MERGING MULTI-VIEW FEATURE MODELS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ELÇİN ATILGAN AYDIN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCES
IN
COMPUTER ENGINEERING

DECEMBER 2011

Approval of the thesis:

MERGING MULTI-VIEW FEATURE MODELS

submitted by **ELÇİN ATILGAN AYDIN** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Adnan Yazıcı _____
Head of Department, **Computer Engineering**

Assoc. Prof. Dr. Halit Oğuztüzün _____
Supervisor, **Computer Engineering Dept., METU**

Assoc. Prof. Dr. Ali Hikmet Doğru _____
Co-Supervisor, **Computer Engineering Dept., METU**

Examining Committee Members:

Prof. Dr. Nihan Kesim Çiçekli _____
Computer Engineering Dept., METU

Assoc. Prof. Dr. Halit Oğuztüzün _____
Computer Engineering Dept., METU

Prof. Dr. Ferda Nur Alpaslan _____
Computer Engineering Dept., METU

Dr. Cevat Şener _____
Computer Engineering Dept., METU

Dr. Ahmet Serkan Karataş _____
K&K Teknoloji Ltd. Şti.

Date: 27.12.2011

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: ELÇİN ATILGAN AYDIN

Signature:

ABSTRACT

MERGING MULTI-VIEW FEATURE MODELS

Atılgan Aydın, Elçin

M.Sc. Department of Computer Engineering

Supervisor: Assoc. Prof. Dr. Halit Oğuztüzün

Co-Supervisor: Assoc. Prof. Dr. Ali Hikmet Doğru

December 2011, 173 pages

Feature models are used for variability management in software product lines. Instead of developing a single feature model, merging small models can be an effective solution to obtain a unified view. Since each stakeholder views the product family from a different perspective, conflicts may occur during merging. In this research, merging of feature models arising from different viewpoints is considered. A normative procedure is proposed to merge feature models by applying local rules. This procedure can merge feature models with cross-tree relationships between sibling features. Application of the local rules is demonstrated with examples.

Keywords: Feature Models, Multiple Views

ÖZ

DEĞİŞİK BAKIŞ AÇILARINA SAHİP ÖZELLİK MODELLERİNİN BİRLEŞTİRİLMESİ

Atılgan Aydın, Elçin

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Doç. Dr. Halit Oğuztüzün

Ortak Tez Yöneticisi: Doç. Dr. Ali Hikmet Doğru

Aralık 2011, 173 sayfa

Özellik modelleri, yazılım ürün hatlarında değişkenlik yönetimi için kullanılmaktadır. Tek bir özellik modeli geliştirmek yerine, bütün bir görünümü elde etmek için küçük modelleri birleştirmek etkin bir çözüm olabilir. Her bir paydaş ürün ailesini farklı bakış açısıyla gördüğü için, birleştirme esnasında fikir ayrılıkları oluşabilir. Bu araştırmada, farklı bakış açılarından doğan özellik modellerinin birleştirilmesi ele alınmıştır. Özellik modellerinin bölgesel kurallar uygulanarak birleştirilebilmesi için kurallar içeren bir prosedür önerilmiştir. Bu prosedür, kardeş özellikleri arasında çapraz ağaç ilişkileri olan özellik modellerini birleştirebilmektedir. Bölgesel kuralların uygulanması örneklerle açıklanmıştır.

Anahtar Kelimeler: Özellik Modelleri, Değişik Bakış Açıları

To my dearest, most precious one and love of my life, İnan…

ACKNOWLEDGMENTS

Many people contributed to this thesis, and I would like to thank them all. I first want to express my deep appreciation to my supervisor, Assoc. Prof. Dr. Halit Oğuztüzün for his encouragement, guidance and support from the initial to the final level of this research.

Also, I would like to thank Assoc. Prof. Dr. Ali Hikmet Doğru for his idea, support and smiling face all the time. Thanks are due to Ahmet Serkan Karataş and Alper Kılıç for helpful discussions.

In addition, I'm very grateful to my parents Fatma and Mustafa Atılgan for being there when I need them.

And finally, I am heartily thankful to my darling, İnan. Thanks for your understanding for the many long weekends spent in my writing room and for having to split up the rock band to get this thesis done. We can fire up the Xbox again.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

Main motivation behind Software Product Line (SPL) Engineering is to achieve high levels of reusability by managing commonality and variability in product families [1]. SPL practice suggests that feature modeling is an effective approach to manage commonality and variability in an SPL [2]. Feature Models (FMs) support specifying, developing and managing reusable assets [3].

Industrial experience suggests that FMs may encounter scalability problems. FMs with thousands of features are common [4]. Scalability issue makes developing a single FM an insurmountable task. Furthermore, such large FMs cannot be easily understood by stakeholders. Adhering to the separation of concerns principle [5], rather than building one big FM for the whole system, smaller FMs can be constructed by each stakeholder separately and these models can be merged, when necessary, to obtain a unified view. This approach holds promise to improve scalability and usability of FMs.

A viewpoint [5] addresses the concerns of some stakeholder. It reflects the stakeholder's interest in the domain. A view that belongs to a specific viewpoint can be represented as an FM. Merging of separate FMs which are produced by different stakeholders, can involve disagreements that may or may not be resolved. For example, the situation where a feature is "mandatory" in one view, but "optional" in another view, indicates a disagreement between views.

There are other sources of disagreements between views, such as different names for essentially the same feature or the same names for substantially different features. Such issues are, however, beyond the scope of this thesis.

Consider the following example. Disagreements are likely to arise on which features must or must not be included in a product between marketing and engineering departments in a company. The engineering department can model a feature as mandatory due to design constraints whereas the marketing department can model the same feature as optional due

to the pricing considerations. When merging these incompatible views, a conflict between them is realized and a resolution is required.

One solution to this problem is asking departments to reach an agreement on a conflicting feature. However, reaching an agreement needs time and effort. Besides, departments can insist on their selections. Another solution is using a default merging procedure and obtaining a resolution that hopefully satisfies both parties of the disagreement.

In this thesis, an approach from [4] is enhanced and a normative procedure for merging FMs by local rules is proposed. Using the procedure, one can merge FMs with or without cross-tree relationships between sibling features. Rules are presented with rationales behind them while the merging procedure is being described step by step. Resolutions are also presented and approach is demonstrated using examples. A summary of results from this work has been presented in [6].

This thesis is organized as follows: Chapter 2 presents the background material about FMs, views in a nutshell; discusses the results obtained and compares the results with related literature. Chapter 3 describes a normative merging procedure with different examples and cases. Chapter 4 gives the rules for constructing resolution FM with examples. Chapter 5 defines an algorithm to merge complete FMs. Chapter 6 characterizes the rules logically. Finally, Chapter 7 presents some conclusions and future work.

CHAPTER 2

BACKGROUND

2.1 Feature Models

Often engineers and customers express the unique characteristics of products in terms of features [7]. A feature is a distinctive characteristic of a product that reflects some stakeholder's concern about the domain [2].

FMs are extensively used in variability management in the context of SPL Engineering [8]. An FM consists of a hierarchically arranged set of features and relationships among them. Feature Modeling allows stakeholders to describe commonalities and variabilities among domain concepts within a family of products in terms of features [9]. FMs have proved to be effective devices for identifying the characteristics of a product family in terms of commonality and variability since they were introduced by Kang *et al.* in Feature-oriented Domain Analysis (FODA) [10].

FMs include two types of relationships: decomposition relationships and cross-tree relationships [3]. Decomposition relationships are between a parent feature and its children. There are five types of decomposition relationships: *Mandatory*, *Optional*, *Alternative*, *Or* and *Not-available*. Note that the *not-available* relation appears in [4] to account for those features that are to be excluded in a particular view. Cross-tree relationships are used to specify constraints between pairs of arbitrary features. Two types of cross-tree relationships are considered: *Requires* and *Excludes*.

Relationships between a parent feature F_p and its child features are given below. Assume that the parent feature F_p is included in a product.

- Mandatory (Figure 1): Specifies that the child feature F_i must be also included in the product.

- Optional (Figure 2): Specifies that the child feature F_i may or may not be included, hence its presence in the product is optional.
- Alternative (Figure 3): Specifies that exactly one child feature from the group of features { F_i, F_j } must be included.
- Or (Figure 4): Specifies that at least one child feature from the group { F_i, F_j } must be included.
- Not-available (Figure 5): Specifies that the child feature { F_i } must not be included.

Figure 1 – Mandatory Relation

Figure 2 – Optional Relation

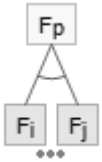

Figure 3 – Alternative Relation

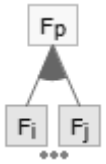

Figure 4 – Or Relation

Figure 5 – Not-available Relation

The *requires* relation shown in Figure 6 and the *excludes* relation shown in Figure 7 supplement the FM with dependency and mutual exclusion relationships, respectively [2].

Figure 6 – Requires Relation

Figure 7 – Excludes Relation

A view that reflects a specific viewpoint of a stakeholder is represented as an FM. Stakeholders may have their own FMs that reflect their points of view to the domain.

Merging process of separate FMs which are produced by different stakeholders often encounters with conflicting views. For example, the fact that a feature is “mandatory” in one view, but “optional” in another view indicates that there is a conflict between the views.

In [4], Mannion *et al.* propose some default conflict resolution rules to solve disagreements between views. They claim that to solve disagreements, the rationale behind variability should be understood. They use views to decrease complexity level, gain mapping of stakeholders to the variability and derive products. Their conflict resolution rules provide a mechanism for improving existing products, building new products easily and obtaining trade-offs that often occur between pricing considerations and design constraints.

2.2 Related Work

Merging of FMs reflecting different views of the product family has recently attracted the attention of many researchers.

Sagura *et al.* [12] provide a catalogue of merging rules using graph transformations. As they stated, their approach needs a clear formal semantics. Equivalent FMs may lead to nonequivalent resulting FMs. However, rules in this thesis have a formal semantic in terms of the all products operation and in addition generate consistent results.

Acher *et al.* [9] define two operators for merging FMs: insert and merge. Merge operator can be applied in two modes: union mode and intersection mode. Definitions of these modes and the differences between modes decrease the applicability of the operations in a simple manner. The algorithm presented by the authors cannot handle equal features effectively due to lack of parent compatibility precondition. Moreover, this algorithm does not cover cross-tree constraints. Furthermore, in some cases the resulting FM may represent products that are not valid with respect to any of the source FMs.

Broek *et al.* [13] propose an approach that can merge FMs that include “requires” and “excludes” constraints. However, in the first phase of the merging process, they replace the FM with an equivalent FM which they call FM “normal”. Their resulting FM has at least all the products of both constituent FMs, but it can have superfluous products compared with our approach, as in [9]. Additionally, their approach seems complicated to implement, while the procedure in this thesis can be implemented easily.

Niu *et al.* [14] proposed a method that can tolerate inconsistencies between FMs to be merged. They model the resulting FM as consisting of different perspectives of stakeholders.

They do not intend to implement a single coherent model as a result of merging, instead they delay inconsistency resolutions until the rationales which cause distinctive choices are better understood. Their approach is opposite to the given approach in this thesis. Here, solving disagreements as soon as they are detected is attempted.

Schobbens *et al.* [15] define an approach that involves three operations to merge FMs but do not provide a rule to implement merging.

Clarke *et al.* [16] propose a method which checks whether two different views, which are produced by different stakeholders, are compatible with each other. If they are so, they provide a method to merge them. In real life, since each stakeholder views the domain from their own perspective, conflicts on their FMs may arise. In this thesis, default rules for solving disagreements are provided.

In [4] by Mannion *et al.,* conflict resolution rules are introduced to integrate some views that were inconsistent before, however, how resolutions are obtained is not explained. Only the FMs to be merged and their resolution FMs are presented. Thus, rationales behind the rules are not clear. Consequently, it may not be straight forward to extend the given rules to cover all possible situations.

CHAPTER 3

MERGING BY CONFORMANCE

Every stakeholder has one or more viewpoints [5]. A view corresponding to a particular viewpoint is represented as an FM. In other words, an FM embodies a view resulting from the viewpoint of some stakeholder.

Since each stakeholder perceives the problem domain from their own point of view, conflicts may occur. Composition by conformance aims to satisfy each stakeholder in the resulting FM. In the scope of this thesis, *merging by conformance* will refer to the compositional approach.

Merging will work by focusing on a single parent feature and a number of first level sibling features at a time. Precondition of this procedure is root compatibility, i.e. the FMs to be merged must have same root (parent) feature.

The partial FMs to be merged in a single step are called as *Local View A* and *Local View B*, where A and B designate two stakeholders. FM resulting from the merging of views is called as the resolution FM, and it is designated as R.

The merging procedure proceeds top to bottom along both input FMs. The full procedure is covered in chapter 5.

Given a local view A with a parent feature *p*, the set of all the *p*'s child-sets is considered, designated $childSets_A$, obtained by deleting the parent feature *p* of A from each product represented by A (treating A as a complete FM). Formally, $childSets_A = \{ P - \{ p \}: P \in allProducts_A \}$.

A representation called *feature selection map for a set of children* is constructed on a set of child features and symbolized as *fs* X or *fs* Y, where $X \in childSets_A$ and $Y \in childSets_B$.

fs X and *fs* Y mark each child feature existing in A or B, respectively, with one of the symbols: '+', '-' and '/'.

Let A and B be two local views to be merged, and let $X \in childSets_A$ and let $Y \in childSets_B$.

Rules for constructing *fs* X (with respect to the local views A and B) are as follows:

- All features that exist in X are marked as '+'.
- All child features that exist in local view A, but do not exist in X are marked as '-'.
- All child features that exist in local view B, but do not exist in X are marked as '/'.

Note that these rules do not mark any feature with '!'; this may come up as a result of combination (defined below).

The rules for constructing *fs* Y (with respect to local views A and B) are defined similarly.

The elements '!', '+', '-' and '/' form a lattice, depicted in figure.

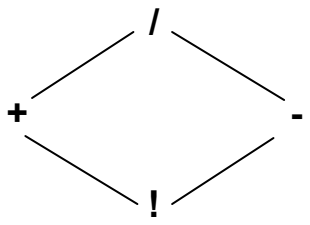

Figure 8 – The lattice that is used to define combination

The '!' is the minimum, '/' is the maximum element, and '+' and '-' are not comparable. The combination of two selections, x and y, designated $x \otimes y$, is defined as the greatest lower bound (glb) of the given symbols. For example, $/ \otimes - = -$, $+ \otimes - = !$, etc.

Note that these rules can be interpreted in terms of voting: A '+' means a "yes" vote, a '-' means a "no" vote, '/' means "abstain" (or "don't care"), and '!' indicates a conflict situation that cannot be resolved.

Combining two feature selection maps, *fs* X and *fs* Y, denoted *fs* X $\otimes$ *fs* Y, is defined as the feature-wise extension of the combination operation on symbols defined above.

Note the following properties of the $\otimes$ operation:

- It is commutative, i.e. *fs* X $\otimes$ *fs* Y = *fs* Y $\otimes$ *fs* X for any child-sets X and Y.
- It is idempotentive, i.e. *fs* X $\otimes$ *fs* X = *fs* X for any child-set X.
- It is associative, i.e. let C be another local view to be merged and let $Z \in childSets_C$ (*fs* X $\otimes$ *fs* Y) $\otimes$ *fs* Z = *fs* X $\otimes$ (*fs* Y $\otimes$ *fs* Z) for any child-sets X, Y and Z.

An operation called *conform* is constructed on *fs* X and *fs* Y representations and symbolized as $conform_A$ and $conform_B$. As a result of $conform_A$ and $conform_B$ operations, the $childSets_R$ is produced. *Conform* is essentially a filtering method that involves combination.

Once *fs* X ⊗ *fs* Y is obtained for each X and Y, $conform_A$ filters in the Y's and $conform_B$ filters in the X's that are in some sense "conforming". More precisely, to apply $conform_A$ on $childSets_B$, for each Y ∈ $childSets_B$ whether *fs* X ⊗ *fs* Y = *fs* Y for some X ∈ $childSets_A$ is checked. If this is the case then Y is considered conforming to the local view A. Similarly, to apply $conform_B$ on $childSets_A$, for each X ∈ $childSets_A$ whether *fs* X = *fs* Y ⊗ *fs* X for some Y ∈ $childSets_B$ is checked. If this is the case then X is considered conforming to the local view B.

The set of the child-sets of the resolution is constructed as follows:

$childSets_R = conform_A(childSets_B) \cup conform_B(childSets_A)$

Note the following relationship:

$childSets_A \cap childSets_B \subseteq childSets_R \subseteq childSets_A \cup childSets_B$

The results of *conform* operations make-up $childSets_R$. All of the '+' marked features are included in $childSets_R$ and '-' marked features are discarded from $childSets_R$. Finally, this set is used to build the FM R. Note that when $childSets_R$ is empty, no merging is possible by default rules.

3.1 Introductory Example

An example from the home security system domain is presented. It is adapted from [1].

Figure 9 and Figure 10 represent local views of marketing and engineering departments, respectively.

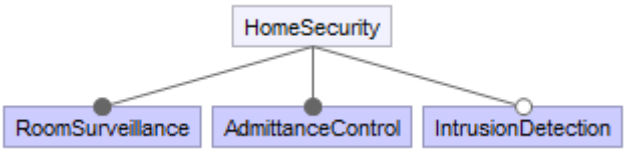

Figure 9 – Marketing Local View

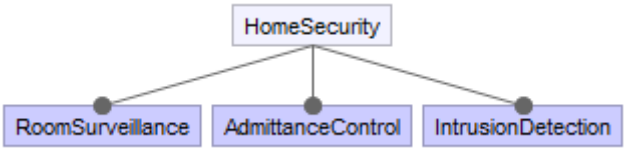

Figure 10 – Engineering Local View

As indicated in the Figure 9 and Figure 10 the marketing department models the Intrusion Detection as an optional feature because of pricing considerations, but the engineering department models the same feature as mandatory because of security constraints. When a merger of these views is needed, a conflict between them is realized and a resolution is required. Using *merging by conformance*, a resolution is obtained that hopefully satisfies both parties of the disagreement.

First, $childSets_{Mar}$ and $childSets_{Eng}$ are defined.

The abbreviations *rs*, *ac* and *id* for Room Surveillance, Admittance Control and Intrusion Detection are used, respectively.

$childSets_{Mar}$ = { { *rs*, *ac*, *id* }, { *rs*, *ac* } }.

$childSets_{Eng}$ = { { *rs*, *ac*, *id* } }.

Second, *fs* X and *fs* Y representations are constructed as Table 1 and Table 2.

Table 1 - Feature Selection Map *fs* X

row number		***rs***	***ac***	***id***
1	***fs { rs, ac, id }***	+	+	+
2	***fs { rs, ac }***	+	+	-

Table 2 - Feature Selection Map *fs* Y

row number		***rs***	***ac***	***id***
1	***fs { rs, ac, id }***	+	+	+

Next, combination for each *fs* X and *fs* Y is constructed. When there is a conflict on feature *f*, result of combination is marked as '!'. Table 3 shows the conflict situation that cannot be resolved by the default rules. In other words, Table 3 gives the result of combination where *fs* X and *fs* Y are not preserved.

Table 3 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		***rs***	***ac***	***id***
1	*fs* { *rs*, *ac* }	+	+	-
2	*fs* { *rs*, *ac*, *id* }	+	+	+
3	***fs* { *rs*, *ac* } ⊗ *fs* { *rs*, *ac*, *id* }**	+	+	!

As mentioned above, to apply $conform_{Eng}$ on $childSets_{Mar}$, for each X ∈ $childSets_{Mar}$ whether *fs* X = *fs* Y ⊗ *fs* X for some Y ∈ $childSets_{Eng}$ is checked. If this is the case then X is considered conforming to the local view engineering.

When $conform_{Eng}$ is applied on $childSets_{Mar}$, first whether *fs* { *rs*, *ac, id* } = *fs* { *rs*, *ac, id* } ⊗ *fs* { *rs*, *ac, id* } is checked and { *rs*, *ac, id* } is considered conforming to the local view engineering. Then, whether *fs* { *rs*, *ac* } = *fs* { *rs*, *ac, id* } ⊗ *fs* { *rs*, *ac* } is checked and presented *fs* { *rs*, *ac* } ≠ *fs* { *rs*, *ac, id* } ⊗ *fs* { *rs*, *ac* }. So, { *rs*, *ac* } is not considered conforming to the local view engineering and shown in Table 3.

$conform_{Eng}$ operation picks the *fs* X's that are preserved in the resolution set. This *fs* X is on the 3rd row of Table 4.

Table 4 - *fs* X ⊗ *fs* Y = *fs* X

row number		***rs***	***ac***	***id***
1	*fs* { *rs*, *ac,id* }	+	+	+
2	*fs* { *rs*, *ac*, *id* }	+	+	+
3	***fs* { *rs*, *ac*, *id* } ⊗ *fs* { *rs*, *ac*, *id* }**	+	+	+

Similarly, to apply $conform_{Mar}$ on $childSets_{Eng}$, for each Y ∈ $childSets_{Eng}$ whether *fs* X ⊗ *fs* Y = *fs* Y for some X ∈ $childSets_{Mar}$ is checked. If this is the case then Y is considered to the local view marketing.

When $conform_{Mar}$ is applied on $childSets_{Eng}$, first whether *fs* { *rs*, *ac, id* } ⊗ *fs* { *rs*, *ac, id* } = *fs* { *rs*, *ac, id* } is checked and { *rs*, *ac, id* } is considered conforming to the local view marketing.

$conform_{Mar}$ operation picks the *fs* Y's that are preserved in the resolution set. Table 5 shows this *fs* Y which is on the 3rd row.

Table 5 - *fs* X ⊗ *fs* Y = *fs* Y

row number		***rs***	***ac***	***id***
1	*fs* { *rs*, *ac,id* }	+	+	+
2	*fs* { *rs*, *ac*, *id* }	+	+	+
3	***fs* { *rs*, *ac*, *id* } ⊗ *fs* { *rs*, *ac*, *id* }**	+	+	+

After constructing combinations, the situation in both tables *fs* X's and *fs* Y's are preserved as noticed. Table 4 and Table 5 present the results of combinations where *fs* X's and *fs* Y's are preserved.

Performing the $conform_A$ and $conform_B$ operations, $childSets_{Man}$ is obtained as following:

$childSets_{Man}$ = { { rs, ac, id } }.

When the FM Management View is built by following rules given in chapter 4, the model given in Figure 11 is obtained.

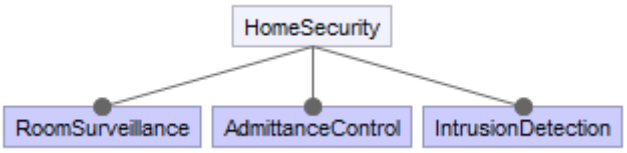

Figure 11 – Management Local View

Rule number 2 in [4] produces the same result as obtained by given approach.

3.2 Case 1: Revising Mannion *et al.* Rules

Before illustrating *merging by conformance* using Mannion *et al.* Rules, some of the rules given in Table 2 in [4] should be revised.

In [4], decomposition relationship *or* is denoted as *multiple*. In this thesis, *or* is used instead of *multiple* as in [3]. Besides, in [4] propositional logic of viewpoints and resolutions are misinterpreted in some cases. For example, to express the situation that "one set of children have their selection constraint values set to *mandatory*", they used a logical expression such that (p ↔ (i ∧ j)), but it should be ((p ↔ i) ∧ (p ↔ j)) [11]. In here, 'p' represents a parent feature F_p; 'i', 'j' represents first level sibling features of F_p; F_i and F_j from Table 2 given in [4]. *If and only if* does not distribute over *and*. Therefore, these two expressions are not logically equivalent to each other. It is proven below using the truth table.

Table 6 - Truth Table for Showing ((p ↔ i) ∧ (p ↔ j)) ≠ (p ↔ (i ∧ j))

p	**i**	**j**	**p ↔ i**	**p ↔ j**	**i ∧ j**	**(p ↔ i) ∧ (p ↔ j)**	**p ↔ (i ∧ j))**
0	0	1	1	0	0	0	1
0	1	0	0	1	0	0	1

These logical expressions are corrected and given in the table below.

To correct rule number 1(I) and 1(II) in [4], parent feature F_p is included into the expressions. In rule number 2, viewpoints and resolution are expressed using propositional logic instead of literary language. In rule number 5(III), resolution is corrected according to the given rule description. In addition, rule number 3(V), 5(IV), 5(V), 5(VIII), 5(IX) and 5(X) are reproduced according to the *merging by conformance* operation. Proofs of these rules are given in the following section: *Case 2: Reproducing Mannion et al. Rules*. Furthermore, all misspelling errors related to propositional logic and literary language are corrected.

The table consists of rules that are corrected in terms of misspelling and logical error.

Table 7 - Revised Rules

Rule	Viewpoint A	Viewpoint B	Resolution
1(I) When a feature in one viewpoint has a selection constraint value of *not-available* but the same feature in another viewpoint has a selection constraint value of *mandatory* then the complexity of the conflict is too great to propose an automatic solution and require human intervention.	$F_p \leftrightarrow \neg F_i$	$F_p \leftrightarrow F_i$	Too complex to solve automatically.
1(II) When a feature in one viewpoint has a selection constraint value of *not-available* but the same feature in another viewpoint has a selection constraint value of *optional* then the feature's selection constraint value becomes *not-available*.	$F_p \leftrightarrow \neg F_i$	$F_i \rightarrow F_p$	$F_p \leftrightarrow \neg F_i$
2 When a feature in one viewpoint has a selection constraint value of *mandatory* but the same feature in another viewpoint has a selection constraint value of *optional* then the feature's selection constraint value becomes *mandatory*.	$F_p \leftrightarrow F_i$	$F_i \rightarrow F_p$	$F_p \leftrightarrow F_i$
3(I) When the single child of a parent feature in one viewpoint has its selection constraint value set to *mandatory* and the set of children of the same parent feature in another viewpoint have their selection constraint values set to *alternative* then the selection constraint value of the child that is the same become *mandatory* but the selection constraint values of the children who are different become *not-available*.	$F_p \leftrightarrow F_i$	$F_p \leftrightarrow (F_i \oplus F_j \oplus F_k)$	$(F_p \leftrightarrow F_i) \wedge (F_p \leftrightarrow \neg F_j) \wedge (F_p \leftrightarrow \neg F_k)$
3(II) When the single child of a parent feature in one viewpoint has its selection constraint value set to *mandatory* and the set of children of the same parent feature in another viewpoint have their selection constraint values set to *or* then the selection constraint value of the child that is the same become *mandatory* butthe selection constraint values of the children who are different become *optional*.	$F_p \leftrightarrow F_i$	$F_p \leftrightarrow (F_i \vee F_j \vee F_k)$	$(F_p \leftrightarrow F_i) \wedge (F_j \rightarrow F_p) \wedge (F_k \rightarrow F_p)$

Table 7 (continued)

Rule	Viewpoint A	Viewpoint B	Resolution
3(III) When the single child of a parent feature in one viewpoint has its selection constraint value set to *mandatory* and the set of children of the same parent feature in another viewpoint have their selection constraint values set to *optional* then the selection constraint value of the child that is the same become *mandatory* but the selection constraint values of the children who are different become *optional*.	$F_p \leftrightarrow F_i$	$(F_i \rightarrow F_p) \wedge (F_j \rightarrow F_p) \wedge (F_k \rightarrow F_p)$	$(F_p \leftrightarrow F_i) \wedge (F_j \rightarrow F_p) \wedge (F_k \rightarrow F_p)$
3(IV) When the single child of a parent feature in one viewpoint has its selection constraint value set to *optional* and the set of children of the same parent feature in another viewpoint have their selection constraint values set to *alternative* then the selection constraint value of the child that is the same become *alternative* and the selection constraint values of the children who are different become *alternative*.	$F_i \rightarrow F_p$	$F_p \leftrightarrow (F_i \oplus F_j \oplus F_k)$	$F_p \leftrightarrow (F_i \oplus F_j \oplus F_k)$
3(V) When the single child of a parent feature in one viewpoint has its selection constraint value set to *optional* and the set of children of the same parent feature in another viewpoint have their selection constraint values set to *or* then the selection constraint value of the child that is the same become *or* and the selection constraint values of the children who are different become *or*.	$F_i \rightarrow F_p$	$F_p \leftrightarrow (F_i \vee F_j \vee F_k)$	$F_p \leftrightarrow (F_i \vee F_j \vee F_k)$
4(I) When the children of a parent feature are the same in each viewpoint but one set of children have their selection constraint values set to *mandatory* and the other set of children have their selection constraint values set to *alternative* then the complexity of the conflict is too great to propose an automatic solution and require human intervention.	$(F_p \leftrightarrow F_i) \wedge (F_p \leftrightarrow F_j)$	$F_p \leftrightarrow (F_i \oplus F_j)$	Too complex to solve automatically.

Table 7 (continued)

Rule	Viewpoint A	Viewpoint B	Resolution
4(II) When the children of a parent feature are the same in each viewpoint but one set of children have their selection constraint values set to *mandatory* and the other set of children have their selection constraint values set to *or* then the children's selection constraint values become *mandatory*.	$(F_p \leftrightarrow F_i) \wedge (F_p \leftrightarrow F_j)$	$F_p \leftrightarrow (F_i \vee F_j)$	$(F_p \leftrightarrow F_i) \wedge (F_p \leftrightarrow F_j)$
4(III) When the children of a parent feature are the same in each viewpoint but one set of children have their selection constraint values set to *mandatory* and the other set of children have their selection constraint values set to *optional* then the children's selection constraint values become *mandatory*.	$(F_p \leftrightarrow F_i) \wedge (F_p \leftrightarrow F_j)$	$(F_i \rightarrow F_p) \wedge (F_j \rightarrow F_p)$	$(F_p \leftrightarrow F_i) \wedge (F_p \leftrightarrow F_j)$
4(IV) When the children of a parent feature are the same in each viewpoint but one set of children have their selection constraint values set to *alternative* and the other set of children have their selection constraint values set to *or* then the children's selection constraint values become *alternative*.	$F_p \leftrightarrow (F_i \oplus F_j)$	$F_p \leftrightarrow (F_i \vee F_j)$	$F_p \leftrightarrow (F_i \oplus F_j)$
4(V) When the children of a parent feature are the same in each viewpoint but one set of children have their selection constraint values set to *alternative* and the other set of children have their selection constraint values set to *optional* then the children's selection constraint values become *alternative*.	$F_p \leftrightarrow (F_i \oplus F_j)$	$(F_i \rightarrow F_p) \wedge (F_j \rightarrow F_p)$	$F_p \leftrightarrow (F_i \oplus F_j)$
4(VI) When the children of a parent feature are the same in each viewpoint but one set of children have their selection constraint values set to *or* and the other set of children have their selection constraint values set to *optional* then the children's selection constraint values become *or*.	$F_p \leftrightarrow (F_i \vee F_j)$	$(F_i \rightarrow F_p) \wedge (F_j \rightarrow F_p)$	$F_p \leftrightarrow (F_i \vee F_j)$

Table 7 (continued)

Rule	Viewpoint A	Viewpoint B	Resolution
5(I) When the children of a parent feature in one viewpoint have their selection constraint values set to *mandatory* and the different but overlapping set of children of the same parent in another viewpoint have their selection constraint values set to *alternative* then the complexity of the conflict is too great to propose an automatic solution and require human intervention.	$(F_p \leftrightarrow F_i) \wedge (F_p \leftrightarrow F_j)$	$F_p \leftrightarrow (F_i \oplus F_j \oplus F_k)$	Too complex to solve automatically.
5(II) When the children of a parent feature in one viewpoint have their selection constraint values set to *mandatory* and the different but overlapping set of children of the same parent in another viewpoint have their selection constraint values set to *or* then the selection constraint values of the children who are the same become *mandatory* but the selection constraint values of the children who are different become *optional*.	$(F_p \leftrightarrow F_i) \wedge (F_p \leftrightarrow F_j)$	$F_p \leftrightarrow (F_i \vee F_j \vee F_k)$	$(F_p \leftrightarrow F_i) \wedge (F_p \leftrightarrow F_j) \wedge (F_k \rightarrow F_p)$
5(III) When the children of a parent feature in one viewpoint have their selection constraint values set to *mandatory* and the different but overlapping set of children of the same parent in another viewpoint have their selection constraint values set to *optional* then the selection constraint values of the children who are the same become *mandatory* but the selection constraint values of the children who are different become *optional*.	$(F_p \leftrightarrow F_i) \wedge (F_p \leftrightarrow F_j)$	$(F_i \rightarrow F_p) \wedge (F_j \rightarrow F_p) \wedge (F_k \rightarrow F_p)$	$(F_p \leftrightarrow F_i) \wedge (F_p \leftrightarrow F_j) \wedge (F_k \rightarrow F_p)$

Table 7 (continued)

Rule	Viewpoint A	Viewpoint B	Resolution
5(IV) When the children of a parent feature in one viewpoint have their selection constraint values set to *alternative* and the different but overlapping set of children of the same parent in another viewpoint have their selection constraint values set to *or*then the selection constraint values of the children who are the same become *alternative* but the selection constraint values of the children who are different become *optional*.	$F_p \leftrightarrow (F_i \oplus F_j)$	$F_p \leftrightarrow (F_i \vee F_j \vee F_k)$	$(F_p \leftrightarrow (F_i \oplus F_j)) \wedge (F_k \rightarrow F_p)$
5(V) When the children of a parent feature in one viewpoint have their selection constraint values set to *alternative* and the different but overlapping set of children of the same parent in another viewpoint have their selection constraint values set to *optional* then the selection constraint values of the children who are the same become *alternative* but the selection constraint values of the children who are different become *optional*.	$F_p \leftrightarrow (F_i \oplus F_j)$	$(F_i \rightarrow F_p) \wedge (F_j \rightarrow F_p) \wedge (F_k \rightarrow F_p)$	$(F_p \leftrightarrow (F_i \oplus F_j)) \wedge (F_k \rightarrow F_p)$
5(VI) When the children of a parent feature in one viewpoint have their selection constraint values set to *multiple* and the different but overlapping set of children of the same parent in another viewpoint have their selection constraint values set to *optional* then the selection constraint values of the children who are the same become *or* but the selection constraint values of the children who are different become *optional*.	$F_p \leftrightarrow (F_i \vee F_j)$	$(F_i \rightarrow F_p) \wedge (F_j \rightarrow F_p) \wedge (F_k \rightarrow F_p)$	$(F_p \leftrightarrow (F_i \vee F_j)) \wedge (F_k \rightarrow F_p)$

Table 7 (continued)

Rule	Viewpoint A	Viewpoint B	Resolution
5(VII) When the children of a parent feature in one viewpoint have their selection constraint values set to *alternative* and the different but overlapping set of children of the same parent in another viewpoint have their selection constraint values set to *alternative* and there is only one overlapping feature then the selection constraint value of the overlapping feature becomes *mandatory* and the rest become *not-available*.	$F_p \leftrightarrow (F_i \oplus F_j \oplus F_k)$	$F_p \leftrightarrow (F_i \oplus F_m \oplus F_n)$	$(F_p \leftrightarrow F_i) \wedge (F_p \leftrightarrow \neg F_j) \wedge (F_p \leftrightarrow \neg F_k) \wedge (F_p \leftrightarrow \neg F_m) \wedge (F_p \leftrightarrow \neg F_n)$
5(VIII) When the children of a parent feature in one viewpoint have their selection constraint values set to *alternative* and the different but overlapping set of children of the same parent in another viewpoint have their selection constraint values set to *alternative* and there is more than one overlapping feature then the selection constraint value of the overlapping features become *alternative* and the rest become *not-available*.	$F_p \leftrightarrow (F_i \oplus F_j \oplus F_k)$	$F_p \leftrightarrow (F_i \oplus F_j \oplus F_m \oplus F_n)$	$(F_p \leftrightarrow (F_i \oplus F_j)) \wedge (F_p \leftrightarrow \neg F_k) \wedge (F_p \leftrightarrow \neg F_m) \wedge (F_p \leftrightarrow \neg F_n)$
5(IX) When the children of a parent feature in one viewpoint have their selection constraint values set to *or* and the different but overlapping set of children of the same parent in another viewpoint have their selection constraint values set to *or* then the selection constraint value of the overlapping features become *or*, the rest become *optional* and between non-overlapping features from different viewpoints become *excludes.*	$F_p \leftrightarrow (F_i \vee F_j \vee F_k)$	$F_p \leftrightarrow (F_i \vee F_j \vee F_m \vee F_n)$	$(F_p \leftrightarrow (F_i \vee F_j)) \wedge (F_k \rightarrow F_p) \wedge (F_m \rightarrow F_p) \wedge (F_n \rightarrow F_p) \wedge (\neg (F_k \wedge F_m)) \wedge (\neg (F_k \wedge F_n))$

Table 7 (continued)

Rule	Viewpoint A	Viewpoint B	Resolution
5(X) When the children of a parent feature in one viewpoint have their selection constraint values set to *mandatory* and the different but overlapping set of children of the same parent in another viewpoint have their selection constraint values set to *mandatory* then the complexity of the conflict is too great to propose an automatic solution and require human intervention.	$(F_p \leftrightarrow F_i) \wedge (F_p \leftrightarrow F_j) \wedge (F_p \leftrightarrow F_k)$	$(F_p \leftrightarrow F_i) \wedge (F_p \leftrightarrow F_j) \wedge (F_p \leftrightarrow F_m) \wedge (F_p \leftrightarrow F_n)$	Too complex to solve automatically.

