A Generic Framework for Improving Software Product Line using an Ontological Rule-Based Approach

A Thesis

Submitted in fulfillment of the requirement

for the award of the degree of

Doctor of Philosophy
in
Computer Science and Engineering

Submitted By

Megha
(Registration No. 901303001)

Under the guidance of

Dr. Ajay Kumar
Assistant Professor
Department of Computer Science & Engineering
Thapar Institute of Engineering & Technology
Patiala-147004, India

Dr. Shivani Goel
Professor
Department of Computer Science Engineering
Bennett University
Greater Noida 201310, U.P., India

Department of Computer Science & Engineering
Thapar Institute of Engineering & Technology
Patiala – 147004

February - 2018

DEDICATED WITH EXTREME AFFECTION AND GRATITUDE

To the memory of my grandparents,

My parents Dr. Kartar Chand and Mrs. Sharda Kumari,

and My loving nephew Kartavya

CERTIFICATE

I hereby certify that work which is being presented in the thesis entitled *"A Generic Framework for Improving Software Product Line using an Ontological Rule-Based Approach"* in fulfillment of the requirements of **DOCTOR OF PHILOSOPHY** submitted in the **DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**, THAPAR INSTITUTE OF ENGINEERING & TECHNOLOGY, PATIALA is an authentic record of my own work carried out under the supervision of Dr. Ajay Kumar and Dr. Shivani Goel.

The matter embodied in this thesis has not been submitted by me for the award of any other degree of this or any other University.

Megha

Megha

Registration No. 901303001

This to certify that the above made statement by the candidate is correct to the best of my knowledge and belief.

A Kumar

Dr. Ajay Kumar
Assistant Professor
Department of Computer Science & Engineering
Thapar Institute of Engineering & Technology
Patiala-147004, India

Shivani Goel.
07/02/18

Dr. Shivani Goel
Professor
Department of Computer Science Engineering
Bennett University
Greater Noida 201310, U.P., India

ACKNOWLEDGEMENT

First and foremost, I would like to thank Almighty God, the most gracious and merciful, for providing me this opportunity and granting me the capability to proceed successfully.

I take this opportunity to express my profound gratitude and deep regards to my respected Supervisors, *Dr. Ajay Kumar, Assistant Professor, Department of Computer Science & Engineering, Thapar Institute of Engineering & Technology, Patiala, India* and *Dr. Shivani Goel, Professor, Department of Computer Science Engineering, Bennett University, Greater Noida, U.P., India* for their invaluable guidance, support, and encouragement throughout the course of this thesis. The blessing, help, and guidance given by them shall carry me a long way in the journey of life on which I am about to embark. I feel very lucky to get a lifetime opportunity to work under their knowledgeable supervision. It has been my great pleasure and honor to have worked with them.

I also express my gratitude to the Doctoral committee comprising of Dr. Maninder Singh (Head, CSED), Dr. Parteek Kumar (Associate Professor, CSED), Dr. Ashutosh Mishra (Assistant Professor, CSED), and Dr. Mahesh Sharma (Associate Professor, SoM) for monitoring the progress and providing valuable suggestions for the improvement of this Ph.D. work. I am very thankful to Dean of Research and Sponsored Projects Dr. O.P. Pandey for his valuable suggestions. I express my gratitude to Dr. Seema Bawa (Professor, CSED) and Dr. Karamjit Kaur (Assistant Professor, CSED) for their continuous motivation for my work.

I would like to acknowledge the most important people in my life – my parents. I owe my personal regards to them for their love, blessings, care, inspiration, encouragement and motivation which have enabled me to complete my task. My sister Mrs. Priyanka, brother-in-law Mr. Naresh Kumar, and brother Mr. Chaksham Bhushan have also supported me and encouraged me to carry out this work. In spite of all the odds, they always stood beside me at every point of my life. I am also thankful to my nephews Abeer Koul, Kartavya and niece Rusham Koul as their smiles have always encouraged me to work. I especially thank to Major Arun Negi, Indian Army, for his faith, support and constant encouragement throughout the entire process. I would honestly say my respected parents and respected supervisors continuous support and motivation that

ultimately made it possible for me. This task would not have been possible, if they were not there with me.

I am also indebted to Mrs. Alka Pandita, Mr. Deepak Koul, Dr. Balveer Painam (Associate professor, K L University, Andhra Pradesh), Ms. Anju Bala (Coach, TIET, Patiala, Punjab), Mr. Sumit Gupta (Lecturer, GCET, Jammu), Ms. Sujata Rani and Mr. Vikram Jeet Singh (Tehsildar, Moorang, Distt. Kinnaur, HP) not only for all their useful suggestions but also for being there to listen when I needed an ear.

Sincere, thanks to all my friends especially, Vaibhav Aggarwal, Dr. Divya Pandove, Ayush Sahu, Cherish Sham, Shallini Rajpoot, Sharad Tiwari, Abdullah AI-Qudaimi, Sachendra Singh Chauhan, Mohammad Abuzar Sayeed, Shubham Goel, Mukti Yadav, Madhvi Verma, Dr. Jainy Sachdeva (Assistant Professor, EIED), Dr. Avleen Malhi (Assistant Professor, CSED), and Dr. Niyati Baliyan (Assistant Professor, CSE, JIIT, Noida, U.P.) for their support and cooperation throughout the entire process.

I would like to thank the research scholars of CSE Ph.D. Lab, faculty members, staff members of department of CSE and my seniors who were always there at need of the hour and provided with the help and facilities, which I required for the completion of my thesis.

I gratefully acknowledge the funding received towards my Ph.D. from the RGNF by University Grants Commission (UGC), New Delhi, Government of India.

Finally, my thanks go to all the people who have supported me to complete the research work directly or indirectly.

(Megha)

ABSTRACT

Software product line engineering (SPLE) is an emergent technical paradigm for generating software products. A software product line (SPL) is a family of related software intensive systems, sharing a common and managed set of features that fulfill the specific requirements of a certain domain. The main focus of SPL is software reuse in an attempt to improve the quality and productivity while reducing cost as well as time to market. Feature model (FM) is a well-known notation that represents commonality and variability of SPL. The accurate combination of features in FM allows deriving a valid product from SPL. The development and growing size of FMs may inevitably introduce inaccurate feature relationships. These relationships may cause defects in models such as defects due to redundancy, anomaly, inconsistency and wrong cardinality. These defects can be inherited in the software products built from a defective product line model. These defects diminish the quality of FMs and benefits of SPL.

A defect in FM is a critical issue in the SPL community as software products are derived by reusing models. One of the major factors behind the successful derivation of defect free valid software products from SPL is the quality of FM, which in turn depends on how a defect in FM is resolved. Moreover, manually inspecting defects in large-sized FMs is a laborious task. Therefore, it is of utmost importance to resolve defects to support reusability in the industrial domain. As, it further leads to mass production of software products and provides the benefits of SPL, i.e., reduced development time, cost and improves quality of software products. The identification of defects in FM is well researched, however, providing cause of defects with their correction in a language which is easily understandable by product line (PL) developers is still a challenge. The lack of methods to explain the causes and corrections of defects in a user-friendly language for resolving defects motivated us to propose an effective approach to assure the quality of FMs in SPL, which in turn ensures the benefits of reusability in industry.

In this thesis, a framework is proposed to improve SPL by analyzing defects in FM using an ontological rule-based approach. The problem of enhancing the quality of software products derived from SPLE is tackled by dealing with defects due to redundancy, anomaly, inconsistency and wrong cardinality in FMs. The classification of FM defects is defined in the form of cases. The proposed approach formalizes FM by transforming it

into first-order logic (FOL) predicate based feature model ontology (FMO) and it also provides a communication between FM representation and FMO. A set of FOL rules is developed and applied using Prolog on the FMO to deal with defects in FMs. These rules identify defects with their causes and provide corrections in a user-friendly natural language which are easily understandable by PL developers. This information assists developers to eliminate relationships involved in the source of defects to resolve defects. The proposed approach is validated using a corpus of 108 models which includes the maximum size of real-world FMs and automatically-generated 3-CNF-FMs (considered as the toughest benchmark in SPL) available in online software product lines online tools repository as well as models generated using the FeatureIDE tool. The results of experiment evaluation show that the approach is effective, accurate and scalable as it handles defects in large-sized FMs up to 30,000 features. As defects are found and resolved early in the development process of FM, consequently, it allows deriving defect free valid software products from PL by subsequently enhancing the reusability and quality of FMs in SPL.

Keywords: Software product line, Product line model, Feature model, Feature model ontology, Rule-based approach, Ontology, Dead feature, False-optional feature, Void model, Inconsistency, Redundancy, Wrong cardinality, Defects, First-order logic, Identification, Cause, Correction, Reusability, Quality.

PREFACE

This manuscript presents my research work done at the Department of Computer Science & Engineering, Thapar Institute of Engineering & Technology, Patiala, Punjab, India from July 2013 to October 2017. This manuscript mainly focuses on designing and development of a generic framework to improve SPL using an ontological rule-based approach. The work presented in this manuscript is as follows:

1. Studying, analyzing, and summarizing existing approaches for defects in FMs, including their identification, explanation of their causes and corrections.
2. Classification of FM defects in the form of cases.
3. A generic framework to improve SPL by analyzing defects in FM using an ontological rule-based approach.
4. Identifying defects in FMs with their causes and corrections.
5. Causes and corrections in a user-friendly natural language assist PL developers for resolving defects.
6. Deriving defect free valid software products from SPL.
7. Improving reusability and quality of FMs in SPL.

LIST OF PUBLICATIONS

JOURNAL PUBLICATIONS

1. **Megha Bhushan** and **Shivani Goel**, Improving Software Product Line using an Ontological Approach, Sādhanā, volume 41(12), 1381-1391, 2016. DOI: 10.1007/s12046-016-0571-y [**SCIE Indexed**, Impact Factor: 0.465]
2. **Megha Bhushan, Shivani Goel** and **Karamjit Kaur**, Analyzing Inconsistencies in Software Product Lines using an Ontological Rule-Based Approach, The Journal of Systems and Software, volume 137, 605-617, 2018. DOI: http://dx.doi.org/10.1016/j.jss. 2017.06.002 [**SCIE Indexed**, Impact Factor: 2.444]
3. **Megha, Shivani Goel** and **Ajay Kumar**, Improving Quality of Software Product Line by Analyzing Inconsistencies in Feature Models using an Ontological Rule-Based Approach, Expert Systems, 2017. DOI: 10.1111/exsy.12256 [**SCIE Indexed**, Impact Factor: 1.18]
4. **Megha, Shivani Goel** and **Ajay Kumar**, A Classification and Systematic Review of Product Line Feature Model Defects, Software Quality Journal. [**Communicated**, **SCIE Indexed**, Impact Factor: 1.816]

CONFERENCE PUBLICATIONS

5. **Megha**, Defects in Software Product Line: A Review, Poster presentation at ACM-W India Celebration of Women in Computing 2016 (AICWiC'16), 6-7 October, 2016, University of Petroleum and Energy Studies, Dehradun, India.
6. **Megha**, **Arun Negi** and **Karamjit Kaur**, Method to Resolve Software Product Line Errors. In: Kaushik S., Gupta D., Kharb L., Chahal D. (eds) Information, Communication and Computing Technology. ICICCT 2017. Communications in Computer and Information Science, volume 750, 258-268, 2017, Springer, Singapore. DOI: https://doi.org/10.1007/978-981-10-6544-6_24 [SCOPUS Indexed]
7. **Megha, Shivani Goel, Ajay Kumar** and **Arun Negi**, Managing Software Product Line using an Ontological Rule-Based Framework, International Conference on Infocom Technologies and Unmanned Systems (ICTUS'2017), IEEE, ADET, Amity University, Dubai, UAE, 18-20 December, 2017.
8. **Megha**, An Ontological Rule Based Method to Improve Software Product Line, Poster presented and awarded Student Presenter Scholarship, Grace Hopper Celebration India (GHCI) 2017, 16-17 November, 2017, Bangalore, India.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

BDD: Binary decision diagram

BFM: Basic feature model

CBFM: Cardinality-based feature model

CLP: Constraint logic programming

CNF: Conjunctive normal form

CP: Constraint program

CRS: Contradictory relationship set

CSP: Constraint satisfaction problem

DE: Domain engineering

DL: Description logic

DSL: Domain specific language

EFM: Extended feature model

EMF: Eclipse modeling framework

FaMa: Feature model analyser

FD: Feature Diagram

FM: Feature model

FMO: Feature model ontology

FMT: Feature modeling tool

FODA: Feature oriented domain analysis

FOL: First-order logic

MCSes: Minimal correction subsets

OVM: Orthogonal variability model

OWL: Web ontology language

PL: Product line

PLM: Product line model

SPL: Software product line

SPLE: Software product line engineering

SPLOT: Software product lines online tools

SWRL: Semantic web rule language

SXFM: Simple XML feature model

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: Introduction

A software product line (SPL) is a family of related software intensive systems, sharing a common and managed set of features that fulfill the exact requirements of an appropriate market segment [1]. The objective of software product line engineering (SPLE) is reusability of software products to improve the quality and productivity while reducing cost as well as time to market. It allows several organizations to amplify the quality of software products by diminishing development cost and time. For instance, Microsoft offers some well-known product lines (PLs) such as Microsoft Office, Windows, Visual Studio and Skype. Organizations such as Motorola and Nokia have attained their business goals using SPL approach.

Although, other variability models for modeling commonality and variability exist, such as decision models [2], dopler variability models [3], orthogonal variability models (OVMs) [4] and textual variability language [5]. However, feature model (FM) is the most popular variability model for SPL that illustrates the features and their relationships [6]. A feature is defined as property, characteristic or requirement of a software system relevant to the user. A feature can be a chunk of code (feature-oriented programming) [7], a component in architecture [8], a requirement [9], or a non-functional characteristic (quality). For instance, *call waiting* and *call forwarding are two* basic features of a phone. Another instance, a car model incorporates features such as color which can be red, black, blue, etc. A valid and distinct combination of features is used to represent each product in SPL, where a feature set characterizes each software product. A high-quality FM provides accurate combinations of features.

Software Engineering Institute identifies 29 practice areas for the success of SPL strategy [10]. These practice areas are grouped into technical management, software engineering and organizational management, where each group includes several essential activities for the development of valid software products in SPL, e.g. ensuring integrity of the PL, handling the challenge of customer satisfaction by eliminating poor customer interface. An SPL based organization needs to manage several aspects of its customer interface, including identifying needs, providing technical support and providing information.

Testing is an important process for finding defects in software products, verifying and validating these products for a successful SPL. The maintenance of PL becomes more challenging due to high-level reusability. Growing PL consists of activities such as functionality evolution, bug fixing, performance optimization which diminishes reusability [11]. Many issues are encountered during the development of SPL, such as:

(a) overloaded software development system,
(b) poor inter-group communication,
(c) division of personnel in multi-directions,
(d) underutilization and overutilization of resources,
(e) limited awareness of new technological trends,
(f) painful transition,
(g) lack of management support,
(h) inadequate technical and administrative effort,
(i) an excessive number of inter-module relationships within the architecture,
(j) multi-tier architectures in PLs, leading to complicated software systems, and
(k) reusability decreases with developing PL which includes bug fixing, low performance etc.

However, the quality of software products is another essential part for developing PL. Defects arising from the development process can affect all products in the PL which further leads to the development of defective new products. Therefore, it is of utmost importance to ensure quality during each phase of the development cycle. The quality of products can be improved by reducing the load of the development team. There can be some defects when multiple modules are configured individually by several developer teams. The configuration of modules is based either on the selection of features by developer teams or the existing selection rules as per the goals of developer teams, their experience and evaluation criteria to develop large-scale SPL. It leads to the misuse of business resources, reduced productivity and conflicts with customer requirements. Further, the conflicts between developer teams restrict the software development process. It becomes more complex with changing customer requirements for product.

Figure 1.1 depicts a car FM available in online software product lines online tools

(SPLOT[1]) repository. It consists of features, namely, *Carbody*, *Engine* and *Gearshift*. *Engine* can be *Gasoline*, *Electric* or both. *Gearshift* can be *Automatic* or *Manual*. For instance, a mutual exclusion between *Carbody* and *Gasoline* would result in a defect as both cannot exist simultaneously. Another example includes the business units affected by defects at the Samsung Tech Advanced Research Center, Georgia Tech. Several case studies [4, 12, 13] have also highlighted the problems due to defects such as LG Industrial Systems [14], Nokia [15-17], etc.

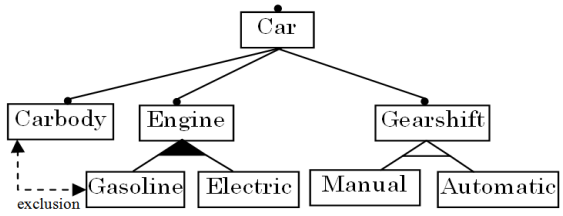


Figure 1.1: Car feature model from SPLOT repository.

In another scenario, multiple architects work in parallel on their workstations and produce a single architectural model with different versions. These versions must be merged episodically to enable detection of errors and collaboration between these architects. For instance, merging of different software products or product variants (i.e. all products undergo consistent evolution to fulfill their relevant environmental changes and customer requirement changes) developed by a company to facilitate reusability between the products. The merging of the model without the help of a tool may result in inconsistent syntactic or semantic version.

The number of defects and complexity of FM increases with growing number of features and relationships. Developing FMs might also incorporate inaccurate relationships among features which cause numerous defects in models such as defects due to redundancy, anomaly, inconsistency and wrong cardinality. These defects arise due to the incorrect combination of features derived from multiple software products to develop a new product. Moreover, a defective model will not allow deriving valid software products, as these products are derived by reusing models which may include defects. Therefore, it is of utmost importance to ensure quality during each phase of the software development cycle.

[1]http://www.splot-research.org/

Although several methods exist that have identified defects [18-33] but only a few methods have provided an explanation for their causes in a manner which is difficult to understand by developers [34-42]. Rincón et al. [43] have only identified and provided explanations for false-optional and dead features using natural language. However, there is no information for the correction of defects. Thüm et al. [44] have only detected the constraints involved in dead features along with false-optional features and Elfaki [45] has only prevented indirect inconsistency.

In this chapter, Section 1.1 briefly describes the preliminary concepts used throughout the rest of the thesis. Sub-section 1.1.1 focuses on SPL. Sub-section 1.1.2 describes the well-known notations of FM using an example. Sub-section 1.1.3 describes various types of defects in FM with examples. The problem statement and research motivations are described in Section 1.2. The research objectives are presented in Section 1.3. Section 1.4 summarizes the research contributions. Finally, Section 1.5 describes organization of thesis.

1.1 Preliminary Concepts

Following preliminary concepts are used in this thesis:

1.1.1 Software Product Lines

An SPL is a set of related software products sharing some common functions that address specific requirements of a certain domain [1]. These software products are built using a common set of core assets in a predefined manner. Software reuse is the main goal behind SPL for enhancing the quality and productivity of software products by diminishing cost and time to market.

1.1.2 Feature Models

In SPLE, feature modeling is one of the famous notation used for representing the variabilities and commonalities in the form of relationships among the features. Initially, Kang et al. [6] formulated the concept of FM as a part of the feature oriented domain analysis (FODA) method. FM is a hierarchical structure where node and edge represent feature and relationship between two features respectively. Subordinate and superordinate

feature are known as child and parent feature respectively. The root feature represents the entire SPL and it is mandatory to be incorporated in all valid products of the PL.

Basic feature model (BFM), cardinality-based feature model (CBFM) and extended feature model (EFM) are the well-known FM notations used to describe SPL [46]. FM notations are described using FM of a laptop system as shown in Figure 1.2, where "*Laptop*" is the root feature. The cross-tree constraints and cardinalities are intentionally injected in this FM to describe defects in FM. Following are the well-known feature modeling notations:

(i) Basic feature model: It explains the basic relationships between features. Figure 1.2 illustrates a FM which is motivated by the laptop system, and it describes several relationships in BFM. Table 1.1 illustrates the types of relationships in BFM with examples using Figure 1.2.

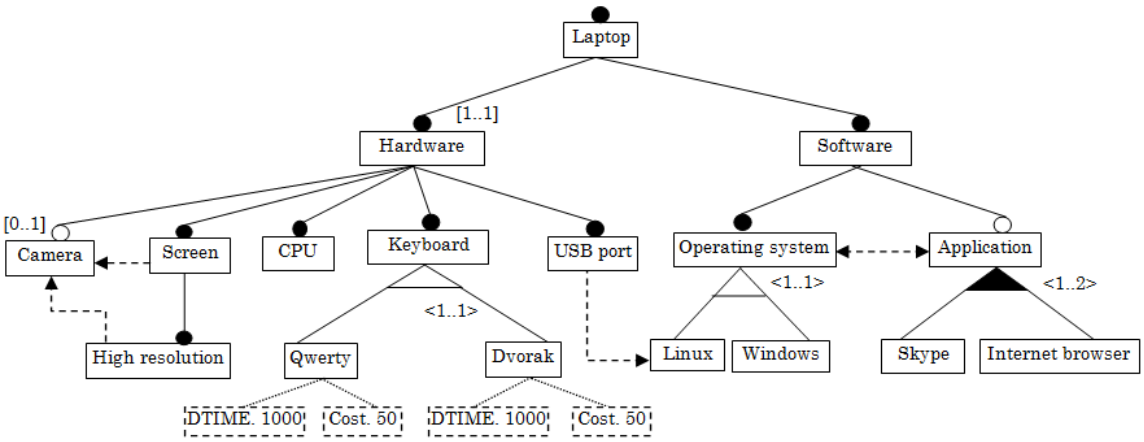


Figure 1.2: Feature model of laptop system.

(ii) Cardinality-based feature model: In CBFM, BFM is extended with UML-like multiplicities of the type [min..max] where *min* and *max* denote the lower and upper bound respectively [48]. This model describes the feature and group cardinalities as shown in Table 1.2. In Tables 1.1 and 1.2, p, c, c1, c2 and c3 represent parent, child, first child, second child and third child feature respectively.

(iii) Extended feature model: In EFM, feature attributes are used to provide additional information regarding features [47]. Attributes can be used to model the evident properties of the product in FM. This model is also called as advanced or attributed

FM. For example, COST and DTIME are the attributes for all sub-features of *Keyboard* as shown in Figure 1.1. These attributes are assigned integer values in the domain.

Table 1.1: Types of relationships in basic feature model.

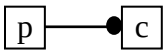
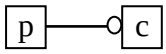
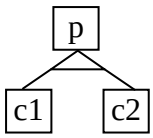
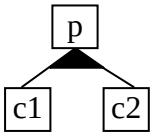
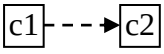
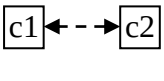
Notation	Type of relationship	Example
	**Mandatory** [6]: Child feature *c* must be incorporated in all the valid products whenever its parent feature *p* is included and vice versa.	*Hardware* is mandatory to be incorporated whenever *laptop* is selected.
	**Optional** [6]: Child feature *c* may or may not be incorporated in the valid products associated with its parent feature *p*.	*Camera* is optional to be incorporated whenever *Hardware* is selected.
	**Alternative** [46]: Child features *c1* and *c2* are in alternative relationship with their parent feature *p* when exactly one of the child features has to be incorporated for developing any valid product whenever *p* is incorporated.	*Software* for *Laptop* may include support for *Linux* or *Windows* but only one of them can be selected as *Operating system*.
	**Or** [46]: Child features *c1* and *c2* are in or-relationship with their parent feature *p* when one or more child features are included for developing any valid product whenever *p* is incorporated.	Whenever *Application* is selected, *Skype, Internet browser* or both can be selected.
	**Implication** [47]: Features *c1* and *c2* related with implication constraint represent that *c2* should be incorporated in all the valid products with *c1*.	Implication between *USB port* and *Linux*
	**Exclusion** [47]: Features *c1* and *c2* related with exclusion constraint cannot be incorporated simultaneously in the same valid product.	Exclusion between *Operating system* and *Application*

Table 1.2: Types of relationships in cardinality-based feature model.

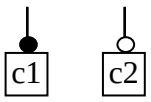
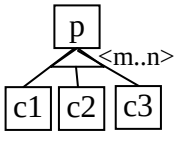
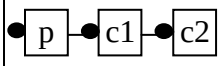
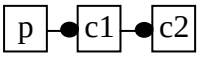
Notation	Type of relationship	Example
	**Feature cardinality:** It is represented by an interval [*min..max*] where *min* is the lower bound and *max* is the upper bound. These intervals indicate the count of occurrence of a single feature as a part of the product. Feature cardinality of mandatory feature *c1* and optional feature *c2* are [1..1] and [0..1] respectively.	*Hardware* and *Camera* have feature cardinality [1..1] and [0..1] respectively.
	**Group cardinality:** It represents the minimum (*m*) and the maximum (*n*) number of child features (*c1..c3*) grouped in a cardinality <*m..n*> that can be incorporated in a valid product whenever their parent feature *p* is selected. A child feature can be included if the parent feature is selected too.	*Skype* and *Internet browser* are in or-relationship with group cardinality <1..2>. *Linux* and *Windows* are in alternative relationship with group cardinality <1..1>.

Table 1.3 describes the general concepts [21, 47] used throughout the thesis for representing defects, their identification, explanations of their causes and corrections.

Table 1.3: Description of general concepts.

Notation	Description
	**Full-mandatory feature:** A feature *c2* is a full-mandatory feature if the feature *c2* and all its predecessor features (i.e. *p* and *c1*) in the FM are also mandatory. Here, *p* can be root feature.
	**Relative full-mandatory feature:** A feature *c2* is called a relative full-mandatory feature to *p* if *c2* itself is a mandatory feature and has all mandatory features on its way to predecessor feature *p*. However, *p* may or may not be a mandatory feature.

1.1.3 Feature Model Defects

A defect is an undesirable property of relationship and it occurs if the product illustrated by FM does not correspond to its SPL [49]. The cross-tree constraints in a FM lead to redundancy, anomaly and inconsistency [47]. These defects affect the semantics as well as ability to represent all the valid products of FM [25]. Defects are also referred as errors or deficiencies [41, 49]. The cross-tree constraints and cardinality are intentionally induced in the models and these models are used as examples to illustrate types of defects in FM. Furthermore, we introduce wrong cardinalities as another form of defect in FM.

(i) Redundancy: A relationship modeled in numerous ways leads to redundancy in FM. It increases the complexity of the model and computational efforts required for deriving products, and decrease the quality as well as maintainability of the model as changes must be applied to all redundant occurrences. Thus, it is considered as a light issue in FM [41].

Redundant constraints: Redundancy arises whenever the elimination of any information does not affect the semantics of a FM. For instance, the elimination of a redundant cross-tree constraint does not affect the validity of configurations. Redundancy can occur due to various sources such as

(a) when a child feature appears several times in a FM with different parent features. However, FeatureIDE[2] hinders the construction of features with identical names. Therefore, this type of redundancy is not considered in this thesis, and

(b) combinations of relationships in a domain within FM and cross-tree constraints [41].

[2]https://featureide.github.io/

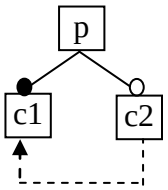


Figure 1.3: A case of redundancy.

***Example 1.1*:** In Figure 1.3, mandatory feature *c1* is implied by optional feature *c2* is redundant as *c1* is always selected.

(ii) Anomaly: Erroneous configuration of instances in a FM leads to an anomaly. It emerges due to unrealistic modeled information induced by wrongly captured domain. The combinations of relationships in FM and cross-tree constraints are accountable for the unrealistic models. Due to the presence of anomalies in a FM, it is difficult to select features for deriving a valid product [41]. It is considered as a medium issue in FM. Following are the defects in FM related to anomalies:

(a) ***Dead features***: A feature which do not appear in any valid product derived from the PL even if it appears in FM is known as a dead feature. It occurs due to the incorrect applicability of cross-tree constraints. It is also called as a not selectable feature [36].

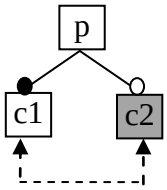


Figure 1.4: A case of dead feature (grey feature is dead).

Example 1.2: In Figure 1.4, optional feature *c2* is dead because full-mandatory feature *c1* excludes it. As a result, *c2* can never be chosen in any valid product of the FM.

(b) ***False-optional features***: A feature included in every valid product of a PL even though it has not been modeled as mandatory is known as a false-optional feature. This defect is associated with an anomaly in FM [41].

Example 1.3: In Figure 1.5, optional feature *c2* becomes false-optional as it is implied by full-mandatory feature *c1*.

(iii) Inconsistency: Inconsistency occurs due to contradictory information in the FM. A valid product can be derived from a FM which incorporates anomaly or redundancy,

but a FM with inconsistency will not allow deriving a valid product. A FM with inconsistency inevitably affects products of the PL. Therefore, inconsistencies are regarded as a very severe issue. Von der Maßen and Lichter [41] identified inconsistency based on two levels:

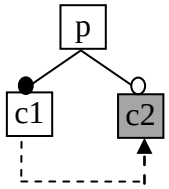


Figure 1.5: A case of false-optional feature (grey feature is false-optional).

(a) Domain level: It occurs due to contradictory information in FM. For example, implication and mutual exclusion between two features.

(b) Product configuration level: It occurs due to incomplete or conflicting product configurations. For example, full-mandatory features have not been chosen for a product configuration.

Void feature models: A void FM does not define any product [49]. In this model, each feature is dead, including the root due to contradictory relationships [49]. It is also known as a consistency checking model, inconsistent model, model satisfiability checking, validation model, invalid model, solvability checking model, unsatisfiable model and model constraints checking [46].

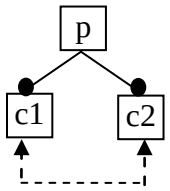


Figure 1.6: A case of void model.

Example 1.4: Figure 1.6 shows that the mutual exclusion between two full-mandatory features *c1* and *c2* makes the selection of both features impossible which leads it to be a void model.

(iv) Wrong cardinality: Wrong cardinality occurs whenever one or more values of the cardinality used in a set-relationship are not attainable. If group cardinality is never used in any product, it leads to wrong cardinality [36]. It usually appears due to

cross-tree constraints in CBFMs.

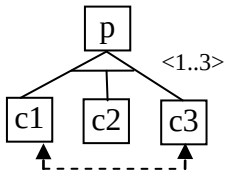


Figure 1.7: A case of wrong cardinality.

Example 1.5: Figure 1.7 shows that the child features *c1, c2* and *c3 are* associated in a group cardinality <1..3>. The value of maximum cardinality 3 is incorrect since features *c1* and *c3* exclude each other.

1.2 Problem Statement and Research Motivation

A defect in FM is a critical issue in the SPL community as software products are derived by reusing models. One of the major factors behind the successful derivation of defect free valid software products from SPL is the quality of FM, which in turn depends on how a defect in FM is resolved. Moreover, manually inspecting defects in large-sized FMs is a laborious task. Therefore, it is of utmost importance to resolve defects to support reusability in the industrial domain. As, it further leads to mass production of software products and provides the benefits of SPL, i.e. reduced development time, cost and improves quality of software products. The identification of a defect in FM is well researched, however, providing a cause of the defect with its correction in a language which is easily understandable by product line (PL) developers is still a challenge. The lack of methods to explain the causes and corrections of defects in a user-friendly language for resolving defects motivated us to propose an effective approach to assure the quality of FMs in SPL, which in turn ensures the benefits of reusability in industry.

1.3 Research Objectives

The aim of this thesis is to accomplish following objectives:

1. To study various existing approaches for defects analysis in feature models (FMs).
2. To analyze defects in existing FMs.
3. To design and develop a generic framework for improving software prodcut line (SPL).
4. To verify and validate the developed generic framework by using case study.

In addition to the above mentioned objectives following objectives are also achieved:

5. Transformation of FM into an equivalent formalism for performing operations on it.
6. Classification of FM defects due to redundancy, anomaly, inconsistency and wrong cardinality in the form of cases and to resolve defects by providing corrections.

1.4 Research Contributions

Following are the contributions to accomplish the **Objective 1:**

(i) Studied, analyzed and compared various existing approaches for analyzing defects in FMs.
(ii) In particular, collected knowledge related to the various types of defects in FM, identification of defects with their causes and corrections.

Following are the contributions to attain the **Objective 2:**

(i) Analyzed various types of defects in FMs.
(ii) Classification of defects in the form of cases.
(iii) Defined and graphically represented each case of the defect with its cause.

Following are the contributions to accomplish the **Objective 3:**

(i) Designed a generic framework to improve SPL by identifying all defects in FM with their causes and corrections in a user-friendly language.
(ii) Developed and implemented the framework for all defects using different real-world FMs from SPLOT repository.

Following are the contributions to attain the **Objective 4:**

(i) Verified and validated the developed generic framework for all defects by using different case studies in the form of real-world FMs from SPLOT repository.
(ii) A corpus of 108 models, which included maximum size of real-world FMs as well as automatically-generated 3-CNF-FMs from SPLOT repository and models generated using the FeatureIDE tool was used as a benchmark to validate the framework.
(iii) Additionally, the framework has been evaluated using accuracy, performance analysis and evaluation.
(iv) The scalability of the framework has been shown by applying it on large-sized FMs.

(v) A comparative study has been carried out with existing approaches.

Following are the contributions to accomplish the **Objective 5**:

(i) Transformed FM into predicate-based feature model ontology (FMO).

(ii) Implemented the transformation process and evaluated its accuracy for all defects using different case studies in the form of real-world FMs from SPLOT repository.

Following are the contributions to attain the **Objective 6**:

(i) Classification of FM defects in the form of cases.

(ii) Defined and graphically represented each case of defect with its cause.

(iii) Provided corrections in a user-friendly natural language for resolving defects.

1.5 Thesis Organization

This thesis is devoted to design and develop a generic framework for improving SPL. It analyzes defects in FMs using an ontological rule-based approach. The results obtained are discussed in chapter 5. Complete chapter-wise summary of thesis work is given as follows:

- **In Chapter 1**, preliminary concepts which include definitions with basic notations, problem statement, motivation, objectives and contributions of research work have been presented.
- **In Chapter 2**, the relevant literature is divided into two categories. The first category is related to the representation of FMs in SPL using ontologies. The second category is associated with the analysis and validation of FMs that deals with defects due to redundancy, anomaly, inconsistency and wrong cardinality in the SPL.
- **In Chapter 3**, an overview of the proposed framework that analyzes defects in FMs has been given. The framework is divided into three stages (i) Stage 1: Transformation of FMs into predicate-based FMOs, (ii) Stage 2: Development and applicability of first-order logic (FOL) based rules, and (iii) Stage 3: Identification of defects, their causes and providing corrections. Various cases of different types of defects in a FM which are supported by the framework are also described. Further, each case of defect is represented graphically for easy interpretation. These defects are also explained with the help of running examples.

- **In Chapter 4**, the methodology followed by proposed approach has been presented. The transformation of FM to FMO is explained based on two-step model transformation, (i) SPLOT Format to the FeatureIDE Format, and (ii) FeatureIDE Format to FMO. The FOL-based rules are developed and illustrated to deal with FM defects in SPL where each rule includes facts in the form of FMO, rule and result. The result includes identified defect with their causes and provides corrections in a user-friendly natural language.
- **In Chapter 5**, the results of experiments carried out to evaluate the proposed approach have been given. The resources and environment used for the experimental evaluation of the proposed approach are also given. The approach is evaluated based on the accuracy, computational scalability, and performance analysis and evaluation. Further, the accuracy is evaluated based on the correct (i) transformation of FMs into predicate-based FMOs, and (ii) identification of defects along with their causes and corrections. The rules are also evaluated; in particular, completeness, consistency and consistency gain of the set of rules, and minimalism of the set of rules. The results of comparative analysis obtained after comparing the proposed approach with existing approaches are also discussed. Additionally, the accuracy of proposed approach is compared with FeatureIDE tool. Finally, threats to validity of the proposed approach are also explained.
- **In Chapter 6**, finally, conclusions and future work are presented followed by references.

CHAPTER 2: State of the Art

The main focus of this thesis is to develop a framework for improving SPL by analyzing defects in FM using an ontological rule-based approach. In this chapter, we have studied and compared various existing approaches to analyze defects in FMs contributing to objectives 1 and 2. Consequently, the relevant literature is divided into two categories. Section 2.1 discusses the first category, related to the representation of FMs in SPL using ontologies. Section 2.2 represents the second category, associated with the validation of FMs that deals with defects due to redundancy, anomaly, inconsistency and wrong cardinality in SPL. Finally, the chapter is summarized in Section 2.3.

2.1 Representation of Feature Models using Ontologies

Many existing notations for representing FMs have been proposed such as UML [50], constraint satisfaction problem (CSP) [51], domain specific language (DSL) [52], higher-order logic [53], extensible markup language, propositional and FOL [18] and description logic (DL) [54]. The main drawback of UML is the use of object constraint language only as a validation tool because it is carried out over a particular object or an object structure [55]. DSL being a domain specific programming language that does not give solutions to general problems [56] and therefore, does not provide standardization. All real-life cases cannot be represented using propositional logic due to the absence of variables [57]. These restrictions have encouraged the use of FOL for validating SPL [45, 58-61] as it is more representative and manages most of the real-life cases [62].

In the Artificial Intelligence domain, knowledge is represented in a formalism and the inferencing capability is used to extract information from it. Researchers have represented FMs based on ontology. An ontology is defined as “a formal, explicit specification of a shared conceptualization” [63]. In the context of expressiveness, ontology is more powerful and richer than a feature. Ontologies, like FMs are also useful in identifying and defining the basic concepts of domain and the relationships among them. Lee et al. [64] worked on analyzing the semantic similarity of the FM. They represented FM using ontology and analyzed the variability and commonality in it. Wang et al. [34] identified invalid configurations in FMs using web ontology language (OWL).

The causes of invalid configurations were explained using ontological terms and an OWL debugging tool. Moreover, they provided the analysis and visualization of FMs using CASE tools. Abo et al. [65] represented FM using an OWL-based ontology. The consistency of FM is validated with Semantic Web Rule Language (SWRL) rules. They resolved conflicts using an automatic mechanism and focused on integrating FMs while representing different views of a PL. However, their work did not include the analysis of FMs.

OWL-DL is a formal language for knowledge representation in terms of ontologies. However, it is inefficient to deal with expressive ontologies like DOLCE [66], Cyc [67] and SUMO [68]. OWL is not expressive: (a) to characterize process models based on web service, and (b) to represent entire as well as only intended interpretations of the particular process model [69]. Like other languages for process modeling, the process model based on OWL should be more expressive for easy interpretation to fix ambiguities. Berardi et al. [70] developed FOL predicates to overcome these limitations since FOL is a more expressive logic-based language.

Similar to OWL-DL, FOL is also a popular expressive formalism to be used for reasoning on ontologies [71]. FOL ontology is used to describe a process model which allows development of a framework for web service process modeling [70]. This framework automates the configuration and searching of web services. Later, an impressive growth of first-order theorem provers was observed in computing inferences that are not handled by DL reasoners [72]. It facilitates better understanding and implementation of the reasoning tasks. First-order language is a predicate-based ontology language for modeling a FM which includes classes of objects and properties represented using binary and ternary predicates respectively [73, 74]. Gruninger and Menzel [75] extended the process specification language to develop a FOL based ontology for web services (i.e. FLOWS). It provides a higher level of expressiveness than OWL-DL, for instance, it instantly incorporates *n*-ary predicates along with rich quantified sentences. Further, this ontology provides a framework to model the semantics of web services and enables reasoning on it [69]. It represents different types of data flow from web services in a more refined manner using FOL than in OWL-S. FOL overcomes the issues related to the expressiveness of OWL-S through its rich expressiveness level and well-known

semantics based on model-theoretic [70]. Therefore, the above limitations motivated us to transform FM into FOL predicate-based ontology as it increases the expressiveness and provides formalization. In other cases, ontology is translated into FOL such as SUMO ontology [76], a part of OWL 2 Full [71,77], Cyc ontology [78] and Adimen-SUMO ontology [79]. The consistency of first-order ontology is checked using the model generation system based on logic programming [80].

2.2 Automated Analysis of feature models

Many approaches focus on validating FMs to deal with defects due to redundancy, anomaly, inconsistency and wrong cardinality. Defects can be solved by debugging the defective FM and providing a cause and correction to fix it.

Von der Maßen and Lichter [47] were the first ones to resolve defects in FMs. However, they have not provided any automated support for it. Both inconsistencies and anomalies are specific cases of false-optional and dead features. They provide some details (i.e. corrective explanations) about how to fix dead features, false-optional features and redundancies. The details provide information to delete redundantly modeled relationships, and an exhaustive analysis is required to resolve false-optional features. Anomalies in the form of dead features can be resolved by deleting relationships that cause these defects by changing the feature's semantics, e.g. changing a full-mandatory feature to an optional feature, and to restructure the feature relationship, i.e. to change optional, mandatory, alternative and or-relationship of the feature. Requiline tool is presented to detect inconsistencies. Fan and Naixiao [19] formalized FM to identify inconsistencies using DLs. Their technique only identified inconsistencies, but did not find any solution to fix them.

Trinidad et al. [81] proposed a method based on finding each product and then exploring unused features to detect dead features. Trinidad et al. [49] used solvers for the validation of FMs. They proposed a framework based on constraint programming and reiter's theory of diagnosis to automate error detection. The framework detected void model, false-optional features, dead features and explanations by converting FM to a diagnose model and then into a CSP. It is based configuring all products in the SPL. The validation of large-sized FMs using solvers may take the inconceivable amount of time for exploring

all products. The explanations involve one or more than one dependencies which can be modified to fix defects. However, there is no automated support provided for the same. The set of dependencies and explanations of dead and false-optional features have been identified by automating their approach in Feature Model Analyser (FaMa) [82]. Hemakumar [83] detected inconsistency in FMs using a connection between propositional logic and context-free grammar. The author identified a direct inconsistency in the configuration process of a software product and concluded that the time needed to identify an inconsistency even in a small-sized FM is significantly high in general.

Knowledge-based method along with FOL rules is used by (a) Osman et al. [84] to validate FM, identify inconsistencies, dead features and explanations of these defects. However, they have not established its comprehensiveness. The explanations are given during the configuration process of an SPL with the selection of invalid feature combinations. The explanations suggest users to select or deselect specific features, and (b) Elfaki [45] and Elfaki et al. [85] to identify and prevent inconsistency in the domain engineering (DE) process without the requirement of the configuration process. The independent FOL rules given by Elfaki [45] identified three types of inconsistencies as well as prevented direct inconsistency (i.e. feature A implies feature B and feature B excludes feature A). These methods have been validated using their own generated data sets.

Thüm et al. [86] addressed the changes to a FM during evolution by using simplified reasoning to reason about randomly generated FMs with 10,000 features. Gheyi et al. [23] have automatically checked the SPL refactoring of automatically-generated FMs up to 2,000 features during SPL evolution. An automated test data generator is developed to generate test data for the analysis of FMs using metamorphic testing [24]. It detected two defects namely void model and dead feature in each of the tool, FaMa and SPLOT for the automated analysis of FMs. This technique is efficient and effective to automatically detect most of the faults using mutation testing and real faults in a few seconds without the requirement for a human oracle. According to Zaid et al. [37], FMs were analyzed based on semantic web technologies. They used SWRL and OWL for modeling and the

Pellet reasoner [87] for the analysis. Their work only identified inconsistencies and provided explanations in a non-user-friendly format.

Trinidad and Ruiz-Cortés [36] used abductive reasoning for the detection of dead and false-optional features with an explanation of their causes. They computed minimal correction subsets **(**MCSes) of constraints that must be eliminated from the constraint programming to generate an accurate model. But, they only provided a list of constraints and no information is suggested to PL modelers about the occurrence of defects due to these constraints in Product Line Models (PLMs). Transforming models into constraint programs (CPs) could loss structure required for explanations. Thus, techniques based on constraint satisfaction are inadequate for providing these explanations. Therefore, the semantics and structure of PLMs are transformed into ontologies, as both properties are of utmost importance for explaining defects. Unfortunately, they do not give any information regarding the implementation of their work.

Van den Broek and Galvão [35] analyzed FMs using an algorithm based on converting a FM to generalized feature tree and computing some of their properties using the functional programming language Miranda [88]. The cross-tree constraints are eliminated, and features can occur multiple times in a generalised feature tree. The algorithm is efficient with a small number of cross-tree constraints. The limitation of their approach is that the time required to build a generalised feature tree is exponential to the number of cross-tree constraints. Therefore, it is not feasible to transform a large-sized FM to generalized feature tree. Moreover, this work only deals with dead features, explanations and a minimal set of conflicting constraints. They validated their approach only with a feature tree consisting of 2 cross-tree constraints and 13 features. Unfortunately, the computations for the efficiency of their algorithm are purely theoretical, and there is no systematic empirical evaluation is provided for the same.

White et al. [89] presented a methodology that detects and suggests changes in feature configurations to correct inconsistencies in FMs. However, it solves inconsistencies during configuration only. White et al. [90] transformed FM into CSP [51]. The valid configurations are generated by implementing corrective explanations (i.e. by selecting or deselecting features) to fix invalid configurations. They developed a FM generator to

randomly create FMs up to 5,000 features for validating their approach. The limitation of their approach is that the time for diagnosing invalid configurations depends on the constraint's structure in the FM.

Benavides et al. [46] proposed an overview of the automated analysis of FMs which incorporates 30 various analysis operations such as checking the validity of products, defect detection, explanations, optimization and refactoring. A dynamic-priority based mechanism is used by Wang et al. [91] to produce consistent FM on the basis of the priorities assigned to dependencies. Their mechanism automatically suggests researchers a correction to fix inconsistency by removing one or more minimum subset of weaker (i.e. lower priority) dependencies. Since this mechanism only provides one correction per iteration, it must be repeated until the requirements of users are fulfilled.

Salinesi et al. [21] explored errors by presenting a typology of FM verification criteria. It includes 15 verification criteria which are classified as per their nature (conformance checking and domain-specific) or the identified error-type. These criteria were formalized using FOL, then explained with two real cases. Later, Salinesi and Mazo [25] analyzed FMs by representing them in CP. A series of algorithms is proposed that verified the models against the typology of verification criteria [21] in single-view and multi-view PLMs. Their approach only identified void models, false-optional features, dead features, false PLM, redundant constraints and wrong cardinality in FMs. The approach is limited to FMs up to 2000 features, as the solver used in this approach does not allow accommodating more than 5000 variables.

Noorian et al. [39] transformed FM from the simple XML feature model (SXFM) format into OWL-DL file. They identified and provided corrections for inconsistencies in the model using Pellet reasoner [87]. The minimal subsets of OWL axioms generated using this reasoner must be eliminated from OWL-DL file to attain model consistency. The corrections in the form of OWL-DL pure are not easy to understand by modelers, which is a limitation of their approach. Also, evaluation results concluded that the time to identify and fix inconsistencies increases with an increase in the count of features and inconsistencies in test cases. Mazo et al. [92] proposed the conformance checking of PLMs (namely FMs) based on constraint logic programming (CLP). A set of 9

conformance checking rules is identified and applied using GNU Prolog constraints solver. The elements related to each particular conformance rule are tested using CLP in EFMs. Their approach is efficient and scalable to industry size FMs, as it was validated by implementing these rules on 50 FMs up to 10, 000 features.

Guo et al. [93] formalized FM using an ontology. The consistency of large FM is checked by diminishing the impact of changes to the concerned parts using an evolutionary strategy. Their approach has been evaluated using randomly generated FMs up to 10,000 features. Instead of considering the entire FM, only the changed part is considered up to the previous consistent version. This leads to reduced complexity of the problem. However, no automated support is provided for the same.

Felfernig et al. [40] explained the causes of anomalies in FM. They discussed the FMCORE and FASTDIAG algorithms to determine minimal sets for non-redundant constraints and minimal diagnoses respectively. The minimal sets of constraints (conflicts) had to be modified or removed from the FM to achieve model consistency. Each type of defect is explained using FASTDIAG which is independent of any solver. The entire set of diagnoses was determined using Hitting Set Directed Acyclic Graph algorithm proposed by Reiter [93] to detect and fix a conflict. Although, Felfernig et al. [40] recommended corrections for anomalies in terms of inconsistencies and redundancies, yet they have not provided any implementation details. Further, no information was provided for the number or types of defects in the FM. The evaluation of their approach is limited to FMs available at SPLOT repository with 172 features. Additionally, their explanations comprise of constraint sets which are not directly concerning the structural information of the model turning it to be challenging for developers to understand the anomaly. The scalability of FASTDIAG for large-scale FMs is missing.

Ripon et al. [26] modeled SPL variants using UML and verified them using FOL. The variants and dependencies among them are modeled using propositional connectives in the form of logical expressions. The model checkerAlloy tool [95] was used for the validation of these expressions. The validity of product instances and the consistency of feature configuration are automatically checked by the Alloy tool. However, their

approach illustrated the detection of dead features, invalid products and void FMs using a case study. The variability in SPL is represented using FM and OVM together, and further, FOL rules were used by (a) Elfaki et al. [96] to identify redundancies and dead features in DE process. The efficiency of their work lies in the direct detection of redundancy in the DE, and (b) Elfaki et al. [30] to deduce wrong cardinality and false-optional features in DE process. These methods have been validated using their own generated data sets. Results conclude that these methods are scalable up to 20,000 features in a reasonable time.

Giraldo et al. [97] used OWL ontology to represent FM and semantic query-enhanced web rule language to identify dead features and some of their causes in natural language. The approach was validated with FMs generated using BeTTy tool which is a framework for Benchmarking and Testing on the Automated Analysis of Feature Models [98]. Their approach was limited as it handled only two cases of dead features. Later, Rincón et al. [43] represented FMs with ontologies and used SWRL based rules to identify and provide causes for false-optional and dead features. Although, the approach provides explanations using natural language in accordance with the situation of each defect, yet it is only viable for FMs up to 150 features. However, this approach is not able to detect defects other than the dead and false-optional features corresponding to the defined rules.

Javed et al. [99] used FODA maturity model to measure and analyze the quality level of FMs. This approach categorizes and elaborates the FM errors, i.e. inconsistencies, redundancies and anomalies with examples. White et al. [100] developed a solver to model and resolved problems due to the multi-step configuration of emerging FMs. The derivation of configurations can be automated by mapping the problem of SPL configuration to a CSP. Additionally, it enables researchers to automatically reason about the evolution of PL over time.

Rincón et al. [101] transformed FMs into CPs. They identified defects such as false PLs, false-optional features, dead features and redundancies by analyzing CP using algorithms given by Salinesi and Mazo [25]. They identified MCSes (i.e. a minimal subset of relationships) of an unsolvable CP for detecting potential corrections of FM defects. MCSes should be filtered out of the FM to fix the minimum single defect. Their method

identified all MCSes for defects by systematically removing relationships from the FM. The corrective solutions for void model include eliminating the constraints set to make CP solvable. Dead features can be corrected by eliminating constraints to permit the configuration of products including this feature. Similarly, the corrections for false-optional features include eliminating a set of constraints to permit the configuration of products without this feature. The identified corrections for redundancies include the elimination of constraints without changing the semantics of the model. The corrections for false PL include: (a) fixing false-optional and dead features and allowing configuration of more products from the PLM to correct false PL, and (b) non identification of any correction of the defect if the model is a false PL as each of its feature is mandatory, on the basis of assumption that it was intended. Designers would easily understand the identified corrections, even without knowing constraint programming. Designers can decide on the correction according to their interests and possible MCSes. However, their method only provides corrections by removing relationships from the FM. Their method is evaluated using 78 models with a varied number of features, up to 120 dependencies.

Perez-Morago et al. [32] presented an algorithm to identify core and dead features using binary decision diagram (BDD). BDD is used to represent boolean functions and encode a variability model. However, the evaluation of their approach is limited to FMs with 5000 features. Lesta et al. [42] translated a FM into CSP. Their work detected and explained the contradictions in attributed FMs by adapting QuickXplain algorithm. The approach explains void model, dead and false-optional features based on a constraint solver. The efficiency of their work is evaluated by using only one FM with less than 500 features. Further, the constraint solver must be modified directly, and there is no open-source implementation provided for their approach.

Javed et al. [33] automated the quality detection system using GenVoca layered architecture based on generative programming technique for detecting inconsistencies in FM. The quality detection technique is applied on the FMs with errors. Each component of GenVoca model is implemented in C++. The architecture is based on bottom-up approach. A configurator is implemented for testing the technique which incorporates the configuration information related to each feature. In configurator, the technique is

validated by allowing the user to input the values. Further, inconsistencies are detected by implementing all features. However, their work only detected inconsistencies and there is no information regarding the scalability of the same.

Kowal et al. [102] proposed a generic algorithm to explain various anomalies such as void model, dead features, false-optional features, and redundant constraints in FMs based on predicate logic. It explains each type of anomaly encoded in a conjunctive normal form (CNF) and a set of assumptions based on initial truth values. These explanations are in a user-friendly way. They computed short explanations and benefited developers by highlighting the most significant parts of them that may be the possible source of an anomaly. Due to the support of their open-source tool in FeatureIDE, the scalability is evaluated with industrial-size FMs. However, the evaluation is limited to FMs including only up to 2,513 features and 2,833 constraints.

Various researchers used tools for handling defects in the SPL. Marcilio Mendonca created SPLOT at the University of Waterloo, Canada in 2009 which is currently the largest repository of real SPL FMs (i.e. 430 models or 11, 500 features) [103]. SPLOT allows researchers to analyze, edit, configure, debug, share and download FMs. FMs in SXFM format are read and saved from FM repository available in SPLOT. One of the limitations of the tool is that it neither enables code generation nor provides any means to build a graphical representation of a FM. Another concern is the privacy of models, as models can be easily accessed and customized by anybody. Also, this tool allows researchers to generate only a single product configuration which cannot be stored in the SPLOT repository.

