non-available work from the SPLC 2010 proceedings. Still, nobody presented that article in the SPLC 2010 conference, and it did not appear in its proceeding. We contacted the paper's authors, who reported that the paper appeared in another conference using a different title. We decided to discard that work. Third, we applied a filter to reject works non-related to the research questions. Finally, we obtain 51 primary studies in the scope of this review. The next section describes the last inclusion criteria.

4.3.3 Inclusion criteria

On the primary results without duplication and non-available data, we applied a general inclusion filter to consider only articles of the AAFM area. We analyzed the title, keywords, and abstract of each found paper to positively support at least one of the following Inclusion Criteria (IC).

4.3.4 Results

Applying inclusion criteria, we obtain 51 research works. Tables 4.2, 4.3, and 4.4 the resulting papers ordered alphabetically by the author's name and ascendingly by the publication year. Values of column *FM Kind* are 1 for FODA FMs, 2 for Cardinality-based FMs, 3 for Extended FMs, and 4 for all these options. We use the yellow background color to mark the accomplished options. The column *Large FM* is yellow for those researches that work with FM of 5000 features or more.

Inclusion Criteria	
IC-1:	Does this paper presents an AAFM solution for the conflict detection?
IC-2:	Does this paper present an AAFM solution for diagnosis?
IC-3:	Does this paper present an AAFM solution for the product completion?

Table 4.1: Quick review research questions.

Table 4.5 presents the number of articles concerning their applied process and main goal. We group the papers in those with use tools only and those which use tools and solvers. This table presents results for the conflict and diagnosis detection only since we do not obtain results for the completion of product configuration in FMs. In the next lines, we answer the RQs along with each sub-question.

4.3.5 RQ1: What are the automated solutions for detecting conflicts in feature models from 2010 to 2019?

Table 4.5 shows that a no-normal tendency exists from 2010 to December of 2019 concerning the research work for the conflict detection in FMs. In the next lines, we answer each sub-question of RQ1.

- Sub-question 1: We classify the obtained papers regarding their main focus in the next classes: i) Non-Consistent Feature Model (NC FM); ii) Non-Consistent Configuration (NC Conf); iii) Non-Consistent Feature Model and Configuration (NC FM / Conf); iv) Mutation or Evolution of Feature Model (Mut / Evol); and v) Specific Fault (Spec Fault) such as void FM, empty FM, dead feature, False optional, partial dependencies, orphan features). The higher number of work focus on the conflict detection on specific FM inconsistencies. Figure 4.2 depicts a summary of the obtained papers for the conflict detection in FMs.

- Sub-question 2: almost all the articles present new solution proposals for conflict detection in FM.

- Sub-question 3: Table 4.5 resumes the following results for articles which focused on conflict detection only:

  - seven articles in 2010: four articles using tools, and three articles using tools and solvers.
  - five articles in 2011: three articles using tools, and two articles using both tools and solvers.

			Source			**Main Goal**				**Solution**			
#	Year	Author	Journal	Conference	Workshop	Conflict	Diagnosis	Completion	Focused Anomaly	Tool	Solver	FMs Kind	Large FMs
1	2013	Acher et al. [12]	✓			✓			Conflict in product / model	✓	✓	1	
2	2014	Achour et al. [87]		✓		✓			Conflict in product / model	✓		1	
3	2014	Afzal et al. [88]		✓		✓			Wrong cardinality, exclusion of mandatory features, requires of excluded features	✓		3	
4	2016	Ananieva et al. [89]	✓			✓			Partial FM dependencies	✓	✓	1	
5	2017	Arcaini et al. [90]	✓			✓			Dead feature, redundant constraint, false optional	✓	✓	1	
6	2015	Arcaini et al. [91]		✓		✓			Faults for mutation	✓	✓	1	
7	2016	Arcaini et al. [92]		✓		✓			Conformance faults	✓	✓	1	
8	2016	Asadi et al. [93]	✓			✓			Potential inconsistency, strong inconsistencs	✓	✓	1	
9	2014	Asadi et al. [94]	✓			✓			Potential inconsistency, strong inconsistency, configuration inconsistency	✓	✓	1	
10	2014	Barreirros & Moreria [95]		✓		✓			Product configuration inconsistencies	✓	✓	1	
11	2018	Bhushan et al. [96]	✓			✓			Conflicts in the FM definition	✓	✓	1	
12	2017	Bhushan et al. [97]	✓			✓			Conflicts in the FM definition	✓	✓	1	
13	2010	Bin Abid [98]			✓	✓			Consistency checking in the product derivation	✓		1	
14	2011	Choi [77]	✓			✓			Dead features, void FM, validity of product configuration	✓	✓	1	
15	2015	Da Silva Costa et al. [99]		✓		✓			False optional, dead features	✓	✓	3	
16	2014	Elfaki et al. [64]	✓			✓			False optional, wrong cardinality	✓		2	
17	2012	Felfernig et al. [22]		✓			✓		Diagnosis	✓	✓	1	
18	2013	Felfernig et al. [21]		✓			✓		Diagnosis	✓	✓	1	
19	2015	Felfernig et al. [100]		✓			✓		Anytime diagnosis	✓	✓	1	
20	2018	Felfernig et al. [101]	✓				✓		Anytime diagnosis	✓	✓	1	
	2020	Vidal-Silva PhD. Thesis		✓	✓	✓	✓	✓	Minimal conflicts set, Diagnosis, and minimal preferred completion of product configuration by diagnosis	✓	✓	1	✓

Table 4.2: List of articles ordered by the authors' last name from A to F in the SLR vs. our thesis work.

			Source			Main Goal				Solution			
#	Year	Author	Journal	Conference	Workshop	Conflict	Diagnosis	Completion	Focused Anomaly	Tool	Solver	FMs Kind	Large FMs
21	2010	Galindo et al. [16]		✓		✓			Discovering inconsistencies between packages	✓	✓	1	
22	2011	Gheyi et al. [102]	✓			✓			Soundness of FM refactoring	✓		1	
23	2012	Guo et al. [75]	✓			✓			Consistent evolution	✓		1	✓
24	2013	Henard et al. [78]		✓			✓		Fixing and re-engineering a FM	✓	✓	1	
25	2019	Hinterreiter et al. [76]		✓		✓			Potential inconsistencies	✓		1	
26	2016	Javed et al. [103]	✓			✓			Void feature model, inconsistent product configuration	✓		1	
27	2015	Khtira et al. [41]	✓			✓			Duplication in evolving FM	✓		1	
28	2015	Khtira et al. [104]	✓			✓			Duplication in evolving FM	✓		1	
29	2016	Kowal et al. [67]		✓			✓		Dead feature, redundancies, false optional	✓	✓	1	
30	2015	Lesta et al. [105]	✓			✓			Void FM, dead feature, false-optional, dead attribute, false-optional attribute values	✓	✓	3	
31	2010	Lisboa et al. [106]	✓			✓			Orphan features, void FMs	✓		1	
32	2011	Lisboa et al. [107]		✓		✓			Redundancies, configuration anomalies, inconsistencies	✓		1	
33	2011	López-Herrejon & Egyed [108]		✓			✓		To effectively identify features to fix for guaranteeing consistent product line configurations	✓	✓	1	
34	2012	López-Herrejon & Egyed [69]		✓			✓		Product configuration	✓	✓	1	
35	2012	Marinho et al. [109]			✓	✓			False optional, dead feature, wrong cardinality	✓		3	
36	2017	Mauro et al. [110]		✓		✓	✓		Void FM, dead features	✓	✓	1	
37	2011	Mazo et al. [35]		✓		✓			Conformance FM & product checking	✓		3	✓
	2020	Vidal-Silva PhD. Thesis		✓	✓	✓	✓	✓	Minimal conflicts set, diagnosis, and minimal preferred completion of configuration by diagnosis	✓	✓	1	✓

Table 4.3: List of articles ordered by the authors' last name from G to M in the SLR vs. our thesis work.

			Source			Main Goal				Solution			
#	Year	Author	Journal	Conference	Workshop	Conflict	Diagnosis	Completion	Focused Anomaly	Tool	Solver	FMs Kind	Large FMs
38	2010	Nakajima [111]		✓		✓			FM propositional interpretation and algorithm for locating bugs	✓	✓	1	
39	2010	Nakajima [112]		✓		✓			Unsatisfiable portion (components) of the unsatisfiable FM	✓	✓	1	
40	2018	Nieke et al. [40]		✓		✓	✓		Anomalies in the FM evolution	✓	✓	1	✓
41	2011	Noorian et al. [113]		✓		✓	✓		Detecting and fixing inconsistencies	✓	✓	1	
41	2011	Noorian et al. [113]		✓		✓	✓		Detecting and fixing inconsistencies	✓	✓	1	
42	2014	Quinton et al. [44]		✓		✓			Range inconsistencies	✓	✓	1	✓
43	2012	Ripon et al. [114]	✓				✓		Analysis and verification of FMs	✓		1	
44	2014	Ripon et al. [115]	✓			✓			FM consistency check	✓	✓	1	
45	2016	Schnabel et al. [49]		✓		✓			Bound & gap of cardinality-based FM configuration	✓		2	
46	2019	Vidal [116]	✓				✓		FM configuration	✓	✓	1	
47	2010	Wang et al. [117]	✓			✓			To tolerate inconsistencies in feature models	✓	✓	1	
48	2014	Wang et al. [118]	✓			✓	✓		Detecting and fixing inconsistencies	✓	✓	1	
49	2016	Weckesser et al. [37]	✓			✓			Analysis of bound and interval gaps	✓	✓	2	✓
50	2010	White et al. [119]	✓				✓		Product configuration	✓	✓	1	✓
51	2010	Zhang et al. [120]		✓		✓			Conflicts & anomalies	✓	✓	1	
	2020	Vidal-Silva PhD. Thesis		✓	✓	✓	✓	✓	Minimal conflicts set, diagnosis, and minimal preferred completion of configuration by diagnosis	✓	✓	1	✓

Table 4.4: List of articles ordered by the authors' last name from N to Z in the SLR vs. our thesis work.

			2010	2011	2012	2013	2014	2015	2016	2017	2018	2019
Inclusion Criteria (IC)	IC-1	Tools	4	3	2	0	3	2	2	0	0	1
		Tools & Solvers	3	2	0	2	4	3	4	2	1	1
		# Papers	7	5	2	2	7	5	6	2	1	2
	IC-2	Tools	0	0	1	0	0	0	0	0	0	0
		Tools & Solvers	0	1	2	1	0	1	1	0	1	0
		# Papers	0	1	3	1	0	1	1	0	1	0
	IC-1 & IC-2	Tools	0	0	0	0	0	0	0	0	0	0
		Tools & Solvers	2	0	0	0	0	0	0	1	1	0
		# Papers	2	0	0	0	0	0	0	1	1	0
	# Papers	Tools	4	3	3	0	3	2	2	0	0	1
		Tools & Solvers	5	3	2	3	4	4	5	3	3	1

Table 4.5: Filtered articles classified by year and way for reaching their main goal.

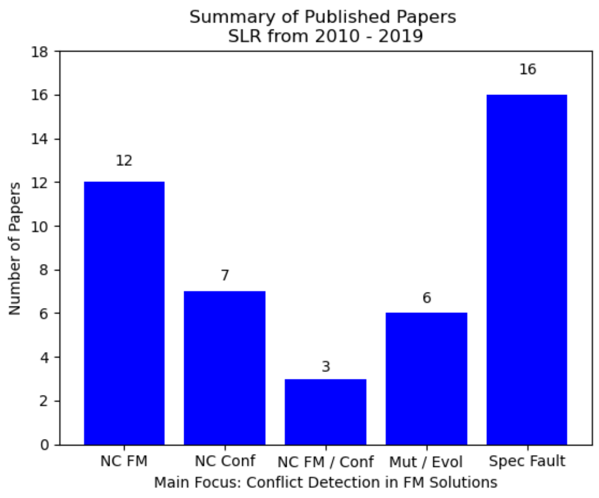


Figure 4.2: Summary of reviewed papers for the conflict detection in FMs from 2010 to 2019.

  - two articles in 2012: both articles using tools.
  - two articles in 2013: both articles using tools and solvers.
  - seven articles in 2014: three articles using tools, and four articles using tools and solvers.
  - five articles in 2015: two articles using tools, and three articles using tools and solvers.
  - six articles in 2016: two articles using tools, and four articles using tools and solvers.
  - two articles in 2017 using tools and solvers.
  - one article in 2018 using tools and solvers.
  - two articles in 2019: 1 article using tools, and 1 article using tools and solvers.

- Sub-question 4: only 6 articles present results on the conflict detection of large-scale FMs.

- Sub-question 5: all the reviewed solutions present new or similar performance results concerning their base solutions; that is, those works that are based in previous works are applied to new data sets, and their results preserve their base ones.

4.3.6 RQ2: What are the automated solutions for diagnosis in feature models from 2010 to 2019?

Table 4.5 shows that a no-normal tendency exists from 2010 to December of 2019 concerning the research work for diagnosis detection in FMs. In the next lines, we answer each sub-question of RQ2.

- Sub-question 1: Such as for sub-question 1 of RQ1, we classify the obtained papers regarding their main focus in the next classes: i) Non-Consistent Feature Model (NC FM); ii) Non-Consistent Configuration (NC Conf); iii) Non-Consistent Feature Model and Configuration (NC FM / Conf); iv) Mutation or Evolution of Feature Model (Mut / Evol); and v) Specific Fault (Spec Fault) such as void FM, empty FM, dead feature, False optional, partial dependencies, and orphan features. We appreciate that the number of papers to diagnosis detection in FM is highly lesser than the number of papers for the conflict detection in FM. Figure 4.3 summarises a classification of the obtained papers for diagnosis detection in FMs.

