TECHNISCHE UNIVERSITÄT DRESDEN

FAKULTÄT INFORMATIK

MASTER THESIS

# Error-Tolerant Direct Bases

by

**Wenqian Wang**

June 2013

| | |
|---|---|
| Supervisor: | Prof. Dr.-Ing. Franz Baader |
| Advisors: | Dr. rer. nat. Rafael Peñaloza |
| | Dipl.-Math. Daniel Borchmann |
| External Advisors: | Dipl.-Wirt.-Inf. Tobias Julius Neubert |
| | Dr. rer. nat. Tilmann Häberle |

Technische Universität Dresden

| | |
|---|---|
| Author: | **Wenqian Wang** |
| Matriculation number: | **3735581** |
| Title: | **Error-tolerant Direct Bases** |
| Degree: | **Master of Science** |
| Date of Submission: | **2013-06-16** |

## Declaration

Hereby I certify that the thesis has been written by me. Any help that I have received in my research work has been acknowledged. Additionally, I certify that I have not used any auxiliary sources and literature except those cited in the thesis.

Wenqian Wang

## Abstract

In software enterprise with large organizations, reduction of the time to determine the processor who should solve an issue reported by customers can accelerate software support. Formal Concept Analysis (FCA) is currently a focused topic for analyzing large data. Applying FCA to historical records of completely processed issues can construct the rules that can be used to determine the potentially best processor (as this processor has already solved previous issues of this kind) for the newly reported issues. A Basis in FCA consists of rules representing the causation between symptoms of issues and the responsible components which the processors are in charge of.

One important fact that we cannot omit is that the historical records which are used to build the basis contain a certain amount of errors or imprecise information. Most of the existing bases like basis of proper premises are not suitable to construct rules from the erroneous data, and there are a few bases that can deal with imprecise information like Luxenburger Basis, but it is still not easy to provide appropriate rules for component determination. In this report we define a basis that is error tolerant, which is called *ET Basis* that contains the rules that can deal with imprecise data; we also provide the degree of recommendation for every candidate component. Furthermore, we provide four approaches that can generate a basis that is error tolerant, and we also show the comparisons of the advantages and disadvantages based on the experiments made over incidents for real software issues. We achieved a rate of classification over 90%, i.e. the candidate component with the highest degree of recommendation is the right component in more than 90% of the test data.

## Acknowledgements

I wish to express my sincere gratitude to Professor Franz Baader for giving me the possibility to do research in his group. I sincerely thank Dr. Rafael Peñaloza for all his valuable guidance and consistent encouragement. Deepest thanks to Mr. Daniel Borchmann for all his help in FCA research and programming.

I also would like to express sincere thanks to Dr. Tilmann Häberle and Mr. Tobias Julius Neubert of SAP AG for providing me an opportunity to do my thesis work in their team. Their cordial support and valuable information helped me complete this work.

I am grateful to Professor Steffen Hölldobler for his guidance and support during my study period of Computational Logic. Many thanks to all staff in International Center for Computational Logic.

Most importantly, I would like to thank my family and friends for everything. Special thanks go to my parent, without whom completion of this work would not have been possible.

# Contents

*To my parents.*

# Chapter 1

# Introduction

It is a challenging task to give support for business software. Business critical issues have to be solved in short time with a high quality. This is especially challenging in case of a software suite with broad functionality which is developed and supported by a large organization.

The time an issue needs to be resolved is a composite of the time to determine the processor who should resolve the issue and the time the processor needs to provide a solution. The time frame to determine the processor is wasted time as it does not directly contribute to providing a solution. A study on bug assignment of Eclipse and Mozilla [JKZ09] shows that for verified bugs it takes 19.3 days for Eclipse and 38.1 days for Mozilla to do the first assignment task. Support engineers have to check each issue, detect the root cause and determine who should process this issue. Some issues even need to be reassigned more than once, which increases the time needed to find the right processor.

In the business software area we consider, the time a support engineer spends on the first assignment is much less but the reassignment effect slows the whole procedure down. The main reason for a slow assignment is manual forwarding of the issue until the correct processor is found. This is all manual (human) work, which is very expensive. In order to reduce such cost, determining the correct processor for issues (or incidents) with a high level of accuracy is desired. When an issue is found in a complex software system, which is built from several interconnected modules, it is important to identify which modules are responsible for the problem, and the group of developers in charge of correcting it. *Component* is a term used to represent the module, and a group of developers are responsible to process the issues which belong to this certain component.

Our goal is to determine the right components for newly created software issues. In other words, we want to find the rules which can predict the component from a given set of symptoms (or features). To successfully find the responsible component of a software issue, there are two things we need to do: constructing a group of rules of causation information from historical data, and predicting the right component of the newly created issues using the rules. We use the term *component determination* to describe the procedure to find the right component for the newly created incident.

Formal Concept Analysis (FCA) is a research area of high interest in algebra, mathematics, theoretical computer science, machine learning and data mining, and it is also broadly applied in software technology to process and analyze large data. In FCA theory, data is organized as a set of objects, a set of attributes and a set of relations between objects and attributes. We use *context* to describe the data with this organization. When the data comes from software issues, attributes can be used to describe both features and components. Software issues can result in symptoms which are represented by different combinations of features. We are interested in which combinations of features have strong relation to which components. A context can be represented through a table, where rows represent different incidents and the columns represent features identified by the issue occurrence and the responsible component for each issue. For example, if we want to indicate that the incident with ID “1” using program “UpdateSalesOrder” should be processed in component “CustomerRelationshipManagement”, we mark a cross in the cell where the row indicates “incident 1” and the columns indicate “UpdateSalesOrder” and “CustomerRelationshipManagement” shown in Table 1.1.

| IncidentID | UpdateSalesOrder | CustomerRelationshipManagement |
|---|---|---|
| 1 | × | × |

Table 1.1: example showing an incident

One approach for component determination is applying FCA techniques to historical records about completely processed issues. Briefly, this historical data can be used as training information to identify causation between different features and the component known to be responsible for it. Therefore, the problem of finding these causal relations reduces to the construction of a set of rules, which can tell the causation between features and components. We call such set a *basis* in FCA. We will introduce bases in detail in Chapter 3. One basis in FCA that we can investigate is the *direct basis*, where we can get the exact premises responsible for one attribute. This basis can be used in our case because we can view the combination of features as a premise for component.

The direct base can be used to predict the components for newly created software issues. As a very simple example, consider a rule $x \to y$ where $x$ is a program called "UpdateSalesOrder" and $y$ is the component "CustomerRelationshipManagement". Then when we observe one newly created issue with a feature saying the program name is "UpdateSalesOrder", we can easily say that this issue can be put into the queue of component "CustomerRelationshipManagement".

One direct basis that is commonly used, and which we will look into is the basis of proper premises. A proper premise is a set of features. In the basis of proper premises, for every component we can compute all the sets of features that act as proper premises. The rules between these features and the component can be seen as the causation. We will present this basis in detail in Section 3.2.

A given context might not suffice for fully characterizing the relevant component: the historical data we use to generate the rules might contain errors, or the descriptive attributes might not suffice for distinguishing between different cases. For example, if the component "ServiceApplication" also appears in one record together with the program "UpdateSalesOrder", we need to decide which component is most relevant, "CustomerRelationshipManagement" or "ServiceApplication". In this case, absolute direct bases may be too restrictive for the desired application. Therefore, we then investigate on another basis that allows erroneous data—Luxenburger Basis.

Luxenburger Basis was first introduced by Michael Luxenburger in 1991. Luxenburger worked "implications with exception" also called "partial implications" in [Lux91], where he formalized partial implication and Luxenburger basis. We will give introduction of Luxenburger Basis in detail in Section 3.3.

Since Luxenburger Basis can deal with data containing errors, we try to directly apply this idea to component determination. However, Luxenburger Bases do not produce rules with only one component on the conclusion side, therefore for every new incident, we have to find all components that can be candidate components by computing the closure of the features of the new incident. This approach is computationally very expensive, as shown by experiments in Section 5.1. One solution to this problem is to create one subcontext for each component and use frequent itemsets (as defined for the Luxenburger Basis) to get relevant sets of attributes for each subcontext. The evaluation in Section 5.2 shows that in the approach of subcontext it is hard to find the right component among those candidate components, therefore we try to partition in a different way: randomly and equally partitioning but not necessarily with same component in one subcontext. Then in each subcontext, we compute the proper promises of each component appearing in the subcontext. This approach is shown to be good to find the right components by evaluation in Section 5.3. An extreme case of this approach is that there is exactly one object in one subcontext. Then for one new incident, going through all subcontexts to find candidates is equal to exploring the whole original context, and we use the probabilistic way to find the right components. The corresponding evaluation is shown in Section 5.4.