3.3 Case 2: Reproducing Mannion *et al.* Rules

To illustrate given merging procedure, rules are going to be reproduced in the order given in Table 7 - . Rules for "Difficult-to-Resolve Views" are going to be handled in the following section. Here, these rules are skipped.

Regarding Rule 1(II):

Figure 12 and Figure 13 represent the local views A and B, respectively. These are the views from the rule number 1(II).

Figure 12 – Local View A from rule 1(II)

Figure 13 – Local View B from rule 1(II)

First, $childSets_A$ and $childSets_B$ are determined.

The indices i, j, k etc. are used in the tables instead of F_i, F_j, F_k etc. for better readability.

$childSets_A = \{ \{ \} \}$.

$childSets_B = \{ \{ \}, \{ i \} \}$.

Second, *fs* X and *fs* Y representations are constructed as Table 8 and Table 9.

Table 8 - Feature Selection Map *fs* X

row number		**i**
1	***fs* { }**	-

Table 9 - Feature Selection Map *fs* Y

row number		**i**
1	***fs* { }**	-
2	***fs* { i }**	+

Next, combination for each *fs* X and *fs* Y is constructed. Table 10 shows the conflict situations that cannot be resolved.

Table 10 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**
1	*fs* { }	-
2	*fs* { i }	+
3	***fs* { } ⊗ *fs* { i }**	!

To reduce clutter, hereafter, the tables that present the operation details for *fs* X ⊗ *fs* Y do not show the cases where X = Y since *fs* X ⊗ *fs* X = *fs* X. Combining two feature selection maps is a commutative operation, so when tables present *fs* X ⊗ *fs* Y operation, they do not present *fs* Y ⊗ *fs* X operation for the same child-sets X and Y.

As mentioned above, the $conform_B$ operation picks the *fs* X's that are preserved and the $conform_A$ operation picks the *fs* Y's that are preserved in the resolution set. From the results of *conform* operations, $childSets_R$ is constructed as follows: $childSets_R$ = { { } }.

When the FM R is built, the model given in Figure 14 is obtained.

Figure 14 – Resolution of Figure 12 and Figure 13

Regarding Rule 2:

Figure 15 and Figure 16 represent the local views A and B, respectively, from the rule number 2.

Figure 15 – Local View A from rule 2

Figure 16 – Local View B from rule 2

First, $childSets_A$ and $childSets_B$ are determined.

$childSets_A$ = { { i } }.

$childSets_B$ = { { }, { i } }.

Second, *fs* X and *fs* Y representations are constructed as Table 11 and Table 12.

Table 11 - Feature Selection Map *fs* X

row number		**i**
1	***fs* { i }**	+

Table 12 - Feature Selection Map *fs* Y

row number		**i**
1	***fs* { }**	-
2	***fs* { i }**	+

Next, combination for each *fs* X and *fs* Y is constructed. Table 13 shows the conflict situations that cannot be resolved.

Table 13 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**
1	*fs* { i }	+
2	*fs* { }	-
3	***fs* { i } ⊗ *fs* { }**	!

From the results of *conform* operations, $childSets_R$ is constructed as follows: $childSets_R$ = { { i } }.

When the FM R is built, the model given in Figure 17 is obtained.

Figure 17 – Resolution of Figure 15 and Figure 16

Regarding Rule 3(I):

Figure 18 and Figure 19 represent the local views A and B, respectively, from the rule number 3(I).

Figure 18 – Local View A from rule 3(I)

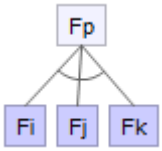

Figure 19 – Local View B from rule 3(I)

First, $childSets_A$ and $childSets_B$ are determined.

$childSets_A$ = { { i } }.

$childSets_B$ = { { i }, { j }, { k } }.

Second, *fs* X and *fs* Y representations are constructed as Table 14 and Table 15.

Table 14 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**
1	***fs* { i }**	+	/	/

Table 15 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**
1	***fs* { i }**	+	-	-
2	***fs* { j }**	-	+	-
3	***fs* { k }**	-	-	+

Next, combination for each *fs* X and *fs* Y is constructed. Table 16 shows the conflict situations that cannot be resolved.

Table 16 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**	**k**
1	*fs* { i }	+	/	/
2	*fs* { j }	-	+	-
3	***fs* { i } ⊗ *fs* { j }**	!	+	-
4	*fs* { i }	+	/	/
5	*fs* { k }	-	-	+
6	***fs* { i } ⊗ *fs* { k }**	!	-	+

From the results of *conform* operations, $childSets_R$ is constructed as follows: $childSets_R$ = { { i } }.

When the FM R is built, the model given in Figure 20 is obtained.

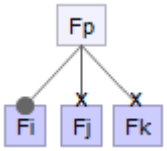

Figure 20 – Resolution of Figure 18 and Figure 19

Regarding Rule 3(II):

Figure 21 and Figure 22 represent the local views A and B, respectively, from the rule number 3(II).

Figure 21 – Local View A from rule 3(II)

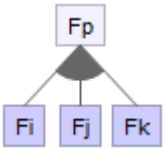

Figure 22 – Local View B from rule 3(II)

First, $childSets_A$ and $childSets_B$ are determined.

$childSets_A$ = { { i } }.

$childSets_B$ = { { i }, { j }, { k }, { i, j }, { i, k }, { j, k }, { i, j, k } }.

Second, *fs* X and *fs* Y representations are constructed as Table 17 and Table 18.

Table 17 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**
1	***fs* { i }**	+	/	/

Table 18 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**
1	***fs* { i }**	+	-	-
2	***fs* { j }**	-	+	-
3	***fs* { k }**	-	-	+
4	***fs* { i, j }**	+	+	-
5	***fs* { i, k }**	+	-	+
6	***fs* { j, k }**	-	+	+
7	***fs* { i, j, k }**	+	+	+

Next, combination for each *fs* X and *fs* Y is constructed. Table 19 shows the conflict situations that cannot be resolved.

Table 19 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**	**k**
1	*fs* { i }	+	/	/
2	*fs* { j }	-	+	-
3	***fs* { i } ⊗ *fs* { j }**	!	+	-
4	*fs* { i }	+	/	/
5	*fs* { k }	-	-	+
6	***fs* { i } ⊗ *fs* { k }**	!	-	+
7	*fs* { i }	+	/	/
8	*fs* { j, k }	-	+	+
9	***fs* { i } ⊗ *fs* { j, k }**	!	+	+

Table 20 shows the *fs* Y's that are preserved.

Table 20 - *fs* X ⊗ *fs* Y = *fs* Y

row number		**i**	**j**	**k**
1	*fs* { i }	+	/	/
2	*fs* { i, j }	+	+	-
3	***fs* { i } ⊗ *fs* { i, j } = *fs* { i, j }**	+	+	-
4	*fs* { i }	+	/	/
5	*fs* { i, k }	+	-	+
6	***fs* { i } ⊗ *fs* { i, k } = *fs* { i, k }**	+	-	+
7	*fs* { i }	+	/	/
8	*fs* { i, j, k }	+	+	+
9	***fs* { i } ⊗ *fs* { i, j, k } = *fs* { i, j, k }**	+	+	+

From the results of *conform* operations, $childSets_R$ is constructed as follows: $childSets_R$ = { { i }, { i, j }, { i, k }, { i, j, k } }.

When the FM R is built, the model given in Figure 23 is obtained.

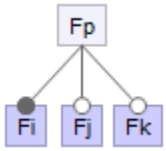

Figure 23 – Resolution of Figure 21 and Figure 22

Regarding Rule 3(III):

Figure 24 and Figure 25 represent the local views A and B, respectively, from the rule number 3(III).

Figure 24 – Local View A from rule 3(III)

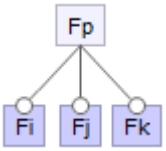

Figure 25 – Local View B from rule 3(III)

First, $childSets_A$ and $childSets_B$ are determined.

$childSets_A$ = { { i } }.

$childSets_B$ = { { },{ i }, { j }, { k }, { i, j }, { i, k }, { j, k }, { i, j, k } }.

Second, *fs* X and *fs* Y representations are constructed as Table 21 and Table 22.

Table 21 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**
1	***fs* { i }**	+	/	/

Table 22 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**
1	***fs* { }**	-	-	-
2	***fs* { i }**	+	-	-
3	***fs* { j }**	-	+	-
4	***fs* { k }**	-	-	+
5	***fs* { i, j }**	+	+	-
6	***fs* { i, k }**	+	-	+
7	***fs* { j, k }**	-	+	+
8	***fs* { i, j, k }**	+	+	+

Next, combination for each *fs* X and *fs* Y is constructed. Table 23 shows the conflict situations that cannot be resolved.

Table 23 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**	**k**
1	*fs* { i }	+	/	/
2	*fs* { }	-	-	-
3	***fs* { i } ⊗ *fs* { }**	!	-	-
4	*fs* { i }	+	/	/
5	*fs* { j }	-	+	-
6	***fs* { i } ⊗ *fs* { j }**	!	+	-
7	*fs* { i }	+	/	/
8	*fs* { k }	-	-	+
9	***fs* { i } ⊗ *fs* { k }**	!	-	+
10	*fs* { i }	+	/	/
11	*fs* { j, k }	-	+	+
12	***fs* { i } ⊗ *fs* { j, k }**	!	+	+

Table 24 shows the *fs* Y's that are preserved.

Table 24 - *fs* X ⊗ *fs* Y = *fs* Y

row number		**i**	**j**	**k**
1	*fs* { i }	+	/	/
2	*fs* { i, j }	+	+	-
3	***fs* { i } ⊗ *fs* { i, j } = *fs* { i, j }**	+	+	-
4	*fs* { i }	+	/	/
5	*fs* { i, k }	+	-	+
6	***fs* { i } ⊗ *fs* { i, k } = *fs* { i, k }**	+	-	+
7	*fs* { i }	+	/	/
8	*fs* { i, j, k }	+	+	+
9	***fs* { i } ⊗ *fs* { i, j, k } = *fs* { i, j, k }**	+	+	+

From the results of *conform* operations, $childSets_R$ is constructed as follows: $childSets_R$ = { { i }, { i, j }, { i, k }, { i, j, k } }.

When the FM R is built, the model given in Figure 26 is obtained.

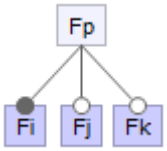

Figure 26 – Resolution of Figure 24 and Figure 25

Regarding Rule 3(IV):

Figure 27 and Figure 28 represent the local views A and B, respectively, from the rule number 3(IV).

Figure 27 – Local View A from rule 3(IV)

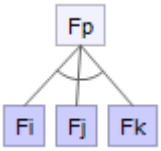

Figure 28 – Local View B from rule 3(IV)

First, $childSets_A$ and $childSets_B$ are determined.

$childSets_A$ = { { }, { i } }.

$childSets_B$ = { { i }, { j }, { k }}.

Second, *fs* X and *fs* Y representations are constructed as Table 25 and Table 26.

Table 25 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**
1	***fs* { }**	-	/	/
2	***fs* { i }**	+	/	/

Table 26 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**
1	***fs* { i }**	+	-	-
2	***fs* { j }**	-	+	-
3	***fs* { k }**	-	-	+

Next, combination for each *fs* X and *fs* Y is constructed. Table 27 shows the conflict situations that cannot be resolved.

Table 27 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**	**k**
1	*fs* { }	-	/	/
2	*fs* { i }	+	-	-
3	***fs* { } ⊗ *fs* { i }**	!	-	-
4	*fs* { i }	+	/	/
5	*fs* { j }	-	+	-
6	***fs* { i } ⊗ *fs* { j }**	!	+	-
7	*fs* { i }	+	/	/
8	*fs* { k }	-	-	+
9	***fs* { i } ⊗ *fs* { k }**	!	-	+

Table 28 shows the *fs* Y's that are preserved.

Table 28 - *fs* X ⊗ *fs* Y = *fs* Y

row number		**i**	**j**	**k**
1	*fs* { }	-	/	/
2	*fs* { j }	-	+	-
3	***fs* { } ⊗ *fs* { j } = *fs* { j }**	-	+	-
4	*fs* { }	-	/	/
5	*fs* { k }	-	-	+
6	***fs* { } ⊗ *fs* { k } = *fs* { k }**	-	-	+

From the results of *conform* operations, $childSets_R$ is constructed as follows: $childSets_R$ = { { i }, { j }, { k } }.

When the FM R is built, the model given in Figure 29 is obtained.

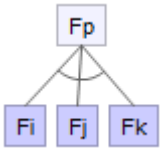

Figure 29 – Resolution of Figure 27 and Figure 28

Regarding Rule 3(V):

Figure 30 and Figure 31 represent the local views A and B, respectively, from the rule number 3(V).

Figure 30 – Local View A from rule 3(V)

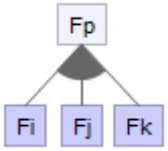

Figure 31 – Local View B from rule 3(V)

First, $childSets_A$ and $childSets_B$ are determined.

$childSets_A$ = { { }, { i } }.

$childSets_B$ = { { i }, { j }, { k }, { i, j }, { i, k }, { j, k }, { i, j, k } }.

Second, *fs* X and *fs* Y representations are constructed as Table 29 and Table 30.

Table 29 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**
1	***fs* { }**	-	/	/
2	***fs* { i }**	+	/	/

Table 30 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**
1	***fs* { i }**	+	-	-
2	***fs* { j }**	-	+	-
3	***fs* { k }**	-	-	+
4	***fs* { i, j }**	+	+	-
5	***fs* { i, k }**	+	-	+
6	***fs* { j, k }**	-	+	+
7	***fs* { i, j, k }**	+	+	+

Next, combination for each *fs* X and *fs* Y is constructed. Table 31 shows the conflict situations that cannot be resolved.

Table 31 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**	**k**
1	*fs* { }	-	/	/
2	*fs* { i }	+	-	-
3	***fs* { } ⊗ *fs* { i }**	!	-	-
4	*fs* { }	-	/	/
5	*fs* { i, j }	+	+	-
6	***fs* { } ⊗ *fs* { i, j }**	!	+	-
7	*fs* { }	-	/	/
8	*fs* { i, k }	+	-	+
9	***fs* { } ⊗ *fs* { i, k }**	!	-	+
10	*fs* { }	-	/	/
11	*fs* { i, j, k }	+	+	+
12	***fs* { } ⊗ *fs* { i, j, k }**	!	+	+
13	*fs* { i }	+	/	/
14	*fs* { j }	-	+	-
15	***fs* { i } ⊗ *fs* { j }**	!	+	-
16	*fs* { i }	+	/	/
17	*fs* { k }	-	-	+
18	***fs* { i } ⊗ *fs* { k }**	!	-	+
19	*fs* { i }	+	/	/
20	*fs* { j, k }	-	+	+
21	***fs* { i } ⊗ *fs* { j, k }**	!	+	+

Table 32 shows the *fs* Y's that are preserved.

Table 32 - *fs* X ⊗ *fs* Y = *fs* Y

row number		**i**	**j**	**k**
1	*fs* { }	-	/	/
2	*fs* { j }	-	+	-
3	***fs* { } ⊗ *fs* { j } = *fs* { j }**	-	+	-
4	*fs* { }	-	/	/
5	*fs* { k }	-	-	+
6	***fs* { } ⊗ *fs* { k } = *fs* { k }**	-	-	+
7	*fs* { }	-	/	/
8	*fs* { j, k }	-	+	+
9	***fs* { } ⊗ *fs* { j, k } = *fs* { j, k }**	-	+	+
10	*fs* { i }	+	/	/
11	*fs* { i, j }	+	+	-
12	***fs* { i } ⊗ *fs* { i, j } = *fs* { i, j }**	-	+	-
13	*fs* { i }	+	/	/
14	*fs* { i, k }	+	-	+
15	***fs* { i } ⊗ *fs* { i, k } = *fs* { i, k }**	+	-	+
16	*fs* { i }	+	/	/
17	*fs* { i, j, k }	+	+	+
18	***fs* { i } ⊗ *fs* { i, j, k } = *fs* { i, j, k }**	+	+	+

From the results of *conform* operations, $childSets_R$ is constructed as follows: $childSets_R$ = { { i }, { j }, { k }, { j, k }, { i, j }, { i, k }, { i, j, k } }.

When the FM R is built, the model given in Figure 32 is obtained.

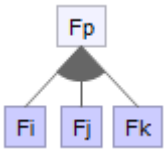

Figure 32 – Resolution of Figure 30 and Figure 31

Regarding Rule 4(II):

Figure 33 and Figure 34 represent the local views A and B, respectively, from the rule number 4(II).

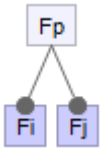

Figure 33 – Local View A from rule 4(II)

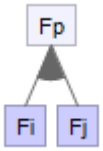

Figure 34 – Local View B from rule 4(II)

First, $childSets_A$ and $childSets_B$ are determined.

$childSets_A$ = { { i, j } }.

$childSets_B$ = { { i }, { j }, { i, j } }.

Second, *fs* X and *fs* Y representations are constructed as Table 33 and Table 34.

Table 33 - Feature Selection Map *fs* X

row number		**i**	**j**
1	***fs* { i, j }**	+	+

Table 34 - Feature Selection Map *fs* Y

row number		**i**	**j**
1	***fs* { i }**	+	-
2	***fs* { j }**	-	+
3	***fs* { i, j }**	+	+

Next, combination for each *fs* X and *fs* Y is constructed. Table 35 shows the conflict situations that cannot be resolved.

Table 35 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		i	**j**
1	*fs* { i, j }	+	+
2	*fs* { i }	+	-
3	***fs* { i, j } ⊗ *fs* { i }**	+	!
4	*fs* { i, j }	+	+
5	*fs* { j }	-	+
6	***fs* { i, j } ⊗ *fs* { j }**	!	+

From the results of *conform* operations, $childSets_R$ is constructed as follows: $childSets_R$ = { { i, j } }.

When the FM R is built, the model given in Figure 35 is obtained.

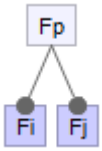

Figure 35 – Resolution of Figure 33 and Figure 34

Regarding Rule 4(III):

Figure 36 and Figure 37 represent the local views A and B, respectively, from the rule number 4(III).

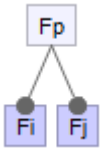

Figure 36 – Local View A from rule 4(III)

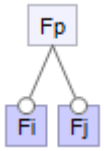

Figure 37 – Local View B from rule 4(III)

First, $childSets_A$ and $childSets_B$ are determined.

$childSets_A$ = { { i, j } }.

$childSets_B$ = { { }, { i }, { j }, { i, j } }.

Second, *fs* X and *fs* Y representations are constructed as Table 36 and Table 37.

Table 36 - Feature Selection Map *fs* X

row number		**i**	**j**
1	***fs* { i, j }**	+	+

Table 37 - Feature Selection Map *fs* Y

row number		**i**	**j**
1	***fs* { }**	-	-
2	***fs* { i }**	+	-
3	***fs* { j }**	-	+
4	***fs* { i, j }**	+	+

Next, combination for each *fs* X and *fs* Y is constructed. Table 38 shows the conflict situations that cannot be resolved.

Table 38 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**
1	*fs* { i, j }	+	+
2	*fs* { }	-	-
3	***fs* { i, j } ⊗ *fs* { }**	!	!
4	*fs* { i, j }	+	+
5	*fs* { i }	+	-
6	***fs* { i, j } ⊗ *fs* { i }**	+	!
7	*fs* { i, j }	+	+
8	*fs* { j }	-	+
9	***fs* { i, j } ⊗ *fs* { j }**	!	+

From the results of *conform* operations, $childSets_R$ is constructed as follows: $childSets_R$ = { { i, j } }.

When the FM R is built, the model given in Figure 38 is obtained.

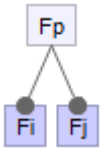

Figure 38 – Resolution of Figure 36 and Figure 37

Regarding Rule 4(IV):

Figure 39 and Figure 40 represent the local views A and B, respectively, from the rule number 4(IV).

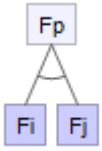

Figure 39 – Local View A from rule 4(IV)

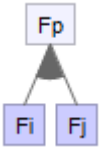

Figure 40 – Local View B from rule 4(IV)

First, $childSets_A$ and $childSets_B$ are determined.

$childSets_A$ = { { i }, { j } }.

$childSets_B$ = { { i }, { j }, { i, j } }.

Second, *fs* X and *fs* Y representations are constructed as Table 39 and Table 40.

Table 39 - Feature Selection Map *fs* X

row number		**i**	**j**
1	***fs* { i }**	+	-
2	***fs* { j }**	-	+

Table 40 - Feature Selection Map *fs* Y

row number		**i**	**j**
1	***fs* { i }**	+	-
2	***fs* { j }**	-	+
3	***fs* { i, j }**	+	+

Next, combination for each *fs* X and *fs* Y is constructed. Table 41 shows the conflict situations that cannot be resolved.

Table 41 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**
1	*fs* { i }	+	-
2	*fs* { j }	-	+
3	***fs* { i } ⊗ *fs* { j }**	!	!
4	*fs* { i }	+	-
5	*fs* { i, j }	+	+
6	***fs* { i } ⊗ *fs* { i, j }**	+	!
7	*fs* { j }	-	+
8	*fs* { i, j }	+	+
9	***fs* { j } ⊗ *fs* { i, j }**	!	+

From the results of *conform* operations, $childSets_R$ is constructed as follows: $childSets_R$ = { { i }, { j } }.

When the FM R is built, the model given in Figure 41 is obtained.

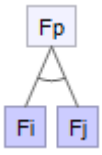

Figure 41 – Resolution of Figure 39 and Figure 40

Regarding Rule 4(V):

Figure 42 and Figure 43 represent the local views A and B, respectively, from the rule number 4(V).

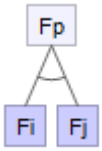

Figure 42 – Local View A from rule 4(V)

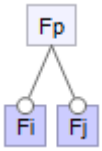

Figure 43 – Local View B from rule 4(V)

First, $childSets_A$ and $childSets_B$ are determined.

$childSets_A$ = { { i }, { j } }.

$childSets_B$ = { { }, { i }, { j }, { i, j } }.

Second, *fs* X and *fs* Y representations are constructed as Table 42 and Table 43.

Table 42 - Feature Selection Map *fs* X

row number		**i**	**j**
1	***fs* { i }**	+	-
2	***fs* { j }**	-	+

Table 43 - Feature Selection Map *fs* Y

row number		**i**	**j**
1	***fs* { }**	-	-
2	***fs* { i }**	+	-
3	***fs* { j }**	-	+
4	***fs* { i, j }**	+	+

Next, combination for each *fs* X and *fs* Y is constructed. Table 44 shows the conflict situations that cannot be resolved.

Table 44 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**
1	*fs* { i }	+	-
2	*fs* { }	-	-
3	***fs* { i } ⊗ *fs* { }**	!	-
4	*fs* { i }	+	-
5	*fs* { j }	-	+
6	***fs* { i } ⊗ *fs* { j }**	!	!
7	*fs* { i }	+	-
8	*fs* { i, j }	+	+
9	***fs* { i } ⊗ *fs* { i, j }**	+	!
10	*fs* { j }	-	+
11	*fs* { }	-	-
12	***fs* { j } ⊗ *fs* { }**	-	!
13	*fs* { j }	-	+
14	*fs* { i, j }	+	+
15	***fs* { j } ⊗ *fs* { i, j }**	!	+

From the results of conform operations, $childSets_R$ is constructed as follows: $childSets_R$ = { { i }, { j } }.

When the FM R is built, the model given in Figure 44 is obtained.

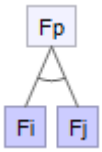

Figure 44 – Resolution of Figure 42 and Figure 43

Regarding Rule 4(VI):

Figure 45 and Figure 46 represent the local views A and B, respectively, from the rule number 4(VI).

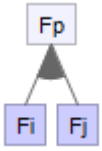

Figure 45 – Local View A from rule 4(VI)

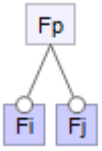

Figure 46 – Local View B from rule 4(VI)

First, $childSets_A$ and $childSets_B$ are determined.

$childSets_A$ = { { i }, { j }, { i, j } }.

$childSets_B$ = { { }, { i }, { j }, { i, j } }.

Second, *fs* X and *fs* Y representations are constructed as Table 45 and Table 46.

Table 45 - Feature Selection Map *fs* X

row number		**i**	**j**
1	***fs* { i }**	+	-
2	***fs* { j }**	-	+
3	***fs* { i, j }**	+	+

Table 46 - Feature Selection Map *fs* Y

row number		**i**	**j**
1	***fs* { }**	-	-
2	***fs* { i }**	+	-
3	***fs* { j }**	-	+
4	***fs* { i, j }**	+	+

Next, combination for each *fs* X and *fs* Y is constructed. Table 47 shows the conflict situations that cannot be resolved.

Table 47 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**
1	*fs* { i }	+	-
2	*fs* { }	-	-
3	***fs* { i } ⊗ *fs* { }**	!	-
4	*fs* { i }	+	-
5	*fs* { j }	-	+
6	***fs* { i } ⊗ *fs* { j }**	!	!
7	*fs* { i }	+	-
8	*fs* { i, j }	+	+
9	***fs* { i } ⊗ *fs* { i, j }**	+	!
10	*fs* { j }	-	+
11	*fs* { }	-	-
12	***fs* { j } ⊗ *fs* { }**	-	!
13	*fs* { j }	-	+
14	*fs* { i, j }	+	+
15	***fs* { j } ⊗ *fs* { i, j }**	!	+
16	*fs* { i, j }	+	+
17	*fs* { }	-	-
18	***fs* { i, j } ⊗ *fs* { }**	!	!

From the results of *conform* operations, $childSets_R$ is constructed as follows: $childSets_R$ = { { i }, { j }, { i, j } }.

When the FM R is built, the model given in Figure 47 is obtained.

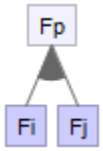

Figure 47 – Resolution of Figure 45 and Figure 46

Regarding Rule 5(II):

Figure 48 and Figure 49 represent the local views A and B, respectively, from the rule number 5(II).

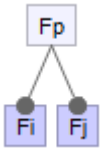

Figure 48 – Local View A from rule 5(II)

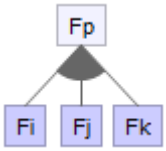

Figure 49 – Local View B from rule 5(II)

First, $childSets_A$ and $childSets_B$ are determined.

$childSets_A$ = { { i, j } }.

$childSets_B$ = { { i }, { j }, { i, j }, { i, k }, { j, k }, { i, j ,k } }.

Second, *fs* X and *fs* Y representations are constructed as Table 48 and Table 49.

Table 48 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**
1	***fs* { i, j }**	+	+	/

Table 49 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**
1	***fs* { i }**	+	-	-
2	***fs* { j }**	-	+	-
3	***fs* { i, j }**	+	+	-
4	***fs* { i, k }**	+	-	+
5	***fs* { j, k }**	-	+	+
6	***fs* { i, j, k }**	+	+	+

Next, combination for each *fs* X and *fs* Y is constructed. Table 50 shows the conflict situations that cannot be resolved.

Table 50 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**	**k**
1	*fs* { i, j }	+	+	/
2	*fs* { i }	+	-	-
3	***fs* { i, j } ⊗ *fs* { i }**	+	!	-
4	*fs* { i, j }	+	+	/
5	*fs* { j }	-	+	-
6	***fs* { i, j } ⊗ *fs* { j }**	!	+	-
7	*fs* { i, j }	+	+	/
8	*fs* { i, k }	+	-	+
9	***fs* { i, j } ⊗ *fs* { i, k }**	+	!	+
10	*fs* { i, j }	+	+	/
11	*fs* { j, k }	-	+	+
12	***fs* { i, j } ⊗ *fs* { j, k }**	!	+	+

Table 51 shows the *fs* Y's that are preserved.

Table 51 - *fs* X ⊗ *fs* Y = *fs* Y

row number		**i**	**j**	**k**
1	*fs* { i, j }	+	+	/
2	*fs* { i, j, k }	+	+	+
3	***fs* { i, j } ⊗ *fs* { i, j, k } = *fs* { i, j, k }**	+	+	+

From the results of *conform* operations, $childSets_R$ is constructed as follows: $childSets_R$ = { { i, j }, { i, j, k } }.

When the FM R is built, the model given in Figure 50 is obtained.

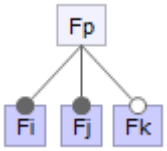

Figure 50 – Resolution of Figure 48 and Figure 49

Regarding Rule 5(III):

Figure 51 and Figure 52 represent the local views A and B, respectively, from the rule number 5(III).

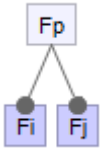

Figure 51 – Local View A from rule 5(III)

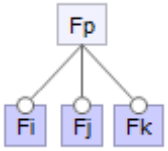

Figure 52 – Local View B from rule 5(III)

First, $childSets_A$ and $childSets_B$ are determined.

$childSets_A$ = { { i, j } }.

$childSets_B$ = { { }, { i }, { j }, { i, j }, { i, k }, { j, k }, { i, j ,k } }.

Second, *fs* X and *fs* Y representations are constructed as Table 52 and Table 53.

Table 52 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**
1	***fs* { i, j }**	+	+	/

Table 53 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**
1	***fs* { }**	-	-	-
2	***fs* { i }**	+	-	-
3	***fs* { j }**	-	+	-
4	***fs* { i, j }**	+	+	-
5	***fs* { i, k }**	+	-	+
6	***fs* { j, k }**	-	+	+
7	***fs* { i, j, k }**	+	+	+

Next, combination for each *fs* X and *fs* Y is constructed. Table 54 shows the conflict situations that cannot be resolved.

Table 54 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**	**k**
1	*fs* { i, j }	+	+	/
2	*fs* { }	-	-	-
3	***fs* { i, j } ⊗ *fs* { }**	!	!	-
4	*fs* { i, j }	+	+	/
5	*fs* { i }	+	-	-
6	***fs* { i, j } ⊗ *fs* { i }**	+	!	-
7	*fs* { i, j }	+	+	/
8	*fs* { j }	-	+	-
9	***fs* { i, j } ⊗ *fs* { j }**	!	+	-
10	*fs* { i, j }	+	+	/
11	*fs* { i, k }	+	-	+
12	***fs* { i, j } ⊗ *fs* { i, k }**	+	!	+
13	*fs* { i, j }	+	+	/
14	*fs* { j, k }	-	+	+
15	***fs* { i, j } ⊗ *fs* { j, k }**	!	+	+

Table 55 shows the *fs* Y's that are preserved.

Table 55 - *fs* X ⊗ *fs* Y = *fs* Y

row number		**i**	**j**	**k**
1	*fs* { i, j }	+	+	/
2	*fs* { i, j, k }	+	+	+
3	***fs* { i, j } ⊗ *fs* { i, j, k } = *fs* { i, j, k }**	+	+	+

From the results of *conform* operations, $childSets_R$ is constructed as follows: $childSets_R$ = { { i, j }, { i, j, k } }.

When the FM R is built, the model given in Figure 53 is obtained.

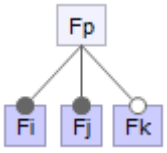

Figure 53 – Resolution of Figure 51 and Figure 52

Regarding Rule 5(IV):

Figure 54 and Figure 55 represent the local views A and B, respectively, from the rule number 5(IV).

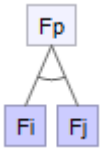

Figure 54 – Local View A from rule 5(IV)

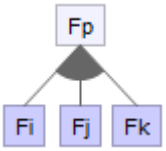

Figure 55 – Local View B from rule 5(IV)

First, $childSets_A$ and $childSets_B$ are determined.

$childSets_A$ = { { i }, { j } }.

$childSets_B$ = { { i }, { j }, { i, j }, { i, k }, { j, k }, { i, j ,k } }.

Second, *fs* X and *fs* Y representations are constructed as Table 56 and Table 57.

Table 56 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**
1	***fs* { i }**	+	-	/
2	***fs* { j }**	-	+	/

Table 57 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**
1	***fs* { i }**	+	-	-
2	***fs* { j }**	-	+	-
3	***fs* { i, j }**	+	+	-
4	***fs* { i, k }**	+	-	+
5	***fs* { j, k }**	-	+	+
6	***fs* { i, j, k }**	+	+	+

Next, combination for each *fs* X and *fs* Y is constructed. Table 58 shows the conflict situations that cannot be resolved.

Table 58 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**	**k**
1	*fs* { i }	+	-	/
2	*fs* { j }	-	+	-
3	***fs* { i } ⊗ *fs* { j }**	!	!	-
4	*fs* { i }	+	-	/
5	*fs* { i, j }	+	+	-
6	***fs* { i } ⊗ *fs* { i, j }**	+	!	-
7	*fs* { i }	+	-	/
8	*fs* { j, k }	-	+	+
9	***fs* { i } ⊗ *fs* { j, k }**	!	!	+
10	*fs* { i }	+	-	/
11	*fs* { i, j, k }	+	+	+
12	***fs* { i } ⊗ *fs* { i, j, k }**	+	!	+
13	*fs* { j }	-	+	/
14	*fs* { i, j }	+	+	-
15	***fs* { j } ⊗ *fs* { i, j }**	!	+	-
16	*fs* { j }	-	+	/
17	*fs* { i, k }	+	-	+
18	***fs* { j } ⊗ *fs* { i, k }**	!	!	+
19	*fs* { j }	-	+	/
20	*fs* { i, j, k }	+	+	+
21	***fs* { j } ⊗ *fs* { i, j, k }**	!	+	+

Table 59 shows the *fs* Y's that are preserved.

Table 59 - *fs* X ⊗ *fs* Y = *fs* Y

row number		**i**	**j**	**k**
1	*fs* { i }	+	-	/
2	*fs* { i, k }	+	-	+
3	***fs* { i } ⊗ *fs* { i, k } = *fs* { i, k }**	+	-	+
4	*fs* { j }	-	+	/
5	*fs* { j, k }	-	+	+
6	***fs* { j } ⊗ *fs* { j, k } = *fs* { j, k }**	-	+	+

From the results of *conform* operations, $childSets_R$ is constructed as follows: $childSets_R$ = { { i }, { j }, { i, k }, { j, k } }.