- Sub-question 2: Almost all the articles present new solution proposals. The work of Felfernig et al. [22] proposes FastDiag and [21] present its application for the FM analysis. The work of [100] presents FlexDiag, base works for [101]. Likewise, Vidal [116] highlight efficient results of FASTDIAG and

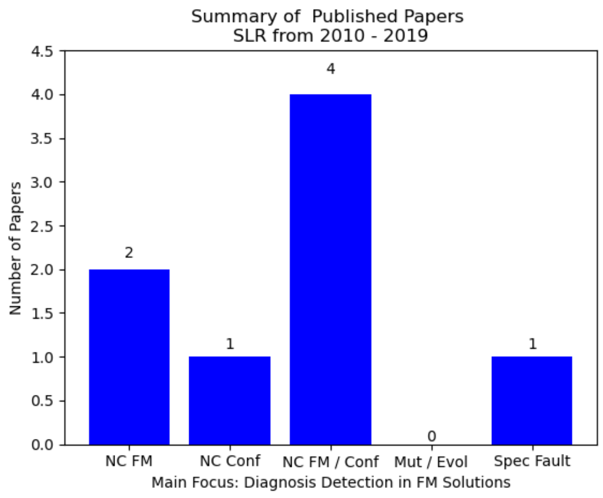


Figure 4.3: Resume of published papers for diagnosis in FMs from 2010 to 2019.

FLEXDIAG solutions. The work of [108] focuses on identifying features necessary to fix for getting valid product configurations. Other selected articles present automatic solutions, and then they require a diagnosis process.

- Sub-question 3: Table 4.5 resumes the following results for articles which focused on diagnosis detection only:
  - one article in 2011 using tools and solvers.
  - three articles in 2012: one article using tools, and two articles using tools and solvers.
  - one article in 2013 using tools and solvers.
  - one article in 2015 using tools and solvers.
  - one article in 2016 using tools and solvers.
  - one article in 2018 using tools and solvers.
- Sub-question 4: no article presents results on large-scale FMs, that is, FMs with 5000 or more features.
- Sub-question 5: all the reviewed solutions present new or similar performance results concerning their base solutions; that is, those works based in previous works are applied to new data sets, and their results preserve their base ones.

Figure 4.4 depicts a summary of the regarding the papers for this study that focus in conflict and diagnosis detection.

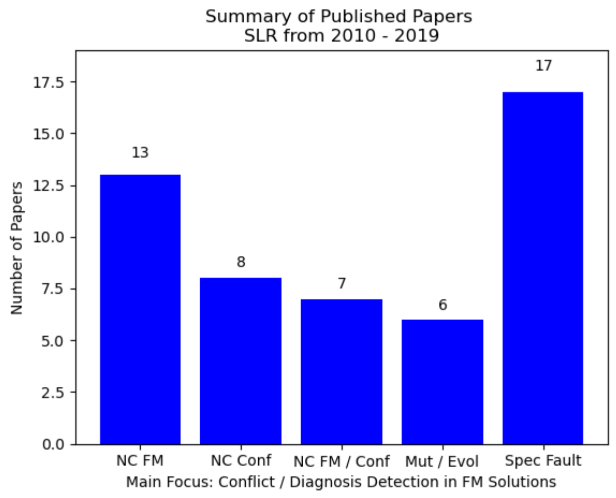


Figure 4.4: Summary of published papers for the conflict detection, diagnosis and product completion from 2010 to 2019.

4.3.7 RQ3: What are the automated solutions for the completion of products in feature models from 2010 to 2019?

Concerning the completion of partial products, we did not find research works in the search process. The work of Ananieva et al. [89] focus on the partial feature model, but they did not consider the completion of those models.

4.4 Discussion

The surveyed research points out that AUTOMATED ANALYSIS OF FEATURE MODEL is a research that is getting mature. Almost half of the obtained papers appeared in journals. Concretely, journals published 24 of the 51 reviewed articles, that is, a $47,1\%$; and 27 articles of 51 were presented in conferences and workshops, that is, $52,9\%$.

4.5 Summary

In this review, we pinpointed the main trends and current state in the AAFM area regarding conflict, diagnosis detection, and product completion. This review will guide us for defining contributions. We also discovered new forums to publish our research and to look for related work.

We discovered that there is still work to get done in the AAFM process for the conflict and diagnosis detection and product completion operations. We did not find work concerning speculative programming in AAFM operations, and neither for the product completion exists.

Chapter 5

Parallel QuickXPlain: efficient conflict detection in AAFM

Abstract

Conflict detection is used in many scenarios ranging from interactive decision making to the diagnosis of potentially faulty components or models (such as feature models or configuration knowledge bases). Conflict detection is about identifying a set of (minimal) restrictions that are causing a conflict. The efficient identification of conflicts is an important task. Junker's QuickXPlain is a divide-and-conquer based algorithm for the determination of *preferred minimal conflicts*. Especially in large-scale models, conflict detection is often a challenge in terms of limited runtime performance. In this chapter, we present a novel approach based on *speculative programming* for minimal conflict detection. We present a parallelization of QuickXPlain that improves runtime performance, specifically in large–scale models. We evaluate our approach empirically using synthesized feature models and configurations of different size obtaining a significant gain in performance. Our results open the door for new approaches leveraging speculative programming in the automated analysis of conflicting models.[†]

[†]Part of this chapter is published in the proceedings of 25th International Symposium on Methodologies for Intelligent Systems ISMIS 2020 [25]. An extended version of it is under review in the Journal of Intelligent Information Systems to appear in 2021. We have the agreement of all the authors to include the corresponding text as part of this manuscript.

Chapter 6

Parallel FastDiag: efficient minimal diagnosis in AAFM

Abstract

Diagnosis detection is used in many scenarios ranging from interactive decision making to the diagnosis of faulty components or models (such as product or feature model). Diagnosis detection is about identifying a set of (minimal) restrictions that are responsible for the unintended behavior. The efficient identification of diagnosis is an important task. Felfernig's FASTDIAG is a divide-and-conquer based algorithm for the determination of *preferred minimal diagnosis*. Especially in large-scale models, diagnosis detection is often a challenge in terms of limited runtime performance. This chapter presents a novel approach based on *speculative programming* for the preferred minimal diagnosis detection. We present a parallelization of FASTDIAG that improves runtime performance, specifically in large–scale models. We evaluate our approach empirically using synthesized feature models and configurations of different size obtaining a significant performance improvement. Our results open the door for new approaches leveraging speculative programming in the automated analysis of conflicting models.

6.1 Introduction

Diagnosis detection is used in many applications of constraint-based representations (and beyond). Examples thereof are *feature modeling* [121] and *knowledge-based configuration* [122], where users define requirements (feature selections) and diagnosis detection is in charge of figuring out the minimal sets of requirements necessary to change in order to restore consistency (if the underlying reasoning engine is not able to identify a solution). Further example applications of diagnosis detection range from *recommender systems* [123, 124], *model-based diagnosis* of hardware designs [125], the *analysis of spreadsheets* [126], to the *analysis of feature models* [4]. Especially in interactive settings, there is often a need of identifying preferred diagnosis in a short time [100, 101, 127], for example, with users of a car configurator or a camera recommender who have strict preferences regarding the upper price limit, the response times should be below one second for those users.

Diagnosis detection helps to figure out set of constraints in the knowledge base that are responsible for inconsistent behavior. FASTDIAG is such a diagnosis detection algorithm that works for constraint-based representations, description logics, and SAT solvers [22]. The algorithm is based on a divide-and-conquer approach where consistency analysis operations works on partitions of a constraint set $S = \{s_1..s_n\}$, $S_a = \{s_1..s_k\}$ and $S_b = \{s_{k+1}..s_n\}$, assuming for example, $k = \lfloor \frac{n}{2} \rfloor$. If S_b is *inconsistent*, the consideration set S can be reduced by half since S_a must not be analyzed anymore because at least one source of diagnosis exists in S_b. Based on a lexicographical constraint ordering, FASTDIAG determines one *preferred minimal diagnosis* at a time.

A diagnosis solution permits determining a *conflict resolution*, that is, to find a set of constraints that permit resolving all existing conflicts in a model (or a knowledge base). In this context, the minimality (irreducability) of diagnosis sets is important since that property allows to resolve all conflicts by simply updating the state (or deleting) of each elements in the diagnosis set. The elements to update to restore global consistency are denoted as *hitting set* (a diagnosis) and can also be determined on the basis of a hitting set directed acyclic graph [128].

Although FASTDIAG is an efficient solution in many scenarios, there are cases where computing preferred diagnosis is a challenging task, for example, when it comes to the *analysis of large-scale feature models* [129, 130, 131, 132, 133, 134]. In this chapter, we present a parallelized version of FASTDIAG based on the idea that some expensive steps of the algorithm can be previously calculated following principles of *speculative programming* [135]. Although the speculative programming principles are not new, with the appearance of modern CPUs with parallel computation capabilities, some speculative approaches are practically possible nowadays.

The major contributions of this chapter are the following. *First*, we show how to parallelize diagnosis detection on the basis of a flexible look-ahead strategy that scales depending on the number of available computing cores. *Second*, we show how to integrate the proposed approach with the standard FASTDIAG algorithm for using it in interactive constraint-based applications. *Third*, we show the applicability and improvements of our approach regarding the performance evaluations to work on large-scale feature model and configurations.

We evaluate our speculative programming version of FASTDIAG in the context of inconsistent feature model configurations of randomly generated large-scale feature model and configurations using a generation tool [136]. With our approach, we improve the performance of diagnosis detection tasks scaling with the available CPU cores, making it possible to efficiently solve more complex diagnosis detection problems. The obtained results show a significant gain in performance and general scalability of parallelized FASTDIAG with the number of available CPU cores.

6.2 Related work

Solution Search. Such as previous chapter described, an increasing need exists to further improve the performance of solution search in the increasing size and complexity of knowledge bases [129, 131, 132]. Parallelizations of algorithms in these scenarios have been implemented in different contexts. Approaches to parallelization have, for example, been proposed on the *reasoning level* [129] where the determination of a solution is based on the identification of subproblems that can be solved to some degree independently by the available cores. Due to today's multi-core CPU architectures, such parallelizations become increasingly popular in order to be able to better exploit the offered computing resources and more efficiently obtain results.

Conflict Detection. The efficient determination of minimal conflicts is a core requirement in many application settings [137]. Especially in the context of constraint-based reasoning scenarios, the identification of minimal conflict sets is frequently based on [20]. In contrast to sequential approaches [138], QUICKXPLAIN follows a divide-and-conquer based strategy that helps to significantly reduce the number of needed consistency checks. Although the algorithm is often used to support users in interactive settings, the guarantee of efficient runtimes is still often quite challenging. Conflict detection such as QUICKXPLAIN and PARALLELQUICKXPLAIN give precondition for *conflict resolution* (performed on the basis of model-based diagnosis [128, 139]). Based on the principles of speculative programming [135], we propose an algorithm that enables a parallelized conflict-independent diagnosis and thus helps to significantly improve the runtime performance of conflict resolution processes.

Conflict Resolution. Conflict resolution can be used to identify sets of (minimal) diagnoses [128, 133, 139, 140]. For instance, [137, 141] propose an approach to parallelize the computation of hitting sets (diagnoses). In their approach, Reiter's approach to model-based diagnosis is parallelized by a level-wise expansion of a breadth-first search tree with the goal of computing minimal (cardinality) diagnoses. On each level, different (minimal) conflict sets are determined in parallel, however, the determination of individual conflict sets is still a sequential process (based on QUICKXPLAIN [20]). In an extended version, [141] replace the level-wise expansion with a full parallelization of hitting set determination with additional mechanisms to ensure the minimality of proposed diagnoses. In contrast to the work presented in those papers, the work of [137] focus on the parallelization of the conflict resolution (diagnosis) step but do not offer solutions to increase the efficiency of conflict detection.

Model Analysis Operations. The Automated Analysis of Feature Models (AAFM) [142] is a computer-aided process that (1) translates feature model constraints into a logical representation (e.g., SAT or CSP), (2) applies some defined operations by the use of off-the-shelf solvers or specific programming solutions, and (3) provides feedback on specific model properties as a process result. Feature model diagnosis is an AAFM task for resolving conflicts in feature models in order to be able to come up with an error-free feature model or configuration [21]. For instance, if a user wants to select and deselect a set of features but some selections and deselections are incompatible, diagnosis (conflict resolution) would suggest which features have to be selected/deselected from the original configuration to fix the problem and convert the original inconsistent configuration into a valid one [119]. Further example applications of model-based diagnosis in the context of feature model analysis are the identification of minimal sets of constraints responsible for the existence of dead features in a feature model or the identification of minimal sets of constraints that responsible for a void feature model.

6.3 Calculating minimal diagnosis

In the remainder of this chapter, we introduce our approach to parallelized diagnosis detection on the basis of constraint-based knowledge representations [143]. First, we provide some basic definitions and examples to understand the approach.

Definition 3 *A* Configuration Tasks *can be defined as a* Constraint Satisfaction Problem *(CSP) is a triple (V,D,C) with a set of variables* $V = \{v_1..v_n\}$*, a set of domain definitions* $D = \{dom(v_1)..dom(v_n)\}$*, and a set of constraints* $C = \{c_1..c_m\}$*.*

Definition 4 *Assuming the inconsistency of C, a* conflict set *can be defined as a subset* $CS \subseteq C : CS$ *is inconsistent.* CS *is minimal if* $\neg\exists CS' : CS' \subset CS$*.*

Definition 5 *A* Configuration *for a given configuration task (V, D, C) is an instantiation* $I = \{v_1 = ins_1, v_2 = ins_2, ..., v_n = ins_n\}$ *where* $ins_k \in dom(v_k)$*.*

A configuration is consistent if the assignments in I are consistent with the $c_i \in C$, that is, no conflicts exist in I. Furthermore, a configuration is complete if all variables in V are instantiated. Finally, a configuration is valid if it is consistent and complete.