VMWare tool is developed to automatically verify the structural and semantic correctness of cardinality based FMs [20]. The verification criteria such as the identification of inconsistent constraints, poorly defined cardinalities, redundant features and cyclic relationships were implemented by this tool. Each verification criterion is evaluated using graph navigation algorithms on two FMs with 21 and 49 features. However, the results showed the effectiveness of their approach with major issues related to the scalability and extensibility.

Segura et al. [38] developed a FaMa tool suite, dedicated to the verification, editing and

automated analysis of FMs. It supports cardinality-based FMs, the import of FMs from XMI and export of FMs to XML, analysis operations of FMs and GUI for representing models. This tool compares different solvers such as SAT, CSP, BDD and Java. FaMa tool automatically selects the most proficient solver according to the user request for the operation. The tool enables FMs to incorporate numerical constraints and to derive optimal configurations. It also finds the overall number along with the feasible products of a FM, void FM, dead features, valid products and analyzes the feature commonality. This tool includes neither feature interactions nor non-functional property values. Also, instead of generating explanations in natural language, it generates a list of relationships incorporated in the defect that must be modified by modelers to fix the defect. Moreover, it does not provide corrective explanations for defects.

Research Group of GIRO developed a Feature Modeling Tool[3] (FMT) at the University of Valladolid and Burgos, Spain in 2008. It allows researchers to model the features graphically. FMT models and configures FMs within Visual Studio IDE. It provides visualization of an indented list along with tree structure where nodes and links represent features and component hierarchy respectively. The modeler design facilitates addition, deletion or modification of features. Eclipse Modeling Framework[4] (EMF) FM project is a popular open source Eclipse plug-in under the Eclipse Modeling Framework Technology Project. It is developed by a team at the IBM Cary NC lab. EMF is a standard representation of FMs within the platform of Eclipse. EMF defines:

(a) a feature meta-model,
(b) an extendible framework for evaluation engine,
(c) Feature Diagram (FD) editor for graphical representation of FD,
(d) extendible visualizations as well as editors for the FMs, and
(e) additional Eclipse projects.

The main limitation of EMF is the huge count and size of the dependencies in the model at runtime.

[3]http://giro.infor.uva.es/FeatureTool.html

[4]http://www.eclipse.org/proposals/feature-model/

Hydra[5] is an eclipse plugin which is based on Ecore or GMF as a standard feature modeling tools [104, 105]. Further, it provides interoperability among other tools. This tool allows researchers to configure, validate and automatically generate minimal configurations of cardinality-based FM with clonable features [106]. It provides support to the full graphical capabilities, both in configuring and editing FMs. Hydra creates a valid configuration by adding a minimum number of features to a configuration which is ensured as chosen configuration by its constraint solver. The time required for creating a configuration relies on the count of features chosen in the configuration and it is tedious to model the configuration.

Zhang et al. [41] developed a FMV-Tool for generating FMs and their validation. They detected and provided explanations of false-optional and dead features in FM by eliminating contradictory feature relationships. A set of feature relationships which is the cause of defects leading to contradictory configuration conclusion is termed as a contradictory relationship set (CRS). Minimal explanations are generated for defects on the basis of CRSs that cause the defect. It fixes defect by eliminating all the CRSs that cause defect. Eliminating a CRS includes modifying one or more than one feature relationship from the CRS. The optimal solution to remove CRS should consist of single feature relationship in the CRS using cartesian product. The identified minimal explanations are the correct explanations of the defect in Reiter's theory. The corrective solution for dead and false-optional features is to remove cross-tree constraints or to modify hierarchical relationships involved in these defects. The results of a comparative evaluation between FAMA [107] and FMV-Tool tools lead to the effectiveness of FMV-Tool. However, their work did not provide any dataset or metrics in support of the findings and conclusions reported by them.

Leich et al. [108] implemented FeatureIDE tool at the University of Magdeburg, Germany in 2005. It is integrated with Eclipse to directly produce code (C++ or Java) from a FM and graphically represent a model [44]. This tool provides support for the

[5]http://caosd.lcc.uma.es/spl/hydra

complete life cycle of an SPL (i.e. from DE to feature-oriented software development). It allows researchers to edit the model in its editor by importing and exporting different formats. FeatureIDE deals with void models, false-optional as well as dead features in FM by providing corrective solutions. The exclusion and implication relationships involved in the defect are automatically identified by the tool to correct the identified defect. This tool does not support corrective solutions for the defect which requires the elimination of more than one relationship to fix it.

Table 2.1 summarizes the comparison of existing approaches based on various parameters. The parameters are discussed as follows:

(a) Redundancy: It deals with the defects arising due to redundancy in FMs.
(b) Dead features: It deals with the defects arising due to dead feature in FMs.
(c) False-optional features: It deals with the defects arising due to false-optional feature in FMs.
(d) Inconsistency: It deals with the defects arising due to contradictory information in FMs.
(e) Wrong cardinality: It occurs whenever one or more values of cardinality used by a set- relationship are not attainable.
(f) Identification: It deals with the identification of a defect.
(g) Cause: It provides the cause of defects.
(h) Correction: It provides corrections to resolve defects.
(i) Classification: It classifies the defects due to redundancy, anomaly, inconsistency and wrong cardinality in FMs.
(j) Ontology: It deals with the defects using ontology.
(k) Real-world model: In real-world models, the methods to handle defects are evaluated using either real life FMs or industrial SPL FMs
(l) Random models: In random models, the methods to handle defects are evaluated using randomly generated FMs.
(m) Generated data sets: In generated data sets, the methods to handle defects are evaluated using either own data sets or randomly created FMs with their developed tool.
(n) Tool support: Tool is developed for creating FMs and their automated analysis.

(o) Basic FM: Defects in FM occur while defining the basic relationships among features.
(p) Cardinality-based FM: Defects occur in FM extended with cardinalities.
(q) Extended FM: Defects in FM occur while defining additional information related to its features using attributes.
(r) Automated support: It handles defects using an automated approach.
(s) Formalization: A formal approach is used to deal with defects.

Table 2.2 summarizes the articles that suggested corrective explanations on FM defects. It provides details of the technique used and the corrections to resolve defects. It represents the defects corresponding to corrective solutions and '-' in a cell shows that the article of the row does not provide information about the corresponding column heading. The majority of the works suggested corrections only for void FMs.

2.3 Summary

This chapter summarizes various existing works related to the representation of FMs in SPL using ontologies. Further, the relevant literature to analyze, validate and handle FM defects due to redundancy, anomaly, inconsistency and wrong cardinality in SPL is discussed. The existing literature based on various parameters is compared in Table 2.1. Further, corrections suggested by existing works is summarized in Table 2.2.

Table 2.1: Comparison of various existing approaches.

Parameters / Articles	Von der Maßen and Lichter [47]	Fan and Naixiao [19]	Wang et al. [34]	Hemakumar [83]	Osman et al. [84]	Trinidad et al. [49]	Elfaki et al. [85]	Salinesi et al. [20]	Trinidad and Ruiz-Cortés [36]	Van den Broek and Galvão [35]	Zaid et al. [37]	Salinesi et al. [21]	Segura et al. [38]	White et al. [90]	Wang et al. [91]	Mazo et al. [92]	Noorian et al. [39]	Gheyi et al. [23]	Segura et al. [24]	Guo et al. [93]	Salinesi and Mazo [25]	Elfaki et al.[96]	Felfernig et al.[40]	Giraldo et al. [97]	Ripon et al. [26]	Zhang et al. [41]	Elfaki et al.[30]	Rincón et al [43]	White et al. [100]	Thüm et al. [44]	Lesta et al. [42]	Perez-Morago et al. [32]	Rincón et al. [101]	Elfaki [45]	Kowal et al. [102]	Javed et al. [33]
Redundancy	+	-	-	-	-	-	-	-	-	-	-	+	-	-	-	+	-	-	-	-	+	+	+	-	-	-	-	-	-	-	-	+	-	+	-
Dead feature	+	-	-	-	+	+	-	+	+	+	-	+	+	-	-	-	-	-	+	-	+	+	+	+	+	+	-	+	-	+	+	+	+	+	-
False-optional feature	+	-	-	-	-	+	-	-	+	-	-	+	-	-	-	-	-	-	-	-	+	-	+	-	+	+	+	+	-	+	+	-	+	+	-
Inconsistency	-	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	-	+	-	+	-	-	-	+	+	+	-	+	+	+
Wrong cardinality	-	-	-	-	-	-	-	-	+	-	-	+	-	-	-	+	-	-	-	-	+	-	-	-	-	-	+	-	-	-	-	-	-	-	-
Identification	+	+	+	-	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Cause	-	-	+	-	+	+	+	-	+	+	+	-	+	+	+	+	+	-	-	+	-	+	+	+	-	+	-	+	+	-	+	-	-	+	-
Correction	+	-	-	-	+	+	-	-	+	-	-	-	-	+	+	-	+	-	-	+	-	-	+	-	-	+	-	-	+	+	-	-	+	+	-
Classification	-	-	-	-	-	-	-	-	-	-	-	+	-	-	-	-	-	-	-	-	+	-	-	-	-	-	+	-	-	-	-	-	-	-	-
Ontology	-	-	+	-	-	-	-	-	-	-	+	-	-	-	-	-	+	+	-	+	-	-	-	+	-	-	-	+	-	-	-	-	-	-	-
Real-world model	-	-	-	-	-	+	-	-	-	-	-	+	-	-	-	+	+	+	-	-	+	-	+	-	-	-	-	+	-	-	+	+	+	+	-
Random models	-	-	-	-	-	+	-	-	-	-	-	-	-	+	-	-	-	+	-	+	-	-	-	+	+	-	-	+	+	-	-	+	+	+	-
Generated data sets	-	-	-	-	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-	-	-	-	+	-	-	+	-	-	-	-	-
Tool support	+	-	-	-	-	-	-	+	-	-	-	-	+	-	-	-	+	-	+	-	-	-	-	-	-	+	-	+	-	+	-	-	+	-	-
Basic FM	+	+	+	+	-	+	-	-	-	+	+	-	+	+	+	-	+	+	+	+	-	-	+	-	+	-	-	+	+	+	-	+	+	-	+
Cardinality-based FM	-	-	-	-	+	-	+	+	+	-	-	+	+	-	-	-	-	-	+	+	+	-	-	+	-	+	-	-	-	-	-	-	-	-	-
Extended FM	-	-	-	-	+	-	-	-	+	-	+	-	-	-	-	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-	-	-	-
Automated support	-	-	-	-	+	+	-	+	-	-	-	-	+	+	+	+	+	+	+	-	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Formalization	-	+	-	-	+	+	-	-	-	-	-	+	-	-	-	+	+	+	-	+	+	+	+	+	+	+	+	+	+	-	-	-	-	+	-

Here, “+”and “-” indicate support and no support respectively, corresponding to author's work and the mentioned parameter.

Table 2.2: Summary of various existing approaches for corrections.

Authors	Defect (related to corrections)	Techniques	Description of correction	Merits	Demerits
Von der Maßen and Lichter [47]	Void model, Dead features, False-optional, Redundancy	-	• For dead features: delete the irritating relationship, change the semantics of feature and restructure feature • A thorough analysis for false-optional features • To delete redundant relationship	• Provide hints about how to fix defects and tool support	• No automated support • No implementation of corrective solutions
Trinidad et al. [49]	Void model, Dead features, False-optional	Constraint programming	• Relationships involved in explanations can be modified	• Provides a list of relationships to modify	• No automated support • No implementation of corrective solutions
Trinidad and Ruiz-Cortés [36]	Void model, Dead features, Wrong cardinalities	Abductive reasoning	• Void model: remove or relax one or more relationships from explanations • Dead features: change or remove at least one relationship from explanations • Wrong cardinality: remove cardinality or the relationships from explanations	-	• No automated support is explicitly proposed • No implementation of corrective explanations
Wang et al. [91]	Void model	Constraint hierarchy theory with a dynamic-priority based method	• Automatically suggest a solution • More desirable solution based on the priorities assigned to constraints • Delete one or more than one weaker constraints to resolve void FM	• Provides automated support • Provides desirable solutions to defect	-
Noorian et al. [39]	Void model	DL, Pellet reasoner, Minimal subsets (in the form of OWL axioms)	• Remove minimal subsets for the cause of void model to fix defect	• Provides automated support • Corrections for void model	• Difficult to understand the corrections in the form of OWL-DL pure
Guo et al. [93]	Void model	Ontology	• Resolve changes to preserve consistency of the model	• Improve complexity of the problem by not studying entire model but deals with the only part that changes since last version of consistency • Handled changes in model after implementing correction	• No automated support
Felfernig et al.	Void model,	Reiter theory of	• Void model and dead features: delete minimal sets of	• Provides automated	• No implementation of

[40]	Dead features, False-optional, Redundancy	diagnosis	constraints responsible for defect • False-optional features: change relationship of feature involved in this defect • Redundancy: delete redundant relationships from FM without affecting the semantics of model	support	corrections
Zhang et al. [41]	Dead features, False-optional	Reiter theory of diagnosis, Minimal explanations	• Modify right set of relationships to delete the defect • Dead and false-optional: remove crosstree constraints or modify hierarchical relationships involved in the defects	• Provides automated support	• No implementation of corrections
Thüm et al. [44]	Void model, Dead features, False-optional	FeatureIDE tool	• Delete exclude and require relationships involved in the cause of defects	• Provides automated support	• Cannot provide corrections for the defect that requires deleting more than one relationship for fixing it
Rincón et al. [101]	Void model, False PL, Dead features, False-optional, Redundancy	Constraint programming, MCSes	• Eliminate Minimal subset of relationships from the FM to fix defect • Void model: eliminate collection of constraints to make CP solvable • Dead features: eliminate the constraints to permit configuration of products including this feature • False-optional: eliminate constraints to permit configuration of products without this feature • False PL: fix false-optional and dead features to correct false PL • Redundancies: eliminate constraints which are not going to affect the semantics of model	• Provides automated support • Provides all potential and possible corrections for defects	• Detects corrections only by removing relationships
Elfaki [45]	Inconsistency	FOL rules	• Add new constraint rule to the DE process to prevent direct inconsistency	• Provides automated support	• Prevented only direct inconsistency

CHAPTER 3: The Proposed Framework for Improving SPL

The focus of this chapter is to present a framework that analyzes defects in FM using an ontological rule-based approach to improve SPL. In this chapter, Section 3.1 gives a brief overview of the generic framework and its three stages contributing to objectives 3 and 5. Section 3.2 analyzes various types of defects due to redundancy, anomaly, inconsistency and wrong cardinality in FMs to achieve objectives 2 and 6. Additionally, provides the classification of defects in the form of cases, where each case of defect is graphically represented and defined with its cause. These defects are explained with the help of running examples in Section 3.3. Finally, Section 3.4 summarizes the chapter.

3.1 Overview of the Framework

The primary focus of the proposed framework is to provide a foundation for improving SPL by enabling PL developers to produce defect free valid end products from FM. As shown in Figure 3.1, the framework is divided into following three stages

(i) transformation of FM into predicate-based FMO [109],
(ii) development and applicability of FOL-based rules, and
(iii) identification of defects, their causes and providing corrections.

This information would help software engineering community by enabling PL developers to find the types of defects along with their causes and corrections to resolve defects, in order to generate valid products from FMs, thereby enhancing the overall quality of SPL.

Stage 1: Transformation of FM into Predicate-based FMO

At this stage, FM is transformed into FMO using a predicate-based ontology to provide formalization. FM should be incorporated in a comprehensive formalism to represent the PL. It enables representation of the input FM in terms of the features and the relationships between them. Additionally, this formalism enables to deal with defects, i.e. the identification of defects along with their causes and provides corrections. The construction of FMO includes two-step model transformation (i) SPLOT format to the FeatureIDE format, and (ii) FeatureIDE format to FMO. The transformation process

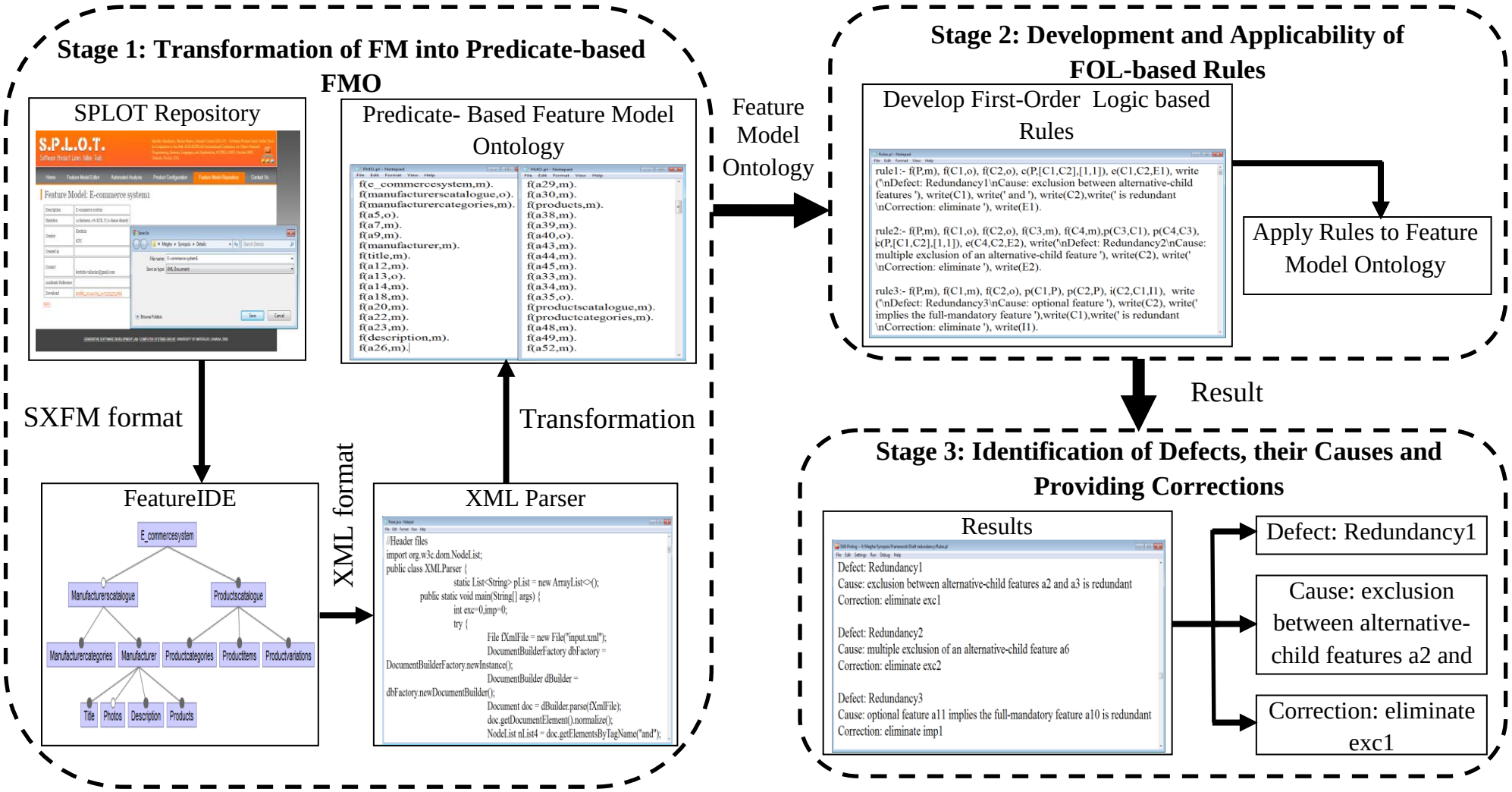


Figure 3.1: Overview of the proposed framework.

describes that a FM available in online SPLOT repository is automatically read by FeatureIDE tool. An XML parser developed in Java [110] that transforms input FM from FeatureIDE format into FOL predicate-based ontology. The generated FMO comprises of facts which correspond to the features and their relationships represented in the input FM.

Stage 2: Development and Applicability of FOL-based Rules

In this stage, a set of FOL-based rules is developed which represents specific cases of misuse among the relationships in a FM that leads to defects. These rules are implemented as FOL queries to the generated FMO using Prolog [111, 112] to handle defects in FM. Each rule is a source for originating the identified defect along with its causes and corrections.

Stage 3: Identification of Defects, their Causes and Providing Corrections

Stage 3 of the framework is represented using *Results* which further comprise of defect, cause and correction as shown in Figure 3.1. In this stage, defects in FM are identified along with their causes in a user-friendly natural language and also provide corrections to resolve defects.

3.2 Defects Supported by Framework

As per literature, FM defects due to redundancy, anomaly, inconsistency and wrong cardinality are classified into sets of possible cases. In order to illustrate each type of defect with its cases, cross-tree constraints implication and exclusion are represented by *imp* and *exc*, respectively. Additionally, a unique name is assigned to each feature. Following are the sets of cases for all defects:

3.2.1 Redundancy

In this Sub-section, we discuss various cases of defect due to redundancy in FM [21, 40, 92, 96]:

(i) Redundancy due to exclusion and group cardinality [21]**:** Figure 3.2 depicts the the cases of redundancy due to exclusion of an alternative-child feature, where parent feature *p* can be either root or full-mandatory feature.

Rule R.1 The alternative-child features *c1* and *c2* belong to the group cardinality

<1..1> with parent feature *p* where *c1* excludes *c2*. Thus, the exclusion becomes redundant.

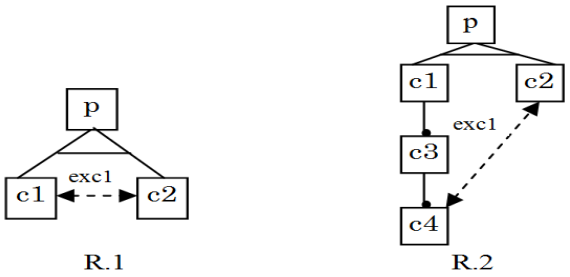


Figure 3.2: Cases of redundancy due to exclusion and group cardinality.

Rule R.2 The alternative-child features *c1* and *c2* belong to the group cardinality <1..1> with parent feature *p* and mandatory feature *c3* has a parent *c1*. The feature *c2* is excluded by mandatory feature *c4* which has a parent *c3*. It represents a multiple exclusion of an alternative-child feature *c2* which is redundant.

(ii) Redundancy due to mandatory feature and implication [21, 47, 96, 99, 102]**:** Figure 3.3 represents the cases of redundancy caused by a full-mandatory feature and implication constraint(s).

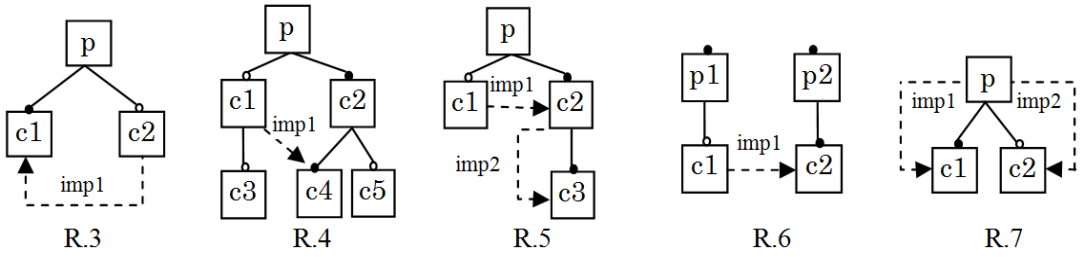


Figure 3.3: Cases of redundancy due to mandatory feature and implication.

Rule R.3 The full-mandatory feature *c1* and an optional feature *c2* have same parent feature *p* (in this case, *p* can be either root or full-mandatory feature). In any case, *c1* is always selected and therefore, implication from *c2* to *c1* is redundant.

Rule R.4 An optional child feature *c1* and mandatory child feature *c2* have same parent feature *p* (in this case, *p* can be either root or full-mandatory feature) where an optional feature *c3* has parent *c1*. Both, full-mandatory child feature *c4* and optional child feature *c5* have same parent *c2* where *c1* implies *c4*. In any respect, *c4* is always incorporated in all the configurations and therefore, implication from *c1* to *c4*

is redundant.

Rule R.5 An optional childfeature *c1* and full-mandatory childfeature *c2* have same parent feature *p* (in this case, *p* can be either root or full-mandatory feature) where *c2* is implied by *c1*.In any case, *c2* appears in all the configurations. Thus, implication (*imp1*) from *c1* to *c2* is redundant. The mandatory child feature *c3* is implied by its mandatory parent feature *c2*. This implication (*imp2*) is also redundant as both *c2* and *c3* are relative-mandatory to each other and will always emerge simultaneously in a configuration.

Rule R.6 An optional child feature *c1* has a parent full-mandatory feature *p1* and another full-mandatory child feature *c2* has a parent mandatory feature *p1*. It means *c2* is incorporated in every product and thus, *c1* implies *c2* is redundant.

Rule R.7 Both mandatory child features *c1* and *c2* have same parent feature *p* (in this case, *p* can be either root or full-mandatory feature)and these features are implied by *p*. In any case, *c1* and *c2* appear in all the configurations and thus, implications (*imp1* and *imp2*) are redundant.

(iii) Redundancy due to multiple implications [21, 47, 92, 96, 99]**:** Figure 3.4 shows the cases of redundancy due to numerous implication constraints, where parent feature *p* can be either root or full-mandatory feature.

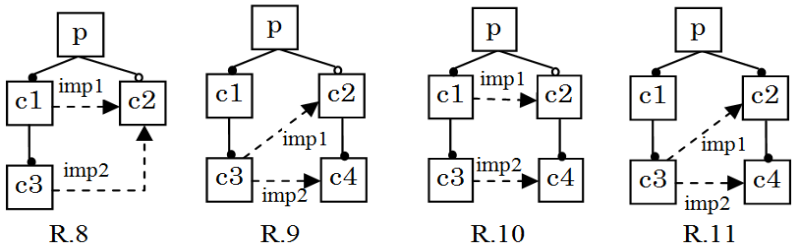


Figure 3.4: Cases of redundancy due to multiple implications.

Rule R.8 The mandatory child feature *c1* and optional child feature *c2* have same parent feature *p* where *c2* is implied by *c1*. Since *c1* is a parent of mandatory child feature *c3*, the implication from *c3* to *c2* is superfluous and thus, the implication (*imp2*) from *c3* is redundant.

Rule R.9 The mandatory child feature *c1* and optional child feature *c2* have same

parent feature *p*. The mandatory child feature *c3* has a parent *c1* where *c3* implies *c2* this means that mandatory child feature *c4* of parent *c2* is also included. Therefore, implication (*imp2*) on *c4* after *c2* is implied is redundant.

Rule R.10 The full-mandatory child feature *c1* and optional child feature *c2* have same parent feature *p* where *c1* implies *c2*. This implication constraint will be reflected in each mandatory child feature of the two features *c1* and *c2*. Simultaneously, the mandatory child feature *c3* has a parent *c1* implies another mandatory child feature *c4* which has a parent *c2* is redundant.

Rule R.11 Both the full-mandatory child features *c1* and *c2* have same parent feature *p*. The mandatory feature *c3* has a parent *c1* implies *c2* and its mandatory child feature *c4*.Both implications are redundant because *c2* and *c3* are already incorporated in each product.

(iv) Redundancy due to multiple exclusions [21, 47, 96, 99]**:** Figure 3.5 depicts the cases of redundancy due to exclusion constraints are shown, where parent feature *p* can be either root or full-mandatory feature.

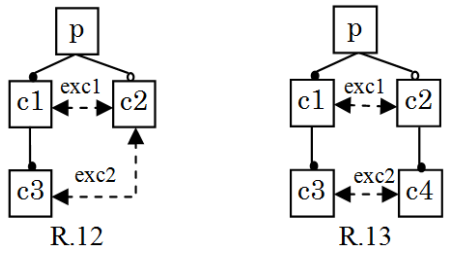


Figure 3.5: Cases of redundancy due to multiple exclusions.

Rule R.12 The mandatory child feature *c1* and optional child feature *c2* have same parent feature *p* where *c2* is excluded by *c1*. Since *c1* is a parent of mandatory child feature *c3*, thus, exclusion (*exc2*) from *c3* to *c2* is redundant.

Rule R.13 The full-mandatory child feature *c1* and optional child feature *c2* have same parent feature *p* where *c1* excludes *c2*. It means that all the mandatory child features of *c1* exclude by default all the mandatory child features of the excluded *c2*. Simultaneously, the mandatory child feature *c3* has a parent *c1* excludes another mandatory child feature *c4* which has a parent *c2* is redundant.

(v) Redundancy due to cyclic implications [21, 47]**:** Figure 3.6 represents a case of redundancy due to cyclic implication constraints.

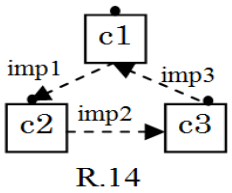


Figure 3.6: A case of redundancy due to cyclic implications.

Rule R.14 The feature *c1* implies feature *c2*, *c2* implies feature *c3*and *c3* implies *c1*. The cycle can start from any feature. The precedent implication constraint is redundant in any case, as it already incorporates the triggered feature. In this case, the selection of *c3* leads to *c3* implies *c1*and *c1* implies *c2*, therefore *c3* implies *c1*is redundant since feature *c3* is already chosen.

(vi) Redundancy due to transitive implications [21, 47, 49, 102]**:** Figure 3.7 illustrates the cases of redundancy due to transitive implication constraints.

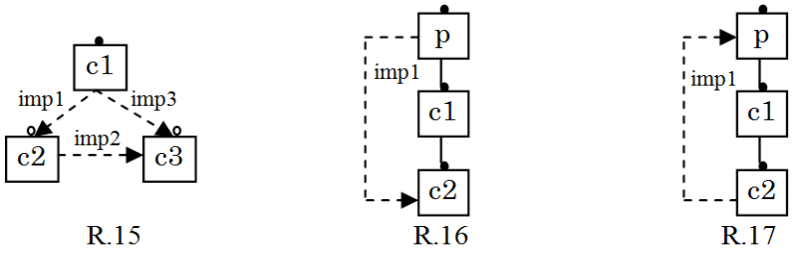


Figure 3.7: Cases of redundancy due to transitive implications.

Rule R.15 A feature *c*3 is directly implied by features *c*1 and *c*2 where *c*2 is implied by *c*1. As *c*3 is already implied by the transitive implication from *c*1 through *c*2, the direct implication (*imp3*) from *c*1 to *c*3 is redundant.

Rule R.16 The mandatory child feature *c1* has a parent feature *p* (in this case, *p* can be either root or full-mandatory feature) and mandatory child feature *c2* has a parent *c1* where *p* implies *c2*.The implication (*imp1)* from *p* to *c2* is redundant due to a transitive relationship between *p* and *c2*.

Rule R.17 The mandatory child feature *c1* has a parent feature *p* (in this case, *p* can be either root or full-mandatory feature) and mandatory child feature *c2* has a parent

c1 where *c2* implies *p*. The implication (*imp1)* is redundant due to a transitive relationship between *c2 and p*.

3.2.2 Dead Features

In this Sub-section, we discuss various cases of defects due to dead feature in FM.

(i) Dead features due to mutual exclusion between full-mandatory feature and optional feature [35, 43, 47, 49, 84, 99, 102, 113]**:** Figure 3.8 depicts the cases of dead feature in which a full-mandatory feature excludes an optional feature, where grey features represent dead features. Here, parent feature *p* can be either root or full-mandatory feature.

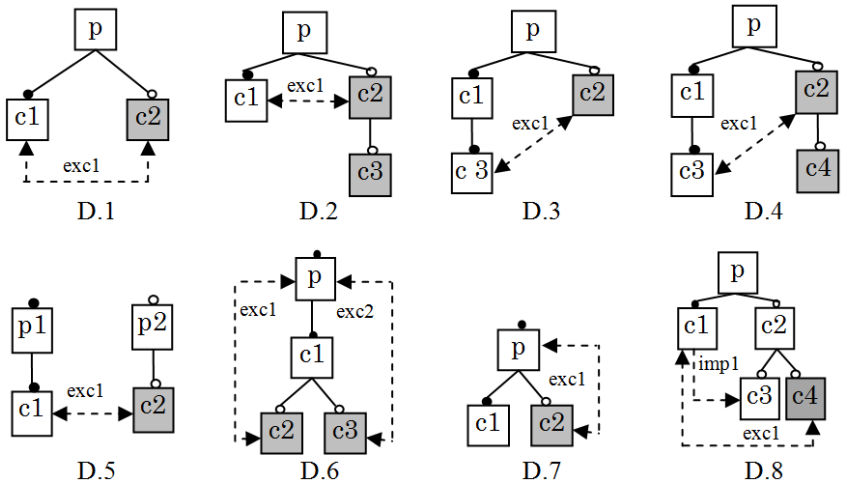


Figure 3.8: Cases of dead feature due to mutual exclusion between full-mandatory feature and optional feature.

Rule D.1 The full-mandatory child feature *c1* and optional child feature *c2* have same parent feature *p* where *c1* excludes *c2*. The feature *c1* being mandatory must be incorporated in each product. According, to the exclusion relationship, both the features *c1* and *c2* cannot be incorporated together in any product. It means *c2* is excluded from all the products and thus, *c2* becomes a dead feature.

Rule D.2 The full-mandatory child feature *c1* and optional child feature *c2* have same parent feature *p* where *c1* excludes *c2* and turns *c2* into a dead feature. An optional feature *c3* also becomes a dead feature as its parent *c2* is dead also.

Rule D.3 The mandatory child feature *c1* and optional child feature *c2* have same

parent feature *p*. The mandatory child feature *c3* has a parent *c1* where *c2* is excluded by *c1* and thus, *c2* becomes a dead feature.

Rule D.4 The mandatory child feature *c1* and optional child feature *c2* have same parent feature *p*. The mandatory child feature *c3* has a parent *c1* and optional child feature *c4* has a parent *c2*. The feature *c2* becomes a dead feature as it is excluded by *c3* and *c4* also becomes dead as its parent *c2* is dead too.

Rule D.5 The mandatory child feature *c1* which has a parent mandatory feature *p1* excludes an optional child feature *c2* that has a parent optional feature *p2* (in this case, *p2* can be either optional or mandatory feature). Thus, *c2* becomes a dead feature.

Rule D.6 The mandatory child feature *c1* has a parent feature *p*. Two optional child features *c2* and *c3* have same parent *c1* where *p* excludes both *c2* and *c3*. Thus, *c2* and *c3* become dead features.

Rule D.7 The mandatory child feature *c1* and optional child feature *c2* have same parent feature *p*. Therefore, *c2* becomes a dead feature as it is excluded by *p*.

Rule D.8 The full-mandatory child feature *c1* and optional child feature *c2* have same parent feature *p*. The optional child features *c3* and *c4* have same parent *c2* where *c1* implies *c3* and *c4* becomes a dead feature as it is excluded by *c1*.

(ii) Dead features due to implication and exclusion [41, 43, 102]**:** In Figure 3.9, the cases of dead feature caused by implication and exclusion constraints are illustrated, where grey features represent dead features. Here, parent feature *p* can be either root or full-mandatory feature.

Rule D.9 The full-mandatory child feature *c1* and optional child feature *c2* have same parent feature *p*. The feature *c2* is excluded by *c1* which turns *c2* into a dead feature. The optional feature *c3* which has a parent *c1* also becomes a dead feature as it implies dead feature *c2*.

Rule D.10 The full-mandatory child feature *c1* and optional child feature *c2* have same parent feature *p*. The feature *c2* is implied by *c1* which turns *c2* into a false-optional feature as it is incorporated in all the products in which *c1* is included. The

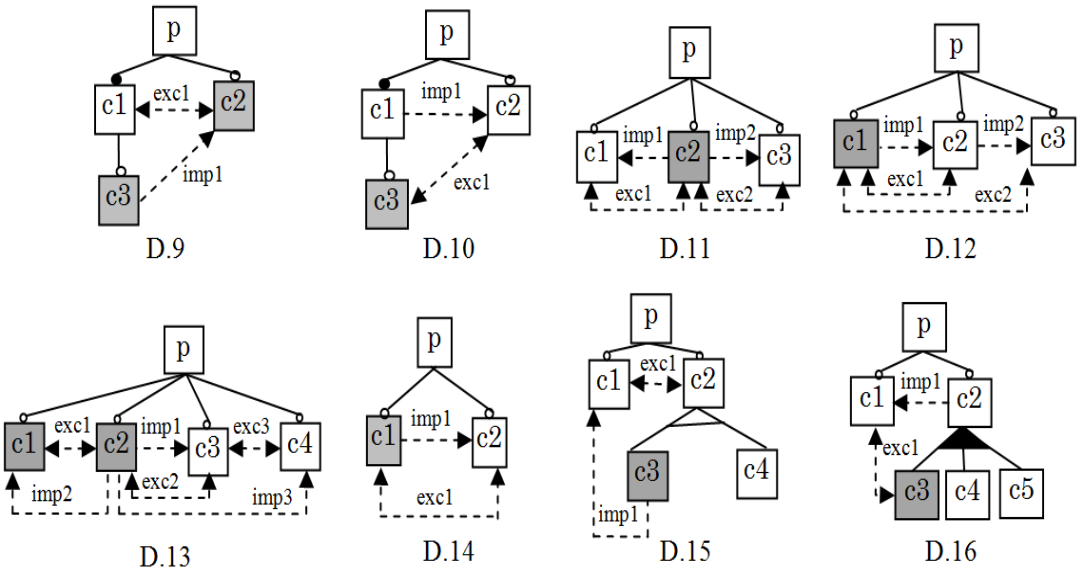


Figure 3.9: Cases of dead feature due to implication and exclusion.

optional feature *c3* which has a parent *c1* becomes a dead feature as it is excluded by false-optional feature *c2*.

Rule D.11 The optional child features *c1*, *c2* and *c3* have same parent feature *p*. The feature *c2* is dead as it implies *c1* as well as *c3* and both features *c1* and *c3* excludes *c2*.

Rule D.12 The optional child features *c1*, *c2* and *c3* have same parent feature *p*. The feature *c2* is implied by *c1* and *c2* implies *c3* where both the features *c2* and *c3* excludes *c1*. Consequently, *c1* becomes a dead feature.

Rule D.13 The optional child features *c1*, *c2*, *c3* and *c4* have same parent feature *p*. The feature *c2* is excluded by *c1*, *c2* implies *c3* and *c4* is excluded by *c3*. The feature *c2* implies *c1* and *c4* along with it is excluded by *c3* and thus, both the features *c1* and *c2* become dead.

Rule D.14 Both optional child features *c1* and *c2* have same parent feature *p*. The feature *c1* becomes dead, as it implies *c2* which further excludes *c1*.

Rule D.15 The optional child features *c1* and *c2* have same parent feature *p* where *c1* excludes *c2*. The alternative-child features *c3* and *c4* belong to the group cardinality <1..1> with parent feature *c2* where *c1* is implied by *c3*. Thus, *c3* becomes a dead feature.

Rule D.16 The optional child feature *c1* and full-mandatory child feature *c2* have same parent feature *p* where *c1* is implied by *c2*. The or-child features *c3*, *c4* and *c5* belong to the group cardinality <1..3> with parent feature *c2* where *c3* is excluded by *c1* which turns *c3* into a dead feature.

(iii) Dead features due to implication and group cardinality [38, 43, 49, 99, 107, 114]**:** Figure 3.10 describes the cases of dead feature in which an alternative-child feature(s) is implied by another optional or mandatory feature, where grey features represent dead features.

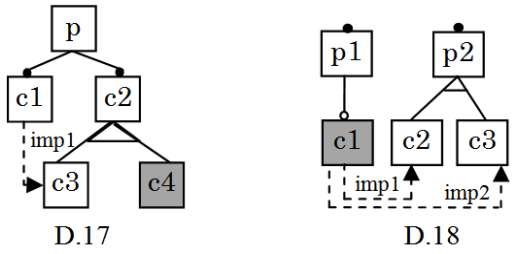


Figure 3.10: Cases of dead feature due to implication and group cardinality.

Rule D.17 The full-mandatory child feature *c1* and mandatory child feature *c2* (in this case, *c2* can be either optional or mandatory) have same parent feature *p* (in this case, *p* can be either root or full-mandatory feature). The alternative-child features *c3* and *c4* belong to the group cardinality <1..1> with parent feature *c2* where *c1* implies *c3*. Accordingly, the remaining alternative-child feature *c4* becomes a dead feature.

Rule D.18 An optional child feature *c1* has a parent feature *p1* and alternative-child features *c2* and *c3* have parent feature *p1* (in this case, *p1* and *p2* are full-mandatory features) where *c2* and *c3* are implied by *c1*. Thus, *c1* becomes a dead feature.

(iv) Dead features due to exclusion and group cardinality [38, 49, 99, 102, 107, 114]**:** In Figure 3.11, the cases of dead feature in which an or-child feature or alternative-child feature is excluded by a full-mandatory feature are illustrated, where grey features represent dead features. Here, parent feature *p* can be either root or full-mandatory feature.

Rule D.19 The full-mandatory child feature *c1* and mandatory child feature *c2* (in this case, *c2* can be either optional or mandatory) have same parent feature *p*. The alternative-child features *c3* and *c4* belong to the group cardinality <1..1> with parent

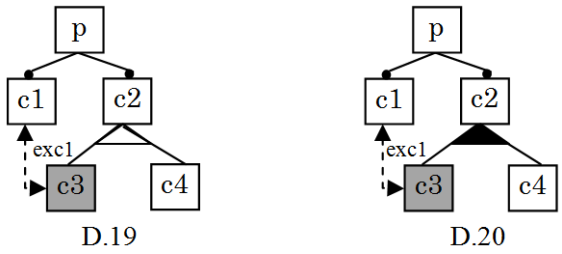


Figure 3.11: Cases of dead feature due to exclusion and group cardinality.

feature *c2* where *c1* excludes *c3*. Consequently, *c3* can never be selected and it becomes a dead feature.

Rule D.20 The full-mandatory child feature *c1* and mandatory child feature *c2* (in this case, *c2* can be either optional or mandatory) have same parent feature *p*. The or-child features *c3* and *c4* belong to the group cardinality <1..2> with parent feature *c2* where *c3* is excluded by *c1* which turns *c3* into a dead feature.

3.2.3 False-Optional Features

In this Sub-section, various cases of defects due to false-optional feature in FM are discussed.

(i) False-optional features due to implication between full-mandatory feature and optional feature [21, 30, 43, 47, 49, 102]**:** Figure 3.12 represents the cases of false-optional feature in which a full-mandatory feature implies an optional feature, where

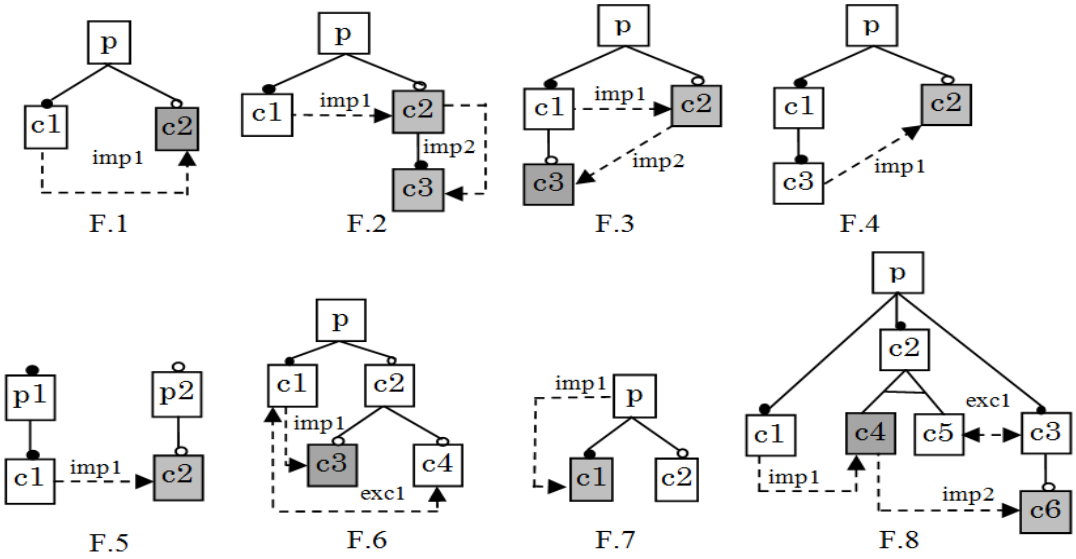


Figure 3.12: Cases of false-optional feature due to implication between full-mandatory feature and optional feature.

grey features represent false-optional features. Here, parent feature *p* can be either root or full-mandatory feature.

Rule F.1 The full-mandatory child feature *c1* and optional child feature *c2* have same parent feature *p* where *c1* implies *c2*. The feature *c2* is expected to be an optional feature, however, due to implication constraint (*imp1*) it is incorporated in all the products in which *c1* is included. Thus, *c2* becomes a false-optional feature.

Rule F.2 The full-mandatory child feature *c1* and optional child feature *c2* have same parent feature *p*. The feature *c2* turns into a false-optional feature as it is implied by *c1*. The optional child feature *c3* also becomes false-optional as it is implied by its parent feature *c2* which is already a false-optional feature.

Rule F.3 The full-mandatory child feature *c1* and optional child feature *c2* have same parent feature *p*. The feature *c2* turns into a false-optional feature as it is implied by *c1* and optional feature *c3* which has a parent *c1* also becomes a false-optional feature as it is implied by false-optional feature *c2*.

Rule F.4 The mandatory child feature *c1* and optional child feature *c2* have same parent feature *p*. The full-mandatory feature *c3* has a parent *c1* where *c2* is implied by *c3* and turns *c2* into a false-optional feature.

Rule F.5 The full-mandatory child feature *c1* which has a parent mandatory feature *p1* implies an optional child feature *c2* that has a parent optional feature *p2*. Thus, *c2* becomes a false-optional feature.

Rule F.6 The full-mandatory child feature *c1* and optional child feature *c2* have same parent feature *p*. The optional child features *c3* and *c4* have same parent *c2* where *c1* excludes *c4*. The feature *c3* becomes a false-optional feature as it is implied by *c1*.

Rule F.7 The optional child features *c1* and *c2* have same parent feature *p*. The feature *c1* becomes a false-optional feature, as it is implied by *p*.

Rule F.8 The full-mandatory child features *c1, c2* and *c3* have same parent feature *p*. The alternative-child features *c4* and *c5* with parent *c2* where *c3* excludes *c5*. The feature *c4* becomes a false-optional feature since it is implied by *c1*. An optional child feature *c6* which has a parent *c3* turns into a false-optional feature as it is

implied by *c4* which is already a false-optional feature.

(ii) False-optional features due to implication and exclusion [41, 43]**:** Figure 3.13 illustrates the cases of false-optional feature caused by implication and exclusion constraints, where grey features represent false-optional features. Here, parent feature *p* can be either root or full-mandatory feature.

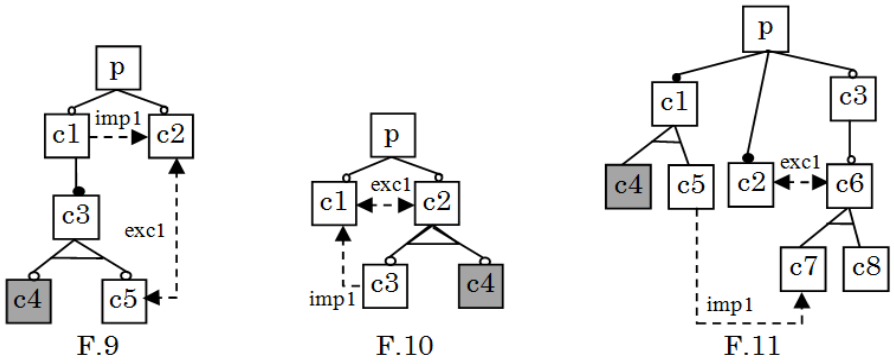


Figure 3.13: Cases of false-optional feature due to implication and exclusion.

Rule F.9 The optional child features *c1* and *c2* have same parent feature *p* where *c2* is implied by *c1*. The feature *c3* is a mandatory child feature of *c1*. The alternative-child features *c4* and *c5* belong to the group cardinality <1..1> with parent feature *c3* where *c5* is excluded by *c2*. Accordingly, the remaining alternative-child feature *c4* becomes a false-optional feature.

Rule F.10 The optional child features *c1* and *c2* have same parent feature *p* where *c1* excludes *c2*. The alternative-child features *c3* and *c4* belong to the group cardinality <1..1> with parent feature *c3* where *c1* is implied by *c3*. Thus, *c4* becomes a false-optional feature.

Rule F.11 The full-mandatory child features *c1*, *c2* and an optional child feature *c3* have same parent feature *p*. The alternative-child features *c4* and *c5* belong to the group cardinality <1..1> with parent feature *c1* where *c2* excludes an optional child feature *c6* which has a parent *c3*. The alternative-child features *c7* and *c8* belong to the group cardinality <1..1> with parent feature *c6* where *c7* is implied by *c5* and thus, it turns *c4* into a false-optional feature.

(iii) False-optional features due to implication and group cardinality [30, 47, 46, 49,

99, 102]**:** In Figure 3.14, the cases of false-optional feature in which an or-child feature or alternative-child feature is implied by a full-mandatory feature are illustrated, where grey features represent false-optional features. Here, parent feature *p* can be either root or full-mandatory feature.

Rule F.12 The full-mandatory child features *c1* and *c2* have same parent feature *p*. The alternative-child features *c3* and *c4* belong to the group cardinality <1..1> with parent feature *c2* where *c3* is implied by *c1*. Consequently, *c3* becomes a false-optional feature.

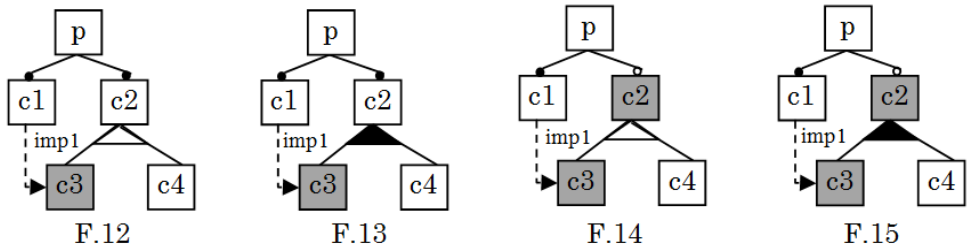


Figure 3.14: Cases of false-optional feature due to implication and group cardinality.

Rule F.13 The full-mandatory child features *c1* and *c2* have same parent feature *p*. The or-child features *c3* and *c4* belong to the group cardinality <1..2> with parent feature *c2* where *c3* is implied by *c1*. Subsequently, *c3* becomes a false-optional feature.

Rule F.14 The full-mandatory child feature *c1* and an optional child feature *c2* have same parent feature *p*. The alternative-child features *c3* and *c4* belong to the group cardinality <1..1> with parent feature *c2* where *c1* implies *c3*. Consequently, *c2* and *c3* become false-optional features.

Rule F.15 The full-mandatory child feature *c1* and optional child feature *c2* have same parent feature *p*. The or-child features *c3* and *c4* belong to the group cardinality <1..2> with parent feature *c2* where *c1* implies *c3*. Consequently, *c2* and *c3* become false-optional features.

(iv) False-optional features due to exclusion and group cardinality [46, 47, 49, 99]**:** Figure 3.15 represents the cases of false-optional feature in which an or-child feature or alternative-child feature is excluded by a full-mandatory feature, where grey

features represent false-optional features. Here, parent feature *p* can be either root or full-mandatory feature.

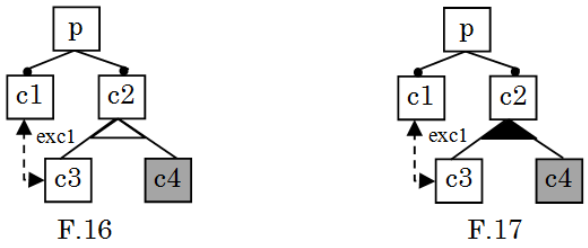


Figure 3.15: Cases of false-optional feature due to exclusion and group cardinality.

Rule F.16 The full-mandatory child features *c1* and *c2* have same parent feature *p*. The alternative-child features *c3* and *c4* belong to the group cardinality <1..1> with parent feature *c2* where *c1* excludes *c3*. Consequently, *c4* becomes a false-optional feature.

Rule F.17 The full-mandatory child features *c1* and *c2* have same parent feature *p*. The or-child features *c3* and *c4* belong to the group cardinality <1..2> with parent feature *c2* where *c3* is excluded by *c1*. Subsequently, *c4* becomes a false-optional feature.

3.2.4 Inconsistency

In this Sub-section, we discuss various cases of defects due to inconsistency in FM [21, 41, 43, 45, 46, 49, 99, 102, 107, 114]**:**

(i) Inconsistency due to implication and exclusion simultaneously: Figure 3.16 depicts a case of inconsistency in which a mandatory feature excludes and implies another mandatory feature at the same time.

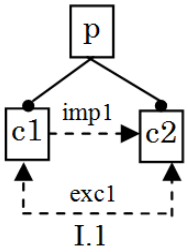


Figure 3.16: A case of inconsistency due to implication and exclusion simultaneously.

Rule I.1 Both the mandatory child features *c1* and *c2* have same parent feature *p* (in

this case, *p* can be either root or full-mandatory feature) where *c1* implies *c2* and *c2* excludes *c1*. Thus, it is an inconsistency as both *c1* and *c2* can never be selected simultaneously for the configuration of a valid product.