When the FM R is built, the model given in Figure 56 is obtained.

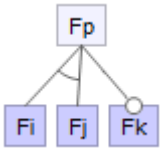

Figure 56 – Resolution of Figure 54 and Figure 55

Regarding Rule 5(V):

Figure 57 and Figure 58 represent the local views A and B, respectively, from the rule number 5(V).

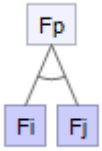

Figure 57 – Local View A from rule 5(V)

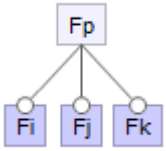

Figure 58 – Local View B from rule 5(V)

First, $childSets_A$ and $childSets_B$ are determined.

$childSets_A$ = { { i }, { j } }.

$childSets_B$ = { { }, { i }, { j }, { k }, { i, j }, { i, k }, { j, k }, { i, j, k } }.

Second, *fs* X and *fs* Y representations are constructed as Table 60 and Table 61.

Table 60 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**
1	***fs* { i }**	+	-	/
2	***fs* { j }**	-	+	/

Table 61 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**
1	***fs* { }**	-	-	-
2	***fs* { i }**	+	-	-
3	***fs* { j }**	-	+	-
4	***fs* { k }**	-	-	+
5	***fs* { i, j }**	+	+	-
6	***fs* { i, k }**	+	-	+
7	***fs* { j, k }**	-	+	+
8	***fs* { i, j, k }**	+	+	+

Next, combination for each *fs* X and *fs* Y is constructed. Table 62 - shows the conflict situations that cannot be resolved.

Table 62 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**	**k**
1	*fs* { i }	+	-	/
2	*fs* { }	-	-	-
3	***fs* { i } ⊗ *fs* { }**	!	-	-
4	*fs* { i }	+	-	/
5	*fs* { j }	-	+	-
6	***fs* { i } ⊗ *fs* { j }**	!	!	-
7	*fs* { i }	+	-	/
8	*fs* { k }	-	-	+
9	***fs* { i } ⊗ *fs* { k }**	!	-	+
10	*fs* { i }	+	-	/
11	*fs* { i, j }	+	+	-
12	***fs* { i } ⊗ *fs* { i, j }**	+	!	-
13	*fs* { i }	+	-	/
14	*fs* { j, k }	-	+	+
15	***fs* { i } ⊗ *fs* { j, k }**	!	!	+
16	*fs* { i }	+	-	/
17	*fs* { i, j, k }	+	+	+
18	***fs* { i } ⊗ *fs* { i, j, k }**	+	!	+
19	*fs* { j }	-	+	/
20	*fs* { }	-	-	-
21	***fs* { j } ⊗ *fs* { }**	-	!	-
22	*fs* { j }	-	+	/
23	*fs* { k }	-	-	+
24	***fs* { j } ⊗ *fs* { k }**	-	!	+
25	*fs* { j }	-	+	/
26	*fs* { i, j }	+	+	-
27	***fs* { j } ⊗ *fs* { i, j }**	!	+	-
28	*fs* { j }	-	+	/
29	*fs* { i, k }	+	-	+
30	***fs* { j } ⊗ *fs* { i, k }**	!	!	+
31	*fs* { j }	-	+	/
32	*fs* { i, j, k }	+	+	+
33	***fs* { j } ⊗ *fs* { i, j, k }**	!	+	+

Table 63 shows the *fs* Y's that are preserved.

Table 63 - *fs* X ⊗ *fs* Y = *fs* Y

row number		**i**	**j**	**k**
1	*fs* { i }	+	-	/
2	*fs* { i, k }	+	-	+
3	***fs* { i } ⊗ *fs* { i, k } = *fs* { i, k }**	+	-	+
4	*fs* { j }	-	+	/
5	*fs* { j, k }	-	+	+
6	***fs* { j } ⊗ *fs* { j, k } = *fs* { j, k }**	-	+	+

From the results of *conform* operations, $childSets_R$ is constructed as follows: $childSets_R$ = { { i }, { j }, { i, k }, { j, k } }.

When the FM R is built, the model given in Figure 59 is obtained.

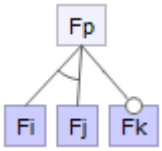

Figure 59 – Resolution of Figure 57 and Figure 58

Regarding Rule 5(VI):

Figure 60 and Figure 61 represent the local views A and B, respectively, from the rule number 5(VI).

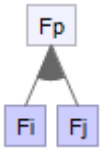

Figure 60 – Local View A from rule 5(VI)

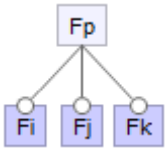

Figure 61 – Local View B from rule 5(VI)

First, $childSets_A$ and $childSets_B$ are determined.

$childSets_A$ = { { i }, { j }, { i, j } }.

$childSets_B$ = { { }, { i }, { j }, { k }, { i, j }, { i, k }, { j, k }, { i, j, k } }.

Second, *fs* X and *fs* Y representations are constructed as Table 64 and Table 65.

Table 64 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**
1	***fs* { i }**	+	-	/
2	***fs* { j }**	-	+	/
3	***fs* { i, j }**	+	+	/

Table 65 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**
1	***fs* { }**	-	-	-
2	***fs* { i }**	+	-	-
3	***fs* { j }**	-	+	-
4	***fs* { k }**	-	-	+
5	***fs* { i, j }**	+	+	-
6	***fs* { i, k }**	+	-	+
7	***fs* { j, k }**	-	+	+
8	***fs* { i, j, k }**	+	+	+

Next, combination for each *fs* X and *fs* Y is constructed. Table 66 shows the conflict situations that cannot be resolved.

Table 66 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**	**k**
1	*fs* { i }	+	-	/
2	*fs* { }	-	-	-
3	***fs* { i } ⊗ *fs* { }**	!	-	-
4	*fs* { i }	+	-	/
5	*fs* { j }	-	+	-
6	***fs* { i } ⊗ *fs* { j }**	!	!	-
7	*fs* { i }	+	-	/
8	*fs* { k }	-	-	+
9	***fs* { i } ⊗ *fs* { k }**	!	-	+
10	*fs* { i }	+	-	/
11	*fs* { i, j }	+	+	-
12	***fs* { i } ⊗ *fs* { i, j }**	+	!	-
13	*fs* { i }	+	-	/
14	*fs* { j, k }	-	+	+
15	***fs* { i } ⊗ *fs* { j, k }**	!	!	+
16	*fs* { i }	+	-	/
17	*fs* { i, j, k }	+	+	+
18	***fs* { i } ⊗ *fs* { i, j, k }**	+	!	+
19	*fs* { j }	-	+	/
20	*fs* { }	-	-	-
21	***fs* { j } ⊗ *fs* { }**	-	!	-
22	*fs* { j }	-	+	/
23	*fs* { k }	-	-	+
24	***fs* { j } ⊗ *fs* { k }**	-	!	+
25	*fs* { j }	-	+	/
26	*fs* { i, j }	+	+	-
27	***fs* { j } ⊗ *fs* { i, j }**	!	+	-
28	*fs* { j }	-	+	/
29	*fs* { i, k }	+	-	+
30	***fs* { j } ⊗ *fs* { i, k }**	!	!	+
31	*fs* { j }	-	+	/
32	*fs* { i, j, k }	+	+	+
33	***fs* { j } ⊗ *fs* { i, j, k }**	!	+	+
34	*fs* { i, j }	+	+	/
35	*fs* { }	-	-	-
36	***fs* { i, j } ⊗ *fs* { }**	!	!	-
37	*fs* { i, j }	+	+	/
38	*fs* { k }	-	-	+
39	***fs* { i, j } ⊗ *fs* { k }**	!	!	+
40	*fs* { i, j }	+	+	/
41	*fs* { i, k }	+	-	+
42	***fs* { i, j } ⊗ *fs* { i, k }**	+	!	+
43	*fs* { i, j }	+	+	/
44	*fs* { j, k }	-	+	+
45	***fs* { i, j } ⊗ *fs* { j, k }**	!	+	+

Table 67 shows the *fs* Y's that are preserved.

Table 67 - *fs* X ⊗ *fs* Y = *fs* Y

row number		**i**	**j**	**k**
1	*fs* { i }	+	-	/
2	*fs* { i, k }	+	-	+
3	***fs* { i } ⊗ *fs* { i, k } = *fs* { i, k }**	+	-	+
4	*fs* { j }	-	+	/
5	*fs* { j, k }	-	+	+
6	***fs* { j } ⊗ *fs* { j, k } = *fs* { j, k }**	-	+	+
7	*fs* { i, j }	+	+	/
8	*fs* { i, j, k }	+	+	+
9	***fs* { i, j } ⊗ *fs* { i, j, k } = *fs* { i, j, k }**	+	+	+

From the results of *conform* operations, $childSets_R$ is constructed as follows: $childSets_R$ = { { i }, { j }, { i, j }, { i, k }, { j, k }, { i, j, k } }.

When the FM R is built, the model given in Figure 62 is obtained.

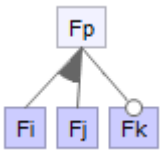

Figure 62 – Resolution of Figure 60 and Figure 61

Regarding Rule 5(VII):

Figure 63 and Figure 64 represent the local views A and B, respectively, from the rule number 5(VII).

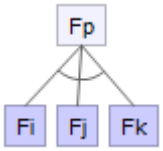

Figure 63 – Local View A from rule 5(VII)

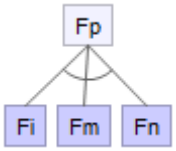

Figure 64 – Local View B from rule 5(VII)

First, $childSets_A$ and $childSets_B$ are determined.

$childSets_A$ = { { i }, { j }, { k } }.

$childSets_B$ = { { i }, { m }, { n } }.

Second, *fs* X and *fs* Y representations are constructed as Table 68 and Table 69.

Table 68 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**	**m**	**n**
1	***fs* { i }**	+	-	-	/	/
2	***fs* { j }**	-	+	-	/	/
3	***fs* { k }**	-	-	+	/	/

Table 69 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**	**m**	**n**
1	***fs* { i }**	+	/	/	-	-
2	***fs* { m }**	-	/	/	+	-
3	***fs* { n }**	-	/	/	-	+

Next, combination for each *fs* X and *fs* Y is constructed. Table 70 presents the results of combinations where *fs* X's and *fs* Y's are not preserved.

Table 70 - *fs* X ⊗ *fs* Y

row number		**i**	**j**	**k**	**m**	**n**
1	*fs* { i }	+	-	-	/	/
2	*fs* { m }	-	/	/	+	-
3	***fs* { i } ⊗ *fs* { m }**	!	-	-	+	-
4	*fs* { i }	+	-	-	/	/
5	*fs* { n }	-	/	/	-	+
6	***fs* { i } ⊗ *fs* { n }**	!	-	-	-	+
7	*fs* { j }	-	+	-	/	/
8	*fs* { i }	+	/	/	-	-
9	***fs* { j } ⊗ *fs* { i }**	!	+	-	+	-
10	*fs* { j }	-	+	-	/	/
11	*fs* { m }	-	/	/	+	-
12	***fs* { j } ⊗ *fs* { m } = *fs* { j, m }**	-	+	-	+	-
13	*fs* { j }	-	+	-	/	/
14	*fs* { n }	-	/	/	-	+
15	***fs* { j } ⊗ *fs* { n } = *fs* { j, n }**	-	+	-	-	+
16	*fs* { k }	-	-	+	/	/
17	*fs* { i }	+	/	/	-	-
18	***fs* { k } ⊗ *fs* { i }**	!	-	+	-	-
19	*fs* { k }	-	-	+	/	/
20	*fs* { m }	-	/	/	+	-
21	***fs* { k } ⊗ *fs* { m } = *fs* { k, m }**	-	-	+	+	-
22	*fs* { k }	-	-	+	/	/
23	*fs* { n }	-	/	/	-	+
24	***fs* { k } ⊗ *fs* { n } = *fs* { k, n }**	-	-	+	-	+

On the 12th row of Table 70, result of *fs* { j } ⊗ *fs* { m } operation is equal to *fs* { j, m } which is not equal to *fs* { j } or *fs* { m }. Similar situations are presented on the 15th, 21st and 24th rows. This means that on these rows, *fs* X's and *fs* Y's are not preserved.

From the results of *conform* operations, $childSets_R$ is constructed as follows: $childSets_R$ = { { i } }.

When the FM R is built, the model given in Figure 65 is obtained.

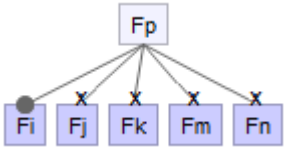

Figure 65 – Resolution of Figure 63 and Figure 64

From the $childSets_R$'s that are obtained at the end of the solutions, resolution FMs can be constructed by following the rules given in chapter 4.

In [4], all default conflict resolution rules produces the same results as obtained by given approach.

3.4 Case 3: Difficult-to-Resolve Views

As mentioned above, some default conflict resolution rules for multi-view FMs are proposed by Mannion *et al.* in [4]. But they do not suggest a solution for some cases. They remark that "the complexity of the conflict is too great to propose an automatic solution and require human intervention" for cases in rule number 1(I), 4(I), 5(I), 5(VIII) and 5(IX). With *merging by conformance*, proposing reasonable resolutions to the rules number 5(VIII) and 5(IX) in [4] is possible.

Regarding Rule 5(VIII):

Figure 66 and Figure 67 represent the local views A and B, respectively. These are the views from the rule number 5(VIII).

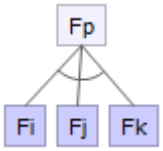

Figure 66 – Local View A from rule 5(VIII)

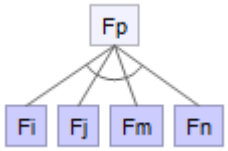

Figure 67 – Local View B from rule 5(VIII)

At first, $childSets_A$ and $childSets_B$ are determined.

$childSets_A$ = { { i }, { j }, { k } }.

$childSets_B$ = { { i }, { j }, { m }, { n } }.

Second, *fs* X and *fs* Y representations are constructed as Table 71 and Table 72.

Table 71 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**	**m**	**n**
1	***fs* { i }**	+	-	-	/	/
2	***fs* { j }**	-	+	-	/	/
3	***fs* { k }**	-	-	+	/	/

Table 72 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**	**m**	**n**
1	***fs* { i }**	+	-	/	-	-
2	***fs* { j }**	-	+	/	-	-
3	***fs* { m }**	-	-	/	+	-
4	***fs* { n }**	-	-	/	-	+

Then, combination for each *fs* X and *fs* Y is constructed. Table 73 presents the results of combinations where *fs* X's and *fs* Y's are not preserved.

Table 73 - *fs* X ⊗ *fs* Y

row number		**i**	**j**	**k**	**m**	**n**
1	*fs* { i }	+	-	-	/	/
2	*fs* { j }	-	+	/	-	-
3	***fs* { i } ⊗ *fs* { j }**	!	!	-	-	-
4	*fs* { i }	+	-	-	/	/
5	*fs* { m }	-	-	/	+	-
6	***fs* { i } ⊗ *fs* { m }**	!	-	-	+	-
7	*fs* { i }	+	-	-	/	/
8	*fs* { n }	-	-	/	-	+
9	***fs* { i } ⊗ *fs* { n }**	!	-	-	-	+
10	*fs* { j }	-	+	-	/	/
11	*fs* { m }	-	-	/	+	-
12	***fs* { j } ⊗ *fs* { m }**	-	!	-	+	-
13	*fs* { j }	-	+	-	/	/
14	*fs* { n }	-	-	/	-	+
15	***fs* { j } ⊗ *fs* { n }**	-	!	-	-	+
16	*fs* { k }	-	-	+	/	/
17	*fs* { i }	+	-	/	-	-
18	***fs* { k } ⊗ *fs* { i }**	!	-	+	-	-
19	*fs* { k }	-	-	+	/	/
20	*fs* { j }	-	+	/	-	-
21	***fs* { k } ⊗ *fs* { j }**	-	!	+	-	-
22	*fs* { k }	-	-	+	/	/
23	*fs* { m }	-	-	/	+	-
24	***fs* { k } ⊗ *fs* { m } = *fs* { k, m }**	-	-	+	+	-
25	*fs* { k }	-	-	+	/	/
26	*fs* { n }	-	-	/	-	+
27	***fs* { k } ⊗ *fs* { n } = *fs* { k, n }**	-	-	+	-	+

On the 24th row of Table 73, result of *fs* { k } ⊗ *fs* { m } operation is equal to *fs* { k, m } which is not equal to *fs* { k } or *fs* { m }. Similarly, on the 27th row of Table 73, result of *fs* { k } ⊗ *fs* { n } operation is equal to *fs* { k, n } which is not equal to *fs* { k } or *fs* { n }. This means that on the 24th and 27th rows of Table 73, *fs* X's and *fs* Y's are not preserved.

After constructing combinations, $conform_A$ and $conform_B$ operations are performed and $childSets_R$ is obtained as follows:

$childSets_R$ = { { i }, { j } }.

When the FM R is built, the model given in Figure 68 is obtained.

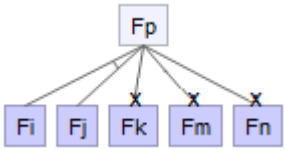

Figure 68 – Resolution of Figure 66 and Figure 67

Regarding Rule 5(IX):

Figure 69 and Figure 70 represent the local views A and B, respectively, from the rule number 5(IX).

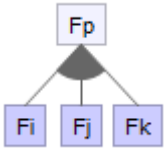

Figure 69 – Local View A from rule 5(IX)

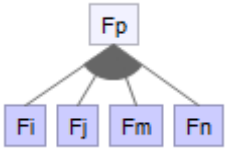

Figure 70 – Local View B from rule 5(IX)

At first, $childSets_A$ and $childSets_B$ are determined.

$childSets_A$ = { { i }, { j }, { k }, { i, j }, { i, k }, { j, k }, { i, j, k } }.

$childSets_B$ = { { i }, { j }, { m }, { n }, { i, j }, { i, m }, { i, n }, { j, m }, { j, n }, { m, n }, { i, j, m }, { i, j, n }, { i, m, n }, { j, m, n }, { i, j, m, n } }.

Then, *fs* X and *fs* Y representations are constructed as Table 74 and Table 75.

Table 74 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**	**m**	**n**
1	***fs* { i }**	+	-	-	/	/
2	***fs* { j }**	-	+	-	/	/
3	***fs* { k }**	-	-	+	/	/
4	***fs* { i, j }**	+	+	-	/	/
5	***fs* { i, k }**	+	-	+	/	/
6	***fs* { j, k }**	-	+	+	/	/
7	***fs* { i, j, k }**	+	+	+	/	/

Table 75 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**	**m**	**n**
1	***fs* { i }**	+	-	/	-	-
2	***fs* { j }**	-	+	/	-	-
3	***fs* { m }**	-	-	/	+	-
4	***fs* { n }**	-	-	/	-	+
5	***fs* { i, j }**	+	+	/	-	-
6	***fs* { i, m }**	+	-	/	+	-
7	***fs* { i, n }**	+	-	/	-	+
8	***fs* { j, m }**	-	+	/	+	-
9	***fs* { j, n }**	-	+	/	-	+
10	***fs* { m, n }**	-	-	/	+	+
11	***fs* { i, j, m }**	+	+	/	+	-
12	***fs* { i, j, n }**	+	+	/	-	+
13	***fs* { i, m, n }**	+	-	/	+	+
14	***fs* { j, m, n }**	-	+	/	+	+
15	***fs* { i, j, m, n }**	+	+	/	+	+

Then, combination for each *fs* X and *fs* Y is constructed. Table 76 presents the results of combinations where *fs* X's and *fs* Y's are not preserved.

Table 76 - *fs* X ⊗ *fs* Y

row number		**i**	**j**	**k**	**m**	**n**
1	*fs* { i }	+	-	-	/	/
2	*fs* { j }	-	+	/	-	-
3	***fs* { i } ⊗ *fs* { j }**	!	!	-	-	-
4	*fs* { i }	+	-	-	/	/
5	*fs* { m }	-	-	/	+	-
6	***fs* { i } ⊗ *fs* { m }**	!	-	-	+	-
7	*fs* { i }	+	-	-	/	/
8	*fs* { n }	-	-	/	-	+
9	***fs* { i } ⊗ *fs* { n }**	!	-	-	-	+
10	*fs* { i }	+	-	-	/	/
11	*fs* { i, j }	+	+	/	-	-
12	***fs* { i } ⊗ *fs* { i, j }**	+	!	-	-	-
13	*fs* { i }	+	-	-	/	/
14	*fs* { j, m }	-	+	/	+	-
15	***fs* { i } ⊗ *fs* { j, m }**	!	!	-	+	-
16	*fs* { i }	+	-	-	/	/
17	*fs* { j, n }	-	+	/	-	+
18	***fs* { i } ⊗ *fs* { j, n }**	!	!	-	-	+
19	*fs* { i }	+	-	-	/	/
20	*fs* { m, n }	-	-	/	+	+
21	***fs* { i } ⊗ *fs* { m, n }**	!	-	-	+	+

Table 76 (continued)

row number		**i**	**j**	**k**	**m**	**n**
22	*fs* { i }	+	-	-	/	/
23	*fs* { i, j, m }	+	+	/	+	-
24	***fs* { i } ⊗ *fs* { i, j, m }**	+	!	-	+	-
25	*fs* { i }	+	-	-	/	/
26	*fs* { i, j, n }	+	+	/	-	+
27	***fs* { i } ⊗ *fs* { i, j, n }**	+	!	-	-	+
28	*fs* { i }	+	-	-	/	/
29	*fs* { j, m, n }	-	+	/	+	+
30	***fs* { i } ⊗ *fs* { j, m, n }**	!	!	-	+	+
31	*fs* { i }	+	-	-	/	/
32	*fs* { i, j, m, n }	+	+	/	+	+
33	***fs* { i } ⊗ *fs* { i, j, m, n }**	+	!	-	+	+
34	*fs* { j }	-	+	-	/	/
35	*fs* { m }	-	-	/	+	-
36	***fs* { j } ⊗ *fs* { m }**	-	!	-	+	-
37	*fs* { j }	-	+	-	/	/
38	*fs* { n }	-	-	/	-	+
39	***fs* { j } ⊗ *fs* { n }**	-	!	-	-	+
40	*fs* { j }	-	+	-	/	/
41	*fs* { i, j }	+	+	/	-	-
42	***fs* { j } ⊗ *fs* { i, j }**	!	+	-	-	-
43	*fs* { j }	-	+	-	/	/
44	*fs* { i, m }	+	-	/	+	-
45	***fs* { j } ⊗ *fs* { i, m }**	!	!	-	+	-
46	*fs* { j }	-	+	-	/	/
47	*fs* { i, n }	+	-	/	-	+
48	***fs* { j } ⊗ *fs* { i, n }**	!	!	-	-	+
49	*fs* { j }	-	+	-	/	/
50	*fs* { m, n }	-	-	/	+	+
51	***fs* { j } ⊗ *fs* { m, n }**	-	!	-	+	+
52	*fs* { j }	-	+	-	/	/
53	*fs* { i, j, m }	+	+	/	+	-
54	***fs* { j } ⊗ *fs* { i, j, m }**	!	+	-	+	-
55	*fs* { j }	-	+	-	/	/
56	*fs* { i, j, n }	+	+	/	-	+
57	***fs* { j } ⊗ *fs* { i, j, n }**	!	+	-	-	+
58	*fs* { j }	-	+	-	/	/
59	*fs* { i, m, n }	+	-	/	+	+
60	***fs* { j } ⊗ *fs* { i, m, n }**	!	!	-	+	+
61	*fs* { j }	-	+	-	/	/
62	*fs* { i, j, m, n }	+	+	/	+	+
63	***fs* { j } ⊗ *fs* { i, j, m, n }**	!	+	-	+	+
64	*fs* { k }	-	-	+	/	/

Table 76 (continued)

row number		**i**	**j**	**k**	**m**	**n**
65	*fs* { i }	+	-	/	-	-
66	***fs* { k } ⊗ *fs* { i }**	!	-	+	-	-
67	*fs* { k }	-	-	+	/	/
68	*fs* { j }	-	+	/	-	-
69	***fs* { k } ⊗ *fs* { j }**	-	!	+	-	-
70	*fs* { k }	-	-	+	/	/
71	*fs* { m }	-	-	/	+	-
72	***fs* { k } ⊗ *fs* { m } = *fs* { k, m }**	-	-	+	+	-
73	*fs* { k }	-	-	+	/	/
74	*fs* { n }	-	-	/	-	+
75	***fs* { k } ⊗ *fs* { n } = *fs* { k, n }**	-	-	+	-	+
76	*fs* { k }	-	-	+	/	/
77	*fs* { i, j }	+	+	/	-	-
78	***fs* { k } ⊗ *fs* { i, j }**	!	!	+	-	-
79	*fs* { k }	-	-	+	/	/
80	*fs* { i, m }	+	-	/	+	-
81	***fs* { k } ⊗ *fs* { i, m }**	!	-	+	+	-
82	*fs* { k }	-	-	+	/	/
83	*fs* { i, n }	+	-	/	-	+
84	***fs* { k } ⊗ *fs* { i, n }**	!	-	+	-	+
85	*fs* { k }	-	-	+	/	/
86	*fs* { j, m }	-	+	/	+	-
87	***fs* { k } ⊗ *fs* { j, m }**	-	!	+	+	-
88	*fs* { k }	-	-	+	/	/
89	*fs* { j, n }	-	+	/	-	+
90	***fs* { k } ⊗ *fs* { j, n }**	-	!	+	-	+
91	*fs* { k }	-	-	+	/	/
92	*fs* { m, n }	-	-	/	+	+
93	***fs* { k } ⊗ *fs* { m, n } = *fs* { k, m, n }**	-	-	+	+	+
94	*fs* { k }	-	-	+	/	/
95	*fs* { i, j, m }	+	+	/	+	-
96	***fs* { k } ⊗ *fs* { i, j, m }**	!	!	+	+	-
97	*fs* { k }	-	-	+	/	/
98	*fs* { i, j, n }	+	+	/	-	+
99	***fs* { k } ⊗ *fs* { i, j, n }**	!	!	+	-	+
100	*fs* { k }	-	-	+	/	/
101	*fs* { i, m, n }	+	-	/	+	+
102	***fs* { k } ⊗ *fs* { i, m, n }**	!	-	+	+	+
103	*fs* { k }	-	-	+	/	/
104	*fs* { j, m, n }	-	+	/	+	+
105	***fs* { k } ⊗ *fs* { j, m, n }**	-	!	+	+	+
106	*fs* { k }	-	-	+	/	/
107	*fs* { i, j, m, n }	+	+	/	+	+

Table 76 (continued)

row number		**i**	**j**	**k**	**m**	**n**
108	***fs* { k } ⊗ *fs* { i, j, m, n }**	!	!	+	+	+
109	*fs* { i, j }	+	+	-	/	/
110	*fs* { m }	-	-	/	+	-
111	***fs* { i, j } ⊗ *fs* { m }**	!	!	-	+	-
112	*fs* { i, j }	+	+	-	/	/
113	*fs* { n }	-	-	/	-	+
114	***fs* { i, j } ⊗ *fs* { n }**	!	!	-	-	+
115	*fs* { i, j }	+	+	-	/	/
116	*fs* { i, m }	+	-	/	+	-
117	***fs* { i, j } ⊗ *fs* { i, m }**	+	!	-	+	-
118	*fs* { i, j }	+	+	-	/	/
119	*fs* { i, n }	+	-	/	-	+
120	***fs* { i, j } ⊗ *fs* { i, n }**	+	!	-	-	+
121	*fs* { i, j }	+	+	-	/	/
122	*fs* { j, m }	-	+	/	+	-
123	***fs* { i, j } ⊗ *fs* { j, m }**	!	+	-	+	-
124	*fs* { i, j }	+	+	-	/	/
125	*fs* { j, n }	-	+	/	-	+
126	***fs* { i, j } ⊗ *fs* { j, n }**	!	+	-	-	+
127	*fs* { i, j }	+	+	-	/	/
128	*fs* { m, n }	-	-	/	+	+
129	***fs* { i, j } ⊗ *fs* { m, n }**	!	!	-	+	+
130	*fs* { i, j }	+	+	-	/	/
131	*fs* { i, m, n }	+	-	/	+	+
132	***fs* { i, j } ⊗ *fs* { i, m, n }**	+	!	-	+	+
133	*fs* { i, j }	+	+	-	/	/
134	*fs* { j, m, n }	-	+	/	+	+
135	***fs* { i, j } ⊗ *fs* { j, m, n }**	!	+	-	+	+
136	*fs* { i, k }	+	-	+	/	/
137	*fs* { j }	-	+	/	-	-
138	***fs* { i, k } ⊗ *fs* { j }**	!	!	+	-	-
139	*fs* { i, k }	+	-	+	/	/
140	*fs* { m }	-	-	/	+	-
141	***fs* { i, k } ⊗ *fs* { m }**	!	-	+	+	-
142	*fs* { i, k }	+	-	+	/	/
143	*fs* { n }	-	-	/	-	+
144	***fs* { i, k } ⊗ *fs* { n }**	!	-	+	-	+
145	*fs* { i, k }	+	-	+	/	/
146	*fs* { i, j }	+	+	/	-	-
147	***fs* { i, k } ⊗ *fs* { i, j }**	+	!	+	-	-
148	*fs* { i, k }	+	-	+	/	/
149	*fs* { i, m }	+	-	/	+	-
150	***fs* { i, k } ⊗ *fs* { i, m } = *fs* { i, k, m }**	+	-	+	+	-

Table 76 (continued)

row number		**i**	**j**	**k**	**m**	**n**
151	*fs* { i, k }	+	-	+	/	/
152	*fs* { i, n }	+	-	/	-	+
153	***fs* { i, k } ⊗ *fs* { i, n } = *fs* { i, k, n }**	+	-	+	-	+
154	*fs* { i, k }	+	-	+	/	/
155	*fs* { j, m }	-	+	/	+	-
156	***fs* { i, k } ⊗ *fs* { j, m }**	!	!	+	+	-
157	*fs* { i, k }	+	-	+	/	/
158	*fs* { j, n }	-	+	/	-	+
159	***fs* { i, k } ⊗ *fs* { j, n }**	!	!	+	-	+
160	*fs* { i, k }	+	-	+	/	/
161	*fs* { m, n }	-	-	/	+	+
162	***fs* { i, k } ⊗ *fs* { m, n }**	!	-	+	+	+
163	*fs* { i, k }	+	-	+	/	/
164	*fs* { i, j, m }	+	+	/	+	-
165	***fs* { i, k } ⊗ *fs* { i, j, m }**	+	!	+	+	-
166	*fs* { i, k }	+	-	+	/	/
167	*fs* { i, j, n }	+	+	/	-	+
168	***fs* { i, k } ⊗ *fs* { i, j, n }**	+	!	+	-	+
169	*fs* { i, k }	+	-	+	/	/
170	*fs* { i, m, n }	+	-	/	+	+
171	***fs* { i, k } ⊗ *fs* { i, m, n } = *fs* { i, k, m, n }**	+	-	+	+	+
172	*fs* { i, k }	+	-	+	/	/
173	*fs* { j, m, n }	-	+	/	+	+
174	***fs* { i, k } ⊗ *fs* { j, m, n }**	!	!	+	+	+
175	*fs* { i, k }	+	-	+	/	/
176	*fs* { i, j, m, n }	+	+	/	+	+
177	***fs* { i, k } ⊗ *fs* { i, j, m, n }**	+	!	+	+	+
178	*fs* { j, k }	-	+	+	/	/
179	*fs* { i }	+	-	/	-	-
180	***fs* { j, k } ⊗ *fs* { i }**	!	!	+	-	-
181	*fs* { j, k }	-	+	+	/	/
182	*fs* { m }	-	-	/	+	-
183	***fs* { j, k } ⊗ *fs* { m }**	-	!	+	+	-
184	*fs* { j, k }	-	+	+	/	/
185	*fs* { n }	-	-	/	-	+
186	***fs* { j, k } ⊗ *fs* { n }**	-	!	+	-	+
187	*fs* { j, k }	-	+	+	/	/
188	*fs* { i, j }	+	+	/	-	-
189	***fs* { j, k } ⊗ *fs* { i, j }**	!	+	+	-	-
190	*fs* { j, k }	-	+	+	/	/
191	*fs* { i, m }	+	-	/	+	-
192	***fs* { j, k } ⊗ *fs* { i, m }**	!	!	+	+	-

Table 76 (continued)

row number		**i**	**j**	**k**	**m**	**n**
193	*fs* { j, k }	-	+	+	/	/
194	*fs* { i, n }	+	-	/	-	+
195	***fs* { j, k } ⊗ *fs* { i, n }**	!	!	+	-	+
196	*fs* { j, k }	-	+	+	/	/
197	*fs* { j, m }	-	+	/	+	-
198	***fs* { j, k } ⊗ *fs* { j, m } = *fs* { j, k, m }**	-	+	+	+	-
199	*fs* { j, k }	-	+	+	/	/
200	*fs* { j, n }	-	+	/	-	+
201	***fs* { j, k } ⊗ *fs* { j, n } = *fs* { j, k, n }**	-	+	+	-	+
202	*fs* { j, k }	-	+	+	/	/
203	*fs* { m, n }	-	-	/	+	+
204	***fs* { j, k } ⊗ *fs* { m, n }**	-	!	+	+	+
205	*fs* { j, k }	-	+	+	/	/
206	*fs* { i, j, m }	+	+	/	+	-
207	***fs* { j, k } ⊗ *fs* { i, j, m }**	!	+	+	+	-
208	*fs* { j, k }	-	+	+	/	/
209	*fs* { i, j, n }	+	+	/	-	+
210	***fs* { j, k } ⊗ *fs* { i, j, n }**	!	+	+	-	+
211	*fs* { j, k }	-	+	+	/	/
212	*fs* { i, m, n }	+	-	/	+	+
213	***fs* { j, k } ⊗ *fs* { i, m, n }**	!	!	+	+	+
214	*fs* { j, k }	-	+	+	/	/
215	*fs* { j, m, n }	-	+	/	+	+
216	***fs* { j, k } ⊗ *fs* { j, m, n } = *fs* { j, k, m, n }**	-	+	+	+	+
217	*fs* { j, k }	-	+	+	/	/
218	*fs* { i, j, m, n }	+	+	/	+	+
219	***fs* { j, k } ⊗ *fs* { i, j, m, n }**	!	+	+	+	+
220	*fs* { i, j, k }	+	+	+	/	/
221	*fs* { i }	+	-	/	-	-
222	***fs* { i, j, k } ⊗ *fs* { i }**	+	!	+	-	-
223	*fs* { i, j, k }	+	+	+	/	/
224	*fs* { j }	-	+	/	-	-
225	***fs* { i, j, k } ⊗ *fs* { j }**	!	+	+	-	-
226	*fs* { i, j, k }	+	+	+	/	/
227	*fs* { m }	-	-	/	+	-
228	***fs* { i, j, k } ⊗ *fs* { m }**	!	!	+	+	-
229	*fs* { i, j, k }	+	+	+	/	/
230	*fs* { n }	-	-	/	-	+
231	***fs* { i, j, k } ⊗ *fs* { n }**	!	!	+	-	+
232	*fs* { i, j, k }	+	+	+	/	/
233	*fs* { i, m }	+	-	/	+	-

Table 76 (continued)

row number		**i**	**j**	**k**	**m**	**n**
234	***fs* { i, j, k } ⊗ *fs* { i, m }**	+	!	+	+	-
235	*fs* { i, j, k }	+	+	+	/	/
236	*fs* { i, n }	+	-	/	-	+
237	***fs* { i, j, k } ⊗ *fs* { i, n }**	+	!	+	-	+
238	*fs* { i, j, k }	+	+	+	/	/
239	*fs* { j, m }	-	+	/	+	-
240	***fs* { i, j, k } ⊗ *fs* { j, m }**	!	+	+	+	-
241	*fs* { i, j, k }	+	+	+	/	/
242	*fs* { j, n }	-	+	/	-	+
243	***fs* { i, j, k } ⊗ *fs* { j, n }**	!	+	+	-	+
244	*fs* { i, j, k }	+	+	+	/	/
245	*fs* { m, n }	-	-	/	+	+
246	***fs* { i, j, k } ⊗ *fs* { m, n }**	!	!	+	+	+
247	*fs* { i, j, k }	+	+	+	/	/
248	*fs* { i, j, m }	+	+	/	+	-
249	***fs* { i, j, k } ⊗ *fs* { i, j, m } = *fs* { i, j, k, m }**	+	+	+	+	-
250	*fs* { i, j, k }	+	+	+	/	/
251	*fs* { i, j, n }	+	+	/	-	+
252	***fs* { i, j, k } ⊗ *fs* { i, j, n } = *fs* { i, j, k, n }**	+	+	+	-	+
253	*fs* { i, j, k }	+	+	+	/	/
254	*fs* { i, m, n }	+	-	/	+	+
255	***fs* { i, j, k } ⊗ *fs* { i, m, n }**	+	!	+	+	+
256	*fs* { i, j, k }	+	+	+	/	/
257	*fs* { j, m, n }	-	+	/	+	+
258	***fs* { i, j, k } ⊗ *fs* { j, m, n }**	!	+	+	+	+
259	*fs* { i, j, k }	+	+	+	/	/
260	*fs* { i, j, m, n }	+	+	/	+	+
261	***fs* { i, j, k } ⊗ *fs* { i, j, m, n } = *fs* { i, j, k, m, n }**	+	+	+	+	+

On the 72nd row of Table 76, result of *fs* { k } ⊗ *fs* { m } operation is equal to *fs* { k, m } which is not equal to *fs* { k } or *fs* { m }. Similar situations are presented on the 75th, 93rd, 150th, 153rd, 171st, 198th, 201st, 216th, 249th, 252nd, and 261st rows. This means that on these rows, *fs* X's and *fs* Y's are not preserved.