This report is organized as following: we start in Chapter 2 by introducing background knowledge including basic notions of lattice theory, contexts and concepts of FCA, and closure system used in this report. In Chapter 3 we first focus on "attribute logic" in FCA, where we explain the basic concepts of constructing a basis in detail, and then we introduce some popular bases like those of proper premises and Luxenburger Basis. In Chapter 4, we introduce four approaches, which are the extension of existing bases presented in Chapter 3. The first approach is using closure system; the second one is applying frequent itemsets to partitioned contexts; the third one is partitioning context in a different way but applying proper premises; the last one is using the machine learning. These approaches can deal with errors in the data and can be applied to software issues to find the right components. The implementation evaluations will be shown in Chapter 5. At the end, we present conclusions and future work.

# Chapter 2

# Formal Concept Analysis

Based on order theory and lattice theory, Formal Concept Analysis is a mathematical theory for analyzing data from a group of objects, their attributes and the relations. FCA was first introduced by Rudolf Wille, Bernhard Ganter and Peter Burmeister in early 1980s.

In this chapter, we will first introduce lattice theory, including ordered sets and structure of lattices, which are the foundations of the following chapters. Then contexts and concepts will be introduced in Section 2.2, where we show the most used definitions in this report: concept lattices and many valued contexts. Additionally we show closure systems shortly, which are used in constructing basis.

## 2.1 Lattice Theory

FCA is based on the mathematical order theory and lattice theory. As start point, we introduce ordered sets.

**Definition 1.** *Let $M$ and $N$ be two sets, a* binary relation *$R$ is a set of pairs $(x, y)$ where $x \in M$ and $y \in N$. We write $xRy$ instead of $(x, y) \in R$ to indicate there is a binary relation between $x$ and $y$. An* order relation *(or shortly an* order*)$R$ on a set $M$ is a binary relation which satisfies the following conditions for all $x, y, z \in M$:*

- reflexivity*: $xRx$*
- antisymmetry*: $xRy$ and $x \neq y \Rightarrow$ not $yRx$*
- transitivity*: $xRy$ and $yRz \Rightarrow xRz$*

Usually, we use the symbol $\leq$ to denote the order relation $R$. We write $x < y$ when $x \leq y$ and $x \neq y$. An *ordered set* is a pair $(M, \leq)$ where $M$ is a set and $\leq$ is an order on $M$. We call $x$ the *lower neighbour* of $y$ if $x < y$ and there is no element $z$ such that $x < z < y$. In this case, $y$ is called *upper neighbour* of $x$, and denoted it by $x \prec y$.

**Definition 2.** *The* Supremum *(*least upper bound*) of a subset $S$ of an ordered set $T$ is the least element of $T$ which is greater than or equal to all elements in $S$. Dually, the* infimum *(*greatest lower bound*) of a subset $S$ of an ordered set $T$ is the greatest element of $T$ which is less than or equal to all elements in $S$.*

We can use *Hasse diagrams* to represent ordered sets. Each element in an ordered set $S$ is represented as a small circle. If $x, y \in M$ and $x \prec y$, we draw the circle of $x$ above circle of $y$ and add a line connecting $x$ and $y$. In Figure 2.1 we show all Hasse diagrams of three elements.

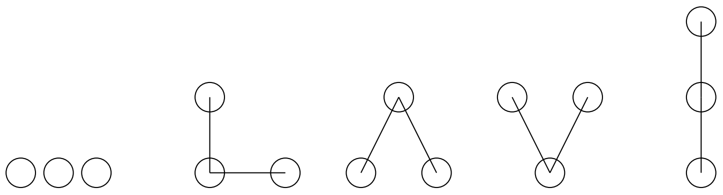

Figure 2.1: Hasse diagrams of three elements

**Definition 3.** *A* lattice *is an ordered set* S:$= (S, \leq)$ *such that: for any two elements $x$ and $y$ in the set $S$, there always exist a unique supremum $x \vee y$ and a unique infimum $x \wedge y$.*

For every two elements $x, y \in S$, it holds that $x \vee (x \wedge y) = x = x \wedge (x \vee y)$. Both lattice operators— supremum and infimum are idempotent, associative and commutative. The order relation of any two elements $x$ and $y$ can be constructed using supremum and infimum:

$$x \leq y \Longleftrightarrow x = x \wedge y \Longleftrightarrow x \vee y = y$$

In a Hasse diagram, the supremum of two nodes is the lowest element that is represented above both of them, and dually the infimum is the largest element below both.

## 2.2 Contexts and Concepts

In this section, we introduce the basic notions of Formal Concept Analysis (FCA), including the notions of formal context and formal concept. The word "formal" is referring to the mathematical notions which are only used in context and concept in standard language. From now on, we leave the word "formal" out and write "context" and "concept" for "formal context" and "formal concept" [GW99]. In FCA, data is organized by a set of objects, a set of attributes, and a set of relations. The software issues we are dealing with can also be organized in the same way. The objects are incidents created by customers; the attributes are the symptoms and the component where the incident belongs. The incident can be described by a list of causation and the component responsible for the incident.

**Definition 4.** *A* (formal) context $\mathbb{K} := (G, M, I)$ *consists of two sets* $G$ *and* $M$ *and a relation* $I$ *between* $G$ *and* $M$*, where* $G$ *is the set of* objects *of the context,* $M$ *is the set of* attributes *of the context. We use* $gIm$ *to express that the object* $g$ *is in* relation $I$ *with the attribute* $m$*.*

A context can also be described using a *cross-table.* The rows and columns of the table correspond to the objects and attributes respectively, and a Boolean value is represented as a cross in the cell $(g, m)$ if object $g$ has attribute $m$. Consider the following example, whose cross-table is shown in Table 2.1, and we use this example to show all relations from cross table.

**Example 1.** *Context* $\mathbb{K} = (M_1, G_1, I_1)$ *contains the set of objects is* $G_1 = \{1, 2, 3, 4, 5\}$*, and the set of attributes is* $M_1 = \{a, b, c, d\}$*. From the positions of crosses, we can tell the set of relations is* $I_1$*:* $(1, a)$*,* $(1, b)$*,* $(1, c)$*,* $(1, d)$*,*$(2, a)$*,* $(2, b)$*,* $(3, b)$*,* $(3, c)$*,* $(3, d)$*,* $(4, b)$*,* $(5, b)$*,* $(5, c)$*.*

| ID | a | b | c | d |
|---|---|---|---|---|
| 1 | × | × | × | × |
| 2 | × | × | | |
| 3 | | × | × | × |
| 4 | | × | | |
| 5 | | × | × | |

Table 2.1: Cross-table

### 2.2.1 Many-Valued Contexts

As defined in the previous section, the attributes from a context are unary, which means the context is one-valued. The software issues we want to deal with are not unary any more. In the example represented in Table 2.2, there are two values in attribute "Program": "Program c" and "Program d", thus to represent this context in the cross-table, it is not sufficient to only use Boolean values for attribute "Program". In this case, the attribute in the

| ID | Interface | Program | Component |
|---|---|---|---|
| 1 | a | c | X |
| 2 | a | d | Y |
| 3 | a | d | Y |

Table 2.2: Many-valued context

context is not Boolean-valued, but many-valued. These attributes are called *many-valued attributes.*

**Definition 5.** *A* many-valued context $\mathbb{K} = (G, M, W, I)$ *consists of sets* $G$*,* $M$ *and* $W$ *and a relation* $I : G \times M \to W$*. We call* $G$ *the set of* objects*,* $M$ *the set of* (many-valued) attributes*, and* $W$ *the set of* attribute values*.*

We use $m(g) = w$ to represent that the object $g$ has attributes $m$ with value $w$. In the table that represents many valued context, the entry in row $g$ and column $m$ represents the attribute $m(g)$. It is sufficient to study the one-valued ones, since many-valued can be transformed to equivalent one-valued. This transformation can be done by *scaling.* The simplest scaling is called *plain scaling.*

**Definition 6.** *A* scale *for the attribute* $m$ *of a many-valued context is a one-valued context* $\mathbb{S} := (G_m, M_m, I_m)$ *with* $m(G) \subseteq G_m$*. The objects of a scale are called* scale values*, the attributes are called* scale attributes*.*

Plain scaling transforms many-valued context to one-valued context with many-valued context $(G, M, W, I)$ and the scale contexts $\mathbb{S}$: The set $G$ of objects stays the same, and every many-valued attribute $m$ is replaced by the scale attribute of the scale $\mathbb{S}$. In other words, every value $m(g)$ is replaced by the row of the scale context $\mathbb{S}_m \subseteq m(g)$ [GW99]. The many valued context in Table 2.2 after scaling is shown in Table 2.3.

| ID | [Interface a] | [Program c] | [Program d] | [Component X] | [Component Y] |
|---|---|---|---|---|---|
| 1 | × | × |  | × |  |
| 2 | × |  | × |  | × |
| 3 | × |  | × |  | × |

Table 2.3: Many-valued context after scaling

### 2.2.2 Concept Lattices

Following the previous sections about context, we introduce concepts and the concept lattice.