(ii) Inconsistency due to mutual exclusion between mandatory features: In Figure 3.17, the cases of inconsistency in which a mandatory feature excludes another mandatory feature are illustrated, where parent feature *p* can be either root or full-mandatory feature.

Rule I.2 Both the mandatory child features *c1* and *c2* have same parent feature *p* where *c1* excludes *c2*. Thus, it is an inconsistency as both *c1* and *c2* can never be selected together for the configuration of a valid product.

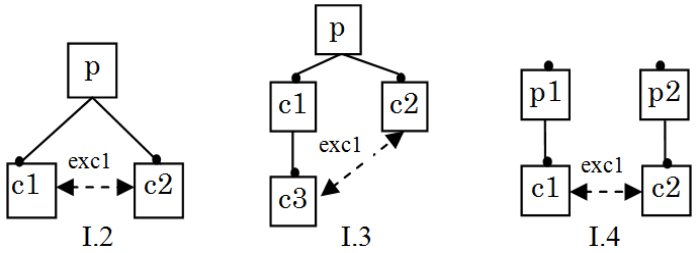


Figure 3.17: Cases of inconsistency due to mutual exclusion between mandatory features.

Rule I.3 Both the mandatory child features *c1* and *c2* have same parent feature *p*. Thus, it is an inconsistency as the mandatory child feature *c3* which has a parent c1 is excluded by *c2*.

Rule I.4 A mandatory child feature *c1* has a parent full-mandatory feature *p1* excludes another mandatory child feature *c2* that has a parent full-mandatory feature *p2*. Consequently, this case turns into an inconsistency.

(iii) Inconsistency due to implication and exclusion: Figure 3.18 illustrates the cases of inconsistency caused by implication and exclusion constraints, where *p* can be either root or full-mandatory feature.

Rule I.5 Both the mandatory child features *c1* and *c2* have same parent feature *p* where c*1* excludes *c2*. The mandatory child feature *c3* has a parent *c1* where *c3* implies *c2* and thus, it leads to an inconsistency.

Rule I.6 Both the mandatory child features *c1*and *c2* have same parent feature *p*

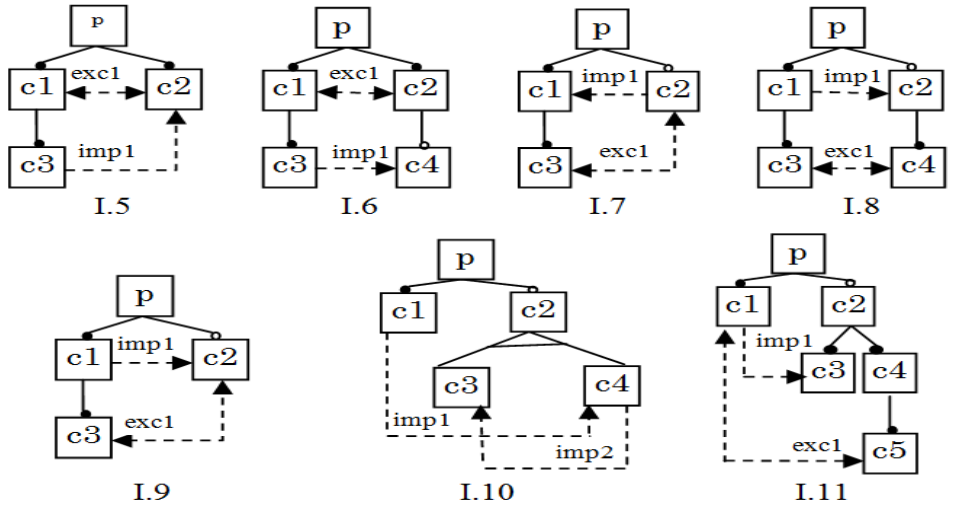


Figure 3.18: Cases of inconsistency due to implication and exclusion.

where *c1* excludes *c2*. The feature *c1* has a mandatory child feature *c3* which implies an optional child feature *c4* whose parent is *c2*. Consequently, this case turns into an inconsistency.

Rule I.7 An optional child feature *c1* and mandatory child feature *c2* have same parent feature *p* where *c1* is implied by *c2*. The feature *c1* has a mandatory child feature *c3* which excludes c2 and thus, it becomes an inconsistency.

Rule I.8 The mandatory child feature *c1* and optional child feature c*2* have same parent feature *p* where *c2* is implied by *c1*.The feature *c1* has a mandatory child feature *c3* which excludes the other mandatory child feature *c4* whose parent is *c2*. Consequently, this case turns into an inconsistency.

Rule I.9 The mandatory child feature *c1* and optional child feature *c2* have same parent feature *p* where *c2* is implied by *c1*. The feature *c1* has a mandatory child feature *c3* which excludes *c2* and thus, it becomes an inconsistency.

Rule I.10 The mandatory child feature *c1* and optional child feature *c2* have same parent feature *p*. Both the mandatory child features *c3* and *c4* have same parent *c2* where *c3* is implied by *c1*. The mandatory child feature *c5* whose parent is *c4* excludes c1 and consequently, this case turns into an inconsistency.

Rule I.11 The full-mandatory child feature *c1* and optional child feature *c2* have same parent feature *p*. Both the alternative-child features *c3* and *c4* belong to the

group cardinality <1..1> with parent feature *c2* where *c4* is implied by *c1*and *c3* is implied by *c4*. Thus, it leads to an inconsistency.

(iv) Inconsistency due to implication between alternative-child features: Figure 3.19 depicts a case of inconsistency caused by an implication between two alternative-child features grouped in a cardinality.

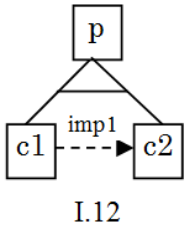


Figure 3.19: A case of inconsistency due to implication between alternative-child features.

Rule I.12 Both the alternative-child features *c1* and *c2* belong to the group cardinality <1..1> with parent feature *p* (in this case, *p* can be either root or full-mandatory feature) where *c1* implies *c2*. The implication between the child features exceeds the upper limit of the group cardinality and does not even allow incorporating one sub-feature. Thus, it becomes an inconsistency.

3.2.5 Wrong Cardinality

In this Sub-section, we discuss various cases of defects due to wrong cardinality in FM [30, 48, 84]. The cardinalities are explicitly shown in Figures 3.20, 3.21 and 3.22 to illustrate the cases of defects due to wrong cardinality, where parent feature *p* can be either root or full-mandatory feature.

(i) Not well-defined boundaries in cardinality: It is represented by an interval <*min..max*>. The *min* value specifies the minimum count of features allowed to be included in a cardinality relationship and it should be inferior to the number of features assembled in a group cardinality. The *max* value specifies the maximum count of features allowed to be included in a cardinality relationship and it should be inferior or equal to the number of features assembled in a group cardinality. It includes following cases:

Incorrect values of min and max [21, 48]**.** In Figure 3.20, *W.1* illustrates a case of wrong cardinality in which both *min* and *max* are wrongly defined.

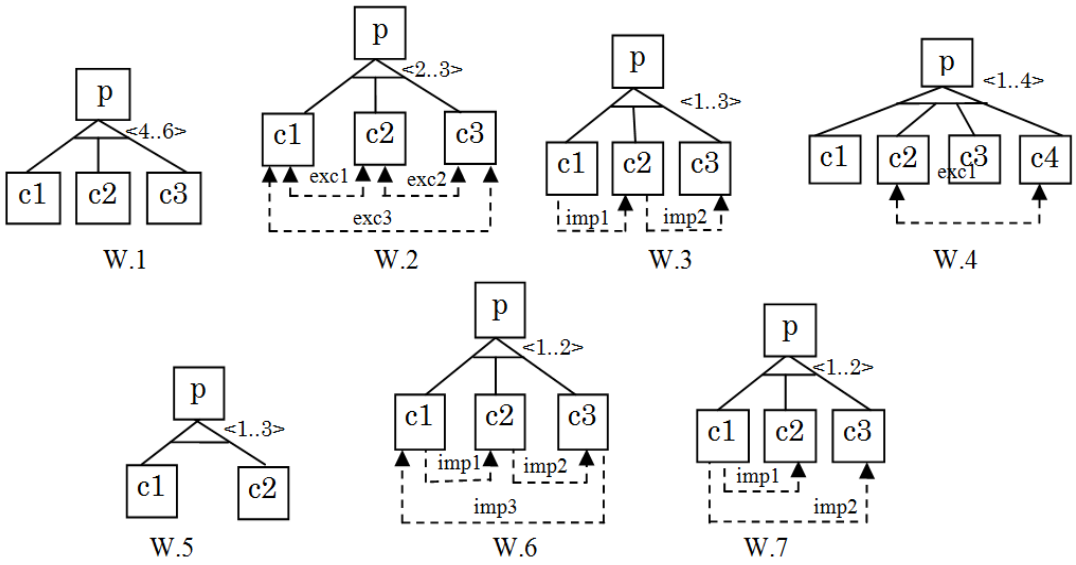


Figure 3.20: Cases of wrong cardinality due to not well-defined boundaries in cardinality.

Rule W.1 The child features *c1*, *c2* and *c3* belong to the group cardinality <4..6> with parent feature *p* where the number of child features grouped in the group cardinality is 3. Thus, it leads to the occurrence of wrong cardinality as 4 (*min*) is not inferior to 3 and 6 (*max)* is not inferior or equal to 3.

Incorrect value of min [30]. In Figure 3.20, *W.2* and *W.3* represent the cases of wrong cardinality where *min* is wrongly defined.

Rule W.2 The child features *c1*, *c2* and *c3* belong to the group cardinality <2..3> with parent feature *p* where *c1* excludes *c2*, *c3* is excluded by *c2*, *c1* is excluded by *c3* and min=2. Consequently, this case turns into a wrong cardinality as the minimum cardinality cannot be attained due to exclusion constraints.

Rule W.3 The child features *c1*, *c2* and *c3* belong to the group cardinality <1..3> with parent feature *p*. The selection of *c3* is correct, but the selection of *c1* or *c2* is incorrect as *c2* is implied by *c1* and *c3* is implied by *c2*. Thus, it causes a wrong cardinality as the inclusion of *c1* is followed by inclusion of *c2* and inclusion of *c2* is followed by the inclusion of *c3*.

Incorrect value of max [30, 36, 46, 115, 116]**:** In Figure 3.20, *W.4*, *W.5*, *W.6* and *W.7* illustrate the cases of wrong cardinality where *max* is incorrectly defined.

Rule W.4 The child features *c1*, *c2*, *c3* and *c4* belong to the group cardinality <1..4>

with parent feature *p* where min=1 and *max*=4. Thus, it leads to the occurrence of a wrong cardinality as the exclusion between *c2* and *c3* does not allow the selection of four child features.

Rule W.5 The child features *c1* and *c2* belong to the group cardinality <1..3> with parent feature *p* where *min*=1, *max*=3 and the number of child features grouped in the group cardinality is 2. Subsequently, this case turns into a wrong cardinality as the value of *max*(3) cardinality is incorrect due to the presence of only two features *c1* and *c2* in the group cardinality.

Rule W.6 The child features *c1*, *c2* and *c3* belong to the group cardinality <1..2> with parent feature *p* where *min*=1 and *max*=2. Thus, it causes a wrong cardinality as *c2* is implied by *c1*, *c3* is implied by *c2*, *c1* is implied by *c3* and due to these implications the maximum cardinality 2 (*max*) cannot be attained.

Rule W.7 The child features *c1*, *c2* and *c3* belong to the group cardinality <1..2> with parent feature *p* where *min*=1 and *max*=2. The selection of *c2* and *c3* is correct, however the selection of *c1* implies *c2* and *c1* implies *c3* are incorrect. Consequently, this case turns into a wrong cardinality as maximum cardinality 2 (*max*) cannot be attained due to these implication constraints.

(ii) Incorrect domain of cardinalities: In Figure 3.21, *W.8* depicts a case of wrong cardinality due to incorrect domain of cardinalities. In this case <*x*..*z*> is the cardinality where *x* is the min value which should be an integer and *z* is the max value which should be either an integer or an indefinite value represented by the symbol *. The value *x* must be inferior to the value *z*.

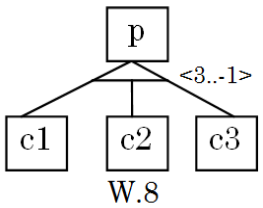


Figure 3.21: A case of wrong cardinality due to an incorrect domain of cardinalities.

Rule W.8 The child features *c1*, *c2* and *c3* belong to the group cardinality <3..-1> with parent feature *p* where min=3 and *max*=-1. Thus, it leads to the occurrence of a

wrong cardinality as *min* value 3 and *max* value -1 of the group cardinality are incorrect.

(iii) Incorrect number of chosen features from a group cardinality [21, 48, 84]**:** In Figure 3.22, *W.9* represents a case of wrong cardinality where the count of incorporated features from a group cardinality must be superior to *min* value and inferior to *max* value.

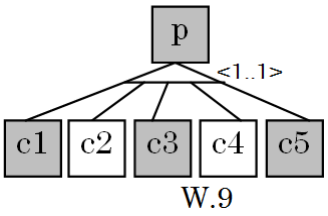


Figure 3.22: A case of wrong cardinality due to an incorrect number of chosen features from a group cardinality.

Rule W.9 The child features *c1*, *c2*, *c3*, *c4* and *c5* belong to the group cardinality <1..1> with parent feature *p* where *min*=1 and *max*=1. Thus, it causes a wrong cardinality as the number of selected child features is 2 which is superior to the value of maximum cardinality 1 (*max)*. Therefore, grey features are not related to the FM.

3.3 Running Examples for Explaining Defects

Defects are explained using the running example of laptop system FM discussed in Sub-section 1.1.2 and an example of void model [82]. In this Section, defects are illustrated using Figures 3.23 and 3.24.

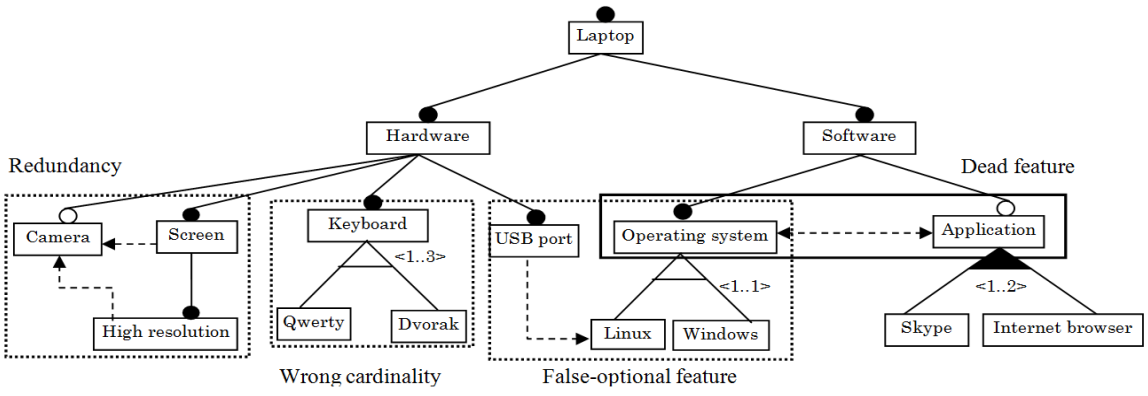


Figure 3.23: Feature model of laptop system with redundancy, dead feature, false-optional feature and wrong cardinality.

(i) Illustration of redundancy using running example: An optional feature *Camera* is implied by multiple mandatory features *Screen* and *High resolution*where *Screen* is a parent of *High resolution*. Therefore, the implication from *High resolution* is redundant.

(ii) Illustration of dead feature using running example: A full-mandatory feature *Operating system* excludes an optional feature *Application*. Consequently, both features can never be included simultaneously in any valid product. However, *Operating system* being mandatory must be incorporated in each product. It means *Application* is excluded from all the products and thus, *Application* becomes a dead feature.

(iii) Illustration of false-optional feature using running example: The alternative-child features *Linux* and *Windows* belong to the group cardinality <1..1> with parent feature *Operating system* where *Linux* is implied by a full-mandatory feature *USB port*. Consequently, *Linux* becomes a false-optional feature.

(iv) Illustration of wrong cardinality using running example: The child features *Qwerty* and *Dvorak* belong to the group cardinality <1..3> with parent feature *Keyboard.* It causes a wrong cardinality as the value of maximum cardinality is 3(*max)*, which is incorrect since the number of features grouped in the group cardinality is 2.

(v) Illustration of inconsistency using running example: Figure 3.24 shows that the features *texteditor*, *bash* and *gui* cannot be selected due to the exclusions between them, whereas feature *games* will never get selected because of implication with

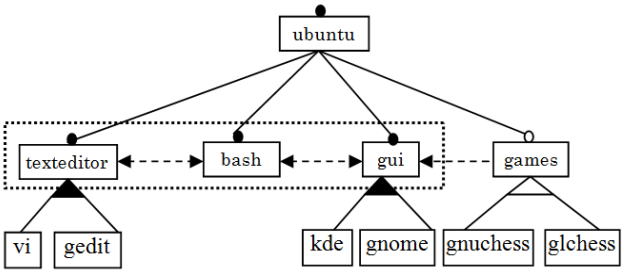


Figure 3.24: Example of void model.

feature *gui*. Therefore, this FM is a void model due to the exclusions between *texteditor* and *bash*, *bash* and *gui* as well as implication from *games* to *gui*.

3.4 Summary

This chapter presented an overview of the three stages of proposed framework in this thesis. FM defects due to redundancy, anomaly and inconsistency are classified in the form of various cases. Additionally, various cases of defects due to wrong cardinality are also illustrated. The FM of laptop system is used as a running example to explain defects due to redundancy, dead feature, false-optional feature and wrong cardinality. Further, an example of void model is used as a running example to explain void model.

CHAPTER 4: Implementing the Proposed Framework

This focus of this chapter is on the development and implementation of the framework for all defects using different real-world FMs from SPLOT repository to accomplish objective 3. Section 4.1 describes a methodology followed by the proposed approach. Section 4.2 explains and implements the two-step model transformation of FM to FMO using a running example of FM from SPLOT repository to achieve objective 5. The FOL-based rules that are developed and applied to deal with FM defects in SPL are provided in Section 4.3. The generated results include identified defects with their causes and corrections in a user-friendly natural language for resolving defects contributing to objective 6. Finally, Section 4.4 summarizes the chapter.

4.1 Introduction

The proposed framework improves SPL by analyzing defects in FMs using an ontological rule-based approach. The methodology followed by the proposed approach is described below:

(i) Analysis of FM: The input FM (in SXFM format is available in online SPLOT repository) is automatically read and analyzed using FeatureIDE tool through importing its model from SPLOT (SXFM format).

(ii) Injecting features and cross-tree constraints: Additional features and cross-tree constraints are introduced in the input model by enabling editing in FeatureIDE's model editor to cause defects.

(iii) Transformation: The modified input model (i.e. in XML format) generated from FeatureIDE is transformed to FOL predicate-based FMO using an XML parser. In the case of wrong cardinality, cardinalities are introduced explicitly in the generated FMO to cause defects as FeatureIDE doesn't support FMs with wrong cardinality.

(iv) Development and applicability of FOL-based rules: To identify defects along with their causes and corrections, FOL-based rules are developed and applied to the generated FMO.

(v) Results: The results obtained after applying all rules consist of identified defects

along with their causes and provides corrections. These results enable PL developers to eliminate cross-tree constraints related to the source of defects for resolving defects.

4.2 Transformation

Prerequisites: The root feature in FM is mandatory to be incorporated in every product. A parent feature can have more than one child features. Each feature has a unique name in FM. The cross-tree constraint relationships have also assigned unique names for better understanding of our approach to deal with defects in FM.

Transformation Process: The construction of FMO includes two-step model transformation. The transformation process is applied to the input FM in order to identify defects by means of FOL-based rules. The transformation process has been explained using the running example of *E-commerce system1* FM available in SPLOT repository (as shown in Figure 4.1). This running example is further used to illustrate the implementation of our approach to handle defects due to redundancy as shown in Sub-section 4.3.1. Following is the two-step model transformation used to construct FMO:

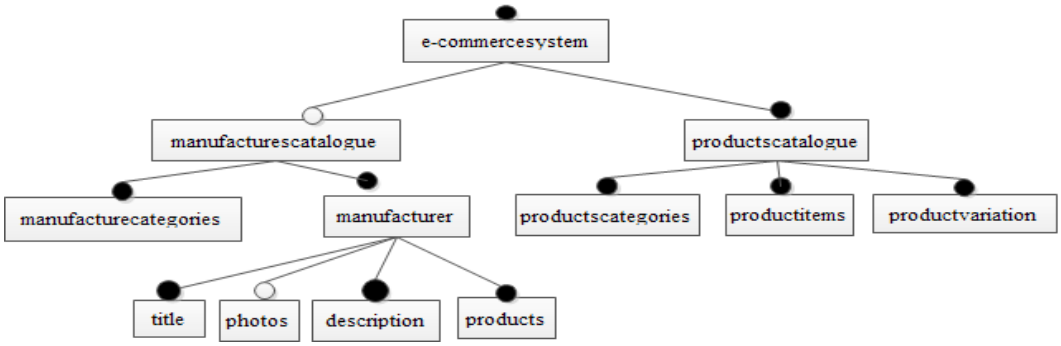


Figure 4.1: E-commerce system1 feature model from SPLOT repository.

4.2.1 SPLOT Format to the FeatureIDE Format

The *E-commerce system1* FM available in FM repository of SPLOT is automatically read by FeatureIDE tool through importing its model in SXFM format from SPLOT repository. The graphical representation of the corresponding model can be obtained by means of FeatureIDE's visualization functionalities. FeatureIDE further enables editing in its model editor. In order to illustrate our approach, additional features (i.e. *a1*, *a2*, *a3*,

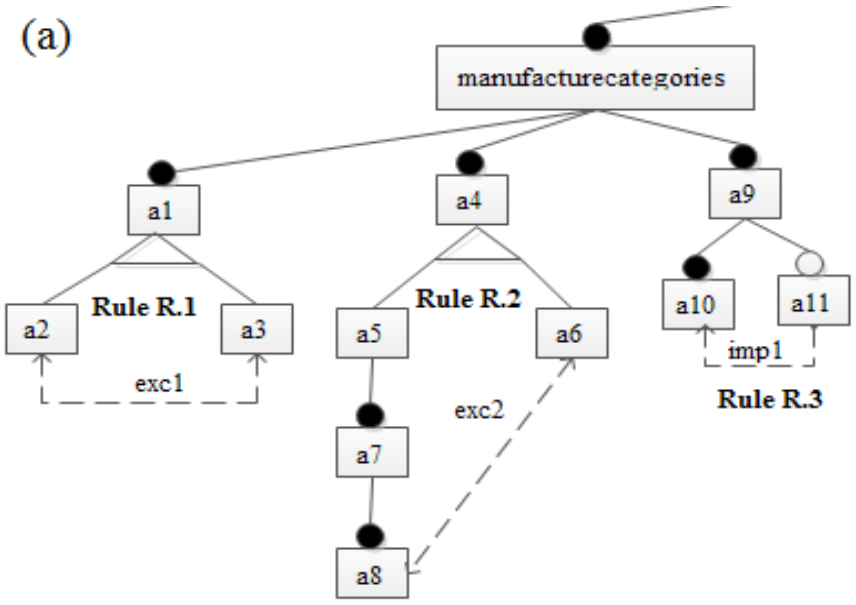
a4, *a5*, *a6*, *a7*, *a8*, *a9*, *a10*, *a11*, *a12*, *a13*, *a14*, *a15*, *a16*, *a17*, *a18*, *a19*, *a20*, *a21*, *a22*, *a23*, *a24*, *a25*, *a26*, *a27*, *a28*, *a29*, *a30*, *a31*, *a32*, *a33*, *a34*, *a35*, *a36*, *a37*, *a38*, *a39*, *a40*, *a41*, *a42*, *a43*, *a44*, *a45*, *a46*, *a47*, *a48*, *a49*, *a50*, *a51*, *a52*, *a53*, *a54*, *a55*, *a56*, *a57*, *a58*, *a59*, *a60*, *a61*, *a62*, *a63*, *a64*, *a65*, *a66*, *a67*, *a68*) and cross-tree constraints (i.e. *imp1*, *imp2*, *imp3*, *imp4*, *imp5*, *imp6*, *imp7*, *imp8*, *imp9*, *imp10*, *imp11*, *imp12*, *imp13*, *imp14*, *imp15*, *imp16*, *imp17*, *imp18*, *imp19*, *imp20*, *imp21*, *imp22*, *imp23*, *exc1*, *exc2*, *exc3*, *exc4*, *exc5*, *exc6)* are intentionally injected in the input *E-commerce system1* FM file to cause defects due to redundancy as shown in Figure 4.2.

For better understanding of the *E-commerce system1* FM generated in FeatureIDE, relationships in the graphically represented model shown in Figure 4.2 are illustrated as follows:

(i) Implication relationship is represented using a dashed arrow that begins from the source feature and ends towards the target feature and each implication relationship has been assigned a unique name such as *imp1*

(ii) Exclusion relationship is represented using a double headed dashed arrow and each exclusion relationship has been assigned a unique name such as *exc1*

(iii) The child features related in the group cardinality are basically optional features [42, 54].



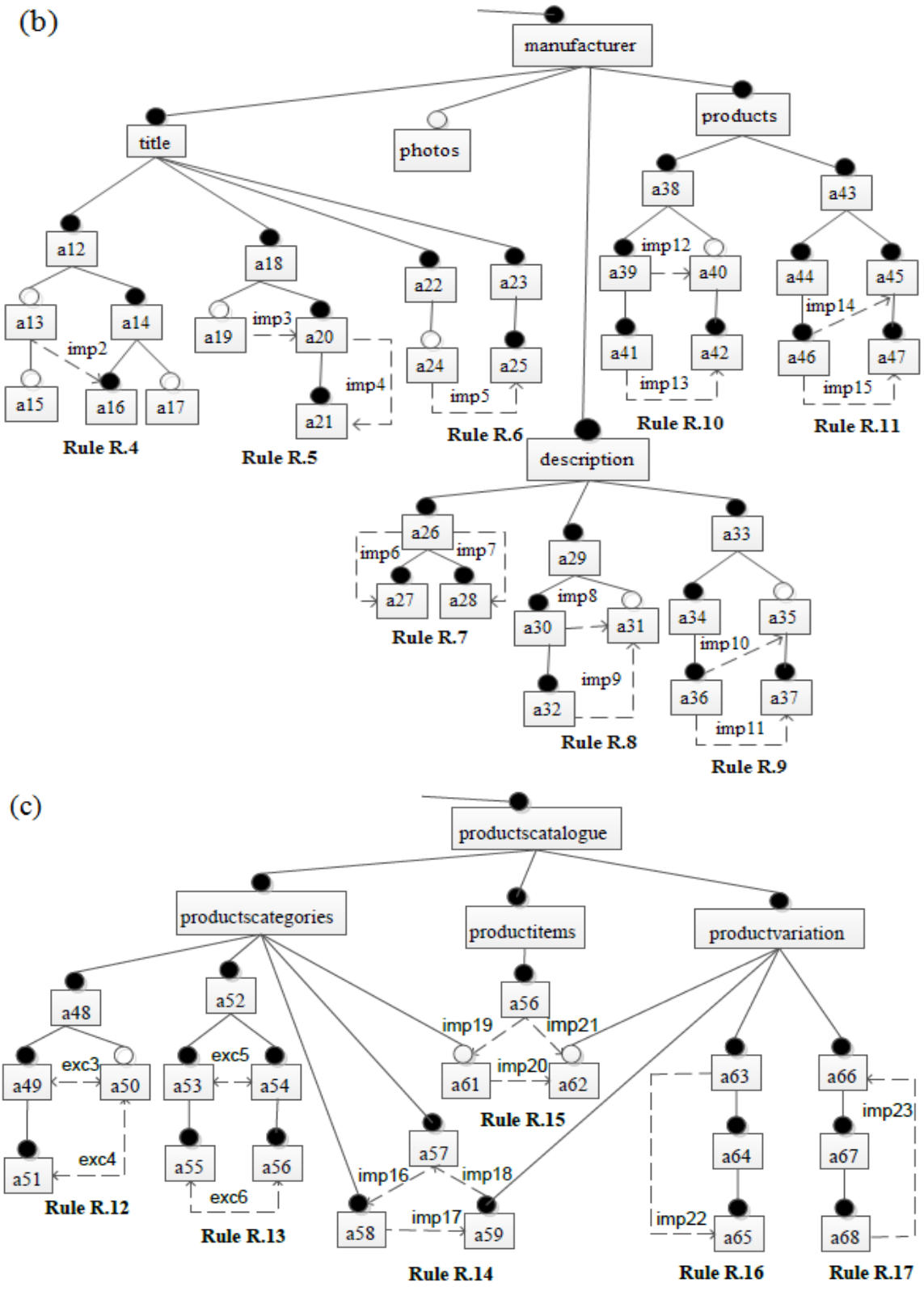
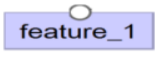
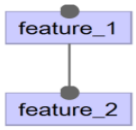
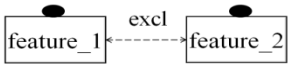
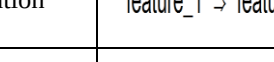
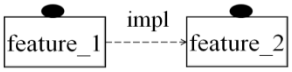
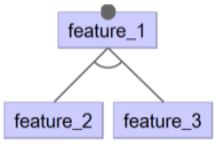
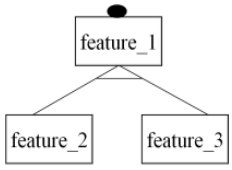
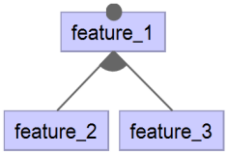
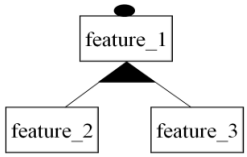


Figure 4.2: E-commerce system1 feature model representing defects due to redundancy (a) R.1-R.3, (b) R.4-R.11, and (c) R.12-R.17.

4.2.2 FeatureIDE Format to FMO

This step receives modified input *E-commerce system1* FM file from FeatureIDE and transforms it into FMO. An XML parser (see Appendix A) is developed that transforms an XML file of *E-commerce system1* FM (given in Appendix B) from FeatureIDE to FOL predicate-based FMO as shown in Figure 4.3. The generated ontology corresponds to the features and their relationships represented in the *E-commerce system1* FM. The XML parser translates input XML file to an ontology based on five types of predicates which are defined in Prolog. The predicates *feature*, *parent*, *exclusion*,*implication* and *cardinality* are represented using "*f* ", "*p*", "*e*", "*i*" and"*c*" respectively. The predicates represent classes using binary predicates and properties using ternary predicates in FMO. The transforming patterns followed to acquire predicate-based ontology from FM are given in Table 4.1. The predicates are explained using Figure 4.3:

Table 4.1: Transforming patterns from feature model into predicate-based ontology.

Names	FeatureIDE representation	Feature model representation	Predicate-based ontology representation
Optional			f (feature_1, o)
Mandatory			f (feature_1, m)
Parent			p(feature_2, feature_1)
Exclusion			e(feature_1, feature_2, exc1)
Implication			i(feature_1, feature_2, imp1)
Group cardinality (Alternative relationship)			c(feature_1, [feature_2, feature_3], [1,1])
Group cardinality (Or- relationship)			c(feature_1, [feature_2, feature_3], [1,2])

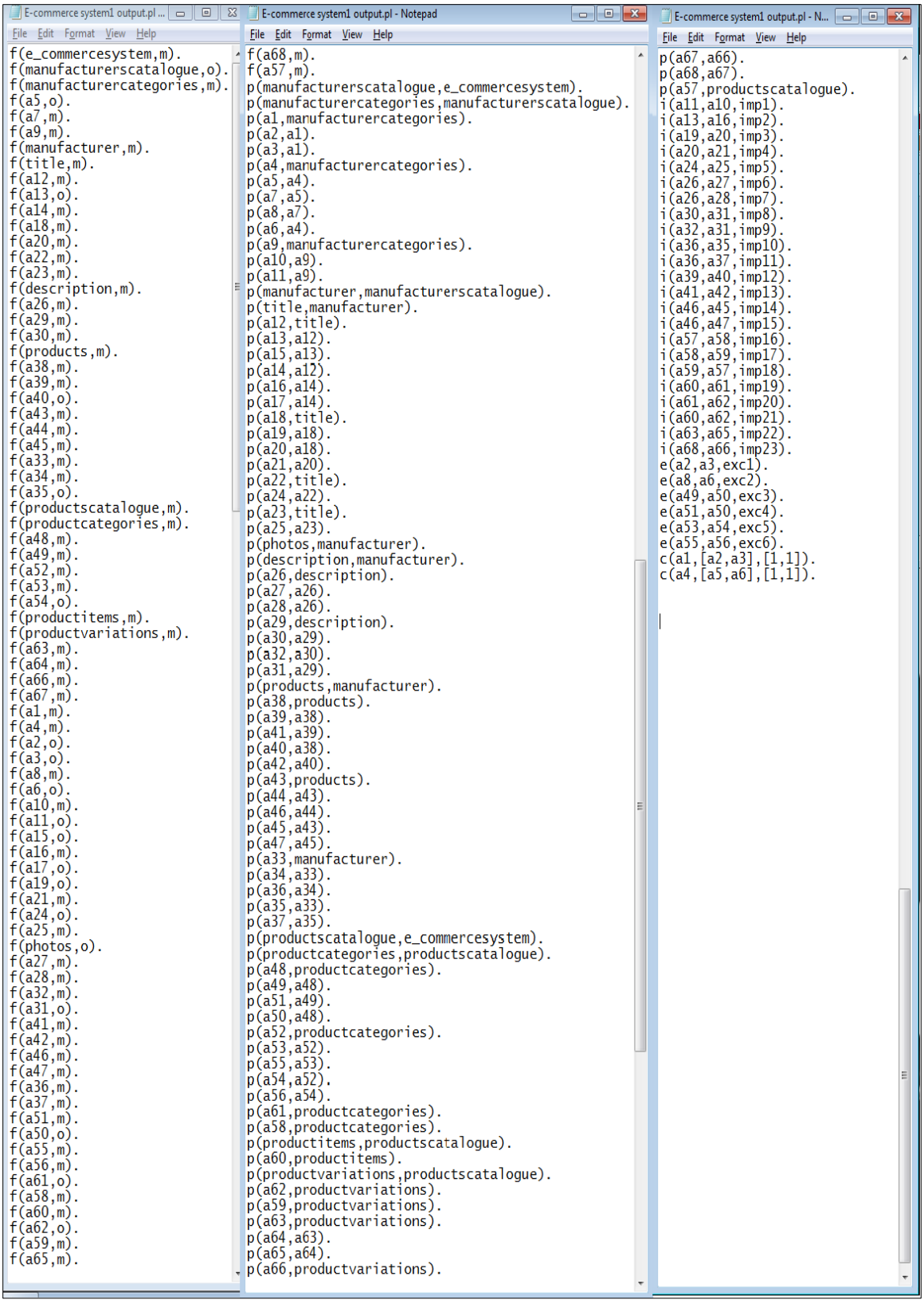


Figure 4.3: Feature model ontology of E-commerce system1 feature model.

***f(feature_1, o)*:** It incorporates *feature_1* that indicates a feature and *o* represents that *feature_1* is an optional feature, e.g. f(photos,o).

***f(feature_1, m)*:** It includes *feature_1* which demonstrates a feature and *m* represents that *feature_1* is a mandatory feature, e.g. f(manufacturer,m).

***p(feature_1, feature_2)*:** It involves child feature which is represented by *feature_1* and *feature_2* signifies its parent feature. It shows that *feature_1* has parent *feature_2* e.g. p(photos,manufacturer).

***e(feature_1, feature_2, exc1)*:** It comprises of *feature_1* and *feature_2 which* correspondingly indicate the names of source and target features intervening in the relationship respectively, where *exc1* indicates an exclusion relationship. It represents *feature_1* excludes *feature_2*, e.g. e(a2,a3,exc1).

***i(feature_1, feature_2, imp1)*:** It includes *feature_1* and *feature_2* which indicate the names of source and target features intervening in the relationship respectively, where *imp1* describes an implication relationship. It represents *feature_1* implies *feature_2*, e.g. i(a11,a10,imp1).

***c(feature_1, [feature_2, feature_3], [Min, Max])*:** It indicates that both child features *feature_2* and *feature_3* having same parent *feature_1* belong to the group cardinality <min..max>.It determines the minimum (*Min*) and the maximum (*Max*) number of child features which are allowed to be specified in a cardinality relationship. In this case, child features can be more than two. For example, the alternative relationship is represented using c(a1,[a2,a3],[1,1]) which means that only one child feature can be selected from a set of features (i.e. a2 and a3) with group cardinality <1..1>. An or-relationship for two child features c1 and c2 having same parent p is equivalent to a group cardinality <1...2> where 2 is the maximum count of features and it is represented using c(p,[c1,c2],[1,2]).

4.3 Rules for Identification of Defects, their Causes and Providing Corrections

This Section describes the rules to deal with FM defects in SPL. It is based on cross-tree constraints, i.e. implication and exclusion relationships. The predicate-based FMO is a

group of Prolog predicates that represents the existing facts. A set of FOL rules is developed which represents specific cases of misuse among the relationships in a FM that caused defects. These rules are applied as FOL queries to the generated FMO using SWI-Prolog. The results include identified defects along with their causes and provide corrections. The generated results assist PL developers to eliminate cross-tree constraints involved in the source of defects to resolve defects.

Each rule is a source for originating the identified defect. These rules can be implemented independently as well as simultaneously in Prolog. For all rules we, (i) present the existing facts, (ii) state the FOL rules, (iii) present the corresponding results after implementing each rule, which includes the identified defect with its causes and corrections, and (iv) present the implementation of all rules to the running example. Following Sub-sections illustrate the rules to deal with FM defects in SPL:

4.3.1 Redundancy

Sub-section 3.2.1 includes a detailed discussion on each case of defect arising due to redundancy. Following illustrates each case which includes facts in the form of FMO, rule and result.

Rule R.1

Facts: f(p,m). f(c1,o). f(c2,o). c(p,[c1,c2],[1,1]). e(c1,c2,exc1).

rule1:- f(P,m), f(C1,o), f(C2,o), c(P,[C1,C2],[1,1]), e(C1,C2,E1), write ('\nDefect: Redundancy1\nCause: exclusion between alternative-child features '), write(C1), write(' and '), write(C2), write(' is redundant\nCorrection: eliminate '), write (E1).

Result: Figure 4.4 represents the result produced after applying this rule to FMO.

```
Defect: Redundancy1
Cause: exclusion between alternative-child features c1 and c2 is redundant
Correction: eliminate exc1
```

Figure 4.4: Result of rule R.1.

Rule R.2

Facts: f(p,m). f(c1,o). f(c2,o). f(c3,m). f(c4,m). p(c3,c1). p(c4,c3). c(p,[c1,c2], [1,1]).

e(c4,c2,exc1).

rule2:- f(P,m), f(C1,o), f(C2,o), f(C3,m), f(C4,m), p(C3,C1), p(C4,C3), c(P,[C1,C2],[1,1]), e(C4,C2,E1), write('\nDefect: Redundancy2\nCause: multiple exclusion of an alternative-child feature '), write(C2), write('\nCorrection: eliminate '), write(E1).

Result: The generated result after implementing this rule to FMO is shown in Figure 4.5.

```
Defect: Redundancy2
Cause: multiple exclusion of an alternative-child feature c2
Correction: eliminate exc1
```

Figure 4.5: Result of rule R.2.

Rule R.3

Facts: f(p,m). f(c1,m). f(c2,o). p(c1,p). p(c2,p). i(c2,c1,imp1).

rule3:- f(P,m), f(C1,m), f(C2,o), p(C1,P), p(C2,P), i(C2,C1,I1), write('\nDefect: Redundancy3\nCause: optional feature '), write(C2), write(' implies the full-mandatory feature '), write(C1), write(' is redundant\nCorrection: eliminate '), write(I1).

Result: Figure 4.6 shows the result obtained after applying this rule to FMO.

```
Defect: Redundancy3
Cause: optional feature c2 implies the full-mandatory feature c1 is redundant
Correction: eliminate imp1
```

Figure 4.6: Result of rule R.3.

Rule R.4

Facts: f(p,m). f(c1,o). f(c2,m). f(c3,o). f(c4,m). f(c5,o). p(c1,p). p(c2,p). p(c3,c1). p(c4,c2). p(c5,c2). i(c1,c4,imp1).

rule4:- f(P,m), f(C1,o), f(C2,m), f(C3,o), f(C4,m), f(C5,o), p(C1,P), p(C2,P), p(C3,C1), p(C4,C2), p(C5,C2), i(C1,C4,I1), write('\nDefect: Redundancy4\nCause: optional feature '), write(C1), write(' implies the full-mandatory feature '), write(C4), write(' is redundant \n Correction: eliminate '), write(I1).

Result: The produced result after applying this rule to FMO is shown in Figure 4.7.

```
Defect: Redundancy4
Cause: optional feature c1 implies the full-mandatory feature c4 is redundant
Correction: eliminate imp1
```

Figure 4.7: Result of rule R.4.

Rule R.5

Facts: f(p,m). f(c1,o). f(c2,m). f(c3,m). p(c1,p). p(c2,p). p(c3,c2). i(c1,c2,imp1).

i(c2,c3,imp2).

rule5:- f(P,m1), f(C1,o1), f(C2,m2),f(C3,m3), p(C1,P), p(C2,P), p(C3,C2), i(C1,C2,I1),

i(C2,C3,I2), write('\nDefect: Redundancy5\nCauses: optional feature '), write(C1), write(' implies a full-mandatory feature '),write(C2), write('\nand mandatory child feature '), write(C3), write(' is implied by its parent '), write(C2), write(' are redundant'),write('\nCorrections: eliminate '), write(I1), write(' and '), write(I2).

Result: Figure 4.8 represents the result obtained after applying this rule to FMO.

```
Defect: Redundancy5
Causes: optional feature c1 implies a full-mandatory feature c2
and mandatory child feature c3 is implied by its mandatory parent feature c2 are redundant
Corrections: eliminate imp1 and imp2
```

Figure 4.8: Result of rule R.5.

Rule R.6

Facts: f(p1,m). f(p2,m). f(c1,o). f(c2,m). p(c1,p1). p(c2,p2). i(c1,c2,imp1).

rule6:- f(P1,m), f(P2,m), f(C1,o), f(C2,m), p(C1,P1), p(C2,P2), i(C1,C2,I1), write('\nDefect: Redundancy6\nCause: optional feature '), write(C1), write(' implies a full-mandatory feature'), write(C2), write(' is redundant\nCorrection: eliminate '), write(I1).

Result: The generated result after implementing this rule to FMO is shown in Figure 4.9.

```
Defect: Redundancy6
Cause: optional feature c1 implies a full-mandatory feature c2 is redundant
Correction: eliminate imp1
```

Figure 4.9: Result of rule R.6.

Rule R.7

Facts: f(p,m). f(c1,m). f(c2,m). p(c1,p). p(c2,p). i(p,c1,imp1). i(p,c2,imp2).

rule7:- f(P,m), f(C1,m), f(C2,m), p(C1,P), p(C2,P), i(P,C1,I1), i(P,C2,I2), write('\n Defect: Redundancy7\nCauses: both mandatory child features '), write(C1), write(' and '),

write(C2), write(' are implied by their parent '), write(P), write(' are redundant\n Corrections: eliminate '), write (I1), write(' and '), write(I2).

Result: Figure 4.10 shows the result obtained after applying this rule to FMO.

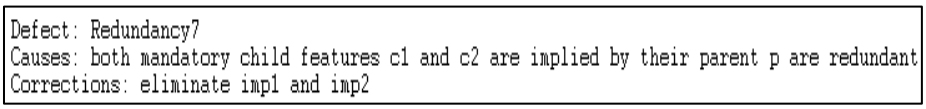


Figure 4.10: Result of rule R.7.

Rule R.8

Facts: f(p,m). f(c1,m). f(c2,o). f(c3,m). p(c1,p). p(c2,p). p(c3,c1). i(c1,c2,imp1). i(c3,c2,imp2).

rule8:- f(P,m), f(C1,m), f(C2,o), f(C3,m), p(C1,P), p(C2,P), p(C3,C1), i(C1,C2,I1), i(C3,C2,I2), write('\nDefect: Redundancy8\nCause: multiple implication of an optional feature '), write(C2), write('\nCorrection: eliminate '), write(I2).

Result: The produced result after applying this rule to FMO is shown in Figure 4.11s.

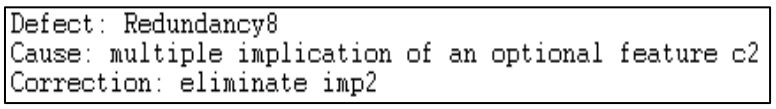


Figure 4.11: Result of rule R.8.

Rule R.9

Facts: f(p,m1). f(c1,m2). f(c2,o). f(c3,m3). f(c4,m4). p(c1,p). p(c2,p). p(c3,c1). p(c4,c2). i(c3,c2,imp1). i(c3,c4,imp2).

rule9:- f(P,m), f(C1,m), f(C2,o), f(C3,m), f(C4,m), p(C1,P), p(C2,P), p(C3,C1), p(C4,C2), i(C3,C2,I1), i(C3,C4,I2), write('\nDefect: Redundancy9\nCause: implication on '), write(C4), write(' is redundant after '), write(C2), write(' is implied by a mandatory feature '), write(C3), write('\nCorrection: eliminate '), write(I2).

Result: Figure 4.12 represents the result produced after applying this rule to FMO.

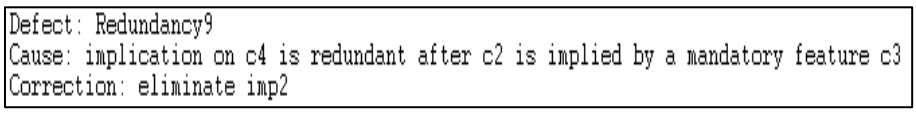


Figure 4.12: Result of rule R.9.

Rule R.10

Facts: f(p,m). f(c1,m). f(c2,o). f(c3,m). f(c4,m). p(c1,p). p(c2,p). p(c3,c1). p(c4,c2). i(c1, c2,imp1). i(c3,c4,imp2).

rule10:- f(P,m), f(C1,m), f(C2,o), f(C3,m), f(C4,m), p(C1,P), p(C2,P), p(C3,C1), p(C4,C2) , i(C1,C2,I1), i(C3,C4,I2), write('\nDefect: Redundancy10\nCause: mandatory feature '), write(C3), write(' implies another mandatory feature '), write(C4), write(' is redundant\n Correction: eliminate '), write(I2).

Result: The generated result after applying this rule to FMO is shown in Figure 4.13.

```
Defect: Redundancy10
Cause: mandatory feature c3 implies another mandatory feature c4 is redundant
Correction: eliminate imp2
```

Figure 4.13: Result of rule R.10.

Rule R.11

Facts: f(p,m). f(c1,m). f(c2,m). f(c3,m). f(c4,m). p(c1,p). p(c2,p). p(c3,c1). p(c4,c2). i(c3,c2,imp1). i(c3,c4,imp2).

rule11:- f(P,m), f(C1,m), f(C2,m), f(C3,m), f(C4,m), p(C1,P), p(C2,P), p(C3,C1), p(C4, C2), i(C3,C2,I1), i(C3,C4,I2), write('\nDefect: Redundancy11\nCauses: implications on mandatory features '), write(C2), write(' and '), write(C4), write(' are redundant '), write('\nCorrections: eliminate '), write(I1), write(' and '), write(I2).

Result: Figure 4.14 shows the result obtained after implementing this rule to FMO.

```
Defect: Redundancy11
Causes: implications on mandatory features c2 and c4 are redundant
Corrections: eliminate imp1 and imp2
```

Figure 4.14: Result of rule R.11.

Rule R.12

Facts: f(p,m). f(c1,m). f(c2,o). f(c3,m). p(c1,p). p(c2,p). p(c3,c1). e(c1,c2,exc1). e(c3,c2,exc2).

rule12:- f(P,m), f(C1,m), f(C2,o), f(C3,m), p(C1,P), p(C2,P), p(C3,C1), e(C1,C2,E1), e(C3,C2,E2), write('\nDefect: Redundancy12\nCause: multiple exclusion of an optional

feature '), write(C2),write('\nCorrection: eliminate '), write(E2).

Result: The produced result after applying this rule to FMO is shown in Figure 4.15.

```
Defect: Redundancy12
Cause:multiple exclusion of an optional feature c2
Correction: eliminate exc2
```

Figure 4.15: Result of rule R.12.

Rule R.13

Facts: f(p,m). f(c1,m). f(c2,o). f(c3,m). f(c4,m). p(c1,p). p(c2,p). p(c3,c1). p(c4,c2). e(c1,c2,exc1). e(c3,c4,exc2).

rule13:- f(P,m), f(C1,m), f(C2,o), f(C3,m), f(C4,m), p(C1,P), p(C2,P), p(C3,C1), p(C4,C2), e(C1,C2,E1), e(C3,C4,E2), write('\nDefect: Redundancy13\nCause: mandatory feature '), write(C3), write(' excludes another mandatory feature '), write(C4), write(' is redundant\nCorrection: eliminate '), write(E2).

Result: Figure 4.16 represents the result produced after implementing this rule to FMO.

```
Defect: Redundancy13
Cause: mandatory feature c3 excludes another mandatory feature c4 is redundant
Correction: eliminate exc2
```

Figure 4.16: Result of rule R.13.

Rule R.14

Facts: i(c1,c2,imp1). i(c2,c3,imp2). i(c3,c1,imp3).

rule14:- i(C1,C2,I1), i(C2,C3,I2), i(C3,C1,I3), write('\nDefect: Redundancy15\nCause: implication from '), write(C3), write(' to '), write(C1), write(' is redundant\nCorrection: eliminate '), write(I3).

Result: The generated result after applying this rule to FMO is shown in Figure 4.17.

```
Defect: Redundancy14
Cause: implication from c3 to c1 is redundant
Correction: eliminate imp3
```

Figure 4.17: Result of rule R.14.

Rule R.15

Facts: i(c1,c2,imp1). i(c2,c3,imp2). i(c1,c3,imp3).

rule15:- i(C1,C2,I1), i(C2,C3,I2), i(C1,C3,I3), write('\nDefect: Redundancy14\nCause: implication from '), write(C1), write(' to '), write(C3), write(' is redundant\nCorrection: eliminate '), write(I3).

Result: Figure 4.18 shows the result obtained after implementing this rule to FMO.

```
Defect: Redundancy15
Cause: implication from c1 to c3 is redundant
Correction: eliminate imp3
```

Figure 4.18: Result of rule R.15.

Rule R.16

Facts: f(p,m). f(c1,m). f(c2,m). p(c1,p). p(c2,c1). i(p,c2,imp1).

rule16:- f(P,m), f(C1,m), f(C2,m), p(C1,P), p(C2,C1), i(P,C2,I1), write(' '\nDefect: Redundancy16\nCause: implication from '), write(P), write(' to '),write(C2), write(' is redundant\nCorrection: eliminate '), write(I1).

Result: The result produced after applying this rule to FMO is shown in Figure 4.19.

```
Defect: Redundancy16
Cause: implication from p to c2 is redundant
Correction: eliminate imp1
```

Figure 4.19: Result of rule R.16.

Rule R.17

Facts: f(p,m). f(c1,m). f(c2,m). p(c1,p). p(c2,c1). i(c2,p,imp1).

rule17:- f(P,m), f(C1,m), f(C2,m), p(C1,P), p(C2,C1), i(C2,P,I1), write("\nDefect: Redundancy17\nCause: implication from '), write(C2), write(' to '),write(P), write(' is redundant\nCorrection: eliminate '), write(I1).

Result: Figure 4.20 represents the result produced after implementing this rule to FMO.

```
Defect: Redundancy17
Cause: implication from c2 to p is redundant
Correction: eliminate imp1
```

Figure 4.20: Result of rule R.17.

Implementation to the running example of E-commerce system1 FM: The implementation of our approach to handle defects due to redundancy along with their

causes and corrections is explained using the running example of *E-commerce system1* available in FM repository of SPLOT. The *E-commerce system1* FM has already been explained in Sub-section 4.2.1 using Figures 4.1 and 4.2.

Stage 1 - Transformation of FM into Predicate-based FMO: The transformation of *E-commerce system1* FM (see Figure 4.2) into predicate-based FMO (see Figure 4.5) has been explained in Section 4.2.

Stage 2 - Development and Applicability of FOL-based Rules: The rules discussed in Sub-section 4.3.1 are applied to the generated predicate-based FMO of *E-commerce system1* FM (see Figure 4.3).

Stage 3 – Identification of Defects, their Causes and Providing Corrections: Figure 4.22 represents a snapshot of the results generated after applying all rules to *E-commerce system1* FMO. The results comprise of identified defects due to redundancy along with their causes and provide corrections in a user-friendly natural language. The generated results can be further used by the PL developers to resolve defects due to redundancy, i.e. by eliminating cross-tree constraints involved in the cause of defects.

4.3.2 Dead Features

Sub-section 3.2.2 includes a detailed discussion on each case of defect arising due to dead feature. Following illustrates each case which includes facts in the form of FMO, rule and result.