Table 77 shows the *fs* Y's that are preserved.

Table 77 - *fs* X ⊗ *fs* Y = *fs* Y

row number		**i**	**j**	**k**	**m**	**n**
1	*fs* { i }	+	-	-	/	/
2	*fs* { i, m }	+	-	/	+	-
3	***fs* { i } ⊗ *fs* { i, m } = *fs* { i, m }**	+	-	-	+	-
4	*fs* { i }	+	-	-	/	/
5	*fs* { i, n }	+	-	/	-	+
6	***fs* { i } ⊗ *fs* { i, n } = *fs* { i, n }**	+	-	-	-	+
7	*fs* { i }	+	-	-	/	/
8	*fs* { i, m, n }	+	-	/	+	+
9	***fs* { i } ⊗ *fs* { i, m, n } = *fs* { i, m, n }**	+	-	-	+	+
10	*fs* { j }	-	+	-	/	/
11	*fs* { j, m }	-	+	/	+	-
12	***fs* { j } ⊗ *fs* { j, m } = *fs* { j, m }**	-	+	-	+	-
13	*fs* { j }	-	+	-	/	/
14	*fs* { j, n }	-	+	/	-	+
15	***fs* { j } ⊗ *fs* { j, n } = *fs* { j, n }**	-	+	-	-	+
16	*fs* { j }	-	+	-	/	/
17	*fs* { j, m, n }	-	+	/	+	+
18	***fs* { j } ⊗ *fs* { j, m, n } = *fs* { j, m, n }**	-	+	-	+	+
19	*fs* { i, j }	+	+	-	/	/
20	*fs* { i, j, m }	+	+	/	+	-
21	***fs* { i, j } ⊗ *fs* { i, j, m } = *fs* { i, j, m }**	+	+	-	+	-
22	*fs* { i, j }	+	+	-	/	/
23	*fs* { i, j, n }	+	+	/	-	+
24	***fs* { i, j } ⊗ *fs* { i, j, n } = *fs* { i, j, n }**	+	+	-	-	+
25	*fs* { i, j }	+	+	-	/	/
26	*fs* { i, j, m, n }	+	+	/	+	+
27	***fs* { i, j } ⊗ *fs* { i, j, m, n } = *fs* { i, j, m, n }**	+	+	-	+	+

Table 78 shows the *fs* X's that are preserved.

Table 78 - *fs* X ⊗ *fs* Y = *fs* X

row number		**i**	**j**	**k**	**m**	**n**
1	*fs* { i, k }	+	-	+	/	/
2	*fs* { i }	+	-	/	-	-
3	***fs* { i, k } ⊗ *fs* { i } = *fs* { i, k }**	+	-	+	-	-
4	*fs* { j, k }	-	+	+	/	/
5	*fs* { j }	-	+	/	-	-
6	***fs* { j, k } ⊗ *fs* { j } = *fs* { j, k }**	-	+	+	-	-
7	*fs* { i, j, k }	+	+	+	/	/
8	*fs* { i, j }	+	+	/	-	-
9	***fs* { i, j, k } ⊗ *fs* { i, j } = *fs* { i, j, k }**	+	+	+	-	-

From the results of *conform* operations, $childSets_R$ is constructed as follows: $childSets_R$ = { { i }, { j }, { i, j }, { i, k }, { i, m }, { i, n }, { j, k }, { j, m }, { j, n }, { i, m, n }, { j, m, n }, { i, j, k }, { i, j, m }, { i, j, n }, { i, j, m, n } }.

When the FM R is built, the model given in Figure 71 is obtained.

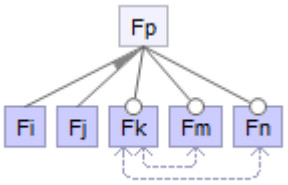

Figure 71 – Resolution of Figure 69 and Figure 70

From the $childSets_R$'s that are obtained at the end of the solutions, resolution FMs can be constructed by following the rules given in chapter 4.

3.5 Case 4: Non-Resolvable Views

For cases in rule number 1(I), 4(I), 5(I) and 5(X), proposing an automatic default solution seems not possible. To solve the complexity of conflict, human intervention is required.

They suggest a solution for case in rule number 5(X) in [4], but viewpoints and rule definition are not compatible to each other. Their solution is invalid when rule definition is accepted as correct. When rules are revised on section 3.2, case in rule number 5(X) is determined as "difficult-to-resolve".

Regarding Rule 1(I):

Figure 72 and Figure 73 represent the local views A and B, respectively. These are the views from the rule number 1(I).

Figure 72 – Local View A from rule 1(I)

Figure 73 – Local View B from rule 1(I)

First, $childSets_A$ and $childSets_B$ are determined.

$childSets_A$ = { { } }.

$childSets_B$ = { { i } }.

Second, *fs* X and *fs* Y representations are constructed as Table 79 and Table 80.

Table 79 - Feature Selection Map *fs* X

row number		**i**
1	***fs* { }**	-

Table 80 - Feature Selection Map *fs* Y

row number		**i**
1	***fs* { i }**	+

Next, combination for each *fs* X and *fs* Y is constructed. Table 81 shows the conflict situations that cannot be resolved.

Table 81 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**
1	*fs* { }	-
2	*fs* { i }	+
3	***fs* { } ⊗ *fs* { i }**	!

The $conform_B$ operation picks the *fs* X's that are preserved and the $conform_A$ operation picks the *fs* Y's that are preserved in the resolution set. There is no such *fs* X's and *fs* Y's. So, solving the conflict with *merging by conformance* is not possible.

Regarding Rule 4(I):

Figure 74 and Figure 75 represent the local views A and B, respectively, from the rule number 4(I).

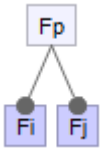

Figure 74 – Local View A from rule 4(I)

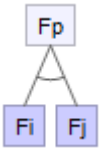

Figure 75 – Local View B from rule 4(I)

First, $childSets_A$ and $childSets_B$ are determined.

$childSets_A$ = { { i, j } }.

$childSets_B$ = { { i }, { j } }.

Second, *fs* X and *fs* Y representations are constructed as Table 82 and Table 83.

Table 82 - Feature Selection Map *fs* X

row number		**i**	**j**
1	***fs* { i, j }**	+	+

Table 83 - Feature Selection Map *fs* Y

row number		**i**	**j**
1	***fs* { i }**	+	-
2	***fs* { j }**	-	+

Next, combination for each *fs* X and *fs* Y is constructed. Table 84 shows the conflict situations that cannot be resolved.

Table 84 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**
1	*fs* { i, j }	+	+
2	*fs* { i }	+	-
3	***fs* { i, j } ⊗ *fs* { i }**	+	!
4	*fs* { i, j }	+	+
5	*fs* { j }	-	+
6	***fs* { i, j } ⊗ *fs* { j }**	!	+

There is no *fs* X's and *fs* Y's that are preserved in the resolution set. So, solving the conflict with *merging by conformance* is not possible.

Regarding Rule 5(I):

Figure 76 and Figure 77 represent the local views A and B, respectively, from the rule number 5(I).

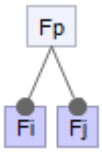

Figure 76 – Local View A from rule 5(I)

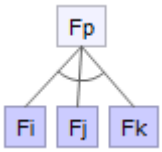

Figure 77 – Local View B from rule 5(I)

First, $childSets_A$ and $childSets_B$ are determined.

$childSets_A$ = { { i, j } }.

$childSets_B$ = { { i }, { j }, { k } }.

Second, *fs* X and *fs* Y representations are constructed as Table 85 and Table 86.

Table 85 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**
1	***fs* { i, j }**	+	+	/

Table 86 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**
1	***fs* { i }**	+	-	-
2	***fs* { j }**	-	+	-
3	***fs* { k }**	-	-	+

Next, combination for each *fs* X and *fs* Y is constructed. Table 87 - shows the conflict situations that cannot be resolved.

Table 87 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**	**k**
1	*fs* { i, j }	+	+	/
2	*fs* { i }	+	-	-
3	***fs* { i, j } ⊗ *fs* { i }**	+	!	-
4	*fs* { i, j }	+	+	/
5	*fs* { j }	-	+	-
6	***fs* { i, j } ⊗ *fs* { j }**	!	+	-
7	*fs* { i, j }	+	+	/
8	*fs* { k }	-	-	+
9	***fs* { i, j } ⊗ *fs* { k }**	!	!	+

There is no *fs* X's and *fs* Y's that are preserved in the resolution set. So, solving the conflict with *merging by conformance* is not possible.

Regarding Rule 5(X):

Figure 78 and Figure 79 represent the local views A and B, respectively, from the rule number 5(X).

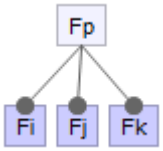

Figure 78 – Local View A from rule 5(X)

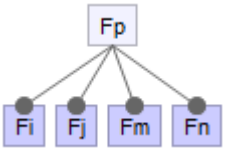

Figure 79 – Local View B from rule 5(X)

First, $childSets_A$ and $childSets_B$ are determined.

$childSets_A$ = { { i, j, k } }.

$childSets_B$ = { { i, j, m, n } }.

Second, *fs* X and *fs* Y representations are constructed as Table 88 and Table 89 - .

Table 88 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**	**m**	**n**
1	***fs* { i, j, k }**	+	+	+	/	/

Table 89 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**	**m**	**n**
1	***fs* { i, j, m, n }**	+	+	/	+	+

Next, combination for each *fs* X and *fs* Y is constructed. Table 90 presents the results of combinations where *fs* X's and *fs* Y's are not preserved.

Table 90 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**	**k**	**m**	**n**
1	*fs* { i, j, k }	+	+	+	/	/
2	*fs* { i, j, m, n }	+	+	/	+	+
3	***fs* { i, j, k } ⊗ *fs* { i, j, m, n } = { i, j, k, m, n }**	+	+	+	+	+

There is no *fs* X's and *fs* Y's that are preserved in the resolution set. So, solving the conflict with *merging by conformance* is not possible.

3.6 Case 5: Cross-tree Constraints Between Siblings

Merging procedure also constructs resolutions for conflicts within views involving cross-tree constraints between child features. Examples can be found in Appendix A.

CHAPTER 4

CONSTRUCTING RESOLUTION FM

In this chapter, a procedure to construct the (partial) resolution model from its child sets, resulting from the merger of two local views is presented.

4.1 Rules for Constructing Resolution FM

First, a table with columns for the child features of the parent *p*, and rows for $childSets_R$ is constructed. Mark each entry of the table with a '1' if the feature is in the childset, with a '0' otherwise.

Second, the following rules are applied in the order given.

i) A column with all 1's belongs to a mandatory feature.

ii) A column with all 0's belongs to a not-available feature.

iii) If a column, say, for feature i, has both 1 and 0 with the same set of combinations of the other rows, i is an optional feature. (If i is required by another feature in local view A or B, say j requires i, rows including both i and j should be disregarded when deciding the optionality of i.)

iv) If two or more features are never all 1's on the same row, but exactly one of them is 1 on each row, they must be in an alternative relation.

v) If two or more features (neither mandatory nor optional) are all 1's on the same row they must be in an or relation.

vi) If a non-mandatory feature, say j, has 1's on all the rows where another feature, say i, is 1, they must be in a requires relation, that is, i requires j.

vii) If two features, say i and j, that are not both mandatory, and not in alternative relation, are never both 1's on the same row, they must be in an excludes relation.

Note that the “requires” and “excludes” relations between siblings are introduced only when necessary.

4.2 Constructing Resolution FMs for Mannion *et al.* Rules

In the following examples, let use some of the child sets of R’s given in section “Case 2: Reproducing Mannion *et al.* Rules”.

Regarding Rule 1(II):

$childSets_R$ is constructed above for the rule number 1(II) as follows:

$childSets_R$ = { { } }.

Steps of constructing a table are followed and the Table 91 given below is obtained.

Table 91 - Marking for $childSets_R$ for rule 1(II)

	i
{ }	0

If the given rules are applied, i is determined as not-available as indicated in the second rule and as shown in Figure 14.

Regarding Rule 2:

Other $childSets_R$ is constructed above for the rule number 2 as follows:

$childSets_R$ = { { i } }.

Steps of constructing a table are followed and the Table 92 given below is obtained.

Table 92 - Marking for $childSets_R$ for rule 2

	i
{ i }	1

If the given rules are applied, i is determined as mandatory as indicated in the first rule and as shown in Figure 17.

Regarding Rule 3(II):

Another $childSets_R$ is constructed above for the rule number 3(II) as follows:

$childSets_R$ = { { i }, {i, j}, { i, k }, { i, j, k} }.

Steps of constructing a table are followed and the Table 93 given below is obtained.

Table 93 - Marking for $childSets_R$ for rule 3(II)

	i	**j**	**k**
{ i }	1	0	0
{ i, j }	1	1	0
{ i, k }	1	0	1
{ i, j, k }	1	1	1

If the given rules above are applied, i is determined as mandatory as indicated in the first rule, j and k are determined as optional as indicated in the third rule and they are as shown in Figure 23.

Regarding Rule 3(IV):

Another $childSets_R$ is constructed above for the rule number 3(IV) as follows:

$childSets_R$ = { { i }, { j }, { k } }.

Steps of constructing a table are followed and the Table 94 given below is obtained.

Table 94 - Marking for $childSets_R$ for rule 3(IV)

	i	**j**	**k**
{ i }	1	0	0
{ j }	0	1	0
{ k }	0	0	1

If the given rules above are applied; i, j and k are determined as in an alternative relationship as indicated in the fourth rule and as shown in Figure 29.

Regarding Rule 3(V):

Another $childSets_R$ is constructed above for the rule number 3(V) as follows:

$childSets_R$ = { { i }, { j }, { k }, { j, k }, { i, j }, { i, k }, { i, j, k } }.

Steps of constructing a table are followed and the Table 95 given below is obtained.

Table 95 - Marking for $childSets_R$ for rule 3(V)

	i	**j**	**k**
{ i }	1	0	0
{ j }	0	1	0
{ k }	0	0	1
{ j, k }	0	1	1
{ i, j }	1	1	0
{ i, k }	1	0	1
{ i, j, k }	1	1	1

If the given rules above are applied, i, j and k are determined as in an or relationship as indicated in the fifth rule and as shown in Figure 29.

Regarding Rule 5(III):

Another $childSets_R$ is constructed above for the rule number 5(III) as follows:

$childSets_R$ = { { i, j }, { i, j, k} }.

Steps of constructing a table are followed and the Table 96 given below is obtained.

Table 96 - Marking for $childSets_R$ for rule 5(III)

	i	**j**	**k**
{ i, j }	1	1	0
{ i, j, k }	1	1	1

If the given rules above are applied, i and j are determined as mandatory as indicated in the first rule, k is determined as optional as indicated in the third rule and they are as shown in Figure 53.

Regarding Rule 5(IV):

Another $childSets_R$ is constructed above for the rule number 5(IV) as follows:

$childSets_R$ = { { i }, { j }, { i, k }, { j, k } }.

Steps of constructing a table are followed and the Table 97 given below is obtained.

Table 97 - Marking for $childSets_R$ for rule 5(IV)

	i	j	k
{ i }	1	0	0
{ j }	0	1	0
{ i, k }	1	0	1
{ j, k }	0	1	1

If the given rules above are applied, i and j are determined as in an alternative relationship as indicated in the fourth rule, k is determined as optional as indicated in the third rule and they are as shown in Figure 56.

Regarding Rule 5(VI):

Another $childSets_R$ is constructed above for the rule number 5(VI) as follows:

$childSets_R$ = { { i }, { j }, { i, j }, { i, k }, { j, k }, { i, j, k } }.

Steps of constructing a table are followed and the Table 98 given below is obtained.

Table 98 - Marking for $childSets_R$ for rule 5(VI)

	i	j	k
{ i }	1	0	0
{ j }	0	1	0
{ i, j }	1	1	0
{ i, k }	1	0	1
{ j, k }	0	1	1
{ i, j, k }	1	1	1

If the given rules above are applied, i and j are determined as in an or relationship as indicated in the fifth rule, k is determined as optional as indicated in the third rule and they are as shown in Figure 62.

Regarding Rule 5(VII):

Another $childSets_R$ is constructed above for the rule number 5(VII) as follows:

$childSets_R$ = { { i } }.

Steps of constructing a table are followed and the Table 99 given below is obtained.

Table 99 - Marking for $childSets_R$ for rule 5(VII)

	i	j	k	m	n
{ i }	1	0	0	0	0

If the given rules above are applied, i is determined as mandatory as indicated in the first rule and j, k, m and n are determined as not-available as indicated in the second rule and they are as shown in Figure 65.

4.3 For Difficult-to-Resolve Views

In the following examples, let use some of the child sets of R's given in section "Case 3: Difficult-to-Resolve Views".

Regarding Rule 5(VIII):

$childSets_R$ is constructed above for the rule number 5(VIII) as follows:

$childSets_R$ = { { i }, { j } }.

Steps of constructing a table are followed and the Table 100 given below is obtained.

Table 100 - Marking for $childSets_R$ for rule 5(VIII)

	i	**j**	**k**	**m**	**n**
{ i }	1	0	0	0	0
{ j }	0	1	0	0	0

If the given rules above are applied, i and j are determined as in an alternative relationship as indicated in the fourth rule and k, m and n are determined as not-available as indicated in the second rule and they are as shown in Figure 68.

Regarding Rule 5(IX):

Other $childSets_R$ is constructed above for the rule number 5(IX) as follows:

$childSets_R$ = { { i }, { j }, { i, j }, { i, k }, { i, m }, { i, n }, { j, k }, { j, m }, { j, n }, { i, m, n }, { j, m, n }, { i, j, k }, { i, j, m }, { i, j, n }, { i, j, m, n } }.

Steps of constructing a table are followed and the Table 101 given below is obtained.

Table 101 - Marking for $childSets_R$ for rule 5(IX)

	i	j	k	m	n
{ i }	1	0	0	0	0
{ j }	0	1	0	0	0
{ i, j }	1	1	0	0	0
{ i, k }	1	0	1	0	0
{ i, m }	1	0	0	1	0
{ i, n }	1	0	0	0	1
{ j, k }	0	1	1	0	0
{ j, m }	0	1	0	1	0
{ j, n }	0	1	0	0	1
{ i, m, n }	1	0	0	1	1
{ j, m, n }	0	1	0	1	1
{ i, j, k }	1	1	1	0	0
{ i, j, m }	1	1	0	1	0
{ i, j, n }	1	1	0	0	1
{ i, j, m, n }	1	1	0	1	1

If the given rules above are applied, i and j are determined as in an or relationship as indicated in the fifth rule and k, m and n are determined as optional as indicated in the third rule, k and m are determined as in an excludes relationship and similarly k and n are determined as in an excludes relationship as indicated in the seventh rule and they are as shown in Figure 71.

4.4 For Views Including Cross-tree Constraints Between Siblings

The procedure also constructs resolutions for conflicts within views involving cross-tree constraints between sibling features. Examples can be found in Appendix B.

CHAPTER 5

MERGING COMPLETE VIEWS

An algorithm is defined as shown in Figure 80 to merge complete parent-compatible FMs based on *merging by conformance*. A top-down procedure is described that applies the local view merge procedure at each parent. Examples can be found in Appendix C.

```
1mergeByConformance (viewA: FeatureModel, viewB: FeatureModel)
2/* output: R: FeatureModel */
3begin
4   level = 0
5   while ( ( level ≤ viewA.depth ) and ( level ≤ viewB.depth ) ) do
6       featuresA = getFeatures( viewA, level )
7       featuresB = getFeatures( viewB, level )
8       featuresAtLevel = featuresA ∪ featuresB
9       for each p ∈ featuresAtLevel do
10          parentA = p
11          parentB = p
12          conformA = Ø
13          conformB = Ø
14/* obtain children at only one level below */
15          childSetsA = getChilds( viewA, parentA )
16          childSetsB = getChilds( viewB, parentB )
17          if ( childSetsA != Ø and childSetsB = Ø ) then
18/* carry child sets branch of parentA to view R */
19              moveToViewR( viewA, parentA, childSetsA )
20          end if
21          if ( childSetsA = Ø and childSetsB != Ø ) then
22/* carry child sets branch of parentB to view R */
23              moveToViewR( viewB, parentB, childSetsB )
24          end if
25          if ( childSetsA != Ø and childSetsB != Ø ) then
26              fsXSet = constructFsX( childSetsA )
27              fsYSet = contructFsY( childSetsB )
28              for each fsX ∈ fsXSet do
29                  search for fsY ∈ fsYSet such that fsX ⊗ fsY = fsX
30                      if found
31/* add the child set corresponding to fs to conformB */
32                          conformB = conformB ∪ {selectedSet( fsX )}
33                      end if
34              end for
35              for each fsY ∈ fsYSet do
36                  search for fsX ∈ fsXSet such that fsX ⊗ fsY = fsY
37                      if found
38                    /* add the child set corresponding to fs to conformA */
39                          conformA = conformA ∪ {selectedSet( fsY )}
40                      end if
41              end for
42              childSetsR = conformA ∪ conformB
43              constructModelR( childSetsR, level )
44          end if
45      end for
46      level++
47  end while
48end
```

Figure 80 – Algorithm for merging complete views

Complexity of the Algorithm:

Big O notation is going to be used to describe the complexity of an algorithm given in Figure 80. Big O characterizes the algorithms according to changes in input size. The worst-case scenario is considered.

Assume *viewA.depth* > *viewB.depth*. Approximately, the complexity is *viewA.depth* * *featuresAtLevel.size* * |*fsXSet*| * |*fsYSet*|. In here, "*" symbolizes the multiplication operation.

Let D be max (*viewA.depth*, *viewB.depth*). Let C be the maximum number of children at any level (in view A or B). Let F be the total number of features (in view A or B). Thus, the complexity of the algorithm is $O(D * F * 2^C)$.

CHAPTER 6

LOGICAL CHARACTERIZATION OF MANNION *ET AL*. RULES

On section 3.2, Mannion *et al.* rules are revised and updated. In this chapter, these updated rules are characterized logically.

6.1 Approach for Logical Characterization

While those rules are expressed using propositional logic instead of literary language, just using logical and (∧) between local views may not be sufficient to obtain resolution. Expressing merging procedure as "view A ∧ view B" may not be a correct approach for some rules.

Using the tautology given in (1), a general logical expression for all rules can be obtained.

The abbreviations *A*, *B* and *R*are used for expressing Local View A, Local View B and Resolution of Local View A and Local View B logically, respectively.

$$A \wedge B \wedge X \leftrightarrow R \wedge X \quad (1)$$

X is an abbreviation of a logical expression. It is used to accomplish a tautology containing *A*, *B* and *R* for all rules. Generally, it can be expressed as follows:

$$X \leftrightarrow A \wedge B \wedge R \quad (2)$$

The given *X* definition in (2) is the most powerful one but it should be expressed as weak as possible for simplicity.

At first *X* should be simplified, then it should be expressed in Conjunctive Normal Form (CNF) and lastly it should be weakened.

When *X* is in CNF, for the weakening process all the single clauses and conjunction of clauses should be put instead of *X* in (1), to find out if the clause(s) satisfies the tautology or not. Similarly, when the clause satisfies the tautology, all the single literals and disjunction of literals in the clause should be put instead of *X* in (1) to find out if the literal(s) satisfies the tautology or not. When the literal satisfies the tautology, it should be used instead of *X*. The weakest *X* should be found in this way.

Note that, the weakest X is the “true”. For all tautologies, “true” should be put instead of *X* to find out if it satisfies the tautology or not.

There may be more than one weakest *X*'s that satisfy the tautology. In the following table below, the entire weakest *X*'s for rules are given.

As an example, *X* for the rule number 5(IX) is going to be simplified, expressed in CNF and weakened.

The following logical expression represents *A*:

$F_p \leftrightarrow (F_i \vee F_j \vee F_k)$

The following logical expression represents *B*:

$F_p \leftrightarrow (F_i \vee F_j \vee F_m \vee F_n)$

The following logical expression represents *R*:

$(F_p \leftrightarrow (F_i \vee F_j)) \wedge (F_k \rightarrow F_p) \wedge (F_m \rightarrow F_p) \wedge (F_n \rightarrow F_p) \wedge (\neg (F_k \wedge F_m)) \wedge (\neg (F_k \wedge F_n))$

Now, *X* can be defined as follows:

$X \leftrightarrow (F_p \leftrightarrow (F_i \vee F_j \vee F_k)) \wedge (F_p \leftrightarrow (F_i \vee F_j \vee F_m \vee F_n)) \wedge [(F_p \leftrightarrow (F_i \vee F_j)) \wedge (F_k \rightarrow F_p) \wedge (F_m \rightarrow F_p) \wedge (F_n \rightarrow F_p) \wedge (\neg (F_k \wedge F_m)) \wedge (\neg (F_k \wedge F_n))]$

$X \leftrightarrow (F_p \rightarrow (F_i \vee F_j \vee F_k)) \wedge ((F_i \vee F_j \vee F_k) \rightarrow F_p) \wedge (F_p \rightarrow (F_i \vee F_j \vee F_m \vee F_n)) \wedge ((F_i \vee F_j \vee F_m \vee F_n) \rightarrow F_p) \wedge (F_p \rightarrow (F_i \vee F_j)) \wedge ((F_i \vee F_j) \rightarrow F_p) \wedge (F_k \rightarrow F_p) \wedge (F_m \rightarrow F_p) \wedge (F_n \rightarrow F_p) \wedge (\neg (F_k \wedge F_m)) \wedge (\neg (F_k \wedge F_n))$

$X \leftrightarrow (\neg F_p \vee F_i \vee F_j \vee F_k) \wedge (F_i \rightarrow F_p) \wedge (F_j \rightarrow F_p) \wedge (F_k \rightarrow F_p) \wedge (\neg F_p \vee F_i \vee F_j \vee F_m \vee F_n) \wedge (F_i \rightarrow F_p) \wedge (F_j \rightarrow F_p) \wedge (F_m \rightarrow F_p) \wedge (F_n \rightarrow F_p) \wedge (\neg F_p \vee F_i \vee F_j) \wedge (F_i \rightarrow F_p) \wedge (F_j \rightarrow F_p) \wedge (F_k \rightarrow F_p) \wedge (F_m \rightarrow F_p) \wedge (F_n \rightarrow F_p) \wedge (\neg (F_k \wedge F_m)) \wedge (\neg (F_k \wedge F_n))$

Below, the expression is rewritten by simplifying same clauses.

$X \leftrightarrow (\neg F_p \vee F_i \vee F_j \vee F_k) \wedge (F_i \rightarrow F_p) \wedge (F_j \rightarrow F_p) \wedge (F_k \rightarrow F_p) \wedge (\neg F_p \vee F_i \vee F_j \vee F_m \vee F_n) \wedge (F_m \rightarrow F_p) \wedge (F_n \rightarrow F_p) \wedge (\neg F_p \vee F_i \vee F_j) \wedge (\neg (F_k \wedge F_m)) \wedge (\neg (F_k \wedge F_n))$

When the absorption rule $(a \wedge (a \vee b) = a)$ is applied to the expression, it is expressed as follows:

$X \leftrightarrow (\neg F_p \vee F_i \vee F_j) \wedge (F_i \rightarrow F_p) \wedge (F_j \rightarrow F_p) \wedge (F_k \rightarrow F_p) \wedge (F_m \rightarrow F_p) \wedge (F_n \rightarrow F_p) \wedge (\neg (F_k \wedge F_m)) \wedge (\neg (F_k \wedge F_n))$

When the expression is written in CNF, it is as follows:

$X \leftrightarrow (\neg F_p \vee F_i \vee F_j) \wedge (\neg F_i \vee F_p) \wedge (\neg F_j \vee F_p) \wedge (\neg F_k \vee F_p) \wedge (\neg F_m \vee F_p) \wedge (\neg F_n \vee F_p) \wedge (\neg F_k \vee \neg F_m) \wedge (\neg F_k \vee \neg F_n)$

After putting all the single clauses and conjunction of clauses instead of *X* in (1), the expression $(\neg F_k \vee \neg F_m) \wedge (\neg F_k \vee \neg F_n)$ is found out as the weakest *X*.

All the single literals and disjunction of literals in the expression $(\neg F_k \vee \neg F_m) \wedge (\neg F_k \vee \neg F_n)$ is put instead of *X* in (1), but neither literal nor “true” is satisfied the tautology.

In the following table below, given *X*'s are the weakest ones for Mannion *et al.* rules.

Table 102 - Weakest *X*'s for Mannion *et al.* Rules

Rule Number	View A	View B	Resolution	*X*
1(I)	$F_p \leftrightarrow \neg F_i$	$F_p \leftrightarrow F_i$	Too complex to solve automatically.	Not available.
1(II)	$F_p \leftrightarrow \neg F_i$	$F_i \rightarrow F_p$	$F_p \leftrightarrow \neg F_i$	$\neg F_i \vee F_p$
2	$F_p \leftrightarrow F_i$	$F_i \rightarrow F_p$	$F_p \leftrightarrow F_i$	True
3(I)	$F_p \leftrightarrow F_i$	$F_p \leftrightarrow (F_i \oplus F_j \oplus F_k)$	$(F_p \leftrightarrow F_i) \wedge (F_p \leftrightarrow \neg F_j) \wedge (F_p \leftrightarrow \neg F_k)$	i or p
3(II)	$F_p \leftrightarrow F_i$	$F_p \leftrightarrow (F_i \vee F_j \vee F_k)$	$(F_p \leftrightarrow F_i) \wedge (F_j \rightarrow F_p) \wedge (F_k \rightarrow F_p)$	$\neg F_k \vee F_p$
3(III)	$F_p \leftrightarrow F_i$	$(F_i \rightarrow F_p) \wedge (F_j \rightarrow F_p) \wedge (F_k \rightarrow F_p)$	$(F_p \leftrightarrow F_i) \wedge (F_j \rightarrow F_p) \wedge (F_k \rightarrow F_p)$	True
3(IV)	$F_i \rightarrow F_p$	$F_p \leftrightarrow (F_i \oplus F_j \oplus F_k)$	$F_p \leftrightarrow (F_i \oplus F_j \oplus F_k)$	True
3(V)	$F_i \rightarrow F_p$	$F_p \leftrightarrow (F_i \vee F_j \vee F_k)$	$F_p \leftrightarrow (F_i \vee F_j \vee F_k)$	True
4(I)	$(F_p \leftrightarrow F_i) \wedge (F_p \leftrightarrow F_j)$	$F_p \leftrightarrow (F_i \oplus F_j)$	Too complex to solve automatically.	Not available.
4(II)	$(F_p \leftrightarrow F_i) \wedge (F_p \leftrightarrow F_j)$	$F_p \leftrightarrow (F_i \vee F_j)$	$(F_p \leftrightarrow F_i) \wedge (F_p \leftrightarrow F_j)$	$\neg F_i \vee \neg F_j \vee \neg F_p$ or $F_i \vee F_j \vee F_p$
4(III)	$(F_p \leftrightarrow F_i) \wedge (F_p \leftrightarrow F_j)$	$(F_i \rightarrow F_p) \wedge (F_j \rightarrow F_p)$	$(F_p \leftrightarrow F_i) \wedge (F_p \leftrightarrow F_j)$	True

Table 102 (continued)

Rule Number	View A	View B	Resolution	*X*
4(IV)	$F_p \leftrightarrow (F_i \oplus F_j)$	$F_p \leftrightarrow (F_i \vee F_j)$	$F_p \leftrightarrow (F_i \oplus F_j)$	True
4(V)	$F_p \leftrightarrow (F_i \oplus F_j)$	$(F_i \rightarrow F_p) \wedge (F_j \rightarrow F_p)$	$F_p \leftrightarrow (F_i \oplus F_j)$	True
4(VI)	$F_p \leftrightarrow (F_i \vee F_j)$	$(F_i \rightarrow F_p) \wedge (F_j \rightarrow F_p)$	$F_p \leftrightarrow (F_i \vee F_j)$	True
5(I)	$(F_p \leftrightarrow F_i) \wedge (F_p \leftrightarrow F_j)$	$F_p \leftrightarrow (F_i \oplus F_j \oplus F_k)$	Too complex to solve automatically.	Not available.
5(II)	$(F_p \leftrightarrow F_i) \wedge (F_p \leftrightarrow F_j)$	$F_p \leftrightarrow (F_i \vee F_j \vee F_k)$	$(F_p \leftrightarrow F_i) \wedge (F_p \leftrightarrow F_j) \wedge (F_k \rightarrow F_p)$	True
5(III)	$(F_p \leftrightarrow F_i) \wedge (F_p \leftrightarrow F_j)$	$(F_i \rightarrow F_p) \wedge (F_j \rightarrow F_p) \wedge (F_k \rightarrow F_p)$	$(F_p \leftrightarrow F_i) \wedge (F_p \leftrightarrow F_j) \wedge (F_k \rightarrow F_p)$	True
5(IV)	$F_p \leftrightarrow (F_i \oplus F_j)$	$F_p \leftrightarrow (F_i \vee F_j \vee F_k)$	$(F_p \leftrightarrow (F_i \oplus F_j)) \wedge (F_k \rightarrow F_p)$	True
5(V)	$F_p \leftrightarrow (F_i \oplus F_j)$	$(F_i \rightarrow F_p) \wedge (F_j \rightarrow F_p) \wedge (F_k \rightarrow F_p)$	$(F_p \leftrightarrow (F_i \oplus F_j)) \wedge (F_k \rightarrow F_p)$	True
5(VI)	$F_p \leftrightarrow (F_i \vee F_j)$	$(F_i \rightarrow F_p) \wedge (F_j \rightarrow F_p) \wedge (F_k \rightarrow F_p)$	$(F_p \leftrightarrow (F_i \vee F_j)) \wedge (F_k \rightarrow F_p)$	True
5(VII)	$F_p \leftrightarrow (F_i \oplus F_j \oplus F_k)$	$F_p \leftrightarrow (F_i \oplus F_m \oplus F_n)$	$(F_p \leftrightarrow F_i) \wedge (F_p \leftrightarrow \neg F_j) \wedge (F_p \leftrightarrow \neg F_k) \wedge (F_p \leftrightarrow \neg F_m) \wedge (F_p \leftrightarrow \neg F_n)$	i

Table 102 (continued)

Rule Number	View A	View B	Resolution	*X*
5(VIII)	$F_p \leftrightarrow (F_i \oplus F_j \oplus F_k)$	$F_p \leftrightarrow (F_i \oplus F_j \oplus F_m \oplus F_n)$	$(F_p \leftrightarrow (F_i \oplus F_j)) \wedge (F_p \leftrightarrow \neg F_k) \wedge (F_p \leftrightarrow \neg F_m) \wedge (F_p \leftrightarrow \neg F_n)$	i or j
5(IX)	$F_p \leftrightarrow (F_i \vee F_j \vee F_k)$	$F_p \leftrightarrow (F_i \vee F_j \vee F_m \vee F_n)$	$(F_p \leftrightarrow (F_i \vee F_j)) \wedge (F_k \rightarrow F_p) \wedge (F_m \rightarrow F_p) \wedge (F_n \rightarrow F_p) \wedge (\neg (F_k \wedge F_m)) \wedge (\neg (F_k \wedge F_n))$	$(\neg F_k \vee \neg F_m) \wedge (\neg F_k \vee \neg F_n)$
5(X)	$(F_p \leftrightarrow F_i) \wedge (F_p \leftrightarrow F_j) \wedge (F_p \leftrightarrow F_k)$	$(F_p \leftrightarrow F_i) \wedge (F_p \leftrightarrow F_j) \wedge (F_p \leftrightarrow F_m) \wedge (F_p \leftrightarrow F_n)$	Too complex to solve automatically.	Not available.