**Definition 7.** *For a set $A \subseteq G$ of objects we define*

$$A' := \{m \in M \mid gIm \text{ for all } g \in A\}$$

*For a set $B \subseteq M$ of attributes we define*

$$B' := \{g \in G \mid gIm \text{ for all } m \in B\}$$

Intuitively, $A'$ is a set of common attributes belonging to all objects in $A$, and $B'$ is set of objects which have all attributes in $B$.

**Definition 8.** *A* (formal) concept *for a context $(G, M, I)$ is a pair $(A, B)$ with $A \subseteq G$, $B \subseteq M$ such that $A' = B$, $B' = A$. We call $A$ the* extent *and $B$ the* intents *of concept $(A, B)$.*

Intuitively, a concept is a maximal rectangle of the table that is full of crosses. In the context $\mathbb{K}_1$ from Example 1, $\{1, 3\}' = \{b, c, d\}$ and $\{b, c, d\}' = \{1, 3\}$. Hence the pair $(\{1, 3\}, \{b, c, d\})$ is a concept in context $\mathbb{K}_1$. This concept in shown in Table 2.4. Similarly, we can find all concepts in Example 1: $(\{1, 2, 3, 4, 5\}, \{b\})$, $(\{1, 2\}, \{a, b\})$, $(\{1, 3, 5\}, \{b, c\})$, $(\{1, 3\}, \{b, c, d\})$, $(\emptyset, \{a, b, c, d\})$

| ID | b | c | d |
|---|---|---|---|
| 1 | × | × | × |
| 3 | × | × | × |

Table 2.4: A concept in the context from Example 1

**Definition 9.** *If $(A_1, B_1)$ and $(A_2, B_2)$ are concepts of a context, $(A_1, B_1)$ is called* subconcept *if $A_1 \subseteq A_2$, and $B_2 \subseteq B_1$. In this case, $(A_2, B_2)$ is a* superconcept *of $(A_1, B_1)$, and we write $(A_1, B_1) \leq (A_2, B_2)$. The relation $\leq$ is called the* order of concepts. *The* concept lattice *of a formal content $(G, M, I)$ is the set of all formal concepts of $(G, M, I)$, associated with the partial order*

$$(A_1, B_1) \leq (A_2, B_2) \textit{ iff } A_1 \subseteq A_2 \textit{ and } B_2 \subseteq B_1$$

In the concept lattice, for every two concepts $(A_1, B_1)$ and $(A_2, B_2)$, the greatest common subconcept is $(A_1 \cap A_2, (B_1 \cup B_2)'')$ and the least common superconcept is $((A_1 \cup A_2)'', B_1 \cap B_2)$. The concept lattice of Example 1 is shown in Figure 2.2.2.

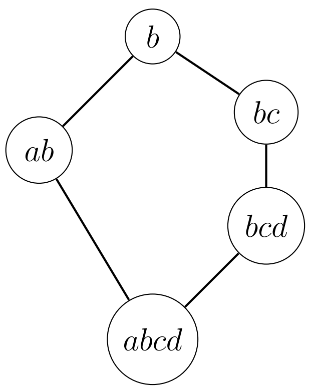

Figure 2.2: Concept Lattice of Example 1

## 2.3 Closure System

**Definition 10.** *A* closure system $\mathbb{F}$ *on a set $S$ is a set of subsets which contains $S$ ($S \in \mathbb{F}$) and is also* closed under intersection*:*

$$\textit{If } F_1, F_2 \subseteq \mathbb{F}, \textit{ then } F_1 \cap F_2 \in \mathbb{F}$$

*A* closure operator *$\varphi$ is a map on the powers set of $S$. The closure operator satisfies:*

- *monotonic: for all $X, Y \subseteq S, X \subseteq Y \Rightarrow \varphi(X) \subseteq \varphi(Y)$*
- *extensive: for all $X \subseteq S, X \subseteq \varphi(X)$*
- *idempotent: for all $X \subseteq S, \varphi^2(X) = \varphi(\varphi(X)) = \varphi(X)$*

From the property of monotonicity and idempotency, we can get a the following conclusion: $\forall X, Y \subseteq S$: $X \subseteq \varphi(Y) \Leftrightarrow \varphi(X) \subseteq \varphi(Y)$. The set $\varphi(X)$ is called the *closure* of $X$ by $\varphi$. The set $X$ is said to be *closed* by $\varphi$ when $\varphi(X) = X$.

**Proposition 1.** *[BM10] Every complete lattice is isomorphic to the lattice of all closures of a closure system, and vice versa.*

For each element in the power set of $M$, the concept intents are the largest sets among those with the same closure [Stu02]. The set of all concept intents of a formal context form a closure system, with the closure operator is $\varphi(X) = X''$. In Example 1, for each element in the power set of $M$, the corresponding closure is shown in Table 2.5.

| closure | set of attributes |
|---|---|
| $\emptyset$ | $\emptyset$ |
| $\{b\}$ | $\{b\}$ |
| $\{a, b\}$ | $\{a\}, \{a, b\}$ |
| $\{b, c\}$ | $\{c\}, \{b, c\}$ |
| $\{b, c, d\}$ | $\{d\}, \{b, d\}, \{c, d\}, \{b, c, d\}$ |
| $\{a, b, c, d\}$ | $\{a, c\}, \{a, d\}, \{a, b, c\}, \{a, b, d\}, \{a, c, d\}, \{a, b, c, d\}$ |

Table 2.5: Sets with same closure in Example 1

In the next chapter, concept lattices and closure systems will be used to construct bases and help us find the right components.

# Chapter 3

# Bases in FCA

From the previous chapter, we can view the records of software issues as contexts. To find the right component for newly created issues, we need to investigate the methods that can generate rules, and with these rules we can predict the right components for the newly created issues.

When all incidents caused by feature $a$ and feature $b$ belong to component $c$, we can say there is a *dependency* between feature $a$ and feature $b$ and component $c$. In general, when all objects that have all attributes in a set $P$ also have all attributes in a set $Q$, then there is a *dependency* between $P$ and $Q$, which can be expressed as a rule $P \to Q$. In FCA, we call $P \to Q$ an *implication*, and we use *basis* to represent a set of implications between attributes, from which other rules can be derived. There are many commonly used bases, such as the Guigues-Duquenne Basis [GD86], the basis of proper premises [GW99] and the Luxenburger basis [Lux91]. A basis of a context is *complete* if every implication of the context is entailed from the basis. A basis of a context is *sound* if the context satisfies all implications.

In this chapter, we focus on attribute logic first, then we introduce basis of proper premises and Luxenburger bases in detail.

## 3.1 Attribute Logic

To solve the component determination problem, we must find the dependencies between features and components first. Thus we are interested in the logic between attributes. In FCA, *attribute logic* is a method to get the rules between attributes. From a context, we can infer the rules. For example, when $A = \{a, b, c\}$ and $B = \{d, e\}$, the implication $A \to B$ expresses that every object that has attributes $\{a, b, c\}$ must also have the attributes $\{d, e\}$.

We omit the brackets if there is only one element in the set $B$, i.e. we write $A \to m$ instead of $A \to \{m\}$. We are interested in the rules with this form because in the software issue scenario we are dealing with $B$ is always a component and $A$ is a set of features.

**Definition 11.** *[GW99] An* implication between attributes *(in $M$) is a pair of subsets of the attribute set $M$. It is denoted by $X \to Y$. $X \to Y$ holds in a context if every object having all attributes in $X$ also has all attributes in $Y$. $X$ is a* premise *of $Y$ and $Y$ is* conclusion *of $X$.*

The closure can also be obtained using implications. Let $\Sigma$ be a basis with only one attribute on the head. Implication $P \to_\Sigma q$ means there is an implication $P \to q$ in basis $\Sigma$. The closure operator $\varphi_\Sigma$ (closure operator associated with $\Sigma$) can be obtained by the iteration of the following steps:

$$\varphi_\Sigma(X) = \pi_\Sigma(X) \cup \pi^2_\Sigma(X) \cup \pi^3_\Sigma(X) \cup \dots$$

where

$$\pi_\Sigma(X) = X \cup \bigcup\{q \mid P \subseteq X \text{ and } P \to_\Sigma q\}$$

and

$$\pi^2_\Sigma(X) = \pi_\Sigma(X) \cup \bigcup\{q \mid P \subseteq \pi_\Sigma(X) \text{ and } P \to_\Sigma q\}$$

This means we add everything that is implied by elements in $X$ through the implications from $\Sigma$. The computation of $\varphi_\Sigma(X)$ terminates after finitely many steps because $S$ in finite. Thus, $\varphi_\Sigma(X) = \varphi^n_\Sigma(X)$ where $n$ is the minimal integer satisfies $\varphi^n_\Sigma(X) = \varphi^{n+1}_\Sigma(X)$. The corresponding set of implications is

$$\Sigma_\varphi(X) = \{X \to y \mid y \in \varphi(X) \text{ and } X \subseteq S\}$$

A basis is *minimal* (or *non-redundant*) if for all $P \to q \in \Sigma$, $\Sigma \setminus \{P \to q\}$ is not equivalent to $\Sigma$. A basis is *direct* or *iteration-free* if for every $X \in S$, $\varphi_\Sigma(X) = \pi_\Sigma(X)$. Intuitively, a direct basis is a basis where we do not have to iterate when computing closed sets. $\varphi(X)$ is then the smallest set that contains $X$ and is closed under $\Sigma$ in the sense that whenever $(A \to b) \in \Sigma$ then $A \subseteq \Sigma(X)$ implies $b \subseteq \Sigma(X)$.