Rule D.1

Facts: f(p,m). f(c1,m). f(c2,o). p(c1,p). p(c2,p). e(c1,c2,exc1).

rule1:- f(P,m), f(C1,m), f(C2,o), p(C1,P), p(C2,P), e(C1,C2,E1), write('Dead feature: '), write(C2), write('\nCause: full-mandatory feature '), write(C1), write(' excludes an optional feature '), write(C2), write('\nCorrection: eliminate '), write(E1).

Result: Figure 4.21 represents the result produced after implementing this rule to FMO.

```
Dead feature: c2
Cause: full-mandatory feature c1 excludes an optional feature c2
Correction: eliminate exc1
```

Figure 4.21: Result of rule D.1.

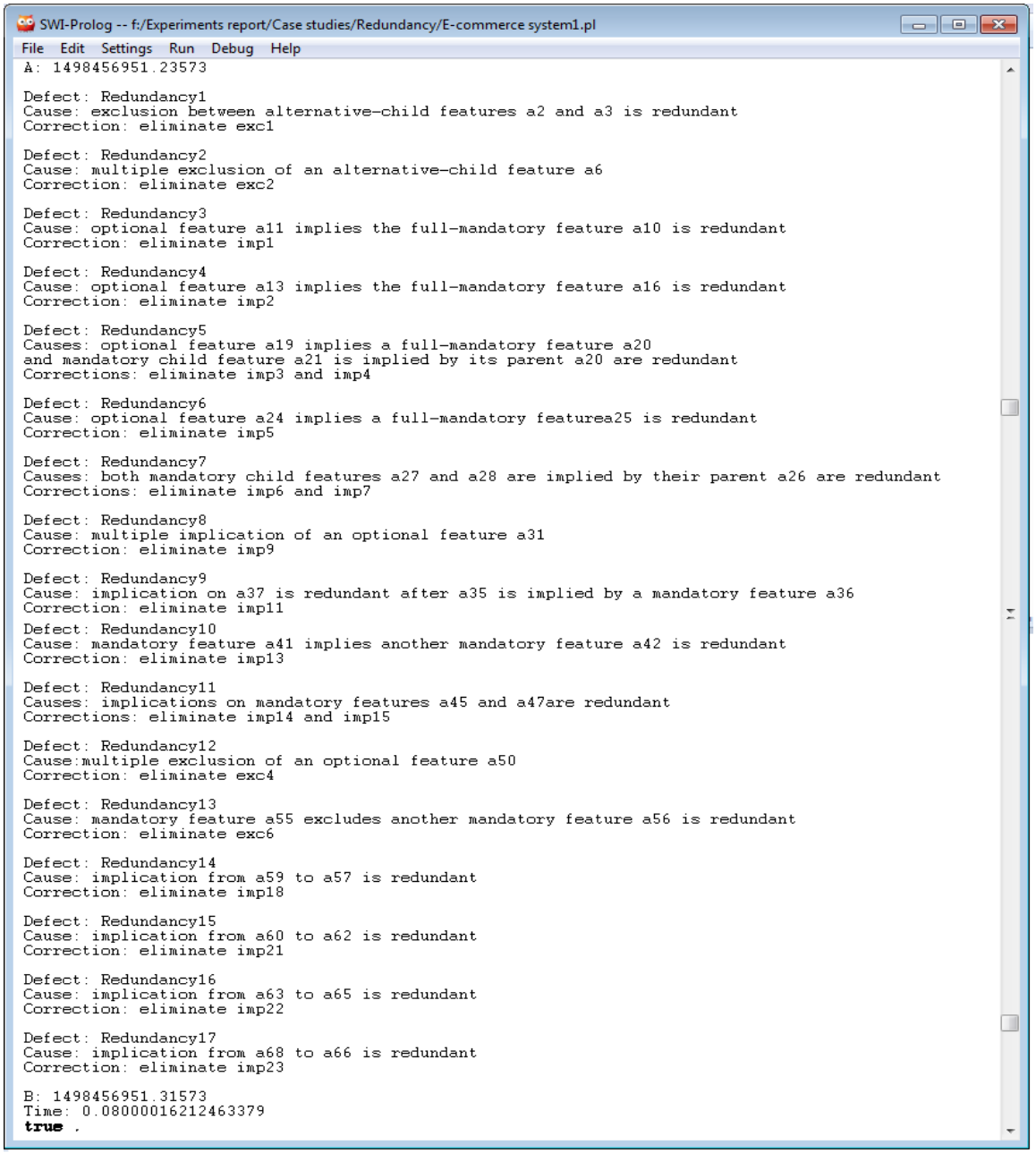


Figure 4.22: Results generated after applying all rules to E-commerce system1 feature model.

Rule D.2

Facts: f(p,m). f(c1,m). f(c2,o). f(c3,o). p(c1,p). p(c2,p). p(c3,c2). e(c1,c2,exc1).

rule2:- f(P,m), f(C1,m), f(C2,o), f(C3,o), p(C1,P), p(C2,P), p(C3,C2), e(C1,C2,E1), write ('Dead features: '), write(C2), write(' and '), write(C3), write('\nCauses: full-mandatory feature '), write(C1), write(' excludes an optional feature '), write(C2), write('\nand optional feature '), write(C3), write(' is a child feature of dead feature '), write(C2), write('

\nCorrection: eliminate '), write(E1).

Result: The generated result after applying this rule to FMO is shown in Figure 4.23.

```
Dead features: c2 and c3
Causes: full-mandatory feature c1 excludes an optional feature c2
and optional feature c3 is a child feature of dead feature c2
Correction: eliminate exc1
```

Figure 4.23: Result of rule D.2.

Rule D.3

Facts: f(p,m). f(c1,m). f(c2,o). f(c3,m). p(c1,p). p(c2,p). p(c3,c1). e(c3,c2,exc1).

rule3:- f(P,m), f(C1,m), f(C2,o), f(C3,m), p(C1,P), p(C2,P), p(C3,C1), e(C3,C2,E1), write('Dead feature: '), write(C2), write('\nCause: full-mandatory feature '), write(C3), write(' excludes an optional feature '), write(C2), write('\nCorrection: eliminate '), write(E1).

Result: Figure 4.24 shows the result obtained after implementing this rule to FMO.

```
Dead feature: c2
Cause: full-mandatory feature c3 excludes an optional feature c2
Correction: eliminate exc1
```

Figure 4.24: Result of rule D.3.

Rule D.4

Facts: f(p,m). f(c1,m). f(c2,o). f(c3,m). f(c4,o). p(c1,p). p(c2,p). p(c3,c1). p(c4,c2). e(c3,c2,exc1).

rule4:- f(P,m), f(C1,m), f(C2,o), f(C3,m), f(C4,o), p(C1,P), p(C2,P), p(C3,C1), p(C4,C2), e(C3,C2,E1), write('Dead features: '), write(C2), write(' and '), write(C4), write('\nCauses: full-mandatory feature '), write(C3), write(' excludes an optional feature '), write(C2),write('\nand optional feature '), write(C4), write(' is a child feature of dead feature '), write(C2), write('\nCorrection: eliminate '), write(E1).

Result: The produced result after applying this rule to FMO is shown in Figure 4.25.

```
Dead features: c2 and c4
Causes: full-mandatory feature c3 excludes an optional feature c2
and optional feature c4 is a child feature of dead feature c2
Correction: eliminate exc1
```

Figure 4.25: Result of rule D.4.

Rule D.5

Facts: f(p1,m). f(p2,o). f(c1,m). f(c2,o). p(c1,p1). p(c2,p2). e(c1,c2,exc1).

rule5:- f(P1,m), f(P2,o), f(C1,m), f(C2,o), p(C1,P1), p(C2,P2), e(C1,C2,E1), write('Dead feature: '), write(C2), write('\nCause: full-mandatory feature '), write(C1), write(' excludes an optional feature '), write(C2), write('\nCorrection: eliminate '), write(E1).

Result: Figure 4.26 represents the result produced after implementing this rule to FMO.

```
Dead feature: c2
Cause: full-mandatory feature c1 excludes an optional feature c2
Correction: eliminate exc1
```

Figure 4.26: Result of rule D.5.

Rule D.6

Facts: f(p,m). f(c1,m). f(c2,o). f(c3,o). p(c1,p). p(c2,c1). p(c3,c1). e(p,c2,exc1). e(p,c3,exc2).

rule6:- f(P,m), f(C1,m), f(C2,o), f(C3,o), p(C1,P), p(C2,C1), p(C3,C1), e(P,C2,E1), e(P,C3,E2), write('Dead features: '), write(C2), write(' and '), write(C3), write('\nCauses: full-mandatory parent feature '), write(P), write(' excludes optional child features '), write(C2), write(' and '), write(C3), write('\nCorrections: eliminate '), write(E1), write(' and '), write(E2).

Result: The generated result after applying this rule to FMO is shown in Figure 4.27.

```
Dead features: c2 and c3
Causes: full-mandatory parent feature p excludes optional child features c2 and c3
Corrections: eliminate exc1 and exc2
```

Figure 4.27: Result of rule D.6.

Rule D.7

Facts: f(p,m). f(c1,m). f(c2,o). p(c1,p). p(c2,p). e(p,c2,exc1).

rule7:- f(P,m), f(C1,m), f(C2,o), p(C1,P), p(C2,P), e(P,C2,E1), write('Dead feature: '), write(C2) , write('\nCause: full-mandatory parent feature '), write(P), write(' excludes an optional child feature '), write(C2), write('\nCorrection: eliminate '), write(E1).

Result: Figure 4.28 shows the result obtained after implementing this rule to FMO.

```
Dead feature: c2
Cause: full-mandatory parent feature p excludes an optional child feature c2
Correction: eliminate exc1
```

Figure 4.28: Result of rule D.7.

Rule D.8

Facts: f(p,m). f(c1,m). f(c2,o). f(c3,o). f(c4,o). p(c1,p). p(c2,p). p(c3,c2). p(c4,c2). i(C1,C3,imp1). e(c1,c4,exc1).

rule8:- f(P,m), f(C1,m), f(C2,o), f(C3,o), f(C4,o), p(C1,P), p(C2,P), p(C3,C2), p(C4,C2), i(C1,C3,I1), e(C1,C4,E1), write('Dead feature: '), write(C4), write('\nCause: full-mandatory feature '), write(C1), write(' excludes an optional feature '), write(C4), write('\nCorrection: eliminate '), write(E1).

Result: The produced result after applying this rule to FMO is shown in Figure 4.29.

```
Dead feature: c4
Cause: full-mandatory feature c1 excludes an optional feature c4
Correction: eliminate exc1
```

Figure 4.29: Result of rule D.8.

Rule D.9

Facts: f(p,m). f(c1,m). f(c2,o). f(c3,o). p(c1,p). p(c2,p). p(c3,c1). e(c1,c2,exc1). i(c3,c2,imp1).

rule9:- f(P,m), f(C1,m), f(C2,o), f(C3,o), p(C1,P), p(C2,P), p(C3,C1), e(P1,C2,E1), i(C3,C2,I1), write('Dead features: '), write(C2), write(' and '), write(C3), write('\nCauses: full-mandatory feature '), write(C1), write(' excludes an optional feature '), write(C2), write('\nand optional feature '), write(C3), write(' implies the dead feature '), write(C2), write('\nCorrections: eliminate '), write(E1), write(' and '), write(I1).

Result: Figure 4.30 represents the result produced after implementing this rule to FMO.

```
Dead features: c2 and c3
Causes: full-mandatory feature c1 excludes an optional feature c2
and optional feature c3 implies the dead feature c2
Corrections: eliminate exc1 and imp1
```

Figure 4.30: Result of rule D.9.

Rule D.10

Facts: f(p,m). f(c1,m). f(c2,o). f(c3,o). p(c1,p). p(c2,p). p(c3,c1). i(C1,C2,imp1). e(c2,c3,exc1).

rule10:- f(P,m), f(C1,m), f(C2,o), f(C3,o), p(C1,P), p(C2,P), p(C3,C1), i(C1,C2,I1), e(C2,C3,E1) , write('Dead feature: '), write(C3), write('\nCauses: full-mandatory feature '), write(C1), write(' implies an optional feature '), write(C2), write('\nand false-optional feature '), write(C2), write(' excludes an optional feature '), write(C3), write('\n Corrections: eliminate '), write(I1), write(' and '), write(E1).

Result: The generated result after applying this rule to FMO is shown in Figure 4.31.

```
Dead feature: c3
Causes: full-mandatory feature c1 implies an optional feature c2
and false-optional feature c2 excludes an optional feature c3
Corrections: eliminate imp1 and exc1
```

Figure 4.31: Result of rule D.10.

Rule D.11

Facts: f(p,m). f(c1,o). f(c2,o). f(c3,o). p(c1,p). p(c2,p). p(c3,p). i(c2,c1,imp1). i(c2,c3,imp2). e(c1,c2,exc1). e(c3,c2,exc2).

rule11:- f(P,m), f(C1,o), f(C2,o), f(C3,o), p(C1,P), p(C2,P), p(C3,P), i(C2,C1,I1), i(C2,C3,I2), e(C1,C2,E1), e(C3,C2,E2), write('Dead feature: '), write(C2), write('\n Causes: optional feature '), write(C2), write(' implies optional features '), write(C1), write(' and '), write(C3), write('\nwhere both features exclude '), write(C2), write('\nCorrections: eliminate '), write(I1), write(', '), write(I2), write(', '), write(E1), write(' and '), write(E2).

Result: Figure 4.32 shows the result obtained after implementing this rule to FMO.

```
Dead feature: c2
Causes:optional feature c2 implies optional features c1 and c3
where both features exclude c2
Corrections: eliminate imp1, imp2, exc1 and exc2
```

Figure 4.32: Result of rule D.11.

Rule D.12

Facts: f(p,m). f(c1,o). f(c2,o). f(c3,o). p(c1,p). p(c2,p). p(c3,p). i(c1,c2,imp1).

i(c2,c3,imp2). e(c2,c1,exc1). e(c3,c1,exc2).

rule12:- f(P,m), f(C1,o), f(C2,o), f(C3,o), p(C1,P), p(C2,P), p(C3,P), i(C1,C2,I1), i(C2, C3,I2), e(C2,C1,E1), e(C3,C1,E2), write('Dead feature: '), write(C1), write('\nCauses: optional feature '), write(C2), write(' is implied by another optional feature '), write(C1), write(', '), write(C2), write(' implies an optional feature '), write(C3), write('\n'), write(C1), write(' is excluded by both features '), write(C2), write(' and '), write(C3),write('\nCorrections: eliminate '), write(I6), write(', '), write(I7), write(', '), write(E1), write(' and '), write(E2).

Result: The produced result after applying this rule to FMO is shown in Figure 4.33.

```
Dead feature: c1
Causes: optional feature c2 is implied by another optional feature c1, c2 implies an optional feature c3
c1 is excluded by both features c2 and c3
Corrections: eliminate imp1, imp2, exc1 and exc2
```

Figure 4.33: Result of rule D.12.

Rule D.13

Facts: f(p,m).f(c1,o). f(c2,o). f(c3,o). f(c4,o). p(c1,p). p(c2,p). p(c3,p). p(c4,p). e(c1,c2,exc1). i(c2,c3,imp1). e(c3,c4,exc2). i(c2,c1,imp2). i(c2,c4,imp3). e(c3,c2,exc3).

rule13:- f(P,m), f(C1,o), f(C2,o), f(C3,o), f(C4,o), p(C1,P), p(C2,P), p(C3,P), p(C4,P), e(C1,C2,E1), i(C2,C3,I1), e(C3,C4,E2), i(C2,C1,I2), i(C2,C4,I3), e(C3,C2,E3), write('Dead features: '), write(C1), write(' and '), write(C2), write('\nCauses: optional feature '), write(C2), write(' is excluded by another optional feature '), write(C1), write(', '), write(C2), write(' implies an optional feature '), write(C3), write(', '), write(C4), write(' is excluded by '), write(C3), write('\n'), write(C2), write(' implies '), write(C1), write(' and'), write(C4), write(' as well as it is excluded by '), write(C3), write('\nCorrections: eliminate '), write(E1), write(', '), write(I1), write(', '), write (E2), write(', '), write(I2), write(', '), write(I3), write(' and '), write(E3).

Result: Figure 4.34 represents the result produced after implementing this rule to FMO.

Rule D.14

Facts: f(p,m). f(c1,o). f(c2,o). p(c1,p). p(c2,p). i(c1,c2,imp1). e(c2,c1,exc1).

```
Dead features: c1 and c2
Causes:optional feature c2 is  excluded by another optional feature c1, c2 implies an optional feature c3, c4 is excluded by c3
c2 implies c1 and c4 as well as it is excluded by  c3
Corrections: eliminate exc1, imp1, exc2, imp2, imp3 and exc3
```

Figure 4.34: Result of rule D.13.

rule14:- f(P,m), f(C1,o), f(C2,o), p(C1,P), p(C2,P), i(C1,C2,I1), e(C2,C1,E1),write('Dead feature: '), write(C1), write('\nCause: implication and exclusion between '), write(C1), write(' and '), write(C2), write('\nCorrections: eliminate '), write(I1), write(' and '), write(E1).

Result: The generated result after applying this rule to FMO is shown in Figure 4.35.

```
Dead feature: c1
Cause: implication and exclusion between c1 and c2
Corrections: eliminate imp1 and exc1
```

Figure 4.35: Result of rule D.14.

Rule D.15

Facts: f(p,m). f(c1,o). f(c2,o). f(c3,o). f(c4,o). p(c1,p). p(c2,p). c(c2,[c3,c4],[1,1]). e(c1,c2,exc1). i(c3,c1,imp1).

rule15:- f(P,m), f(C1,o), f(C2,o), f(C3,o), f(C4,), p(C1,P), p(C2,P), c(C2,[C3,C4],[1,1]), e(C1,C2,E16), i(C3,C1,I14), write('Dead feature: '), write(C3), write('\nCauses: optional feature '), write(C2), write(' is excluded by another optional feature '), write(C1), write('\nand alternative-child feature '), write(C3), write(' implies '), write(C1), write('\nCorrections: eliminate '), write(E1), write(' and '), write(I1).

Result: Figure 4.36 shows the result obtained after implementing this rule to FMO.

```
Dead feature: c3
Causes: optional feature c2 is excluded by another optional feature c1
and alternative-child feature c3 implies c1
Corrections: eliminate exc1 and imp1
```

Figure 4.36: Result of rule D.15.

Rule D.16

Facts: f(p,m). f(c1,o). f(c2,m). f(c3,o). f(c4,o). f(c5,o). p(c1,p). p(c2,p). c(c2,[c3,c4,c5],[1,3]). i(c2,c1,imp1). e(c1,c3,exc1).

rule16:- f(P,m), f(C1,o), f(C2,m), f(C3,o), f(C4,o), f(C5,o), p(C1,P), p(C2,P), c(C2,[C3,C4,C5],[1,3]), i(C2,C1,I1), e(C1,C3,E1), write('Dead feature: '), write(C3), write('\nCauses: full-mandatory feature '), write(C2), write(' implies an optional feature '), write(C1), write('\nand false-optional feature '), write(C1), write(' excludes an or-child feature '), write(C3), write('\nCorrections: eliminate '), write(I1), write(' and '), write(E1).

Result: The produced result after applying this rule to FMO is shown in Figure 4.37.

```
Dead feature: c3
Causes: full-mandatory feature c2 implies an optional feature c1
and false-optional feature c1 excludes an or-child feature  c3
Corrections: eliminate imp1 and exc1
```

Figure 4.37: Result of rule D.16.

Rule D.17

Facts: f(p,m). f(c1,m). f(c2,m). f(c3,o). f(c4,o). p(c1,p). p(c2,p). c(c2,[c3,c4],[1,1]). i(c1,c3,imp1).

rule17:- f(P,m), f(C1,m), f(C2,m), f(C3,o), f(C4,o), p(C1,P), p(C2,P), c(C2,[C3,C4],[1,1]), i(C1,C3,I1), write('Dead feature: '), write(C4), write('\nCauses: full-mandatory feature'), write(C1), write(' implies an alternative-child feature '), write(C2), write('\nCorrection: eliminate '), write(I1).

Result: Figure 4.38 represents the result produced after implementing this rule to FMO.

```
Dead feature: c4
Causes: full-mandatory feature c1 implies an alternative-child feature c2
Correction: eliminate imp1
```

Figure 4.38: Result of rule D.17.

Rule D.18

Facts: f(p1,m). f(p2,m). f(c1,o). f(c2,o). f(c3,o). p(c1,p1). c(p2,[c2,c3],[1,1]). i(c1,c2,imp1). i(c1,c3,imp2).

rule18:- f(P1,m), f(P2,m), f(C1,o1),f(C2,o2), f(C3,o3),p(C1,P1), c(P2,[C2,C3],[1,1]), i(C1,C2,I1), i(C1,C3,I2), write('Dead feature: '), write(C1), write('\nCause: optional feature'),write(C1),write(' implies alternative-child features '), write(C2), write(' and '), write(C3), write ('\nCorrections: eliminate '), write(I1), write(' and '), write(I2).

Result: The generated result after applying this rule to FMO is shown in Figure 4.39.

```
Dead feature: c1
Cause: optional feature c1 implies alternative-child features c2 and c3
Corrections: eliminate imp1 and imp2
```

Figure 4.39: Result of rule D.18.

Rule D.19

Facts: f(p,m). f(c1,m). f(c2,m). f(c3,o). f(c4,o). p(c1,p). p(c2,p). c(c2,[c3,c4],[1, 1]). e(c1,c3,exc1).

rule19:- f(P,m), f(C1,m), f(C2,m), f(C3,o), f(C4,o), p(C1,P), p(C2,P), c(C2,[C3,C4],[1,1]), e(C1,C3,E1), write('Dead feature: '), write(C3), write('\nCause: full-mandatory feature'), write(C1), write(' excludes an alternative-child feature '), write(C3), write('\nCorrection: eliminate '), write(E1).

Result: Figure 4.40 shows the result obtained after implementing this rule to FMO.

```
Dead feature: c3
Cause: full-mandatory feature c1 excludes an alternative-child feature c3
Correction: eliminate exc1
```

Figure 4.40: Result of rule D.19.

Rule D.20

Facts: f(p,m). f(c1,m1). f(c2,m2). f(c3,o1). f(c4,o2). p(c1,p). p(c2,p). c(c2,[c3,c4],[1,2]). e(c1,c3,exc1).

rule20:- f(P,m), f(C1,m), f(C2,m), f(C3,o), f(C4,o), p(C1,P), p(C2,P), c(C2,[C3,C4],[1,2]), e(C1,C3,E18), write('Dead feature: '), write(C3), write('\nCause: full-mandatory feature '), write(C1), write(' excludes an or-child feature '), write(C3), write('\nCorrection: eliminate '), write(E1).

Result: The produced result after applying this rule to FMO is shown in Figure 4.41.

```
Dead feature: c3
Cause: full-mandatory feature c1 excludes an or-child feature c3
Correction: eliminate exc1
```

Figure 4.41: Result of rule D.20.

Implementation to the running example of CIMS PL FM: The *CIMS PL* available in FM repository of SPLOT is used as a running example to explain the implementation of our approach to handle defects due to dead feature along with their causes and corrections. The graphical representation of the corresponding model is shown in Figure 4.42. In order to illustrate our approach, additional features (i.e. *b1*, *b2*, *b3*, *b4*, *b5*, *b6*, *b7*, *b8*, *b9*, *b10*, *b11*, *b12*, *b13*, *b14*, *b15*, *b16*, *b17*, *b18*, *b19*, *b20*, *b21*, *b22*, *b23*, *b24*, *b25*, *b26*, *b27*, *b28*, *b29*, *b30*, *b31*, *b32*, *b33*, *b34*, *b35*, *b36*, *b37*, *b38*, *b39*, *b40*, *b41*, *b42*, *b43*, *b44*, *b45*, *b46*, *b47*, *b48*, *b49*, *b50*, *b51*, *b52*, *b53*, *b54*, *b55*, *b56*, *b57*, *b58*, *b59*, *b60*, *b61*, *b62*, *b63*, *b64*, *b65*, *b66*, *b67*, *b68*, *b69*, *b70*, *b71*, *b72*, *b73*, *b74*, *b75*, *b76*, *b77*, *b78*) and cross-tree constraints (i.e. *imp1*, *imp2*, *imp3*, *imp4*, *imp5*, *imp6*, *imp7*, *imp8*, *imp9*, *imp10*, *imp11*, *imp12*, *imp13*, *imp14*, *imp15*, *imp16*, *exc1*, *exc2*, *exc3*, *exc4*, *exc5*, *exc6*, *exc7*, *exc8*, *exc9*, *exc10*, *exc11*, *exc12*, *exc13*, *exc14*, *exc15*, *exc16*, *exc17*, *exc18*, *exc19*, *exc20*, *exc21*, *exc22*, *exc23)* are intentionally injected in the input *CIMS PL* FM to cause defects due to dead feature as shown in Figure 4.43.

Stage 1 - Transformation of FM into Predicate-based FMO: The running example of *CIMS PL* FM (see Figure 4.43) is transformed into predicate-based FMO (see Figure 4.44) using transformation details explained in Section 4.2.

Stage 2 - Development and Applicability of FOL-based Rules: The rules discussed in Sub-section 4.3.2 are applied to the generated predicate-based FMO of *CIMS PL* FM (see Figure 4.44).

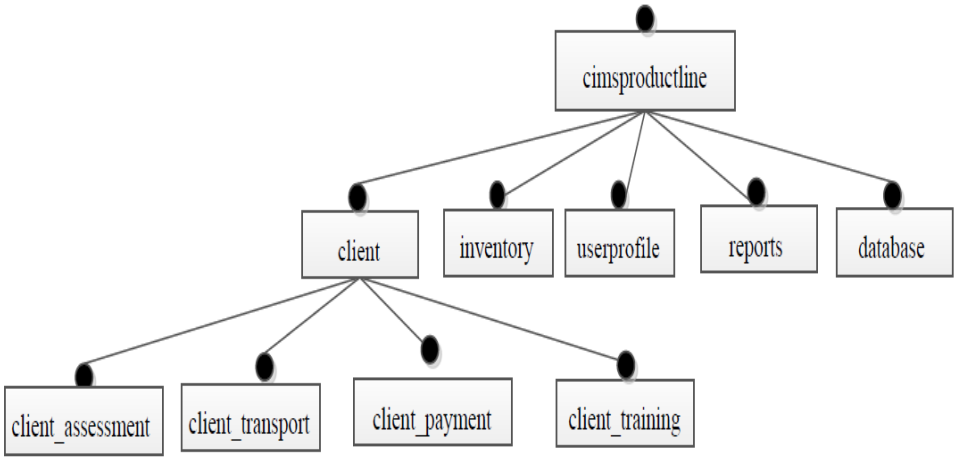
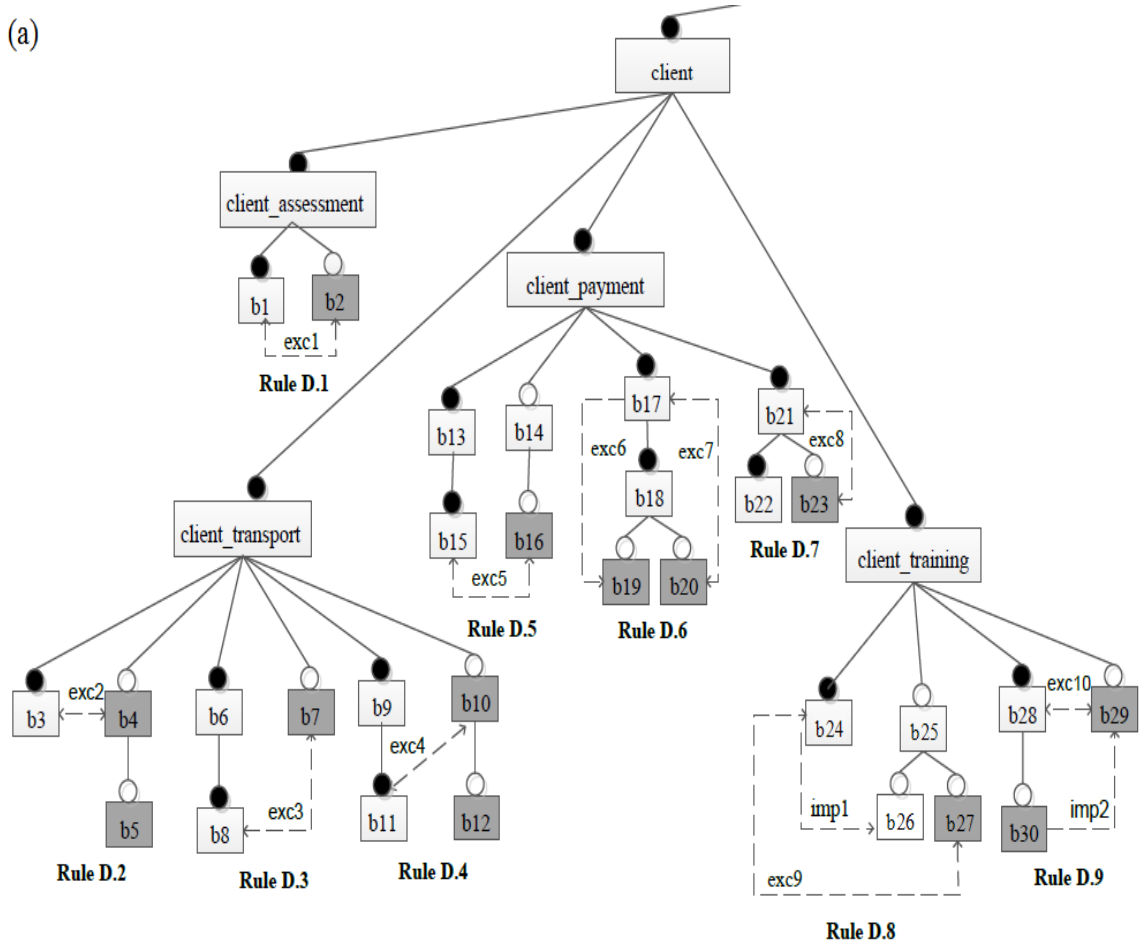
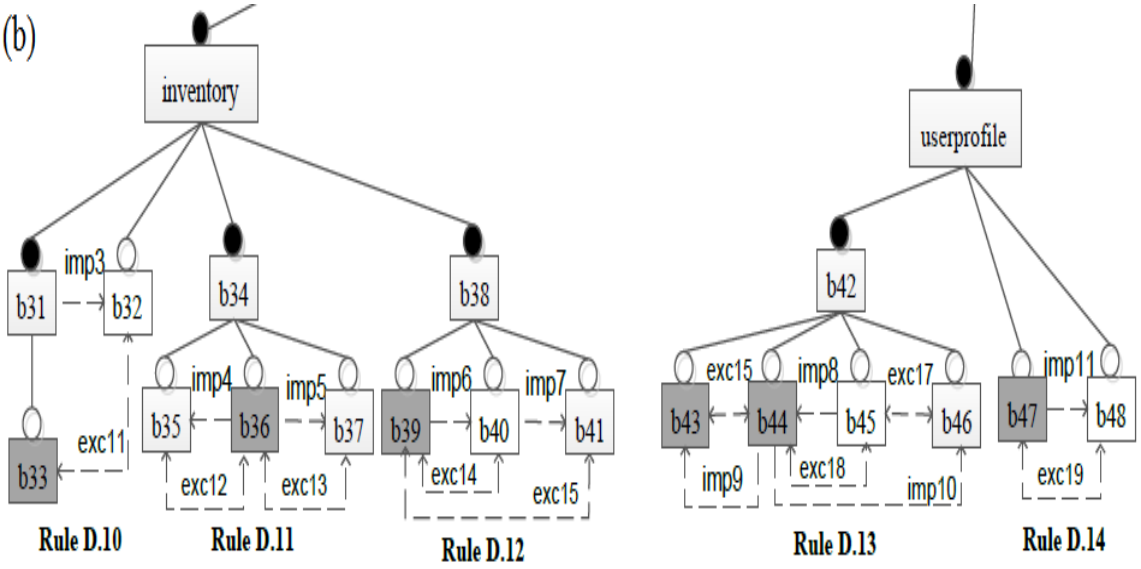


Figure 4.42: CIMS PL feature model from SPLOT repository.


(a)
client
client_assessment
client_payment
client_transport
client_training
b1
b2
exc1
Rule D.1
b13
b14
b15
b16
exc5
Rule D.5
b17
exc6
exc7
b18
b19
b20
Rule D.6
b21
exc8
b22
b23
Rule D.7
b3
exc2
b4
b5
Rule D.2
b6
b7
b8
exc3
Rule D.3
b9
exc4
b10
b11
b12
Rule D.4
b24
b25
b26
imp1
b27
b28
exc10
b29
b30
imp2
exc9
Rule D.8
Rule D.9


(b)
inventory
userprofile
b31
imp3
b32
b33
exc11
Rule D.10
b34
b35
imp4
b36
imp5
b37
exc12
exc13
Rule D.11
b38
b39
imp6
b40
imp7
b41
exc14
exc15
Rule D.12
b42
b43
exc15
b44
imp8
b45
exc17
b46
imp9
exc18
imp10
Rule D.13
b47
imp11
b48
exc19
Rule D.14

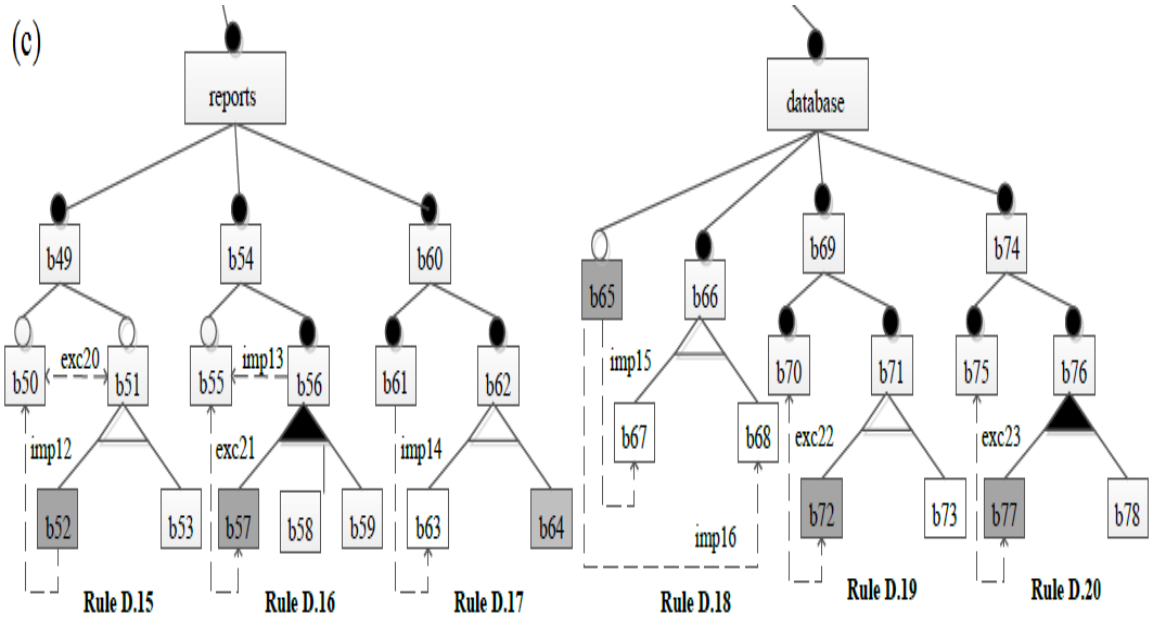


Figure 4.43: CIMS PL feature model where grey features represent defects due to dead feature (a) D.1-D.9, (b) D.10-D.14, and (c) D.15-D.20.

Stage 3 - Identification of Defects, their Causes and Providing Corrections: Figure 4.45 represents a snapshot of the results generated after applying all rules to *CIMS PL* FMO. The results comprise of identified defects due to dead feature along with their causes and corrections in a user-friendly natural language. The generated results can be further used by the PL developers to resolve defects due to dead feature, i.e. by eliminating cross-tree constraints involved in the cause of defects.

4.3.3 False-Optional Features

Sub-section 3.2.3 provides a detailed discussion on each case of defect arising due to false-optional feature. Following illustrates each case which includes facts in the form of FMO, rule and result.

Rule F.1

Facts: f(p,m). f(c1,m). f(c2,o). p(c1,p). p(c2,p). i(c1,c2,imp1).

rule1:- f(P,m), f(C1,m), f(C2,o), p(C1,P), p(C2,P), i(C1,C2,I1), write('False-optional feature: '), write(C2), write('\n Cause: full-mandatory feature '), write(C1), write(' implies an optional feature '), write(C2), write('\nCorrection: eliminate '), write(I1).

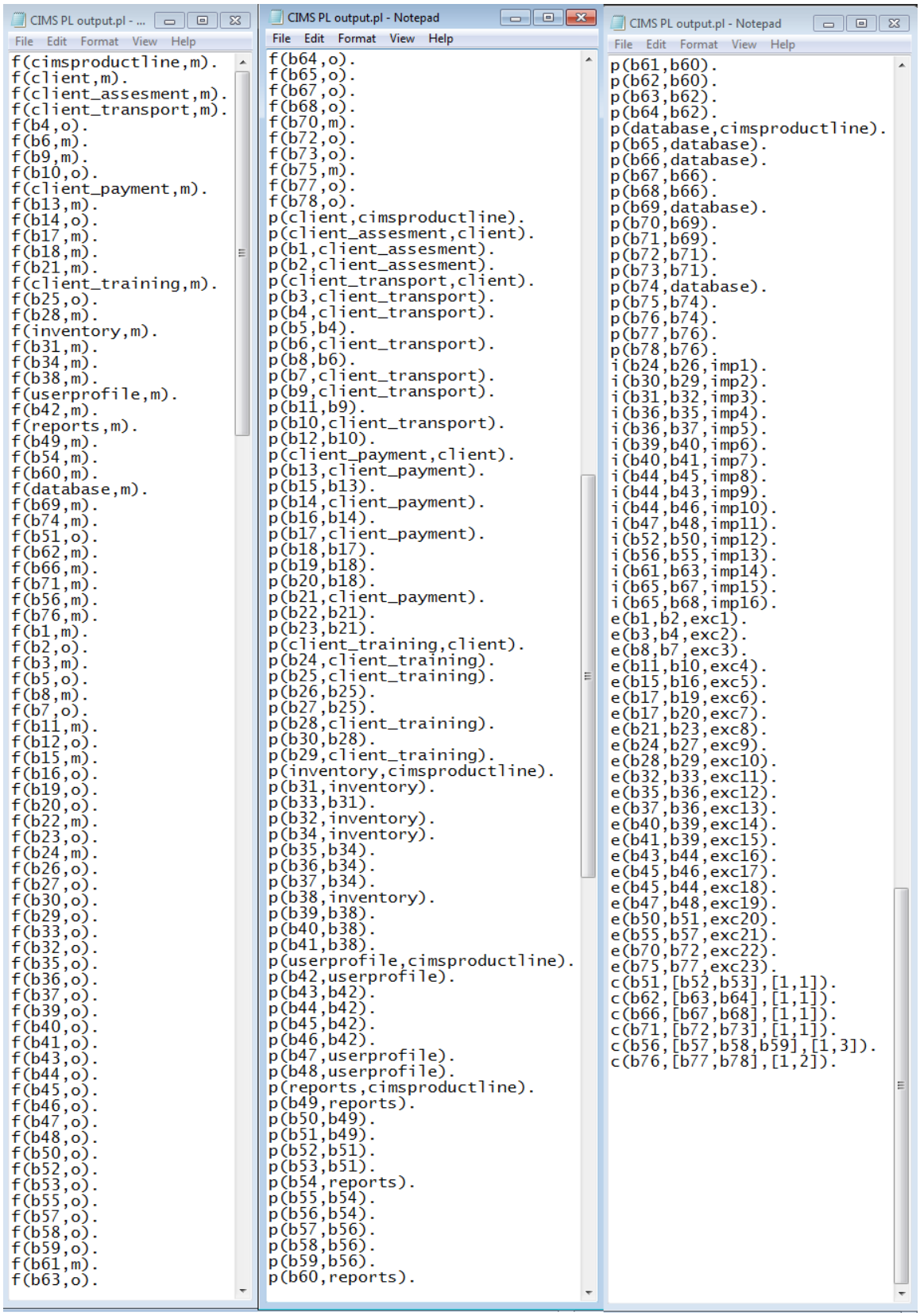


Figure 4.44: Feature model ontology of CIMS PL feature model.

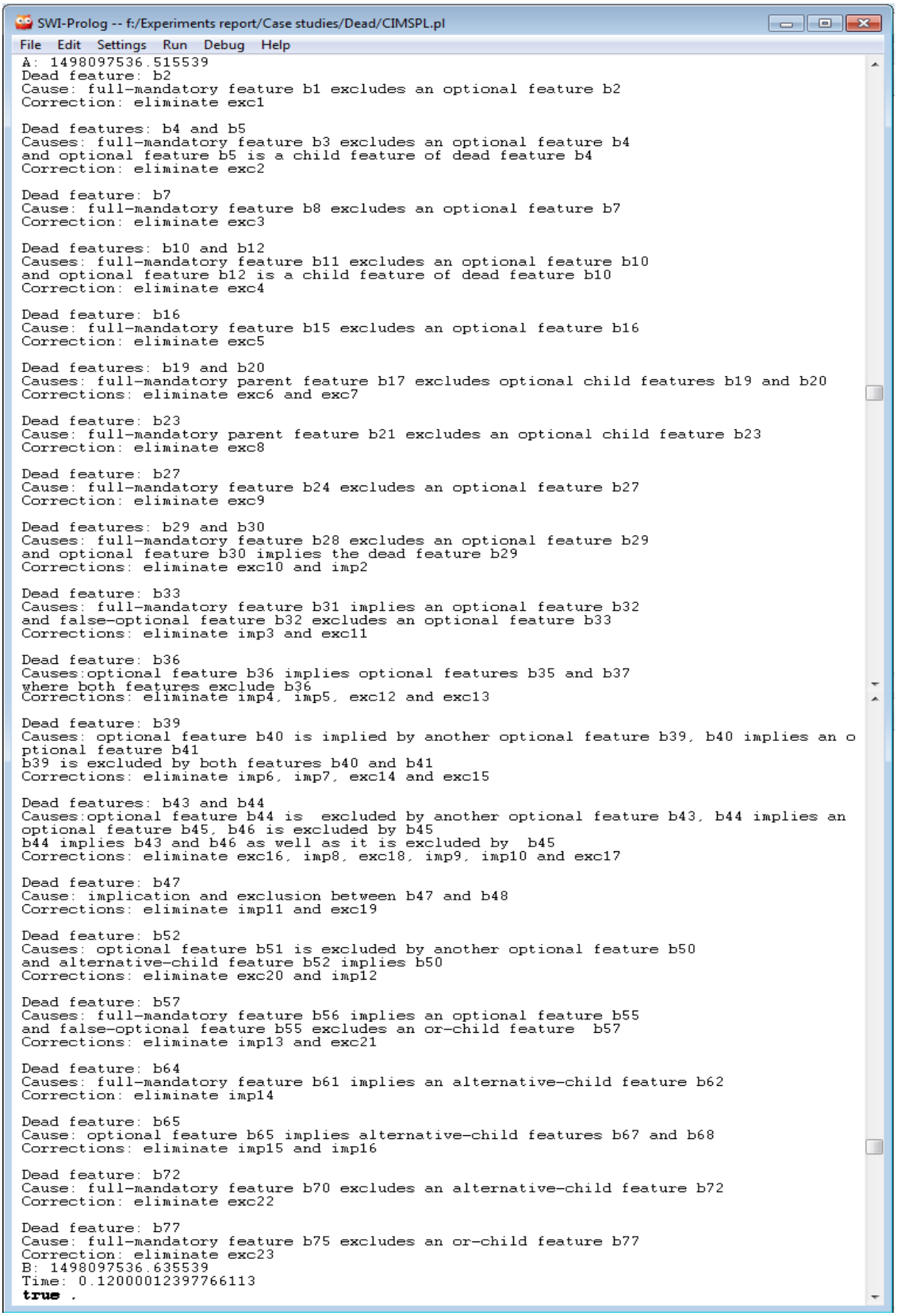


Figure 4.45: Results generated after applying all rules to CIMS PL feature model.

Result: Figure 4.46 represents the result produced after implementing this rule to FMO.

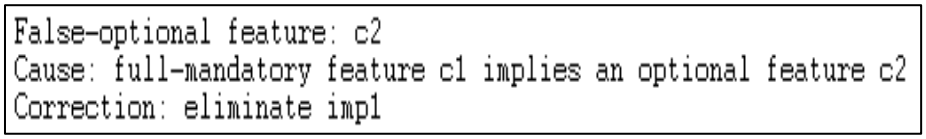


Figure 4.46: Result of rule F.1.

Rule F.2

Facts: f(p,m). f(c1,m). f(c2,o). f(c3,o). p(c1,p). p(c2,p). p(c3,c2). i(c1,c2,imp1). i(c2,c3,imp2).

rule2:- f(P,m), f(C1,m), f(C2,o), f(C3,o), p(C1,P), p(C2,P), p(C3,C2), i(C1,C2,I1), i(C2,C3,I2), write('False-optional features: '), write(C2), write(' and '), write(C3), write('\nCauses: full-mandatory feature '), write(C1), write(' implies an optional feature '), write(C2), write('\nand optional feature '), write(C3), write(' is implied by false-optional feature '), write(C2), write('\nCorrections: eliminate '), write(I1), write(' and '), write(I2).

Result: The generated result after applying this rule to FMO is shown in Figure 4.47.

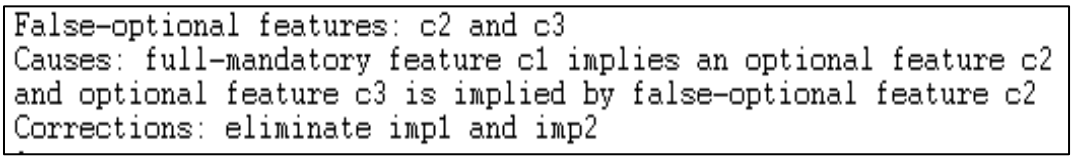


Figure 4.47: Result of rule F.2.

Rule F.3

Facts: f(p,m). f(c1,m). f(c2,o). f(c3,o). p(c1,p). p(c2,p). p(c3,c1). i(c1,c2,imp1). i(c2,c3,imp2).

rule3:- f(P,m), f(C1,m), f(C2,o), f(C3,o), p(C1,P), p(C2,P), p(C3,C1), i(C1,C2,I1), i(C2,C3,I2), write('False-optional features: '), write(C2), write(' and '), write(C3), write('\nCauses: full-mandatory feature '), write(C1), write(' implies an optional feature '), write(C2), write('\nand false-optional feature '), write(C2), write(' implies another optional feature '), write(C3), write('\nCorrections: eliminate '), write(I1), write(' and '), write(I2).

Result: Figure 4.48 shows the result obtained after implementing this rule to FMO.

```
False-optional features: c2 and c3
Causes: full-mandatory feature c1 implies an optional feature c2
and false-optional feature c2 implies another optional feature c3
Corrections: eliminate imp1 and imp2
```

Figure 4.48: Result of rule F.3.

Rule F.4

Facts: f(p,m). f(c1,m). f(c2,o). f(c3,m). p(c1,p). p(c2,p). p(c3,c1). i(c3,c2,imp1).

rule4:- f(P,m), f(C1,m), f(C2,o), f(C3,m), p(C1,P), p(C2,P), p(C3,C1), i(C3,C2,I1), write('False-optional feature: '), write(C2), write('\nCause: full-mandatory feature '), write(C3), write(' implies an optional feature '), write(C2), write('\nCorrection: eliminate '), write(I1).

Result: The produced result after applying this rule to FMO is shown in Figure 4.49.

```
False-optional feature: c2
Cause: full-mandatory feature c3 implies an optional feature c2
Correction: eliminate imp1
```

Figure 4.49: Result of rule F.4.

Rule F.5

Facts: f(p1,m). f(p2,o). f(c1,m). f(c2,o). p(c1,p1). p(c2,p2). i(c1,c2,imp1).

rule5:- f(P1,m), f(P2,o), f(C1,m), f(C2,o), p(C1,P1), p(C2,P2), i(C1,C2,I1), write('False-optional feature: '), write(C2), write('\nCause: full-mandatory feature '), write(C1), write(' implies an optional feature '), write(C2), write('\nCorrection: eliminate '), write(I1).

Result: Figure 4.50 represents the result produced after implementing this rule to FMO.

```
False-optional feature: c2
Cause: full-mandatory feature c1 implies an optional feature c2
Correction: eliminate imp1
```

Figure 4.50: Result of rule F.5.

Rule F.6

Facts: f(p,m). f(c1,m). f(c2,o). f(c3,o). f(c4,o). p(c1,p). p(c2,p). p(c3,c2). p(c4,c2). i(c1,c3,imp1). e(c1,c4,exc1).

rule6:- f(P,m), f(C1,m), f(C2,o), f(C3,o), f(C4,o), p(C1,P), p(C2,P), p(C3,C2), p(C4,C2),

i(C1,C3,I1), e(C1,C4,E1), write('False-optional feature: '), write(C3), write('\nCause: full-mandatory feature '), write(C1), write(' implies an optional feature '), write(C3), write('\nCorrection: eliminate '), write(I1).

Result: The generated result after applying this rule to FMO is shown in Figure 4.51.

```
False-optional feature: c3
Cause: full-mandatory feature c1 implies an optional feature c3
Correction: eliminate imp1
```

Figure 4.51: Result of rule F.6.

Rule F.7

Facts: f(p,m). f(c1,o). f(c2,o). p(c1,p). p(c2,p). i(p,c1,imp1).

rule7:- f(P,m), f(C1,o), f(C2,o), p(C1,P), p(C2,P), i(P,C1,I1), write('False-optional feature: '), write(C1), write('\nCause: full-mandatory feature '), write(P), write(' implies its optional child feature '), write(C1), write('\nCorrection: eliminate '), write(I1).

Result: Figure 4.52 shows the result obtained after implementing this rule to FMO.

```
False-optional feature: c1
Cause: full-mandatory feature p implies its optional child feature c1
Correction: eliminate imp1
```

Figure 4.52: Result of rule F.7.

Rule F.8

Facts: f(p,m). f(c1,m). f(c2,m). f(c3,m). f(c4,o). f(c5,o). f(c6,o). p(c1,p). p(c2,p). p(c3,p). c(c2,[c4,c5],[1,1]). i(c1,c4,imp1). i(c4,c6,imp2). e(c3,c5,exc1).

rule8:- f(P,m), f(C1,m), f(C2,m), f(C3,m), f(C4,o), f(C5,o), f(C6,o), p(C1,P), p(C2,P), p(C3,P), c(C2,[C4,C5],[1,1]), i(C1,C4,I1), i(C4,C6,I2), e(C3,C5,E1), write('False-optional features: '), write(C4), write(' and '), write(C6),write('\nCauses:alternative-child feature '), write(C4),write(' is implied by full-mandatory feature '), write(C1), write('\nand false-optional feature '), write(C4), write(' implies another optional feature '), write(C6), write('\nCorrections: eliminate '), write(I1), write(' and '), write(I2).

Result: The produced result after applying this rule to FMO is shown in Figure 4.53.

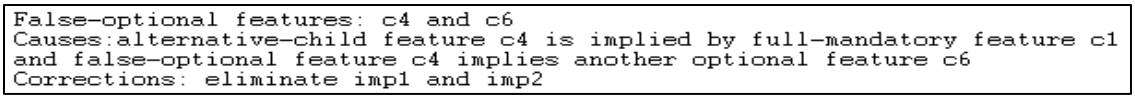


Figure 4.53: Result of rule F.8.

Rule F.9

Facts: f(p,m). f(c1,o). f(c2,o). f(c3,m). f(c4,o). f(c5,o). p(c1,p). p(c2,p). p(c3,c1). c(c3,[c4,c5],[1,1]). i(c1,c2,imp1). e(c2,c5,exc1).

rule9:- f(P,m), f(C1,o), f(C2,o), f(C3,m), f(C4,o), f(C5,o), p(C1,P), p(C2,P), p(C3,C1), c(C3,[C4,C5],[1,1]), i(C1,C2,I1), e(C2,C5,E1), write('False-optional feature: '), write(C4), write('\n Causes: optional feature '), write(C2), write(' is implied by another optional feature '), write(C1), write('\nand alternative-child feature '), write(C5), write(' is excluded by '), write(C2), write('\nCorrection: eliminate '), write(I1), write(' and '), write(E1).

Result: Figure 4.54 represents the result produced after implementing this rule to FMO.

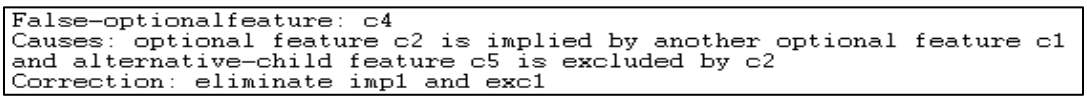


Figure 4.54: Result of rule F.9.

Rule F.10

Facts: f(p,m). f(c1,o1). f(c2,o2). f(c3,o3). f(c4,o4). p(c1,p). p(c2,p). c(c2,[c3,c4],[1,1]). e(c1,c2,exc1). i(c3,c1,imp1).

rule10:- f(P,m), f(C1,o), f(C2,o), f(C3,o), f(C4,o), p(C1,P), p(C2,P), c(C2,[C3,C4],[1,1]), e(C1,C2,E1), i(C3,C1,I1), write('False-optional feature: '), write(C4) , write('\nCauses: optional feature '), write(C2), write(' is excluded by another optional feature '), write(C1), write('\nand alternative-child feature '), write(C3), write(' implies '), write(C1), write('\nCorrections: eliminate '), write(E1), write(' and '), write(I1).