CHAPTER 7

CONCLUSION

In this thesis, a normative procedure has been introduced for merging FMs reflecting different viewpoints. Using this procedure, one can merge FMs with and without cross-tree relationships between sibling features. The procedure may terminate by signaling unresolved conflicts as well.

Rules have been presented with resolutions by describing the merging procedure step by step. With examples, in particular, by reproducing Mannion *et al.* default resolution rules [4], an approach has been verified. An obvious improvement of this approach is that the merging of conflicting FMs is defined in a general way, rather than being illustrated case by case as in [4].

The method is flexible in that domain-specific default resolution schemes can be devised by adopting different lattices (or tables, in general) for combination than the one is used in the present work.

The main limitation of the present work is that the merging procedure relies on local views (one level of decomposition) at every step. Taking the whole FMs into consideration would allow a much wider range of resolution strategies, and yield more flexible, albeit more complicated, merging procedures. Future work will aim to relax this constraint of locality.

REFERENCES

[1] K. Paul, G. Böckle, and F. van der Linden, “Software product line engineering,” ISBN-13 978-3-540-24372-4, pp. 100, 2005.

[2] K. Lee, Kyo C. Kang, and J. Lee, “Concepts and guidelines of feature modeling for product line software engineering,” Proc. 7th ICSR, 2002, pp. 62—77.

[3] D. Benavides, S. Segura, and A. Ruiz-Cortes, “Automated analysis of feature models 20 years later: a literature review,” doi:10.1016/j.is2010.01.001, Information Systems. Elsevier. 2010.

[4] M. Mannion, J. Savolainen, and T. Asikainen, “Viewpoint-oriented variability modeling,” COMPSAC, 2009.

[5] B. Nuseibeh, J. Kramer, and A. Finkelstein, “Viewpoints meaningful relationships are difficult,” ICSE, 2003.

[6] E. A. Aydın, H. Oğuztüzün, A. H. Doğru and A. S. Karataş, “Merging multi-view feature models by local rules,” 9th International Conference on Software Engineering Research, Management and Applications, 10-12 Aug. 2011, pp. 140-147.

[7] K. C. Kang, J. Lee, and P. Donohoe, “Feature-oriented product line engineering,” IEEE Software, July/August 2002, pp. 58 -65.

[8] M. A. Babar, L. Chen, and F. Shull, “Managing variability in software product lines,” IEEE Software, May/June 2010, pp. 89 -94.

[9] M. Acher, P. Collet, P. Lahire, and R. France, “Composing feature models,” Software Language Engineering, 2010

[10] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, “Feature-oriented domain analysis (FODA) feasibility study,” Software Engineering Institute, Carnegie Mellon University (Report), November 1990.

[11] M. Mannion and J. Camara, “Theorem proving for product line model verification,” 5th International Workshop on Software Product-Family Engineering, Springer, 2003, pp. 211-224.

[12] S. Segura, D. Benavides, A. Ruiz-Cortés, and P. Trinidad, “Automated merging of feature models using graph transformations,” GTTSE, 2007.

[13] P. van den Broek, and J. Noppen, “Merging feature models,” 14th International Software Product Line Conference, 14 September 2010.

[14] N. Niu, J. Savolainen, and Y. Yu, “Variability modeling for product line viewpoints integration,” 34th Annual IEEE Computer Software and Applications Conference, 19-23 July 2010.

[15] P. Y. Schobbens, P. Heymans, J. C. Trigaux, and Y. Bontemps, “Generic semantics of feature diagrams,” Computer Networks (2006), doi:10.1016/j.comnet.2006.08.008, special issue on feature interactions in emerging application domains, page 38, 2006.

[16] D. Clarke, and J. Proença, “Towards a theory of views for feature models,” FMSPLE, 2010.

APPENDIX A

EXAMPLES OF MERGING BY CONFORMANCE WITH CROSS-TREE CONSTRAINTS BETWEEN SIBLINGS

An example includes a view with constraint. As shown in Figure 82, local view B consists of requires relationship.

Figure 81 – Local View A

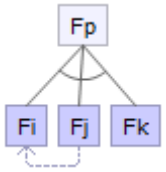

Figure 82 – Local View B

First step in *merging by conformance* is determining the sets of sets of sibling features of views.

$childSets_A$ = { { i } }.

$childSets_B$ = { { i }, { k } }.

Second step is constructing *fs* X and *fs* Y representations.

Table 103 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**
1	***fs* { i }**	+	/	/

Table 104 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**
1	***fs* { i }**	+	-	-
2	***fs* { k }**	-	-	+

Third step is performing combination for each *fs* X and *fs* Y. Table 105 shows the conflict situation that cannot be resolved.

Table 105 - The case where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**	**k**
1	*fs* { i }	+	/	/
2	*fs* { k }	-	-	+
3	***fs* { i } ⊗ *fs* { k }**	!	-	+

Executing $conform_A$ and $conform_B$ operations, $childSets_R$ is obtained as follows:

$childSets_R$ = { { i } }.

When the FM R is built, the model given in Figure 83 is obtained.

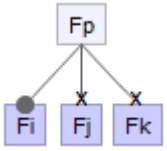

Figure 83 – Resolution of Figure 81 and Figure 82

Following example includes a view with constraint. As shown in Figure 85, local view B consists of requires relationship.

Figure 84 – Local View A

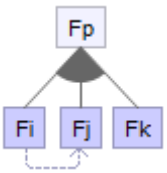

Figure 85 – Local View B

First step in *merging by conformance* is determining the sets of sets of sibling features of views.

$childSets_A$ = { { i } }.

$childSets_B$ = { { j }, { k }, { i, j }, { j, k }, { i, j, k } }.

Second step is constructing *fs* X and *fs* Y representations.

Table 106 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**
1	***fs* { i }**	+	/	/

Table 107 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**
1	***fs* { j }**	-	+	-
2	***fs* { k }**	-	-	+
3	***fs* { i, j }**	+	+	-
4	***fs* { j, k }**	-	+	+
5	***fs* { i, j, k }**	+	+	+

Third step is performing combination for each *fs* X and *fs* Y. Table 108 shows the conflict situations that cannot be resolved.

Table 108 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**	**k**
1	*fs* { i }	+	/	/
2	*fs* { j }	-	+	-
3	***fs* { i } ⊗ *fs* { j }**	!	+	-
4	*fs* { i }	+	/	/
5	*fs* { k }	-	-	+
6	***fs* { i } ⊗ *fs* { k }**	!	-	+
7	*fs* { i }	+	/	/
8	*fs* { j, k }	-	+	+
9	***fs* { i } ⊗ *fs* { j, k }**	!	+	+

Table 109 gives the result of combinations where *fs* Y's are preserved.

Table 109 - *fs* X ⊗ *fs* Y = *fs* Y

row number		**i**	**j**	**k**
1	*fs* { i }	+	/	/
2	*fs* { i, j }	+	+	-
3	***fs* { i } ⊗ *fs* { i, j } = *fs* { i, j }**	+	+	-
4	*fs* { i }	+	/	/
5	*fs* { i, j, k }	+	+	+
6	***fs* { i } ⊗ *fs* { i, j, k } = *fs* { i, j, k }**	+	+	+

Executing $conform_A$ and $conform_B$ operations, $childSets_R$ is obtained as follows:

$childSets_R$ = { { i, j }, { i, j, k } }.

When the FM R by is built, the model given in Figure 86 is obtained.

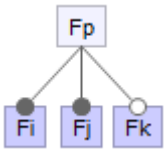

Figure 86 – Resolution of Figure 84 and Figure 85

Following example includes a view with constraints. As shown in Figure 88, local view B consists of requires and excludes relationships.

Figure 87 – Local View A

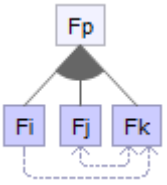

Figure 88 – Local View B

First step in *merging by conformance* is determining the sets of sets of sibling features of views.

$childSets_A$ = { { i } }.

$childSets_B$ = { { j }, { k }, { i, k } }.

Second step is constructing *fs* X and *fs* Y representations.

Table 110 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**
1	***fs* { i }**	+	/	/

Table 111 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**
1	***fs* { j }**	-	+	-
2	***fs* { k }**	-	-	+
3	***fs* { i, k }**	+	-	+

Third step is performing combination for each *fs* X and *fs* Y. Table 112 shows the conflict situations that cannot be resolved.

Table 112 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**	**k**
1	*fs* { i }	+	/	/
2	*fs* { j }	-	+	-
3	***fs* { i } ⊗ *fs* { j }**	!	+	-
4	*fs* { i }	+	/	/
5	*fs* { k }	-	-	+
6	***fs* { i } ⊗ *fs* { k }**	!	-	+

Table 113 gives the result of *fs* { i } ⊗ *fs* { i, k } where *fs* Y is preserved.

Table 113 - *fs* X ⊗ *fs* Y = *fs* Y

row number		**i**	**j**	**k**
1	*fs* { i }	+	/	/
2	*fs* { i, k }	+	-	+
3	***fs* { i } ⊗ *fs* { i, k } = *fs* { i, k }**	+	-	+

Executing $conform_A$ and $conform_B$ operations, $childSets_R$ is obtained as follows:

$childSets_R$ = { { i, k } }.

When the FM R is built, the model given in Figure 89 is obtained.

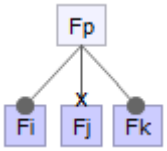

Figure 89 – Resolution of Figure 87 and Figure 88

Following example includes a view with constraint. As shown in Figure 91, local view B consists of excludes relationship.

Figure 90 – Local View A

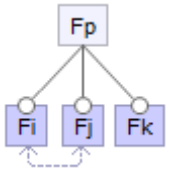

Figure 91 – Local View B

First step in *merging by conformance* is determining the sets of sets of sibling features of views.

$childSets_A$ = { { i } }.

$childSets_B$ = { { }, { i }, { j }, { k }, { i, k }, { j, k } }.

Second step is constructing *fs* X and *fs* Y representations.

Table 114 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**
1	***fs* { i }**	+	/	/

Table 115 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**
1	***fs* { }**	-	-	-
2	***fs* { i }**	+	-	-
3	***fs* { j }**	-	+	-
4	***fs* { k }**	-	-	+
5	***fs* { i, k }**	+	-	+
6	***fs* { j, k }**	-	+	+

Third step is performing combination for each *fs* X and *fs* Y. Table 116 shows the conflict situations that cannot be resolved.

Table 116 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**	**k**
1	*fs* { i }	+	/	/
2	*fs* { }	-	-	-
3	***fs* { i } ⊗ *fs* { }**	!	-	-
4	*fs* { i }	+	/	/
5	*fs* { j }	-	+	-
6	***fs* { i } ⊗ *fs* { j }**	!	+	-
7	*fs* { i }	+	/	/
8	*fs* { k }	-	-	+
9	***fs* { i } ⊗ *fs* { k }**	!	-	+
10	*fs* { i }	+	/	/
11	*fs* { j, k }	-	+	+
12	***fs* { i } ⊗ *fs* { j, k }**	!	+	+

Table 117 gives the result of *fs* { i } ⊗ *fs* { i, k } where *fs* Y is preserved.

Table 117 - *fs* X ⊗ *fs* Y = *fs* Y

row number		**i**	**j**	**k**
1	*fs* { i }	+	/	/
2	*fs* { i, k }	+	-	+
3	***fs* { i } ⊗ *fs* { i, k } = *fs* { i, k }**	+	-	+

Executing *conform*$_A$ and *conform*$_B$ operations, childSets$_R$ is obtained as follows:

childSets$_R$ = { { i }, { i, k } }.

When the FM R is built, the model given in Figure 92 is obtained.

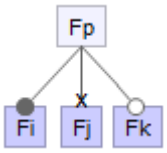

Figure 92 – Resolution of Figure 90 and Figure 91

Following example includes a view with constraints. As shown in Figure 94, local view B consists of requires and excludes relationships.

Figure 93 – Local View A

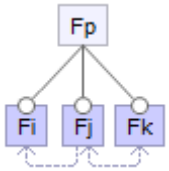

Figure 94 – Local View B

First step in *merging by conformance* is determining the sets of sets of sibling features of views.

$childSets_A$ = { { i } }.

$childSets_B$ = { { }, { i }, { k }, { i, j }, { i, k } }.

Second step is constructing *fs* X and *fs* Y representations.

Table 118 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**
1	***fs* { i }**	+	/	/

Table 119 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**
1	***fs* { }**	-	-	-
2	***fs* { i }**	+	-	-
3	***fs* { k }**	-	-	+
4	***fs* { i, j }**	+	+	-
5	***fs* { i, k }**	+	-	+

Third step is performing combination for each *fs* X and *fs* Y. Table 120 shows the conflict situations that cannot be resolved.

Table 120 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**	**k**
1	*fs* { i }	+	/	/
2	*fs* { }	-	-	-
3	***fs* { i } ⊗ *fs* { }**	!	-	-
4	*fs* { i }	+	/	/
5	*fs* { k }	-	-	+
6	***fs* { i } ⊗ *fs* { k }**	!	-	+

Table 121 gives the result of combinations where *fs* Y's are preserved.

Table 121 - *fs* X ⊗ *fs* Y = *fs* Y

row number		**i**	**j**	**k**
1	*fs* { i }	+	/	/
2	*fs* { i, j }	+	+	-
3	***fs* { i } ⊗ *fs* { i, j } = *fs* { i, j }**	+	+	-
4	*fs* { i }	+	/	/
5	*fs* { i, k }	+	-	+
6	***fs* { i } ⊗ *fs* { i, k } = *fs* { i, k }**	+	-	+

Executing $conform_A$ and $conform_B$ operations, $childSets_R$ is obtained as follows:

$childSets_R$ = { { i }, { i, j }, { i, k } }.

When the FM R is built, the model given in Figure 95 is obtained.

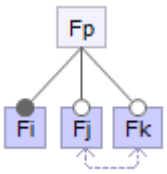

Figure 95 – Resolution of Figure 93 and Figure 94

Following example includes a view with constraint. As shown in Figure 97, local view B consists of requires relationship.

Figure 96 – Local View A

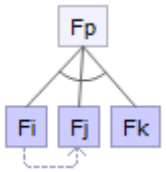

Figure 97 – Local View B

First step in *merging by conformance* is determining the sets of sets of sibling features of views.

$childSets_A$ = { { }, { i } }.

$childSets_B$ = { { j }, { k } }.

Second step is constructing *fs* X and *fs* Y representations.

Table 122 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**
1	***fs* { }**	-	/	/
2	***fs* { i }**	+	/	/

Table 123 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**
1	***fs* { j }**	-	+	-
2	***fs* { k }**	-	-	+

Third step is performing combination for each *fs* X and *fs* Y. Table 124 shows the conflict situations that cannot be resolved.

Table 124 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**	**k**
1	*fs* { i }	+	/	/
2	*fs* { j }	-	+	-
3	***fs* { i } ⊗ *fs* { j }**	!	+	-
4	*fs* { i }	+	/	/
5	*fs* { k }	-	-	+
6	***fs* { i } ⊗ *fs* { k }**	!	-	+

Table 125 gives the result of combinations where *fs* Y's are preserved.

Table 125 - *fs* X ⊗ *fs* Y = *fs* Y

row number		**i**	**j**	**k**
1	*fs* { }	-	/	/
2	*fs* { j }	-	+	-
3	***fs* { } ⊗ *fs* { j } = *fs* { j }**	-	+	-
4	*fs* { }	-	/	/
5	*fs* { k }	-	-	+
6	***fs* { } ⊗ *fs* { k } = *fs* { k }**	-	-	+

Executing $conform_A$ and $conform_B$ operations, $childSets_R$ is obtained as follows:

$childSets_R$ = { { j }, { k } }.

When the FM R is built, the model given in Figure 98 is obtained.

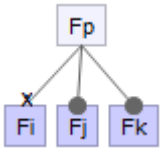

Figure 98 – Resolution of Figure 96 and Figure 97

Following example includes a view with constraints. As shown in Figure 100, local view B consists of requires and excludes relationships.

Figure 99 – Local View A

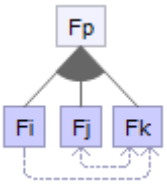

Figure 100 – Local View B

First step in *merging by conformance* is determining the sets of sets of sibling features of views.

$childSets_A$ = { { }, { i } }.

$childSets_B$ = { { j }, { k }, { i, k } }.

Second step is constructing *fs* X and *fs* Y representations.

Table 126 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**
1	***fs* { }**	-	/	/
2	***fs* { i }**	+	/	/

Table 127 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**
1	***fs* { j }**	-	+	-
2	***fs* { k }**	-	-	+
3	***fs* { i, k }**	+	-	+

Third step is performing combination for each *fs* X and *fs* Y. Table 128 shows the conflict situations that cannot be resolved.

Table 128 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**	**k**
1	*fs* { }	-	/	/
2	*fs* { i, k }	+	-	+
3	***fs* { } ⊗ *fs* { i, k }**	!	-	+
4	*fs* { i }	+	/	/
5	*fs* { j }	-	+	-
6	***fs* { i } ⊗ *fs* { j }**	!	+	-
7	*fs* { i }	+	/	/
8	*fs* { k }	-	-	+
9	***fs* { i } ⊗ *fs* { k }**	!	-	+

Table 129 gives the result of combinations where *fs* Y's are preserved.

Table 129 - *fs* X ⊗ *fs* Y = *fs* Y

row number		**i**	**j**	**k**
1	*fs* { }	-	/	/
2	*fs* { j }	-	+	-
3	***fs* { } ⊗ *fs* { j } = *fs* { j }**	-	+	-
4	*fs* { }	-	/	/
5	*fs* { k }	-	-	+
6	***fs* { } ⊗ *fs* { k } = *fs* { k }**	-	-	+
7	*fs* { i }	+	/	/
8	*fs* { i, k }	+	-	+
9	***fs* { i } ⊗ *fs* { i, k } = *fs* { i, k }**	+	-	+

Executing $conform_A$ and $conform_B$ operations, $childSets_R$ is obtained as follows:

$childSets_R$ = { { j }, { k }, { i, k } }.

When the FM R is built, the model given in Figure 101 is obtained.

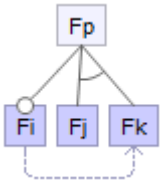

Figure 101 – Resolution of Figure 99 and Figure 100

Following example includes a view with constraint. As shown in Figure 103, local view B consists of requires relationship.

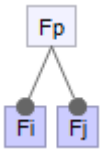

Figure 102 – Local View A

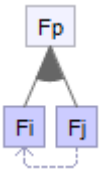

Figure 103 – Local View B

First step in *merging by conformance* is determining the sets of sets of sibling features of views.

$childSets_A$ = { { i, j } }.

$childSets_B$ = { { i }, { i, j } }.

Second step is constructing *fs* X and *fs* Y representations.

Table 130 - Feature Selection Map *fs* X

row number		**i**	**j**
1	***fs* { i, j }**	+	+

Table 131 - Feature Selection Map *fs* Y

row number		**i**	**j**
1	***fs* { i }**	+	-
2	***fs* { i, j }**	+	+

Third step is performing combination for each *fs* X and *fs* Y. Table 132 shows the conflict situation that cannot be resolved.

Table 132 - The case where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**
1	*fs* { i, j }	+	+
2	*fs* { i }	+	-
3	***fs* { i, j } ⊗ *fs* { i }**	+	!

Executing $conform_A$ and $conform_B$ operations, $childSets_R$ is obtained as follows:

$childSets_R$ = { { i, j } }.

When the FM R is built, the model given in Figure 104 is obtained.

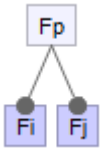

Figure 104 – Resolution of Figure 102 and Figure 103

Following example includes a view with constraint. As shown in Figure 106, local view B consists of requires relationship.

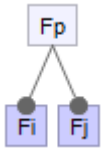

Figure 105 – Local View A

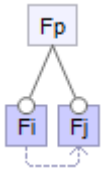

Figure 106 – Local View B

First step in *merging by conformance* is determining the sets of sets of sibling features of views.

$childSets_A$ = { { i, j } }.

$childSets_B$ = { { }, { j }, { i, j } }.

Second step is constructing *fs* X and *fs* Y representations.

Table 133 - Feature Selection Map *fs* X

row number		**i**	**j**
1	***fs* { i, j }**	+	+

Table 134 - Feature Selection Map *fs* Y

row number		**i**	**j**
1	***fs* { }**	-	-
2	***fs* { j }**	-	+
3	***fs* { i, j }**	+	+

Third step is performing combination for each *fs* X and *fs* Y. Table 135 shows the conflict situations that cannot be resolved.

Table 135 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**
1	*fs* { i, j }	+	+
2	*fs* { }	-	-
3	***fs* { i, j } ⊗ *fs* { }**	!	!
4	*fs* { i, j }	+	+
5	*fs* { j }	-	+
6	***fs* { i, j } ⊗ *fs* { j }**	!	+

Executing $conform_A$ and $conform_B$ operations, $childSets_R$ is obtained as follows:

$childSets_R$ = { { i, j } }.

When the FM R is built, the model given in Figure 107 is obtained.

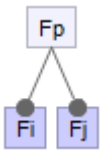

Figure 107 – Resolution of Figure 105 and Figure 106

Following example includes views with constraints. As shown in Figure 108 and Figure 109, views consist of requires and excludes relationships.

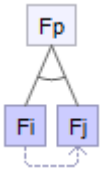

Figure 108 – Local View A

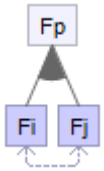

Figure 109 – Local View B

First step in *merging by conformance* is determining the sets of sets of sibling features of views.

$childSets_A$ = { { j } }.

$childSets_B$ = { { i }, { j } }.

Second step is constructing *fs* X and *fs* Y representations.

Table 136 - Feature Selection Map *fs* X

row number		**i**	**j**
1	***fs* { j }**	-	+

Table 137 - Feature Selection Map *fs* Y

row number		**i**	**j**
1	***fs* { i }**	+	-
2	***fs* { j }**	-	+

Third step is performing combination for each *fs* X and *fs* Y. Table 138 shows the conflict situation that cannot be resolved.

Table 138 - The case where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**
1	*fs* { j }	-	+
2	*fs* { i }	+	-
3	***fs* { j } ⊗ *fs* { i }**	!	!

Executing $conform_A$ and $conform_B$ operations, $childSets_R$ is obtained as follows:

$childSets_R$ = { { j } }.

When the FM R is built, the model given in Figure 110 is obtained.

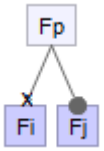

Figure 110 – Resolution of Figure 108 and Figure 109

Following example includes views with constraints. As shown in Figure 111 and Figure 112, views consist of requires and excludes relationships.

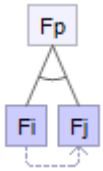

Figure 111 – Local View A

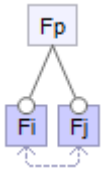

Figure 112 – Local View B

First step in *merging by conformance* is determining the sets of sets of sibling features of views.

$childSets_A$ = { { j } }.

$childSets_B$ = { { }, { i }, { j } }.

Second step is constructing *fs* X and *fs* Y representations.

Table 139 - Feature Selection Map *fs* X

row number		**i**	**j**
1	***fs* { j }**	-	+

Table 140 - Feature Selection Map *fs* Y

row number		**i**	**j**
1	***fs* { }**	-	-
2	***fs* { i }**	+	-
3	***fs* { j }**	-	+

Third step is performing combination for each *fs* X and *fs* Y. Table 141 shows the conflict situations that cannot be resolved.

Table 141 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**
1	*fs* { j }	-	+
2	*fs* { }	-	-
3	***fs* { j } ⊗ *fs* { }**	-	!
4	*fs* { j }	-	+
5	*fs* { i }	+	-
6	***fs* { j } ⊗ *fs* { i }**	!	!

Executing $conform_A$ and $conform_B$ operations, $childSets_R$ is obtained as follows:

$childSets_R$ = { { j } }.

When the FM R is built, the model given in Figure 113 is obtained.

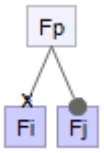

Figure 113 – Resolution of Figure 111 and Figure 112

Following example includes views with constraints. As shown in Figure 114 and Figure 115, views consist of requires and excludes relationships.

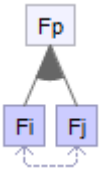

Figure 114 – Local View A

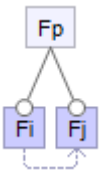

Figure 115 – Local View B

First step in *merging by conformance* is determining the sets of sets of sibling features of views.

$childSets_A$ = { { i }, { j } }.

$childSets_B$ = { { }, { j }, { i, j } }.

Second step is constructing *fs* X and *fs* Y representations.

Table 142 - Feature Selection Map *fs* X

row number		**i**	**j**
1	***fs* { i }**	+	-
2	***fs* { j }**	-	+

Table 143 - Feature Selection Map *fs* Y

row number		**i**	**j**
1	***fs* { }**	-	-
2	***fs* { j }**	-	+
3	***fs* { i, j }**	+	+

Third step is performing combination for each *fs* X and *fs* Y. Table 144 shows the conflict situations that cannot be resolved.

Table 144 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**
1	*fs* { i }	+	-
2	*fs* { }	-	-
3	***fs* { i } ⊗ *fs* { }**	!	-
4	*fs* { i }	+	-
5	*fs* { j }	-	+
6	***fs* { i } ⊗ *fs* { j }**	!	!
7	*fs* { i }	+	-
8	*fs* { i, j }	+	+
9	***fs* { i } ⊗ *fs* { i, j }**	+	!
10	*fs* { j }	-	+
11	*fs* { }	-	-
12	***fs* { j } ⊗ *fs* { }**	-	!
13	*fs* { j }	-	+
14	*fs* { i, j }	+	+
15	***fs* { j } ⊗ *fs* { i, j }**	!	+

Executing $conform_A$ and $conform_B$ operations, $childSets_R$ is obtained as follows:

$childSets_R$ = { { j } }.

When the FM R is built, the model given in Figure 116 is obtained.

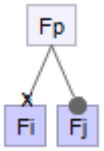

Figure 116 – Resolution of Figure 114 and Figure 115

Following example includes views with constraints. As shown in Figure 117 and Figure 118 views consist of requires relationships.

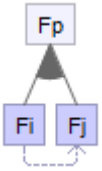

Figure 117 – Local View A

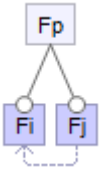

Figure 118 – Local View B

First step in *merging by conformance* is determining the sets of sets of sibling features of views.

$childSets_A$ = { { j }, { i, j } }.

$childSets_B$ = { { }, { i }, { i, j } }.

Second step is constructing *fs* X and *fs* Y representations.

Table 145 - Feature Selection Map *fs* X

row number		**i**	**j**
1	***fs* { j }**	-	+
2	***fs* { i, j }**	+	+

Table 146 - Feature Selection Map *fs* Y

row number		**i**	**j**
1	***fs* { }**	-	-
2	***fs* { i }**	+	-
3	***fs* { i, j }**	+	+

Third step is performing combination for each *fs* X and *fs* Y. Table 147 shows the conflict situations that cannot be resolved.

Table 147 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**
1	*fs* { j }	-	+
2	*fs* { }	-	-
3	***fs* { j } ⊗ *fs* { }**	-	!
4	*fs* { j }	-	+
5	*fs* { i }	+	-
6	***fs* { j } ⊗ *fs* { i }**	!	!
7	*fs* { j }	-	+
8	*fs* { i, j }	+	+
9	***fs* { j } ⊗ *fs* { i, j }**	!	+
10	*fs* { i, j }	+	+
11	*fs* { }	-	-
12	***fs* { i, j } ⊗ *fs* { }**	!	!
13	*fs* { i, j }	+	+
14	*fs* { i }	+	-
15	***fs* { i, j } ⊗ *fs* { i }**	+	!

Executing $conform_A$ and $conform_B$ operations, $childSets_R$ is obtained as follows:

$childSets_R$ = { { i, j } }.

When the FM R is built, the model given in Figure 119 is obtained.

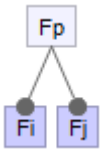

Figure 119 – Resolution of Figure 117 and Figure 118

Following example includes a view with constraints. As shown in Figure 121, local view B consists of requires and excludes relationships.

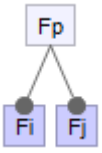

Figure 120 – Local View A

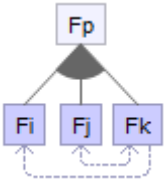

Figure 121 – Local View B

First step in *merging by conformance* is determining the sets of sets of sibling features of views.

$childSets_A$ = { { i, j } }.

$childSets_B$ = { { i }, { j }, { i, j }, { i, k } }.

Second step is constructing *fs* X and *fs* Y representations.

Table 148 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**
1	***fs* { i, j }**	+	+	/

Table 149 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**
1	***fs* { i }**	+	-	-
2	***fs* { j }**	-	+	-
3	***fs* { i, j }**	+	+	-
4	***fs* { i, k }**	+	-	+

Third step is performing combination for each *fs* X and *fs* Y. Table 150 shows the conflict situations that cannot be resolved.

Table 150 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**	**k**
1	*fs* { i, j }	+	+	/
2	*fs* { i }	+	-	-
3	***fs* { i, j } ⊗ *fs* { i }**	+	!	-
4	*fs* { i, j }	+	+	/
5	*fs* { j }	-	+	-
6	***fs* { i, j } ⊗ *fs* { j }**	!	+	-
7	*fs* { i, j }	+	+	/
8	*fs* { i, k }	+	-	+
9	***fs* { i, j } ⊗ *fs* { i, k }**	+	!	+

Executing $conform_A$ and $conform_B$ operations, $childSets_R$ is obtained as follows:

$childSets_R$ = { { i, j } }.

When the FM R by is built, the model given in Figure 122 is obtained.

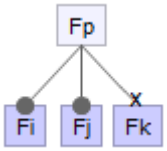

Figure 122 – Resolution of Figure 120 and Figure 121

Following example includes a view with constraint. As shown in Figure 124, local view B consists of requires relationship.

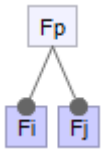

Figure 123 – Local View A

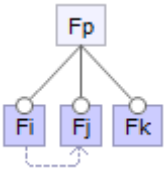

Figure 124 – Local View B

First step in *merging by conformance* is determining the sets of sets of sibling features of views.

$childSets_A$ = { { i, j } }.

$childSets_B$ = { { }, { j }, { k }, { i, j }, { j, k }, { i, j, k } }.

Second step is constructing *fs* X and *fs* Y representations.

Table 151 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**
1	***fs* { i, j }**	+	+	/

Table 152 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**
1	***fs* { }**	-	-	-
2	***fs* { j }**	-	+	-
3	***fs* { k }**	-	-	+
4	***fs* { i, j }**	+	+	-
5	***fs* { j, k }**	-	+	+
6	***fs* { i, j, k }**	+	+	+

Third step is performing combination for each *fs* X and *fs* Y. Table 153 shows the conflict situations that cannot be resolved.

Table 153 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**	**k**
1	*fs* { i, j }	+	+	/
2	*fs* { }	-	-	-
3	***fs* { i, j } ⊗ *fs* { }**	!	!	-
4	*fs* { i, j }	+	+	/
5	*fs* { j }	-	+	-
6	***fs* { i, j } ⊗ *fs* { j }**	!	+	-
7	*fs* { i, j }	+	+	/
8	*fs* { k }	-	-	+
9	***fs* { i, j } ⊗ *fs* { k }**	!	!	+
10	*fs* { i, j }	+	+	/
11	*fs* { j, k }	-	+	+
12	***fs* { i, j } ⊗ *fs* { j, k }**	!	+	+

Table 154 gives the result of *fs* { i, j } ⊗ *fs* { i, j, k } where *fs* Y is preserved.

Table 154 - *fs* X ⊗ *fs* Y = *fs* Y

row number		**i**	**j**	**k**
1	*fs* { i, j }	+	+	/
2	*fs* { i, j, k }	+	+	+
3	***fs* { i, j } ⊗ *fs* { i, j, k } = *fs* { i, j, k }**	+	+	+

Executing $conform_A$ and $conform_B$ operations, $childSets_R$ is obtained as follows:

$childSets_R$ = { { i, j }, { i, j, k } }.

When the FM R is built, the model given in Figure 125 is obtained.

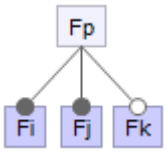

Figure 125 – Resolution of Figure 123 and Figure 124

Following example includes a view with constraints. As shown in Figure 127, local view B consists of requires and excludes relationships.

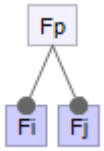

Figure 126 – Local View A

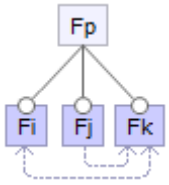

Figure 127 – Local View B

First step in *merging by conformance* is determining the sets of sets of sibling features of views.

$childSets_A$ = { { i, j } }.

$childSets_B$ = { { }, { i }, { k }, { i, j }, { j, k } }.

Second step is constructing *fs* X and *fs* Y representations.

Table 155 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**
1	***fs* { i, j }**	+	+	/

Table 156 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**
1	***fs* { }**	-	-	-
2	***fs* { i }**	+	-	-
3	***fs* { k }**	-	-	+
4	***fs* { i, j }**	+	+	-
5	***fs* { j, k }**	-	+	+

Third step is performing combination for each *fs* X and *fs* Y. Table 157 shows the conflict situations that cannot be resolved.

Table 157 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**	**k**
1	*fs* { i, j }	+	+	/
2	*fs* { }	-	-	-
3	***fs* { i, j } ⊗ *fs* { }**	!	!	-
4	*fs* { i, j }	+	+	/
5	*fs* { i }	+	-	-
6	***fs* { i, j } ⊗ *fs* { i }**	+	!	-
7	*fs* { i, j }	+	+	/
8	*fs* { k }	-	-	+
9	***fs* { i, j } ⊗ *fs* { k }**	!	!	+
10	*fs* { i, j }	+	+	/
11	*fs* { j, k }	-	+	+
12	***fs* { i, j } ⊗ *fs* { j, k }**	!	+	+

Executing $conform_A$ and $conform_B$ operations, $childSets_R$ is obtained as follows:

$childSets_R$ = { { i, j } }.