## 3.2 Proper Premises

One commonly used set of implications is often called the *stem base* or *canonical Duquenne-Guigues base*. The stem basis is sound, complete, and of minimal cardinality [GW99]. The stem base is defined using pseudo-intents.

**Definition 12.** *A set of attributes $P \subseteq M$ is a* pseudo-intent *if and only if $P$ is a set of attributes but not an intent, and $P$ contains the closure of every pseudo-intent that is properly contained in $P$. The* stem base *of a formal context $(G, M, I)$ is*

$$\{P \to P'' \mid P \textit{ pseudo-intent of } \mathbb{K}\}$$

A sound and complete set of implications can also be obtained using *proper premises*. The basis of proper premises is a canonical direct basis.

**Definition 13.** *Given a set of attributes $B \subseteq M$, we define $B^\bullet$ as the set of those attributes in $M \setminus B$, which can be implied by $B$ but not from a strict subset of $B$:*

$$B^\bullet = B'' \setminus (B \cup \bigcup S'')$$

*where $S$ is strict subset of $B$. $B$ is called a* proper premise *if $B^\bullet$ is not empty.*

$B$ is called *proper premise* for $m \in M$ if $m \in B^\bullet$. It has been shown that $\mathcal{L} = \{B \to B^\bullet \mid B \text{ proper premise}\}$ is sound and complete [Kuz04], and it has minimal cardinality among all direct bases of $\mathbb{K}$. We use the arrow relation $\swarrow$ to determine the proper premise of an attribute $m$. We write $g \swarrow m$ in the cross-table if $g'$ is maximal *w.r.t.* the subset order among all object intents which do not contain $m$.

**Proposition 2.** *$P \subseteq M$ is a proper premise for $m$ if and only if*

$$(M \setminus g') \cap P \neq \emptyset$$

*holds for all $g \in G$ with $g \swarrow m$. $P$ is minimal with respect to the property that $(M \setminus g') \cap P \neq \emptyset$ holds for all $b \in G$ with $g \swarrow m$.*

There is a connection between proper premises and minimal hypergraph transversals [RDB11]. Intuitively, a *hypergraph* is a graph where an edge can connect any number of vertices. Let $V$ be a set of vertices. A hypergraph $H$ on $V$ is a pair $(V, E)$ where $V$ is a set of vertices and $E$ is a set of non-empty subsets of $V$ called hyperedges. Therefore $E \in \mathcal{P}(V) \setminus \{\emptyset\}$ where $\mathcal{P}(V)$ is the power set of $V$. A set $S \in V$ is called *hypergraph transversal* of $E$ if $S$ intersects every hyperedge. $S$ is called a *minimal hypergraph transversal* $Tr(E)$ of $E$ if $S$ is minimal with respect to the subset order among all hypergraph transversals of $H$. The *transversal hypergraph* of $H$ is the set of all minimal hypergraph transversals of $H$, denoted by $Tr(H)$ [Hag07].

**Example 2.** *Given hypergraph $H = (V, E)$ where $V = \{v_1, v_2, v_3, v_4\}$ and $E = \{\{v_1, v_2\}, \{v_2, v_3, v_4\}\}$. $H$ has hypergraph transversals:*

$$\{\{v_2\}, \{v_1, v_2\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_2, v_3\}, \{v_2, v_4\},$$
$$\{v_1, v_2, v_3\}, \{v_1, v_2, v_4\}, \{v_2, v_3, v_4\}, \{v_1, v_2, v_3, v_4\}\}.$$

*$H$ has transversal hypergraph*

$$Tr(\{\{v_1, v_2\}, \{v_2, v_3, v_4\}\}) = \{\{v_1, v_3\}, \{v_1, v_4\}, \{v_2\}\}.$$

In the following corollary we show exactly the connection between proper premises and hypergraph transversals.

**Proposition 3.** *$P$ is a proper premise of an attribute $m \in M$ if and only if $P$ is a hypergraph transversal of $(M, H)$ where*

$$H := \{M \setminus g' \mid g \in G, g \swarrow m\}.$$

*The set of all proper premises of $m$ is the transversal hypergraph*

$$Tr(\{M \setminus g' \mid g \in G, g \swarrow m\})$$

There are two steps to compute all proper premises for one context. First, we determine arrow relations. Then, we compute the transversal hypergraph of each attribute. In Example 1, we find all arrow relations: $(2, c)$, $(2, d)$, $(3, a)$ and $(5, d)$. $(2, c)$ and $(2, d)$ are down-arrows for the following reasons: $(2, c)$ is down-arrow because $2'$ and $4'$ are in the same subset of order, and $2'$ is maximal since $2' = \{a, b\} > 4' = \{b\}$. In other words, we write down-arrow in cell $(2, c)$ because in all other rows (in this case 1) which contain the same features also feature $c$ is included. $(2, d)$ is down-arrow, because $2'$ and $4'$ are in the same subset of order, and $2'$ is maximal since $2' = \{a, b\} > 4' = \{b\}$. In other words, we write down-arrow in cell $(2, d)$ because in all other rows (in this case 1) which contain the same features also feature $d$ is included. The rest two down-arrow relations can be obtained by the same way. We show the arrow relations in Table 3.1.

We recall proposition 3 above and compute minimal hypergraph transversals of each attribute. For example, for attribute $c$, there is arrow relation

| ID | a | b | c | d |
|---|---|---|---|---|
| 1 | $\times$ | $\times$ | $\times$ | $\times$ |
| 2 | $\times$ | $\times$ | $\swarrow$ | $\swarrow$ |
| 3 | $\swarrow$ | $\times$ | $\times$ | $\times$ |
| 4 |  | $\times$ |  |  |
| 5 |  | $\times$ | $\times$ | $\swarrow$ |

Table 3.1: down-arrows in Example 1

$2 \swarrow c$. Since $\{M \setminus 2'\} \setminus \{c\} = \{d\}$ and $Tr(\{d\}) = \{\{d\}\}$, proper premise of $c$ is $d$. For other attributes, we can get the proper premises using a similar method.

When this theory is applied to the contexts of software issues, the component of an issue and the other attributes should be investigated separately, since we are interested in the implications whose premises are feature attributes and conclusions are components attributes.

Imagine we have a contexts of software issues $\mathbb{K} = (G, M_f, M_c, I)$ where $G$ is set of incidents, $M_f$ is the set of feature attributes, like "BussinessObject" and "RuntimeError". $M_c$ is the set of component attributes, which is our "goal conclusion". We also name implication in FCA "rule" in the contexts of issues. In Algorithm 1 we show how to get the rules of proper premises for all components appearing in the given context. $\mathcal{P}$ is set of proper premises for a component $m$. $R$ is a set of all rules of proper premises.

**Algorithm 1** Algorithm to compute rules for component

```
1: $\mathcal{P} = \emptyset$, $\mathcal{R} = \emptyset$
2: **for all** $m \in M_c$ **do**
3:     $\mathcal{P} = \mathrm{Tr}(\{M_f \setminus g' \mid g \in G, g \swarrow m\})$
4:     **for all** $p \in \mathcal{P}$ **do**
5:         $\mathcal{R} = \mathcal{R} \cup \{p \to m\}$
6:     **end for**
7: **end for**
8: **return** $\mathcal{R}$
```

In the algorithm the statement "$\mathcal{P} = \mathrm{Tr}(\{M_f \setminus g' \mid g \in G, g \swarrow m\})$" represents proper premises for an component $m$. For each component, the statement $\mathcal{R} = \mathcal{R} \cup \{p \to m\}$ generates all implications whose conclusion is the component.