Figure 6.1 shows an example of a feature model of a Debian operating system configuration. Dark grey features are features that are selected while light grey features are features that are not desired in the configuration. A conflict exists in this configuration because it contains $gnuchess$ and $glchess$ that are alternative options of $game$. Consequently, we want to identify minimal sets of constraints ($c_i \in C$) which have to be deleted (or adapted) in order to be able to identify a solution (restore the consistency).

It is well known that a feature model can be mapped to a CSP [121]. To make it simpler and more general, in the rest of the paper we will refer directly to the derived CSP from the feature model of Figure 6.1.

Simple examples of a CSP is the following (see Examples 1.

Example 1 An example of a CSP for car configuration (inspired by the model of Figure 6.1) is the following:
V = {Debian, texteditor, bash, gui, game},
$D = \{dom(Debian) = [True, False], dom(texteditor) = [True, False], dom(bash) = [True, False], dom(gui) = [True, False], dom(game) = True, False], dom(vi) = [True, False], dom(gedit) = [True, False], dom(writer1) = [True, False], dom(gnome) =$

$[True, Falsen]$, $dom(kde) = [True, False]$, $dom(gnuchess) = [True, False]$, $dom(glchess) = [True, False]$, $dom(writer1.1) = [True, False]$, $dom(writer1.2) = [True, False]\}$,
and
$C = \{c_1 : Debian = True,\ c_2 : Debian \leftrightarrow texteditor,\ c_3 : Debian \leftrightarrow bash,\ c_4 : Debian \leftrightarrow gui,\ c_5 : game \rightarrow Debian,\ c_6 : texteditor \rightarrow (vi \vee gedit \vee writer1) \wedge vi \rightarrow texteditor \wedge gedit \rightarrow texteditor \wedge writer1 \rightarrow texteditor,\ c_7 : gui \rightarrow (gnome \vee kde) \wedge gnome \rightarrow gui \wedge kde \rightarrow gui,\ c_8 : game \rightarrow (gnuchess \vee glchess) \wedge gnuchess \rightarrow game \wedge glchess \rightarrow game \wedge \neg(glchess \wedge glchess),\ c_9 : writer1.1 \rightarrow writer1,\ c_{10} : writer1.2 \rightarrow writer1,\ c_{11} : writer \rightarrow gnome,\ c_{12} : game = True,\ c_{13} : gnuchess = True,\ c_{14} : glchess = True,\ c_{15} : texteditor = True,\ c_{16} : vi = True,\ c_{17} : gedit = True,\ c_{18} : writer1 = False,\ c_{19} : writer1.1 = False,\ c_{20} : writer1.2 = False\}$.

It is convenient to distinguish between a *consistent background knowledge B* of constraints that cannot be relaxed (in our case, $B = \{c_1..c_1 1\}$) and a *consideration set S* of relaxable constraints (in our case, requirements $C = \{c_{12}..c_{20}\}$). We focus on detecting diagnosis in the feature selections because we assume that the constraints of the feature model are consistent.

Definition 6 *A* Diagnosis Problem *for a given set of customer configuration requirements S is defined as a tuple (C_{KB}, S) where C_{KB} represents the constraints part of the configuration knowledge base (the feature model base constraints in our example) and S is the set of given customer requirements (selected features)*

Next, we define Diagnosis for a given Diagnosis Problem is the following (see Definition 7).

Definition 7 *A diagnosis for a diagnosis problem (C_{KB}, S) is a set $\Delta \subseteq C$ such that $C_{KB} \cup (C - \Delta)$ is consistent. Δ is minimal if there does not exist a diagnosis $\Delta' \subset \Delta$ such that $C_{KB} \cup (C - \Delta')$ is consistent.*

Thus, a diagnosis for our example is the following.

Example 2 *Taking into account the constraints order as the relevance order for the user selection in example 1, the preferred minimal diagnosis is $\Delta = \{c_{13}\}$, since the only conflict that exist is the set $\{c_{12}, c_{13}\}$, and c_{12} is the more relevance for the user.*

Chapter 3 presents the functions FASTDIAG (Algorithm 3.3) and FD (Algorithm 3.4) of algorithm FASTDIAG, and exemplifies their functioning on a different configuration for the same FM of Figure 6.1. Figure 6.2 depicts the execution trace of function FD of FASTDIAG for this working example.

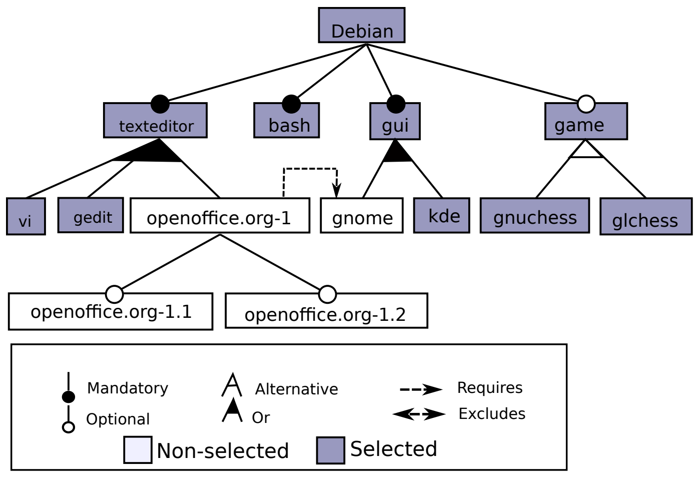


Figure 6.1: Feature model example of Debian derivatives along with a non-valid product configuration (gray features).

6.4 PARALLELFASTDIAG solution proposal

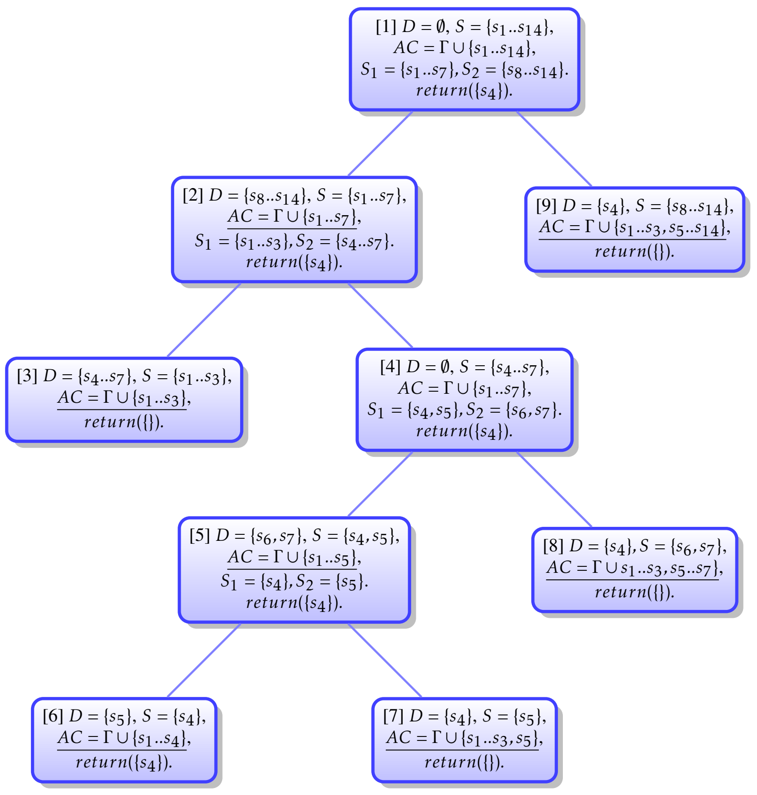
In this section, we describe PARALLELFASTDIAG, a speculative algorithm inspired by the divide-and-conquer approach of FASTDIAG [22].

Such as Felfernig et al. [2] argue, the consistency checking CC is an expensive computing step. Our approach to parallelize the consistency checks in FD substitutes the *direct* solver call CONSISTENT(AC) in FD with the activation of a lookahead function (FDGEN) in which consistency checks are not only triggered to directly provide feedback to FD requests, but also to be able to provide fast answers for consistency checks potentially relevant in upcoming states of a FD instance. We follow the principles of speculative programming [135]: we start calculating consistency checks that could be useful in the future. The advantage is that we can anticipate resource-intensive reasoning tasks. The drawback is that we use some computation resources that will be wasted if the pre-calculation is finally not used. Therefore, the challenge in this kind of technique is finding algorithms able to anticipate as many reusable calculations as possible while reducing the calculation tasks that are not reusable.

We can appreciate the following CC appear in the execution flow tracking of FASTDIAG (S, AC):

- A first CC exists over $(AC - S)$ always takes place in the function FASTDIAG. For a valid case to diagnosis, function FASTDIAG calls function FD to return future results.

  Function FD executes a CC over its current AC always when D is not empty. Then, FD does not executes a CC in its first call. Next, Diag partitions S (S =


[1] $D = \emptyset$, $S = \{s_1..s_{14}\}$,
$\underline{AC = \Gamma \cup \{s_1..s_{14}\}}$,
$S_1 = \{s_1..s_7\}, S_2 = \{s_8..s_{14}\}$.
$return(\{s_4\})$.
[2] $D = \{s_8..s_{14}\}$, $S = \{s_1..s_7\}$,
$\underline{AC = \Gamma \cup \{s_1..s_7\}}$,
$S_1 = \{s_1..s_3\}, S_2 = \{s_4..s_7\}$.
$return(\{s_4\})$.
[9] $D = \{s_4\}$, $S = \{s_8..s_{14}\}$,
$\underline{AC = \Gamma \cup \{s_1..s_3, s_5..s_{14}\}}$,
$return(\{\})$.
[3] $D = \{s_4..s_7\}$, $S = \{s_1..s_3\}$,
$\underline{AC = \Gamma \cup \{s_1..s_3\}}$,
$return(\{\})$.
[4] $D = \emptyset$, $S = \{s_4..s_7\}$,
$AC = \Gamma \cup \{s_1..s_7\}$,
$S_1 = \{s_4, s_5\}, S_2 = \{s_6, s_7\}$.
$return(\{s_4\})$.
[5] $D = \{s_6, s_7\}$, $S = \{s_4, s_5\}$,
$\underline{AC = \Gamma \cup \{s_1..s_5\}}$,
$S_1 = \{s_4\}, S_2 = \{s_5\}$.
$return(\{s_4\})$.
[8] $D = \{s_4\}, S = \{s_6, s_7\}$,
$\underline{AC = \Gamma \cup s_1..s_3, s_5..s_7\}}$,
$return(\{\})$.
[6] $D = \{s_5\}$, $S = \{s_4\}$,
$\underline{AC = \Gamma \cup \{s_1..s_4\}}$,
$return(\{s_4\})$.
[7] $D = \{s_4\}$, $S = \{s_5\}$,
$\underline{AC = \Gamma \cup \{s_1..s_3, s_5\}}$,
$return(\{\})$.

For $S = \{s_1..s_{14}\}$ and $AC = \Gamma \cup \{s_1..s_{14}\}$ assuming a minimal preferred diagnosis set $\Delta = \{s_4\}$. Underlined ACs denote FD *consistency checks. For example, in the incarnation [2] of the* FD *function, the consistency check activated is $\Gamma \cup \{s_1..s_7\}$.*

Figure 6.2: FD execution trace example.

S_1 U S_2) to execute two recursive calls: Δ_1 = FD(D=S_2, S=S_1, AC=AC - S_2) and Δ_2 = FD(D=Δ_2, S=S_2, AC=AC - Δ_1).

- A second CC is over (AC - S_2) in the second call of the function FD; that is, FD executes a CC over AC less the less preferred partition of S. That CC occurs on a non-unary S. Next, we detail the execution flow after the second CC in the function FD:
  - Because D is not empty, a false CC means that the current D (S_2) does not contain a complete diagnosis. If the current S (S_1) is not minimal, then FD partitions the current S ($S_1 = S_{11} \cup S_{12}$) for new recursive execution of that function. The first recursive execution of FD reviews if the most preferred partition of the current S (S_{12}) contains the preferred

diagnosis for the new AC (AC - S_2 - S_{12}), that is, a new CC will take place on the new AC(AC - S_2 - S_{12}).

– On the other hand, D is not empty, and the second CC returns a true value, that is, S_2 contains a complete explanation and FD returns an empty set. Then, the next recursive call of FD (FD($D = \emptyset$, $S = S_2$, AC)) will not execute a CC. For a non-unary S (S_2), FD partitions its current S ($S_2 = S_{21} \cup S_{22}$), and executes itself again twice. We consider the first recursive call (Δ_1 = FD($D = S_{22}$, $S = S_{21}$, $AC = AC - S_{22}$)) that runs a third CC on (AC - $S22$).

This analysis shows us that from a new CC, we can speculate three possible CC. This reasoning constitutes the basement of our speculative-programming solution. We can define a max level of speculation for recursively obtaining additional CC concerning the execution flow of FD until reaching a maximum level of speculation.

In a one-thread scenario, a speculative FASTDIAG solution will be more expensive than the original FASTDIAG. If a new CC value were necessary, we would calculate it. At the same time, if we have not reached the max level of speculations, possibly speculate on the next three described CC for the current speculation level. We store each CC in a lookup table. Hence, each time a new CC occurs, we first review if that CC exists in the table for getting the stored value or calculating it.