Result: The generated result after applying this rule to FMO is shown in Figure 4.55.

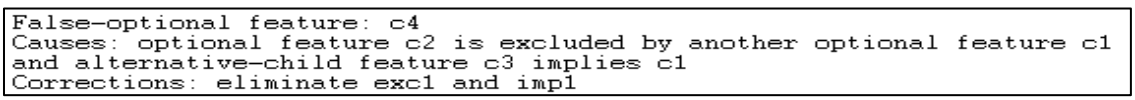


Figure 4.55: Result of rule F.10.

Rule F.11

Facts: f(p,m). f(c1,m). f(c2,m). f(c3,o). f(c4,o). f(c5,o). f(c6,o). f(c7,o). f(c8,o). p(c1,p). p(c2,p). p(c3,p). p(c6,c3). c(c1,[c4,c5],[1,1]). c(c6,[c7,c8],[1,1]). i(c5,c7,imp1). e(c2,c6,exc1).

rule11:-f(P,m), f(C1,m) ,f(C2,m), f(C3,o), f(C4,o), f(C5,o), f(C6,o), f(C7,o), f(C8,o), p(C1,P), p(C2,P), p(C3,P), p(C6,C3), c(C1,[C4,C5],[1,1]), c(C6,[C7,C8],[1,1]), i(C5,C7,I1), e(C2,C6,E1), write('False-optional feature: '), write(C4), write('\nCauses: full-mandatory feature '), write(C2),write(' excludes an optional feature '), write(C6), write('\nand alternative-child feature '), write(C7), write(' is implied by another alternative-child feature '), write(C5), write('\nCorrections: eliminate '), write(I1), write(' and '), write(E1).

Result: Figure 4.56 shows the result obtained after implementing this rule to FMO.

```
False-optional feature: c4
Causes: full-mandatory feature c2 excludes an optional feature c6
and alternative-child feature c7 is implied by another alternative-child feature c5
Corrections: eliminate imp1 and exc1
```

Figure 4.56: Result of rule F.11.

Rule F.12

Facts: f(p,m). f(c1,m). f(c2,m). f(c3,o). f(c4,o). p(c1,p). p(c2,p). c(c2,[c3,c4],[1,1]). i(c1,c3,imp1).

rule12:- f(P,m), f(C1,m), f(C2,m), f(C3,o), f(C4,o), p(C1,P), p(C2,P), c(C2,[C3,C4], [1,1]), i(C1,C3,I1), write('False-optional feature: '), write(C3), write('\nCause: full-mandatory feature '), write(C1), write(' implies an alternative-child feature '), write(C3), write('\nCorrection: eliminate '), write(I1).

Result: The produced result after applying this rule to FMO is shown in Figure 4.57.

```
False-optional feature: c3
Cause: full-mandatory feature c1 implies an alternative-child feature c3
Correction: eliminate imp1
```

Figure 4.57: Result of rule F.12.

Rule F.13

Facts: f(p,m). f(c1,m). f(c2,m). f(c3,o). f(c4,o). p(c1,p). p(c2,p). c(c2,[c3,c4],[1,2]). i(c1,c3,imp1).

rule13:- f(P,m), f(C1,m), f(C2,m), f(C3,o), f(C4,o), p(C1,P), p(C2,P), c(C2,[C3,C4], [1,2]), i(C1,C3,I1), write('False-optional feature: '), write(C3), write('\nCause: full-mandatory feature '), write(C1), write(' implies an or-child feature '), write(C3), write('\nCorrection: eliminate '), write(I1).

Result: Figure 4.58 represents the result produced after implementing this rule to FMO.

```
False-optional feature: c3
Cause: full-mandatory feature c1 implies an alternative-child feature c3
Correction: eliminate imp1
```

Figure 4.58: Result of rule F.13.

Rule F.14

Facts: f(p,m). f(c1,m). f(c2,o). f(c3,o). f(c4,o). p(c1,p). p(c2,p). c(c2,[c3,c4],[1,1]). i(c1,c3,imp1).

rule14:- f(P,m), f(C1,m), f(C2,o), f(C3,o), f(C4,o), p(C1,P), p(C2,P), c(C2,[C3,C4], [1,1]), i(C1,C3,I1), write('False-optional features: '), write(C2), write(' and '), write(C3), write('\nCauses: full-mandatory feature '), write(C1), write(' implies an alternative-child feature '), write(C3), write(' of '), write(C2), write('\nCorrection: eliminate '), write(I1).

Result: The generated result after applying this rule to FMO is shown in Figure 4.59.

```
False-optional features: c2 and c3
Causes: full-mandatory feature c1 implies an alternative-child feature c3 of c2
Correction: eliminate imp1
```

Figure 4.59: Result of rule F.14.

Rule F.15

Facts: f(p,m). f(c1,m). f(c2,o). f(c3,o). f(c4,o). p(c1,p). p(c2,p). c(c2,[c3,c4],[1,2]). i(c1,c3,imp1).

rule15:- f(P,m), f(C1,m), f(C2,o), f(C3,o), f(C4,o), p(C1,P), p(C2,P), c(C2,[C3,C4], [1,2]), i(C1,C3,I1), write('False-optional features: '), write(C2), write(' and '), write(C3),

write('\nCauses: full-mandatory feature '), write(C1), write(' implies an or-child feature '), write(C3), write(' of '), write(C2), write('\nCorrection: eliminate '), write(I1).

Result: Figure 4.60 shows the result obtained after implementing this rule to FMO.

```
False-optional features: c2 and c3
Causes: full-mandatory feature c1 implies an or-child feature c3 of c2
Correction: eliminate imp1
```

Figure 4.60: Result of rule F.15.

Rule F.16

Facts: f(p,m). f(c1,m). f(c2,m). f(c3,o). f(c4,o). p(c1,p). p(c2,p). c(c2,[c3,c4],[1,1]). e(c1,c3,exc1).

rule16:- f(P,m), f(C1,m), f(C2,m), f(C3,o), f(C4,o), p(C1,P), p(C2,P), c(C2,[C3,C4],[1,1]), e(C1,C3,E1), write('False-optional feature: '), write(C4), write('\nCause: alternative-child feature '), write(C3), write(' is excluded by a full-mandatory feature '), write(C1), write('\nCorrection: eliminate '), write(E1).

Result: The produced result after applying this rule to FMO is shown in Figure 4.61.

```
False-optional feature: c4
Cause: an alternative-child feature c3 is excluded by a full-mandatory feature c1
Correction: eliminate exc1
```

Figure 4.61: Result of rule F.16.

Rule F.17

Facts: f(p,m). f(c1,m). f(c2,m). f(c3,o). f(c4,o). p(c1,p). p(c2,p). c(c2,[c3,c4],

[1,2]). e(c1,c3,exc1).

rule17:- f(P,m), f(C1,m), f(C2,m), f(C3,o), f(C4,o), p(C1,P), p(C2,P), c(C2,[C3,C4], [1,2]), e(C1,C3,E1), write('\nCause: an or-child feature '), write(C3) , write(' is excluded by a full-mandatory feature '), write(C1), write('\nCorrection: eliminate '), write(E1).

Result: Figure 4.62 represents the result produced after implementing this rule to FMO.

```
False-optional feature: c4
Cause: an or-child feature c3 is excluded by a full-mandatory feature c1
Correction: eliminate exc1
```

Figure 4.62: Result of rule F.17.

Implementation to the running example of Computer Selection FM: The *Computer Selection* available in FM repository of SPLOT is used as a running example to explain the implementation of our approach to handle defects due to false-optional feature along with their causes and corrections. The graphical representation of the corresponding model is shown in Figure 4.63. In order to illustrate our approach, additional features (i.e. *c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18, c19, c20, c21, c22, c23, c24, c25, c26, c27, c28, c29, c30, c31, c32, c33, c34, c35, c36, c37, c38, c39, c40, c41, c42, c43, c44, c45, c46, c47, c48, c49, c50, c51, c52, c53, c54, c55, c56, c57, c58, c59, c60, c61, c62, c63, c64, c65, c66, c67,c68, c69, c70, c71, c72, c73, c74, c75, c76, c77, c78, c79, c80, c81, c82, c83*) and cross-tree constraints (i.e. *imp1,imp2, imp3, imp4, imp5, imp6, imp7, imp8, imp9, imp10, imp11, imp12, imp13, imp14, imp15, imp16, imp17, imp18, exc1, exc2, exc3, exc4, exc5, exc6, exc7)* are intentionally injected in the input *Computer Selection* FM to cause defects due to false-optional feature as shown in Figure 4.64.

Stage 1 - Transformation of FM into Predicate-based FMO:: The running example of *Computer Selection* FM (see Figure 4.63) is transformed into predicate-based FMO (see Figure 4.64) using transformation details explained in Section 4.2.

Stage 2 - Development and Applicability of FOL-based Rules: The rules discussed in Sub-section 4.3.3 are applied to the generated predicate-based FMO of *Computer Selection* FM (see Figure 4.65).

Stage 3 – Identification of Defects, their Causes and Providing Corrections: Figure 4.66 represents a snapshot of the results generated after applying all rules to *Computer*

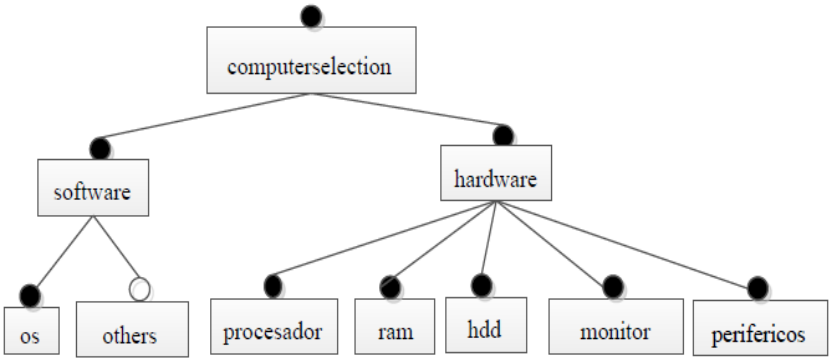


Figure 4.63: Computer Selection feature model from SPLOT repository.

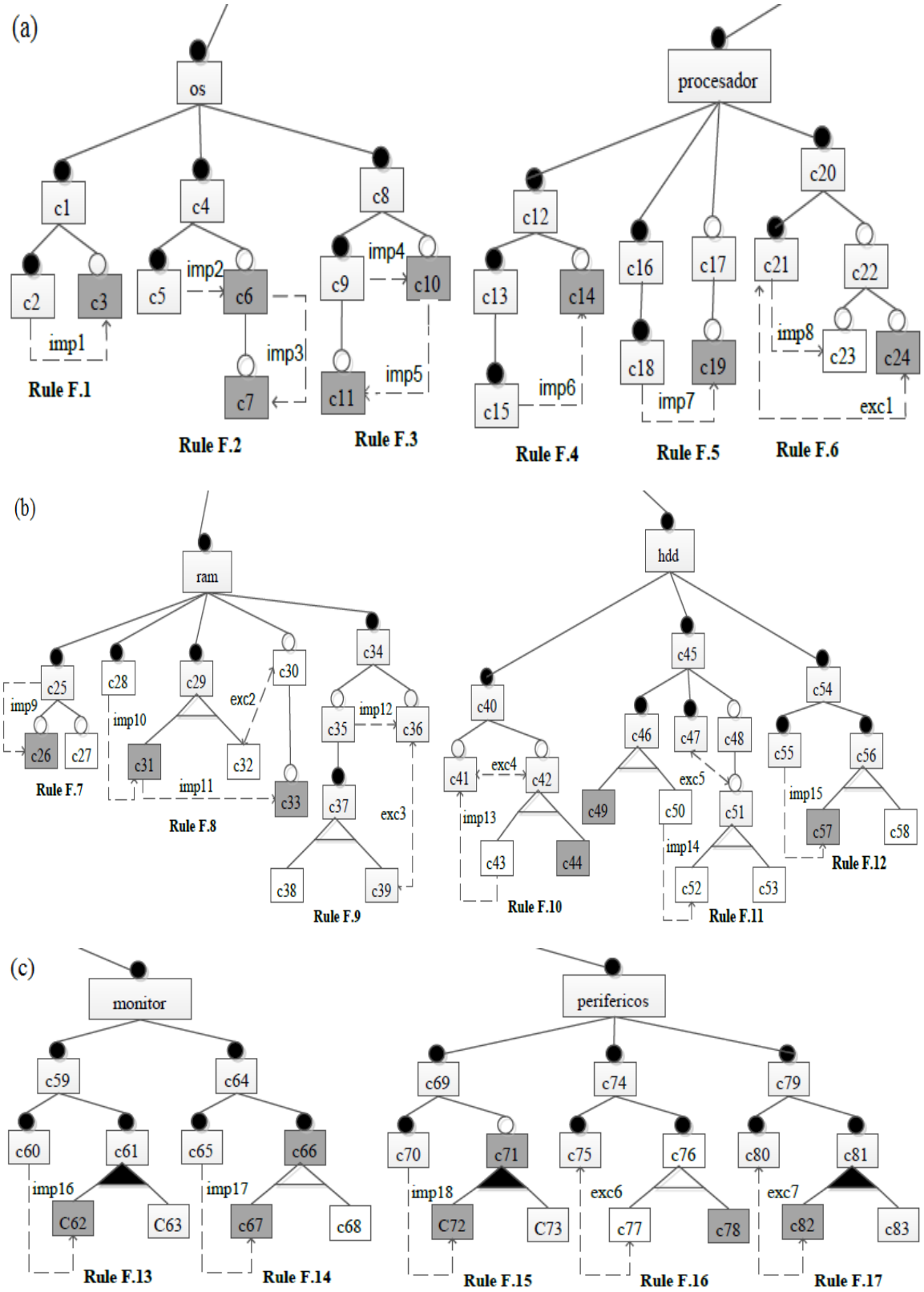


Figure 4.64: Computer Selection feature model where grey features represent defects due to false-optional feature (a) F.1-F.6, (b) F7-F12, and (c) F.13-F.17.

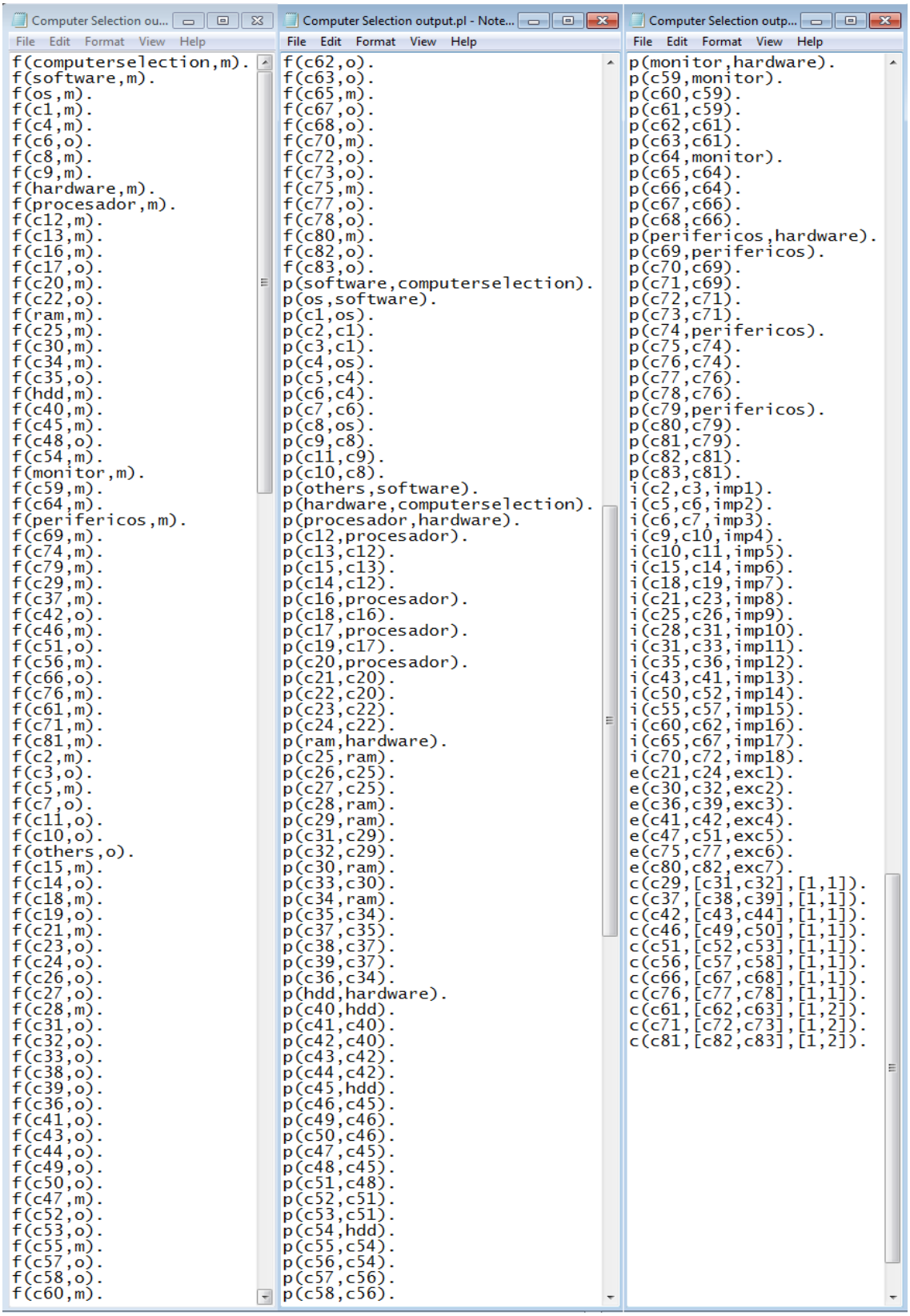


Figure 4.65: Feature model ontology of Computer Selection feature model.

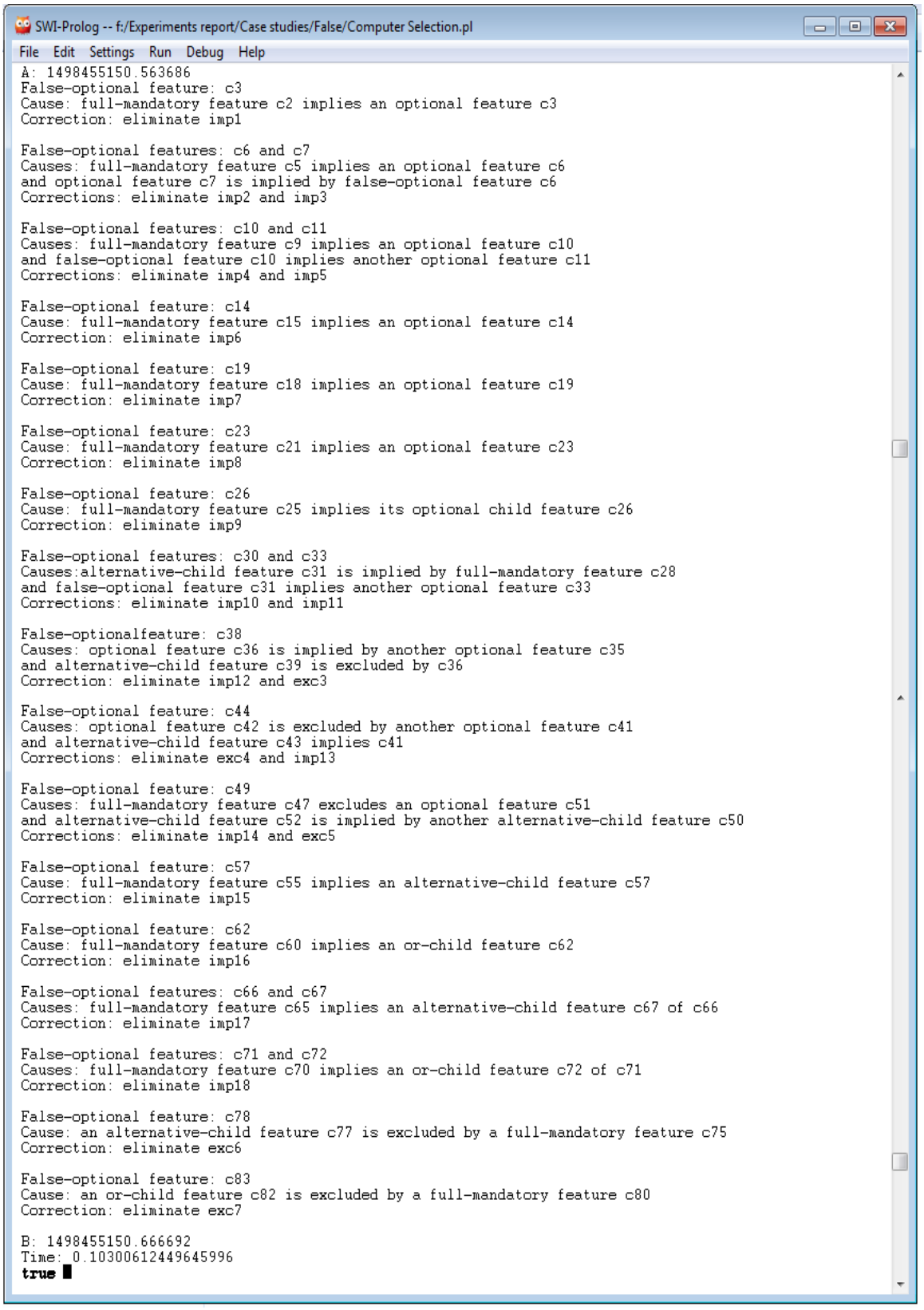


Figure 4.66: Results generated after applying all rules to Computer Selection feature model.

Selection FMO. The results comprise of identified defects due to false-optional feature along with their causes and corrections in a user-friendly natural language. The generated results can be further used by the PL developers to resolve defects due to false-optional feature, i.e. by eliminating cross-tree constraints involved in the cause of defects.

4.3.4 Inconsistency

Sub-section 3.2.4 provides a detailed discussion on each case of defect arising due to inconsistency. Following illustrates each case which includes facts in the form of FMO, rule and result.

Rule I.1

Facts: f(p,m). f(c1,m). f(c2,m). p(c1,p). p(c2,p). i(c1,c2,imp1). e(c2,c1,exc1).

rule1:- f(P,m), f(C1,m), f(C2,m), p(C1,P), p(C2,P), i(C1,C2,I1), e(C2,C1,E1), write('\nDefect: Inconsistency1\nCause: implication and exclusion between mandatory features '), write(C1), write(' and '), write(C2), write('\nCorrections: eliminate '), write(I1), write(' and '), write(E1).

Result: Figure 4.67 represents the result produced after implementing this rule to FMO.

```
Defect:Inconsistency1
Cause: implication and exclusion between mandatory featuresc1 and c2
Corrections: eliminate imp1 and exc1
```

Figure 4.67: Result of rule I.1.

Rule I.2

Facts: f(p,m). f(c1,m). f(c2,m). p(c1,p). p(c2,p). e(c1,c2,exc1).

rule2:- f(P,m), f(C1,m), f(C2,m), p(C1,P), p(C2,P), e(C1,C2,E1), write('\nDefect: : Inconsistency2\nCause: exclusion between mandatory features '), write(C1), write(' and '), write(C2), write('\nCorrection: eliminate '), write(E1).

Result: The generated result after applying this rule to FMO is shown in Figure 4.68.

```
Defect: Inconsistency2
Cause: exclusion between mandatory features c1 and c2
Correction: eliminate exc1
```

Figure 4.68: Result of rule I.2.

Rule I.3

Facts: f(p,m). f(c1,m). f(c2,m). f(c3,m). p(c1,p). p(c2,p). p(c3,c1). e(c2,c3,exc1).

rule3:- f(P,m), f(C1,m), f(C2,m), f(C3,m), p(C1,P), p(C2,P), p(C3,C1), e(C2,C3,E1), write('Defect: Inconsistency3\nCause: exclusion between mandatory features '), write(C2), write(' and '), write(C3) , write('\nCorrection: eliminate '), write(E1).

Result: Figure 4.69 shows the result obtained after implementing this rule to FMO.

```
Defect: Inconsistency3
Cause: exclusion between mandatory features c2 and c3
Correction: eliminate exc1
```

Figure 4.69: Result of rule I.3.

Rule I.4

Facts: f(p1,m). f(p2,m). f(c1,m). f(c2,m). p(c1,p1). p(c2,p2). e(c1,c2,exc1).

rule4:- f(P1,m), f(P2,m), f(C1,m), f(C2,m), p(C1,P1), p(C2,P2), e(C1,C2,E1), write('

Defect: Inconsistency4\nCause: exclusion between mandatory features '), write(C1), write(' and '), write(C2), write('\nCorrection: eliminate '), write(E1).

Result: The result produced after applying this rule to FMO is shown in Figure 4.70.

```
Defect: Inconsistency4
Cause: exclusion between mandatory features c1 and c2
Correction: eliminate exc1
```

Figure 4.70: Result of rule I.4.

Rule I.5

Facts: f(p,m). f(c1,m). f(c2,m). f(c3,m). p(c1,p). p(c2,p). p(c3,c1). e(c1,c2,exc1). i(c3,c2,imp1).

rule5:- f(P,m), f(C1,m), f(C2,m), f(C3,m), p(C1,P), p(C2,P), p(C3,C1), e(C1,C2,E1), i(C3,C2,I1), write('\nDefect: Inconsistency5\nCause: exclusion between mandatory features '), write(C1), write(' and '), write(C2), write('\nand mandatory feature '), write(C3), write(' implies '), write(C2), write('\nCorrections : eliminate '), write(E1), write(' and '), write(I1).

Result: Figure 4.71 represents the result produced after implementing this rule to FMO.

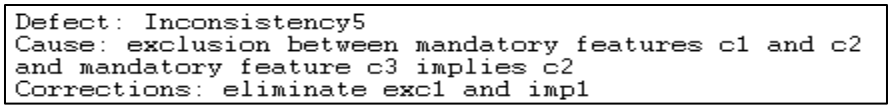


Figure 4.71: Result of rule I.5.

Rule I.6

Facts: f(p,m). f(c1,m). f(c2,m). f(c3,m). p(c1,p). p(c2,p). p(c3,c1). e(c1,c2,exc1). i(c3,c2,imp1).

rule6:- f(P,m), f(C1,m), f(C2,m), f(C3,m), f(C4,o), p(C1,P), p(C2,P), p(C3,C1), p(C4,C2), e(C1, C2,E1), i(C3,C4,I1), write('\nDefect: Inconsistency6\nCauses: exclusion between mandatory features '), write(C1), write(' and '), write(C2), write('\nand optional feature '), write(C4), write(' is implied by mandatory feature '), write(C3), write('\nCorrections : eliminate '), write(E1), write(' and '), write(I1).

Result: The generated result after applying this rule to FMO is shown in Figure 4.72.

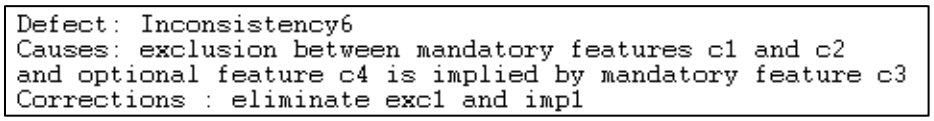


Figure 4.72: Result of rule I.6.

Rule I.7

Facts: f(p,m1). f(c1,o). f(c2,m). f(c3,m). p(c1,p). p(c2,p). p(c3,c1). i(c2,c1,imp1). e(c3,c2,exc1).

rule7:- f(P,m), f(C1,o), f(C2,m), f(C3,m), p(C1,P), p(C2,P), p(C3,C1), i(C2,C1,I1), e(C3,C2,E1), write('\nDefect: Inconsistency7\nCauses: optional feature '), write(C1), write(' is implied by mandatory feature '), write(C2), write('\nand exclusion between mandatory features '), write(C2), write(' and '), write(C3), write('\nCorrections : eliminate '), write(I1), write(' and '), write(E1).

Result: Figure 4.73 shows the result obtained after implementing this rule to FMO.

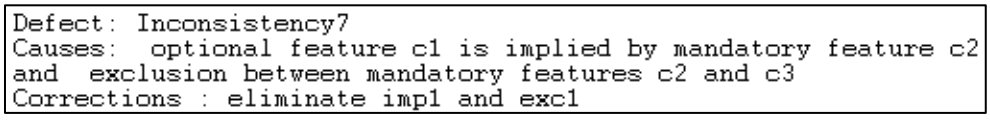


Figure 4.73: Result of rule I.7.

Rule I.8

Facts: f(p,m). f(c1,m). f(c2,o). f(c3,m). f(c4,m). p(c1,p). p(c2,p). p(c3,c1). p(c4,c2). i(c1,c2,imp1). e(c3,c4,exc1).

rule8:- f(P,m), f(C1,m), f(C2,o), f(C3,m), f(C4,m), p(C1,P), p(C2,P), p(C3,C1), p(C4,C2), i(C1,C2,I1), e(C3,C4,E1), write('\nDefect: Inconsistency8\nCauses: optional feature'), write(C2), write(' is implied by mandatory feature '), write(C1), write('\nand exclusion between mandatory features '), write(C3), write(' and '), write(C4), write('\nCorrections : eliminate '), write(I1), write(' and '), write(E1).

Result: The produced result after applying this rule to FMO is shown in Figure 4.74.

```
Defect: Inconsistency8
Causes: optional feature c2 is implied by mandatory feature c1
and exclusion between mandatory features c3 and c4
Corrections: eliminate imp1 and exc1
```

Figure 4.74: Result of rule I.8.

Rule I.9

Facts: f(p,m). f(c1,m). f(c2,o). f(c3,m). p(c1,p). p(c2,p). p(c3,c1). i(C1,C2,imp1). e(C3,C2,exc1).

rule9:- f(P,m), f(C1,m), f(C2,o), f(C3,m), p(C1,P), p(C2,P), p(C3,C1), i(C1,C2,I1), e(C3,C2,E1), write('\nDefect: Inconsistency9\nCauses: optional feature '), write(C2), write(' is implied by mandatory feature '), write(C1), write('\nand exclusion between mandatory features '), write(C2), write(' and '), write(C3), write('\nCorrections : eliminate '), write(I1), write(' and '), write(E1).

Result: Figure 4.75 represents the result produced after implementing this rule to FMO.

```
Defect: Inconsistency9
Causes: optional feature c2 is implied by mandatory feature c1
and  exclusion between mandatory features c2 and c3
Corrections: eliminate imp1 and exc1
```

Figure 4.75: Result of rule I.9.

Rule I.10

Facts: f(p,m). f(c1,m). f(c2,o). f(c3,m). f(c4,m). f(c5,m). p(c1,p). p(c2,p). p(c3,c2). p(c4,c2). p(c5,c4). i(c1,c3,imp1). e(c1,c5,exc1).

rule10:- f(P,m1), f(C1,m2), f(C2,o), f(C3,m3), f(C4,m4), f(C5,m5), p(C1,P), p(C2,P), p(C3,C2), p(C4,C2), p(C5,C4), i(C1,C3,I1), e(C5,C1,E1), write('\nDefect: inconsistency 10'), write('\nCauses: full-mandatory feature '), write(C1), write(' implies mandatory feature '), write(C3), write('\nand mandatory feature '), write(C5), write(' excludes '), write(C1), write('\nCorrections: eliminate '), write(I1), write(' and '), write(E1).

Result: The generated result after applying this rule to FMO is shown in Figure 4.76.

```
Defect: Inconsistency10
Causes: full-mandatory feature c1 implies mandatory feature c3
and mandatory feature c5 excludes c1
Corrections: eliminate imp1 and exc1
```

Figure 4.76: Result of rule I.10.

Rule I.11

Facts: f(p,m). f(c1,m). f(c2,o). f(c3,o). f(c4,o). p(c1,p). p(c2,p). c(c2,[c3,c4],[1,1]).

i(c1,c4,imp1). i(c4,c3,imp2).

rule11:- f(P,m), f(C1,m), f(C2,o), f(C3,o), f(C4,o), p(C1,P), p(C2,P), c(C2,[C3,C4],[1,1]), i(C1,C4,I1), i(C4,C3,I2), write('\nDefect: Inconsistency10'), write ('\nCauses: full-mandatory feature '), write(C1), write(' implies an alternative-child feature '), write(C4), write('\nand another alternative-child feature '), write(C3), write(' is implied by '), write(C4), write('\nCorrections: eliminate '), write(I1), write(' and '), write (I2).

Result: Figure 4.77 shows the result obtained after implementing this rule to FMO.

```
Defect: Inconsistency11
Causes: full-mandatory feature c1 implies an alternative-child feature c4
and another alternative-child feature c3 is implied by c4
Corrections: eliminate imp1 and imp2
```

Figure 4.77: Result of rule I.11.

Rule I.12

Facts: f(p,m). f(c1,o). f(c2,o). c(p,[c1,c2],[1,1]). i(c1,c2,imp1).

rule12:- f(P,m), f(C1,o), f(C2,o), c(P,[C1,C2],[1,1]), i(C1,C2,I1), write('\nDefect: Inconsistency12\nCause: implication between alternative-child features '), write(C1), write(' and '), write(C2), write('\nCorrection: eliminate '), write(I1).

Result: The produced result after applying this rule to FMO is shown in Figure 4.78.

```
Defect: Inconsistency12
Cause: implication between alternative-child features c1 and c2
Correction: eliminate imp1
```

Figure 4.78: Result of rule I.12.

Implementation to the running example of Address Book FM: The *Address Book* available in FM repository of SPLOT is used as a running example to explain the implementation of our approach to handle defects due to inconsistency along with their causes and corrections. The graphical representation of the corresponding model is shown in Figure 4.79. In order to illustrate our approach, additional features (i.e. *d1*, *d2*, *d3*, *d4*, *d5*, *d6*, *d7*, *d8*, *d9*, *d10*, *d11*, *d12*, *d13*, *d14*, *d15*, *d16*, *d17*, *d18*, *d19*, *d20*, *d21*, *d22*, *d23*, *d24*, *d25*, *d26*, *d27*, *d28*, *d29*, *d30*, *d31*, *d32*, *d33*, *d34*, *d35*, *d36*, *d37*, *d38*, *d39*, *d40*, *d41*, *d42*, *d43*, *d44*, *d45*, *d46*, *d47*, *d48*, *d49*) and cross-tree constraints (i.e. *imp1*, *imp2*, *imp3*, *imp4*, *imp5*, *imp6*, *imp7*, *imp8*, *imp9*, *imp10*, *exc1*, *exc2*, *exc3*, *exc4*, *exc5*, *exc6*, *exc7*, *exc8*, *exc9*, *exc10)* are intentionally injected in the input *Address Book* FM to cause defects due to inconsistency as shown in Figure 4.80.

Stage 1 - Transformation of FM into Predicate-based FMO: The running example of *Address Book* FM (see Figure 4.80) is transformed into predicate-based FMO (see Figure 4.81) using transformation details explained in Section 4.2.

Stage 2 - Development and Applicability of FOL-based Rules: The rules discussed in Sub-section 4.3.4 are applied to the generated predicate-based FMO of Address *Book* FM (see Figure 4.81).

Stage 3 – Identification of Defects, their Causes and Providing Corrections: Figure 4.82 represents a snapshot of the results generated after applying all rules to *Address*

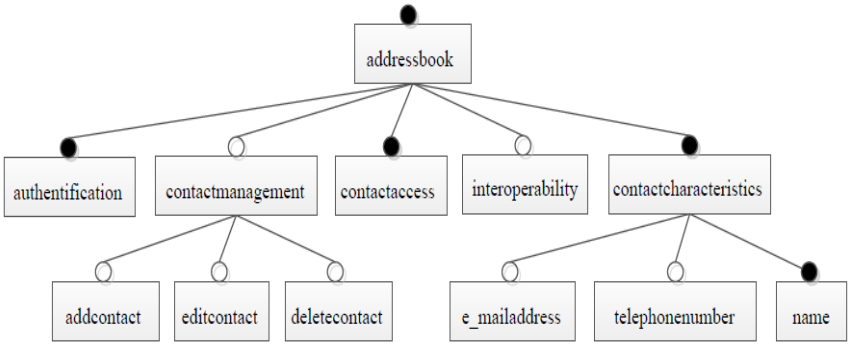


Figure 4.79: Address Book feature model from SPLOT repository.

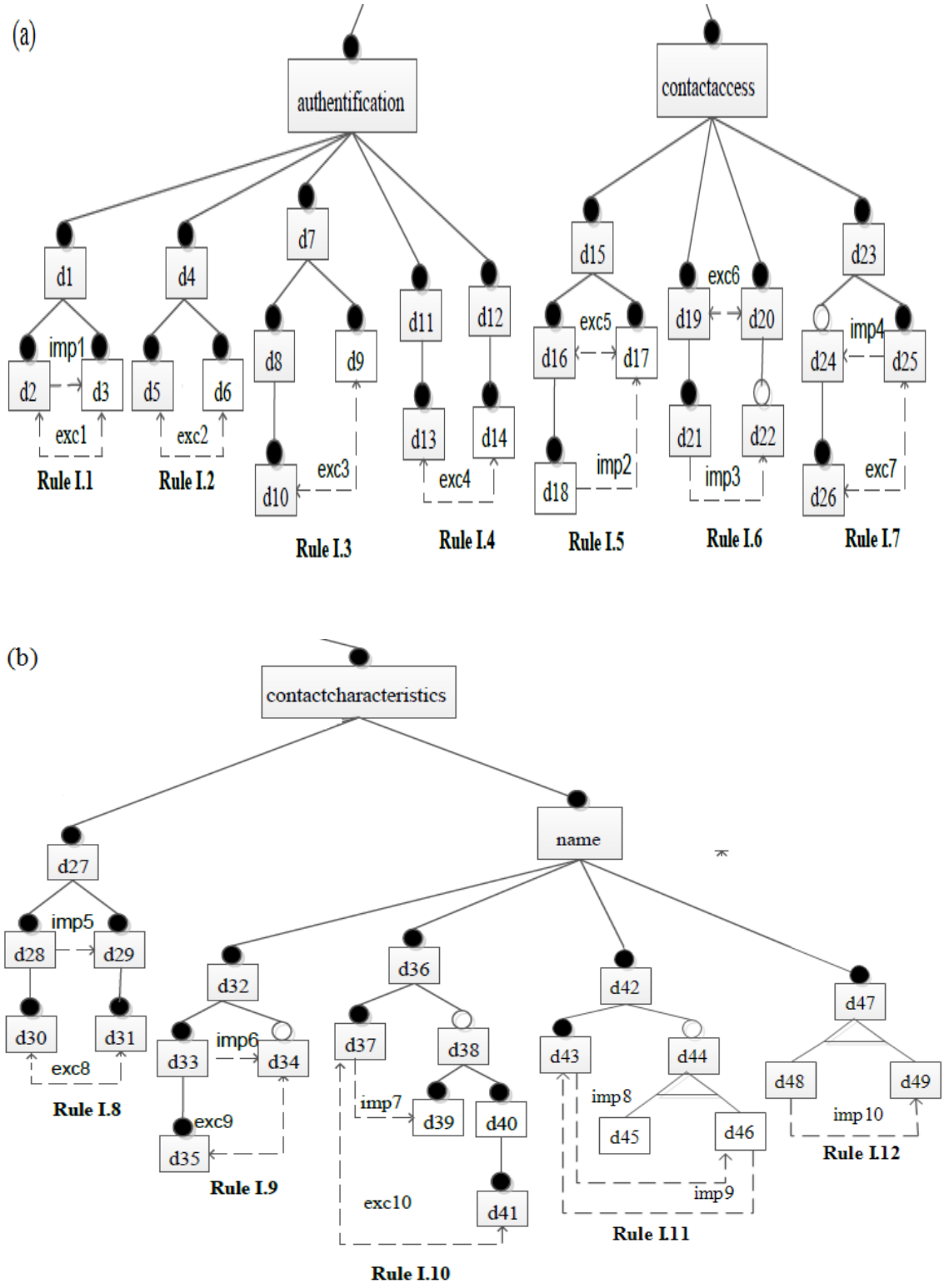


Figure 4.80: Address Book feature model representing defects due to inconsistency (a) I.1-I.7, and (b) I.8-I.12.

Address output.pl - Notepad

File Edit Format View Help

```
f(addressbook,m).
f(authentification,m).
f(d1,m).
f(d4,m).
f(d7,m).
f(d8,m).
f(d11,m).
f(d12,m).
f(contactmanagement,o).
f(contactaccess,m).
f(d15,m).
f(d16,m).
f(d19,m).
f(d20,m).
f(d23,m).
f(d24,o).
f(contactcharacteristics,m).
f(d27,m).
f(d28,m).
f(d29,o).
f(name,m).
f(d32,m).
f(d33,m).
f(d36,m).
f(d38,o).
f(d40,m).
f(d42,m).
f(d44,o).
f(d47,m).
f(d2,m).
f(d3,m).
f(d5,m).
f(d6,m).
f(d10,m).
f(d9,m).
f(d13,m).
f(d14,m).
f(addcontact,o).
f(edictcontact,o).
f(deletecontact,o).
f(d18,m).
f(d17,m).
f(d21,m).
f(d22,o).
f(d26,m).
f(d25,m).
f(e_mailaddress,o).
f(telephonenumber,o).
f(d30,m).
f(d31,m).
f(d35,m).
f(d34,o).
f(d37,m).
f(d39,m).
f(d41,m).
f(d43,m).
f(d45,o).
f(d46,o).
f(d48,o).
f(d49,o).
f(interoperability,o).
p(authentification,addressbook).
p(d1,authentification).
p(d2,d1).
p(d3,d1).
p(d4,authentification).
p(d5,d4).
p(d6,d4).
p(d7,authentification).
p(d8,d7).
p(d10,d8).
p(d9,d7).
p(d11,authentification).
p(d13,d11).
p(d12,authentification).
p(d14,d12).
p(contactmanagement,addressbook).
p(addcontact,contactmanagement).
p(edictcontact,contactmanagement).
```

Address output.pl - Notepad

File Edit Format View Help

```
p(deletecontact,contactmanagement).
p(contactaccess,addressbook).
p(d15,contactaccess).
p(d16,d15).
p(d18,d16).
p(d17,d15).
p(d19,contactaccess).
p(d21,d19).
p(d20,contactaccess).
p(d22,d20).
p(d23,contactaccess).
p(d24,d23).
p(d26,d24).
p(d25,d23).
p(contactcharacteristics,addressbook).
p(e_mailaddress,contactcharacteristics).
p(telephonenumber,contactcharacteristics).
p(d27,contactcharacteristics).
p(d28,d27).
p(d30,d28).
p(d29,d27).
p(d31,d29).
p(name,contactcharacteristics).
p(d32,name).
p(d33,d32).
p(d35,d33).
p(d34,d32).
p(d36,name).
p(d37,d36).
p(d38,d36).
p(d39,d38).
p(d40,d38).
p(d41,d40).
p(d42,name).
p(d43,d42).
p(d44,d42).
p(d45,d44).
p(d46,d44).
p(d47,name).
p(d48,d47).
p(d49,d47).
p(interoperability,addressbook).
i(d2,d3,imp1).
i(d18,d17,imp2).
i(d21,d22,imp3).
i(d25,d24,imp4).
i(d28,d29,imp5).
i(d33,d34,imp6).
i(d37,d39,imp7).
i(d43,d46,imp8).
i(d46,d45,imp9).
i(d48,d49,imp10).
e(d3,d2,exc1).
e(d5,d6,exc2).
e(d9,d10,exc3).
e(d13,d14,exc4).
e(d16,d17,exc5).
e(d19,d20,exc6).
e(d26,d25,exc7).
e(d30,d31,exc8).
e(d35,d34,exc9).
e(d41,d37,exc10).
c(d44,[d45,d46],[1,1]).
c(d47,[d48,d49],[1,1]).
```

Figure 4.81: Feature model ontology of Address Book feature model.

Book FMO. The results comprise of identified defects due to inconsistency along with their causes and corrections in a user-friendly natural language. The generated results can be further used by the PL developers to resolve defects due to inconsistency, i.e. by eliminating cross-tree constraints involved in the cause of defects.

Figure 4.82: Results generated after applying all rules to Address Book feature model.

4.3.5 Wrong Cardinality

Sub-section 3.2.5 provides a detailed discussion on each case of defect arising due to . Following illustrates each case which includes facts in the form of FMO, rule and result.

Rule W.1

Facts: f(p,m). f(c1,o). f(c2,o). f(c3,o). c(p,[c1,c2,c3],[4,6]).

rule1:- f(P,m), f(C1,o), f(C2,o), f(C3,o), c(P,[C1,C2,C3],[4,6]), write('Defect: Wrong cardinality 1\nCause: wrong cardinalities <4..6>'), write('\nCorrections: min(4) must be inferior to 3 and max(6) must be inferior or equal to 3').

Result: Figure 4.83 represents the result produced after implementing this rule to FMO.

```
Defect: Wrong cardinality1
Cause: wrong cardinalities <4..6>
Corrections: min(4) must be inferior to 3 and max(6) must be inferior or equal to 3
```

Figure 4.83: Result of rule W.1.

Rule W.2

Facts: f(p,m). f(c1,o). f(c2,o). f(c3,o). c(p,[c1,c2,c3],[2,3]). e(c1,c2,exc1). e(c2,c3,exc2). e(c3,c1,exc3).

rule2:- f(P,m), f(C1,o), f(C2,o), f(C3,o), c(P,[C1,C2,C3],[2,3]), e(C1,C2,E1), e(C2,C3,E2), e(C3, C1,E3), write('Defect: Wrong cardinality2\nCauses: exclusions from '), write(C1) , write(' to '), write(C2), write(C2), write(' to '), write(C3), write(' and '), Write(C3), write (' to '), write(C1), write('\nCorrections: eliminate '), write(E1), write(', '),write(E2), write(' and '), write(E3).

Result: The generated result after applying this rule to FMO is shown in Figure 4.84.

```
Defect: Wrong cardinality2
Causes: exclusions from c1 to c2, c2 to c3 and c3 to c1
Corrections: eliminate exc1, exc2 and exc3
```

Figure 4.84: Result of rule W.2.

Rule W.3

Facts: f(p,m). f(c1,o). f(c2,o). f(c3,o). c(p,[c1,c2,c3],[1,3]). i(c1,c2,imp1). i(c2,c3,imp2).

rule3:- f(P,m), f(C1,o), f(C2,o), f(C3,o), c(P,[C1,C2,C3],[1,3]), i(C1,C2,I1), i(C2,C3,I2),

write('Defect: Wrong cardinality3\nCauses: '), write(C2), write(' is implied by '), write(C1), write (' and '), write(C3), write(' is implied by '), write(C2), write('\nCorrections: eliminate '), write(I1) , write(' and '),write(I2).

Result: Figure 4.85 shows the result obtained after implementing this rule to FMO.

```
Defect: Wrong cardinality3
Causes: c2 is implied by c1 and c3 is implied by c2
Corrections: eliminate imp1 and imp2
```

Figure 4.85: Result of rule W.3.

Rule W.4

Facts: f(p,m). f(c1,o). f(c2,o). f(c3,o). f(c4,o). c(p,[c1,c2,c3,c4],[1,4]). e(c2,c4,exc1).

rule4:-f(P,m), f(C1,o), f(C2,o), f(C3,o), f(C4,o), c(P,[C1,C2,C3,C4],[1,4]), e(C2,C4,E1), write(' Defect: Wrong cardinality4\nCause: mutual exclusion between '), write(C2), write(' and '), write(C4), write(' does not allow the selection of four child features '), write('\nCorrection: eliminate '), write(E1).

Result: The produced result after applying this rule to FMO is shown in Figure 4.86.

```
Defect: Wrong cardinality4
Cause: mutual exclusion between c2 and c4 does not allow the selection of four child features
Correction: eliminate exc1
```

Figure 4.86: Result of rule W.4.

Rule W.5

Facts: f(p,m). f(c1,o1). f(c2,o2). c(p,[c1,c2],[1,3]).

rule5:- f(P,m), f(C1,o), f(C2,o), c(P,[C1,C2],[1,3]), write('Defect: Wrong cardinality5\n Cause: maximum (max) cardinality value 3 '), write('\nCorrection: max(3) must be inferior or equal to 2').

Result: Figure 4.87 represents the result produced after implementing this rule to FMO.

```
Defect: Wrong cardinality5
Cause: maximum (max) cardinality value 3
Correction: max(3) must be inferior or equal to 2
```

Figure 4.87: Result of rule W.5.

Rule W.6

Facts: f(p,m). f(c1,o). f(c2,o). f(c3,o). c(p,[c1,c2,c3],[1,2]). i(c1,c2,imp1). i(c2,c3,imp2). i(c3,c1,imp3).

rule6:- f(P,m), f(C1,o), f(C2,o), f(C3,o), c(P,[C1,C2,C3],[1,2]), i(C1,C2,I1), i(C2,C3,I2), i(C3,C1,I3), write('Defect: Wrong cardinality6\nCauses: maximum cardinality 2 cannot be attained due to implications from '), write(C1), write(' to '), write(C2), write(', '), write(C2), write(' to '), write(C3), write(' and '), write(C3), write(' to '), write(C1), write('\nCorrections: eliminate '), write(I1), write(', '),write(I2), write(' and '),write(I3).

Result: The generated result after applying this rule to FMO is shown in Figure 4.88.

```
Defect: Wrong cardinality6
Causes: maximum cardinality 2 cannot be attained due to implications from c1 to c2, c2 to c3 and c3 to c1
Corrections: eliminate imp1, imp2 and imp3
```

Figure 4.88: Result of rule W.6.

Rule W.7

Facts: f(p,m). f(c1,o). f(c2,o). f(c3,o). c(p,[c1,c2,c3],[1,2]). i(c1,c2,imp1). i(c1,c3,imp2).

rule7:-f(P,m), f(C1,o), f(C2,o), f(C3,o), c(P,[C1,C2,C3],[1,2]), i(C1,C2,I1), i(C1,C3,I2), write('Defect: Wrong cardinality7\nCauses: maximum cardinality 2 cannot be attained due to implications from '), write(C1), write(' to '), write(C2), write(' and '), write(C1), write(' to '), write(C3), write('\nCorrections: eliminate '), write(I1), write(' and '), write(I2).

Result: Figure 4.89 shows the result obtained after implementing this rule to FMO.

```
Defect: Wrong cardinality7
Causes: maximum cardinality 2 cannot be attained due to implications from c1 to c2 and c1 to c3
Corrections: eliminate imp1 and imp2
```

Figure 4.89: Result of rule W.7.

Rule W.8

Facts: f(p,m). f(c1,o). f(c2,o). f(c3,o). c(p,[c1,c2,c3],[3,-1]).

rule8:- f(P,m), f(C1,o), f(C2,o), f(C3,o), c(P,[C1,C2,C3],[3,-1]), write('Defect: Wrong cardinality8\nCause: the group cardinality <3..-1>'), write('\nCorrections: min must be

inferior to max,\n min and max values must be ordered in an incremental manner as well as include non-negative values').

Result: The produced result after applying this rule to FMO is shown in Figure 4.90.

```
Defect: Wrong cardinality8
Cause: the group cardinality <3..-1>
Corrections: min must be inferior to max,
 min and max values must be ordered in an incremental manner as well as include non-negative values
```

Figure 4.90: Result of rule W.8.

Rule W.9

Facts: f(p,m). f(c1,o). f(c2,o). f(c3,o). f(c4,o). f(c5,o). c(p,[c1,c2,c3,c4,c5],[1,1]).

rule9:- f(P,m), f(C1,o), f(C2,o), f(C3,o), f(C4,o), f(C5,o), c(P,[C1,C2,C3,C4,C5],[1,1]), write('Defect: Wrong cardinality9\nCause: the group cardinality <1..1> as the number of selected child features is 2 which is superior to the max value 1 '), write('\nCorrection: the number of selected features must be superior to min and inferior to max').

Result: Figure 4.91 represents the result produced after implementing this rule to FMO.

```
Defect: Wrong cardinality9
Cause: the group cardinality <1..1> as the number of selected child features is 2 which is superior to the max value 1
Correction: the number of selected features must be superior to min and inferior to max
```

Figure 4.91: Result of rule W.9.

Implementation to the running example of RE Process FM: The *RE Process* available in FM repository of SPLOT is used as a running example to explain the implementation of our approach to handle defects due to wrong cardinality along with their causes and corrections. The graphical representation of the corresponding model is shown in Figure 4.92. In order to illustrate our approach, additional features (i.e. *e1*, *e2*, *e3*, *e4*, *e5*, *e6*, *e7*, *e8*, *e9*, *e10*, *e11*, *e12*, *e13*, *e14*, *e15*, *e16*, *e17*, *e18*, *e19*, *e20*, *e21*, *e22*, *e23*, *e24*, *e25*, *e26*, *e27*, *e28*, *e29*, *e30*, *e31*, *e32*, *e33*, *e34*, *e35*, *e36*, *e37*, *e38*) and cross-tree constraints (i.e. *imp1*, *imp2*, *imp3*, *imp4*, *imp5*, *imp6*, *imp7*, *exc1*, *exc2*, *exc3*, *exc4)* are intentionally injected in the input *RE Process*FM as shown in Figure 4.93.