When the FM R is built, the model given in Figure 128 is obtained.

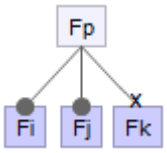

Figure 128 – Resolution of Figure 126 and Figure 127

Following example includes views with constraints. As shown in Figure 129 and Figure 130, views consist of requires and excludes relationships.

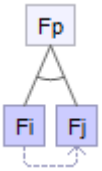

Figure 129 – Local View A

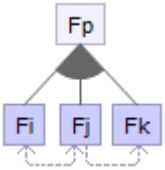

Figure 130 – Local View B

First step in *merging by conformance* is determining the sets of sets of sibling features of views.

$childSets_A$ = { { j } }.

$childSets_B$ = { { i }, { k }, { i, k }, { j, k } }.

Second step is constructing *fs* X and *fs* Y representations.

Table 158 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**
1	***fs* { j }**	-	+	/

Table 159 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**
1	***fs* { }**	-	-	-
2	***fs* { i }**	+	-	-
3	***fs* { k }**	-	-	+
4	***fs* { i, k }**	+	-	+
5	***fs* { j, k }**	-	+	+

Third step is performing combination for each *fs* X and *fs* Y. Table 160 shows the conflict situations that cannot be resolved.

Table 160 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**	**k**
1	*fs* { j }	-	+	/
2	*fs* { }	-	-	-
3	***fs* { j } ⊗ *fs* { }**	-	!	-
4	*fs* { j }	-	+	/
5	*fs* { i }	+	-	-
6	***fs* { j } ⊗ *fs* { i }**	!	!	-
7	*fs* { j }	-	+	/
8	*fs* { k }	-	-	+
9	***fs* { j } ⊗ *fs* { k }**	-	!	+
10	*fs* { j }	-	+	/
11	*fs* { i, k }	+	-	+
12	***fs* { j } ⊗ *fs* { i, k }**	!	!	+

Table 161 gives the result of *fs* { j } ⊗ *fs* { j, k } where *fs* Y is preserved.

Table 161 - *fs* X ⊗ *fs* Y = *fs* Y

row number		**i**	**j**	**k**
1	*fs* { j }	-	+	/
2	*fs* { j, k }	-	+	+
3	***fs* { j } ⊗ *fs* { j, k } = *fs* { j, k }**	-	+	+

Executing *conform*$_A$ and *conform*$_B$ operations, childSets$_R$ is obtained as follows:

childSets$_R$ = { { j, k } }.

When the FM R is built, the model given in Figure 131 is obtained.

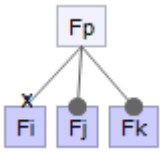

Figure 131 – Resolution of Figure 129 and Figure 130

Following example includes views with constraints. As shown in Figure 132 and Figure 133, views consist of requires and excludes relationships.

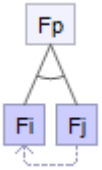

Figure 132 – Local View A

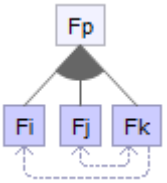

Figure 133 – Local View B

First step in *merging by conformance* is determining the sets of sets of sibling features of views.

childSets$_A$ = { { i } }.

childSets$_B$ = { { i }, { j }, { i, j }, { i, k } }.

Second step is constructing *fs* X and *fs* Y representations.

Table 162 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**
1	***fs* { i }**	+	-	/

Table 163 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**
1	***fs* { i }**	+	-	-
2	***fs* { j }**	-	+	-
3	***fs* { i, j }**	+	+	-
4	***fs* { i, k }**	+	-	+

Third step is performing combination for each *fs* X and *fs* Y. Table 164 shows the conflict situations that cannot be resolved.

Table 164 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**	**k**
1	*fs* { i }	+	-	/
2	*fs* { j }	-	+	-
3	***fs* { i } ⊗ *fs* { j }**	!	!	-
4	*fs* { i }	+	-	/
5	*fs* { i, j }	+	+	-
6	***fs* { i } ⊗ *fs* { i, j }**	+	!	-

Table 165 gives the result of *fs* { i } ⊗ *fs* { i, k } where *fs* Y is preserved.

Table 165 - *fs* X ⊗ *fs* Y = *fs* Y

row number		**i**	**j**	**k**
1	*fs* { i }	+	-	/
2	*fs* { i, k }	+	-	+
3	***fs* { i } ⊗ *fs* { i, k } = *fs* { i, k }**	+	-	+

Executing $conform_A$ and $conform_B$ operations, $childSets_R$ is obtained as follows:

$childSets_R$ = { { i }, { i, k } }.

When the FM R is built, the model given in Figure 134 is obtained.

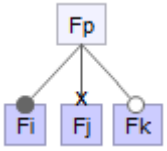

Figure 134 – Resolution of Figure 132 and Figure 133

Following example includes views with constraints. As shown in Figure 135 and Figure 136, views consist of requires relationships.

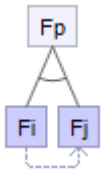

Figure 135 – Local View A

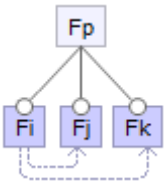

Figure 136 – Local View B

First step in *merging by conformance* is determining the sets of sets of sibling features of views.

$childSets_A$ = { { j } }.

$childSets_B$ = { { }, { j }, { k }, { j, k }, { i, j, k } }.

Second step is constructing *fs* X and *fs* Y representations.

Table 166 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**
1	***fs* { j }**	-	+	/

Table 167 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**
1	***fs* { }**	-	-	-
2	***fs* { j }**	-	+	-
3	***fs* { k }**	-	-	+
4	***fs* { j, k }**	-	+	+
5	***fs* { i, j, k }**	+	+	+

Third step is performing combination for each *fs* X and *fs* Y. Table 168 shows the conflict situations that cannot be resolved.

Table 168 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**	**k**
1	*fs* { j }	-	+	/
2	*fs* { }	-	-	-
3	***fs* { j } ⊗ *fs* { }**	-	!	-
4	*fs* { j }	-	+	/
5	*fs* { k }	-	-	+
6	***fs* { j } ⊗ *fs* { k }**	-	!	+
7	*fs* { j }	-	+	/
8	*fs* { i, j, k }	+	+	+
9	***fs* { j } ⊗ *fs* { i, j, k }**	!	+	+

Table 169 gives the result of *fs* { j } ⊗ *fs* { j, k } where *fs* Y is preserved.

Table 169 - *fs* X ⊗ *fs* Y = *fs* Y

row number		**i**	**j**	**k**
1	*fs* { j }	-	+	/
2	*fs* { j, k }	-	+	+
3	***fs* { j } ⊗ *fs* { j, k } = *fs* { j, k }**	-	+	+

Executing $conform_A$ and $conform_B$ operations, $childSets_R$ is obtained as follows:

$childSets_R$ = { { j }, { j, k } }.

When the FM R is built, the model given in Figure 137 is obtained.

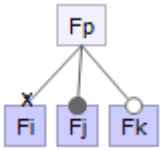

Figure 137 – Resolution of Figure 135 and Figure 136

Following example includes views with constraints. As shown in Figure 138 and Figure 139, views consist of requires and excludes relationships.

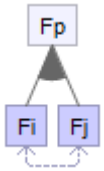

Figure 138 – Local View A

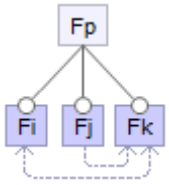

Figure 139 – Local View B

First step in *merging by conformance* is determining the sets of sets of sibling features of views.

$childSets_A$ = { { i }, { j } }.

$childSets_B$ = { { }, { i }, { k }, { j, k } }.

Second step is constructing *fs* X and *fs* Y representations.

Table 170 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**
1	***fs* { i }**	+	-	/
2	***fs* { j }**	-	+	/

Table 171 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**
1	***fs* { }**	-	-	-
2	***fs* { i }**	+	-	-
3	***fs* { k }**	-	-	+
4	***fs* { j, k }**	-	+	+

Third step is performing combination for each *fs* X and *fs* Y. Table 172 shows the conflict situations that cannot be resolved.

Table 172 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**	**k**
1	*fs* { i }	+	-	/
2	*fs* { }	-	-	-
3	***fs* { i } ⊗ *fs* { }**	!	-	-
4	*fs* { i }	+	-	/
5	*fs* { k }	-	-	+
6	***fs* { i } ⊗ *fs* { k }**	!	-	+
7	*fs* { i }	+	-	/
8	*fs* { j, k }	-	+	+
9	***fs* { i } ⊗ *fs* { j, k }**	!	!	+
10	*fs* { j }	-	+	/
11	*fs* { }	-	-	-
12	***fs* { j } ⊗ *fs* { }**	-	!	-
13	*fs* { j }	-	+	/
14	*fs* { i }	+	-	-
15	***fs* { j } ⊗ *fs* { i }**	!	!	-
16	*fs* { j }	-	+	/
17	*fs* { k }	-	-	+
18	***fs* { j } ⊗ *fs* { k }**	-	!	+

Table 173 gives the result of *fs* { j } ⊗ *fs* { j, k } where *fs* Y is preserved.

Table 173 - *fs* X ⊗ *fs* Y = *fs* Y

row number		**i**	**j**	**k**
1	*fs* { j }	-	+	/
2	*fs* { j, k }	-	+	+
3	***fs* { j } ⊗ *fs* { j, k } = *fs* { j, k }**	-	+	+

Executing $conform_A$ and $conform_B$ operations, $childSets_R$ is obtained as follows:

$childSets_R$ = { { i }, { j, k } }.

When the FM R is built, the model given in Figure 140 is obtained.

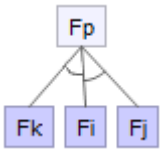

Figure 140 – Resolution of Figure 138 and Figure 139

Following example includes views with constraints. As shown in Figure 141 and Figure 142, views consist of requires and excludes relationships.

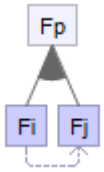

Figure 141 – Local View A

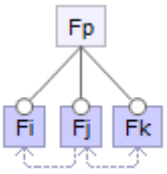

Figure 142 – Local View B

First step in *merging by conformance* is determining the sets of sets of sibling features of views.

$childSets_A$ = { { j }, { i, j } }.

$childSets_B$ = { { }, { i }, { k }, { i, j }, { i, k } }.

Second step is constructing *fs* X and *fs* Y representations.

Table 174 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**
1	***fs* { j }**	-	+	/
2	***fs* { i, j }**	+	+	/

Table 175 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**
1	***fs* { }**	-	-	-
2	***fs* { i }**	+	-	-
3	***fs* { k }**	-	-	+
4	***fs* { i, j }**	+	+	-
5	***fs* { i, k }**	+	-	+

Third step is performing combination for each *fs* X and *fs* Y. Table 176 shows the conflict situations that cannot be resolved.

Table 176 - The cases where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**	**k**
1	*fs* { j }	-	+	/
2	*fs* { }	-	-	-
3	***fs* { j } ⊗ *fs* { }**	-	!	-
4	*fs* { j }	-	+	/
5	*fs* { i }	+	-	-
6	***fs* { j } ⊗ *fs* { i }**	!	!	-
7	*fs* { j }	-	+	/
8	*fs* { k }	-	-	+
9	***fs* { j } ⊗ *fs* { k }**	-	!	+
10	*fs* { j }	-	+	/
11	*fs* { i, j }	+	+	-
12	***fs* { j } ⊗ *fs* { i, j }**	!	+	-
13	*fs* { j }	-	+	/
14	*fs* { i, k }	+	-	+
15	***fs* { j } ⊗ *fs* { i, k }**	!	!	+
16	*fs* { i, j }	+	+	/
17	*fs* { }	-	-	-
18	***fs* { i, j } ⊗ *fs* { }**	!	!	-
19	*fs* { i, j }	+	+	/
20	*fs* { i }	+	-	-
21	***fs* { i, j } ⊗ *fs* { i }**	+	!	-
22	*fs* { i, j }	+	+	/
23	*fs* { k }	-	-	+
24	***fs* { i, j } ⊗ *fs* { k }**	!	!	+
25	*fs* { i, j }	+	+	/
26	*fs* { i, k }	+	-	+
27	***fs* { i, j } ⊗ *fs* { i, k }**	+	!	+

Executing $conform_A$ and $conform_B$ operations, $childSets_R$ is obtained as follows:

$childSets_R$ = { { i, j } }.

When the FM R is built, the model given in Figure 143 is obtained.

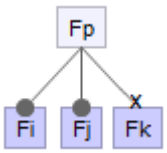

Figure 143 – Resolution of Figure 141 and Figure 142

Following example includes views with constraints. As shown in Figure 144 and Figure 145, views consist of requires and excludes relationships.

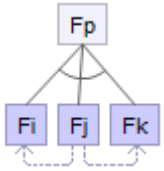

Figure 144 – Local View A

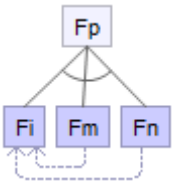

Figure 145 – Local View B

First step in *merging by conformance* is determining the sets of sets of sibling features of views.

$childSets_A$ = { { i }, { k } }.

$childSets_B$ = { { i } }.

Second step is constructing *fs* X and *fs* Y representations.

Table 177 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**	**m**	**n**
1	***fs* { i }**	+	-	-	/	/
2	***fs* { k }**	-	-	+	/	/

Table 178 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**	**m**	**n**
1	***fs* { i }**	+	/	/	-	-

Third step is performing combination for each *fs* X and *fs* Y. Table 179 shows the conflict situation that cannot be resolved.

Table 179 - The case where *fs* X ⊗ *fs* Y involves '!'

row number		**i**	**j**	**k**	**m**	**n**
1	*fs* { k }	-	-	+	/	/
2	*fs* { i }	+	/	/	-	-
3	***fs* { k } ⊗ *fs* { i }**	!	-	+	-	-

Executing $conform_A$ and $conform_B$ operations, $childSets_R$ is obtained as follows:

$childSets_R$ = { { i } }.

When the FM R is built, the model given in Figure 146 is obtained.

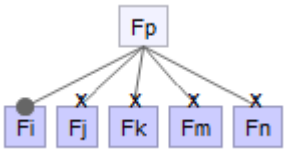

Figure 146 – Resolution of Figure 144 and Figure 145

Following example includes views with constraints. As shown in Figure 147 and Figure 148, views consist of requires relationships.

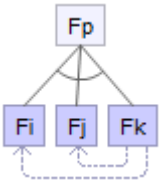

Figure 147 – Local View A

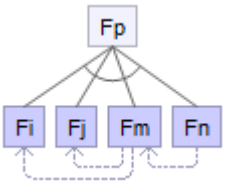

Figure 148 – Local View B

First step in *merging by conformance* is determining the sets of sets of sibling features of views.

$childSets_A$ = { { i }, { j } }.

$childSets_B$ = { { i }, { j } }.

Second step is constructing *fs* X and *fs* Y representations.

Table 180 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**	**m**	**n**
1	***fs* { i }**	+	-	-	/	/
2	***fs* { j }**	-	+	-	/	/

Table 181 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**	**m**	**n**
1	***fs* { i }**	+	-	/	-	-
2	***fs* { j }**	-	+	/	-	-

Third step is performing combination for each *fs* X and *fs* Y. Executing $conform_A$ and $conform_B$ operations, $childSets_R$ is obtained as follows:

$childSets_R$ = { { i }, { j } }.

When the FM R is built, the model given in Figure 149 is obtained.

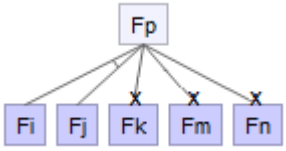

Figure 149 – Resolution of Figure 147 and Figure 148

Following example includes views with constraints. As shown in Figure 150 and Figure 151, views consist of requires and excludes relationships.

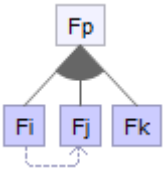

Figure 150 – Local View A

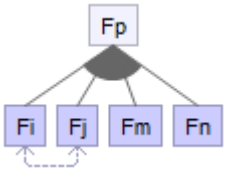

Figure 151 – Local View B

First step in *merging by conformance* is determining the sets of sets of sibling features of views.

$childSets_A$ = { { j }, { k }, { i, j }, { j, k }, { i, j, k } }.

$childSets_B$ = { { i }, { j }, { m }, { n }, { i, m }, { i, n }, { j, m }, { j, n }, { m, n }, { i, m, n }, { j, m, n } }.

Second step is constructing *fs* X and *fs* Y representations.

Table 182 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**	**m**	**n**
1	***fs* { j }**	-	+	-	/	/
2	***fs* { k }**	-	-	+	/	/
3	***fs* { i, j }**	+	+	-	/	/
4	***fs* { j, k }**	-	+	+	/	/
5	***fs* { i, j, k }**	+	+	+	/	/

Table 183 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**	**m**	**n**
1	***fs* { i }**	+	-	/	-	-
2	***fs* { j }**	-	+	/	-	-
3	***fs* { m }**	-	-	/	+	-
4	***fs* { n }**	-	-	/	-	+
5	***fs* { i, m }**	+	-	/	+	-
6	***fs* { i, n }**	+	-	/	-	+
7	***fs* { j, m }**	-	+	/	+	-
8	***fs* { j, n }**	-	+	/	-	+
9	***fs* { m, n }**	-	-	/	+	+
10	***fs* { i, m, n }**	+	-	/	+	+
11	***fs* { j, m, n }**	-	+	/	+	+

Third step is performing combination for each *fs* X and *fs* Y. Table 184 presents the results of combinations where *fs* X's and *fs* Y's are not preserved.

Table 184 - *fs* X ⊗ *fs* Y

row number		**i**	**j**	**k**	**m**	**n**
1	*fs* { j }	-	+	-	/	/
2	*fs* { i }	+	-	/	-	-
3	***fs* { j } ⊗ *fs* { i }**	!	!	-	-	-
4	*fs* { j }	-	+	-	/	/
5	*fs* { m }	-	-	/	+	-
6	***fs* { j } ⊗ *fs* { m }**	-	!	-	+	-
7	*fs* { j }	-	+	-	/	/
8	*fs* { n }	-	-	/	-	+
9	***fs* { j } ⊗ *fs* { n }**	-	!	-	-	+
10	*fs* { j }	-	+	-	/	/
11	*fs* { i, m }	+	-	/	+	-
12	***fs* { j } ⊗ *fs* { i, m }**	!	!	-	+	-
13	*fs* { j }	-	+	-	/	/
14	*fs* { i, n }	+	-	/	-	+
15	***fs* { j } ⊗ *fs* { i, n }**	!	!	-	-	+
16	*fs* { j }	-	+	-	/	/

Table 184 (continued)

row number		**i**	**j**	**k**	**m**	**n**
17	*fs* { m, n }	-	-	/	+	+
18	***fs* { j } ⊗ *fs* { m, n }**	-	!	-	+	+
19	*fs* { j }	-	+	-	/	/
20	*fs* { i, m, n }	+	-	/	+	+
21	***fs* { j } ⊗ *fs* { i, m, n }**	!	!	-	+	+
22	*fs* { k }	-	-	+	/	/
23	*fs* { i }	+	-	/	-	-
24	***fs* { k } ⊗ *fs* { i }**	!	-	+	-	-
25	*fs* { k }	-	-	+	/	/
26	*fs* { j }	-	+	/	-	-
27	***fs* { k } ⊗ *fs* { j }**	-	!	+	-	-
28	*fs* { k }	-	-	+	/	/
29	*fs* { m }	-	-	/	+	-
30	***fs* { k } ⊗ *fs* { m } = *fs* { k, m }**	-	-	+	+	-
31	*fs* { k }	-	-	+	/	/
32	*fs* { n }	-	-	/	-	+
33	***fs* { k } ⊗ *fs* { n } = *fs* { k, n }**	-	-	+	-	+
34	*fs* { k }	-	-	+	/	/
35	*fs* { i, m }	+	-	/	+	-
36	***fs* { k } ⊗ *fs* { i, m }**	!	-	+	+	-
37	*fs* { k }	-	-	+	/	/
38	*fs* { i, n }	+	-	/	-	+
39	***fs* { k } ⊗ *fs* { i, n }**	!	-	+	-	+
40	*fs* { k }	-	-	+	/	/
41	*fs* { j, m }	-	+	/	+	-
42	***fs* { k } ⊗ *fs* { j, m }**	-	!	+	+	-
43	*fs* { k }	-	-	+	/	/
44	*fs* { j, n }	-	+	/	-	+
45	***fs* { k } ⊗ *fs* { j, n }**	-	!	+	-	+
46	*fs* { k }	-	-	+	/	/
47	*fs* { m, n }	-	-	/	+	+
48	***fs* { k } ⊗ *fs* { m, n } = *fs* { k, m, n }**	-	-	+	+	+
49	*fs* { k }	-	-	+	/	/
50	*fs* { i, m, n }	+	-	/	+	+
51	***fs* { k } ⊗ *fs* { i, m, n }**	!	-	+	+	+
52	*fs* { k }	-	-	+	/	/
53	*fs* { j, m, n }	-	+	/	+	+
54	***fs* { k } ⊗ *fs* { j, m, n }**	-	!	+	+	+
55	*fs* { i, j }	+	+	-	/	/
56	*fs* { i }	+	-	/	-	-
57	***fs* { i, j } ⊗ *fs* { i }**	+	!	-	-	-
58	*fs* { i, j }	+	+	-	/	/
59	*fs* { j }	-	+	/	-	-
60	***fs* { i, j } ⊗ *fs* { j }**	!	+	-	-	-
61	*fs* { i, j }	+	+	-	/	/
62	*fs* { m }	-	-	/	+	-

Table 184 (continued)

row number		**i**	**j**	**k**	**m**	**n**
63	***fs* { i, j } ⊗ *fs* { m }**	!	!	-	+	-
64	*fs* { i, j }	+	+	-	/	/
65	*fs* { n }	-	-	/	-	+
66	***fs* { i, j } ⊗ *fs* { n }**	!	!	-	-	+
67	*fs* { i, j }	+	+	-	/	/
68	*fs* { i, m }	+	-	/	+	-
69	***fs* { i, j } ⊗ *fs* { i, m }**	+	!	-	+	-
70	*fs* { i, j }	+	+	-	/	/
71	*fs* { i, n }	+	-	/	-	+
72	***fs* { i, j } ⊗ *fs* { i, n }**	+	!	-	-	+
73	*fs* { i, j }	+	+	-	/	/
74	*fs* { j, m }	-	+	/	+	-
75	***fs* { i, j } ⊗ *fs* { j, m }**	!	+	-	+	-
76	*fs* { i, j }	+	+	-	/	/
77	*fs* { j, n }	-	+	/	-	+
78	***fs* { i, j } ⊗ *fs* { j, n }**	!	+	-	-	+
79	*fs* { i, j }	+	+	-	/	/
80	*fs* { m, n }	-	-	/	+	+
81	***fs* { i, j } ⊗ *fs* { m, n }**	!	!	-	+	+
82	*fs* { i, j }	+	+	-	/	/
83	*fs* { i, m, n }	+	-	/	+	+
84	***fs* { i, j } ⊗ *fs* { i, m, n }**	+	!	-	+	+
85	*fs* { i, j }	+	+	-	/	/
86	*fs* { j, m, n }	-	+	/	+	+
87	***fs* { i, j } ⊗ *fs* { j, m, n }**	!	+	-	+	+
88	*fs* { j, k }	-	+	+	/	/
89	*fs* { i }	+	-	/	-	-
90	***fs* { j, k } ⊗ *fs* { i }**	!	!	+	-	-
91	*fs* { j, k }	-	+	+	/	/
92	*fs* { m }	-	-	/	+	-
93	***fs* { j, k } ⊗ *fs* { m }**	-	!	+	+	-
94	*fs* { j, k }	-	+	+	/	/
95	*fs* { n }	-	-	/	-	+
96	***fs* { j, k } ⊗ *fs* { n }**	!	+	+	-	-
97	*fs* { j, k }	-	+	+	/	/
98	*fs* { i, m }	+	-	/	+	-
99	***fs* { j, k } ⊗ *fs* { i, m }**	!	!	+	+	-
100	*fs* { j, k }	-	+	+	/	/
101	*fs* { i, n }	+	-	/	-	+
102	***fs* { j, k } ⊗ *fs* { i, n }**	!	!	+	-	+
103	*fs* { j, k }	-	+	+	/	/
104	*fs* { j, m }	-	+	/	+	-
105	***fs* { j, k } ⊗ *fs* { j, m } = *fs* { j, k, m }**	-	+	+	+	-
106	*fs* { j, k }	-	+	+	/	/
107	*fs* { j, n }	-	+	/	-	+
108	***fs* { j, k } ⊗ *fs* { j, n } = *fs* { j, k, n }**	-	+	+	-	+
109	*fs* { j, k }	-	+	+	/	/
110	*fs* { m, n }	-	-	/	+	+

Table 184 (continued)

row number		**i**	**j**	**k**	**m**	**n**
111	***fs* { j, k } ⊗ *fs* { m, n }**	-	!	+	+	+
112	*fs* { j, k }	-	+	+	/	/
113	*fs* { i, m, n }	+	-	/	+	+
114	***fs* { j, k } ⊗ *fs* { i, m, n }**	!	!	+	+	+
115	*fs* { j, k }	-	+	+	/	/
116	*fs* { j, m, n }	-	+	/	+	+
117	***fs* { j, k } ⊗ *fs* { j, m, n } = *fs* { j, k, m, n }**	-	+	+	+	+
118	*fs* { i, j, k }	+	+	+	/	/
119	*fs* { i }	+	-	/	-	-
120	***fs* { i, j, k } ⊗ *fs* { i }**	+	!	+	-	-
121	*fs* { i, j, k }	+	+	+	/	/
122	*fs* { j }	-	+	/	-	-
123	***fs* { i, j, k } ⊗ *fs* { j }**	!	+	+	-	-
124	*fs* { i, j, k }	+	+	+	/	/
125	*fs* { m }	-	-	/	+	-
126	***fs* { i, j, k } ⊗ *fs* { m }**	!	!	+	+	-
127	*fs* { i, j, k }	+	+	+	/	/
128	*fs* { n }	-	-	/	-	+
129	***fs* { i, j, k } ⊗ *fs* { n }**	!	!	+	-	+
130	*fs* { i, j, k }	+	+	+	/	/
131	*fs* { i, m }	+	-	/	+	-
132	***fs* { i, j, k } ⊗ *fs* { i, m }**	+	!	+	+	-
133	*fs* { i, j, k }	+	+	+	/	/
134	*fs* { i, n }	+	-	/	-	+
135	***fs* { i, j, k } ⊗ *fs* { i, n }**	+	!	+	-	+
136	*fs* { i, j, k }	+	+	+	/	/
137	*fs* { j, m }	-	+	/	+	-
138	***fs* { i, j, k } ⊗ *fs* { j, m }**	!	+	+	+	-
139	*fs* { i, j, k }	+	+	+	/	/
140	*fs* { j, n }	-	+	/	-	+
141	***fs* { i, j, k } ⊗ *fs* { j, n }**	!	+	+	-	+
142	*fs* { i, j, k }	+	+	+	/	/
143	*fs* { m, n }	-	-	/	+	+
144	***fs* { i, j, k } ⊗ *fs* { m, n }**	!	!	+	+	+
145	*fs* { i, j, k }	+	+	+	/	/
146	*fs* { i, m, n }	+	-	/	+	+
147	***fs* { i, j, k } ⊗ *fs* { i, m, n }**	+	!	+	+	+
148	*fs* { i, j, k }	+	+	+	/	/
149	*fs* { j, m, n }	-	+	/	+	+
150	***fs* { i, j, k } ⊗ *fs* { j, m, n }**	!	+	+	+	+

On the 30th row of Table 184, result of *fs* { k } ⊗ *fs* { m } operation is equal to *fs* { k, m } which is not equal to *fs* { k } or *fs* { m }. Similar situations are presented on the 33rd, 48th, 105th, 108th and 117th rows. This means that on these rows, *fs* X's and *fs* Y's are not preserved.

Table 185 gives the result of combinations where *fs* Y's are preserved.

Table 185 - *fs* X ⊗ *fs* Y = *fs* Y

row number		**i**	**j**	**k**	**m**	**n**
1	*fs* { j }	-	+	-	/	/
2	*fs* { j, m }	-	+	/	+	-
3	***fs* { j } ⊗ *fs* { j, m } = *fs* { j, m }**	-	+	-	+	-
4	*fs* { j }	-	+	-	/	/
5	*fs* { j, n }	-	+	/	-	+
6	***fs* { j } ⊗ *fs* { j, n } = *fs* { j, n }**	-	+	-	-	+
7	*fs* { j }	-	+	-	/	/
8	*fs* { j, m, n }	-	+	/	+	+
9	***fs* { j } ⊗ *fs* { j, m, n } = *fs* { j, m, n }**	-	+	-	+	+

Table 186 gives the result of *fs* { j, k } ⊗ *fs* { j } where *fs* X is preserved.

Table 186 - *fs* X ⊗ *fs* Y = *fs* X

row number		**i**	**j**	**k**	**m**	**n**
1	*fs* { j, k }	-	+	+	/	/
2	*fs* { j }	-	+	/	-	-
3	***fs* { j, k } ⊗ *fs* { j } = *fs* { j, k }**	-	+	+	-	-

From the results of *conform* operations, $childSets_R$ is constructed as follows: $childSets_R$ = { { j }, { j, k }, { j, m }, { j, n }, { j, m, n } }.

When the FM R is built, the model given in Figure 152 is obtained.

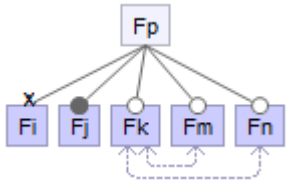

Figure 152 – Resolution of Figure 150 and Figure 151

Following example includes views with constraints. As shown in Figure 153 and Figure 154, views consist of requires relationships.

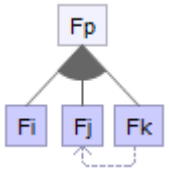

Figure 153 – Local View A

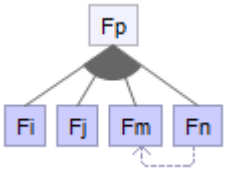

Figure 154 – Local View B

First step in *merging by conformance* is determining the sets of sets of sibling features of views.

$childSets_A$ = { { i }, { j }, { i, j }, { j, k }, { i, j, k } }.

$childSets_B$ = { { i }, { j }, { m }, { i, j }, { i, m }, { j, m }, { m, n }, { i, j, m }, { i, m, n }, { j, m, n }, { i, j, m, n } }.

Second step is constructing *fs* X and *fs* Y representations.

Table 187 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**	**m**	**n**
1	***fs* { i }**	+	-	-	/	/
2	***fs* { j }**	-	+	-	/	/
3	***fs* { i, j }**	+	+	-	/	/
4	***fs* { j, k }**	-	+	+	/	/
5	***fs* { i, j, k }**	+	+	+	/	/

Table 188 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**	**m**	**n**
1	***fs* { i }**	+	-	/	-	-
2	***fs* { j }**	-	+	/	-	-
3	***fs* { m }**	-	-	/	+	-
4	***fs* { i, j }**	+	+	/	-	-
5	***fs* { i, m }**	+	-	/	+	-
6	***fs* { j, m }**	-	+	/	+	-
7	***fs* { m, n }**	-	-	/	+	+
8	***fs* { i, j, m }**	+	+	/	+	-
9	***fs* { i, m, n }**	+	-	/	+	+
10	***fs* { j, m, n }**	-	+	/	+	+
11	***fs* { i, j, m, n }**	+	+	/	+	+

Third step is performing combination for each *fs* X and *fs* Y. Table 189 presents the results of combinations where *fs* X's and *fs* Y's are not preserved.