**Example 3.** *In the cross-table $\mathbb{K}$ from Table 3 where $G = \{1, 2, 3, 4, 5, 6\}$, $M_f = \{a, b, c, d, e, f, g, h, i, j, k, l, m\}$ and $M_c = \{A, B, C, D\}$. We are going to compute the proper premises for every component.*

| ID | a | b | c | d | e | f | g | h | i | j | k | l | m | A | B | C | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | × | × |  |  |  |  | × |  |  |  |  |  | × | × |  |  |  |
| 2 | × | × |  |  |  |  | × |  |  |  |  |  | × | × |  |  |  |
| 3 |  |  |  | × |  | × |  |  | × |  | × |  |  |  | × |  |  |
| 4 |  |  |  | × |  |  |  |  |  | × | × | × |  |  |  | × |  |
| 5 |  |  |  | × |  |  |  |  |  | × | × | × |  |  |  | × |  |
| 6 |  |  | × |  | × |  |  | × |  |  | × |  |  |  |  |  | × |

Table 3.2: Context

*First, for every component $m \in M_c$, the algorithm computes the proper premises. To get proper premise, we first compute all the down arrow relations shown in Table 3.3*

| ID | a | b | c | d | e | f | g | h | i | j | k | l | m | A | B | C | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | × | × |  |  |  |  | × |  |  |  |  |  | × | × | $\swarrow$ | $\swarrow$ | $\swarrow$ |
| 2 | × | × |  |  |  |  | × |  |  |  |  |  | × | × | $\swarrow$ | $\swarrow$ | $\swarrow$ |
| 3 |  |  |  | × |  | × |  |  | × |  | × |  |  | $\swarrow$ | × | $\swarrow$ | $\swarrow$ |
| 4 |  |  |  | × |  |  |  |  |  | × | × | × |  | $\swarrow$ | $\swarrow$ | × | $\swarrow$ |
| 5 |  |  |  | × |  |  |  |  |  | × | × | × |  | $\swarrow$ | $\swarrow$ | × | $\swarrow$ |
| 6 |  |  | × |  | × |  |  | × |  |  | × |  |  | $\swarrow$ | $\swarrow$ | $\swarrow$ | × |

Table 3.3: context in Example 2 with down-arrows

*Then we compute the minimal hypergraph transversals of each component. We just show how to compute the proper premises of component A.*

$\{M_f \setminus 3'\} = \{abceghjlm\}$;

$\{M_f \setminus 4'\} = \{abcefghim\}$;

$\{M_f \setminus 5'\} = \{abcefghim\}$;

$\{M_f \setminus 6'\} = \{abdfgijlm\}$;

*According to the definition of transversal hypergraphs, set of features like $\{h, g\}$ is a proper premise, but actually no record of these six incidents has the feature combination of $\{h, g\}$. We are only interested in those combinations which appear in at least on record. One of the transversal hypergraph we can use as proper premise of component A is: $\{a, b, g, m\}$.*

*The rules we generate for component $A$ are $\{a \to A, b \to A, g \to A, m \to A\}$. Similarly, for component $B, C, D$ we can get rules of proper premises. Finally we get the set of rules: $\mathcal{R} = \{a \to A, b \to A, g \to A, m \to A, f \to B, l \to B, j \to C, l \to C, c \to D, h \to D, e \to D\}$.*

## 3.3 Luxenburger Basis

When the processor is responsible for a group of components including components $X$ and $Y$, he or she could just close the incident in either component by chance. That is one reason why the data in historical records are not precise, and that is where the erroneous data come from.

**Example 4.** *In the context of historical records shown in Table 3.4, there are five incidents. $a$ and $b$ are features and $X$ $Y$ are components.*

We can see that the first three incidents in Table 3.4 show the component $X$ is responsible for incident with features $\{a, b\}$, but component $Y$ is responsible for incident 4 with features $ab$. Incident 4 might be an error data, or some feature rather than $a$ or $b$ is missing in the context to distinguish $X$ and $Y$. From the previous section, both the proper premises of $X$ and $Y$ are $\emptyset$, which means, $\{a, b\}$ cannot be proper premise for either component. But actually, from the probability point of view, component $X$ is responsible for incident with feature $\{a, b\}$ and for the rare case, component $Y$ is. Hence the approach of proper premises is not suitable to deal with erroneous data. In other words, the approach of proper premises is not error tolerant.

| ID | a | b | $X$ | $Y$ |
|---|---|---|---|---|
| 1 | × | × | × | |
| 2 | × | × | × | |
| 3 | × | × | × | |
| 4 | × | × | | × |
| 5 | × | | × | |

Table 3.4: Context with error data

In this applied field we want to deal with errors and missing information, hence we are interested in rules that can handle imprecisions. The errors, also called counterexamples, are taken into account in the following chapter. The interest of imprecise data motivates us to investigate Luxenburger Basis.

**Definition 14.** *The* support *of an implication is the proportion of all objects having all attributes of the implication in both premise and conclusion among all objects.*

$$supp(X) = \frac{|X|}{|G|} \quad supp(X \to Y) = supp(X \cup Y)$$

*The* confidence *of an implication is the proportion of all objects satisfying both premise and conclusion among all objects satisfying premise.*

$$conf(X \to Y) = \frac{supp(X \cup Y)}{supp(X)}$$

In Example 4, $supp(\{a,b\}) = 4/5, supp(\{a,b\} \cup \{X\}\}) = 3/5, supp(\{a,b\} \cup \{Y\}) = 1/5$ and $conf(\{a,b\} \to \{X\}) = 3/4, conf(\{a,b\} \to \{Y\}) = 1/4$. Therefore The strength of a rule can be defined in terms of its support and confidence and the strengths of rules can be used to rank candidate components.

**Definition 15.** *Let $I \subseteq M$, and let* minsupp, minconf $\in [0,1]$*. The support count of the itemset $I$ in $\mathbb{K}$ is $supp(I) := \frac{|g(I)|}{G}$ where $g(I)$ is set of objects which has relation with all attributes in $I$. $I$ is said to be* frequent itemset*(FI) if $supp(I) \geq minsupp$. The set of all frequent itemsets of a context is denoted by FI.*

In Example 4,minimal support is 0.5, $\{a,b,X\}$ is a frequent itemset, but $\{a,b,Y\}$ is not. All subsets of a frequent itemset are also frequent itemsets. In Figure 3.1 all frequent itemsets are marked dark.

An *association rule* is an implication expression of the form $X \to Y$, where $X$ and $Y$ are disjoint itemsets, i.e., $X \cap Y = \emptyset$. The main task in association rules mining is rule generation, which aims at extracting all implications whose confidence is above the minimal confidence threshold. These rules are called strong rules [TSK05]. We will not investigate further on association rules since there are too much useless rules, and it is very time-consuming to obtain all association rules and then remove those useless ones. Therefore, we introduce a commonly used set of imprecise implications—Luxenbureger Basis.

In [Lux91], Luxenburger introduced basis for *partial implications.* A partial implication is an implication where accuracy (i.e. confidence) is considered. It is sufficient to only consider the implications between concept intents, which are the closed sets because conf$(X \to Y) =$ conf $(\varphi(X) \to \varphi(Y))$ [LS05].

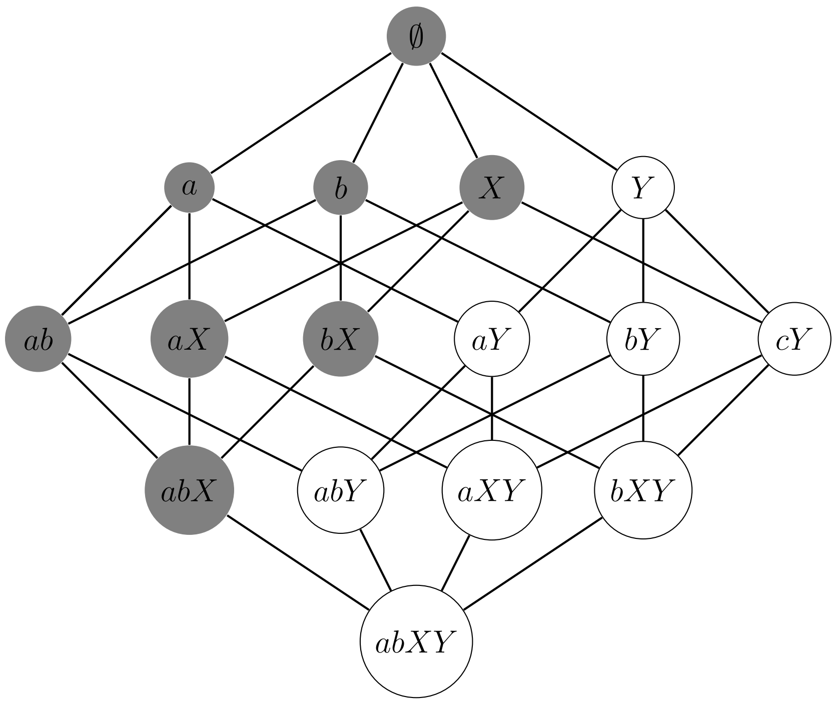

Figure 3.1: Frequent itemsets in context in Example 4

**Definition 16.** *An itemset $X$ is* closed itemset *if $X$ is a closed set and none of its immediate supersets has exactly the same support count as $X$. An itemset $X$ is called* frequent closed itemset*(FC) if it is closed and its support is above the minimal support threshold.*

The closed itemsets from the context of Example 4 are shown in the lattice in Figure 3.2. When minimal support is 0.5, $\emptyset$, $ab$ and $abX$ are frequent closed itemsets, which are marked dark in Figure 3.2.