This reasoning permits us to define PARALLELFASTDIAG as a speculative solution for diagnosis that reduces the number of needed consistency checks and improves the diagnosis computing performance.

In the parallelized variant of PARALLELFASTDIAG that we propose, consistency checking is activated by FD with CONSISTENT(D, S, AC) (see Algorithm 6.1). This also activates FDGEN (see Algorithm 6.2) that starts to generate and trigger (in a parallelized fashion) further consistency checks that might be of relevance in upcoming FD phases. For describing FDGEN, we employ a two-level *ordered set* notation which requires, for example, to embed the FD D into $\{D\}$, S into $\{S\}$, and AC into $\{AC\}$. In FDGEN, D, S, and AC are interpreted as *ordered sets*.

Algorithm 6.1 CONSISTENT(D, S, AC):*Boolean*

```
1: if $\neg$EXISTSCONSISTENCYCHECK($AC$) then
2:     FDGEN($\{D\}, \{S\}, \{AC \cup D\}, \{D\}, 0$)
3: end if
4: return(LOOKUP($AC$))
```

FDGEN-generated consistency checking tasks are stored in a LOOKUP table (see, e.g., Table 6.1). Thus, in PARALLELFASTDIAG, FD has to activate the consistency check with CONSISTENT(D,S,AC). In contrast to the original FASTDIAG approach, the consistency check function requires D and S as additional parameters to conduct inferences about necessary future consistency checks. While in the standard FD version, D and AC are related sets, that is, D is a set of elements omitted from AC ($AC \cap D = \emptyset$), we assume that AC contains D in FDGEN.

node-id	constraint set	consistent
1	$\{\Gamma \cup \{s_1..s_7\}\}$	$false$
1.1	$\{\Gamma \cup \{s_1..s_3\}\}$	$true$
1.1.1	$\{\Gamma \cup \{s_1\}\}$	-
1.1.2	$\{\Gamma \cup \{s_4..s_7\}\}$	-
1.2	$\{\Gamma \cup \{s_8..s_{14}\}\}$	$false$
1.2.1	$\{\Gamma \cup \{s_8..s_{10}\}\}$	-

Table 6.1: LOOKUP table indicating the consistency of individual constraint sets for $lmax = 3$.

For Table 6.1, the consistency checking tasks have been generated by ADDCC in the FDGEN function (see Figure 6.3). Consistency checking are executed in parallel. The '-' entry for the constraints of node-id 1.1.1, node-id 1.1.2 and node-id 1.2.1 indicates that the corresponding consistency check is still ongoing or has not been started up to now. Algorithm 6.1 uses LOOKUP to test the consistency of a constraint set.

The FDGEN function (see Algorithm 6.2) predicts future potentially relevant consistency checks needed by FD and activates individual consistency checking tasks in an asynchronous fashion using the ADDCC (*add consistency check*) function. The ADDCC function triggers an asynchronous service that is in charge of adding consistency checks (parameter of ADDCC) to a LOOKUP table and issuing the corresponding solver calls. The global parameter $lmax$ is used to define the maximum search depth of one activation of FDGEN.

In FDGEN, $|f(X)|$ denotes the number of constraints c_i in X (it is introduced due to the subset structure of parameters of FDGEN). Furthermore, SPLIT(S,S_a,S_b) splits S at position $\lfloor\frac{|S|}{2}\rfloor$ if $|S| > 1$ or it splits S_1 (the first element of S) at position $\lfloor\frac{|S_1|}{2}\rfloor$ if $|S| = 1$ and $|S_1| > 1$ into S_a and S_b. Otherwise, no split is needed (S_1 is a singleton).

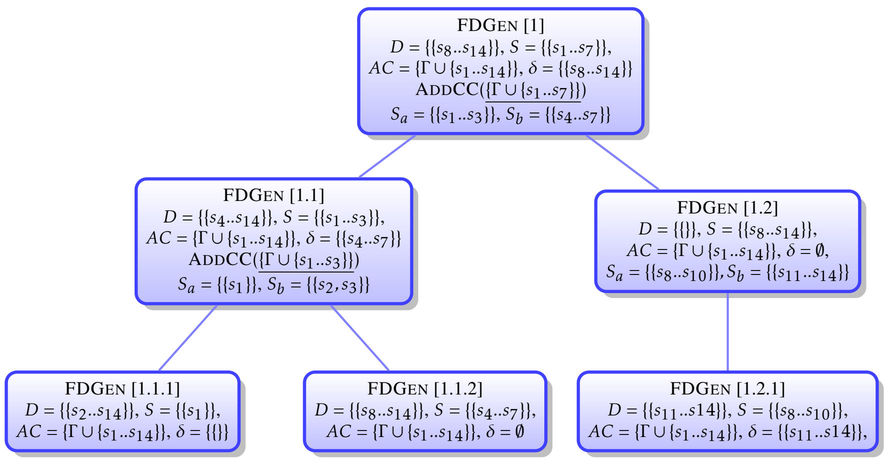
The first inner condition of FDGEN ($|f(\delta)| > 0$) generates a consistency check if this is needed. A consistency check is needed, if D gets extended from S or a singleton S has been identified which was removed from AC. The function ADDCC is used to add consistency check tasks which can then be executed asynchronously in a parallelized fashion. Thus, a consistency check in the LOOKUP table can be easily identified by an ordered constraint set that has also been used as parameter of ADDCC, for example, ADDCC($\{\Gamma \cup \{s_1..s_{14}\}\} - \{\{s_4..s_7\}, \{s_8, s_{14}\}\}$) results in the LOOKUP table enty $\Gamma \cup \{s_1..s_3\}$ which is internally represented with 123.

Algorithm 6.2 FDGEN(D, S, AC, l)

$D = \{d_1..d_r\}$ ... *consideration set* ($subsets\ D_\alpha$)
$S = \{s_1..s_n\}$ ... *set* ($subsets\ S_\beta$) *to consider*
$AC = \Gamma \cup S$ ... *base knowledge and knowledge to consider* ($subsets\ AC_\gamma$)
$\delta = \{\delta_1..\delta_p\}$ ... *to be checked* ($subsets\ \delta_\pi$)
l ... *current lookahead depth*

```
1:  if $l < lmax$ then
2:      if $|f(\delta)| > 0$ then
3:          ADDCC($AC - D$)
4:      end if
5:      {$AC - D$ assumed inconsistent}
6:      if $|f(S)| = 1 \wedge |f(D)| > 0$ then
7:          FDGEN($\emptyset, D, AC - \{S_1\}, \{S_1\}, l + 1$)
8:      else if $|f(S)| > 1$ then
9:          SPLIT($S, S_a, S_b$)
10:         FDGEN($S_b \cup D, S_a, AC, S_b, l + 1$)
11:     end if
12:     {$AC - D$ assumed consistent}
13:     if $|f(D)| > 0 \wedge |f(\delta)| > 0$ then
14:         FDGEN($D - \{D_1\}, \{D_1\}, AC, \emptyset, l + 1$)
15:     end if
16: end if
```

If $(AC - D)$ is assumed to be *inconsistent*, additional elements from S have to be included such that an inconsistent state can be generated (which is needed for identifying a minimal conflict). This extension of D can be achieved by dividing S (if $|f(S)| > 1$, i.e., more than one constraint is contained in S) into two separate sets S_a and S_b and to add S_b to D. If $|f(S)| = 1$, this singleton can be removed from AC, responsible for no collecting constraints that have been identified as part of the minimal diagnosis. That is the case due to the invariant property $(AC - D)$ is inconsistent. If S contains only one constraint, i.e., S_1 is a singleton, it is part of the diagnosis set. If $(AC - D)$ is assumed to be *consistent*, it can be reduced, and at least one conflict element will be identified in the previously added D. If δ does not contain an element, no further recursive calls are needed since $(D - D_1)$ has already been checked previously.


FDGEN [1]
$D = \{\{s_8..s_{14}\}\}, S = \{\{s_1..s_7\}\},$
$AC = \{\Gamma \cup \{s_1..s_{14}\}\}, \delta = \{\{s_8..s_{14}\}\}$
ADDCC($\underline{\{\Gamma \cup \{s_1..s_7\}\}}$)
$S_a = \{\{s_1..s_3\}\}, S_b = \{\{s_4..s_7\}\}$
FDGEN [1.1]
$D = \{\{s_4..s_{14}\}\}, S = \{\{s_1..s_3\}\},$
$AC = \{\Gamma \cup \{s_1..s_{14}\}\}, \delta = \{\{s_4..s_7\}\}$
ADDCC($\underline{\{\Gamma \cup \{s_1..s_3\}\}}$)
$S_a = \{\{s_1\}\}, S_b = \{\{s_2, s_3\}\}$
FDGEN [1.2]
$D = \{\{\}\}, S = \{\{s_8..s_{14}\}\},$
$AC = \{\Gamma \cup \{s_1..s_{14}\}\}, \delta = \emptyset,$
$S_a = \{\{s_8..s_{10}\}\}, S_b = \{\{s_{11}..s_{14}\}\}$
FDGEN [1.1.1]
$D = \{\{s_2..s_{14}\}\}, S = \{\{s_1\}\},$
$AC = \{\Gamma \cup \{s_1..s_{14}\}\}, \delta = \{\{\}\}$
FDGEN [1.1.2]
$D = \{\{s_8..s_{14}\}\}, S = \{\{s_4..s_7\}\},$
$AC = \{\Gamma \cup \{s_1..s_{14}\}\}, \delta = \emptyset$
FDGEN [1.2.1]
$D = \{\{s_{11}..s14\}\}, S = \{\{s_8..s_{10}\}\},$
$AC = \{\Gamma \cup \{s_1..s_{14}\}\}, \delta = \{\{s_{11}..s14\}\},$

For $D = \emptyset$, $S = \{\{s_1..s_{14}\}\}$, $AC = \{\Gamma \cup \{s_1..s_{14}\}\}$, $\delta = \emptyset$, and $lmax = 3$. The consistency checks $\{\Gamma \cup \{s_1..s_7\}\}$ and $\{\Gamma \cup \{s_1..s_3\}\}$ (flattened list generated by ADDCC*) can be used by the* FASTDIAG *instance of Figure 6.2. The* FDGEN *nodes {[1],[1.1],[1.1.2]} represent the first part of the* FASTDIAG *search path in Figure 6.2.*

Figure 6.3: FDGEN execution trace example.

	FOLLOW SETS			
COND.	D	S	AC	δ
$\lvert f(S)\rvert = 1$	$\emptyset$	cD	$AC - \{S_1\}$	$\{S_1\}$
$\lvert f(S)\rvert > 1$	$S_b \cup D$	S_a	AC	S_b
$\lvert f(D)\rvert > 0$	$D - \{D_1\}$	$\{D_1\}$	AC	$\emptyset$

Depending on the assumption about the consistency of $AC - D$, the follow-up activations of FDGEN *have to be parameterized differently.*

Table 6.2: FOLLOW sets of the FDGEN function.

The FDGEN function (Algorithm 6.2) is based on the idea of issuing recursive calls and adapting the parameters of the calls depending on the two possible situations 1) *non-consistent*$(AC - D)$ and 2) *consistent*$(AC - D)$. In Table 6.2, the different parameter settings are shown in terms of FOLLOW sets representing the settings of D, S, and AC in the next activation of FDGEN.

The most remarkable achievements of our solution proposal are i) PARALLELFASTDIAG preserves the base functions and results of FASTDIAG, ii) PARALLELFASTDIAG correctly can be applied in other diagnosis scenarios.

6.5 Analysis

6.5.1 Complexity analysis

FD complexity. Assuming a splitting $d = \lfloor \frac{n}{2} \rfloor$ of $S = \{s_1..s_n\}$, the worst-case time complexity of FD in terms of the number of consistency checks needed for calculating one minimal diagnosis is $2d \times log_2(\frac{n}{d}) + 2d$ where d is the minimal diagnosis set size and n represents the underlying number of constraints [22]. The runtime performance of the underlying algorithms must be optimized because consistency checks are the most time-consuming part of diagnosis detection.

FDGEN *complexity*. The number (nc) of different consistency checks that could be identified with FDGEN for an inconsistent configuration S of n constraints is represented by Formula 6.1.

$$nc(S) = \binom{n}{1} + \binom{n}{2} + ... + \binom{n}{n} \tag{6.1}$$

The upper bound of the space complexity in terms of recursive FDGEN calls for $lmax = n$ is in the worst case $2^{n-1} - 1$. Due to the combinatorial explosion, only those solutions make sense that scale $lmax$ depending on the available computing cores (see the reported evaluation results).

If traditional FASTDIAG is applied, only sequential consistency checks can be performed. The approach presented in this chapter is more flexible since $\#processors$ consistency checks can be performed in parallel. Assuming a maximum FDGEN search depth of $lmax = 4$, the maximum number of generated consistency checks is $\frac{2^{lmax-1}}{2}$ due to the binary structure of the search tree, i.e., 4 in our example. Out of these 4 checks, a maximum of 3 will be relevant to FD; the remaining ones are irrelevant for identifying the conflict in the current FD session. Thus, the upper bound of relevant consistency checks generated by FDGEN is $lmax$, i.e., one per FDGEN search level (see the outer left search path in Figure 6.3), the minimum number of relevant consistency checks is $\lceil \frac{lmax}{2} \rceil$, i.e., 2 in our example. Assuming $\#processors = 16$, our approach can theoretically achieve a performance boost of factor 3 since 3 relevant consistency checks can be performed in parallel. It is important to mention, that within these upper and lower bounds, a performance improvement due to the integration of FDGEN can be guaranteed independently of the knowledge base.