Table 189 - *fs* X ⊗ *fs* Y

row number		**i**	**j**	**k**	**m**	**n**
1	*fs* { i }	+	-	-	/	/
2	*fs* { j }	-	+	/	-	-
3	***fs* { i } ⊗ *fs* { j }**	!	!	-	-	-
4	*fs* { i }	+	-	-	/	/
5	*fs* { m }	-	-	/	+	-
6	***fs* { i } ⊗ *fs* { m }**	!	-	-	+	-
7	*fs* { i }	+	-	-	/	/
8	*fs* { i, j }	+	+	/	-	-
9	***fs* { i } ⊗ *fs* { i, j }**	+	!	-	-	-
10	*fs* { i }	+	-	-	/	/
11	*fs* { j, m }	-	+	/	+	-
12	***fs* { i } ⊗ *fs* { j, m }**	!	!	-	+	-
13	*fs* { i }	+	-	-	/	/
14	*fs* { m, n }	-	-	/	+	+
15	***fs* { i } ⊗ *fs* { m, n }**	!	-	-	+	+
16	*fs* { i }	+	-	-	/	/
17	*fs* { i, j, m }	+	+	/	+	-
18	***fs* { i } ⊗ *fs* { i, j, m }**	+	!	-	+	-
19	*fs* { i }	+	-	-	/	/
20	*fs* { j, m, n }	-	+	/	+	+
21	***fs* { i } ⊗ *fs* { j, m, n }**	!	!	-	+	+
22	*fs* { i }	+	-	-	/	/
23	*fs* { i, j, m, n }	+	+	/	+	+
24	***fs* { i } ⊗ *fs* { i, j, m, n }**	+	!	-	+	+
25	*fs* { j }	-	+	-	/	/
26	*fs* { m }	-	-	/	+	-
27	***fs* { j } ⊗ *fs* { m }**	-	!	-	+	-
28	*fs* { j }	-	+	-	/	/
29	*fs* { i, j }	+	+	/	-	-
30	***fs* { j } ⊗ *fs* { i, j }**	!	+	-	-	-
31	*fs* { j }	-	+	-	/	/
32	*fs* { i, m }	+	-	/	+	-
33	***fs* { j } ⊗ *fs* { i, m }**	!	!	-	+	-
34	*fs* { j }	-	+	-	/	/
35	*fs* { m, n }	-	-	/	+	+
36	***fs* { j } ⊗ *fs* { m, n }**	-	!	-	+	+
37	*fs* { j }	-	+	-	/	/
38	*fs* { i, j, m }	+	+	/	+	-
39	***fs* { j } ⊗ *fs* { i, j, m }**	!	+	-	+	-
40	*fs* { j }	-	+	-	/	/
41	*fs* { i, m, n }	+	-	/	+	+
42	***fs* { j } ⊗ *fs* { i, m, n }**	!	!	-	+	+
43	*fs* { j }	-	+	-	/	/
44	*fs* { i, j, m, n }	+	+	/	+	+
45	***fs* { j } ⊗ *fs* { i, j, m, n }**	!	+	-	+	+

Table 189 (continued)

row number		**i**	**j**	**k**	**m**	**n**
46	*fs* { i, j }	+	+	-	/	/
47	*fs* { m }	-	-	/	+	-
48	***fs* { i, j } ⊗ *fs* { m }**	!	!	-	+	-
49	*fs* { i, j }	+	+	-	/	/
50	*fs* { i, m }	+	-	/	+	-
51	***fs* { i, j } ⊗ *fs* { i, m }**	+	!	-	+	-
52	*fs* { i, j }	+	+	-	/	/
53	*fs* { j, m }	-	+	/	+	-
54	***fs* { i, j } ⊗ *fs* { j, m }**	!	+	-	+	-
55	*fs* { i, j }	+	+	-	/	/
56	*fs* { m, n }	-	-	/	+	+
57	***fs* { i, j } ⊗ *fs* { m, n }**	!	!	-	+	+
58	*fs* { i, j }	+	+	-	/	/
59	*fs* { i, m, n }	+	-	/	+	+
60	***fs* { i, j } ⊗ *fs* { i, m, n }**	+	!	-	+	+
61	*fs* { i, j }	+	+	-	/	/
62	*fs* { j, m, n }	-	+	/	+	+
63	***fs* { i, j } ⊗ *fs* { j, m, n }**	!	+	-	+	+
64	*fs* { j, k }	-	+	+	/	/
65	*fs* { i }	+	-	/	-	-
66	***fs* { j, k } ⊗ *fs* { i }**	!	!	+	-	-
67	*fs* { j, k }	-	+	+	/	/
68	*fs* { m }	-	-	/	+	-
69	***fs* { j, k } ⊗ *fs* { m }**	-	!	+	+	-
70	*fs* { j, k }	-	+	+	/	/
71	*fs* { i, j }	+	+	/	-	-
72	***fs* { j, k } ⊗ *fs* { i, j }**	!	+	+	-	-
73	*fs* { j, k }	-	+	+	/	/
74	*fs* { i, m }	+	-	/	+	-
75	***fs* { j, k } ⊗ *fs* { i, m }**	!	!	+	+	-
76	*fs* { j, k }	-	+	+	/	/
77	*fs* { j, m }	-	+	/	+	-
78	***fs* { j, k } ⊗ *fs* { j, m } = *fs* { j, k, m }**	-	+	+	+	-
79	*fs* { j, k }	-	+	+	/	/
80	*fs* { m, n }	-	-	/	+	+
81	***fs* { j, k } ⊗ *fs* { m, n }**	-	!	+	+	+
82	*fs* { j, k }	-	+	+	/	/
83	*fs* { i, j, m }	+	+	/	+	-
84	***fs* { j, k } ⊗ *fs* { i, j, m }**	!	+	+	+	-
85	*fs* { j, k }	-	+	+	/	/
86	*fs* { i, m, n }	+	-	/	+	+
87	***fs* { j, k } ⊗ *fs* { i, m, n }**	!	!	+	+	+
88	*fs* { j, k }	-	+	+	/	/

Table 189 (continued)

row number		**i**	**j**	**k**	**m**	**n**
89	*fs* { j, m, n }	-	+	/	+	+
90	***fs* { j, k } ⊗ *fs* { j, m, n } = *fs* { j, k, m, n }**	-	+	+	+	+
91	*fs* { j, k }	-	+	+	/	/
92	*fs* { i, j, m, n }	+	+	/	+	+
93	***fs* { j, k } ⊗ *fs* { i, j, m, n }**	!	+	+	+	+
94	*fs* { i, j, k }	+	+	+	/	/
95	*fs* { i }	+	-	/	-	-
96	***fs* { i, j, k } ⊗ *fs* { i }**	+	!	+	-	-
97	*fs* { i, j, k }	+	+	+	/	/
98	*fs* { j }	-	+	/	-	-
99	***fs* { i, j, k } ⊗ *fs* { j }**	!	+	+	-	-
100	*fs* { i, j, k }	+	+	+	/	/
101	*fs* { m }	-	-	/	+	-
102	***fs* { i, j, k } ⊗ *fs* { m }**	!	!	+	+	-
103	*fs* { i, j, k }	+	+	+	/	/
104	*fs* { i, m }	+	-	/	+	-
105	***fs* { i, j, k } ⊗ *fs* { i, m }**	+	!	+	+	-
106	*fs* { i, j, k }	+	+	+	/	/
107	*fs* { j, m }	-	+	/	+	-
108	***fs* { i, j, k } ⊗ *fs* { j, m }**	!	+	+	+	-
109	*fs* { i, j, k }	+	+	+	/	/
110	*fs* { m, n }	-	-	/	+	+
111	***fs* { i, j, k } ⊗ *fs* { m, n }**	!	!	+	+	+
112	*fs* { i, j, k }	+	+	+	/	/
113	*fs* { i, j, m }	+	+	/	+	-
114	***fs* { i, j, k } ⊗ *fs* { i, j, m } = *fs* { i, j, k, m }**	+	+	+	+	-
115	*fs* { i, j, k }	+	+	+	/	/
116	*fs* { i, m, n }	+	-	/	+	+
117	***fs* { i, j, k } ⊗ *fs* { i, m, n }**	+	!	+	+	+
118	*fs* { i, j, k }	+	+	+	/	/
119	*fs* { j, m, n }	-	+	/	+	+
120	***fs* { i, j, k } ⊗ *fs* { j, m, n }**	!	+	+	+	+
121	*fs* { i, j, k }	+	+	+	/	/
122	*fs* { i, j, m, n }	+	+	/	+	+
123	***fs* { i, j, k } ⊗ *fs* { i, j, m, n } = *fs* { i, j, k, m, n }**	+	+	+	+	+

On the 78th row of Table 189, result of *fs* { j, k } ⊗ *fs* { j, m } operation is equal to *fs* { j, k, m } which is not equal to *fs* { j, k } or *fs* { j, m }. Similar situations are presented on the 90th, 114th and 123rd rows. This means that on these rows, *fs* X's and *fs* Y's are not preserved.

Table 190 gives the result of combinations where *fs* Y's are preserved.

Table 190 - *fs* X ⊗ *fs* Y = *fs* Y

row number		**i**	**j**	**k**	**m**	**n**
1	*fs* { i }	+	-	-	/	/
2	*fs* { i, m }	+	-	/	+	-
3	***fs* { i } ⊗ *fs* { i, m } = *fs* { i, m }**	+	-	-	+	-
4	*fs* { i }	+	-	-	/	/
5	*fs* { i, m, n }	+	-	/	+	+
6	***fs* { i } ⊗ *fs* { i, m, n } = *fs* { i, m, n }**	+	-	-	+	+
7	*fs* { j }	-	+	-	/	/
8	*fs* { j, m }	-	+	/	+	-
9	***fs* { j } ⊗ *fs* { j, m } = *fs* { j, m }**	-	+	-	+	-
10	*fs* { j }	-	+	-	/	/
11	*fs* { j, m, n }	-	+	/	+	+
12	***fs* { j } ⊗ *fs* { j, m, n } = *fs* { j, m, n }**	-	+	-	+	+
13	*fs* { i, j }	+	+	-	/	/
14	*fs* { i, j, m }	+	+	/	+	-
15	***fs* { i, j } ⊗ *fs* { i, j, m } = *fs* { i, j, m }**	+	+	-	+	-
16	*fs* { i, j }	+	+	-	/	/
17	*fs* { i, j, m, n }	+	+	/	+	+
18	***fs* { i, j } ⊗ *fs* { i, j, m, n } = *fs* { i, j, m, n }**	+	+	-	+	+

Table 191 gives the result of combinations where *fs* X's are preserved.

Table 191 - *fs* X ⊗ *fs* Y = *fs* X

row number		**i**	**j**	**k**	**m**	**n**
1	*fs* { j, k }	-	+	+	/	/
2	*fs* { j }	-	+	/	-	-
3	***fs* { j, k } ⊗ *fs* { j } = *fs* { j, k }**	-	+	+	-	-
4	*fs* { i, j, k }	+	+	+	/	/
5	*fs* { i, j }	+	+	/	-	-
6	***fs* { i, j, k } ⊗ *fs* { i, j } = *fs* { i, j, k }**	+	+	+	-	-

From the results of *conform* operations, childSets$_R$ is constructed as follows: childSets$_R$ = { { i }, { j }, { i, j }, { i, m }, { j, m }, { j, k }, { i, j, m }, { i, j, k }, { i, m, n }, { j, m, n }, { i, j, m, n } }.

When the FM R is built, the model given in Figure 155 is obtained.

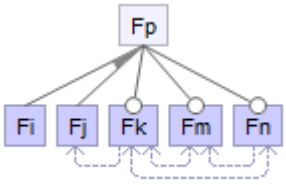

Figure 155 – Resolution of Figure 153 and Figure 154

Following example includes views with constraints. As shown in Figure 156 and Figure 157, views consist of requires and excludes relationships.

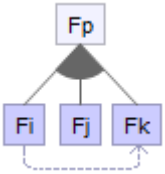

Figure 156 – Local View A

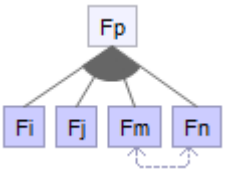

Figure 157 – Local View B

First step in *merging by conformance* is determining the sets of sets of sibling features of views.

$childSets_A$ = { { j }, { k }, { i, k }, { j, k }, { i, j, k } }.

$childSets_B$ = { { i }, { j }, { m }, { n }, { i, j }, { i, m }, { i, n }, { j, m }, { j, n }, { i, j, m }, { i, j, n } }.

Second step is constructing *fs* X and *fs* Y representations.

Table 192 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**	**m**	**n**
1	***fs* { j }**	-	+	-	/	/
2	***fs* { k }**	-	-	+	/	/
3	***fs* { i, k }**	+	-	+	/	/
4	***fs* { j, k }**	-	+	+	/	/
5	***fs* { i, j, k }**	+	+	+	/	/

Table 193 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**	**m**	**n**
1	***fs* { i }**	+	-	/	-	-
2	***fs* { j }**	-	+	/	-	-
3	***fs* { m }**	-	-	/	+	-
4	***fs* { n }**	-	-	/	-	+
5	***fs* { i, j }**	+	+	/	-	-
6	***fs* { i, m }**	+	-	/	+	-
7	***fs* { i, n }**	+	-	/	-	+
8	***fs* { j, m }**	-	+	/	+	-
9	***fs* { j, n }**	-	+	/	-	+
10	***fs* { i, j, m }**	+	+	/	+	-
11	***fs* { i, j, n }**	+	+	/	-	+

Third step is performing combination for each *fs* X and *fs* Y. Table 194 presents the results of combinations where *fs* X's and *fs* Y's are not preserved.

Table 194 - *fs* X ⊗ *fs* Y

row number		**i**	**j**	**k**	**m**	**n**
1	*fs* { j }	-	+	-	/	/
2	*fs* { i }	+	-	/	-	-
3	***fs* { j } ⊗ *fs* { i }**	!	!	-	-	-
4	*fs* { j }	-	+	-	/	/
5	*fs* { m }	-	-	/	+	-
6	***fs* { j } ⊗ *fs* { m }**	-	!	-	+	-
7	*fs* { j }	-	+	-	/	/
8	*fs* { n }	-	-	/	-	+
9	***fs* { j } ⊗ *fs* { n }**	-	!	-	-	+
10	*fs* { j }	-	+	-	/	/
11	*fs* { i, j }	+	+	/	-	-
12	***fs* { j } ⊗ *fs* { i, j }**	!	+	-	-	-
13	*fs* { j }	-	+	-	/	/
14	*fs* { i, m }	+	-	/	+	-
15	***fs* { j } ⊗ *fs* { i, m }**	!	!	-	+	-
16	*fs* { j }	-	+	-	/	/
17	*fs* { i, n }	+	-	/	-	+
18	***fs* { j } ⊗ *fs* { i, n }**	!	!	-	-	+
19	*fs* { j }	-	+	-	/	/
20	*fs* { i, j, m }	+	+	/	+	-
21	***fs* { j } ⊗ *fs* { i, j, m }**	!	+	-	+	-
22	*fs* { j }	-	+	-	/	/
23	*fs* { i, j, n }	+	+	/	-	+
24	***fs* { j } ⊗ *fs* { i, j, n }**	!	+	-	-	+
25	*fs* { k }	-	-	+	/	/

Table 194 (continued)

row number		**i**	**j**	**k**	**m**	**n**
26	*fs* { i }	+	-	/	-	-
27	***fs* { k } ⊗ *fs* { i }**	!	-	+	-	-
28	*fs* { k }	-	-	+	/	/
29	*fs* { j }	-	+	/	-	-
30	***fs* { k } ⊗ *fs* { j }**	-	!	+	-	-
31	*fs* { k }	-	-	+	/	/
32	*fs* { m }	-	-	/	+	-
33	***fs* { k } ⊗ *fs* { m } = *fs* { k, m }**	-	-	+	+	-
34	*fs* { k }	-	-	+	/	/
35	*fs* { n }	-	-	/	-	+
36	***fs* { k } ⊗ *fs* { n } = *fs* { k, n }**	-	-	+	-	+
37	*fs* { k }	-	-	+	/	/
38	*fs* { i, j }	+	+	/	-	-
39	***fs* { k } ⊗ *fs* { i, j }**	!	!	+	-	-
40	*fs* { k }	-	-	+	/	/
41	*fs* { i, m }	+	-	/	+	-
42	***fs* { k } ⊗ *fs* { i, m }**	!	-	+	+	-
43	*fs* { k }	-	-	+	/	/
44	*fs* { i, n }	+	-	/	-	+
45	***fs* { k } ⊗ *fs* { i, n }**	!	-	+	-	+
46	*fs* { k }	-	-	+	/	/
47	*fs* { j, m }	-	+	/	+	-
48	***fs* { k } ⊗ *fs* { j, m }**	-	!	+	+	-
49	*fs* { k }	-	-	+	/	/
50	*fs* { j, n }	-	+	/	-	+
51	***fs* { k } ⊗ *fs* { j, n }**	-	!	+	-	+
52	*fs* { k }	-	-	+	/	/
53	*fs* { i, j, m }	+	+	/	+	-
54	***fs* { k } ⊗ *fs* { i, j, m }**	!	!	+	+	-
55	*fs* { k }	-	-	+	/	/
56	*fs* { i, j, n }	+	+	/	-	+
57	***fs* { k } ⊗ *fs* { i, j, n }**	!	!	+	-	+
58	*fs* { i, k }	+	-	+	/	/
59	*fs* { j }	-	+	/	-	-
60	***fs* { i, k } ⊗ *fs* { j }**	!	!	+	-	-
61	*fs* { i, k }	+	-	+	/	/
62	*fs* { m }	-	-	/	+	-
63	***fs* { i, k } ⊗ *fs* { m }**	!	-	+	+	-
64	*fs* { i, k }	+	-	+	/	/
65	*fs* { n }	-	-	/	-	+
66	***fs* { i, k } ⊗ *fs* { n }**	!	-	+	-	+
67	*fs* { l, k }	+	-	+	/	/
68	*fs* { i, j }	+	+	/	-	-

Table 194 (continued)

row number		**i**	**j**	**k**	**m**	**n**
69	***fs* { i, k } ⊗ *fs* { i, j }**	+	!	+	-	-
70	*fs* { i, k }	+	-	+	/	/
71	*fs* { i, m }	+	-	/	+	-
72	***fs* { i, k } ⊗ *fs* { i, m } = *fs* { i, k, m }**	+	-	+	+	-
73	*fs* { i, k }	+	-	+	/	/
74	*fs* { i, n }	+	-	/	-	+
75	***fs* { i, k } ⊗ *fs* { i, n } = *fs* { i, k, n }**	+	-	+	-	+
76	*fs* { i, k }	+	-	+	/	/
77	*fs* { j, m }	-	+	/	+	-
78	***fs* { i, k } ⊗ *fs* { j, m }**	!	!	+	+	-
79	*fs* { i, k }	+	-	+	/	/
80	*fs* { j, n }	-	+	/	-	+
81	***fs* { i, k } ⊗ *fs* { j, n }**	!	!	+	-	+
82	*fs* { i, k }	+	-	+	/	/
83	*fs* { i, j, m }	+	+	/	+	-
84	***fs* { i, k } ⊗ *fs* { i, j, m }**	+	!	+	+	-
85	*fs* { i, k }	+	-	+	/	/
86	*fs* { i, j, n }	+	+	/	-	+
87	***fs* { i, k } ⊗ *fs* { i, j, n }**	+	!	+	-	+
88	*fs* { j, k }	-	+	+	/	/
89	*fs* { i }	+	-	/	-	-
90	***fs* { j, k } ⊗ *fs* { i }**	!	!	+	-	-
91	*fs* { j, k }	-	+	+	/	/
92	*fs* { m }	-	-	/	+	-
93	***fs* { j, k } ⊗ *fs* { m }**	-	!	+	+	-
94	*fs* { j, k }	-	+	+	/	/
95	*fs* { n }	-	-	/	-	+
96	***fs* { j, k } ⊗ *fs* { n }**	-	!	+	-	+
97	*fs* { j, k }	-	+	+	/	/
98	*fs* { i, j }	+	+	/	-	-
99	***fs* { j, k } ⊗ *fs* { i, j }**	!	+	+	-	-
100	*fs* { j, k }	-	+	+	/	/
101	*fs* { i, m }	+	-	/	+	-
102	***fs* { j, k } ⊗ *fs* { i, m }**	!	!	+	+	-
103	*fs* { j, k }	-	+	+	/	/
104	*fs* { i, n }	+	-	/	-	+
105	***fs* { j, k } ⊗ *fs* { i, n }**	!	!	+	-	+
106	*fs* { j, k }	-	+	+	/	/
107	*fs* { j, m }	-	+	/	+	-
108	***fs* { j, k } ⊗ *fs* { j, m } = *fs* { j, k, m }**	-	+	+	+	-
109	*fs* { j, k }	-	+	+	/	/

Table 194 (continued)

row number		**i**	**j**	**k**	**m**	**n**
110	*fs* { j, n }	-	+	/	-	+
111	***fs* { j, k } ⊗ *fs* { j, n } = *fs* { j, k, n }**	-	+	+	-	+
112	*fs* { j, k }	-	+	+	/	/
113	*fs* { i, j, m }	+	+	/	+	-
114	***fs* { j, k } ⊗ *fs* { i, j, m }**	!	+	+	+	-
115	*fs* { j, k }	-	+	+	/	/
116	*fs* { i, j, n }	+	+	/	-	+
117	***fs* { j, k } ⊗ *fs* { i, j, n }**	!	+	+	-	+
118	*fs* { i, j, k }	+	+	+	/	/
119	*fs* { i }	+	-	/	-	-
120	***fs* { i, j, k } ⊗ *fs* { i }**	+	!	+	-	-
121	*fs* { i, j, k }	+	+	+	/	/
122	*fs* { j }	-	+	/	-	-
123	***fs* { i, j, k } ⊗ *fs* { j }**	!	+	+	-	-
124	*fs* { i, j, k }	+	+	+	/	/
125	*fs* { m }	-	-	/	+	-
126	***fs* { i, j, k } ⊗ *fs* { m }**	!	!	+	+	-
127	*fs* { i, j, k }	+	+	+	/	/
128	*fs* { n }	-	-	/	-	+
129	***fs* { i, j, k } ⊗ *fs* { n }**	!	!	+	-	+
130	*fs* { i, j, k }	+	+	+	/	/
131	*fs* { i, m }	+	-	/	+	-
132	***fs* { i, j, k } ⊗ *fs* { i, m }**	+	!	+	+	-
133	*fs* { i, j, k }	+	+	+	/	/
134	*fs* { i, n }	+	-	/	-	+
135	***fs* { i, j, k } ⊗ *fs* { i, n }**	+	!	+	-	+
136	*fs* { i, j, k }	+	+	+	/	/
137	*fs* { j, m }	-	+	/	+	-
138	***fs* { i, j, k } ⊗ *fs* { j, m }**	!	+	+	+	-
139	*fs* { i, j, k }	+	+	+	/	/
140	*fs* { j, n }	-	+	/	-	+
141	***fs* { i, j, k } ⊗ *fs* { j, n }**	!	+	+	-	+
142	*fs* { i, j, k }	+	+	+	/	/
143	*fs* { i, j, m }	+	+	/	+	-
144	***fs* { i, j, k } ⊗ *fs* { i, j, m } = *fs* { i, j, k, m }**	+	+	+	+	-
145	*fs* { i, j, k }	+	+	+	/	/
146	*fs* { i, j, n }	+	+	/	-	+
147	***fs* { i, j, k } ⊗ *fs* { i, j, n } = *fs* { i, j, k, n }**	+	+	+	-	+

On the 33rd row of Table 194, result of *fs* { k } ⊗ *fs* { m } operation is equal to *fs* { k, m } which is not equal to *fs* { k } or *fs* { m }. Similar situations are presented on the 36th, 72nd, 75th, 108th,

111th, 144th and 147th rows. This means that on these rows, *fs* X's and *fs* Y's are not preserved.

Table 195 gives the result of combinations where *fs* Y's are preserved.

Table 195 - *fs* X ⊗ *fs* Y = *fs* Y

row number		**i**	**j**	**k**	**m**	**n**
1	*fs* { j }	-	+	-	/	/
2	*fs* { j, m }	-	+	/	+	-
3	***fs* { j } ⊗ *fs* { j, m } = *fs* { j, m }**	-	+	-	+	-
4	*fs* { j }	-	+	-	/	/
5	*fs* { j, n }	-	+	/	-	+
6	***fs* { j } ⊗ *fs* { j, n } = *fs* { j, n }**	-	+	-	-	+

Table 196 gives the result of combinations where *fs* X's are preserved.

Table 196 - *fs* X ⊗ *fs* Y = *fs* X

row number		**i**	**j**	**k**	**m**	**n**
1	*fs* { i, k }	+	-	+	/	/
2	*fs* { i }	+	-	/	-	-
3	***fs* { i, k } ⊗ *fs* { i } = *fs* { i, k }**	+	-	+	-	-
4	*fs* { j, k }	-	+	+	/	/
5	*fs* { j }	-	+	/	-	-
6	***fs* { j, k } ⊗ *fs* { j } = *fs* { j, k }**	-	+	+	-	-
7	*fs* { i, j, k }	+	+	+	/	/
8	*fs* { i, j }	+	+	/	-	-
9	***fs* { i, j, k } ⊗ *fs* { i, j } = *fs* { i, j, k }**	+	+	+	-	-

From the results of *conform* operations, $childSets_R$ is constructed as follows: $childSets_R$ = { { j }, { i, k }, { j, m }, { j, n }, { j, k }, { i, j, k } }.

When the FM R is built, the model given in Figure 158 is obtained.

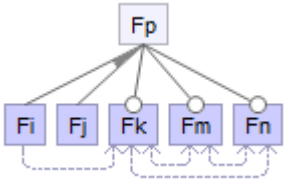

Figure 158 – Resolution of Figure 156 and Figure 157

Following example includes views with constraints. As shown in Figure 159 and Figure 160, views consist of requires and excludes relationships.

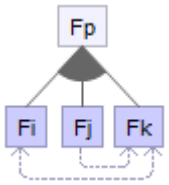

Figure 159 – Local View A

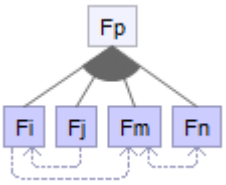

Figure 160 – Local View B

First step in *merging by conformance* is determining the sets of sets of sibling features of views.

$childSets_A$ = { { i }, { k }, { i, j }, { j, k } }.

$childSets_B$ = { { m }, { n }, { i, m }, { i, j, m } }.

Second step is constructing *fs* X and *fs* Y representations.

Table 197 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**	**m**	**n**
1	***fs* { i }**	+	-	-	/	/
2	***fs* { k }**	-	-	+	/	/
3	***fs* { i, j }**	+	+	-	/	/
4	***fs* { j, k }**	-	+	+	/	/

Table 198 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**	**m**	**n**
1	***fs* { m }**	-	-	/	+	-
2	***fs* { n }**	-	-	/	-	+
3	***fs* { i, m }**	+	-	/	+	-
4	***fs* { i, j, m }**	+	+	/	+	-

Third step is performing combination for each *fs* X and *fs* Y. Table 199 presents the results of combinations where *fs* X's and *fs* Y's are not preserved.

Table 199 - *fs* X ⊗ *fs* Y

row number		**i**	**j**	**k**	**m**	**n**
1	*fs* { i }	+	-	-	/	/
2	*fs* { m }	-	-	/	+	-
3	***fs* { i } ⊗ *fs* { m }**	!	-	-	+	-
4	*fs* { i }	+	-	-	/	/
5	*fs* { n }	-	-	/	-	+
6	***fs* { i } ⊗ *fs* { n }**	!	-	-	-	+
7	*fs* { i }	+	-	-	/	/
8	*fs* { i, j, m }	+	+	/	+	-
9	***fs* { i } ⊗ *fs* { i, j, m }**	+	!	-	+	-
10	*fs* { k }	-	-	+	/	/
11	*fs* { m }	-	-	/	+	-
12	***fs* { k } ⊗ *fs* { m } = *fs* { k, m }**	-	-	+	+	-
13	*fs* { k }	-	-	+	/	/
14	*fs* { n }	-	-	/	-	+
15	***fs* { k } ⊗ *fs* { n } = *fs* { k, n }**	-	-	+	-	+
16	*fs* { k }	-	-	+	/	/
17	*fs* { i, m }	+	-	/	+	-
18	***fs* { k } ⊗ *fs* { i, m }**	!	-	+	+	-
19	*fs* { k }	-	-	+	/	/
20	*fs* { i, j, m }	+	+	/	+	-
21	***fs* { k } ⊗ *fs* { i, j, m }**	!	!	+	+	-
22	*fs* { i, j }	+	+	-	/	/
23	*fs* { m }	-	-	/	+	-
24	***fs* { i, j } ⊗ *fs* { m }**	!	!	-	+	-
25	*fs* { i, j }	+	+	-	/	/
26	*fs* { n }	-	-	/	-	+
27	***fs* { i, j } ⊗ *fs* { n }**	!	!	-	-	+
28	*fs* { i, j }	+	+	-	/	/
29	*fs* { i, m }	+	-	/	+	-
30	***fs* { i, j } ⊗ *fs* { i, m }**	+	!	-	+	-
31	*fs* { j, k }	-	+	+	/	/
32	*fs* { m }	-	-	/	+	-
33	***fs* { j, k } ⊗ *fs* { m }**	-	!	+	+	-
34	*fs* { j, k }	-	+	+	/	/
35	*fs* { n }	-	-	/	-	+
36	***fs* { j, k } ⊗ *fs* { n }**	-	!	+	-	+
37	*fs* { j, k }	-	+	+	/	/
38	*fs* { i, m }	+	-	/	+	-
39	***fs* { j, k } ⊗ *fs* { i, m }**	!	!	+	+	-
40	*fs* { j, k }	-	+	+	/	/
41	*fs* { i, j, m }	+	+	/	+	-
42	***fs* { j, k } ⊗ *fs* { i, j, m }**	!	+	+	+	-

On the 12th row of Table 199, result of *fs* { k } ⊗ *fs* { m } operation is equal to *fs* { k, m } which is not equal to *fs* { k } or *fs* { m }. Similar situation is presented on the 15th row. This means that on these rows, *fs* X's and *fs* Y's are not preserved.

Table 200 gives the result of combinations where *fs* Y's are preserved.

Table 200 - *fs* X ⊗ *fs* Y = *fs* Y

row number		**i**	**j**	**k**	**m**	**n**
1	*fs* { i }	+	-	-	/	/
2	*fs* { i, m }	+	-	/	+	-
3	***fs* { i } ⊗ *fs* { i, m } = *fs* { i, m }**	+	-	-	+	-
4	*fs* { i, j }	+	+	-	/	/
5	*fs* { i, j, m }	+	+	/	+	-
6	***fs* { i, j } ⊗ *fs* { i, j, m } = *fs* { i, j, m }**	+	+	-	+	-

From the results of *conform* operations, $childSets_R$ is constructed as follows: $childSets_R$ = { { i, m }, { i, j, m } }.

When the FM R is built, the model given in Figure 161 is obtained.

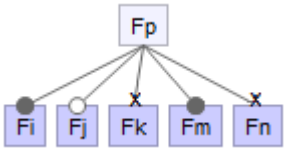

Figure 161 – Resolution of Figure 159 and Figure 160

From the $childSets_R$'s that are obtained at the end of the solutions, resolution FMs can be constructed by following the rules given in chapter 4.

APPENDIX B

EXAMPLES OF RESOLUTION FM CONSTRUCTION WITH CROSS-TREE CONSTRAINTS BETWEEN SIBLINGS

In the following examples, let use some of the child sets of R's given in Appendix A.

$childSets_R$ is constructed above for the 5th example as follows:

$childSets_R$ = { { i }, { i, j }, { i, k } }.

Steps of constructing a table are followed and the Table 201 given below is obtained.

Table 201 - Marking for $childSets_R$ for the 5th example

	i	**j**	**k**
{ i }	1	0	0
{ i, j }	1	1	0
{ i, k }	1	0	1

If the given rules above are applied, i is determined as mandatory as indicated in the first rule, j and k are determined as optional as indicated in the third rule, j and k are determined as in an excludes relationship as indicated in the seventh rule and they are as shown in Figure 95.

Other $childSets_R$ is constructed above for the 7th example as follows:

$childSets_R$ = { { j }, { k }, { i, k } }.

Steps of constructing a table are followed and the Table 202 given below is obtained.

Table 202 - Marking for $childSets_R$ for the 7th example

	i	j	k
{ j }	0	1	0
{ k }	0	0	1
{ i, k }	1	0	1

If the given rules above are applied, i is determined as optional as indicated in the third rule, j and k are determined as in an alternative relationship as indicated in the fourth rule, i and k are determined as in a requires relationship (i requires k) as indicated in the sixth rule and they are as shown in Figure 101.

Another $childSets_R$ is constructed above for the 20th example as follows:

$childSets_R$ = { { i }, { j, k } }.

Steps of constructing a table are followed and the Table 203 given below is obtained.

Table 203 - Marking for $childSets_R$ for the 20th example

	i	j	k
{ i }	1	0	0
{ j, k }	0	1	1

If the given rules above are applied, i and j are determined as in an alternative relationship and similarly i and k are determined as in an alternative relationship as indicated in the fourth rule and they are as shown in Figure 140.

Another $childSets_R$ is constructed above for the 24th example as follows:

$childSets_R$ = { { j }, { j, k }, { j, m }, { j, n }, { j, m, n } }.

Steps of constructing a table are followed and the Table 204 given below is obtained.

Table 204 - Marking for $childSets_R$ for the 24th example

	i	j	k	m	n
{ j }	0	1	0	0	0
{ j, k }	0	1	1	0	0
{ j, m }	0	1	0	1	0
{ j, n }	0	1	0	0	1
{ j, m, n }	0	1	0	1	1

If the given rules above are applied, i is determined as not-available as indicated in the second rule, j is determined as mandatory as indicated in the first rule, k, m and n are determined as optional as indicated in the third rule, k and m are determined as in an

excludes relationship and similarly k and n are determined as in an excludes relationship as indicated in the seventh rule and they are as shown in Figure 152.

Another $childSets_R$ is constructed above for the 25th example as follows:

$childSets_R$ = { { i }, { j }, { i, j }, { i, m }, { j, m }, { j, k }, { i, j, m }, { i, j, k }, { i, m, n }, { j, m, n }, { i, j, m, n } }.

Steps of constructing a table are followed and the Table 205 given below is obtained.

Table 205 - Marking for $childSets_R$ for the 25th example

	i	j	k	m	n
{ i }	1	0	0	0	0
{ j }	0	1	0	0	0
{ i, j }	1	1	0	0	0
{ i, m }	1	0	0	1	0
{ j, m }	0	1	0	1	0
{ j, k }	0	1	1	0	0
{ i, j, m }	1	1	0	1	0
{ i, j, k }	1	1	1	0	0
{ i, m, n }	1	0	0	1	1
{ j, m, n }	0	1	0	1	1
{ i, j, m, n }	1	1	0	1	1

If the given rules above are applied, k, m and n are determined as optional as indicated in the third rule, i, and j are determined as in an or relationship as indicated in the fifth rule, k and j are determined as in a requires relationship (k requires j) and similarly n and m are determined as in a requires relationship (n requires m) as indicated in the sixth rule, k and m are determined as in an excludes relationship and similarly k and n are determined as in an excludes relationship as indicated in the seventh rule and they are as shown in Figure 155.

Another $childSets_R$ is constructed above for the 26th example as follows:

$childSets_R$ = { { j }, { i, k }, { j, m }, { j, n }, { j, k }, { i, j, k } }.

Steps of constructing a table are followed and the Table 206 given below is obtained.

Table 206 - Marking for $childSets_R$ for the 26th example

	i	j	k	m	n
{ j }	0	1	0	0	0
{ i, k }	1	0	1	0	0
{ j, m }	0	1	0	1	0
{ j, n }	0	1	0	0	1
{ j, k }	0	1	1	0	0
{ i, j, k }	1	1	1	0	0

If the given rules above are applied, k, m and n are determined as optional as indicated in the third rule, i and j are determined as in an or relationship as indicated in the fifth rule, i and k are determined as in a requires relationship (i requires k) as indicated in the sixth rule, k and m are determined as in an excludes relationship, k and n are determined as in an excludes relationship and similarly m and n are determined as in an excludes relationship as indicated in the seventh rule and they are as shown in Figure 158.

Another $childSets_R$ is constructed above for the 27th example as follows:

$childSets_R$ = { { i, m }, { i, j, m } }.

Steps of constructing a table are followed and the Table 207 given below is obtained.

Table 207 - Marking for $childSets_R$ for the 27th example

	i	**j**	**k**	**m**	**n**
{ i, m }	1	0	0	1	0
{ i, j, m }	1	1	0	1	0

If the given rules above are applied, i and m are determined as mandatory as indicated in the first rule, k and n are determined as not-available as indicated in the second rule, j is determined as optional as indicated in the third rule and they are as shown in Figure 161.

APPENDIX C

EXAMPLES OF MERGING COMPLETE VIEWS

Following examples are given to verify the algorithm and explain the steps of it by executing.

Figure 162 and Figure 163 represent view A and B, respectively.

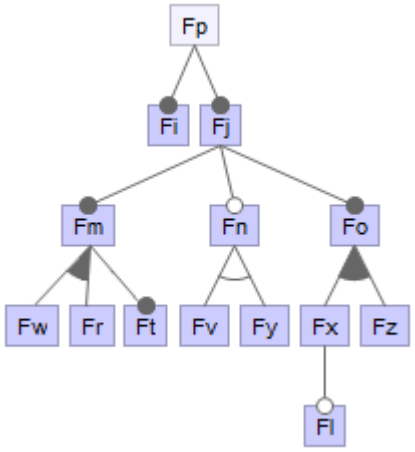

Figure 162 – View A

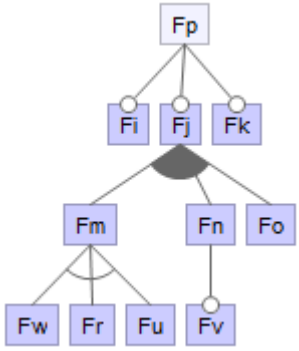

Figure 163 – View B

Assume that these FMs are passed to *mergeByConformance* method as *viewA* and *viewB* parameters.

At the beginning of the execution, value of *level* is assigned to 0. As shown in the figures above, *viewA.depth* is equal to 4 and *viewB.depth* is equal to 3. So, the condition of while is conformed and execution of algorithm passes to line 6. *getFeatures* function returns a set of features of the given FM at the given level. At level 0 of *viewA* and *viewB*, there is a feature p. *featuresA*, *featuresB* and *featuresAtLevel* are equal to { p } after the execution of lines 6, 7 and 8. When the execution of for loop is started, *parentA* and *parentB* is assigned to p and *conformA* and *conformB* is assigned to Ø. *getChilds* function returns the child sets at only

one level below of the given feature of the given FM. After the execution of lines 15 and 16, *childSetsA* is equal to { { i, j } } and *childSetsB* is equal to { { }, { i }, { j }, { k }, { i, j }, { i, k }, { j, k }, { i, j, k } }. Since both of them are not equal to Ø, execution of algorithm passes to line 26. *constructFsX* function returns the given table below:

Table 208 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**
1	***fs* { i, j }**	+	+	/

constructFsY function returns the given table below:

Table 209 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**
1	***fs* { }**	-	-	-
2	***fs* { i }**	+	-	-
3	***fs* { j }**	-	+	-
4	***fs* { k }**	-	-	+
5	***fs* { i, j }**	+	+	-
6	***fs* { i, k }**	+	-	+
7	***fs* { j, k }**	-	+	+
8	***fs* { i, j, k }**	+	+	+

After the execution of lines 26 and 27, *fsXSet* is equal to *fs* X table and *fsYSet* is equal to *fs* Y table which are given above. For loop between lines 28 and 34, performs *fs* X ⊗ *fs* Y operation for each *fs* X ∈ *fsXSet* and searches for *fs* Y ∈ *fsYSet* such that *fs* X ⊗ *fs* Y = *fs* X. *conformB* includes the child sets corresponding to these *fs* X results. After the execution of loop, *conformB* is equal to { { i, j } }. Similarly, for loop between lines 35 and 41, performs *fs* X ⊗ *fs* Y operation for each *fs* Y ∈ *fsYSet* and searches for *fs* X ∈ *fsXSet* such that *fs* X ⊗ *fs* Y = *fs* Y. *conformA* includes the child sets corresponding to these *fs* Y results. After the execution of loop, *conformA* is equal to { { i, j }, { i, j, k } }. *childSetsR* is equal to { { i, j }, { i, j, k } } after the execution of line 42. *constructModelR* builds the FM R from the given set of child sets at the given level by following rules given in chapter 4. The model given in Figure 164 is obtained, after the execution of line 43.