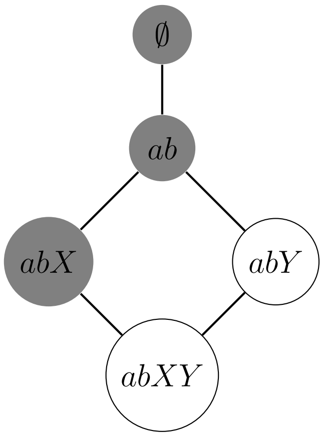

Figure 3.2: Closed itemsets and frequent closed itemsets

**Definition 17.** *The* Luxenburger Basis *is defined as $LB := \{(I_1 \to I_2$, support(r), confidence(r)) where $I_1$ and $I_2$ are closed itemsets and $I_1$ is the direct upper neighbor of $I_2$ in the concept lattice, and confidence(r)$\geq$ minconf, support($I_2$) $\geq$ minsupp$\}$*

**Example 5.** *The context $\mathbb{K} = \{G, M, I\}$ is shown in Table 3.5 where $G = \{1, 2, 3, 4, 5\}, M = \{a, b, c, D, E\}$. $a, b, c$ are features and $D, E$ are components. Given minsupp=minconf=0, the Luxenburger Basis of $\mathbb{K}$ is $\{\emptyset \to c, \emptyset \to bE, bE \to c, c \to ab, c \to bE, acD \to bE, bcE \to a, abcD \to E\}$.*

The Luxenburger Basis consists of association rules such that there are concepts $(A_1, B_1)$ and $(A_2, B_2)$ where $(A_1, B_1)$ is a direct upper neighbour of $(A_2, B_2)$, $X = B_1$ and $X \cup Y = B_2$. In Figure 3.5 we show Luxenburger Basis of Example 5.

There has been extensive research on the relation between Luxenburger basis and association rules. One popular conclusion is that all association rules can be derived from two bases: Duquenne-Guigues Basis [GW99] for exact association rules and the Luxenburger basis for approximate association

| ID | a | b | c | $D$ | $E$ |
|---|---|---|---|---|---|
| 1 | × | | × | × | |
| 2 | | × | × | | × |
| 3 | × | × | × | | × |
| 4 | | × | | | × |
| 5 | × | × | × | | × |

Table 3.5: Lattice of Luxenburger Basis

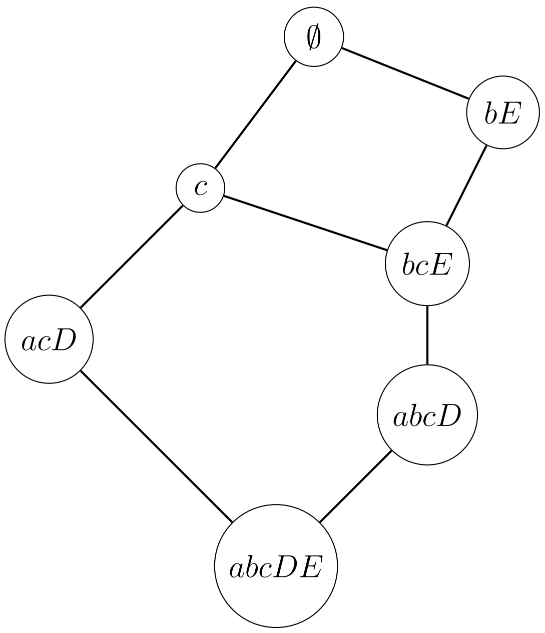

Figure 3.3: Lattice of Example 5

rules [STB$^+$01]. We have introduced basis of proper premises and Luxenburger Basis. These two bases can provide rules between attributes. We are going to show how we can use the existing bases are applied to software issues in the following chapter.

# Chapter 4

# Algorithms for Error-tolerant Bases

From a mathematical logic point of view the newly created issue (or incident) is an *observation*. An observation in this field of application is a set of features without component information. To determine the right component for a newly created issue, we need the rules generated from the historical incidents which have been completely processed and closed in the right components.

Our goal is to generate a basis from the given contexts of historical records. This basis consists of implications with the form of $X \to y$ where $X$ is the set of feature attributes and $y$ the implied component. Since the implications in the basis should work for data with errors or incomplete data, we call such basis that is error tolerant (ET Basis). With this base, we can predict the right component when we know the features of an incident. In this chapter, we first present the approach of Luxenburger Basis since it can deal with erroneous data and we try to use closure system to find all candidate components. Second, we partition the context and apply the idea of frequent itemsets to each small context with same component. Third we show another context partitioning approach while applying proper premises, which shows promising results in the experiments. In the end, we show an extreme way of applying the third approach which corresponds to machine learning theory.

## 4.1 Closure Computation

In Section 3.3, we introduced Luxenburger Basis, which can deal with erroneous data. Since Luxenburger basis can generate imprecise implications, we try to apply idea of Luxenburger Basis to see if we can get proper premises of each component with error tolerance.

**Example 6.** *The context in Table 4.1 contains four historical issues where* $a, b, c$ *are features and* $X, Y$ *are components. The corresponding Luxenburger Basis of this context with minsupp=minconf=0 is shown in Table 4.2.*

| ID | a | b | c | $X$ | $Y$ |
|---|---|---|---|---|---|
| 1 | × | × | | × | |
| 2 | × | × | | | × |
| 3 | | | × | × | |
| 4 | × | × | × | × | |

Table 4.1: context of historical issue

| implication | support | confidence |
|---|---|---|
| $\emptyset \to X$ | 0.75 | 0.75 |
| $\emptyset \to ab$ | 0.75 | 0.75 |
| $X \to c$ | 0.5 | 0.67 |
| $ab \to X$ | 0.5 | 0.67 |
| $X \to ab$ | 0.5 | 0.67 |
| $cX \to ab$ | 0.25 | 0.5 |
| $abX \to c$ | 0.25 | 0.5 |
| $ab \to Y$ | 0.25 | 0.33 |
| $abcX \to Y$ | 0 | 0 |
| $abY \to cX$ | 0 | 0 |

Table 4.2: implications in Luxenburger Basis

We list all implications with minsupp = 0 and minconf = 0 in Table 4.2. The ideal implications we want to generate are of the following format *features* $\to$ *component*, but from Table 4.2 we can see that the premises of implications are not always features and the conclusions are not always components. Implications like $abX \to c$ or $abY \to cX$ cannot tell us explicitly the relation between the features and one component. It is not easy to directly apply Luxenburger Basis to determine the right component for an issue since we cannot produce rules with component on the conclusion side. Therefore we need to make an extension of Luxenburger Basis.

Let us jump back to Example 6. Suppose the $bc$ is an observation, and there is no implication in Table 4.2 which can directly be used to determine the right component. But we notice in the concept lattice of Luxenburger Basis shown in Figure 4.1 that the “candidate” components $X$ and $Y$ (the components which might be the right components) can be found in closure like $abcX$. Therefore to find the right component for an issue, the first step is to find candidate components, and then we rank all candidates and see if the top one is the right one.

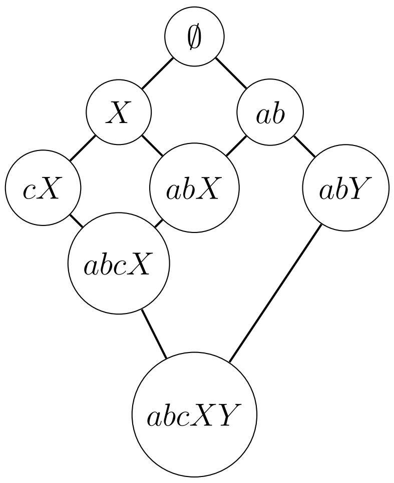

Figure 4.1: Concept Lattice

From Section 2.3, the closure of $S$ contains all attributes that can be implied by $S$ under a set of implications. If some component is in the closure of $S$, then this component can be implied by $S$. We aim to find the smallest closed superset of the observation and the components in the set are the candidate components. Rules of Luxenburger Basis can be visualized as “edges” of a concept lattice. To find the smallest closed superset of observation, we have to include in the Luxenburger basis a base that can represent implications in each “node‘” in the concept lattice, like stem base. In Section 3.2, we shortly introduced the stem basis, which is constructed using pseudo-intent. One node in the concept lattice can represnet an intent. Thus stem base can be used to construct the lattice so that we can find those components in the intents. Therefore Luxenburger basis consists of all imprecise implications, and stem base consists of all valid (or precise) implications. Thus we can find candidate components of observation $S$ from the smallest closed superset of $S$ under the union of Luxenburger Basis and stem base.