Termination of FDGEN. If $lmax = n$, recursive calls of FDGEN stop at level $n-1$. In each recursive step, based on the inconsistency invariant between $S \cup D \cup AC$, *either* 1) S is reduced to S_a and D gets extended with S_b (if $AC - D$ is inconsistent) or to D if S is a singleton, *or* 2) D is further reduced (if $AC - D$ is consistent). In the second case, the constraints in S are not relevant anymore, since a diagnosis can already be found in D (which is inconsistent with AC).

FD-*conformance of* FDGEN. FDGEN correctly predicts FD consistency checks. It follows exactly the criteria of FD. If $|f(S)| = 1$, i.e., S includes only one constraint s_α, $AC - D$ is inconsistent and - as a consequence of the inconsistency invariant -

s_α is a diagnosis element and therefore has to be removed from AC. If $|f(S)| > 1$, $AC - (S_b \cup D)$ has to be checked, since $AC - D$ is inconsistent and by adding S_b we follow the goal of restoring the consistency of $AC - D$. Finally, if the issued consistency checks $AC - D$ is consistent, a diagnosis has already been identified. If $AC - D$ is consistent, no check has to be issued since $AC - (\{D_1\} \cup D)$ has already been checked in the previous FDGEN call considered inconsistent. Thus, FDGEN takes into account all FD states that can occur in the next step, and exactly one of the generated consistency checks (if needed) will be relevant for FD. Finally, the generated consistency checks are irredundant since each check is only generated if D contains new constraints from S or a new constraint (singleton S) is removed from AC.

6.5.2 Runtime analysis

Implementation details. We conducted the experimentation based on the implementation in *Python3* of FASTDIAG and PARALLELFASTDIAG. We used the *multiprocessing Python* package for running parallel tasks. We used Sat4J [144] for representing our test knowledge bases and conducting the corresponding consistency checks since it is one of the most used solvers integrated in many software (product line) engineering tools such as FeatureIDE [10], FaMa framework [145], FAMILIAR [12] among others [8, 146**?**]. Nonetheless, we could use any other technology able for writing and reasoning on AAFM solutions.

Even though we showed the theoretical superiority of PARALLELFASTDIAG compared to sequential FASTDIAG (Section 6.5.1), some implementation details can affect that performance. First, there will be a delay when running parallel algorithms because extra time can be necessary to the orchestration of threads. Second, our solution proposal considers the use of set operations such as *Union* or *Difference* that can take extra time depending on the size of AC, D, and S.

Execution environment. We conducted all experiments using an AMD EPYC 7000 machine equipped with a CPU with 16 cores and 2.50GHz. It had 64 GB of RAM.

Knowledge base Characteristics. For evaluation purposes, we generated configuration knowledge bases (feature models) from the publicly available BETTY tool suite [136] that allows systematic testing for consistency checking and diagnosis approaches for knowledge bases and configurations. The knowledge base instances (represented as background knowledge AC in FASTDIAG) for our evaluation were around 1.000 binary variables (derived from the 1.000 features used). Those base knowledge also varied in terms of the number of included constraints depending on the different feature relationships and derived clauses (around 1,600 SAT clauses in the generated CNF files). Based on these knowledge bases, we randomly generated product configurations ($s_i \in S$) that covered 10% of the variables included in the knowledge base. We also shuffled S to get different constraints orders because that can affect the number of consistency checks needed, as explained in Section 6.5.1. We generated and analyzed conflict sets of different cardinality (see Table 6.3). We repeated each evaluation setting 3× to avoid measuring biases due to side effects in thread execution.

Evaluation purpose. We wanted to analyze if PARALLELFASTDIAG performs better than FASTDIAG. We conjecture that PARALLELFASTDIAG outperforms the FASTDIAG in general. Improvement increases with the number of available cores (represented in our algorithm by $lmax$) and the difficulty of the problem (represented by the knowledge base AC, and the corresponding conflicting requirements S). The difficulty of the problem can be affected by different aspects. One of them is the diagnosis cardinality in AC: the number of elements necessary for a state update to get a consistent status (a number determined for the size and number of conflict sets in AC).

lmax	Conflict Cardinality				
	1	2	4	8	16
1	61,14	62,47	62,48	66,44	66,12
2	56,14	35,78	56,89	57,10	70,73
3	49,78	44,16	45,07	48,54	**50,14**
4	48,78	46,73	46,69	**47,89**	54,34
5	**43,68**	**44,13**	**43,68**	48,25	50,55

lmax=1 is equivalent to sequential FASTDIAG. *Each cell follows a heat map colouring: the darker the slower. In bold the cells with faster time for a given conflict cardinality.*

Table 6.3: Avg. runtime (in $msec$) of FD (lmax=1) and parallelized FD (lmax>1) for determining preferred diagnosis.

Results. Table 6.3 shows a summary of the performance and analysis results of FASTDIAG and FDGEN. On an average, the runtime needed by standard FASTDIAG ($lmax = 1$) to identify a preferred minimal diagnosis for conflict of cardinality 16 is $23,54\%$ slower compared to a parallelized solution for the same purpose based on FDGEN ($lmax = 5$). In Table 6.3, each entry represents the average runtime in $msec$ for all knowledge bases with a conflict set of cardinality n, where the same set of knowledge bases has been evaluated for $lmax$ sizes 1–5 ($lmax = 1$ corresponds to the usage of standard FD without FDGEN integration).

We can observe that with an increasing $lmax$, the performance improvement of FD increases with a few exceptions: the solution for four threads is the best for models with eight conflicts, and the solution for three threads is the best for models with sixteen conflicts. A deterioration can exist with $lmax = 4$ and $lmax = 5$ because the number of pre-generated consistency checks starts to exceed the number of physically available processors. The obtained results support our theoretical analysis of FDGEN, taking into account the overheads for managing the consistency checks in parallel. Figure 6.4 illustrates the performance results of Table 6.3. The performance improvement of PARALLELFASTDIAG presents a scalability tendency even though it is not as notorious as for PARALLELQUICKXPLAIN. After reviewing the results, some conflicts are solvable by updating only one or a few constraints. Then, finding a conflict set with various conflicts can require more computation.

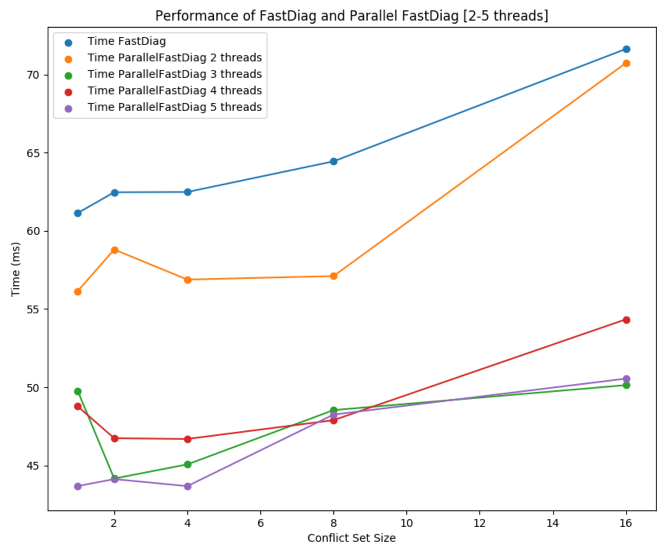


Figure 6.4: Performance of FASTDIAG vs PARALLELFASTDIAG with 2 to 5 threads

6.6 Summary

This chapter introduced a parallelized variant of the FASTDIAG algorithm used to diagnose inconsistent constraints set, such as in conflict product configuration of FM instances. Example applications are the model-based diagnosis of software and hardware designs and the diagnosis of inconsistent user requirements in configuration and recommender applications. Our parallelized variant of FASTDIAG helps exploit multi-core architectures and provide efficient preferred diagnosis detection, especially when dealing with complex knowledge bases. With this approach, we help boost various knowledge-based applications' performance and make them more accessible, especially in interactive settings.

Chapter 7

Minimal completion of products as a diagnosis task in AAFM

Abstract

The completion of partial configurations might represent an expensive computational task. Existing solutions, such as those which use modern constraint satisfaction solvers, perform a complete search, making them unsuitable on large-scale configurations. In this work, we propose an approach to define the completion of a partial configuration like a diagnosis task to solve it by applying the FASTDIAG algorithm, an efficient solution for minimal diagnosis (updates) in the analyzed partial configuration. We evaluate our proposed method in the completion of partial configurations of random medium and large-size features models, and in the completion of partial configurations of a feature model of an adapted version of the Ubuntu Xenial OS. Our experimental analysis shows remarkable improvements in our solution regarding the use of classical CSP-based approaches for the same tasks. [†]

[†]Part of this chapter is published in the proceedings of Industrial Session of the 25th International Symposium on Methodologies for Intelligent Systems ISMIS 2020 [25]. We have the agreement of all the authors to include the corresponding text as part of this manuscript.

Part IV

Final remarks

Chapter 8

Conclusiones and future work

8.1 Conclusions

In this dissertation we have shown that:

Improving the computing efficiency of existing conflict detection, diagnosis, and product completion solutions in the automated analysis of large-scale variability models' configuration are possible tasks.

We developed more efficient solutions than the existing ones to detect minimal conflicts between user's preferences and base knowledge, diagnose or signal a minimal set of updates for getting consistent user's preferences, and complete partial products in large-scale configuration scenarios. We review existing AUTOMATED ANALYSIS OF FEATURE MODEL product configuration solutions and algorithmic techniques to improve the first ones' computing efficiency by using additional computing resources. We compared our solutions' computing efficiency and existing ones for getting results in the same tasks. Then, our solutions resulted in being more efficient, and that represents our main contribution.

8.1.1 Discussion and open challenges

In the Chapter 1, we detected and enumerated the research goals we were willing to address in this thesis document. Table 8.1 shows the chapters where we target each research goal with the contributions we have already published. Also, in the next paragraph, we will go through each contribution to explain how we addressed them.

This dissertation's main contributions were the PARALLELQUICKXPLAIN and PARALLELFASTDIAG solutions for the minimal conflict and diagnosis detection, respectively, by applying speculative computation, and a solution for the product completion by diagnosis. Specifically, in this dissertation, we have shown that:

PARALLELQUICKXPLAIN is a speculative programming solution that permits computation improvements of the QUICKXPLAIN algorithm for conflict

detection. Although we applied our solution on FM configurations, PARALLELQUICKXPLAIN can analyze other configuration scenarios in reasoning solving approaches such as SAT and CSP..

We recognized that conflict detection is a base step for solving configuration issues. We found that QUICKXPLAIN represents an efficient solution for detecting minimal preferred conflict. Even though QUICKXPLAIN uses an efficient divide-and-conquer algorithmic approach, that algorithm takes a long time to analyze large-scale FM and configurations. QUICKXPLAIN cannot use computing resources, such as multiple cores for parallel computing, for its sequential nature. Hence, we analyzed how to parallelize QUICKXPLAIN to develop a more efficient solution for detecting conflicts in large-scale configuration scenarios as our first research goal. Our analysis found a costly operation step that uses data from the previous executions in the QUICKXPLAIN functioning. We pre-calculate that operation by applying speculative computation to look for improvements. Thus, PARALLELQUICKXPLAIN was born. The obtained results validated the efficiency of PARALLELQUICKXPLAIN regarding traditional QUICKXPLAIN for the analysis of large-scale FM and configurations.

PARALLELFASTDIAG is a speculative programming solution that permits computation improvements of the FASTDIAG algorithm for diagnosis detection. We applied our solution over FM configurations, and PARALLELFASTDIAG can work on any system able to represent in reasoning solving approaches such as SAT and CSP.

We recognized that getting a preferred minimal diagnosis permits getting the necessary updates on the current configuration for obtaining a valid one. Even though we found that FASTDIAG represents an efficient solution for detecting minimal preferred diagnosis for using an efficient divide-and-conquer algorithmic approach, FASTDIAG takes a long time to analyze large-scale FM and configurations. FASTDIAG cannot use computing resources, such as multiple cores for parallel computing, for its sequential nature. Hence, we analyzed how to parallelize FASTDIAG to develop a more efficient solution for diagnosis in large-scale configuration scenarios as our second research goal. Like in the analysis of QUICKXPLAIN, our analysis found a costly operation step that uses data from the previous executions in the FASTDIAG functioning. We pre-calculate that operation by applying speculative computation to look for improvements. Thus, PARALLELFASTDIAG was born. The obtained results validated the efficiency of PARALLELFASTDIAG regarding traditional FASTDIAG for the analysis of large-scale FM and configurations.

We appreciated the completion of product configuration as a diagnosis problem for the use of existing diagnosis solutions such as FASTDIAG. Then, we were able to apply efficient computing solutions such as PARALLELFASTDIAG.

We recognized that the product completion is a necessary task for assisting with large-scale configurations. We found that reasoning tools commonly perform product completion tasks using non-efficient computing approaches without guaranteeing a completion using preferred features. This dissertation gave the fundamental steps to appreciate product completion as a diagnosis task to apply diagnosis solutions and analyze any improvement. Hence, the third goal of this thesis was to develop a solution for product completion by diagnosis. Since FASTDIAG is an efficient algorithm for the diagnosis, we planned to apply FASTDIAG for efficient and preferred product completion tasks. The obtained results validate the efficiency of our product completion by diagnosis approach regarding the traditional use of reasoning tools.

Table 8.1 shows the divisions in this thesis in which we target each research goal with the contributions we have already published. Concerning our second research goal, Table 8.1 lists a few papers that consider the problem of diagnosis using the classical solution FASTDIAG. We are currently working on an article of PARALLELFASTDIAG.