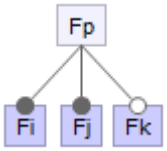

Figure 164 – Level 0 Resolution of Figure 162 and Figure 163

Value of *level* is increased by one and assigned to 1 at line 46 and execution of algorithm is returned to the line 5 again. The condition of while is conformed and execution of algorithm passes to line 6. At level 1 of *viewA* there are i and j features and at level 1 of *viewB* there are i, j and k features. *featuresA* is equal to { i, j }, *featuresB* is equal to { i, j, k } and *featuresAtLevel* is equal to { i, j, k } after the execution of lines 6, 7 and 8. When the execution of for loop is started, *parentA* and *parentB* is assigned to i and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

When the execution of for loop is started again, *parentA* and *parentB* is assigned to j and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* is equal to { { m, o }, { m, n, o } } and *childSetsB* is equal to { { m }, { n }, { o }, { m, n }, { m, o }, { n, o }, { m, n, o } }. Since both of them are not equal to Ø, execution of algorithm passes to line 26. *constructFsX* function returns the given table below:

Table 210 - Feature Selection Map *fs* X

row number		**m**	**n**	**o**
1	***fs* { m, o }**	+	-	+
2	***fs* { m, n, o }**	+	+	+

constructFsY function returns the given table below:

Table 211 - Feature Selection Map *fs* Y

row number		**m**	**n**	**o**
1	***fs* { m }**	+	-	-
2	***fs* { n }**	-	+	-
3	***fs* { o }**	-	-	+
4	***fs* { m, n }**	+	+	-
5	***fs* { m, o }**	+	-	+
6	***fs* { n, o }**	-	+	+
7	***fs* { m, n, o }**	+	+	+

After the execution of lines 26 and 27, *fsXSet* is equal to *fs* X table and *fsYSet* is equal to *fs* Y table which are given above. After the execution of first loop, *conformB* is equal to { { m, o }, { m, n, o } }. After the execution of second loop, *conformA* is equal to { { m, o }, { m, n, o } }. *childSetsR* is equal to { { m, o }, { m, n, o } } after the execution of line 42. The model given in Figure 165 is obtained, after the execution of line 43.

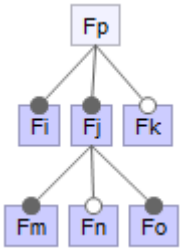

Figure 165 – Level 1 Resolution of Figure 162 and Figure 163

Execution of algorithm returns to line 9 again. When the execution of for loop is started again, *parentA* and *parentB* is assigned to k and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks. Due to there is not any other feature in the *featuresAtLevel,* execution passes to line 46 and value of *level* is increased by one and assigned to 2. Then, execution of algorithm is returned to the line 5 again. The condition of while is conformed and execution of algorithm passes to line 6. At level 2 of *viewA* and *viewB* there are m, n and o features. *featuresA, featuresB* and *featuresAtLevel* are equal to { m, n, o } after the execution of lines 6, 7 and 8. When the execution of for loop is started, *parentA* and *parentB* is assigned to m and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* is equal to { { w, t }, { r, t }, { w, r, t } } and *childSetsB* is equal to { { w }, { r }, { u } }. Since both of them are not equal to Ø, execution of algorithm passes to line 26. *constructFsX* function returns the given table below:

Table 212 - Feature Selection Map *fs* X

row number		**w**	**r**	**t**	**u**
1	***fs* { w, t }**	+	-	+	/
2	***fs* { r, t }**	-	+	+	/
3	***fs* { w, r, t }**	+	+	+	/

constructFsY function returns the given table below:

Table 213 - Feature Selection Map *fs* Y

row number		**w**	**r**	**t**	**u**
1	***fs* { w }**	+	-	/	-
2	***fs* { r }**	-	+	/	-
3	***fs* { u }**	-	-	/	+

After the execution of lines 26 and 27, *fsXSet* is equal to *fs* X table and *fsYSet* is equal to *fs* Y table which are given above. After the execution of first loop, *conformB* is equal to { { w, t }, { r, t } }. After the execution of second loop, *conformA* is equal to Ø. *childSetsR* is equal to { { w, t }, { r, t } } after the execution of line 42. The model given in Figure 166 is obtained, after the execution of line 43.

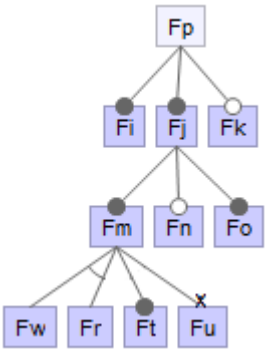

Figure 166 – Part of Level 2 Resolution of Figure 162 and Figure 163

Execution of algorithm returns to line 9 again. When the execution of for loop is started again, *parentA* and *parentB* is assigned to n and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* is equal to { { v }, { y } } and *childSetsB* is equal to { { }, { v } }. Since both of them are not equal to Ø, execution of algorithm passes to line 26. *constructFsX* function returns the given table below:

Table 214 - Feature Selection Map *fs* X

row number		**v**	**y**
1	***fs* { v }**	+	-
2	***fs* { y }**	-	+

constructFsY function returns the given table below:

Table 215 - Feature Selection Map *fs* Y

row number		**v**	**y**
1	***fs* { }**	-	/
2	***fs* { v }**	+	/

After the execution of lines 26 and 27, *fsXSet* is equal to *fs* X table and *fsYSet* is equal to *fs* Y table which are given above. After the execution of first loop, *conformB* is equal to { { v }, { y } }. After the execution of second loop, *conformA* is equal to { { v } }. *childSetsR* is equal to { { v }, { y } } after the execution of line 42. The model given in Figure 167 is obtained, after the execution of line 43.

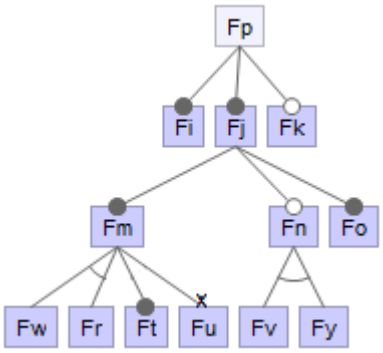

Figure 167 – Part of Level 2 Resolution of Figure 162 and Figure 163

Execution of algorithm returns to line 9 again. When the execution of for loop is started again, *parentA* and *parentB* is assigned to o and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* is equal to { { x }, { z }, { x, z } } and *childSetsB* is equal to Ø. Since *childSetsB* is equal to Ø, execution of algorithm passes to line 19. *moveToViewR* function carries the given child sets branch of the given parent of the given view to the view R. The model given in Figure 168 is obtained, after the execution of line 19.

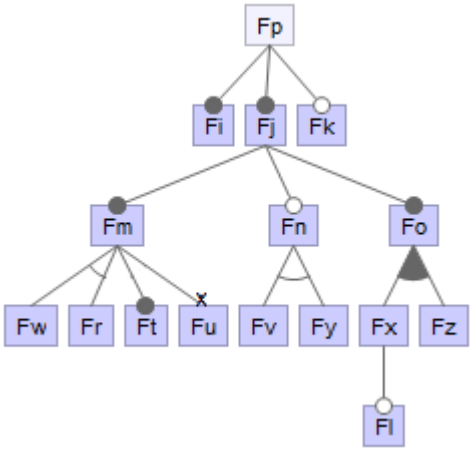

Figure 168 – Level 2 Resolution of Figure 162 and Figure 163

Due to there is not any other feature in the *featuresAtLevel,* execution passes to line 46 and value of *level* is increased by one and assigned to 3. Then execution of algorithm is returned to the line 5 again. The condition of while is conformed and execution of algorithm passes to line 6. At level 3 of *viewA* there are w, r, t, v, y, x and z features and at level 3 of *viewB* there are w, r, u and v features. *featuresA* is equal to { w, r, t, v, y, x, z }, *featuresB* is equal to { w, r, u, v } and *featuresAtLevel* is equal to { w, r, t, u, v, x, y, z } after the execution of lines 6, 7 and 8. When the execution of for loop is started, *parentA* and *parentB* is assigned to w and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

When the execution of for loop is started again, *parentA* and *parentB* is assigned to r and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

When the execution of for loop is started again, *parentA* and *parentB* is assigned to t and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

When the execution of for loop is started again, *parentA* and *parentB* is assigned to u and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

When the execution of for loop is started again, *parentA* and *parentB* is assigned to v and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

When the execution of for loop is started again, *parentA* and *parentB* is assigned to x and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* is equal to { { }, { l } } and *childSetsB* is equal to Ø. Since *childSetsB* is equal to Ø, execution of algorithm passes to line 19. The given child sets branch is already carried to the view R, so *moveToViewR* function does not carry it and execution returns to line 9.

When the execution of for loop is started again, *parentA* and *parentB* is assigned to y and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

When the execution of for loop is started again, *parentA* and *parentB* is assigned to z and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

Due to there is not any other feature in the *featuresAtLevel,* execution passes to line 46 and value of *level* is increased by one and assigned to 4. Then, execution of algorithm is returned to the line 5 again. The condition of while is not conformed this time and execution of algorithm passes to line 47, then 48 and it ends. As a result, FM R is as shown in Figure 168.

For other example, Figure 169 and Figure 170 represent view A and B, respectively.

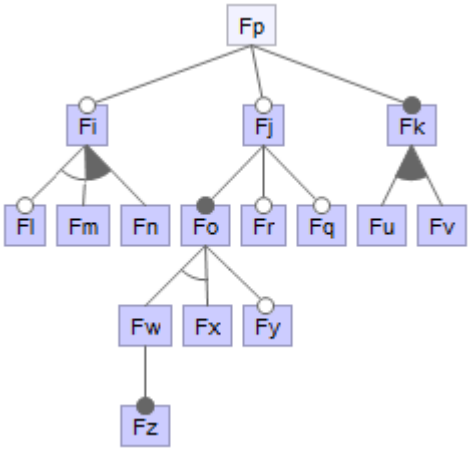

Figure 169 – View A

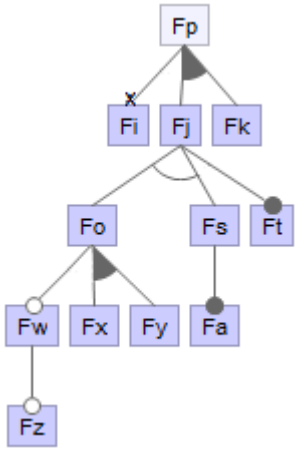

Figure 170 – View B

Assume that these FMs are passed to *mergeByConformance* method as *viewA* and *viewB* parameters.

At the beginning of the execution, value of *level* is assigned to 0. As shown in the figures above, *viewA.depth* and *viewB.depth* are equal to 4. So, the condition of while is conformed and execution of algorithm passes to line 6. At level 0 of *viewA* and *viewB*, there is a feature p. *featuresA*, *featuresB* and *featuresAtLevel* are equal to { p } after the execution of lines 6, 7 and 8. When the execution of for loop is started, *parentA* and *parentB* is assigned to p and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* is equal to { { k }, { i, k }, { j, k }, { i, j, k } } and *childSetsB* is equal to { { j }, { k }, { j, k } }. Since both of them are not equal to Ø, execution of algorithm passes to line 26. *constructFsX* function returns the given table below:

Table 216 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**
1	***fs* { k }**	-	-	+
2	***fs* { i, k }**	+	-	+
3	***fs* { j, k }**	-	+	+
4	***fs* { i, j, k }**	+	+	+

constructFsY function returns the given table below:

Table 217 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**
1	***fs* { j }**	-	+	-
2	***fs* { k }**	-	-	+
3	***fs* { j, k }**	-	+	+

After the execution of lines 26 and 27, *fsXSet* is equal to *fs* X table and *fsYSet* is equal to *fs* Y table which are given above. After the execution of first loop, *conformB* is equal to { { k }, { j, k } }. After the execution of second loop, *conformA* is equal to { { k }, { j, k } }. *childSetsR* is equal to { { k }, { j, k } } after the execution of line 42. The model given in Figure 171 is obtained, after the execution of line 43.

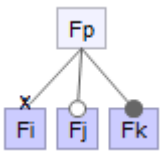

Figure 171 – Level 0 Resolution of Figure 169 and Figure 170

Value of *level* is increased by one and assigned to 1 at line 46 and execution of algorithm is returned to the line 5 again. The condition of while is conformed and execution of algorithm passes to line 6. At level 1 of *viewA* and *viewB* there are i, j and k features. *featuresA*, *featuresB* and *featuresAtLevel* are equal to { i, j, k } after the execution of lines 6, 7 and 8. When the execution of for loop is started, *parentA* and *parentB* is assigned to i and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* is equal to { { m }, { n }, { m, n}, { l, m }, { l, n }, { l, m, n } } and *childSetsB* is equal to Ø. Since *childSetsB* is equal to Ø, execution of algorithm passes to line 19. *moveToViewR* function does not carry the given child sets branch because branch's parent feature i is a not-available feature on the FM R as shown in Figure 171. Execution of algorithm returns to line 9.

When the execution of for loop is started again, *parentA* and *parentB* is assigned to j and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* is equal to { { o }, { o, r }, { o, q }, { o, r, q } } and *childSetsB* is equal to { { o, t }, { s, t } }. Since both of them are not equal to Ø, execution of algorithm passes to line 26. *constructFsX* function returns the given table below:

Table 218 - Feature Selection Map *fs* X

row number		**o**	**r**	**q**	**s**	**t**
1	***fs* { o }**	+	-	-	/	/
2	***fs* { o, r }**	+	+	-	/	/
3	***fs* { o, q }**	+	-	+	/	/
4	***fs* { o, r, q }**	+	+	+		

constructFsY function returns the given table below:

Table 219 - Feature Selection Map *fs* Y

row number		**o**	**r**	**q**	**s**	**t**
1	***fs* { o, t }**	+	/	/	-	+
2	***fs* { s, t }**	-	/	/	+	+

After the execution of lines 26 and 27, *fsXSet* is equal to *fs* X table and *fsYSet* is equal to *fs* Y table which are given above. After the execution of first loop, *conformB* is equal to Ø. After the execution of second loop, *conformA* is equal to { { o, t } }. *childSetsR* is equal to { { o, t } } after the execution of line 42. The model given in Figure 172**Error! Reference source not found.** is obtained, after the execution of line 43.

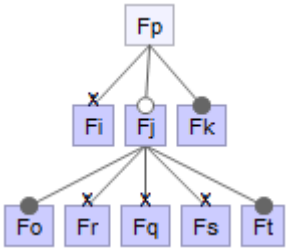

Figure 172 – Part of Level 1 Resolution of Figure 169 and Figure 170

Execution of algorithm returns to line 9. When the execution of for loop is started again, *parentA* and *parentB* is assigned to k and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* is equal to { { u }, { v }, { u, v } } and *childSetsB* is equal to Ø. Since *childSetsB* is equal to Ø, execution of algorithm passes to line 19. The model given in Figure 173 is obtained, after the execution of line 19.

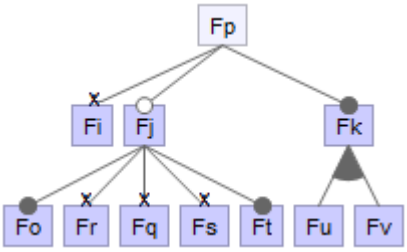

Figure 173 – Level 1 Resolution of Figure 169 and Figure 170

Execution of algorithm returns to line 9. Due to there is not any other feature in the *featuresAtLevel,* execution passes to line 46 and value of *level* is increased by one and assigned to 2. Then, execution of algorithm is returned to the line 5 again. The condition of while is conformed and execution of algorithm passes to line 6. At level 2 of *viewA* there are l, m, n, o, r, q, u and v features and at level 2 of *viewB* there are o, s and t features. *featuresA* is equal to { l, m, n, o, r, q, u, v }, *featuresB* is equal to { o, s, t } and *featuresAtLevel* is equal to { l, m, n, o, r, s, t, q, u, v } after the execution of lines 6, 7 and 8. When the execution of for loop is started, *parentA* and *parentB* is assigned to l and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

When the execution of for loop is started again, *parentA* and *parentB* is assigned to m and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA*

and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

When the execution of for loop is started again, *parentA* and *parentB* is assigned to n and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

When the execution of for loop is started again, *parentA* and *parentB* is assigned to o and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* is equal to { { w }, { x }, { w, y }, { x, y } } and *childSetsB* is equal to { { x }, { y }, { x, y }, { w, x }, { w, y }, { w, x, y } }. Since both of them are not equal to Ø, execution of algorithm passes to line 26. *constructFsX* function returns the given table below:

Table 220 - Feature Selection Map *fs* X

row number		**w**	**x**	**y**
1	***fs* { w }**	+	-	-
2	***fs* { x }**	-	+	-
3	***fs* { w, y }**	+	-	+
4	***fs* { x, y }**	-	+	+

constructFsY function returns the given table below:

Table 221 - Feature Selection Map *fs* Y

row number		**w**	**x**	**y**
1	***fs* { x }**	-	+	-
2	***fs* { y }**	-	-	+
3	***fs* { x, y }**	-	+	+
4	***fs* { w, x }**	+	+	-
5	***fs* { w, y }**	+	-	+
6	***fs* { w, x, y }**	+	+	+

After the execution of lines 26 and 27, *fsXSet* is equal to *fs* X table and *fsYSet* is equal to *fs* Y table which are given above. After the execution of first loop, *conformB* is equal to { { x }, { w, y }, { x, y } }. After the execution of second loop, *conformA* is equal to { { x }, { x, y }, { w, y } }. *childSetsR* is equal to { { x }, { x, y }, { w, y } } after the execution of line 42. The model given in Figure 174 is obtained, after the execution of line 43.

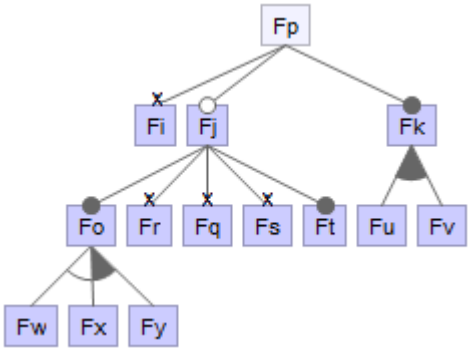

Figure 174 – Level 2 Resolution of Figure 169 and Figure 170

Execution of algorithm returns to line 9. When the execution of for loop is started again, *parentA* and *parentB* is assigned to r and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

When the execution of for loop is started again, *parentA* and *parentB* is assigned to s and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* is equal to Ø and *childSetsB* is equal to { { a } }. Since *childSetsA* is equal to Ø, execution of algorithm passes to line 23. *moveToViewR* function does not carry the given child sets branch because branch's parent feature s is a not-available feature on the FM R as shown in Figure 174. Execution of algorithm returns to line 9.

When the execution of for loop is started again, *parentA* and *parentB* is assigned to t and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

When the execution of for loop is started again, *parentA* and *parentB* is assigned to q and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

When the execution of for loop is started again, *parentA* and *parentB* is assigned to u and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

When the execution of for loop is started again, *parentA* and *parentB* is assigned to v and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA*

and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

Due to there is not any other feature in the *featuresAtLevel,* execution passes to line 46 and value of *level* is increased by one and assigned to 3. Then, execution of algorithm is returned to the line 5 again. The condition of while is conformed and execution of algorithm passes to line 6. At level 3 of *viewA* there are w, x and y features and at level 3 of *viewB* there are a, w, x and y features. *featuresA* is equal to { w, x, y }, *featuresB* is equal to { a, w, x, y } and *featuresAtLevel* is equal to { w, x, y, a } after the execution of lines 6, 7 and 8. When the execution of for loop is started, *parentA* and *parentB* is assigned to w and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* is equal to { { z } } and *childSetsB* is equal to { { }, { z } }. Since both of them are not equal to Ø, execution of algorithm passes to line 26. *constructFsX* function returns the given table below:

Table 222 - Feature Selection Map *fs* X

row number		**z**
1	***fs* { z }**	+

constructFsY function returns the given table below:

Table 223 - Feature Selection Map *fs* Y

row number		**z**
1	***fs* { }**	-
2	***fs* { z }**	+

After the execution of lines 26 and 27, *fsXSet* is equal to *fs* X table and *fsYSet* is equal to *fs* Y table which are given above. After the execution of first loop, *conformB* is equal to { { z } }. After the execution of second loop, *conformA* is equal to { { z } }. *childSetsR* is equal to { { z } } after the execution of line 42. The model given in Figure 175 is obtained, after the execution of line 43.

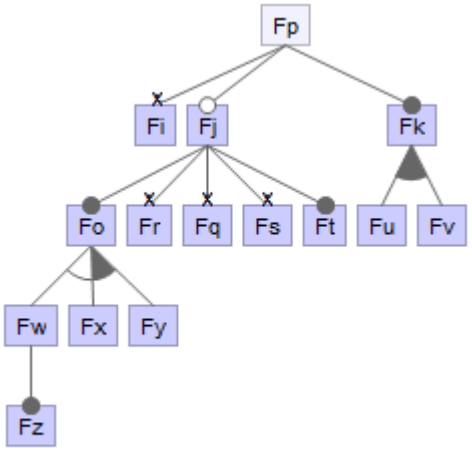

Figure 175 – Level 3 Resolution of Figure 169 and Figure 170

Execution of algorithm returns to line 9. When the execution of for loop is started again, *parentA* and *parentB* is assigned to x and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

When the execution of for loop is started again, *parentA* and *parentB* is assigned to y and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

When the execution of for loop is started again, *parentA* and *parentB* is assigned to a and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

Due to there is not any other feature in the *featuresAtLevel,* execution passes to line 46 and value of *level* is increased by one and assigned to 4. Then, execution of algorithm is returned to the line 5 again. The condition of while is conformed and execution of algorithm passes to line 6. At level 4 of *viewA* and *viewB* there is a feature z. *featuresA*, *featuresB* and *featuresAtLevel* are equal to { z } after the execution of lines 6, 7 and 8. When the execution of for loop is started, *parentA* and *parentB* is assigned to z and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

Due to there is not any other feature in the *featuresAtLevel,* execution passes to line 46 and value of *level* is increased by one and assigned to 5. Then, execution of algorithm is returned to the line 5 again. The condition of while is not conformed this time and execution of algorithm passes to line 47, then 48 and it ends. As a result, FM R is as shown in Figure 175.

For another example, Figure 176 and Figure 177 represent the local views A and B, respectively.

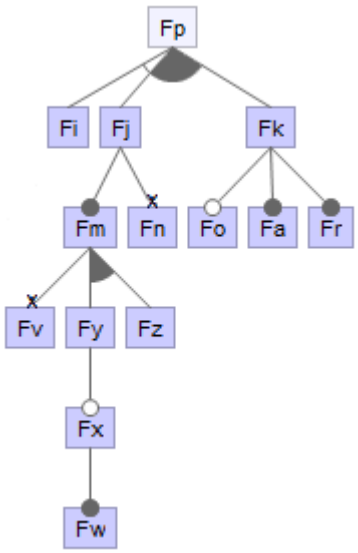

Figure 176 – View A

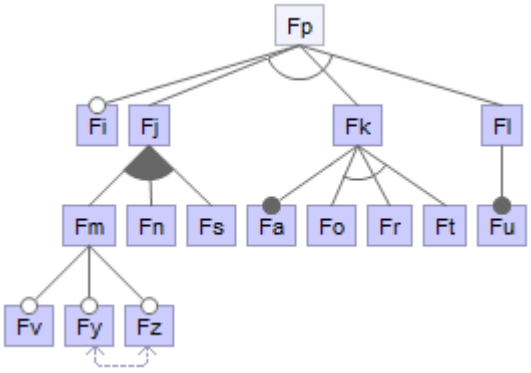

Figure 177 – View B

Assume that these FMs are passed to *mergeByConformance* method as *viewA* and *viewB* parameters.

At the beginning of the execution, value of *level* is assigned to 0. As shown in the figures above, *viewA.depth* is equal to 5 and *viewB.depth* is equal to 3. So, the condition of while is conformed and execution of algorithm passes to line 6. At level 0 of *viewA* and *viewB*, there is a feature p. *featuresA*, *featuresB* and *featuresAtLevel* are equal to { p } after the execution of lines 6, 7 and 8. When the execution of for loop is started, *parentA* and *parentB* is assigned to p and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* is equal to { { j }, { i, k }, { j, k } } and *childSetsB* is equal to { { j }, { k }, { l }, { i, j }, { i, k }, { i, l } }. Since both of them are not equal to Ø, execution of algorithm passes to line 26. *constructFsX* function returns the given table below:

Table 224 - Feature Selection Map *fs* X

row number		**i**	**j**	**k**	**l**
1	***fs* { j }**	-	+	-	/
2	***fs* { i, k }**	+	-	+	/
3	***fs* { j, k }**	-	+	+	/

constructFsY function returns the given table below:

Table 225 - Feature Selection Map *fs* Y

row number		**i**	**j**	**k**	**l**
1	***fs* { j }**	-	+	-	-
2	***fs* { k }**	-	-	+	-
3	***fs* { l }**	-	-	-	+
4	***fs* { i, j }**	+	+	-	-
5	***fs* { i, k }**	+	-	+	-
6	***fs* { i, l }**	+	-	-	+

After the execution of lines 26 and 27, *fsXSet* is equal to *fs* X table and *fsYSet* is equal to *fs* Y table which are given above. After the execution of first loop, *conformB* is equal to { { j }, { i, k } }. After the execution of second loop, *conformA* is equal to { { j }, { i, k } }. *childSetsR* is equal to { { j }, { i, k } } after the execution of line 42. The model given in Figure 178 is obtained, after the execution of line 43.

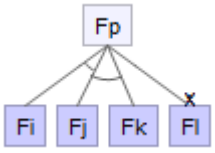

Figure 178 – Level 0 Resolution of Figure 176 and Figure 177

Value of *level* is increased by one and assigned to 1 at line 46 and execution of algorithm is returned to the line 5 again. The condition of while is conformed and execution of algorithm passes to line 6. At level 1 of *viewA* there are i, j and k features and at level 1 of *viewB* there are i, j, k and l features. *featuresA* is equal to { i, j, k }, *featuresB* is equal to { i, j, k, l } and *featuresAtLevel* is equal to { i, j, k, l } after the execution of lines 6, 7 and 8. When the execution of for loop is started, *parentA* and *parentB* is assigned to i and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

When the execution of for loop is started again, *parentA* and *parentB* is assigned to j and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* is equal to { { m } } and *childSetsB* is equal to { { m }, { n }, { s }, { m, n }, { m, s }, { n, s }, { m, n, s } }. Since both of them are not equal to Ø, execution of algorithm passes to line 26. *constructFsX* function returns the given table below:

Table 226 - Feature Selection Map *fs* X

row number		**m**	**n**	**s**
1	***fs* { m }**	+	-	/

constructFsY function returns the given table below:

Table 227 - Feature Selection Map *fs* Y

row number		**m**	**n**	**s**
1	***fs* { m }**	+	-	-
2	***fs* { n }**	-	+	-
3	***fs* { s }**	-	-	+
4	***fs* { m, n }**	+	+	-
5	***fs* { m, s }**	+	-	+
6	***fs* { n, s }**	-	+	+
7	***fs* { m, n, s }**	+	+	+

After the execution of lines 26 and 27, *fsXSet* is equal to *fs* X table and *fsYSet* is equal to *fs* Y table which are given above. After the execution of first loop, *conformB* is equal to { { m } }. After the execution of second loop, *conformA* is equal to { { m }, { m, s } }. *childSetsR* is equal to { { m }, { m, s } } after the execution of line 42. The model given in Figure 179 is obtained, after the execution of line 43.

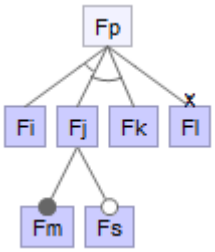

Figure 179 – Part of Level 1 Resolution of Figure 176 and Figure 177

Execution of algorithm returns to line 9. When the execution of for loop is started again, *parentA* and *parentB* is assigned to k and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* is equal to { { p, r }, { o, p, r } } and *childSetsB* is equal to { { o, p }, { p, r } }. Since both of them are not equal to Ø, execution of algorithm passes to line 26. *constructFsX* function returns the given table below:

Table 228 - Feature Selection Map *fs* X

row number		**o**	**p**	**r**	**t**
1	***fs* { p, r }**	-	+	+	/
2	***fs* { p, r, o }**	+	+	+	/

constructFsY function returns the given table below:

Table 229 - Feature Selection Map *fs* Y

row number		**o**	**p**	**r**	**t**
1	***fs* { o, p }**	+	+	-	-
2	***fs* { p, r }**	-	+	+	-

After the execution of lines 26 and 27, *fsXSet* is equal to *fs* X table and *fsYSet* is equal to *fs* Y table which are given above. After the execution of first loop, *conformB* is equal to { { p, r } }. After the execution of second loop, *conformA* is equal to { { p, r } }. *childSetsR* is equal to { { p, r } } after the execution of line 42. The model given in Figure 180 is obtained, after the execution of line 43.

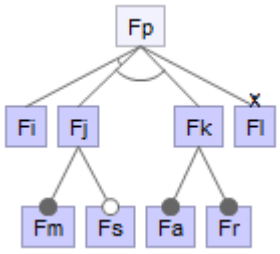

Figure 180 – Level 1 Resolution of Figure 176 and Figure 177

Execution of algorithm returns to line 9. When the execution of for loop is started again, *parentA* and *parentB* is assigned to l and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* is equal to Ø and *childSetsB* is equal to { { u } }. Since *childSetsA* is equal to Ø, execution of algorithm passes to line 23. *moveToViewR* function does not carry the given child sets branch because branch's parent feature l is a

not-available feature on the FM R as shown in Figure 180. Execution of algorithm returns to line 9.

Due to there is not any other feature in the *featuresAtLevel,* execution passes to line 46 and value of *level* is increased by one and assigned to 2. Then, execution of algorithm is returned to the line 5 again. The condition of while is conformed and execution of algorithm passes to line 6. At level 2 of *viewA* there are m, n, o, p and r features and at level 2 of *viewB* there are m, n, o, p, r, s, t and u features. *featuresA* is equal to { m, n, o, p, r }, *featuresB* is equal to { m, n, o, p, r, s, t, u } and *featuresAtLevel* is equal to { m, n, o, p, r, s, t, u } after the execution of lines 6, 7 and 8. When the execution of for loop is started, *parentA* and *parentB* is assigned to m and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* is equal to { { y }, { z }, { y, z } } and *childSetsB* is equal to { { }, { v }, { y }, { z }, { v, y }, { v, z } }. Since both of them are not equal to Ø, execution of algorithm passes to line 26. *constructFsX* function returns the given table below:

Table 230 - Feature Selection Map *fs* X

row number		**v**	**y**	**z**
1	***fs* { y }**	-	+	-
2	***fs* { z }**	-	-	+
3	***fs* { y, z }**	-	+	+

constructFsY function returns the given table below:

Table 231 - Feature Selection Map *fs* Y

row number		**v**	**y**	**z**
1	***fs* { }**	-	-	-
2	***fs* { v }**	+	-	-
3	***fs* { y }**	-	+	-
4	***fs* { z }**	-	-	+
5	***fs* { v, y }**	+	+	-
6	***fs* { v, z }**	+	-	+

After the execution of lines 26 and 27, *fsXSet* is equal to *fs* X table and *fsYSet* is equal to *fs* Y table which are given above. After the execution of first loop, *conformB* is equal to { { y }, { z } }. After the execution of second loop, *conformA* is equal to { { y }, { z } }. *childSetsR* is equal to { { y }, { z } } after the execution of line 42. The model given in Figure 181 is obtained, after the execution of line 43.

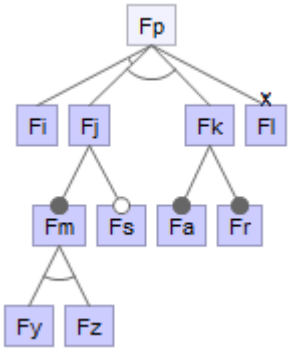

Figure 181 – Level 2 Resolution of Figure 176 and Figure 177

Execution of algorithm returns to line 9. When the execution of for loop is started again, *parentA* and *parentB* is assigned to n and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

When the execution of for loop is started again, *parentA* and *parentB* is assigned to o and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

When the execution of for loop is started again, *parentA* and *parentB* is assigned to p and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

When the execution of for loop is started again, *parentA* and *parentB* is assigned to r and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

When the execution of for loop is started again, *parentA* and *parentB* is assigned to s and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

When the execution of for loop is started again, *parentA* and *parentB* is assigned to t and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

When the execution of for loop is started again, *parentA* and *parentB* is assigned to u and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

Due to there is not any other feature in the *featuresAtLevel,* execution passes to line 46 and value of *level* is increased by one and assigned to 3. Then, execution of algorithm is returned to the line 5 again. The condition of while is conformed and execution of algorithm passes to line 6. At level 3 of *viewA* and *viewB* there are v, y and z features. *featuresA*, *featuresB* and *featuresAtLevel* are equal to { v, y, z } after the execution of lines 6, 7 and 8. When the execution of for loop is started, *parentA* and *parentB* is assigned to v and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

When the execution of for loop is started again, *parentA* and *parentB* is assigned to y and *conformA* and *conformB* is assigned to Ø. After the execution of lines 15 and 16, *childSetsA* is equal to { { }, { x } } and *childSetsB* is equal to Ø. Since *childSetsB* is equal to Ø, execution of algorithm passes to line 19. The model given in Figure 182 is obtained, after the execution of line 19.

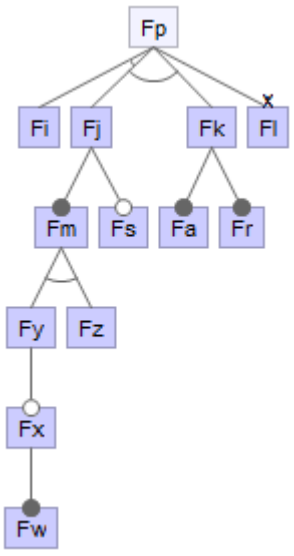

Figure 182 – Level 3 Resolution of Figure 176 and Figure 177

Execution of algorithm returns to line 9. When the execution of for loop is started again, *parentA* and *parentB* is assigned to z and *conformA* and *conformB* is assigned to Ø. After

the execution of lines 15 and 16, *childSetsA* and *childSetsB* are equal to Ø. Since both of them are equal to Ø, execution of algorithm omits the entire if blocks and returns to line 9.

Due to there is not any other feature in the *featuresAtLevel,* execution passes to line 46 and value of *level* is increased by one and assigned to 4. Then, execution of algorithm is returned to the line 5 again. The condition of while is not conformed this time and execution of algorithm passes to line 47, then 48 and it ends. As a result, FM R is as shown in Figure 182.