After obtaining all candidate components $C_1, C_2, ..., C_n$, we can construct the an ET Base of $S$. For every combination of features $F_x \subseteq S$ (i.e. every non-empty element $F_x$ in the power set of observation) where $1 \leq x < 2^{|S|}$, we construct a set of implications $\Sigma_{F_x}$ for every candidate component where the premise is $F_x$:

$$\Sigma_{F_x} = \{F_x \to C_1, F_x \to C_2, ..., F_x \to C_n\}.$$

The ET Base of an observation $S$ is the union of all $\Sigma_{F_x}$ :

$$\Sigma_S = \{ \begin{array}{l} F_1 \to C_1, F_1 \to C_2, ..., F_x \to C_n, \\ F_2 \to C_1, F_2 \to C_2, ..., F_2 \to C_n, \\ ..., \\ F_m \to C_1, F_m \to C_2, ..., F_m \to C_n\} \end{array} \quad \text{where } m = 2^{|S|} - 1$$

In Example 5, the smallest superset of $\{b, c\}$ under the union bases is $\{a, b, c, X\}$ given minimal support 0.2 and minimal confidence 0.2. The ET Base of $bc$ is: $\{b \to X, c \to X, bc \to X\}$. The candidate component in this case is $X$. In Algorithm 2 we show how to get an ET Basis of an observation $S$. Lines 1 and 2 denote computing stem base and Luxenburger Basis respectively from the given context $\mathbb{K}$. Lines 3 is to make union of these two bases. Line 4 shows how to get all candidate components are those appearing in the smallest closed superset of observation S. Line 5 initialize a ET Basis and Lines 6 to 10 is to constructing rules for each component c, and for every c, the premises are elements from the power set of observation S. Each rule is associated with the corresponding degree of recommendation.

**Algorithm 2** construct an ET Basis using closure

```
Input: $\mathbb{K} = \{G, M_f, M_c, I\}$, observation $S$
 1: SB = $\{r \mid r \in$ stem base $\}$
 2: LB = $\{r \mid r \in$ Luxenburger basis$\}$
 3: RULE = SB $\cup$ LB
 4: CC = $\{m \mid m \in M_c \bigcap$ smallest closed superset ( RULE, $S$)$\}$
 5: DR = $\emptyset$
 6: for all $c \in CC$ do
 7:     for all $s \subseteq S$ do
 8:         $DR = DR \bigcup \{s \to c$, dr $(s \to c, \mathbb{K})\}$
 9:     end for
10: end for
11: return $DR$
```

When the amount of historical records is growing, the candidate components we get in the closure will be more than one, it is important to rank these components to see which one has the highest rank. One way to compute the degree of recommendation (dr) is using confidence.

**Algorithm 3** Algorithm to get top ranking components

```
Input: DR
1: maxconf = $\{dr \mid dr$ is maximal among all confidence $\in DR\}$
2: TC = $\{c \mid c \in M_c \bigcap s \rightarrow c \in DR \bigcap$ confidence$(s \rightarrow c, \mathbb{K})$ = maxconf$\}$
3: return $TC$
```

$$\text{dr}(F_x \rightarrow C) = \text{conf}(F_x \rightarrow C)$$

Component of the implication with highest confidence is the most recommended component for the observation $S$. The component with the highest degree of recommendation can be obtained by Algorithm 3. Line 1 is to get the maximal degree of recommendation (maxconf) from all rules in ET basis constructed in Algorithm 2. Line 2 shows the top components are those with the same degree of recommendation as the maximal degree, and the component c in TC can be varified if it is the right component.

The advantage of this approach is that we can use imprecise implications of Luxenburger basis to deal with erroneous data. However, there are also disadvantages. Firstly, construction of lattice from Luxenburger and canonical bases is quite time-consuming. Secondly, it is complex to compute the smallest closed superset of the observation. Thirdly it is not easy to find the suitable minsupp and minconf: if we set a low minimal value, the computation of implications are very expensive and there will be a lot of candidate components, most of which are wrong components; if we set a high minimal value, there will be very few candidate components in the closure, thereby the closure containing the right component might be eliminated.

## 4.2 Frequent Premises in Partitioned Context

Since applying the idea of Luxenburger Basis is computationally very expensive, we present a new approach partitioning the original context into several smaller ones with the same component. Then we apply the idea of frequent itemsets from Luxenburger Basis to get relevant sets of features for each small context. One difference between this approach and closure computation is in frequent premises, the ET Basis we get first from the historical data is not related with certain observation. Then we extract those rules related with certain observation and call the basis of extracted rules the *ET Basis of observation.*

The *subcontext* $\mathbb{K}_A$ is a context containing all incidents with the same component "A" from the original context $\mathbb{K}$. For example, there are three subcontexts $\mathbb{K}_A, \mathbb{K}_B, \mathbb{K}_C$ if there are three components $A, B, C$ in the original context. We use the term *size* of subcontext for the amount of objects in each subcontext.

A *Frequent premise* of component $A$ is a frequent itemset (i.e. set of features whose support is greater than or equal to minimal support) in the subcontext $\mathbb{K}_A$. Given a minimal support, for each subcontext $\mathbb{K}_X$ we can obtain an ET basis of implications, where the premises of implications are frequent itemsets and the conclusion $X$. The features in frequent itemsets are also strong related features for component $X$.

**Example 7.** *Let $\mathbb{K}_A$ be a subcontext of component A shown in Table 4.3 where $M_{\mathbb{K}_A} = \{a, b, A\}$. If minsupp $= 0.5$, $\{b\}$ is the only frequent premise of A. The ET Basis of $\mathbb{K}_A$ is $\{b \to A\}$. If minsupp $= 0.4$, $\{a\}$ $\{b\}$ $\{a, b\}$ are the frequent premises for A. The ET Basis of $K_A$ is $\{a \to A, b \to A, ab \to A\}$.*

In Example 7 we show how to get an ET Basis of a subcontext. In Algorithm 4 we show how to get an ET Basis from a given context. The input is the context of historical records $\mathbb{K}$. Line 1 is initializing the ET Basis. Line 2 to 7 shows for every component c in historical data we to construct an ET Basis of component c, then the union of these bases from every component is the final ET Basis. The ET Basis of an observation $S$ is a set of implications whose premises are contained by $S$. The candidate components of $S$ are those appearing in the ET Basis of $S$.

| ID | a | b | A |
|---|---|---|---|
| 1 | × | × | × |
| 2 | × | × | × |
| 3 |  | × | × |
| 4 |  | × | × |
| 5 |  | × | × |

Table 4.3: subcontext

**Algorithm 4** construct an ET Basis using frequent premises

```
Input: $\mathbb{K} = \{G, M_f, M_c, I\}$, minsupp
1: $\mathcal{R} = \emptyset$
2: for all $c \in M_c$ do
3:     $\mathbb{K}_c = \{G_c, M_f, I_c\}$
4:     $FP = \{FI \mid \text{supp(FI)} \geq \text{minsupp}\}$
5:     $ET_c = \{fp \to c \mid fp \in FP\}$
6:     $\mathcal{R} = \mathcal{R} \cup ET_c$
7: end for
8: return $\mathcal{R}$
```

One approach to define ranking of recommended components is using the support of frequent premises in the subcontexts, together with the "voting weight" of each subcontexts. The voting weight is the size of the subcontext divided by the size of the original context.

$$\text{dr}(FP \to C)\text{=supp}(FP)_{\mathbb{K}_C}\text{*weight}(\mathbb{K}_C)\text{=supp}(FP)_{\mathbb{K}_C} * \frac{size(\mathbb{K}_C)}{size(\mathbb{K})}$$

**Example 8.** *In context $\mathbb{K}$ there are 11 incidents separated in two subcontexts $\mathbb{K}_B$ and $\mathbb{K}_C$ shown in Table 4.4. The minimal support is $0.5$ and observation is ac. In $\mathbb{K}_B$, the ET Basis of ac is $\{a \to B\}$ with support$(\{a\})_{\mathbb{K}_B} = 1$ and weight$(\mathbb{K}_B) = 1/11$. In $\mathbb{K}_C$, the ET Basis is $\{a \to C\}$ with support$(\{a\})_{\mathbb{K}_C} = 1/2$ and weight$(\mathbb{K}_C) = 10/11$. The degree of recommendation of B is*

*support$(\{a\})_{\mathbb{K}_B}*$ weight$(\mathbb{K}_B) = 1/11$, and the degree of recommendation of C is support$(\{a\})_{\mathbb{K}_C}*$ weight$(\mathbb{K}_C) = 10/22$.*

| | a | B |
|---|---|---|
| 1 | × | × |

| | a | b | C |
|---|---|---|---|
| 1 | × | | × |
| 2 | × | | × |
| 3 | × | | × |
| 4 | × | | × |
| 5 | × | | × |
| 6 | | × | × |
| 7 | | × | × |
| 8 | | × | × |
| 9 | | × | × |
| 10 | | × | × |

Table 4.4: subcontexts $\mathbb{K}_B$ and $\mathbb{K}_C$

The main advantage of this approach is subcontexts do not lose any information from the original context and each subcontext is independent hence it is easy to update the corresponding subcontexts if new fully processed incidents are put into historical records. In addition, there is no need to go through the whole context to generate an ET Basis hence it is much faster to generate an ET Basis compared with the previous closure approach.