Research goal	Thesis division	Published contribution
Conflict detection	Sec. 3.2 - Ch. 5	[25] [32] [147]
Diagnosis	Sec. 3.3 - Ch. 6	[116] [32]
Product completion	Sec. 3.4 - Ch. 7	[31]

Table 8.1: Links between research goals, chapters and publications.

By addressing the research goals of Table 8.1, we can now assert that applying speculative computing for improving the efficiency of solutions in the automated analysis of variability intensive system models is entirely possible. We can also assert that we can solve problems more efficiently by addressing them with a different perspective for applying efficient solutions. Both approaches can then permit a cost reduction and increment of efficiency in software engineering activities such as configuration, testing, and evolution of large-scale variability models.

8.2 Future work

In this section we show the future work that arises from the contributions presented in this dissertation. We divided it into two areas: first shows the future work derived from the use of speculative solutions for variability analysis of large-scale models; and second, regarding the use of diagnosis solutions to face other issues in variability models.

8.2.1 Speculative programming

Concerning speculative programming, we plan to develop a solution for a non-online pre-compilation of consistency checks for a more efficient computation of conflict detection, minimal preferred diagnosis, and product configuration. Then, each time these solutions work on a pre-calculated consistency check, the response time cost would be in the worst case the cost of search in the set of previously pre-calculated consistency checks. Each time these solutions work on a new base consistency checking, they would calculate and store it for future configurations. The response time cost would be in the worst case the cost of search in the set of previously pre-calculated consistency checks plus the consistency check cost. Because each consistency check search plays a relevant role in the cost of this solution approach, we need to adequately organize the searched data for efficiently search on it.

Regarding the extension of existing solutions by the use of speculative programming, we plan to extend MERGEXPLAIN [148, 149] to evaluate and highlight the potential improvements for computing multiple conflicts for diagnosis tasks. We plan also to apply speculative computation to produce PARALLEL FLEXDIAG, an extension of FLEXDIAG [100], to evaluate and highlight the potential improvements for the computation of anytime diagnosis in large-scale variability models by the combination of different multi-thread and granularity parameter values. Because the third contribution of this thesis is the product completion by diagnosis, we plan to apply PARALLELFASTDIAG on the product completion of large-scale configurations scenarios to evaluate and highlight each potential computation improvements. We also plan to adapt FASTDIAG and PARALLELFASTDIAG to work on large-scale fragmented feature models for the product completion and analyzing the computing efficiency of their results.

8.2.2 Using parallel diagnosis in other variability issues

Considering the work of Felfernig et al. [21] that describe the use of diagnosis solutions for solving different AUTOMATED ANALYSIS OF FEATURE MODEL issues, we will apply PARALLELFASTDIAG for other analysis operations and evaluate the computing results regarding existing AUTOMATED ANALYSIS OF FEATURE MODEL solutions. For example, we will compare the performance of PARALLELFASTDIAG regarding one of our previous works for enumerating valid product configuration (PAVIA [24]). Regarding the product discovery of large-scale feature models, we will adapt PARALLELFASTDIAG for the product completion of fragmented feature models [19] to analyze its differences regarding traditional PARALLELFASTDIAG. Considering the obtention of anytime diagnosis as a variability issue, we will compare the computing performance and precision of the results of applying FLEXDIAG [100] and our PARALLELFASTDIAG solution in the diagnosis of large-scale feature models and configurations. We expect to obtain results of anytime diagnosis more efficiently using PARALLELFASTDIAG without tradeoffs between diagnosis quality (e.g., minimality) and diagnostic search performance.

Part V

Appendix

Appendix A

Exploiting the enumeration of all feature model configurations

A new perspective with distributed computing

Abstract

Feature models are widely used to encode the configurations of a software product line in terms of mandatory, optional and exclusive features as well as propositional constraints over the features. Numerous computationally expensive procedures have been developed to model check, test, configure, debug, or compute relevant information of feature models. In this paper we explore the possible improvement of relying on the enumeration of all configurations when performing automated analysis operations. The key idea is to pre-compile configurations so that reasoning operations (queries and transformations) can then be performed in polytime. We tackle the challenge of how to scale the existing enumeration techniques. We show that the use of distributed computing techniques might offer practical solutions to previously unsolvable problems and opens new perspectives for the automated analysis of software product lines. [†]

[†]Part of this chapter is published in the proceedings of 20th International Systems and Software Product Line Conference SPLC 2016 [24]. We have the agreement of all the authors to include the corresponding text as part of this manuscript.

Appendix B

Functional testing of conflict detection and diagnosis tools in feature model configuration

Abstract

In configuration scenarios such as the product configuration of a FM, conflict detection and diagnosis operations are high-value tasks for helping create consistent products. Conflict detection and diagnosis are about identifying a set of (minimal) restrictions responsible for conflicts and those restrictions that would enable consistent configuration upon update. Ensuring the quality of product configuration stimulates the development of new solutions. Still, the lack of specific testing mechanisms is becoming a significant obstacle that affects the quality and reliability of new solutions. In this paper, we present FAMAPRODCONF TESTSUITE, a set of implementation-independent test cases to validate the functionality of conflict detection and diagnosis solutions in FM product configuration. FAMAPRODCONF TESTSUITE is an efficient and handy mechanism to help develop new solutions and detect faults in order to solve the found issues and improve the quality of solutions. As an effectiveness proof, we evaluated the test suite using existing solutions for the conflict detection and diagnosis of product configuration: QUICKXPLAIN and FASTDIAG. Obtained results validate our approach for testing conflict detection and diagnosis of FM product configuration solutions. [†]

[†]Part of this chapter is published in the proceedings of 22th International Workshop on Configuration CONFWS 2020 [32]. We have the agreement of all the authors to include the corresponding text as part of this manuscript.

Bibliography

[1] A. Felfernig, G. Friedrich, and D. Jannach. Conceptual modeling for configuration of mass-customizable products. *Artif. Intell. Eng.*, 15:165–176, 2001. 12

[2] A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen. *Knowledge-based Configuration: From Research to Business Cases*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1 edition, 2014. 12, 13, 26, 36, 64

[3] Ubuntu. 16.04.6 lts (xenial xerus). `http://cl.releases.ubuntu.com/16.04/`, 2019. Accessed: 2019-02-01. 12

[4] D. Benavides, S. Segura, and A. Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6):615–636, September 2010. 12, 14, 25, 26, 27, 28, 29, 33, 45, 59

[5] J. A. Galindo, F. Roos-Frantz, D. Benavides, A. Ruiz-Cortés, and J. García-Galán. Automated analysis of diverse variability models with tool support. In *XIX Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2014)*, pages 160–168, 01 2014. 12, 44, 45

[6] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, Carnegie-Mellon Uni-

versity Software Engineering Institute, November 1990. 12, 25

[7] S. Apel, D. Batory, C. Kstner, and G. Saake. *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer Publishing Company, Incorporated, 2013. 13, 14

[8] D. Batory. Feature models, grammars, and propositional formulas. In *Proceedings of the 9th International Conference on Software Product Lines*, SPLC'05, pages 7–20, Berlin, Heidelberg, 2005. Springer-Verlag. 13, 26, 71

[9] M. Mendonca, M. Branco, and D. Cowan. S.p.l.o.t.: Software product lines online tools. In *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*, OOPSLA '09, page 761–762, New York, NY, USA, 2009. Association for Computing Machinery. 13

[10] C. Kastner, T. Thum, G. Saake, J. Feigenspan, T. Leich, F. Wielgorz, and S. Apel. Featureide: A tool framework for feature-oriented software development. In *2009 IEEE 31st International Conference on Software Engineering*, pages 611–614. IEEE, 2009. 13, 71

[11] How to prevent and fix package dependency errors in ubuntu. `https://www.isa.us.es/fama/`. Accessed: 2020-10-02. 13

[12] M. Acher, P. Collet, P. Lahire, and R. B. France. FAMILIAR: A domain-specific language for large scale management of feature models. *Sci. Comput. Program.*, 78(6):657–681, 2013. 13, 50, 71

[13] M. Galster. Variability-intensive software systems: Product lines and beyond. In *Proceedings of the 13th Inter-*

national Workshop on Variability Modelling of Software-Intensive Systems, VAMOS '19, New York, NY, USA, 2019. Association for Computing Machinery. 13

[14] S. She, R. Lotufo, T. Berger, A. Wąsowski, and K. Czarnecki. The variability model of the linux kernel. In D. Benavides, D. S. Batory, and P. Grünbacher, editors, *Fourth International Workshop on Variability Modelling of Software-Intensive Systems, Linz, Austria, January 27-29, 2010. Proceedings*, volume 37 of *ICB-Research Report*, pages 45–51. Universität Duisburg-Essen, 2010. 13, 15, 16

[15] V. Rothberg, N. Dintzner, A. Ziegler, and D. Lohmann. Feature models in linux: From symbols to semantics. In *Proceedings of the Tenth International Workshop on Variability Modelling of Software-intensive Systems*, VaMoS '16, pages 65–72, New York, NY, USA, 2016. ACM. 13

[16] J. Galindo, D. Benavides, and S. Segura. Debian packages repositories as software product line models. towards automated analysis. pages 29–34, 2010. 13, 14, 15, 16, 30, 31, 51

[17] J. A. Galindo, H. Turner, D. Benavides, and J. White. Testing variability-intensive systems using automated analysis: An application to android. *Software Quality Journal*, 24(2):365–405, June 2016. 13, 14, 15, 16

[18] A. B. Sánchez, S. Segura, J.A. Parejo, and A. Ruiz-Cortés. Variability testing in the wild: the Drupal case study. *Software & Systems Modeling*, pages 1–22, 2015. 13

[19] M. Lienhardt, F. Damiani, E. B. Johnsen, and J. Mauro. Lazy product discovery in huge configuration spaces. In *Proceedings of the ACM/IEEE 42nd International*

Conference on Software Engineering, ICSE '20, page 1509–1521, New York, NY, USA, 2020. Association for Computing Machinery. 14, 79

[20] U. Junker. QuickXPlain: Preferred Explanations and Relaxations for Over-constrained Problems. In *19th national conference on Artifical intelligence*, pages 167–172, San Jose, CA, 2004. AAAI Press. 14, 16, 35, 61

[21] A. Felfernig, D. Benavides, J. Galindo, and F. Reinfrank. Towards anomaly explanation in feature models. In *Proceedings of the 15th International Configuration Workshop*, August 2013. 14, 15, 17, 50, 54, 61, 79

[22] A. Felfernig, M. Schubert, and C. Zehentner. An efficient diagnosis algorithm for inconsistent constraint sets. *Artif. Intell. Eng. Des. Anal. Manuf.*, 26(1):53–62, February 2012. 17, 38, 50, 54, 60, 64, 70

[23] E. Bodden, É. Tanter, and M. Inostroza. Join point interfaces for safe and flexible decoupling of aspects. *ACM Trans. Softw. Eng. Methodol.*, 23(1), February 2014. 18

[24] J. A. Galindo, M. Acher, J. M. Tirado, C. Vidal, B. Baudry, and D. Benavides. Exploiting the enumeration of all feature model configurations: a new perspective with distributed computing. In Hong Mei, editor, *Proceedings of the 20th International Systems and Software Product Line Conference, SPLC 2016, Beijing, China, September 16-23, 2016*, pages 74–78. ACM, 2016. 21, 79, 81

[25] C. Vidal-Silva, A. Felfernig, J. A. Galindo, M. Atas, and D. Benavides. A parallelized variant of junker's quickxplain algorithm. In Denis Helic, Gerhard Leitner, Martin Stettinger, Alexander Felfernig, and Zbigniew W. Raś, editors, *Foundations of Intelligent Systems*, pages 457–468,

Cham, 2020. Springer International Publishing. 21, 58, 74, 78

[26] C. Vidal-Silva, D. Benavides, J. A. Galindo, P. Leger, R. Villarroel, and S. Valenzuela. JPI feature models - Exploring a JPI and FOP symbiosis for software modeling. pages 1–6, 2015. 21

[27] C. Vidal, D. Benavides, P. Leger, J. A. Galindo, and H. Fukuda. Mixing of join point interfaces and feature-oriented programming for modular software product line. In Junichi Suzuki, Tadashi Nakano, and Henry Hess, editors, *BICT 2015, Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS), New York City, United States, December 3-5, 2015*, pages 433–437. ICST/ACM, 2015. 21

[28] C. Vidal, D. Benavides, J. A. Galindo, and P. Leger. Exploring the Synergies between Joing Point Interfaces and Feature-Oriented Programming. In *JISBD 2015, XX JORNADAS DE INGENIERÍA DEL SOFTWARE Y BASES DE DATOS, Santander, España, September, 2015*, 2015. 21

[29] C. Vidal, D. Benavides, P. Leger, J. A. Galindo, and H. Fukuda. Mixing of join point interfaces and feature-oriented programming for modular software product line. *EAI Endorsed Trans. Scalable Information Systems*, 3(10):e2, 2016. 21

[30] C. Vidal Silva. Exploring efficient analysis alternatives on feature models. In Maurice H. ter Beek, Walter Cazzola, Oscar Díaz, Marcello La Rosa, Roberto E. Lopez-Herrejon, Thomas Thüm, Javier Troya, Antonio Ruiz Cortés, and David Benavides, editors, *Proceedings of the 21st International Systems and Software Product*

Line Conference, SPLC 2017, Volume B, Sevilla, Spain, September 25-29, 2017, pages 150–155. ACM, 2017. 21

[31] C. Vidal, J. A. Galindo, J. Giráldez, and D. Benavides. Automated completion of partial configurations as a diagnosis task. Using FastDiag to improve performance. In *ISMIS 2020, Industry Session, Graz University of Technology, Graz, Austria, September, 2020*, 2020. 21, 78