Stage 1 - Transformation of FM into Predicate-based FMO: The running example of *RE Process* FM (see Figure 4.93) is transformed into predicate-based FMO (see Figure

4.94) using transformation details explained in Section 4.2. In case of defects due to wrong cardinality, cardinalities are explicitly introduced in the generated FMO to cause defects. As, FeatureIDE does not support FMs with wrong cardinality.

Stage 2 - Development and Applicability of FOL-based Rules: The rules discussed in Sub-section 4.3.5 are applied to the generated predicate-based FMO of *RE Process*FM (see Figure 4.94).

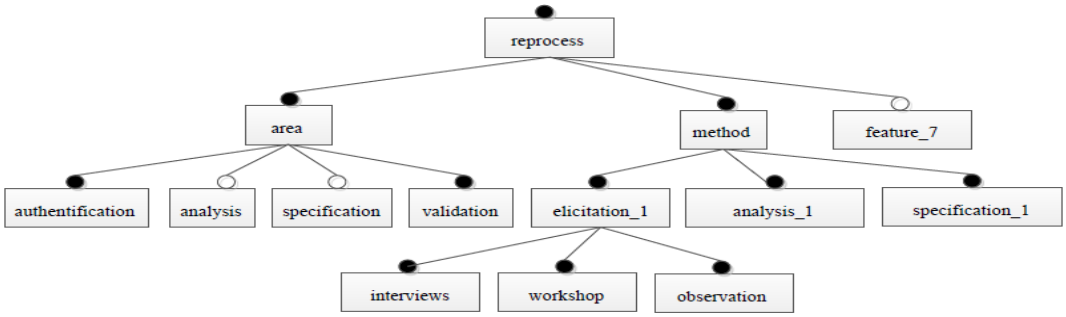


Figure 4.92: RE Process feature model from SPLOT repository.

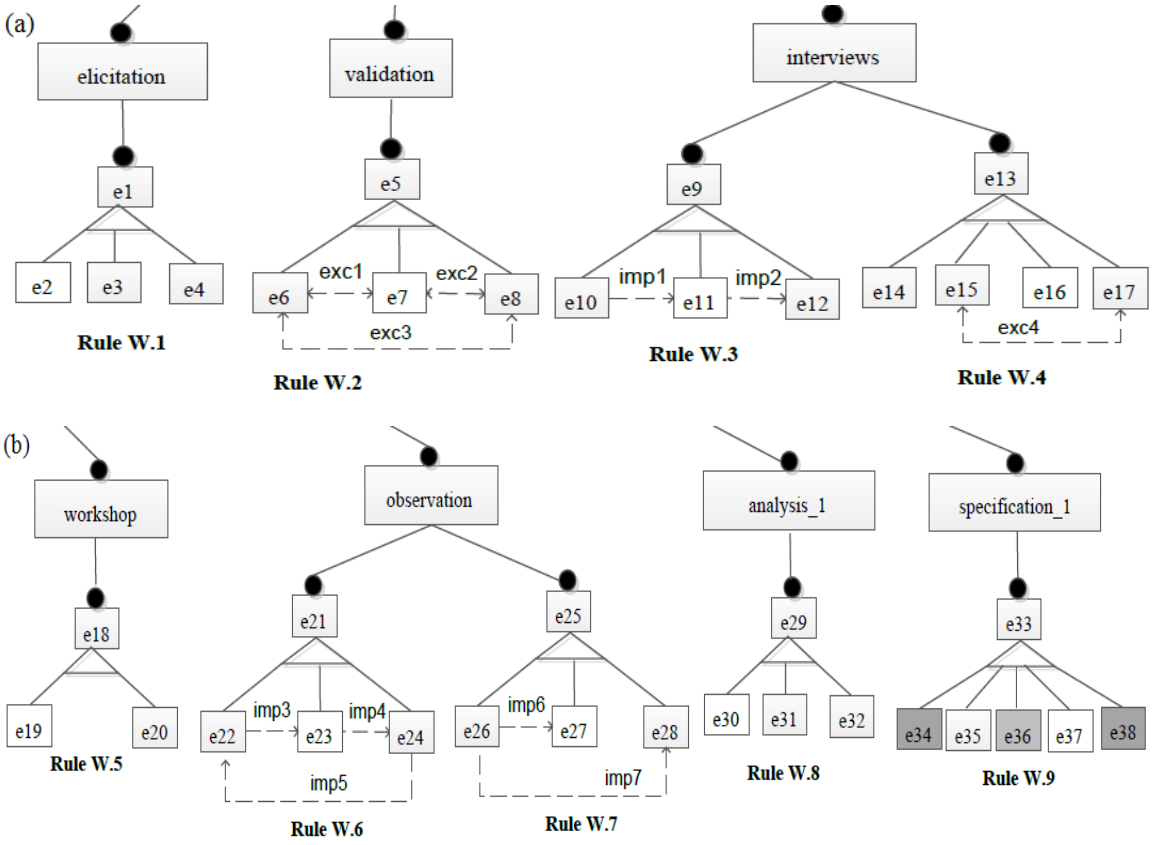


Figure 4.93: RE Process feature model representing defects due to wrong cardinality (a) W.1-W.4, and (b) W.5-W9.

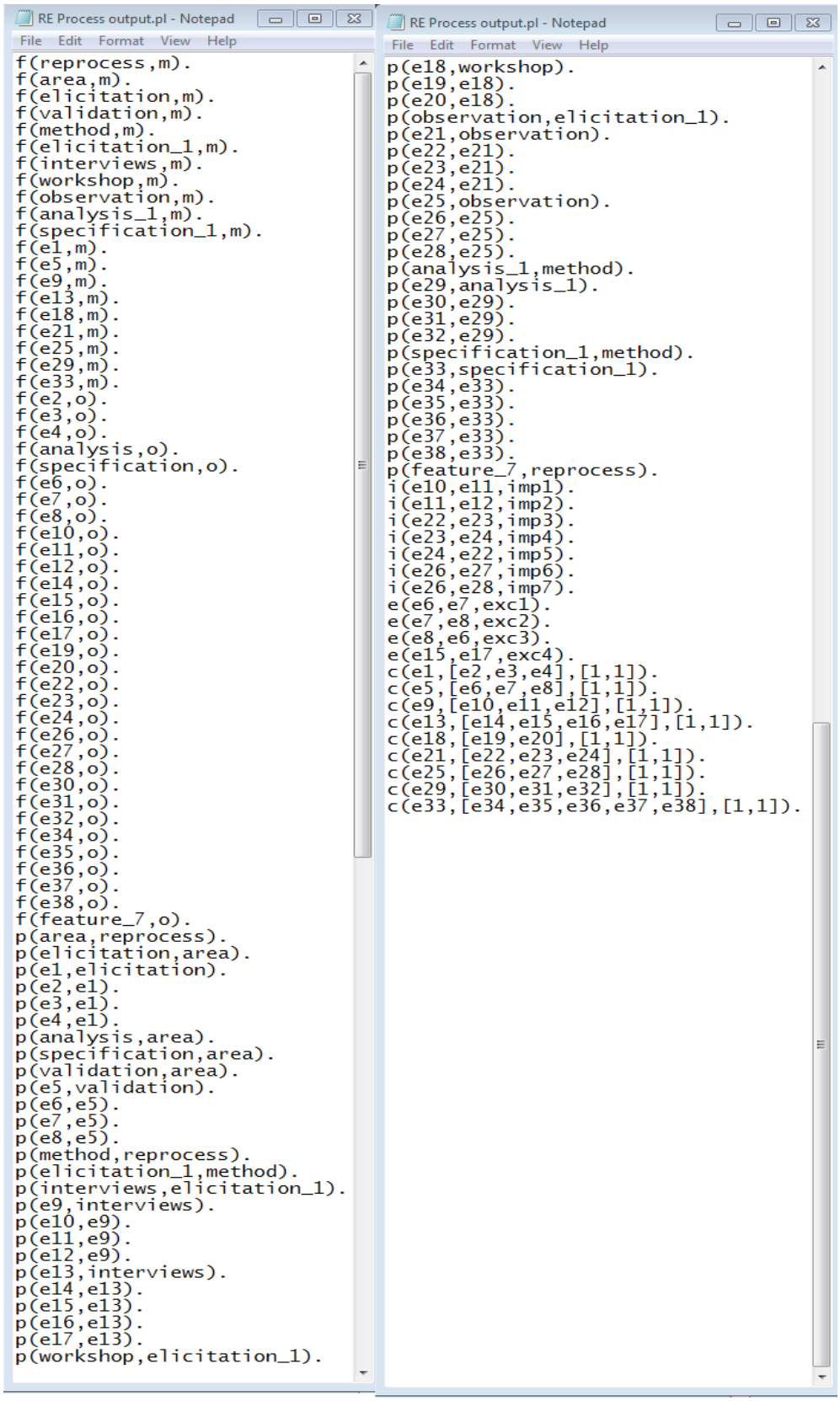


Figure 4.94: Feature model ontology of RE Process feature model.

Stage 3 – Identification of Defects, their Causes and Providing Corrections: Figure 4.95 represents a snapshot of the results generated after applying all rules to RE Process FMO. The results comprise of identified defects due to wrong cardinality along with their causes and corrections in a user-friendly natural language. The generated results can be further used by the PL developers to resolve defects due to wrong cardinality, i.e. by eliminating cross-tree constraints involved in the cause of defects.

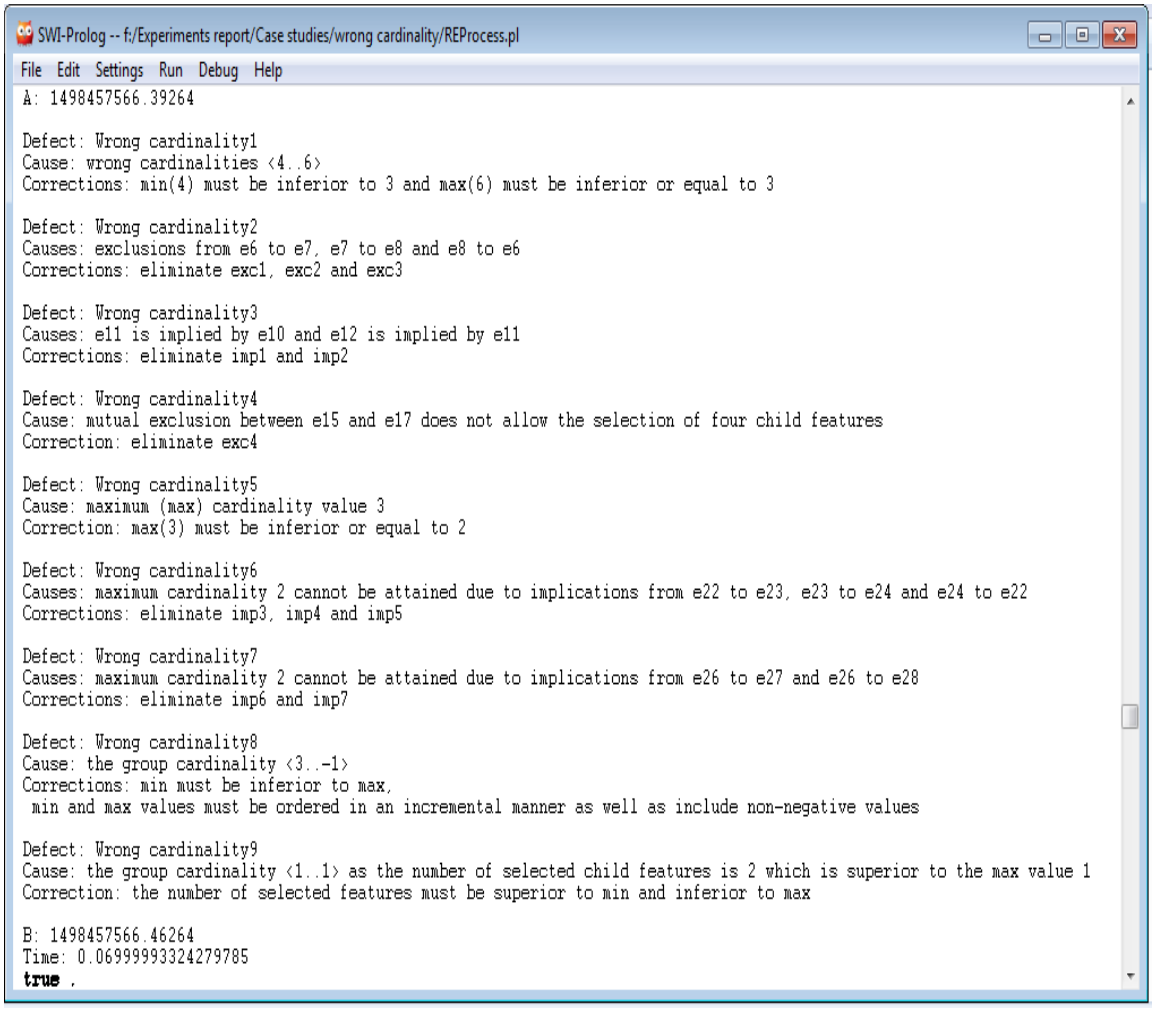


Figure 4.95: Results generated after applying all rules to RE Process feature model.

4.4 Summary

This chapter presented the implementation details of the proposed framework. It explained the methodology followed by the proposed approach. The two-step model transformation of FM to FMO is explained using a running example of *E-commerce*

system1 FM from SPLOT repository, (i) SPLOT Format to the FeatureIDE Format, and (ii) FeatureIDE Format to FMO. The FOL-based rules are developed and applied to deal with FM defects in SPL where each rule includes facts, rule and result. The result includes identified defect with their causes and corrections in a user-friendly natural language. Further, we implemented the proposed framework for all defects using different real-world FMs from SPLOT repository.

CHAPTER 5: Evaluating the Proposed Framework

This chapter presents the results of experiments carried out to evaluate the proposed approach contributing to objective 4. The resources and environment for the evaluation of proposed approach used in the experiments are given in Sections 5.1 and 5.2. The proposed framework is evaluated using accuracy, computational scalability, and performance analysis. The proposed rules are evaluated using completeness, consistency and consistency gain of the set of rules, and minimalism of the set of rules in Section 5.3. The proposed approach is compared with existing approaches in Section 5.5. Threats to validity of the proposed approach are discussed in Section 5.6. Finally, Section 5.7 summarizes the chapter.

5.1 Experimental Environment

The environment for the evaluation of the approach includes a dell workstation T-5600 with Windows 7 Professional N of 64 bits, processor Intel(R) Xeon(R) CPU E5-2650 @2.60 GHz, RAM memory of 8.00 GB 1600MHz, HDD-SATA 500GB @7200RPM, online SPLOT repository, Eclipse Luna SR2 (5.5.2), FeatureIDE 2.7.4 and SWI-Prolog (Version 7.4.2, 64 bits).

5.2 Benchmarks

A corpus of 108 models (99 models from SPLOT repository and 9 models generated using FeatureIDE tool) was used as a benchmark to evaluate the proposed approach.

5.2.1 Feature Models from SPLOT Repository

The models with a varied range of features available in SPLOT repository were considered for the evaluation of the proposed approach. We used 91 models ranging from 10 to 451 features, cross-tree constraints ratio (CTCR) (a percentage of features in the model that are participating in cross-tree constraints) varies between 0 to 93% along with cross-tree constraints clause density (clause density is defined as the density of clauses in the cross-tree constraints) values varies from 0.3 to 4.4 as shown in Table 5.1. The evaluated FMs along with details about their names, number of features (NOF), CTCR (in percentage), clause density (CD) and a description of 91 of these models as given in

SPLOT repository, sorted according to the number of features in the FM are listed in Table 5.1, where N/A represents models with no clause density.

Few models include more number of features when compared with the original FM details given in the repository. For example, *Quality Attributes Functionalities* FM has 77 features instead of 72, *urna_eletronica* FM has 74 features instead of 73, *Speech Recognition* FM has 77 features instead of 75, *Webmail* FM has 82 features instead of 81, *FrameWorkProdemge* FM has 88 features instead of 87, *subseaControlSystem* FM has 145 features instead of 142, *ubuntu1410_* FM has 263 features instead of 261, *Ubuntu12_04* FM has 267 features instead of 263, *Electronic Shopping* FM has 291 features instead of 290, *windows70* FM has 335 features instead of 329 and *windows80* FM has 461 features instead of 451. All these models are selected in order to consider each possible range of features in SPLOT for the validation of the proposed approach. Some of the models have redundant features, which have been renamed as FeatureIDE tool doesn't support a feature that appears twice.

Table 5.1: Overview of evaluated 91 FMs taken from the SPLOT repository.

	Feature models from SPLOT repository	NOF	CTCR (in %)	CD	Description
1	Isolation	10	0%	N/A	Featuremodel
2	CIMS PL	10	0%	N/A	CIMS Product Line
3	Computer Selection	10	0%	N/A	Tests of computer selection system.
4	UPL	11	0%	N/A	Unified Process Line
5	Address Book	12	0%	N/A	Address Book
6	E-commerce system1	12	0%	N/A	E-commerce system
7	Tata Cars	13	0%	N/A	Representing different models of Tata Cars
8	RE Process	14	0%	N/A	RE Process
9	Car demo	14	0%	N/A	Car model
10	Metadata	15	0%	N/A	Metadata
11	Notebook	16	0%	N/A	estructura notebook
12	AudioPlayer	17	0%	N/A	http://en.wikipedia.org/wiki/Comparison_of_audio_player_software
13	Computers	18	0%	N/A	Computers
14	Card Product	20	0%	N/A	Features of Credit Cards
15	Toko	21	0%	N/A	Toko
16	GymManager	22	0%	N/A	Modelo de features para linha de produto de gerenciadores de academias
17	e-Event	23	0%	N/A	Event Management System
18	B2B Website	24	0%	N/A	product display pages
19	eventos2	25	0%	N/A	Parcial2
20	HelpSystem_1	26	0%	N/A	Help System using sensor
21	WindFarm	27	0%	N/A	A simplified wind farm ecosystem.

22	Trabalho Topicos	28	0%	N/A	É um modelo de feature criado para aprendizado
23	ATM Software	29	0%	N/A	Describe model for ATM software
24	SAP Business Suite	30	0%	N/A	Descripcion del ERP de SAP
25	KDDI	31	0%	N/A	Sa
26	RaaS	32	0%	N/A	RaaS case study
27	Pricing	33	0%	N/A	Pricing of the cloud computing
28	MyHolidays	34	0%	N/A	MyHolidays
29	bCMSContext	35	0%	N/A	car crash crisis management system context
30	Tankwar	36	0%	N/A	Game
31	Bike	37	0%	N/A	bike product line
32	ProductosCredito	38	26%	0.7	Diagrama de features de productos de credito
33	KernelLinuxFM	39	0%	N/A	a FM based on Kernel Linux to be extended by VERONTO
34	WebCollaboration	40	0%	N/A	feature model for web collaborative applications
35	Security	41	0%	N/A	Security
36	DBdesign	42	0%	N/A	Describe the variability across THE DATABASE LIFE-CYCLE
37	Adempiere	43	0%	N/A	Descripcion SE Adempiere
38	E-commerce	44	15%	0.6	sple feature model
39	Computadores	45	28%	0.9	Modelo para armar computadores
40	GerenciaLojaVirtual	46	0%	N/A	TEES
41	Market Place	47	0%	N/A	linea de producto para generacion de market place personalizados
42	PGE-FeatureModel_V1	48	0%	N/A	Características Básicas
43	NOVASOFT	49	0%	N/A	SE NOVASOFT
44	EngAGe	50	0%	N/A	EngAGe - assessment configuration
45	BioFM	51	0%	N/A	bio RA project
46	Electronic Drum	52	0%	N/A	Yamaha electronic drums
47	E-commerce system	53	0%	N/A	E-commerce system
48	Thermonator	54	0%	N/A	Model for Thermonator product
49	e-banking-IN0980	55	9%	0.6	e-banking modeling
50	Berkley DB	56	21%	1	Berkley DB
51	Plone Meeting	57	0%	N/A	Feature model of the Plone Meeting
52	MISO4204_2014-2_Marketplace	58	3%	0.5	Marketplace
53	SPL SimulES, PnP	59	0%	N/A	Descreve as características da LP de jogos do SimulES e do PnP
54	Smart Home v2.2	60	6%	0.5	version 2.2. of Smart Home
55	E-science application	61	0%	N/A	E-science application
56	Discipline_Requirements	62	0%	N/A	Descrição de Atividades de Requisitos
57	FraSCati	63	55%	1.3	FraSCati
58	USECASE_Test	64	6%	0.5	a test mdoel
59	Economie moderne locale	65	0%	N/A	Analyse de Buurtpensioen et autres systems d economie locale
60	bCMS system	66	4%	0.7	describes a small distributed car crisis management system
61	HIS	67	11%	0.5	This model describes the features of a possible home integration system.
62	windows7	68	0%	N/A	windows7

63	Nákladný automobil	69	81%	4.4	Nákladný automobil
64	DATABASE_TOOLS	70	8%	0.3	FM_Exercise
65	Video Player	71	0%	N/A	FM of a Video Player's SPL
66	Quality Attributes Functionalities	72	0%	N/A	Variability of the functionalities required by Quality Attributes
67	urna_eletronica	73	0%	N/A	atividade reutilizacao de software
68	Speech Recognition	75	0%	N/A	fm do processo de reconhecimento de fala
69	PhotoSharing	76	0%	N/A	Feature model for Photo Sharing Domain
70	Windows	77	0%	N/A	test
71	FISH	80	2%	0.5	Management FISH
72	Webmail	81	0%	N/A	Feature model of a webmail application
73	FrameWorkProdemge	87	0%	N/A	Este feature model represnta o framework prodemge
74	ModelTransformation	88	0%	N/A	ModelTransformation
75	Coche ecologico	94	4%	0.5	Crear modelos de coches ecológicos
76	SmartTV	96	0%	N/A	smrttv
77	Xtext	137	1%	0.5	xtext feature model
78	Total Informatica	140	0%	N/A	LPS - UFMG - DCC
79	subseaControlSystem	142	18%	0.5	subseaCS
80	BattleofTanks	144	0%	N/A	BattleofTanks
81	FM_Test	168	28%	0.9	Test Model
82	Printers	172	0%	N/A	feature model
83	BankingSoftware	176	2%	0.8	This feature model has most of the banking related features.
84	android60	179	14%	1	Feature Model for Android6 Operating System
85	android510	210	18%	1	Feature model of Android5.1 operating system
86	ubuntu1410_	261	27%	1	accessibility options provided by Ubuntu1410 operating systems
87	Ubuntu12_04	263	31%	1	Accessibility options provided by Ubuntu1204 operating systems
88	Electronic Shopping	290	11%	0.6	This feature model models a B2C system with fixed priced products
89	windows70	329	33%	1	Accessibility options provided by Windows7 operating systems
90	A Model for Decision-making for Investments on Enterprise Information Systems	366	93%	0.6	Implications on EIS investment decisions
91	windows80	451	30%	1	Accessibility options provided by windows8 operating systems

NOF= Number of features, CD= Clause Density

5.2.2 Automatically-Generated Feature Models

Automatically-generated FMs available in the form of 3-CNF Feature Models (3-CNF-FMs) in SPLOT repository (considered as the toughest benchmark in SPL) and FeatureIDE tool were used as real-world large-sized FMs are difficult to acquire. Although such large FMs exist as a part of commercial projects with restricted access to their resources and these FMs are not publicly available. Moreover, industrial FMs are

always confidential. Thus, automatically-generated FMs were used. These models are used as standard benchmarks for the SPL community.

(i) 3-CNF-FMs from SPLOT repository: A 3-CNF-FM comprises of a feature tree with a set of cross-tree constraints represented using a single random 3-CNF formula. We have used 8 automatically-generated 3-CNF-FMs available in SPLOT repository ranging from 500 to 10,000 features. For these models, 3-CNF variable varies between 50 and 1000, and cross-tree constraints clause density varies from 0.1 to 1. Table 5.2 presents all the 8 automatically-generated 3-CNF-FMs along with the name of 3-CNF FM, the number of features, 3-CNF variables and clause density sorted according to the number of features in the FM. Each generated FM includes each type of optional, mandatory, exclusive-OR (<0..1> group cardinality) and inclusive-OR (<0..n> group cardinality) feature with equal probability. In the feature tree, the branching factor (i.e. number of child features per parent feature) varied from 1 to 6. Using a subset of features in the feature tree, the cross-tree constraints were produced as a single random 3-CNF formula. The performance and scalability of the proposed approach were tested using the powerful benchmark of 3-CNF-FMs. More details concerning these automatically-generated FMs can be found in online SPLOT repository.

Table 5.2: Automatically-generated 3-CNF FMs taken from the SPLOT repository.

	Automatically-Generated Feature Models from SPLOT repository (3-CNF-FMs)	Number of Features	3-CNF Variables	3-CNF Clause Density
1	SPLOT-3CNF-FM-500-50-1.00-SAT-1	500	50	1
2	SPLOT-3CNF-FM-500-50-1.00-UNSAT-1	500	50	1
3	SPLOT-3CNF-FM-1000-100-1.00-SAT-1	1000	100	1
4	SPLOT-3CNF-FM-1000-100-1.00-UNSAT-1	1000	100	1
5	SPLOT-3CNF-FM-2000-200-1.00-SAT-1	2000	200	0.5
6	SPLOT-3CNF-FM-2000-200-1.00-UNSAT-1	2000	200	1
7	SPLOT-3CNF-FM-5000-500-1.00-SAT-1	5000	500	0.3
8	SPLOT-3CNF-FM-10000-1000-1.00-SAT-1	10000	1000	0.1

(ii) FeatureIDE generated feature models: The benchmark comprises of 9 models generated using FeatureIDE tool ranging from 500 to 30,000 features. These models are presented in Table 5.3 along with their name and number of features. Each

generated FM comprises of mandatory feature, optional feature and cross-tree constraints where each model includes at least one product, i.e. the model is not void.

Table 5.3: FMs generated using FeatureIDE tool.

	Feature Models generated with FeatureIDE tool	Number of Features
1	FM-500	500
2	FM-1000	1000
3	FM-2000	2000
4	FM-5000	5000
5	FM-10000	10000
6	FM-15000	15000
7	FM-20000	20000
8	FM-25000	25000
9	FM-30000	30000

5.3 Evaluation of the Proposed Framework

A series of experiments were performed to evaluate the proposed ontological rule-based approach. We have used 108 FMs of varied sizes up to 30,000 features, out of which 99 were taken from the SPLOT repository and additionally, 9 other models were generated using FeatureIDE. The goal was to measure the accuracy, computational scalability, and performance analysis of the proposed framework. Following discusses these measurements:

5.3.1 Accuracy

The accuracy of proposed approach is based on the correct (i) transformation of FMs into predicate-based FMOs and (ii) identification of defects along with their causes and corrections**.**

(i) Accuracy of the transformation: An XML parser is developed that transforms SPLOT models from FeatureIDE to FOL predicate-based FMOs based on facts. To check that the FMs have accurately transformed, the number of mandatory features, optional features, implications, exclusions, and cardinalities in the results obtained with the proposed transformation were compared against the input XML file from FeatureIDE (by searching the corresponding tags used in the parser given in Appendix A). These comparisons were made over five FMs with root feature,

features, group cardinalities and cross-tree constraints, and these models consist of 50, 100, 150, 200 and 500 features. The outcomes of these comparisons are same, resulting in 100% accuracy in the transformation of models into predicate-based FMOs with 0% false positives.

(ii) Accuracy of the identification of defects, their causes and corrections: Accuracy is defined by the ratio of the number of defects with their causes and corrections that are accurately identified by the proposed approach to the total number of defects (with their causes and corrections) in the FM. The proposed approach identified 100% of the FM defects with their causes and corrections in 108 FMs (given in Section 5.3) by considering the proposed sets of rules for defects with 0% false positives. Moreover, it was found that all rules individually and simultaneously, identified the expected defects with their causes and corrections signifying 100% accuracy to handle defects considered in the proposed approach.

$$Accuracy = \frac{Number\ of\ identified\ defects\ with\ their\ causes\ and\ corrections}{Total\ number\ of\ defects\ with\ their\ causes\ and\ corrections\ in\ model}$$

5.3.2 Computational Scalability

The performance of proposed approach was tested by calculating the average execution time (in seconds) after executing all rules to deal with defects due to redundancy, dead feature, false-optional feature, inconsistency and wrong cardinality on each one of the six FMs with features 5000, 10,000, 15,000, 20,000, 25,000 and 30,000, as shown in Figure 5.1(a), 5.1(b), 5.1(c), 5.1(d), and 5.1(e) respectively. Number of features in all models and time are represented on x- axis and y-axis respectively. All rules (given in Section 3.2) that deal with defects due to redundancy, dead feature, false-optional feature, inconsistency and wrong cardinality are merged into a single program and implemented over FMs with features 5000, 10,000, 15,000, 20,000, 25,000 and 30,000. The corresponding execution time is shown in Figure 5.2, which determine the scalability of proposed approach to deal with defects.

Table 5.4 represents the results based on Figures 5.1 and 5.2, which describes the minimum and the maximum time required by the queries to deal with defects in models with 5000 features and large-sized FMs with 30,000 features respectively. Results

conclude that in Figures 5.1 and 5.2, queries take a reasonable time of 0.265 and 0.551 sec to deal with defects in large-sized FMs up to 30,000 features. All rules were executed 50 times for each of the 108 models, which implies 405,000 (75*108*50) queries executed in total, in order to attain the valid and reliable measures of execution time. The execution time computed is the average of 50 executions for all rules on each one of the model (i.e. 8100 consolidated results).

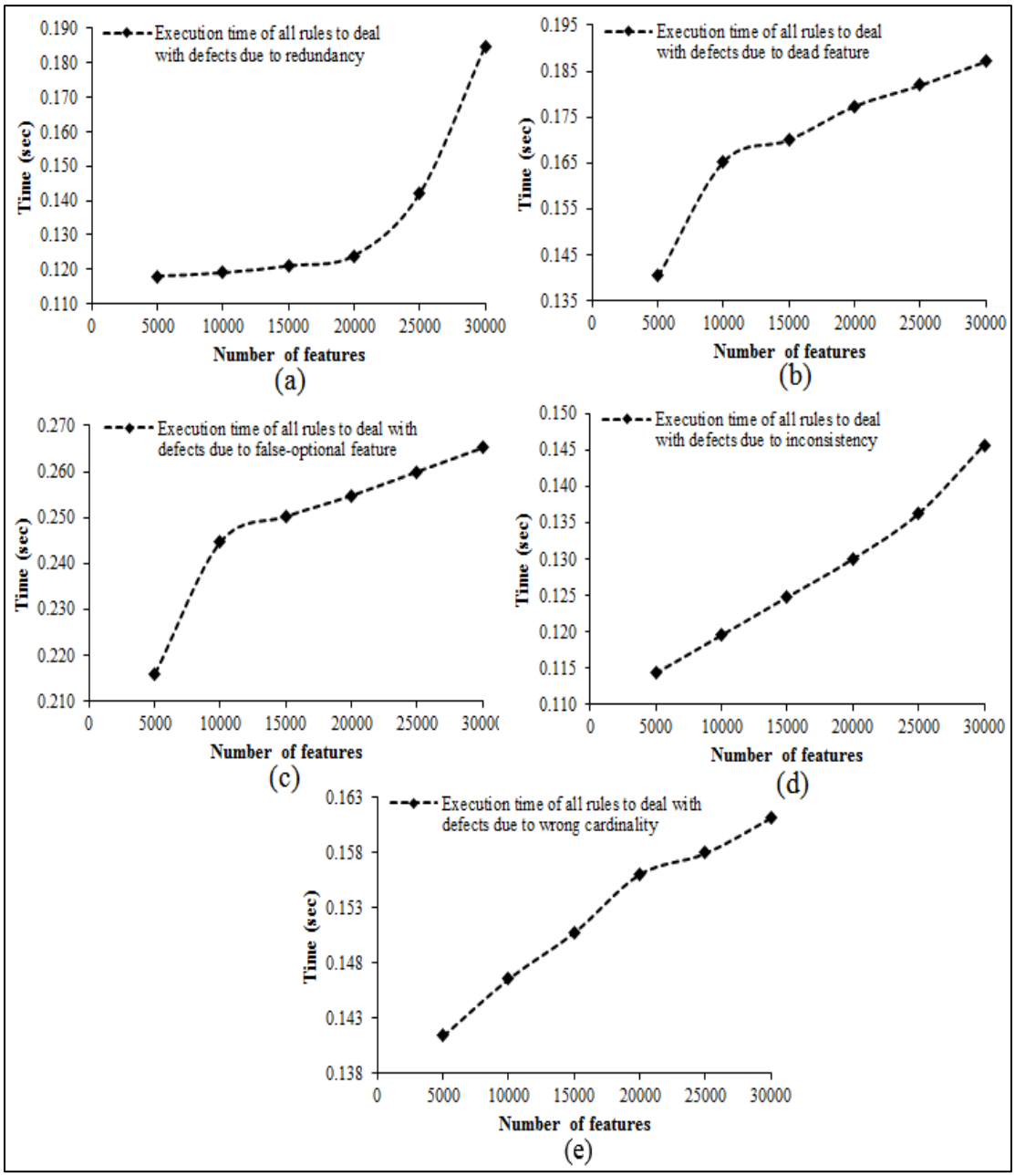


Figure 5.1: Execution time, for all rules, per number of features, to deal with defects due to (a) redundancy, (b) dead feature, (c) false-optional feature, (d) inconsistency, and (e) wrong cardinality.

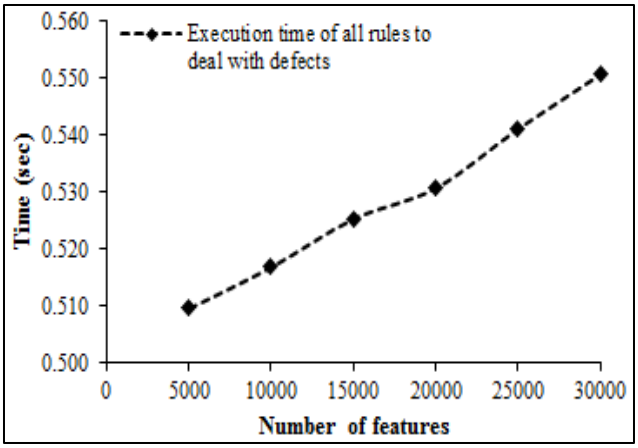


Figure 5.2: Execution time, for all rules, per number of features, to deal with all defects.

Table 5.4: Results of scalability analysis.

Defects	Figure no.	Number of rules	Execution time (in sec) to deal with defects	
			FM (with 5000 features)	FM (with 30,000 features)
Redundancies	5.1(a)	17	0.118	0.185
Dead features	5.1(b)	20	0.140	0.187
False-optional features	5.1(c)	17	0.216	0.265
Inconsistencies	5.1(d)	12	0.114	0.146
Wrong cardinalities	5.1(e)	9	0.141	0.161
All defects	5.2	75	0.510	0.551

5.3.3 Performance Analysis and Evaluation

The performance of the proposed approach has been evaluated by implementing the approach on FMs and measured the execution time consumed by all rules in dealing with defects. This information is used to compare the execution time considered for the factors that affect the performance of proposed approach: number of features and cross-tree constraints (Figure 5.3), height of tree (Figure 5.4), number of alternative and Or relationships (Figures 5.5, 5.6 and 5.7), number of child features related in group cardinality (Figures 5.8 and 5.9), number of defects (Figures 5.10 and 5.11), length of rules (Figure 5.12), and length of cause and correction (Figures 5.13 and 5.14). These factors were chosen due to their impact on execution time. R^2 (R-squared) value for each of the plot in the aforementioned figures has been calculated respectively, in order to measure the data's goodness of fit. The nearer is the value of R^2 to 1, better the fit. It means the line passes closely through all points. Results of performance analysis based on aforementioned factors are shown in Tables 5.6, 5.8, 5.10, 5.12, 5.14, 5.16 and 5.17.

For a better understanding of these results, several notations used and described as follows:

(a) Improved: The performance is improved with model, as it requires less time to deal with defects than from FMs with intermediate and reduced performance.

(b) Intermediate: The performance is average with model, as it requires average time to deal with defects than from FMs with improved and reduced performance.

(c) Reduced: The performance is deteriorated with model, as it requires more time to deal with defects than from FMs with improved and reduced performance.

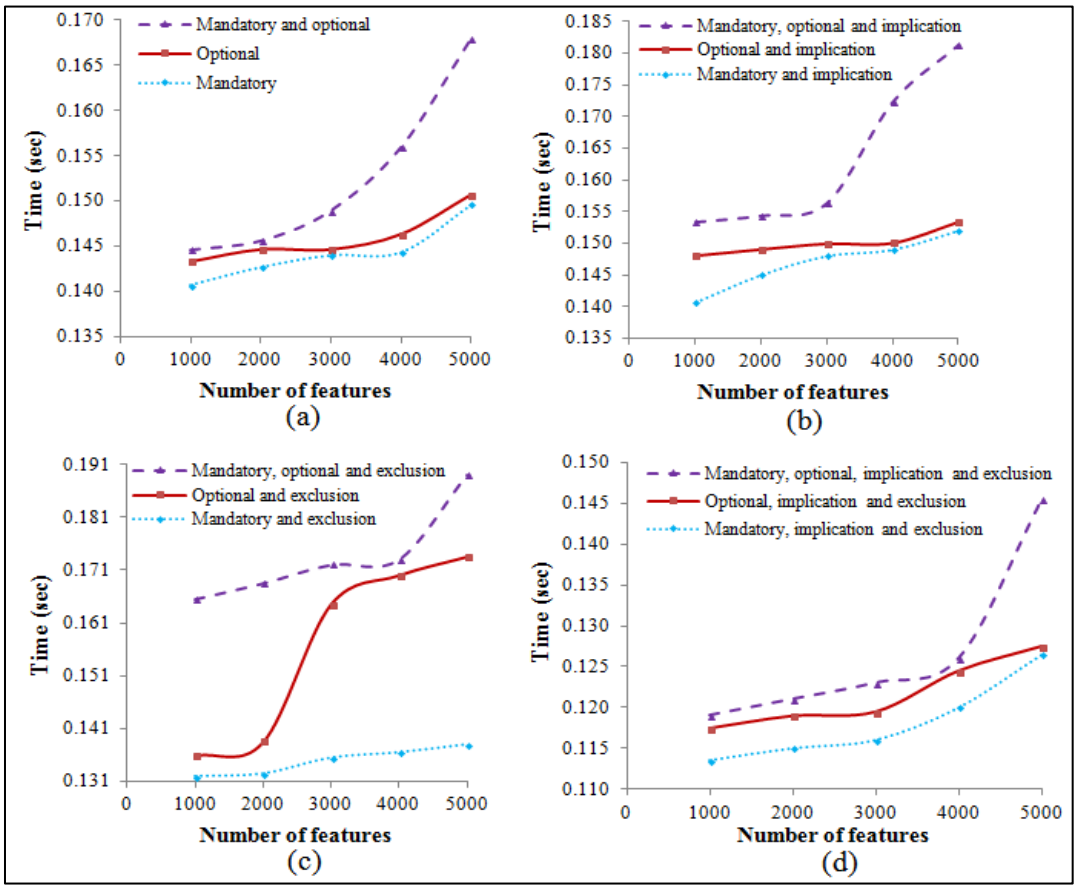


Figure 5.3: Execution time, of all rules, per number features for cross-tree constraints.

Following discusses the number of factors impacting performance of our approach:

(i) Number of features and cross-tree constraints: Figure 5.3 demonstrates the execution time of all rules in FMs with (a) number of features, (b) number of implications, (c) number of exclusions, and (d) number of implications and exclusions in 1:1 proportion. The computed R^2 value for each of the plot given in

Figure 5.3 is shown in Table 5.5. Correspondingly, the results of performance analysis for the same are given in Table 5.6. As an overall trend, (i) improved performance, whereas (iii) deteriorated it, in all cases.

Table 5.5: R^2 value calculated for cross-tree constraints.

Performance parameters	Features Figure 5.3(a)	Implications Figure 5.3(b)	Exclusions Figure 5.3(c)	Implications and exclusions in 1:1 proportion Figure 5.3(d)
(i) Mandatory features	1.0	1.0	1.0	1.0
(ii) Optional features	0.8	0.8	0.9	0.7
(iii) Mandatory and optional features in 1:1 proportion	0.9	0.9	0.8	0.9

Table 5.6: Results of performance analysis for cross-tree constraints.

Performance parameters	Figure no.	Performance analysis and results			
		(i) Mandatory	(ii) Optional	(iii) Mandatory and optional in 1:1 proportion	R^2 value = 1
Features	5.3(a)	Improved	Intermediate	Reduced	(i)
Implications	5.3(b)	Improved	Intermediate	Reduced	(i)
Exclusions	5.3(c)	Improved	Intermediate	Reduced	(i)
Implications and exclusions in 1:1 proportion	5.3(d)	Improved	Intermediate	Reduced	(i)

(ii) Height of the tree: Figure 5.4 demonstrates the execution time of all rules as per the height of tree with increasing levels 10, 20, 30, 40 and 50 in the FMs with features 1000, 2000, 3000, 4000 and 5000 respectively. Moreover, these models are in combination with (a) number of features, (b) number of implications, (c) number of exclusions, and (d) number of implications and exclusions in 1:1 proportion.The computed R^2 value for each of the plot given in Figure 5.4 is shown in Table 5.7. Correspondingly, the results of performance analysis for the same are given in Table 5.8.

(iii) Number of alternative and Or relationships in FM. Figures 5.5, 5.6 and 5.7 demonstrate the execution time of all rules with increasing alternative relationships, Or relationships, and alternative and Or relationships in 1:1 proportion, i.e. 500, 1000, 1500, 2000 and 2500 relationships in FMs with features 1000, 2000, 3000, 4000 and 5000 respectively. Moreover, these models are in combination with (a)

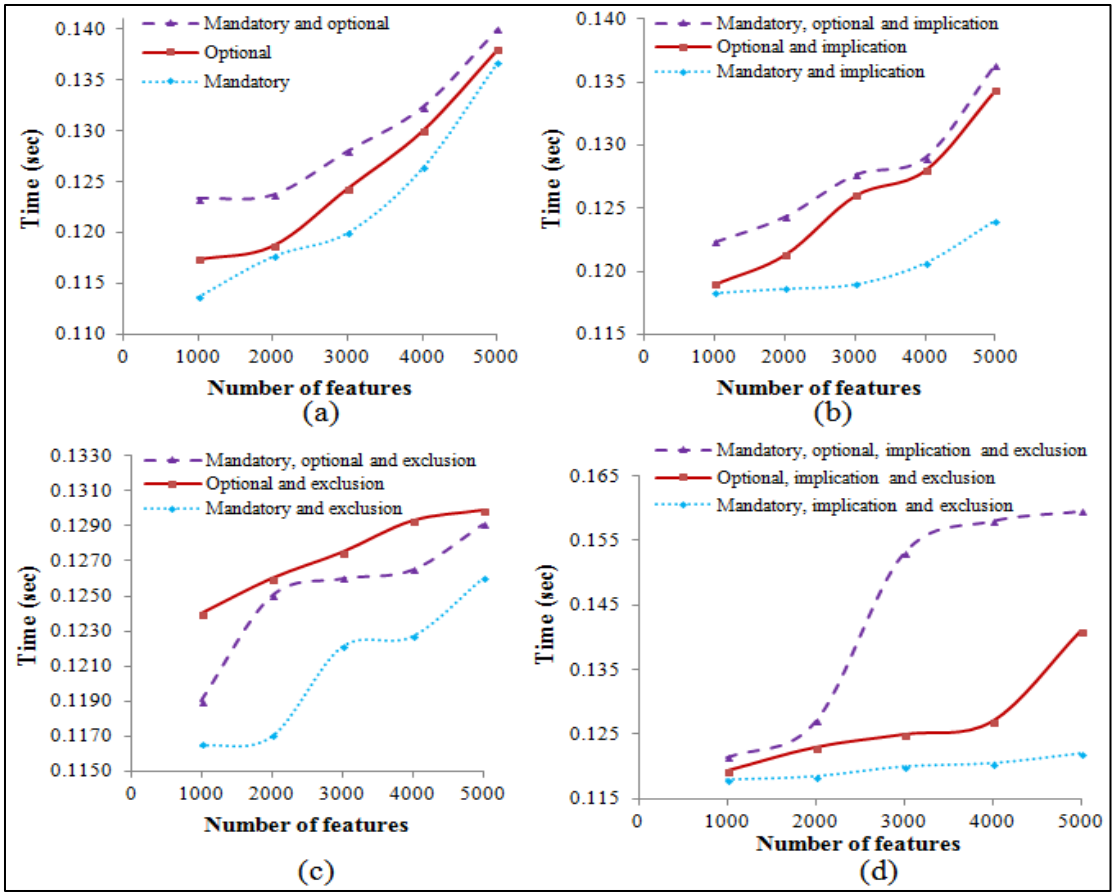


Figure 5.4: Execution time of all rules, per number of features for height of tree.

Table 5.7: R^2 value calculated for height of tree.

With increasing levels in height of tree	Features Figure 5.4(a)	Implications Figure 5.4(b)	Exclusions Figure 5.4(c)	Implications and exclusions in 1:1 proportion Figure 5.4(d)
(i) Mandatory features	0.9	0.8	0.9	1.0
(ii) Optional features	1.0	1.0	1.0	0.8
(iii) Mandatory and optional features in 1:1	0.9	0.9	0.8	0.9

Table 5.8: Results of performance analysis for height of tree.

With increasing levels in height of tree	Figure no.	Performance analysis and results			
		(i) Mandatory	(ii) Optional	(iii) Mandatory and optional in 1:1 proportion	R^2 value = 1
Features	5.4(a)	Improved	Intermediate	Reduced	(ii)
Implications	5.4(b)	Improved	Intermediate	Reduced	(ii)
Exclusions	5.4(c)	Improved	Reduced	Intermediate	(ii)
Implications and exclusions in 1:1 proportion	5.4(d)	Improved	Intermediate	Reduced	(i)

number of features, (b) number of implications, (c) number of exclusions, and (d) number of implications and exclusions in 1:1 proportion. The computed R^2 value for each of the plot given in the aforementioned figures is shown in Tables 5.9, 5.11 and 5.13 respectively. Correspondingly, the results of performance analysis for the same are given in Tables 5.10, 5.12 and 5.14 respectively.

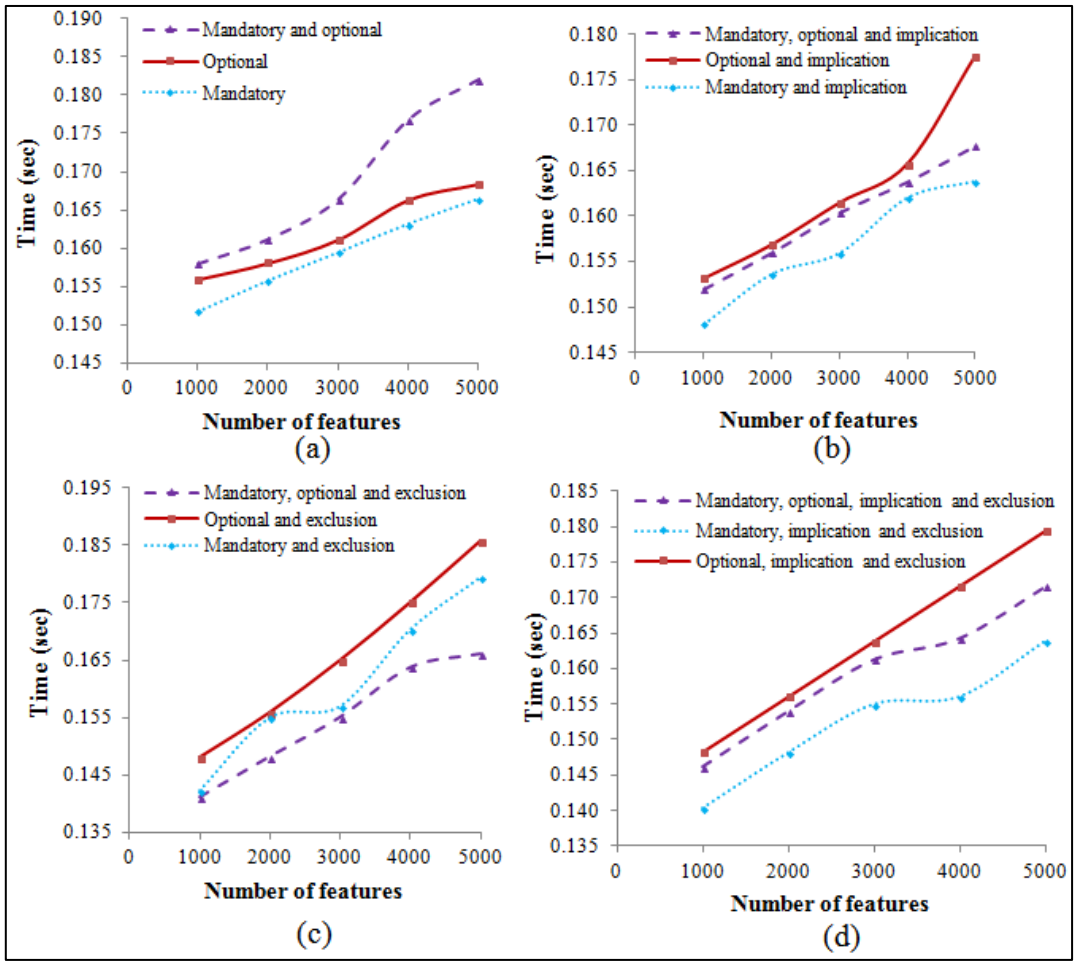


Figure 5.5: Execution time of all rules, per number of features in models with alternative relationships.

Table 5.9: R^2 value calculated for models with alternative relationships.

With alternative relationships	Features Figure 5.5(a)	Implications Figure 5.5(b)	Exclusions Figure 5.5(c)	Implications and exclusions in 1:1 proportion Figure 5.5(d)
(i) Mandatory features	1.00	0.97	0.97	0.96
(ii) Optional features	0.98	0.93	1.00	1.00
(iii) Mandatory and optional features in 1:1 proportion	0.97	1.00	0.98	0.98

Table 5.10: Results of performance analysis for models with alternative relationships.

With alternative relationships	Figure no.	Performance analysis and results			
		(i) Mandatory	(ii) Optional	(iii) Mandatory and optional in 1:1 proportion	R^2 value = 1
Features	5.5(a)	Improved	Intermediate	Reduced	(i)
Implications	5.5(b)	Improved	Reduced	Intermediate	(iii)
Exclusions	5.5(c)	Intermediate	Reduced	Improved	(ii)
Implications and exclusions in 1:1 proportion	5.5(d)	Improved	Reduced	Intermediate	(ii)

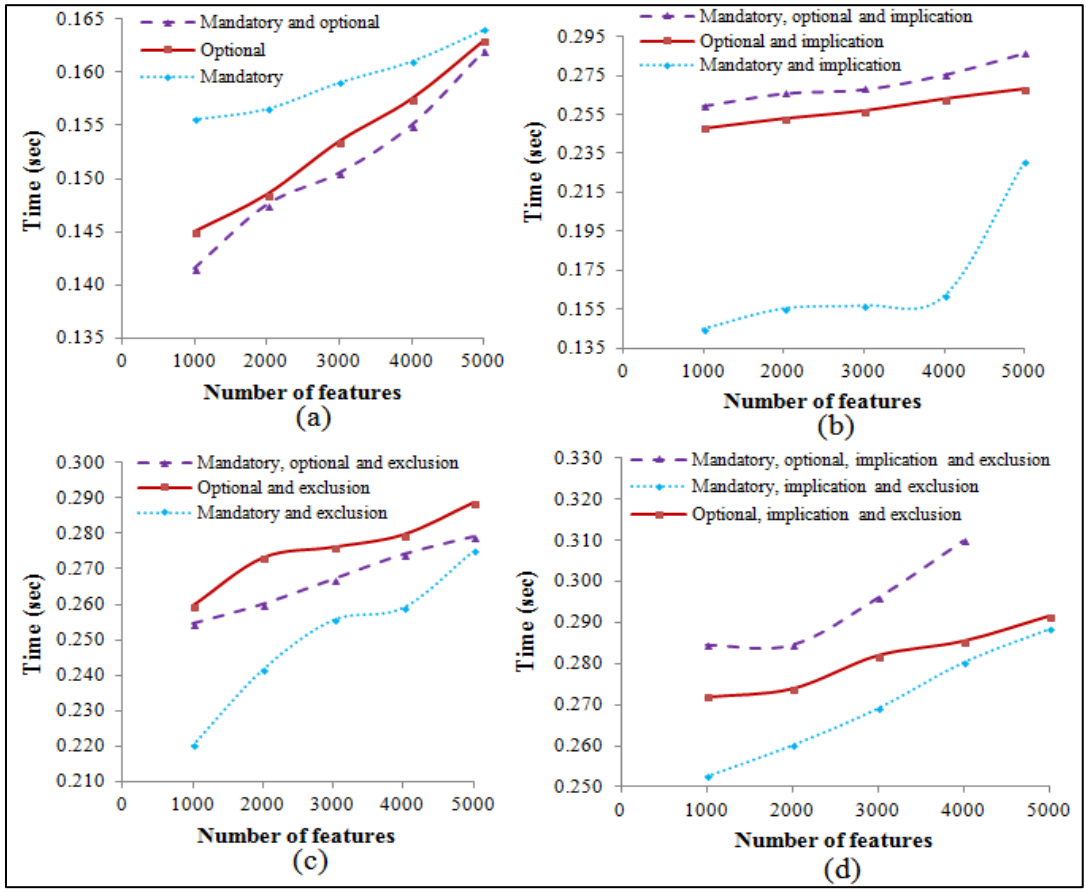


Figure 5.6: Execution time of all rules, per number of features in models with Or relationships.