[32] C. Vidal, J. A. Galindo, and D. Benavides. Functional Testing of Conflict Detection and Diagnosis Tools in Feature Model Configuration: A Test Suite Design. In *ConfWS 2020, 22th International Workshop on Configuration, online event, September, 2020*, 2020. 21, 78, 82

[33] K. Czarnecki, S. Helsen, and U. Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 10:7–29, 01 2005. 24

[34] C. Thörn and K. Sandkuhl. *Feature Modeling: Managing Variability in Complex Systems*, pages 129–162. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. 25

[35] R. Mazo, R. Lopez-Herrejon, C. Salinesi, D. Diaz, and A. Egyed. Conformance checking with constraint logic programming: The case of feature models. pages 456 – 465, 08 2011. 25, 51

[36] F. Zhou, J. Roger Jiao, X. J. Yang, and B. Lei. Augmenting feature model through customer preference mining by hybrid sentiment analysis. *Expert Systems with Applications*, 89:306 – 317, 2017. 25

[37] M. Weckesser, M. Lochau, T. Schnabel, B. Richerzhagen, and A. Schürr. Mind the gap! automated anomaly detection for potentially unbounded cardinality-based feature

models. In *Proceedings of the 19th International Conference on Fundamental Approaches to Software Engineering - Volume 9633*, pages 158–175, New York, NY, USA, 2016. Springer-Verlag New York, Inc. 25, 52

[38] A. R. Santos and E. Santana de Almeida. Do #ifdef-based variation points realize feature model constraints? *SIGSOFT Softw. Eng. Notes*, 40(6):1–5, November 2015. 25

[39] S. Segura, A. B. Sánchez, and A. Ruiz-Cortés. Automated variability analysis and testing of an e-commerce site: An experience report. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ASE '14, pages 139–150, New York, NY, USA, 09/2014 2014. ACM, ACM. 25

[40] M. Nieke, J. Mauro, C. Seidl, T. Thüm, I. C. Yu, and F. Franzke. Anomaly analyses for feature-model evolution. In *Proceedings of the 17th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences*, GPCE 2018, pages 188–201, New York, NY, USA, 2018. ACM. 25, 45, 52

[41] A. Khtira, A. Benlarabi, and B. Asri. Duplication detection when evolving feature models of software product lines. *Information (Switzerland)*, 6:592–612, 10 2015. 26, 51

[42] D. M. Le, H. Lee, K. C. Kang, and L. Keun. Validating consistency between a feature model and its implementation. In John Favaro and Maurizio Morisio, editors, *Safe and Secure Software Reuse*, pages 1–16, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. 26

[43] A. Gómez and I. Ramos. Automatic tool support for cardinality-based feature modeling with model constraints

for information systems development. In *Information Systems Development, Business Systems and Services: Modeling and Development [Proceedings of ISD 2010, Charles University in Prague, Czech Republic, August 25-27, 2010]*, pages 271–284, 2010. 26, 28

[44] C. Quinton, A. Pleuss, D. L. Berre, L. Duchien, and G. Botterweck. Consistency checking for the evolution of cardinality-based feature models. In *Proceedings of the 18th International Software Product Line Conference - Volume 1*, SPLC '14, pages 122–131, New York, NY, USA, 2014. ACM. 26, 52

[45] A. S. Karataş and H. Oğuztüzün. Attribute-based variability in feature models. *Requir. Eng.*, 21(2):185–208, June 2016. 26, 29

[46] F. Roos-Frantz, D. Benavides, A. Ruiz-Cortés, A. Heuer, and K. Lauenroth. Quality-aware analysis in product line engineering with the orthogonal variability model. *Software Quality Journal*, 20(3):519–565, Sep 2012. 26, 29

[47] C. Hwan, P. Kim, and K. Czarnecki. Synchronizing cardinality-based feature models and their specializations. In *Proceedings of the First European Conference on Model Driven Architecture: Foundations and Applications*, ECMDA-FA'05, pages 331–348, Berlin, Heidelberg, 2005. Springer-Verlag. 26

[48] C. Quinton, D. Romero, and L. Duchien. Cardinality-based feature models with constraints: A pragmatic approach. In *Proceedings of the 17th International Software Product Line Conference*, SPLC '13, pages 162–166, New York, NY, USA, 2013. ACM. 28

[49] T. Schnabel, M. Weckesser, R. Kluge, M. Lochau, and A. Schürr. Cardygan: Tool support for cardinality-based

feature models. In *Proceedings of the Tenth International Workshop on Variability Modelling of Software-intensive Systems*, VaMoS '16, pages 33–40, New York, NY, USA, 2016. ACM. 28, 52

[50] D. Benavides, P. Trinidad, and A. Ruiz-Cortés. Automated reasoning on feature models. In Oscar Pastor and João Falcão e Cunha, editors, *Advanced Information Systems Engineering*, pages 491–503, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. 29

[51] J. A. Pereira, S. Krieter, J. Meinicke, R. Schröter, G. Saake, and T. Leich. Featureide: Scalable product configuration of variable systems. In *Proceedings of the 15th International Conference on Software Reuse: Bridging with Social-Awareness - Volume 9679*, ICSR 2016, pages 397–401, New York, NY, USA, 2016. Springer-Verlag New York, Inc. 30

[52] pure-systems - the leading provider of software for product line and variant management tools | pure::variants. `https://www.pure-systems.com/products/pure-variants-9.html`. Accessed: 2020-07-02. 30

[53] K. Pohl, G. Böckle, and F. J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag, Berlin, Heidelberg, 2005. 30

[54] F. Roos-Frantz, D. Benavides, and A. Ruiz-Cortés. Feature model to orthogonal variability model transformations. a first step. In *Actas del VI Taller sobre Desarrollo de Software Dirigido por Modelos. Actas de los talleres de las JISBD09*, volume 3, pages 81–90, San Sebastián, España, 09/2009 2009. 30

[55] J. Galindo, F. Roos-Frantz, D. Benavides, A. Ruiz-Cortés, and J. García-Galán. Automated analysis of diverse variability models with tool support. In *Proceedings of In Jornadas de Ciencia e Ingeniería de Servicios (JCIS) Sistedes*, 01 2014. 30

[56] A. Svendsen, X. Zhang, R. Lind-Tviberg, F. Fleurey, Ø. Haugen, B. Møller-Pedersen, and G. K. Olsen. Developing a software product line for train control: A case study of cvl. In *Proceedings of the 14th International Conference on Software Product Lines: Going Beyond*, SPLC'10, page 106–120, Berlin, Heidelberg, 2010. Springer-Verlag. 31

[57] M. Antkiewicz, K. Bąk, A. Murashkin, R. Olaechea, J. H. (Jimmy) Liang, and K. Czarnecki. Clafer tools for product line engineering. In *Proceedings of the 17th International Software Product Line Conference Co-Located Workshops*, SPLC '13 Workshops, page 130–135, New York, NY, USA, 2013. Association for Computing Machinery. 31

[58] S. Ibraheem and S. Ghoul. Software evolution: A features variability modeling approach. *Journal of Software Engineering*, 11:12–21, 2017. 41

[59] Z. Yin, X. Ma, J. Zheng, Y. Zhou, L. N. Bairavasundaram, and S. Pasupathy. An empirical study on configuration errors in commercial and open source systems. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 159–172, 2011. 41

[60] Facebook. More details on today's outage. `https://m.facebook.com/nt/screen/?params=%7B%22note_id%22%3A10158791436142200%`

`7D&path=%2Fnotes%2F%7Bnote_id%7D&_rdr`. Accessed: 2021-26-01. 41

[61] L. A. Barroso and U. Hoelzle. *The Datacenter As a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 1st edition, 2009. 41

[62] J. Zhanwen-Li, S. He, L. Zhu, X. Xu, M. Fu, L. Bass, A. Liu, and A. B. Tran. Challenges to error diagnosis in hadoop ecosystems. In *Proceedings of the 27th Large Installation System Administration Conference (LISA)*, pages 145–154, 2013. 41

[63] H. Riener and G. Fey. Exact diagnosis using boolean satisfiability. In *Proceedings of the 35th International Conference on Computer-Aided Design*, ICCAD '16, New York, NY, USA, 2016. Association for Computing Machinery. 41

[64] A. Elfaki, L. Sim, P. Vijayaprasad, M. G. Md-Johar, and M. Fadhil. Using a rule-based method for detecting anomalies in software product line. *Research Journal of Applied Sciences, Engineering and Technology*, 7:275–281, 01 2014. 44, 50

[65] A. Durán, D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés. Flame: A formal framework for the automated analysis of software product lines validated by automated specification testing. *Softw. Syst. Model.*, 16(4):1049–1082, October 2017. 44

[66] I. Mistrík, M. Galster, and B. R. Maxim, editors. *Software Engineering for Variability Intensive Systems - Foundations and Applications*. Auerbach Publications / Taylor & Francis, 2019. 44

[67] M. Kowal, S. Ananieva, and T. Thüm. Explaining anomalies in feature models. *SIGPLAN Not.*, 52(3):132–143, October 2016. 44, 51

[68] F. Roos-Frantz, D. Benavides, and A. Ruiz-Cortés. Automated analysis of orthogonal variability models using constraint programming. pages 269–280, 01 2010. 44

[69] R. E. Lopez-Herrejon and A. Egyed. Towards fixing inconsistencies in models with variability. In *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '12, pages 93–100, New York, NY, USA, 2012. ACM. 44, 45, 51

[70] M. Pol'la, A. Buccella, and A. Cechich. Automated analysis of variability models: The sevatax process. In O. Gervasi, B. Murgante, S. Misra, E. Stankova, C. M. Torre, A. M. Rocha, D. Taniar, B. O. Apduhan, E. Tarantino, and Y. Ryu, editors, *Computational Science and Its Applications – ICCSA 2018*, pages 365–381, Cham, 2018. Springer International Publishing. 45

[71] M. Pol'la, A. Buccella, and A. Cechich. Using scope scenarios to verify multiple variability models. In S. Misra, O. Gervasi, B. Murgante, E. Stankova, V. Korkhov, C. Torre, A. M. Rocha, D. Taniar, B. O. Apduhan, and E. Tarantino, editors, *Computational Science and Its Applications – ICCSA 2019*, pages 383–399, Cham, 2019. Springer International Publishing. 45

[72] D. Benavides, A. Felfernig, J. A. Galindo, and F. Reinfrank. Automated analysis in feature modelling and product configuration. In *ICSR*, pages 160–175, Pisa, Italy, 06/2013 2013. 45

[73] A. Nöhrer, A. Biere, and A. Egyed. Managing sat inconsistencies with humus. In *Proceedings of the Sixth Inter-*

national Workshop on Variability Modeling of Software-Intensive Systems, VaMoS '12, pages 83–91, New York, NY, USA, 2012. ACM. 45

[74] A. Nöhrer, A. Biere, and A. Egyed. A comparison of strategies for tolerating inconsistencies during decision-making. In *Proceedings of the 16th International Software Product Line Conference - Volume 1*, SPLC '12, pages 11–20, New York, NY, USA, 2012. ACM. 45

[75] J. Guo, Y. Wang, P. Trinidad, and D. Benavides. Consistency maintenance for evolving feature models. *Expert Syst. Appl.*, 39(5):4987–4998, 2012. 45, 51

[76] D. Hinterreiter, K. Feichtinger, L. Linsbauer, H. Prähofer, and P. Grünbacher. Supporting feature model evolution by lifting code-level dependencies: A research preview. In *Requirements Engineering: Foundation for Software Quality - 25th International Working Conference, REFSQ 2019, Essen, Germany, March 18-21, 2019, Proceedings*, pages 169–175, 2019. 45, 51

[77] S. H. Choi. Verification tool for feature models and configurations using semantic web technologies. *Journal of the Korea society of IT services*, 10:189–201, 09 2011. 45, 50

[78] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. Le-Traon. Towards automated testing and fixing of re-engineered feature models. pages 1245–1248, 05 2013. 45, 51

[79] B. Kitchenham, O. Pearl-Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman. Systematic literature reviews in software engineering - a systematic literature review. *Inf. Softw. Technol.*, 51(1):7–15, January 2009. 45

[80] B. Napoleão, K. Romero-Felizardo, É. Ferreira de Souza, and N. L. Vijaykumar. Practical similarities and differences between systematic literature reviews and systematic mappings: a tertiary study. pages 85–90, 2017. 45

[81] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson. Systematic mapping studies in software engineering. pages 68–77, 2008. 45, 46, 47

[82] A. Halim, R. Ahmad, Z. A. Muhamad-Noh, F. Hazwani, and N. Mohd-Alwi. Systematic review for network survivability analysis in manets. *Procedia - Social and Behavioral Sciences*, 195:1872–1881, 07 2015. 45

[83] C. Aboud, C. Andrucioli de Mattos Pimenta, and M. Nobre. The pico strategy for the research question construction and evidence search. *Revista latino-americana de enfermagem*, 15:508–11, 05 2007. 46

[84] Google. Google Scholar. `https://scholar.google.cl/`, 2020. Accessed: 2020-02-02. 47

[85] Elsevier. Scopus. `https://www.scopus.com/search/form.uri?display=basic`, 2020. Accessed: 2020-01-02. 47

[86] Clarivate Analytics. Web of Science. `https://apps.webofknowledge.com/`, 2020. Accessed: 2020-01-03. 47

[87] I. Achour, L. Jilani, and H. Ben-Ghezala. Towards an extended tool for analysis of extended feature models. pages 1–5, 06 2014. 50

[88] U. Afzal, T. Mahmood, I. Rauf, and Z. A. Shaikh. Minimizing feature model inconsistencies in software product lines. In *17th IEEE International Multi Topic Conference 2014*, pages 137–142, Dec 2014. 50

[89] S. Ananieva, M. Kowal, T. Thüm, and I. Schaefer. Implicit constraints in partial feature models. In *Proceedings of the 7th International Workshop on Feature-Oriented Software Development*, FOSD 2016, pages 18–27, New York, NY, USA, 2016. ACM. 50, 56

[90] P. Arcaini, A. Gargantini, E. Riccobene, and P. Vavassori. A novel use of equivalent mutants for static anomaly detection in software artifacts. *Information and Software Technology*, 81:52 – 64, 2017. 50

[91] P. Arcaini, A. Gargantini, and P. Vavassori. Generating tests for detecting faults in feature models. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, pages 1–10, April 2015. 50

[92] P. Arcaini, A. Gargantini, and P. Vavassori. Automatic detection and removal of conformance faults in feature models. In *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 102–112, April 2016. 50

[93] M. Asadi, G. Gröner, B. Mohabbati, and D. Gašević. Goal-oriented modeling and verification of feature-oriented product lines. *Softw. Syst. Model.*, 15(1):257–279, February 2016. 50

[94] M. Asadi, B. Mohabbati, G. Gröner, and D. Gasevic. Development and validation of customized process models. *Journal of Systems and Software*, 96, 10 2014. 50

[95] J. Barreiros and A. Moreira. Flexible modeling and product derivation in software product lines. *Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE*, 2014:67–70, 01 2014. 50

[96] M. Bhushan, S. Goel, and K. Kaur. Analyzing inconsistencies in software product lines using an ontological rule-based approach. *Journal of Systems and Software*, 137:605 – 617, 2018. 50

[97] M. Bhushan, S. Goel, and A. Loura. Improving quality of software product line by analyzing inconsistencies in feature models using an ontological rule-based approach. *Expert Systems*, 35, 11 2017. 50

[98] S. bin Abid. Resolving feature dependency implementations inconsistencies during product derivation. In *Proceedings of the 6th ECMFA Traceability Workshop*, ECMFA-TW '10, pages 31–38, New York, NY, USA, 2010. ACM. 50

[99] P. Costa, F. Marinho, R. Andrade, and T. Oliveira. Fixture - a tool for automatic inconsistencies detection in context-aware spl. *ICEIS 2015 - 17th International Conference on Enterprise Information Systems, Proceedings*, 2:114–125, 01 2015. 50

[100] A. Felfernig, R. Walter, and S. Reiterer. Flexdiag: Anytime diagnosis for reconfiguration. September 2015. 50, 54, 59, 79

[101] A. Felfernig, R. Walter, J. A. Galindo, D. Benavides, S. Polat Erdeniz, M. Atas, and S. Reiterer. Anytime diagnosis for reconfiguration. *J. Intell. Inf. Syst.*, 51(1):161–182, 2018. 50, 54, 59

[102] R. Gheyi, T. Massoni, and P. Borba. Automatically checking feature model refactorings. *J. UCS*, 17(5):684–711, 2011. 51

[103] M. Javed and M. Naeem. Automated inconsistency detection in feature models: A generative programming based approach. *Selforganizology*, 3:59–74, 06 2016. 51

[104] A. Khtira, A. Benlarabi, and Bo. Asri. A tool support for automatic detection of duplicate features during software product lines evolution. *IJCSI International Journal of Computer Science Issues*, 12:1–10, 07 2015. 51

[105] U. Lesta, I. Schaefer, and Winkelmann T. Detecting and explaining conflicts in attributed feature models. In *Proceedings 6th Workshop on Formal Methods and Analysis in SPL Engineering, FMSPLE@ETAPS 2015, London, UK, 11 April 2015.*, pages 31–43, 2015. 51

[106] L. Barachisio-Lisboa, V. Cardoso-Garcia, S. Romero de Lemos Meira, and E. Santana de Almeida. A support tool for domain analysis. In *Fourth International Workshop on Variability Modelling of Software-Intensive Systems, Linz, Austria, January 27-29, 2010. Proceedings*, pages 175–178, 2010. 51

[107] L. Barachisio-Lisboa, V. Cardoso-Garcia, E. Santana-de Almeida, and Silvio Romero-de Lemos Meira. Toolday: A tool for domain analysis. *Int. J. Softw. Tools Technol. Transf.*, 13(4):337–353, August 2011. 51

[108] R. E. Lopez-Herrejon and A. Egyed. Searching the variability space to fix model inconsistencies: A preliminary assessment. In *Third International Symposium Search Based Software Engineering SSBSE 2011 (2011). Proceedings*, 2011. 51, 55

[109] F. G. Marinho, P. H. M. Maia, R. M. C. Andrade, V. M. P. Vidal, P. A. S. Costa, and C. Werner. Safe adaptation in context-aware feature models. In *Proceedings of the*

4th International Workshop on Feature-Oriented Software Development, FOSD '12, pages 54–61, New York, NY, USA, 2012. ACM. 51

[110] J. Mauro, M. Nieke, C. Seidl, and I. C. Yu. Anomaly detection and explanation in context-aware software product lines. In *Proceedings of the 21st International Systems and Software Product Line Conference - Volume B*, SPLC '17, pages 18–21, New York, NY, USA, 2017. ACM. 51

[111] S. Nakajima. Semi-automated diagnosis of foda feature diagram. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 2191–2197, New York, NY, USA, 2010. ACM. 52

[112] S. Nakajima. Non-clausal encoding of feature diagram for automated diagnosis. In *Proceedings of the 14th International Conference on Software Product Lines: Going Beyond*, SPLC'10, pages 420–424, Berlin, Heidelberg, 2010. Springer-Verlag. 52

[113] M. Noorian, A. Ensan, E. Bagheri, H. Boley, and Y. Biletskiy. Feature model debugging based on description logic reasoning. *Proceedings: DMS 2011 - 17th International Conference on Distributed Multimedia Systems*, pages 158–164, 01 2011. 52

[114] S. Ripon, K. Azad, S. J. Hossain, and M. Hassan. Modeling and analysis of product-line variants. In *Proceedings of the 16th International Software Product Line Conference - Volume 2*, SPLC '12, pages 26–31, New York, NY, USA, 2012. ACM. 52

[115] S. Ripon, M. Piash, A. Hossain, and M. S. Uddin. Semantic webbased analysis of product line variant model. *International Journal of Computer and Electrical Engineering*, pages 1–6, 01 2014. 52

[116] C. Vidal-Silva. Reviewing diagnosis solutions for valid product configurations in the automated analysis of feature models. *International Journal of Advanced Computer Science and Applications*, 10(1), 2019. 52, 54, 78

[117] B. Wang, Z. Hu, Y. Xiong, H. Zhao, W. Zhang, and H. Mei. Tolerating inconsistency in feature models. *CEUR Workshop Proceedings*, 661, 01 2010. 52

[118] B. Wang, Y.-F. Xiong, Z.-J. Hu, H.-Y. Zhao, W. Zhang, and H. Mei. Interactive inconsistency fixing in feature modeling. *Journal of Computer Science and Technology*, 29(4):724–736, Jul 2014. 52

[119] J. White, D. Benavides, D. C. Schmidt, P. Trinidad, B. Dougherty, and A. Ruiz-Cortes. Automated diagnosis of feature model configurations. *J. Syst. Softw.*, 83(7):1094–1107, July 2010. 52, 62

[120] G. Zhang, H. Ye, and Y. Lin. Modelling quality attributes in feature models in software product line engineering. In *Proceedings of the 6th International Conference on Software and Database Technologies - Volume 2: ICSOFT,*, pages 249–254. INSTICC, SciTePress, 2011. 52

[121] D. Benavides, A. Ruiz-Cortés, and P. Trinidad. Automated reasoning on feature models. *LNCS, Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005*, 3520:491–503, 2005. 59, 62

[122] M. Stumptner. An Overview of Knowledge-based Configuration. *Ai Communications*, 10(2):111–125, 1997. 59

[123] A. Felfernig and R. Burke. Constraint-based Recommender Systems: Technologies and Research Issues. In *ACM International Conference on Electronic Commerce (ICEC'08)*, pages 17–26, Innsbruck, Austria, 2008. 59

[124] F. Ricci, L Rokach, B. Shapira, and P. Kantor. *Recommender Systems Handbook.* Springer, 2011. 59

[125] G. Friedrich, M. Stumptner, and F. Wotawa. Model-based Diagnosis of Hardware Designs. *Artificial Intelligence*, 111(1–2):3–39, 1999. 59

[126] T. Schmitz and D. Jannach. An AI-based Interactive Tool for Spreadsheet Debugging. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'17)*, pages 333–334, Raleigh, NC, USA, 2017. IEEE. 59

[127] J. M. Horcas, M. Pinto, and L. Fuentes. Variability models for generating efficient configurations of functional quality attributes. *Information and Software Technology*, 95:147–164, 2018. 59

[128] R. Reiter. A theory of diagnosis from first principles. *AI Journal*, 23(1):57—95, 1987. 60, 61

[129] L. Bordeaux, Y. Hamadi, and H. Samulowitz. Experiments with Massively Parallel Constraint Solving. In *21st International Joint Conference on Artifical Intelligence*, pages 443–448, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers. 60, 61

[130] J. Díaz, J. Pérez, and J. Garbajosa. Agile product-line architecting in practice: A case study in smart grids. *Information and Software Technology*, 56(7):727–748, 2014. 60

[131] I. Gent, I. Miguel, P. Nightingale, C. McCreesh, P. Prosser, N. Nooore, and C. Unsworth. A Review of Literature on Parallel Constraint Solving. *Theory and Practice of Logic Programming*, 18(5–6):725–758, 2018. 60, 61

[132] Y. Hamadi and L. Sais. *Handbook of Parallel Constraint Reasoning*. Springer, 2018. 60, 61

[133] J. Marques-Silva, F. Heras, M. Janota, A. Previti, and A. Belov. On Computing Minimal Correction Subsets. In *23rd international Joint Conference on Artificial Intelligence*, pages 615–622, Beijing, China, 2013. 60, 61

[134] Á. J. Varela-Vaca, J. A. Galindo, B. Ramos-Gutiérrez, M. T. Gómez-López, and D. Benavides. Process mining to unleash variability management: discovering configuration workflows using logs. In *Proceedings of the 23rd International Systems and Software Product Line Conference-Volume A*, pages 265–276, 2019. 60

[135] F. W. Burton. Speculative computation, parallelism, and functional programming. *IEEE Transactions on Computers*, C-34(12):1190–1193, Dec 1985. 60, 61, 64

[136] J. Galindo and D. Benavides. Towards a new repository for feature model exchange. In C. Cetina, O. Díaz, L. Duchien, M. Huchard, R. Rabiser, C. Salinesi, C. Seidl, X. Tërnava, L. Teixeira, T. Thüm, and T. Zadi, editors, *Proceedings of the 23rd International Systems and Software Product Line Conference, SPLC 2019, Volume B, Paris, France, September 9-13, 2019*, pages 85:1–85:4. ACM, 2019. 60, 71

[137] D. Jannach, T. Schmitz, and K. Shchekotykhin. Parallelized Hitting Set Computation for Model-Based Diagnosis. In *29th AAAI Conference on Artificial Intelligence*, pages 1503–1510, Austin, Texas, 2015. AAAI Press. 61

[138] R. Bakker and F. Dikker. Diagnosing and Solving Overdetermined Constraint Satisfaction Problems. In *13th International Joint Conference on Artificial Intelligence (IJCAI'93)*, pages 276–281, Chambéry, France, 1993. 61

[139] J. de Kleer and B. Williams. Diagnosing Multiple Faults. *Artificial Intelligence*, 32(1):97–130, 1987. 61

[140] F. Wotawa. A Variant of Reiter's Hitting-Set Algorithm. *Information Processing Letters*, 79(1):45–51, 2001. 61

[141] D. Jannach, T. Schmitz, and K. Shchekotykhin. Parallel Model-Based Diagnosis on Multi-Core Computers. *Journal of Artificial Intelligence Research*, 55:835–887, 2016. 61

[142] J. Galindo, D. Benavides, P. Trinidad, A. Gutiérrez-Fernández, and A. Ruiz-Cortés. Automated analysis of feature models: Quo vadis? *Computing*, 101(5):387–433, 2019. 61

[143] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993. 62

[144] D. Le Berre and A. Parrain. The sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(2-3):59–64, 2010. 71

[145] M. Alférez, M. Acher, J. A. Galindo, B. Baudry, and D. Benavides. Modeling variability in the video domain: Language and experience report. *Software Quality Journal*, 27(1):307–347, 2019. 71

[146] G. Doux, P. Albert, G. Barbier, J. Cabot, M. D. Del Fabro, and S. U.-J. Lee. An mde-based approach for solving configuration problems: An application to the eclipse platform. In *European Conference on Modelling Foundations and Applications*, pages 160–171. Springer, 2011. 71

[147] C. Vidal-Silva, A. Felfernig, J. A. Galindo, M. Atas, and D. Benavides. Explanations for over-constrained problems with parallelized QUICKXPLAIN. In Denis Helic,

Gerhard Leitner, Martin Stettinger, Alexander Felfernig, and Zbigniew W. Raś, editors, *Journal of Intelligent Information Systems - Integrating Artificial Intelligence and Database Technologies*, pages –, Cham, 2021. Springer International Publishing. 78

[148] K. Shchekotykhin, D. Jannach, and T. Schmitz. MERGEXPLAIN: Fast Computation of Multiple Conflicts for Diagnosis. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 3221–3228. AAAI Press, 2015. 79

[149] K. Shchekotykhin, D. Jannach, and T. Schmitz. *Parallel Model-Based Diagnosis*, pages 547–580. Springer International Publishing, Cham, 2018. 79