Table 5.11: R^2 value calculated for models with Or relationships.

With Or relationships	Features Figure 5.6(a)	Implications Figure 5.6(b)	Exclusions Figure 5.6(c)	Implications and exclusions in 1:1 proportion Figure 5.6(d)
(i) Mandatory features	0.98	0.67	0.95	1.00
(ii) Optional features	1.00	1.00	0.93	0.98
(iii) Mandatory and optional features in 1:1 proportion	0.98	0.94	1.00	0.88

Table 5.12: Results of performance analysis for models with Or relationships.

With Or relationships	Figure no.	Performance analysis and results			
		(i) Mandatory	(ii) Optional	(iii) Mandatory and optional in 1:1 proportion	R^2 value = 1
Features	5.6(a)	Reduced	Intermediate	Improved	(ii)
Implications	5.6(b)	Improved	Intermediate	Reduced	(ii)
Exclusions	5.6(c)	Improved	Reduced	Intermediate	(iii)
Implications and exclusions in 1:1 proportion	5.6(d)	Improved	Intermediate	Reduced	(i)

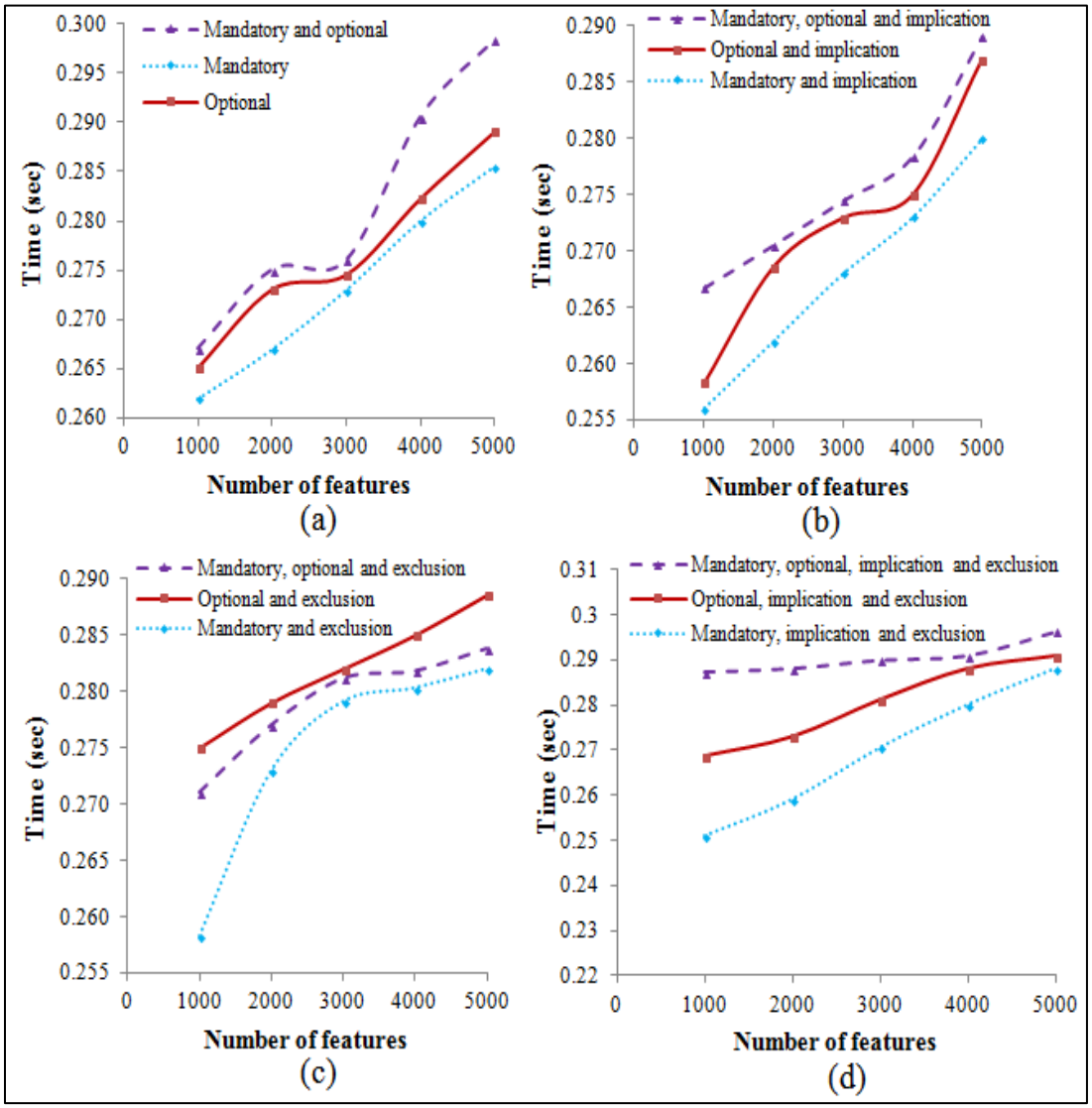


Figure 5.7: Execution time of all rules, per number of features in models with alternative and Or relationships in 1:1 proportion.

Table 5.13: R^2 value calculated for models with alternative and Or relationships in 1:1 proportion.

With alternative and Or relationships in 1:1 proportion	Features Figure 5.7(a)	Implications Figure 5.7(b)	Exclusions Figure 5.7(c)	Implications and exclusions in 1:1 proportion Figure 5.7(d)
(i) Mandatory features	1.00	1.00	0.80	1.00
(ii) Optional features	0.97	0.94	1.00	0.98
(iii) Mandatory and optional features in 1:1 proportion	0.94	0.94	0.89	0.87

Table 5.14: Results of performance analysis for models with alternative and Or relationships in 1:1 proportion.

With alternative and Or relationships in 1:1 proportion	Figure no.	Performance analysis and results			
		(i) Mandatory	(ii) Optional	(iii) Mandatory and optional in 1:1 proportion	R^2 value = 1
Features	5.7(a)	Improved	Intermediate	Reduced	(i)
Implications	5.7(b)	Improved	Intermediate	Reduced	(i)
Exclusions	5.7(c)	Improved	Reduced	Intermediate	(ii)
Implications and exclusions in 1:1 proportion	5.7(d)	Improved	Intermediate	Reduced	(i)

(iv) **Number of child features related in group cardinality:** Figures 5.8 and 5.9 demonstrate the execution time of all rules in FMs consisting of 5000 features with 10 and 50 child features related in each group cardinality respectively, i.e. (a) alternative child features related in group cardinality, (b) Or child features related in group cardinality, and (c) alternative and Or child features related in group cardinality in 1:1 proportion. In Figure 5.8, model (M, O, A, I) required maximum time of 0.30 sec to deal with defects while models (O, A, Or, E and O, A, Or, E, I) consumed minimum time of 0.27 sec for the same. In Figure 5.9, model (O, Or, E)) consumed minimum time of 0.27 sec to deal with defects while model (M, A, Or, I) required maximum time of 0.31 sec for the same. For a better understanding of Figures 5.8, 5.9, 5.10 and 5.11, several notations used and described in Table 5.15.

Table 5.16 represents the results of performance analysis for FMs consist of 5000 features with respect to the number of child features related in each group cardinality

(based on Figures 5.8 and 5.9). As an overall trend, (iii) improved performance in case of 10 child features, whereas (i) deteriorated it, and performance is improved with (i) in case of 50 child features while it is rapidly reduced with (ii).

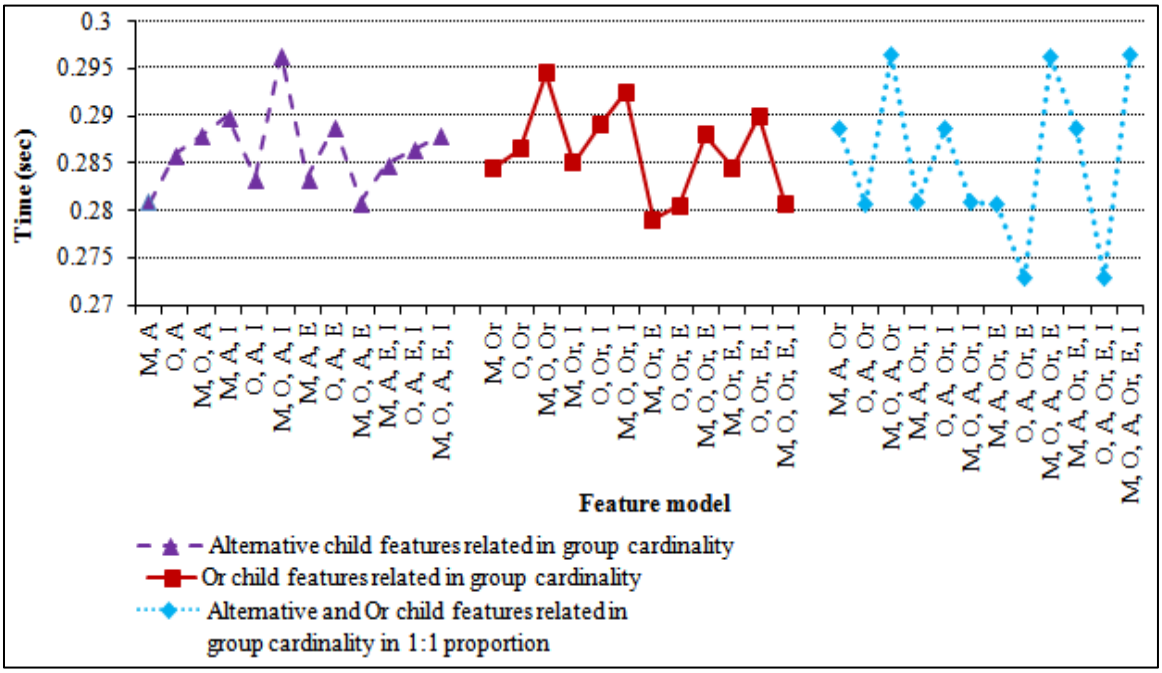


Figure 5.8: Execution time of all rules in models with 5000 features and 10 child features related in each group cardinality.

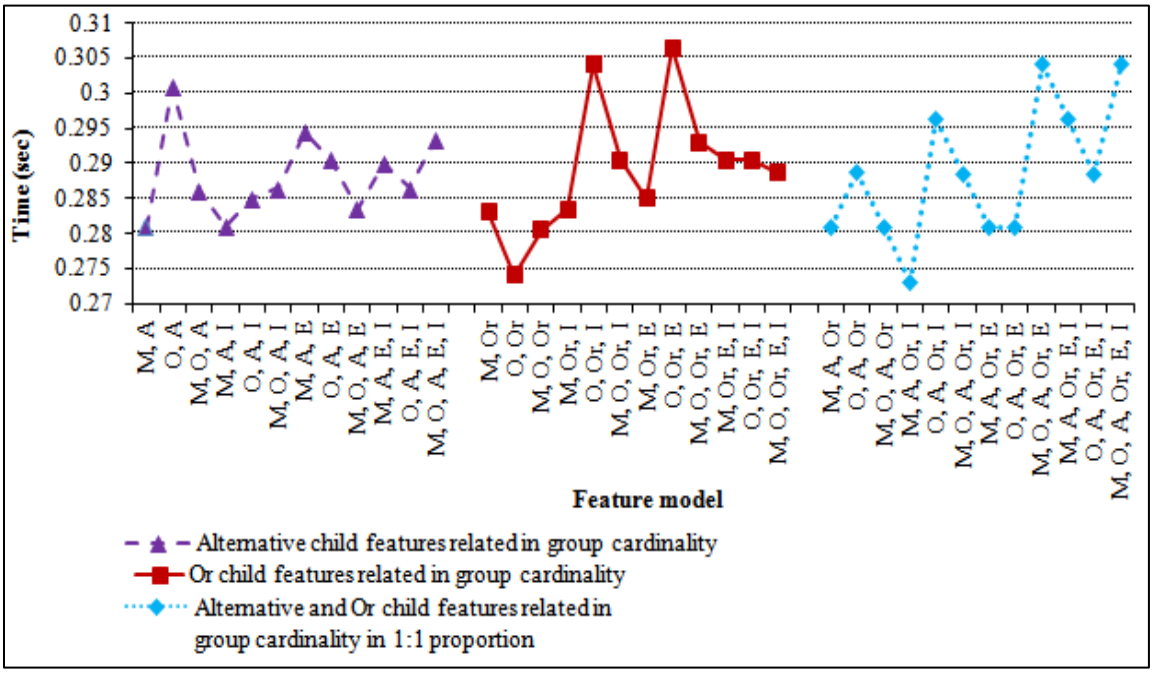


Figure 5.9: Execution time of all rules in models with 5000 features and 50 child features related in each group cardinality.

Table 5.15: Description of notations used in Figures 5.8, 5.9, 5.10 and 5.11.

Notation	Description
M, A	The model comprises of mandatory parent features and alternative child features related in each group cardinality.
O, A	The model includes optional parent features and alternative child features related in each group cardinality.
M, O, A	The model consists of mandatory and optional parent features in 1:1 proportion with alternative child features related in each group cardinality.
M, A, I	The model incorporates mandatory parent features, alternative child features related in each group cardinality and implication relationships.
O, A, I	The model comprises of optional parent features, alternative child features related in each group cardinality and implication relationships
M, O, A, I	The model includes mandatory and optional parent features in 1:1 proportion, alternative child features related in each group cardinality and implication relationships.
M, A, E	The model consists of mandatory parent features, alternative child features related in each group cardinality and exclusion relationships.
O, A, E	The model incorporates optional parent features, alternative child features related in each group cardinality and exclusion relationships.
M, O, A, E	The model comprises of mandatory and optional parent features in 1:1 proportion, alternative child features related in each group cardinality and exclusion relationships.
M, A, E, I	The model includes mandatory parent features, alternative child features related in each group cardinality, exclusion and implication relationships in 1:1 proportion.
O, A, E, I	The model consists of optional parent features, alternative child features related in each group cardinality, exclusion and implication relationships in 1:1 proportion.
M, O, A, E, I	The model incorporates mandatory and optional parent features in 1:1 proportion, alternative child features related in each group cardinality, exclusion and implication relationships in 1:1 proportion.
M, Or	The model comprises of mandatory parent features and Or child features related in each group cardinality.
O, Or	The model includes optional parent features and Or child features related in each group cardinality.
M, O, Or	The model consists of mandatory and optional parent features in 1:1 proportion with Or child features related in each group cardinality.
M, Or, I	The model incorporates mandatory parent features, Or child features related in each group cardinality and implication relationships.
O, Or, I	The model comprises of optional parent features, Or child features related in each group cardinality and implication relationships
M, O, Or, I	The model includes mandatory and optional parent features in 1:1 proportion, Or child features related in each group cardinality and implication relationships
M, Or, E	The model consists of mandatory parent features, Or child features related in each group cardinality and exclusion relationships
O, Or, E	The model incorporates optional parent features, Or child features related in each group cardinality and exclusion relationships.
M, O, Or, E	The model comprises of mandatory and optional parent features in 1:1 proportion, Or child features related in each group cardinality and exclusion relationships.
M, Or, E, I	The model includes mandatory parent features, Or child features related in each group cardinality, exclusion and implication relationships in 1:1 proportion.
O, Or, E, I	The model consists of optional parent features, Or child features related in each group cardinality, exclusion and implication relationships in 1:1 proportion.
M, O, Or, E, I	The model incorporates mandatory and optional parent features in 1:1 proportion, Or child features related in each group cardinality, exclusion and implication relationships in 1:1 proportion.
M, A, Or	The model comprises of mandatory parent features, alternative and Or child features related in each group cardinality in 1:1 proportion.

O, A, Or	The model includes optional parent features, alternative and Or child features related in each group cardinality in 1:1 proportion.
M, O, A, Or	The model consists of mandatory and optional parent features in 1:1 proportion, alternative and Or child features related in each group cardinality in 1:1 proportion.
M, A, Or, I	The model incorporates mandatory parent features, alternative and Or child features related in each group cardinality in 1:1 proportion, and implication relationships.
O, A, Or, I	The model comprises of optional parent features, alternative and Or child features related in each group cardinality in 1:1 proportion, and implication relationships.
M, O, A, Or, I	The model includes mandatory and optional parent features in 1:1 proportion, alternative and Or child features related in each group cardinality in 1:1 proportion, and implication relationships.
M, A, Or, E	The model consists of mandatory parent features, alternative and Or child features related in each group cardinality in 1:1 proportion, and exclusion relationships.
O, A, Or, E	The model incorporates optional parent features, alternative and Or child features related in each group cardinality in 1:1 proportion, and exclusion relationships.
M, O, A, Or, E	The model comprises of mandatory and optional parent features in 1:1 proportion, alternative and Or child features related in each group cardinality in 1:1 proportion, and exclusion relationships.
M, A, Or, E, I	The model includes mandatory parent features, alternative and Or child features related in each group cardinality in 1:1 proportion, exclusion and implication relationships in 1:1 proportion.
O, A, Or, E, I	The model consists of optional parent features, alternative and Or child features related in each group cardinality in 1:1 proportion, exclusion and implication relationships in 1:1 proportion.
M, O, A, Or, E, I	The model incorporates mandatory and optional parent features in 1:1 proportion, alternative and Or child features related in each group cardinality in 1:1 proportion, exclusion and implication relationships in 1:1 proportion.

Table 5.16: Results of performance analysis for models with child features related in group cardinality.

Model with 5000 features and child features related in group cardinality	**Figure no.**	**Performance analysis and results**		
		(i) Alternative	**(ii) Or**	**(iii) Alternative and Or in 1:1 proportion**
10 child features	5.8	Reduced	Intermediate	Improved
50 child features	5.9	Improved	Reduced	Intermediate

(v) Number of defects: Figures 5.10 and 5.11 demonstrate the execution time of all rules in FMs consisting of 5000 features with 35 and 85 defects due to redundancy respectively. In Figure 5.10, model (M, A) with 35 redundancies require the maximum time of 0.42 sec to deal with defects while model (M, A, E, I) required a minimum time of 0.24 sec for the same. In Figure 5.11, model (M, Or) with 85 redundancies require the maximum time of 0.55 sec to deal with defects while model (O, A) required a minimum time of 0.49 sec for the same.

Table 5.17 represents the results of performance analysis for FMs consist of 5000 features with the number of defects due to redundancy (based on Figures 5.10 and

5.11). As an overall trend, (iii) improved performance in case of 35 redundancies, whereas (i) deteriorated it, and performance is improved with (i) in case of 85 redundancies, while it rapidly reduced with (iii).

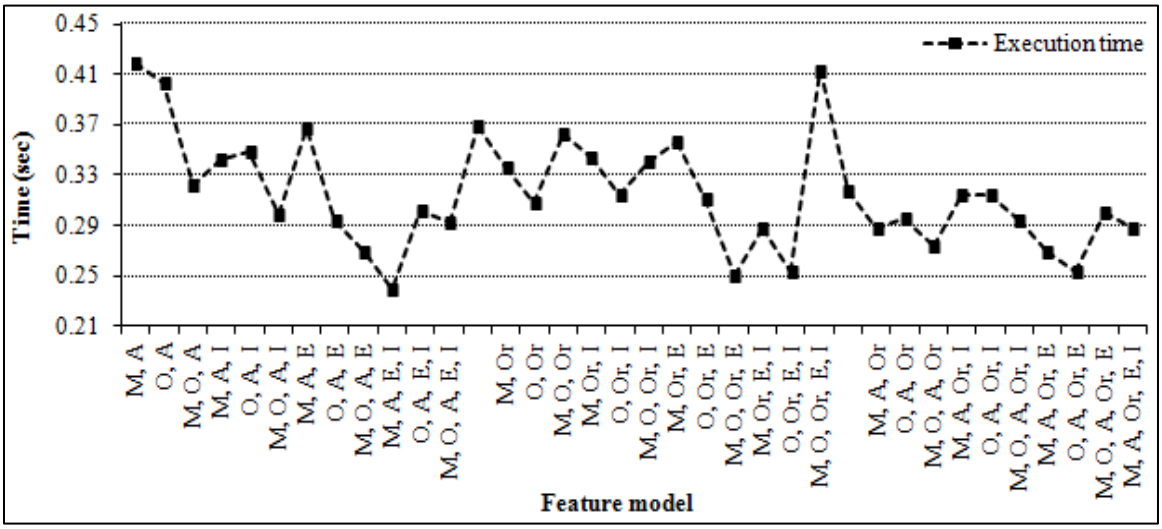


Figure 5.10: Execution time of all rules in models with 5000 features and 35 defects due to redundancy.

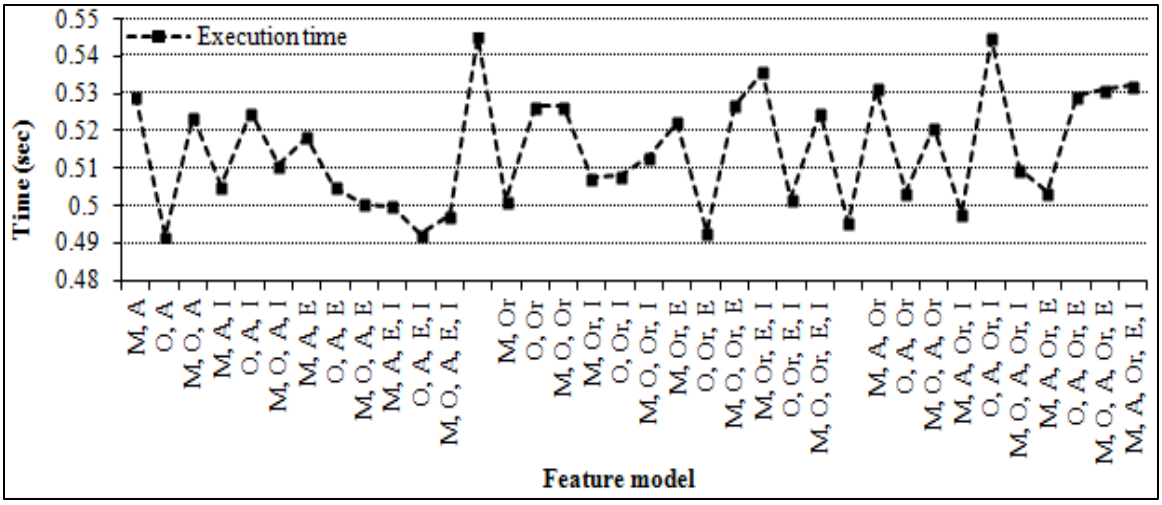


Figure 5.11: Execution time of all rules in models with 5000 features and 85 defects due to redundancy.

Table 5.17: Results of performance analysis for models with a number of defects due to redundancy.

Model with 5000 features and defects due to redundancy	Figure no.	Performance analysis and results		
		(i) Alternative	(ii) Or	(iii) Alternative and Or in 1:1 proportion
35 redundancies	5.8	Reduced	Intermediate	Improved
85 redundancies	5.9	Improved	Intermediate	Reduced

(vi) Length of rules: Figure 5.12 demonstrates the execution time of rules as per length of rule in terms of a number of facts in it. In this thesis, we considered minimum and maximum number of facts 3 and 17 respectively. Results of performance analysis conclude that the rules with 3 and 17 numbers of facts consumed the reasonable time of 0.002 sec and 0.006 sec respectively, to deal with defects. In few cases, rules take same execution time to deal with defects (a) 0.003 sec for 5, 6, 7 and 8 number of facts, (b) 0.004 sec for 9, 10 and 11 number of facts, and (c) 0.006 sec for 14, 15 and 17 number of facts.

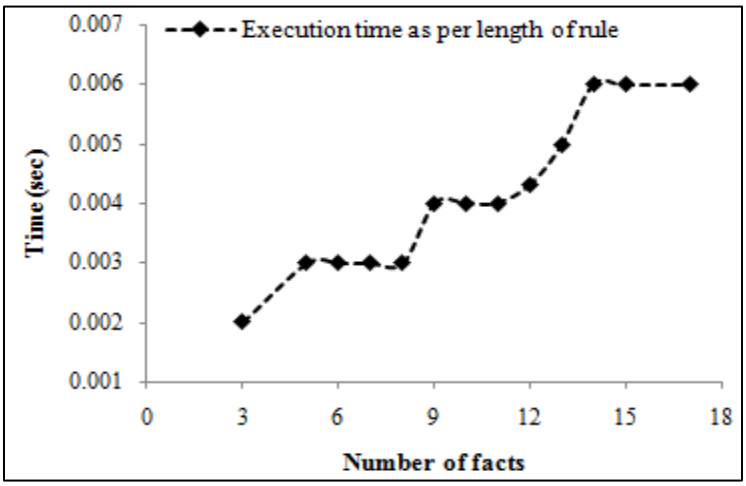


Figure 5.12: Execution time of rules, per length of rules in terms of a number of facts.

(vii) Length of the cause and correction: A rule can generate more than one cause and correction for different defects. The length of a cause and correction for defects is measured in terms of the number of parts involved in them. A part is an elementary unit of a cause or correction representing single reason of defect or solution to fix defect respectively. For instance, Rule D.2 for defect due to dead feature includes 2 parts in causes and 1 part in correction as follows:

Causes: full-mandatory feature c1 excludes an optional feature c2 and optional feature c3 is a child feature of dead feature c2

Correction: eliminate exc1.

Causes and corrections are expressed using a user-friendly natural language since it is the most flexible and the easiest way for PL developers to understand them. Cause of defects generates relevant constraints to the PL developers which are involved in the origin of a defect. The generated correction is minimal to resolve a defect in FM

by eliminating the suggested cross-tree constraints, as it improves the comprehensibility of the PL developers. The overall goal of causes and corrections is to represent relevant parts of the FM leading to a defect in a human-understandable language. The causes and corrections should be short and informative enough to enhance their usability. Accordingly, it was observed that the causes and corrections contain the required information to resolve the defects.

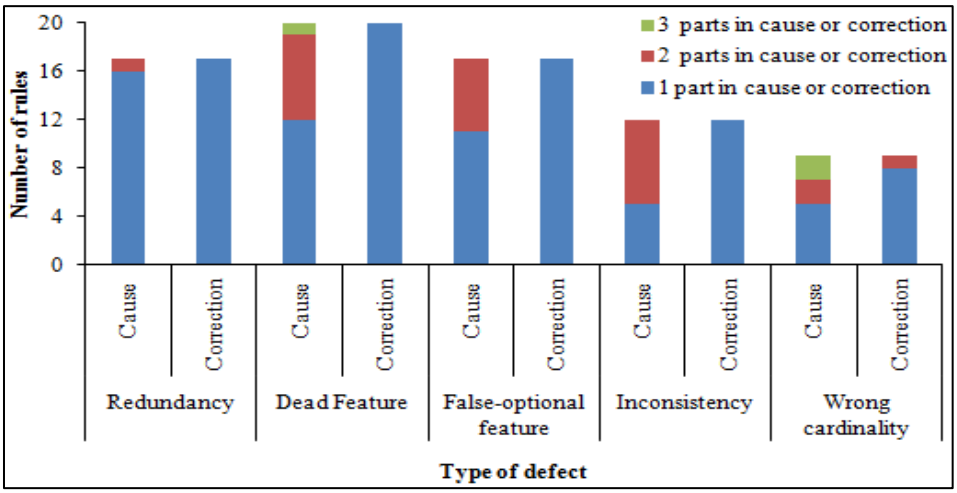


Figure 5.13: Number of parts in cause and correction, per set of rules for each type of defect.

Figure 5.13 illustrates the cause and correction length for each type of FM defect considered in this thesis. The plot reveals that the length of the cause for defects due to (i) redundancy lies at 1 and 2 parts for 16 and 1 rules respectively, (ii) dead feature lies at 1, 2 and 3 parts for 12, 7 and 1 rules respectively, (iii) false-optional feature at 1 and 2 parts for 11 and 6 rules respectively, (iv) inconsistency at 1 and 2 parts for 5 and 7 rules respectively, and (v) wrong cardinality at 1, 2 and 3 parts for 5, 2 and 2 rules respectively. The length of correction is constant at 1 part for defects due to redundancy, dead features, false-optional features and inconsistency. In the case of defects due to wrong cardinality, only single correction consists of 2 parts. The plot reveals that the minimum and maximum length of cause for all types of defects considered in this thesis consist of 1 and 3 parts respectively. All corrections for each type of defect have a length between 1 and 2 parts.

Figure 5.14 summarizes the performance impact for the generation of causes and corrections. A defect can have multiple causes and corrections, which differ in their

length (number of parts involved in them). Each measurement has been repeated five times for each of the rules for defects and their average are used in order to diminish computation bias. The execution time for all parts in cause and correction, per set of rules for each type of defect increases with the growing number of parts in cause and correction. However, a defect with a single part in each cause and correction consumed the minimum time and improved the comprehensibility and readability for PL developers, considering large-sized FMs up to thousands of features.

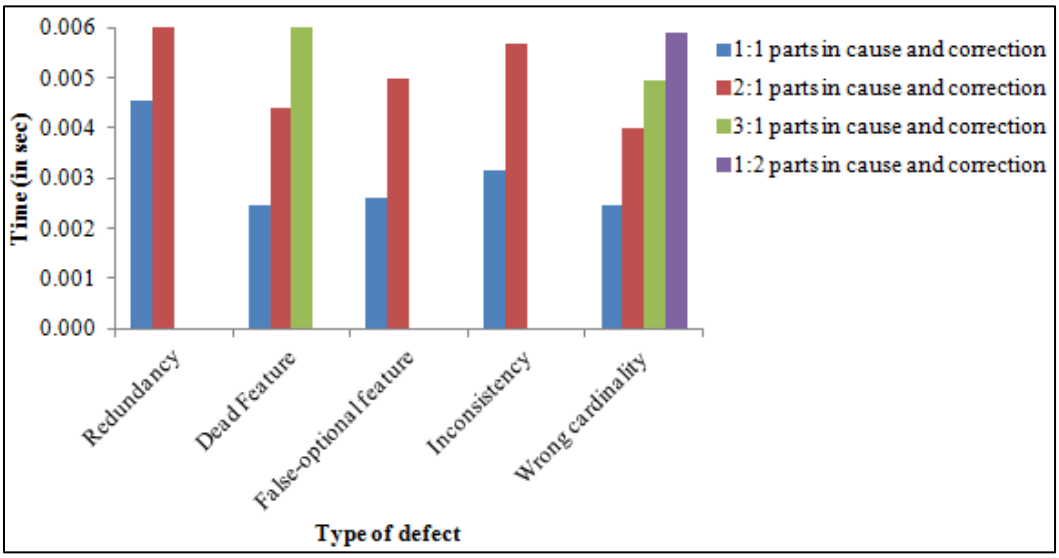


Figure 5.14: Execution time, for all parts in cause and correction, per set of rules for each type of defect.

5.4 Evaluation of Rules

5.4.1 Completeness, Consistency and Consistency Gain of the Set of Rules

To measure the quality of rule, usage of fitness function is considered as an interesting issue in rule-based approaches [117]. The focus is on proving the completeness and consistency of our developed set of rules. In literature, various formulas exist which integrates completeness (ratio of a number of positive examples covered by rule to a total number of positive examples in the training set) and consistency (the ratio of positive examples covered by the rule to the total number of examples covered [118]. Instead of using consistency metric, consistency gain of rule [119] was used, since it takes into account the distribution of positive and negative examples in the training set.

The notation and terminology used for the implementation of the approach are explained,

where rules are represented with a set of rules (a *ruleset*), the term "rule set" is used for "rule," and the term "rule" is used as a component of a rule.

Let *PE* and *NE* be the number of positive and negative examples in the complete training set respectively. Let r be a rule (or a ruleset) developed to cover the examples of that training set, where *pe* and *ne* are the number of positive and negative examples covered by *r*, called positive and negative support respectively.

For *r*, the *completeness (relative support* or *relative coverage)*, *consistency* and *inconsistency (training error rate)* are defined by:

$$comp(r) = pe/PE$$

$$con(r) = pe/(pe + ne)$$

$$incon(r) = ne/(pe + ne)$$

A *complete* cover of the training examples is obtained if the *completeness* of a ruleset for a single class is 100% and a *consistent* cover is obtained if the *inconsistency* of the ruleset is 0%.

The ratio $PE/(PE + NE)$ measures the distribution of positive and negative examples in the training set. The consistency $pe/(pe + ne)$ measures the distribution of positive and negative examples in the set covered by the rule. The gain of the rule consistency over the dataset distribution is given by the difference between $(pe/(pe + ne)) - (PE/(PE + NE))$.

To normalize the expression, we have divided it by $(1 - (PE/(PE + NE)))$, or equivalently by $NE/(PE + NE)$. After rearranging the normalization expression, the consistency gain $(cons(r))$ is defined as:

$$cons(r) = \left(\left(\frac{pe}{pe + ne}\right) - \left(\frac{PE}{PE + NE}\right)\right) \times (PE + NE)/NE$$

This expression determines the value of consistency gain as

(a) Zero, when the distribution of positive and negative examples covered by a rule is identical to the distribution in the entire training set (as a random guess)

(b) One, in the case of perfect consistency

(c) Negative, which turns the rule to be less accurate than a pure random guess.

The normalized expression obtained above calculates the benefits of using the rule over making random guesses. If the rule generates poor results than the random guess, the value of the benefit becomes negative.

The *fitness function* for evaluating the rules is defined by:

$$fitness(r) = \begin{cases} 0, & if\ cons(r) < 0 \\ cons(r) \times \exp\left(comp(r) - 1\right), & otherwise \end{cases}$$

where $comp(r) = pe/PE$ is the completeness of rule.

The exp() function gives preference to the use of consistency gain of the rule, for measuring the quality of rule. The fitness function given above is simple and returns a normalized value between 0 and 1.

5.4.2 Minimalism of the Set of Rules

A rule should be comprised of at least (a) two features, one of them should be the root feature, and (b) one relationship that relates both features. It means that there is no model with a single feature, as a single feature does not leads to defect in a FM. As our domain is FM, the elements features, relationships and cross-tree constraints can represent an entire FM. The notations used for developing the sets of rules can represent a FM in Sub-section 4.2.2 where each developed rule comprises of at least two features represented in the form of *mandatory* or *optional* features and one cross-tree constraint represented in the form of *implication* or *exclusion*. Additionally, *alternative* and *or* relationships are represented by *cardinalities*.

The minimalism of proposed sets of rules for defects can be achieved by minimizing the number of related rules. The elimination of a related rule from a set of rules does not affect the expected outcome which comprises of identified defects, their causes and corrections. For instance, Table 5.18 represents that the elimination of related Rule R.3 from the set of rules for defects due to redundancy does not affect the outcome, as the same defect can be identified using Rule R.5. Eliminating Rule R.3 from the set will reduce the overall execution time by 0.005 sec which in turn will improve the performance of proposed approach. Similarly, the details of other defects are shown in

Table 5.18. The results suggested that the performance is improved by eliminating related rules from their corresponding set of rules.

Table 5.18: Description of rules for the minimalism of sets of rules for defects.

Set of rules for defects	Related rule	Canonical rule	Time (in sec)
Redundancy	R.3	R.5	0.005
Dead feature	D.1	D.2, D.9	0.006
	D.3	D.4	0.007
False-optional feature	F.1	F.2, F.3	0.002
	F.12	F.14	0.007
	F.13	F.15	0.013
Inconsistency	Nil	Nil	Nil
Wrong cardinality	Nil	Nil	Nil

Nil= Not supported by the corresponding row and column.

The accuracy is defined as:

$$A_{max,r} - B_{rel,r} = C_{min,r}$$

$$where, \quad B_{rel,r} = \begin{cases} \emptyset, & if\ there\ is\ no\ related\ rule\ in\ the\ set \\ B_{rel,r} \in A_{max,r} & otherwise \end{cases}$$

$A_{max,r}$ is the set of maximum number of rules for each type of defect handled by our approach, $B_{rel,r}$ is the set of related rules for the corresponding defect, and $C_{min,r}$ is the minimal set of rules that include minimum number of rules which are enough to identify the defects with their causes and corrections handled by proposed approach.

Moreover, the minimal set of rules leads to lower execution times and higher accuracies than a larger randomly chosen set of rules which suggests that there is no gain using additional rules than the minimum set of rules. However, the rules given in column 2 of Table 5.18 are still required to deal with particular defect independently, although these defects are being handled in collaboration with other defects by rules given in column 3.

5.5 Comparative Analysis

5.5.1 Accuracy

The accuracy of the proposed approach is compared with a FeatureIDE tool [44]. The experiments were carried out for defects due to redundancy, dead feature, false-optional feature and inconsistency by considering the practicability of the proposed approach with FeatureIDE. The rules for defects due to the wrong cardinality between FeatureIDE tool

and proposed approach were not compared because they are not supported by FeatureIDE. Table 5.19 shows the results obtained after analyzing *E-commerce system1* FM with defects due to redundancy (a) using a FeatureIDE tool (column 3), and (b) using proposed approach (column 4).

The *Status (S)* represents accuracy where,

(i) *0* indicates that FeatureIDE tool is unable to find accurate defect or constraints involved in the cause of defect or did not identify any defect

(ii) *0.5* indicates that out of two or more constraints involved in the defect, FeatureIDE tool only identified one or half of the constraint involved in the cause of defect

(iii) *1* indicates that FeatureIDE tool finds accurate defect or constraints for the cause of defect

$$accuracy = \frac{1}{m}\sum_{m=1}^{108}\left(\frac{1}{r}\sum_{r=1}^{17} S_{m,r}\right) \qquad (1)$$

where m is the number of models, r is the number of rules, and $S_{m,r}$ represents the status and $S_{m,r} \in [0,1]$.

Table 5.19 demonstrates the calculation of S for one model including defects due to redundancy, namely, *E-commerce system1* FM. As illustrated in Equation 1, the value of Status ($S_{m,r}$) was calculated for each of the considered 17 rules for redundancies, which further were executed for 108 models. The final accuracy of FeatureIDE is 73.53% for redundancies which was computed based on the average of the values as shown in Equation 1. Similarly, all remaining running examples of FMs, i.e. *CIMS PL, Computer Selection* and *Address Book* considered in this thesis with defects due to dead feature, false-optional feature and inconsistency respectively, were also analyzed using FeatureIDE in the same way. The accuracy results for the same are given in Table 5.20 which conclude FeatureIDE's accuracy to be (i) 15% for dead features, (ii) 0% for false-optional features (due to exc2 between c30 and c32 which turns model to be void, as shown in Figure 4.65 (b)), and (iii) 0.42% for inconsistencies. However, the proposed approach handled 100% of the considered defects with 0% false positives. Additionally, all 75 rules individually and jointly, identified, provided causes and corrections of all the expected defects, indicating that the proposed approach is 100% accurate.

Table 5.19: Comparing the accuracy of FeatureIDE tool with the proposed approach using E-commerce system1 FM for redundancies.

Rules	Constraints description	FeatureIDE tool	Proposed approach	Status(S)
R.1	(i) a2 excludes a3	(i) Dead features: manufacturerscatalogue, manufacturercategories, a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a12, a14, a16, title, a18, a20, a21, a22, a23, a25, manufacturer, description, a26, a27, a28, a29, a30, a32, products, a38, a39, a41, a42, a43, a44, a45, a46, a47, a33, a34, a36, a37 False-optional features: a11, a13, a15, a17, a24, a31, a35, a40, photos	Defect: Redundancy1 Cause: exclusion between alternative-child features a2 and a3 is redundant Correction: eliminate exc1	0
R.2	(i) a8 excludes a6	(i) Constraint is redundant and could be removed	Defect: Redundancy2 Cause: multiple exclusion of an alternative-child feature a6 Correction: eliminate exc2	1
R.3	(i) a11 implies a10	(i) Constraint is redundant and could be removed	Defect: Redundancy3 Cause: optional feature a11 implies the full-mandatory feature a10 is redundant Correction: eliminate imp1	1
R.4	(i) a13 implies a16	(i) Constraint is redundant and could be removed	Defect: Redundancy4 Cause: optional feature a13 implies the full-mandatory feature a16 is redundant Correction: eliminate imp2	1
R.5	(i) a19 implies a20 (ii) a20 implies a21	(i) Constraint is redundant and could be removed (ii) Constraint is redundant and could be removed	Defect: Redundancy5 Causes: optional feature a19 implies a full-mandatory feature a20 and mandatory child feature a21 is implied by its parent a20 are redundant Corrections: eliminate imp3 and imp4	1
R.6	(i) a24 implies a25	(i) Constraint is redundant and could be removed	Defect: Redundancy6 Cause: optional feature a24 implies a full-mandatory featurea25 is redundant Correction: eliminate imp5	1
R.7	(i) a26 implies a27 (ii) a26 implies a28	(i) Constraint is redundant and could be removed (ii) Constraint is redundant and could be removed	Defect: Redundancy7 Causes: both mandatory child features a27 and a28 are implied by their parent a26 are redundant Corrections: eliminate imp6 and imp7	1

R.8	(i) a30 implies a31 (ii) a32 implies a31	(i) Constraint is redundant and could be removed (ii) Constraint is redundant and could be removed	Defect: Redundancy8 Cause: multiple implication of an optional feature a31 Correction: eliminate imp9	0.5
R.9	(i) a36 implies a35 (ii) a36 implies a37	(i) Constraint is redundant and could be removed (ii) Constraint is redundant and could be removed	Defect: Redundancy9 Cause: implication on a37 is redundant after a35 is implied by a mandatory feature a36 Correction: eliminate imp11	0.5
R.10	(i) a39 implies a40 (ii) a41 implies a42	(i) Constraint is redundant and could be removed (ii) Constraint is redundant and could be removed	Defect: Redundancy10 Cause: mandatory feature a41 implies another mandatory feature a42 is redundant Correction: eliminate imp13	0.5
R.11	(i) a46 implies a45 (ii) a46 implies a47	(i) Constraint is redundant and could be removed (ii) Constraint is redundant and could be removed	Defect: Redundancy11 Causes: implications on mandatory features a45 and a47are redundant Corrections: eliminate imp14 and imp15	1
R.12	(i) a49 excludes a50 (ii) a51 excludes a50	(i) Constraint is redundant and could be removed (ii) Constraint is redundant and could be removed	Defect: Redundancy12 Cause: multiple exclusion of an optional feature a50 Correction: eliminate exc4	0.5
R.13	(i) a53 excludes a54 (ii) a55 excludes a56	(i) Constraint is redundant and could be removed (ii) Constraint is redundant and could be removed	Defect: Redundancy13 Cause: mandatory feature a55 excludes another mandatory feature a56 is redundant Correction: eliminate exc6	0.5
R.14	(i) a57 implies a58 (ii) a58 implies a59 (iii) a59 implies a57	(i) Constraint is redundant and could be removed (ii) Constraint is redundant and could be removed (iii) Constraint is redundant and could be removed	Defect: Redundancy14 Cause: implication from a59 to a57 is redundant Correction: eliminate imp18	0.5
R.15	(i) a60 implies a61 (ii) a61 implies a62 (iii) a60 implies a62	(i) Constraint is redundant and could be removed (ii) Constraint is redundant and could be removed (iii) Constraint is redundant and could be removed	Defect: Redundancy15 Cause: implication from a60 to a62 is redundant Correction: eliminate imp21	0.5
R.16	(i) a63 implies a65	(i) Constraint is redundant and could be removed	Defect: Redundancy16 Cause: implication from a63 to a65 is redundant Correction: eliminate imp22	1
R.17	(i) a68 implies a66	(i) Constraint is redundant and could be removed	Defect: Redundancy17 Cause: implication from a68	1

	to a66 is redundant Correction: eliminate imp23

Table 5.20: Accuracy results of analyzing models with defects using the FeatureIDE tool.

Feature Model	Defects	Number of rules	Accuracy of FeatureIDE tool (in %)
E-commerce system1	Redundancies	17	73.53
CIMS PL	Dead features	20	15
Computer Selection	False-optional features	17	0
Address Book	Inconsistencies	12	0.42

5.5.2 Comparison with existing works

This Sub-section illustrates the results obtained after comparing the approach with existing approaches based on various factors as follows:

(i) Performance analysis: Table 5.21 summarizes the performance analysis based on the comparison of execution time required to deal with defects among the proposed approach and existing approaches. The proposed approach takes less time to deal with defects (column 7) for particular cases (columns 2-5 in Table 5.21) when compared to existing approaches (column 6). The results indicate the improved performance as compared to existing approaches.

(ii) Number of rules: Table 5.21 shows the total number of rules (column 4) and number of rules per defect type (column 5) identified in existing approaches [30, 41, 43, 45, 96, 97, 115]. For instance, in case of Elfaki et al. [96], the total number of rules are 13 (column 4) and number of rules per defect type are represented as R (9), D (4) which means that the authors gave 9 rules to deal with redundancy and 4 rules to deal with dead features. Further, Salinesi and Mazo [25] presented 15 FM verification criteria where each criterion representing a case of defect while our approach provides 75 rules to handle defects where each rule leads to a particular case of defect. However, the proposed approach not only deals with defects considered by existing works (column 1) but also handled other cases of defects with their causes and corrections (Sub-section 3.2) to resolve defects.

(iii) Data set used for validation: The proposed approach is validated using real-world FMs and automatically-generated FMs available in SPLOT repository, and

Table 5.21: Comparison of existing approaches with the proposed approach based on the number of rules and execution time to deal with defects.

Articles	Description of features, constraints and defects	Defect type	Total Rules	Number of rules per defect type	Execution time (in sec)	
					Existing approaches	Proposed approach
Trinidad et al. [49]	86	D, FO, I	-	-	-	-
Salinesi et al. [20]	49	D, I	-	-	-	-
Thüm et al. [86]	10,000	-	-	-	-	-
Wang et al. [91]	(i) 3751features, 290 constraints (ii) 4433 features, 330 constraints	I	-	-	(i) > 56 (ii) >64	(i) 0.069 (ii) 0.073
White et al. [90]	5000	I	-	-	≈ 50	0.114
Gheyi et al. [23]	2000	I	-	-	Identification (94)	Identification (0.024)
Mazo et al. [115]	10,000	R, I, WC	9	R (1), I (2), WC (1) Others (5)	Execution time of each rule < 0.14	Execution time of each rule < 0.008
Noorian et al. [39]	31	I	-	-	147	0.052
Guo et al. [93]	10,000	I	-	-	0.072	0.055
Salinesi and Mazo [25]	2000	R, D, FO, I, WC		15 verification criteria	Execution time of each verification operation <19 seconds	Execution time of each rule < 0.008
Felfernig et al. [40]	172	I, R	-	-	FASTDIAG algorithm (10.54) HSDAG algorithm (100)	0.126
Elfaki et al. [96]	20,000	R, D	13	R (9), D (4)	Detection with cause (221.516)	Identification with cause (0.058) Identification with cause and correction (0.07)
Giraldo et al. [97]	50	D	2	D (2)	-	0.008
Zhang et al. [41]	1000 features, 50 requires, 50 excludes	D, FO	11	D (6), FO (5)	1.18	0.061
Rincón et al. [43]	150	D, FO	9	D (6), FO (3)	Identification with cause (120)	Identification with cause (0.032) Identification with cause and correction (0.041)
Elfaki et al. [30]	20,000	FO, WC	6	FO (4), WC (2)	Detection FO (221.516)	Identification (0.016)

						Identification with cause and correction (0.025)
White et al. [100]	1000	I	-	-	4	0.062
Perez-Morago et al. [32]	5000	D	-	-	Identification (7.730)	Identification (0.046) Identification with cause and correction (0.092)
Elfaki [45]	20,000	I	21	I identification (13) I prevention (8)	Identification (97.309) Prevention (67.515)	Identification (0.047) Identification with cause and correction (0.103)
Kowal et al. [102]	2513 features, 2,833 constraints, R (563), D (192), FO (12)	R, D, FO	-	-	16,546.44	3.475
Proposed approach	30,000	R, D, F, I, WC	75	R (17), D (20), FO (17), I (12), WC (9)	-	R (0.185), D (0.187), FO (0.265), I (0.146), WC (0.161), Combine result (0.551)

R: redundancy, D: dead features, F: false-optional features, I: inconsistency, WC: wrong cardinality, Others: Rules not for defects, Combine result: execution time of all rules

models generated using FeatureIDE tool in contrast to the works of Elfaki et al.[30], Elfaki [45], Osman et al. [84], Elfaki et al. [85], and Elfaki et al. [96] who have used their own generated data sets for the validation of their methods.

(iv) Scalability: The approaches given by Salinesi et al. [20], Gheyi et al. [23], Salinesi and Mazo [25] Elfaki et al. [30], Perez-Morago et al. [32], Noorian et al. [39], Felfernig et al. [40], Zhang et al. [41], Lesta et al. [42], Rincón et al. [43], Elfaki [45], Trinidad et al. [49], Thüm et al. [86], Wang et al. [91], White et al. [90], Guo et al. [93], Elfaki et al. [96], Giraldo et al. [97], White et al. [100], Kowal et al. [102], and Mazo et al. [115] are scalable up to features as shown in column 2 of Table 5.21 while the proposed method is scalable up to 30, 000 features (as shown in Figure 5.1 of Sub-section 5.3.2).

(v) Discussions and comparative analysis

Following discusses the results of comparative analysis:

(a) Generic framework: In this thesis, a generic framework is proposed which can handle many defects like redundancy, dead feature, false-optional feature, inconsistency and wrong cardinality in FMs. Therefore, any defect which is modeled in the described formalism (i.e. based on FOL predicates) can be handled by the proposed framework.

(b) Level of expressiveness: The proposed approach enriches the level of expressiveness for PL developers by formalizing model into FOL predicate-based FMO. In contrast to the proposed approach, Elfaki et al. [30] and Elfaki et al. [96] who represented variability in SPL by merging FM and OVM.

(c) Wrong cardinality and redundancy support: In the literature, defects due to wrong cardinality and redundancy are not much explored as problems due to other defects such as dead feature, false-optional feature and inconsistency. Trinidad and Ruiz-Cortés [36] provided no automated support to deal with wrong cardinality. Elfaki et al. [30] provided auto-support to identify wrong cardinality. Salinesi et al. [21] and Salinesi and Mazo [25] only identified wrong cardinality and redundancy based on their verification criteria. Further, Elfaki et al. [96] only identified redundancies. Later, the cause of redundancy is highlighted by Kowal et al. [102]. Felfernig et al.

[40] recommended corrections for redundancies, yet they have not provided any implementation details. Moreover, their explanations comprise of constraint sets which increase the difficulty to understand the anomaly for developers. Furthermore, the aforementioned works of researchers did not identify corrections for defects due to wrong cardinality and redundancy, whereas the proposed approach, in addition to identifying these defects, have also provided their causes and corrections in a user-friendly language to resolve defects.

(d) Classification support: Salinesi et al. [21] and Salinesi and Mazo [25] presented a typology of FM verification criteria. It includes 15 verification criteria where each criterion representing a case of defect while the proposed approach classified defects into 75 cases (see Section 3.2). The proposed approach classified defects due to wrong cardinality (in CBFMs) into nine cases (see Sub-section 3.2.5), whereas Elfaki [30] classified wrong cardinality into four types only.

(e) Inconsistency comparison: Most of the existing approaches [27-29] only identified a direct inconsistency in the configuration process, while the proposed approach additionally identified other types of inconsistencies (see Sub-section 3.2.4) without the requirement of the configuration process. Moreover, the cause of defects in natural language is an improvement on earlier work by Mannion [18], Gheyi et al. [23] and Thüm et al. [31], where inconsistency detection is done but no explanation for their causes is given. The proposed approach provided corrections for other cases of inconsistencies in contrast to the Elfaki [45] approach where only a direct inconsistency is prevented.

(f) Causes: Trinidad and Ruiz-Cortés [36] and Zhang et al. [41] only provided a list of constraints and no information is suggested to PL modelers about the occurrence of defects due to these constraints in PLMs. Few researchers [38, 40, 49] only generated a list of relationships involved in the defect that must be modified by modelers to fix the defect. Although, Giraldo et al. [97] identified causes and Rincón et al. [43] provided causes of defective FMs in natural language. In contrast, the proposed approach additionally identifies defects with their causes and corrections in a user-friendly natural language which are comprehensible by PL developers in resolving the defects.

(g) **Correction support**: Few researchers worked on the identification and cause of defects due to redundancy [96, 102], but none of them resolved redundancy, whereas the proposed approach, in addition to identifying defects due to redundancy with their causes, also identified corrections in a user-friendly natural language which is comprehensible by PL developers to resolve defects. Thüm et al. [44] developed a tool that suggests corrective solutions to fix defects. The major drawback of their tool is that it does not support corrections to fix the defects which require deleting more than one relationship. However, the proposed approach provides corrections which include deletion of more than one relationship. It is worth noting that the proposed approach detected actual defective features and the cross-tree constraints for their causes and corrections, and not defective FMs. Additionally, correction provided by the proposed approach is minimal, as it targets the cross-tree constraints itself which are involved in the source of defect.

(h) **Resolving defects early in the FM development process:** Osman et al. [84] provided explanations and Elfaki et al. [30] detected wrong cardinality during the configuration process of SPL. Zhang et al. [41] and Rincón et al. [101] found defects in the later stages of the development process. In contrast to the above mentioned approaches, the proposed approach assists in defect identification with their causes and corrections during feature modeling. The advantage of the proposed approach is that defects are found early in the development process of FM and, therefore, it allows PL developers to create well-formed models. Thereby, significantly reducing the time and cost for finding corrections to resolve FM defects which are found late in the development stages. Further, it allows deriving and reusing defect free valid software products.

(i) **Efficient search process:** Generally, researchers validated SPL by generating all products using solvers and detected a defect in each software product. However, it is a very complex process and almost impossible to generate all products in large-size SPL [46, 49]. Further, Salinesi et al. [21] automated functions by using off-the-shelf solvers to, but they are computationally expensive in large-size PLMs. The approaches given by Trinidad et al. [49] and Lesta et al. [42] rely on using a constraint solver. The constraint solver must be modified directly in the work of

Lesta et al. [42]. In contrast to the approaches mentioned above, the proposed approach is independent of any solver. It also provides an efficient search process in SPL and makes the process low-cost for large-sized FMs as it searches only for the predefined specific cases to identify defects with their causes and corrections instead of the set of dependencies.

5.6 Threats to Validity

The results of experimental evaluation show that the proposed approach is valid. However, the validity of these experiments results may get affected by some conditions. Following are various validity threats that could affect the experiments.

5.6.1 External validity

Even though the majority of the input models used in the experiments are real-world models available in SPLOT repository, there are also some automatically-generated FMs (with 500 to 30,000 features) that we have used for the experimentation, which may not reflect real-world models and may cause a threat to external validity. The complexity of the model and the problem size vary with real-world models. Therefore, this effect is minimized by using automatically-generated 3-CNF-FMs available in SPLOT repository and generating models using a well-known FeatureIDE tool. Moreover, the analysis of FMs is run independently in order to reduce the impact of other threads on the calculated execution time such as threads of operating system. All rules were executed 50 times on each one of the model and its average is used as a result to minimize the impact.

5.6.2 Internal validity

The additional features and cross-tree constraints are introduced in the input FMs to cause defects, and it is one of the threats to internal validity. Results obtained after transforming FMs comprise of these additional features and cross-tree constraints. The threat to internal validity is diminished as the proposed approach identified all defects with their causes and corrections generated due to the additional features and cross-tree constraints, in a reasonable performance time of 0.551 sec in large-sized FMs up to 30,000 features.

Another threat is that the incorrect cardinalities are manually introduced in the FMO as

FeatureIDE doesn't support FMs with the wrong cardinality. This threat is tackled by the proposed approach, as it identified all defects due to wrong cardinality with their causes and corrections generated due to the injected incorrect cardinalities, in an extremely reasonable performance time of 0.161 sec in large FMs with 30,000 features. The calculated execution time to analyze a FM affects the threat to internal validity. Various factors that impact the performance are number of features and cross-tree constraints (Figure 5.3), height of tree (Figure 5.4), number of alternative and Or relationships (Figures 5.5, 5.6 and 5.7), number of child features related in group cardinality (Figures 5.8 and 5.9), number of defects (Figures 5.10 and 5.11), length of rules (Figure 5.12), and length of cause and correction (Figures 5.13 and 5.14). The internal validity of the experiments is improved by experimenting the FMs (ranging from 10 to 30,000 features) with aforementioned factors (Sub-section 5.3.3).

5.7 Summary

This chapter presented the results of experiments carried out to evaluate the proposed approach. The proposed framework is evaluated based on the accuracy, computational scalability, and performance analysis. Further, the accuracy is evaluated using the accurate (i) transformation of FMs into predicate-based FMOs, and (ii) identification of defects along with their causes and corrections. The rules are also evaluated based on the completeness, consistency and consistency gain of the set of rules, and minimalism of the set of rules. The results of comparative analysis obtained after comparing the proposed approach with existing approaches are also discussed. Additionally, the accuracy of proposed approach is compared with FeatureIDE tool. Finally, threats to validity of the proposed approach are also explained.

CHAPTER 6: Conclusions and future work

This chapter summarizes the conclusions and future work.

6.1 Conclusions

One of the major factors behind the successful derivation of defect free valid software products from SPL is the quality of FM. A defect in FM is a critical problem which hinders the derivation of high-quality software products in SPL. The problem of enhancing the quality of software products derived from SPLE is tackled by dealing with defects due to redundancy, anomaly, inconsistency and wrong cardinality in FMs. In this thesis, following work has been done:

(i) Classification: Defects in FM have been classified in the form of cases.

(ii) Generic framework: A framework has been proposed for improving SPL by analyzing defects using an ontological rule-based approach.

(iii) Formalization of the FM: FM in classical formalism cannot be verified as such, and therefore, it has been formalized from an ontological viewpoint by transforming model into a predicate-based FMO. It improves expressivity and provides the correspondence between FM representation and FMO.

(iv) Development and implementation of FOL rules: A set of FOL rules (each rule represents a specific case of a defect) has been developed and applied on the generated FMO to deal with defects. Each rule has been applied individually and all together at the same time in Prolog. These rules are independent of each other.

(v) Automated identification of defects with their causes and corrections: The results include the identified defects with their causes. Additionally, the proposed approach provides corrections to resolve defects.

(vi) Comprehensible causes and corrections: The causes and corrections explained in a user-friendly natural language are easily comprehensible by PL developers. This information assists developers to eliminate cross-tree constraints involved in the source of defects for resolving defects.

(vii) Evaluation with toughest benchmark: The developed framework has been validated using a corpus of 108 models which includes the maximum size of real-

world FMs as well as automatically-generated 3-CNF-FMs (considered as the toughest benchmark in SPL) fom SPLOT repository and models generated using the FeatureIDE tool.

(viii) Improved performance and scalability: The results indicate that the proposed approach is effective, accurate and scalable. It can handle defects in large-sized FMs up to 30,000 features in a reasonable time of 0.551 seconds.

(ix) Improved SPL in terms of reusability and quality: Identified and resolved defects in the early development process of FM. It allows deriving defect free valid software products from PL by subsequently enhancing the reusability and quality of FMs in SPL.

To the best of our knowledge, the proposed approach is unique in handling defects, as it identified defects with their causes and provided corrections in a user-friendly natural language to resolve defects. It further enables PL developers to concentrate on creating well-formed FMs, instead of discovering a way to resolve defects in later stages of the software development process. It is worth noting that the proposed approach detected actual defective features and the cross-tree constraints for their causes and corrections, and not defective FMs. Additionally, correction provided by the proposed approach is minimal, as it targets the cross-tree constraints itself which are involved in the source of defect. The presented approach further classified and handled defects due to wrong cardinality in CBFMs.

6.2 Future work

The proposed approach provides several motivating directions for future work.

(i) Handling defects due to features and attributes: Currently, the defects arising from wrong usage of cross-tree constraints are handled. An improvement for handling other defects caused by the misuse of features, attributes, etc. can be considered as an open problem.

(ii) Improving corrections: The proposed approach only resolves defects by eliminating more than one cross-tree constraints. Consequently, an improvement in corrections includes adding or eliminating features in model to resolve defects.

(iii) **Dealing with other defects:** Presently, the defects due to redundancy, anomaly, inconsistency and wrong cardinality in FMs are handled. The future work includes dealing with other defects in FM such as false product line and implicit constraints.

(iv) **Extending set of rules**: The existing set of rules in the proposed approach could be maximized by accumulating new rules for additional cases of defects.

(v) **Providing support for other modeling notations:** The defects can be further handled in other modeling notations such as EFM, decision model, OVM and textual variability language.

(vi) **Other rules support**: Further, SWRL based rules can also be developed for the auto identification and correction of defects.

(vii) **Evaluating the scalability and performance:** Currently, the evaluation results lead to promising performance and scalability as compared to existing approaches. There is a scope to evaluate the framework with industrial and randomly generated large-sized FMs.

(viii) **Providing tool support to resolve defects.** The corrections can be further exploited, e.g. automatically eliminate the cross-tree constraints to resolve defects.

References

[1] P. Clements and L. Northrop (2001). Software Product Lines: Practices and Pattern, Addison-Wesley Professional.

[2] K. Schmid and I. John (2004). A customizable approach to full lifecycle variability management, Science of Computer Programming, 53(3), 259–284. DOI: 10.1016/j.scico.2003.04.002.

[3] D. Dhungana, P. Grünbacher and R. Rabiser (2011). The DOPLER meta-tool for decision oriented variability modeling: a multiple case study, Automated Software Engineering, 18(1), 77–114. DOI: http://dx.doi.org/10.1007/s10515-010-0076-6.

[4] K. Pohl, G. Böckle and F. Van Der Linden (2005). Software Product Line Engineering: Foundations, Principles and Techniques, Springer Science & Business Media.

[5] A. Classen, Q. Boucher and P. Heymans (2011). A text-based approach to feature modelling: syntax and semantics of TVL, Science of Computer Programming, 76(12), 1130–1143. DOI: 10.1016/j.scico. 2010.10.005.

[6] K.C. Kang, S.G. Cohen, J.A. Hess, W.E. Novak and A.S. Peterson (1990). Feature-Oriented Domain Analysis (FODA) feasibility study, Technical Report CMU/SEI-90-TR-21, ESD-90-TR-222, SEI.

[7] C. Prehofer (2001). Feature-oriented programming: A new way of object composition, Concurrency and Computation: Practice and Experience, 13(6), 465–501. DOI: 10.1002/cpe.583

[8] M. Bernardo, P. Ciancarini and L. Donatiello (2002). Architecting families of software systems with process algebras, ACM Transactions on Software Engineering and Methodology (TOSEM), 11(4), 386– 426. DOI: 10.1145/606612.606614

[9] S. Jarzabek, W.C. Ong and H. Zhang (2003). Handling variant requirements in domain modeling, The Journal of Systems and Software, 68(3),171–182. DOI: 10.1016/S0164-1212(03)00060-8

[10] L.M. Northrop (2008). Software product lines essentials.

[11] G. Deng, G. Lenz and D.C. Schmidt (2006). Addressing domain evolution

challenges in software product lines, Lecture Notes in Computer Science, 3844, 247-261. DOI: 10.1007/11663430_26

[12] N. Niu, J. Savolainen and Y. Yu (2010). Variability modeling for product line viewpoints integration, in Proceedings of the 2010 IEEE 34th Annual Computer Software and Applications Conference (COMPSAC), IEEE, 337–346.

[13] J. Van Gurp and C. Prehofer (2008). From SPLs to open, compositional platforms, in Dagstuhl Seminar, Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[14] G. Böckle and V. Frank (2005). Software product line engineering, Edited by Klaus Pohl, 10, Springer.

[15] M. Ardis, N. Daley, D. Hoffman, H. Siy and D. Weiss (2000). Software product lines: A case study, Software-Practice and Experience, 30(7), 825-47.

[16] D. D'souza, M. Gopinathan, S. Ramesh and P. Sampath (2010). Conflict-tolerant specifications for hybrid systems, Technical Report, IISc-CSA-TR-2010-6, Indian Institute of Science, India.

[17] C. Thao (2012). A configuration management system for software product lines.

[18] M. Mannion (2002). Using first-order logic for product line model validation, in Proceedings of the 2nd International Conference on Software Product Line Conference (SPLC2), San Diego, CA, USA, Springer Verlag, 176–187. DOI: 10.1007/3- 540- 45652- X _ 11.

[19] S. Fan and Z. Naixiao (2006). Feature model based on description logics, Knowledge-Based Intelligent Information and Engineering Systems, Springer, 4252, 1144-1151. DOI: 10.1007/11893004_145

[20] C. Salinesi, C. Rolland and R. Mazo (2009). VMWare: tool support for automatic verification of structural and semantic correctness in product line models, in Proceedings of the International Workshop on Variability Modelling of Software-intensive Systems (VaMoS09), 173–176.

[21] C. Salinesi, R. Mazo and D. Diaz (2010). Criteria for the verification of feature models, in Proceedings of the 28th INFORSID (INFormatique Des ORganisations et Syst`emes d'Information et de D´ecision), 293–308.

[22] J.K. Chhabra and A. Prajapati (2011). A Framework for Vulnerability Analysis during Software Maintenance, In : Mantri A., Nandi S., Kumar G., Kumar S. (eds) High Performance Architecture and Grid Computing, Communications in Computer and Information Science, 169, Springer, Berlin, Heidelberg.

[23] R. Gheyi, T. Massoni and P. Borba (2011). Automatically checking feature model refactorings, Journal of Universal Computer Science, 17(5), 684-71.

[24] S. Segura, R.M. Hierons, D. Benavides and A. Ruiz-Cortés (2011). Automated metamorphic testing on the analyses of feature models, Information and Software Technology, 53(3), 245-258. DOI: http://dx. doi.org/10.1016/j.infsof. 2010.11.002

[25] C. Salinesi and R. Mazo (2012). Defects in product line models and how to identify them, Abdelrahman Elfaki, Software Product Line - Advanced Topic, InTech editions. DOI: <hal-00707461>

[26] S. Ripon, S.J. Hossain and T. Bhuiyan (2013). Managing and analysing software product line requirements, International Journal of Software Engineering and Applications, 4, 63-75. DOI: 10.5121/ijsea.2013.4505

[27] X. Zhang and B. Møller-Pedersen (2012). Towards correct product derivation in model-driven product lines, in Proceedings of the 7th international conference on System Analysis and Modeling: Theory and Practice (SAM'12), Øystein Haugen, Rick Reed, and Reinhard Gotzhein (Eds.), Lecture Notes in Computer Science, Springer-Verlag, Berlin, Heidelberg, 7744, 179-197. DOI: 10.1007/978-3-642-36757-1_11

[28] D. Yang and M. Dong (2013). Applying constraint satisfaction approach to solve product configuration problems with cardinality-based configuration rules, Journal of Intelligent Manufacturing, 24, 99–111. DOI: 10.1007/s10845-011-0544-2

[29] M. Asadi, G. Gröner, B. Mohabbati and D. Gašević (2014). Goal-oriented modeling and verification of feature-oriented product lines, Software & Systems Modeling, 15, 257-279. DOI: 10.1007/s10270-014-04028

[30] A.O. Elfaki , S.L. Fong, P. Vijayaprasad, M.G.M. Johar and M.S. Fadhil (2014). Using a rule-based method for detecting anomalies in software product

line, Research Journal of Applied Sciences, Engineering and Technology, 7(2), 275-281.

[31] T. Thüm, J. Meinicke, F. Benduhn, M. Hentschel, A. Von Rhein and G. Saake (2014). Potential synergies of theorem proving and model checking for software product lines, in Proceedings of the 18th International Software Product Line Conference (SPLC'14), Florence, Italy, 177–186. DOI: 10.1145/2648511.2648530.

[32] H. Perez-Morago, R. Heradio, D. Fernandez-Amoros, R. Bean, and C. Cerrada (2015). Efficient identification of core and dead features in variability models, IEEE Access, 3, 2333-2340. DOI: 10. 1109/ACCESS.2015.2498764

[33] M. Javed, M. Naeem, A.I. Umar and F. Bahadur (2016). Automated inconsistency detection in feature models: A generative programming based approach, Selforganizology, 3(2), 59-74.

[34] H.H. Wang, Y.F. Li, J. Sun, H. Zhang and J. Pan (2007). Verifying feature models using OWL, Web Semantics: Science, Services and Agents on the World Wide Web, 6(2), 117–129. DOI: http://dx.doi.org/10.1016/j.websem.2006.11.006

[35] P. van den Broek and I. Galvão (2009). Analysis of feature models using generalised feature trees, in Proceedings of the International Workshop on Variability Modeling of Software-intensive Systems (VaMoS'09), 71–76. DOI:10.1.1.216.5377

[36] P. Trinidad and A. Ruiz-Cortés (2009). Abductive reasoning and automated analysis of feature models: How are they connected, in Proceedings of the International Workshop on Variability Modelling of Software-Intensive Systems, 145-153. DOI:10.13140/ 2.1.4955.0400

[37] L.A. Zaid, F. Kleinermann and O. De Troyer (2009). Applying semantic web technology to feature modelling, in Proceedings of ACM symposium on Applied Computing (SAC '09), ACM, 1252-1256. DOI:10.1145/1529282.1529563

[38] S. Segura, D. Benavides and A. Ruiz-Cortés (2010). FaMa Test Suite v1.2, Technical Report ISA-10-TR-01, Applied Software Engineering Research

Group, University of Seville, Spain.

[39] M. Noorian, A. Ensan, E. Bagheri, H. Boley and Y. Biletskiy (2011). Feature model debugging based on description logic reasoning, in Proceedings of the 17th International Conference on Distributed Multimedia Systems (DMS'11), 158–164.

[40] A. Felfernig, D. Benavides, J. Galindo and F. Reinfrank (2013). Towards anomaly explanations in feature models, in Proceedings of the 15th International Configuration Workshop (ConfWS-2013), Vienna, Austria, 117-124. DOI: 10.1.1.428.5517

[41] G. Zhang, H. Ye and Y. Lin (2013). An approach for validating feature models in software product lines, Journal of Software Engineering, 7(1), 1-29. DOI:10.3923/jse.2013.1.29

[42] U. Lesta, I. Schaefer and T. Winkelmann (2015). Detecting and explaining conflicts in attributed feature models, in Proceedings of the 6th Workshop on Formal Methods and Analysis in SPL Engineering (FMSPLE 2015), 31–43. DOI: https://arxiv.org/abs/1504.03483

[43] L.F. Rincón, G.L. Giraldo, R. Mazo and C. Salinesi (2014). An ontological rule-based approach for analyzing dead and false optional features in feature models, Electronic notes in Theoretical Computer Science 302, 111–132. DOI: 10.1016/j.entcs.2014.01.023

[44] T. Thüm, C. Kästner, F. Benduhn, J. Meinicke, G. Saake and T. Leich (2014). FeatureIDE: an extensible framework for feature-oriented software development, Science of Computer Programming, 79, 70-85. DOI: 10.1016/j.scico.2012.06.002

[45] A.O. Elfaki (2016). A rule-based approach to detect and prevent inconsistency in the domain-engineering process, Expert Systems, 33(1), 3–13. DOI: 10.1111/exsy.12116

[46] D. Benavides, S. Segura and A. Ruiz-Cortés (2010). Automated analysis of feature models 20 years later: a literature review, Information Systems Journal, 35(6), 615–636.

[47] T. Von der Maßen and H. Lichter (2004). Deficiencies in feature models, in

Proceedings of the Workshop on Software Variability Management for Product Derivation - Towards Tool Support, 59-62.

[48] K. Czarnecki, S. Helsen and U. Eisenecker (2005). Formalizing cardinality-based feature models and their specialization, Software Process: Improvement and Practice, 10(1), 7–29. DOI: 10.1002/spip.213

[49] P. Trinidad, D. Benavides, A. Duran, A. Ruiz-Cortés and M. Toro (2008a). Automated error analysis for the agilization of feature modelling, Journal of Systems and Software, 81(6), 883–896. DOI: 10.1016/j.jss.2007.10.030

[50] M. Alférez, J. Santos, A. Moreira, A. Garcia, U. Kulesza, J. Araújo and V. Amaral (2010). Multi-view composition language for software product line requirements, Book Chapter in Software Language Engineering, ISSN 0302-9743, Springer Berlin, Heidelberg, 103–122.

[51] E. Tsang (1993). Foundations of Constraint Satisfaction: The Classic Text, Academic Press, London.

[52] I. Groher and M. Voelter (2007). Expressing feature-based variability in structural models, in Proceedings of the Workshop on Managing Variability for Software Product Lines, Co-located with the 7th Software Product Line Conference (SPLC), Kyoto, Japan.

[53] M. Janota and J. Kiniry (2007). Reasoning about feature models in higher-order logic, in Proceedings of the 11th International Software Product Line Conference (SPLC07), Heidelberg, Germany.

[54] F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi and P.F. Patel- Schneider (2003). The Description Logic Handbook: Theory Implementation and Applications, Cambridge University Press, New York, NY, USA.

[55] K. Czarnecki (1998). Generative programming: principles and techniques of software engineering based on automated configuration and fragment-based component models, PhD Dissertation, Technical University of Ilmenau, Germany.

[56] M. Mernik, J. Heering and A.M. Sloane (2005). When and how to develop domain-specific languages, ACM Computing Surveys, 37(4), 316–344.

[57] I.M. Copi and C. Cohen (2009). Introduction to Logic, Pearson Prentice Hall,

New Jersey, USA.

[58] J. Sun, H. Zhang, Y.F. Li and H. Wang (2005). Formal semantics and verification for feature modeling, in Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS05), IEEE Computer Society, Washington DC, USA, 303–312. DOI: 10.1109/ICECCS.2005.48.

[59] A.O. Elfaki, S. Phon-Amnuaisuk and C.K. Ho (2009b). Using first order logic to validate feature model, in Proceedings of the the 3rd International Workshop on Variability Modeling of Software-Intensive Systems, Sevilla, Spain.

[60] S.H. Ripon, S.J. Hossain and M.M. Piash (2014). Logic-based analysis and verification of software product line variant requirement model. International Journal of Knowledge and Systems Science (IJKSS), 5(4), 52-76. DOI: 10.4018/ijkss.2014100104

[61] M.F.M.A. Bari and M. Akter (2014). Logic-based verification of software product line feature model, Technical Report, East West University, Dhaka, Bangladesh. DOI: http://hdl.handle.net/123456789/1560

[62] J.H. Gallier (2003). Logic For Computer Science Foundations of Automatic Theorem Proving, University of Pennsylvania, Philadelphia, Pa, USA.

[63] T.R. Gruber (1993). A translation approach to portable ontology specifications, Knowledge Acquisition, 5(2), 199-220. DOI: 10.1006/knac.1993.1008

[64] S. Lee, J. Kim, C. Song and D. Baik (2007). An approach to analyzing commonality and variability of features using ontology in a software product line engineering, in Proceedings of the 5th International Conference on Software Engineering Research, Management and Applications, IEEE, Busan, 727–734. DOI:10.1109/SERA.2007.41.

[65] L. Abo, G. Houben, O.D. Troyer and F. Kleinermann (2008). An OWL-based approach for integration in collaborative feature modelling, in Proceedings of the 4th Workshop on Semantic Web Enabled Software Engineering (SWESE2008), Germany, 93–100.

[66] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari and L. Schneider (2002). Sweetening ontologies with DOLCE, in Proceedings of the 13th International

Conference on Knowledge Engineering and Knowledge Management, Ontologies and the Semantic Web (EKAW 2002), Berlin Heidelberg, Springer, 166–181.

[67] C. Matuszek, J. Cabral, M.J. Witbrock and J. Deoliveira (2006). An introduction to the syntax and content of Cyc, in Proceedings of the AAAI Spring Symposium: Formalizing and Compiling Background Knowledge and its Applications to Knowledge Representation and Question Answering, 44–49.

[68] I. Niles and A. Pease (2001). Towards a standard upper ontology, in Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS'01), ACM, New York, NY, USA, 2001, 2–9. DOI: http://dx.doi.org/10.1145/505168.505170

[69] M. Gruninger, R. Hull and S.A. Mcilraith (2008). A short overview of FLOWS: A first-order logic ontology for web services, Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 31(3), 3–7.

[70] D. Berardi, M. Grüninger, R. Hull and S. Mcilraith (2004). Towards a first-order ontology for semantic web services, in Proceedings of the W3C Workshop on Constraints and Capabilities for Web Services.

[71] M. Schneider, J. Carroll, I. Herman and P.F. Patel-Schneider (2009). OWL 2 Web Ontology Language: RDF-Based Semantics, W3C Recommendation.

[72] D. Tsarkov, A. Riazanov, S. Bechhofer and I. Horrocks (2004). Using vampire to reason with OWL, in Proceedings of the 3rd International Semantic Web Conference (ISWC 2004), Springer Berlin Heidelberg, 471–485.

[73] R.C. Goldstein and V.C. Storey (1991). Database and expert systems applications, in Proceedings of the International Conference in Berlin, Springer, Vienna Wien GmbH, Federal Republic of Germany, 124–129. DOI:10.1007/ 978-3-7091-7555-2_21.

[74] J.D. Bruijn and S. Heymans (2006). Translating ontologies from predicate-based to frame-based languages, in Proceedings of the 2nd International Conference on Rules and Rule Markup Languages for the Semantic Web, IEEE Computer Society, Washington, DC, USA, 7–16. DOI:10.1109/RULEML. 2006.23.

[75] M. Gruninger and C. Menzel (2003). The process specification language (PSL) theory and applications, AI magazine, 24(3), 63–74.

[76] A. Pease and G. Sutcliffe (2007). First-order reasoning on a large ontology, in Proceedings of the Workshop on Empirically Successful Automated Reasoning in Large Theories (CADE-21), CEUR Workshop, Bremen, Germany, 257, 59-69.

[77] M. Schneider and G. Sutcliffe (2011). Reasoning in the OWL2 full ontology language using first-order automated theorem proving, in Proceedings of the 23rd International Conference on Automated Deduction (CADE-23), Springer Berlin Heidelberg, 461–475.

[78] D. Ramachandran, R.P. Reagan and K. Goolsbey (2005). First-orderized ResearchCyc: Expressivity and efficiency in a common-sense ontology, In P. Shvaiko, J. Euzenat, A. Leger, D. L. McGuinness, & H. Wache (Eds.), in Proceedings of the AAAI 2005 Workshop on Contexts and Ontologies: Theory, Practice and Applications, AAAI Press, 33-40.

[79] J. Àlvez, P. Lucio and G. Rigau (2012). Adimen-SUMO: reengineering an ontology for first-order reasoning, International Journal on Semantic Web and Information Systems, 8(4), 80–116. DOI: 10.4018/ jswis.20121001, 05.

[80] P. Baumgartner and F.M. Suchanek (2006). Automated reasoning support for first-order ontologies, in Proceedings of the International Workshop on Principles and practice of semantic web reasoning, Springer Berlin Heidelberg, 18–32.

[81] P. Trinidad, A. Benavides and A. Ruiz-Cortés (2006). Isolated features detection in feature model, in Proceedings of the Conference on Advanced Information Systems Engineering (CAiSE'06) Forum, Luxembour. DOI:http://www.ceur-ws.org/Vol-231/Paper19.pdf

[82] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura S and A. Jimenez (2008b). FAMA framework, in Proceedings of the 12th International Software Product Line Conference (SPLC'12), IEEE Computer Society, 359. DOI:10.1109/ SPLC.2008.50

[83] A. Hemakumar (2008). Finding contradictions in feature models, in Steffen

Thiel & Klaus Pohl, ed., SPLC (2), Lero Int. Science Centre, University of Limerick, Ireland, 183-190.

[84] A. Osman, S.P. Amnuaisuk and C.K. Ho (2008). Knowledge based method to validate feature models, in Proceedings of the 12th International Conference Software Product Lines Conference (SPLC 2008), Limerick, Ireland, 217–225.

[85] A. Elfaki, S. Phon-Amnuaisuk and C.K. Ho (2009a). Investigating inconsistency detection as a validation operation in software product line, Studies in Computational Intelligence, Springer, 253, 159–168. DOI: 10.1007/978-3-642-05441-9_14

[86] T. Thüm, D. Batory and C. Kästner (2009). Reasoning about edits to feature models, in Proceedings of the 31th International Conference on Software Engineering (ICSE'09), IEEE Computer Society, Washington, DC, USA, 254-264. DOI:10.1109/ICSE.2009.5070526 http://dx.doi.org/10.1109/ICSE.2009.5070526

[87] E. Sirin, B. Parsia, B.C. Grau, A. Kalyanpur and Y. Katz (2007). Pellet: A practical OWL-DL reasoned, Web Semantics: Science, Services and Agents on the World Wide Web, 5, 51–53. DOI:10.1016/j.websem.2007.03.004

[88] D. Turner (1985). Miranda: a non-strict functional language with polymorphic types, in Proceedings of the conference on Functional Programming Languages and Computer Architecture, Lecture Notes in Computer Science, 201, Jean-Pierre Jouannaud (Ed.), Springer-Verlag, New York, NY, USA, 1-16.

[89] J. White, B. Dougherty, D.C. Schmidt and D. Benavides (2009). Automated reasoning for multi-step feature model configuration problems, in Proceedings of the 13th International Software Product Line Conference (SPLC'09), Carnegie Mellon University, Pittsburgh, PA, USA, 11-20.

[90] J. White, D. Benavides, D.C. Schmidt, P. Trinidad, B. Dougherty and A. Ruiz-Cortés (2010). Automated diagnosis of feature model configurations, Journal of Systems and Software, 83(7), 1094–1107. DOI: 10.1016/j.jss.2010.02.017

[91] B. Wang, Y. Xiong, Z. Hu, H. Zhao, W. Zhang and H. Mei (2010). A dynamic-priority based approach to fixing inconsistent feature models, in Proceedings of the 13th international conference on Model driven engineering languages and

systems: Part I(MODELS'10), Dorina C. Petriu, Nicolas Rouquette, and Øystein Haugen (Eds.), Oslo, Norway, Springer-Verlag, 181-195.

[92] R. Mazo (2011). A generic approach for automated verification of product line models, PhD Dissertation, Université Panthéon-Sorbonne-Paris I.

[93] J. Guo, Y. Wang, P. Trinidad and D. Benavides (2012). Consistency maintenance for evolving feature models, Expert Systems with Applications, 39(5), 4987–4998. DOI: 10.1016/j.eswa.2011.10.014

[94] R. Reiter (1987). A theory of diagnosis from first principles, Artificial intelligence, 32(1), 57-95. DOI: 10.10 16/0004-3702(87)90062-2

[95] D. Jackson (2002). Alloy: A lightweight object modelling notation. ACM Transactions on Software Engineering and Methodology (TOSEM), 11(2), 256–290. DOI=http://dx.doi.org/10.1145/505145.505 149

[96] A.O. Elfaki, S.L.Fong, K.L.T. Aik and M.G.M. Johar (2013). Towards detecting redundancy in domain engineering process using first order logic rules, International Journal of Knowledge Engineering and Soft Data Paradigms, 4(1), 1-20. DOI: http://dx.doi.org/10.1504/IJKESDP.2013.052716

[97] L.G. Giraldo, L. Rincón-Perez and Raul Mazo (2013). Identifying dead features and their causes in product line models: an ontological approach, Revista DYNA, 81, 68–77. DOI: http://dx.doi.org/10. 15446/dyna.v81n183.36348

[98] S. Segura, J.A. Galindo, D. Benavides, J.A. Parejo and A. Ruiz-Cortés (2012). BeTTy: Benchmarking and testing on the automated analysis of feature models, in Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems (VaMoS'12), ACM, New York, NY, USA, 63-71. DOI : http://doi.acm.org /10.1145 /2110147.2110155

[99] M. Javed, M. Naeem and H.A. Wahab (2014). Towards the maturity model for feature oriented domain analysis, Computational Ecology and Software, 4(3), 170-182.

[100] J. White, J.A. Galindo, T. Saxena, B. Dougherty, D. Benavides and D.C. Schmidt (2014). Evolving feature model configurations in software product lines, Journal of Systems and Software, 87, 119-136. DOI: http://dx.doi.org/10. 1016/j.jss.2013.10.010

[101] L. Rincón, G.L. Giraldo, R. Mazo, C. Salinesi and D. Diaz (2015). Method to identify corrections of defects on product line models, Electronic Notes in Theoretical Computer Science, 314, 61–81. DOI:10.1016/j.entcs.2015.05.005

[102] M. Kowal, S. Ananieva and T. Thüm (2016). Explaining anomalies in feature models, in Proceedings of the 2016 ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE 2016), ACM, New York, NY, USA, 132-143. DOI: https://doi.org/10.1145/2993236.2993248

[103] M. Mendonca, M. Branco and D. Cowan (2009). 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), ACM, New York, USA, 761-762. DOI: 10.1145/1639950.1640002

[104] M. Stephan and M. Antkiewicz (2008). Ecore.fmp: A tool for editing and instantiating class models as feature models, Technical Report, University of Waterloo.

[105] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick and T.J. Grose (2003). Eclipse modeling framework, Addison-Wesley Professional.

[106] N. Gamez and L. Fuentes (2013). Architectural evolution of FamiWare using cardinality-based feature models, Information and Software Technology, 55(3), 563-580. DOI: 10.1016/j.infsof.2012.06.012

[107] D. Benavides, S. Segura, P. Trinidad and A. Ruiz-Cortés (2007). FAMA: Tooling a framework for the automated analysis of feature models, in Proceedings of the 1st International Workshop on Variability Modelling of Software Intensive Systems, 16-18, Limerick, Ireland.

[108] T. Leich, S. Apel, L. Marnitz and G. Saake (2005). Tool support for feature-oriented software development: featureIDE: an Eclipse-based approach, in Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange (eclipse'05), ACM, New York, NY, USA, 55-59. DOI: 10.1145/1117696.1117708

[109] J. Bagherzadeh and S. Arun-Kumar (2005). Layered clausal resolution in the

multi-modal logic of beliefs angoals, Lecture Notes in Computer Science, 3452, 544–559.

[110] S. Sahni and R. Kumar (2003). Software Development in Java, Silicon Press, New Jersey, 462 pages.

[111] S. Kaushik (2007). Logic and Prolog Programming, New Age International (P) Ltd., publisher, ISBN 8122414095, 9788122414097, 340 pages.

[112] J. Wielemaker (2015). SWI-Prolog (Version 7.2.3), free software, Amsterdam, VU University Amsterdam, University of Amsterdam. Available at: http://www.swi-prolog.org

[113] W. Zhang, H. Zhao and H. Mei (2004). A propositional logic-based method for verification of feature models, in Proceedings of the 6th International Conference on Formal Engineering Methods, (ICFEM 2004), Seattle, WA, USA, Springer Berlin, Heidelberg, 115–130. DOI: http://dx.doi.org/10.1007/978-3-540-30482-1_16

[114] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura and M. Toro (2006). Explanations for agile feature models, in Proceedings of the 1st International Workshop on Agile Product Line Engineering (APLE'06), Baltimore, Maryland, U.S.A. DOI: http://dx.doi.org /10.1.1.333.3310

[115] R. Mazo, R. Lopez-Herrejon, C. Salinesi, D. Diaz and A. Egyed (2011). Conformance checking with constraint logic programming: The case of feature models, in Proceedings of the 35th Annual International Computer Software and Applications Conference (COMPSAC), IEEE Press, 456-465 Best Paper Award, Munich-Germany.

[116] A.O. Elfaki, O.A. Abouabdalla, S.L. Fong, M.G.M. Johar, K.L.T. Aik and R. Bachok (2012). Review and future directions of the automated validation in software product line engineering, Journal of Theoretical and Applied Information Technology, 42(1), 75-93. DOI: http://dx.doi.org/10.1.1.299. 7749

[117] C. Zhou, W. Xiao, T.M. Tirpak and P. C. Nelson (2003). Evolving accurate and compact classification rules with gene expression programming, IEEE Transactions on Evolutionary Computation, 7, 519–531.

[118] I. Bruha (1997). Quality of decision rules: Definitions and classification

Schemes for multiple rules, In G. Nakhaeizadeh and C.C. Taylor (Eds.), Machine Learning and Statistics: The Interface, John Wiley, New York, 107–131.

[119] R.S. Michalski and K.A. Kaufman (2001). Learning patterns in noisy data: The AQ approach, In: Paliouras G., Karkaletsis V., Spyropoulos C.D. (eds) Machine Learning and Its Applications, ACAI 1999, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2049, 22–38.

Appendices

Appendix A: XML parser

```
package test;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.util.ArrayList;
import java.util.List;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

public class Transformer {
	static List<String> pList = new ArrayList<>();
	public static void main(String[] args) {
		int exc=0,imp=0;
		try {
			File fXmlFile = new File("E-commercesystem1.xml");
			DocumentBuilderFactory dbFactory =
			DocumentBuilderFactory.newInstance();
			DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
			Document doc = dBuilder.parse(fXmlFile);
			doc.getDocumentElement().normalize();
			NodeList nList4 = doc.getElementsByTagName("and");
			for (int temp4 = 0; temp4 < nList4.getLength(); temp4++) {
				Node nNode4 = nList4.item(temp4);
```

```
            if (nNode4.getNodeType() == Node.ELEMENT_NODE) {
            Element e4 =  (Element) nNode4;
            if(e4.getAttribute("mandatory").equals("true")&&
            !e4.getParentNode().getNodeName().equals("alt")&&
            !e4.getParentNode().getNodeName().equals("or"))
            {
                  pList.add("f(" + e4.getAttribute("name") + ",m).");
            }
            else
            {
                  pList.add("f(" + e4.getAttribute("name") + ",o).");
            }
            }
}
nList4 = doc.getElementsByTagName("alt");
 for (int temp4 = 0; temp4 < nList4.getLength(); temp4++) {
            Node nNode4 = nList4.item(temp4);
            if (nNode4.getNodeType() == Node.ELEMENT_NODE) {
            Element e4 =  (Element) nNode4;
            if(e4.getAttribute("mandatory").equals("true")&&
            !e4.getParentNode().getNodeName().equals("alt")&&
            !e4.getParentNode().getNodeName().equals("or"))
            {
                  pList.add("f(" + e4.getAttribute("name") + ",m).");
            }
            else
            {
                  pList.add("f(" + e4.getAttribute("name") + ",o).");
            }
            }
}
```

```
nList4 = doc.getElementsByTagName("or");
for (int temp4 = 0; temp4 < nList4.getLength(); temp4++) {
        Node nNode4 = nList4.item(temp4);
//      count=count+nNode4.getChildNodes().getLength();
//System.out.println((nNode4.getChildNodes().getLength()));
        if (nNode4.getNodeType() == Node.ELEMENT_NODE) {
        Element e4 =  (Element) nNode4;
//      countNode(e4);
        if(e4.getAttribute("mandatory").equals("true")&&
        !e4.getParentNode().getNodeName().equals("alt")&&
        !e4.getParentNode().getNodeName().equals("or"))
        {
                pList.add("f(" + e4.getAttribute("name") + ",m).");
        }
        else
        {
                pList.add("f(" + e4.getAttribute("name") + ",o).");
        }
        }
}
nList4 = doc.getElementsByTagName("feature");
for (int temp4 = 0; temp4 < nList4.getLength(); temp4++) {
        Node nNode4 = nList4.item(temp4);
        if (nNode4.getNodeType() == Node.ELEMENT_NODE) {
        Element e4 =  (Element) nNode4;
        if(e4.getAttribute("mandatory").equals("true")&&
        !e4.getParentNode().getNodeName().equals("alt")&&
        !e4.getParentNode().getNodeName().equals("or"))
        {
              pList.add("f(" + e4.getAttribute("name") + ",m).");
         }
```

```
                else
                {
                        pList.add("f(" + e4.getAttribute("name") + ",o).");
                }
                }
        }
        NodeList nList7 = doc.getElementsByTagName("struct");
        for (int temp7 = 0; temp7 < nList7.getLength(); temp7++) {
                Node nNode7 = nList7.item(temp7);
         if (nNode7.getNodeType() == Node.ELEMENT_NODE) {
                        NodeList nList8 = nNode7.getChildNodes();
         for (int temp8 = 0; temp8 < nList8.getLength(); temp8++) {
                                Node nNode8 = nList8.item(temp8);
                if (nNode8.getNodeType() == Node.ELEMENT_NODE) {
                                        Element e8 =  (Element) nNode8;
                                                if (e8.hasChildNodes())
                                        printNote(e8.getChildNodes());
                                }
                        }
                }
        }
        NodeList nList = doc.getElementsByTagName("imp");
        for (int temp = 0; temp < nList.getLength(); temp++) {
                Node nNode = nList.item(temp);
                if (nNode.getNodeType() == Node.ELEMENT_NODE) {
                Element e =  (Element) nNode;
                imp++;
pList.add("i("+e.getElementsByTagName("var").item(0).getTextContent() + "," +
 e.getElementsByTagName("var").item(1).getTextContent() + ",imp"+imp+").");
                }
        }
```

```
// Case: Dsj All cases
                NodeList disjList = doc.getElementsByTagName("disj");
                for (int temp14 = 0; temp14 < disjList.getLength(); temp14++)
                {
                                Node nNode14 = disjList.item(temp14);
                                 if (nNode14.getNodeType() == Node.ELEMENT_NODE)
                                {
                                                Element e =  (Element) nNode14;
            Element et = (Element) e.getParentNode();
                                if(et.getNodeName().equals("rule")){
                                Object ParentOfFirstElementInDisj =
e.getElementsByTagName("var").item(0).getParentNode().getNodeName();
                                                Object ParentOfSecondElementInDisj =
e.getElementsByTagName("var").item(1).getParentNode().getNodeName();

                                                //now match ParentOfFirstElementInDisj and
ParentOfSecondElementInDisj to "disj" or "not"
                                //we match ParentOfFirstElementInDisj to "disj",
                                //if true then ParentOfFirstElementInDisj is "disj" => A
                                //else ParentOfFirstElementInDisj is "not" => -A
                                //now for ParentOfSecondElementInDisj compared to "disj"
                                //if true then ParentOfSecondElementInDisj is "disj" => B
                                //else ParentOfSecondElementInDisj is "not" => -B
                                //-AvB
if(!(ParentOfFirstElementInDisj.equals("disj"))&&(ParentOfSecondElementInDisj.equal
s("disj")))
                                                {
                                                                imp++;
                                                                pList.add("i(" +
e.getElementsByTagName("var").item(0).getTextContent()+ ","+
e.getElementsByTagName("var").item(1).getTextContent() + ",imp"+imp+").");
```

```
                                        }
//-Av-B, exclusion case
elseif(!(ParentOfFirstElementInDisj.equals("disj"))&&!(ParentOfSecondElementInDisj.equals("disj")))
                                        {
                                                exc++;
pList.add("e(" + e.getElementsByTagName("var").item(0).getTextContent()+ ","+ e.getElementsByTagName("var").item(1).getTextContent() + ",exc"+exc+").");
                                        }
//AvB
elseif((ParentOfFirstElementInDisj.equals("disj"))&&(ParentOfSecondElementInDisj.equals("disj")))
                                        {
                                                imp++;
                                                pList.add("i(" + e.getElementsByTagName("var").item(0).getTextContent()+ ","+ e.getElementsByTagName("var").item(1).getTextContent() + ",imp"+imp+").");
                                        }
//Av-B
elseif((ParentOfFirstElementInDisj.equals("disj"))&&!(ParentOfSecondElementInDisj.equals("disj")))
                                        {
                                                imp++;
pList.add("i(" + e.getElementsByTagName("var").item(0).getTextContent()+ ","+
        e.getElementsByTagName("var").item(1).getTextContent() + ",imp"+imp+").");
                                        }
//wrong input
                                        else
                                        {
                                                System.out.println("Unknown condition");
                                        }
```

```
                    }
                }
            }
            NodeList nList2 = doc.getElementsByTagName("eq");
                for (int temp2 = 0; temp2 < nList2.getLength(); temp2++) {
                    Node nNode2 = nList2.item(temp2);
                    if (nNode2.getNodeType() == Node.ELEMENT_NODE) {
                    Element e2 =  (Element) nNode2;
                    exc++;
pList.add("e(" + e2.getElementsByTagName("var").item(0).getTextContent() + "," +
        e2.getElementsByTagName("var").item(1).getTextContent()+ ",exc"+exc+").");
                    }
                }
                    NodeList nList3 = doc.getElementsByTagName("alt");
                    for (int temp3 = 0; temp3 < nList3.getLength(); temp3++) {
                    Node nNode3 = nList3.item(temp3);
                    if (nNode3.getNodeType() == Node.ELEMENT_NODE) {
                    Element e3 =  (Element) nNode3;
pList.add("c(" + e3.getAttribute("name") + ",[" + getCardString1(e3) + "],[1,1]).");
                    }
                }
                NodeList nList10 = doc.getElementsByTagName("or");
                for (int temp10 = 0; temp10 < nList10.getLength(); temp10++) {
                    Node nNode10 = nList10.item(temp10);
                    if (nNode10.getNodeType() == Node.ELEMENT_NODE)
{
                    Element e10 =  (Element) nNode10;
pList.add("c(" + e10.getAttribute("name") + ",[" + getCardString1(e10) + "],[1,"+count
Node(e10)+"]).");
                    }
                }
```

```
                File outputFile = new File("output.pl");
                BufferedWriter br = new BufferedWriter(new FileWriter(outputFile));
                        for(String s:pList){
                                System.out.println(s.toLowerCase());
                                br.append(s.toLowerCase());
                                br.append(System.lineSeparator()); //(for Java 1.7 and 1.8)
        //              br.append(System.getProperty("line.separator")); // (Java 1.6 and below)
                        }
                        br.flush();
                        br.close();
                 }catch(Exception e){
                         e.printStackTrace();
                 }
        }
//getting immediate child nodes name
        private static String getCardString1(Element e){
                StringBuilder res = new StringBuilder();
                NodeList nodes = e.getChildNodes();
                for (int i = 0; i < nodes.getLength(); i++) {
                  Node node = nodes.item(i);
                  if (node instanceof Element) {
                    Element childElement = (Element) node;
                    res.append(childElement.getAttribute("name"));
                    res.append(",");
                  }
                }

                res.deleteCharAt(res.length()-1);
                return res.toString();
        }
//counting immediate child nodes
```

```
        private static int countNode(Element e){
                int i=0, count=0, l=0;
                String itemString;
                if(e.hasChildNodes()){ //just
                l=e.getChildNodes().getLength();
                for(i=0;i<l;i++){
                        Node item = e.getChildNodes().item(i);
                        itemString=item.toString();
                        /*if (item.getNodeType()==Node.ELEMENT_NODE) {
                                ++count;
                        }*/
                        if (itemString.matches("(.*)and(.*)")||
                                itemString.matches("(.*)or(.*)")||
                                itemString.matches("(.*)alt(.*)")||
                                itemString.matches("(.*)feature(.*)")) {
                                ++count;
                        }
                        }
                }
        return count;
}
private static void printNote(NodeList nodeList) {
   for (int count = 0; count < nodeList.getLength(); count++) {
        Node tempNode = nodeList.item(count);

        if (tempNode.getNodeType() == Node.ELEMENT_NODE) {
                if (tempNode.hasAttributes()) {
                        NamedNodeMap nodeMap = tempNode.getAttributes();
                        for (int i = 0; i < nodeMap.getLength(); i++) {
                                Node node = nodeMap.item(i);
                                if(node.getNodeName().equals("name")){
```

```
                                    Element p = (Element) tempNode.getParentNode();
                pList.add("p(" + node.getNodeValue() +","+ p.getAttribute("name")+").");
                                    }
                            }
                        }
                        if (tempNode.hasChildNodes()) {
                            // loop again if has child nodes
                            printNote(tempNode.getChildNodes());
                        }
                    }
                 }
                 }
            }
```

Appendix B: E-commercesystem1

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
    <featureModel chosenLayoutAlgorithm="1">
        <struct>
            <and mandatory="true" name="E_commercesystem">
                <and name="Manufacturerscatalogue">
                <and mandatory="true" name="Manufacturercategories">
                        <alt mandatory="true" name="a1">
                        <feature mandatory="true" name="a2"/>
                        <feature mandatory="true" name="a3"/>
                        </alt>
                        <alt mandatory="true" name="a4">
                            <and mandatory="true" name="a5">
                            <and mandatory="true" name="a7">
                        <feature mandatory="true" name="a8"/>
```

```
                    </and>
                </and>
            <feature mandatory="true" name="a6"/>
            </alt>
            <and mandatory="true" name="a9">
            <feature mandatory="true" name="a10"/>
                <feature name="a11"/>
            </and>
</and>
<and mandatory="true" name="Manufacturer">
            <and mandatory="true" name="Title">
            <and mandatory="true" name="a12">
                    <and name="a13">
                    <feature name="a15"/>
                    </and>
            <and mandatory="true" name="a14">
            <feature mandatory="true" name="a16"/>
            <feature name="a17"/>
                    </and>
                </and>
            <and mandatory="true" name="a18">
            <feature name="a19"/>
            <and mandatory="true" name="a20">
            <feature mandatory="true" name="a21"/>
                    </and>
                </and>
            <and mandatory="true" name="a22">
            <feature name="a24"/>
                </and>
            <and mandatory="true" name="a23">
            <feature mandatory="true" name="a25"/>
```

```
            </and>
  </and>
  <feature name="Photos"/>
<and mandatory="true" name="Description">
<and mandatory="true" name="a26">
  <feature mandatory="true" name="a27"/>
  <feature mandatory="true" name="a28"/>
            </and>
  <and mandatory="true" name="a29">
  <and mandatory="true" name="a30">
  <feature mandatory="true" name="a32"/>
                    </and>
                    <feature name="a31"/>
            </and>
  </and>
  <and mandatory="true" name="Products">
  <and mandatory="true" name="a38">
  <and mandatory="true" name="a39">
  <feature mandatory="true" name="a41"/>
                    </and>
                    <and name="a40">
  <feature mandatory="true" name="a42"/>
                    </and>
            </and>
  <and mandatory="true" name="a43">
  <and mandatory="true" name="a44">
  <feature mandatory="true" name="a46"/>
                    </and>
  <and mandatory="true" name="a45">
  <feature mandatory="true" name="a47"/>
                    </and>
```

```
                    </and>
                </and>
                <and mandatory="true" name="a33">
                <and mandatory="true" name="a34">
                <feature mandatory="true" name="a36"/>
                    </and>
                    <and name="a35">
                <feature mandatory="true" name="a37"/>
                    </and>
                </and>
            </and>
        </and>
        <and mandatory="true" name="Productscatalogue">
            <and mandatory="true" name="Productcategories">
                <and mandatory="true" name="a48">
                <and mandatory="true" name="a49">
                <feature mandatory="true" name="a51"/>
                    </and>
                    <feature name="a50"/>
                </and>
                <and mandatory="true" name="a52">
                <and mandatory="true" name="a53">
                <feature mandatory="true" name="a55"/>
                    </and>
                    <and name="a54">
                <feature mandatory="true" name="a56"/>
                    </and>
                </and>
                <feature name="a61"/>
                <feature mandatory="true" name="a58"/>
            </and>
```

```
                    <and mandatory="true" name="Productitems">
                        <feature mandatory="true" name="a60"/>
                    </and>
                    <and mandatory="true" name="Productvariations">
                        <feature name="a62"/>
                        <feature mandatory="true" name="a59"/>
                        <and mandatory="true" name="a63">
                        <and mandatory="true" name="a64">
                        <feature mandatory="true" name="a65"/>
                            </and>
                        </and>
                        <and mandatory="true" name="a66">
                        <and mandatory="true" name="a67">
                        <feature mandatory="true" name="a68"/>
                            </and>
                        </and>
                    </and>
                    <feature mandatory="true" name="a57"/>
                </and>
            </and>
        </struct>
        <constraints>
            <rule>
                <eq>
                    <var>a2</var>
                    <var>a3</var>
                </eq>
            </rule>
            <rule>
                <eq>
                    <var>a8</var>
```

<var>a6</var>
</eq>
</rule>
<rule>
<imp>
<var>a11</var>
<var>a10</var>
</imp>
</rule>
<rule>
<imp>
<var>a13</var>
<var>a16</var>
</imp>
</rule>
<rule>
<imp>
<var>a19</var>
<var>a20</var>
</imp>
</rule>
<rule>
<imp>
<var>a20</var>
<var>a21</var>
</imp>
</rule>
<rule>
<imp>
<var>a24</var>
<var>a25</var>

```
        </imp>
</rule>
<rule>
        <imp>
                <var>a26</var>
                <var>a27</var>
        </imp>
</rule>
<rule>
        <imp>
                <var>a26</var>
                <var>a28</var>
        </imp>
</rule>
<rule>
        <imp>
                <var>a30</var>
                <var>a31</var>
        </imp>
</rule>
<rule>
        <imp>
                <var>a32</var>
                <var>a31</var>
        </imp>
</rule>
<rule>
        <imp>
                <var>a36</var>
                <var>a35</var>
        </imp>
```

```
</rule>
<rule>
        <imp>
                <var>a36</var>
                <var>a37</var>
        </imp>
</rule>
<rule>
        <imp>
                <var>a39</var>
                <var>a40</var>
        </imp>
</rule>
<rule>
        <imp>
                <var>a41</var>
                <var>a42</var>
        </imp>
</rule>
<rule>
        <imp>
                <var>a46</var>
                <var>a45</var>
        </imp>
</rule>
<rule>
        <imp>
                <var>a46</var>
                <var>a47</var>
        </imp>
</rule>
```

```
<rule>
        <eq>
                <var>a49</var>
                <var>a50</var>
        </eq>
</rule>
<rule>
        <eq>
                <var>a51</var>
                <var>a50</var>
        </eq>
</rule>
<rule>
        <eq>
                <var>a53</var>
                <var>a54</var>
        </eq>
</rule>
<rule>
        <eq>
                <var>a55</var>
                <var>a56</var>
        </eq>
</rule>
<rule>
        <imp>
                <var>a57</var>
                <var>a58</var>
        </imp>
</rule>
<rule>
```

<imp>
<var>a58</var>
<var>a59</var>
</imp>
</rule>
<rule>
<imp>
<var>a59</var>
<var>a57</var>
</imp>
</rule>
<rule>
<imp>
<var>a60</var>
<var>a61</var>
</imp>
</rule>
<rule>
<imp>
<var>a61</var>
<var>a62</var>
</imp>
</rule>
<rule>
<imp>
<var>a60</var>
<var>a62</var>
</imp>
</rule>
<rule>
<imp>

<var>a63</var>
<var>a65</var>
</imp>
</rule>
<rule>
<imp>
<var>a68</var>
<var>a66</var>
</imp>
</rule>
</constraints>
<calculations Auto="true" Constraints="true" Features="true" Redundant="true" Tautology="true"/>
<comments/>
<featureOrder userDefined="false"/>
</featureModel>