There is also disadvantage. The way we define rate of recommendation still needs to be improved. Because some frequent itemset has appeared often in every subcontext. The rules containing such frequent itemsets may recommend several components with same or similar degree, thus it is hard to find the right one among them.

## 4.3 Proper Premises in Partitioned Context

The experiments in Section 5.2.2 show that the approach of applying frequent premises in partitioned context is not precise enough to determine the right component. Therefore we try to partition the context in a different way. As shown in Section 2.2, the big advantage of basis of proper premises is that the rules in the basis are very precise and can be used directly to determine the right component. We want to keep this advantage hence we now take these ideas from proper premises while trying to construct an ET basis.

The disadvantage of proper premises is that some suitable premises cannot "survive" because of erroneous data. However, the new approach presented in this section can minimize the effect of erroneous data to the size of the subcontext. We partition the original context and compute proper premises in each subcontext. It is shown in the experiments that this approach is suitable to determine right component.

We divide the original training contexts into subcontexts randomly and equally but not necessarily containing the same component. Then we generate the basis of proper premises for each of these subcontexts. The union of all bases of subcontexs is an ET Basis of the whole context.

*Equally partitioning of the context* with size $n$ is to divide a context with $n$ objects into same size subcontexts where there is no common object in any two subcontexts and every object must appear in one of the subcontexts. But there might be a case that the size $m$ of the original context is not a multiple of $n$, thus the size of one subcontext is remainder when $m$ is divided by $n$. In Algorithm 5 we show how to generate an ET Basis using proper premises in the partitioned context.

**Example 9.** *Given context $\mathbb{K} = (G, M, I)$ shown in Table 4.5, $\mathbb{K}$ is equally partitioned. $\mathbb{K}_1$ and $\mathbb{K}_2$ are subcontexts shown in Table 4.6. The basis of proper premises for $\mathbb{K}_1$ is $\{ac \to X, b \to Y\}$, and the basis of proper premises for $\mathbb{K}_2$ is $\{a \to X\}$. The union of bases of proper premises for every subcontext is the ET Basis: $\{ac \to X, b \to Y, a \to X\}$. If we do not partition context, the basis of proper premises of the original context $\mathbb{K}$ in Table 4.5 is $\{ac \to X, a \to X\}$, where the rule for component $Y$ cannot exist.*

| $\mathbb{K}$ | a | b | c | X | Y |
|---|---|---|---|---|---|
| 1 | × | | × | × | |
| 2 | | × | | | × |
| 3 | × | × | | × | |
| 4 | | × | | | × |

Table 4.5: Context

| $\mathbb{K}_1$ | a | b | c | X | Y |
|---|---|---|---|---|---|
| 1 | × | | × | × | |
| 2 | | × | | | × |

| $\mathbb{K}_2$ | a | b | c | X | Y |
|---|---|---|---|---|---|
| 3 | × | × | | × | |
| 4 | | × | | | × |

Table 4.6: Partitioned context

The ET Basis of an observation $S$ is set of implications whose premise is contained in observation. The candidate components of observation $S$ are those appearing in the ET Basis of an observation $S$. One way to define the ranking of candidate components is using the confidence of rules in an ET Basis of an observation $S$ among the original context.

$$\text{dr}(pp \to C)\text{=confidence}_{\mathbb{K}}(pp \to C)$$

Let us go back to Example 9. The ET Basis of *abc* is $\{ac \to X, b \to Y, a \to X\}$. The candidate components are $X, Y$. The degrees of recommendation of $X$ and $Y$ are 1 and 0.67 respectively.

**Algorithm 5** construct an ET Basis using proper premises

```
Input: $\mathbb{K} = \{G, M_f, M_c, I\}$, size $n$
 1: $\mathcal{S} = \{s = \{G_s, M_s, I_s\} \mid s \in \text{partition}(\mathbb{K}, n)\}$
 2: $\mathcal{R} = \emptyset$
 3: for all $s = \{(G_s, M_s, I_s)\} \in \mathcal{S}$ do
 4:   $\mathcal{C} = \{c \mid c \in M_s \bigcap M_c\}$
 5:   for all $c \in \mathcal{C}$ do
 6:     $\mathcal{P}_c = \{p \mid p \in \text{proper premises of } c\}$
 7:     if $\mathcal{P}_c \neq \emptyset$ then
 8:       $\mathcal{R}_c = \emptyset$
 9:       for all $p \in \mathcal{P}_c$ do
10:         if support$(p) \neq 0$ then
11:           $\mathcal{R}_c = \mathcal{R}_c \bigcup \{(p \to c,\ (\text{rd } (p \to c), \mathbb{K}_{train}))\}$
12:         end if
13:       end for
14:     end if
15:     $\mathcal{R} = \mathcal{R} \bigcup \mathcal{R}_c$
16:   end for
17: end for
18: return $\mathcal{R}$
```

In Algorithm 5, the input we give are the context $\mathbb{K}$ of historical data and the size n of subcontext. The code in Line 1 shows the set S is the set of all subcontexts. Each subcontext is from partitioned context with size n. Line 2 is to initializing an ET Basis. Lines 3 to 17 shows for every subcontext how we get the ET Basis $\mathcal{R}_c$. Line 4 is collecting all components c appearing in the subcontexts. For each c, Line 6 shows the algorithm computes all proper premises of c. Lines 7 to 14 shows for every proper premise whose support is greater than 0, the algorithm construct a rule of proper premise associated with the support of this rule. In Algorithm 6 we show how to generate an ET Basis of an observation. The input id the ET Basis we get from Algorithm 5 and an observation S. Line 1 is initializing a basis. Line 3 is checking for every rule in the ET Basis if the premise in the rule is contained in the observation S. If it is contained, we collect this rule to the ET Basis of observation S.

**Algorithm 6** Algorithm to generate an ET Basis of one observation

```
Input: $\mathcal{R}$ = proper premises rules, observation $S$
1: $\mathcal{DR} = \emptyset$
2: for all $P \to c \in \mathcal{DR}$ do
3:     if $P \subseteq S$ then
4:         $\mathcal{DR} = \mathcal{DR} \cup \{(P \to c, \mathrm{rd}(P \to c))\}$
5:     end if
6: end for
7: return $\mathcal{DR}$
```

We show the experiments to evaluate this approach in Section 5.2. Firstly, it keeps the advantage of proper premise to get those precise rules from subcontexts and it minimizes the effects of erroneous data. Secondly it also keeps the advantage of partitioned context—very easy to update an ET Basis.

## 4.4 Subset Searching

The extreme case of partitioning context is setting the size of each subcontext as one. This means that each record represents one rule in the ET Basis. The ET Basis of an observation consists of all implications whose premises are contained by the observation. Therefore, this approach is corresponding to machine learning. A study on bug assignment using machine learning approach is shown in [ORS06] where they reach precision level of 57% and 64% on the Eclipse and Firefox development projects respectively.

**Definition 18.** The applicable Object *for an observation $S$, denoted as $o_S$, is the object whose feature-attributes are contained in observation $S$. The set $A^S$ is the set of all applicable objects of observation $S$. The* Matched Object *of component $c$ for an observation $S$, is the object whose component-attribute is component $c$ and feature-attributes $F_{o_S}$ are contained in observation $S$, which means $F_{o_S} \subseteq S$. The set $B^S(c)$ is a set of all matched objects for component $c$ of observation $S$.*

In this approach, we define the degree of recommendation using statistics on the whole context. The degree of recommendation of component $c$ is the proportion of matched records in whole records.

$$dr^S_{\mathbb{K}}(c) = \frac{|B^S(c)|}{|A^S|}$$

Given a context of historical records and an observation, In Algorithm 7 we show all candidate components with their recommended degrees.

**Algorithm 7** finding candidate components with degree of recommendation for observation

```
Input: $\mathbb{K}$, $S$
  $\mathcal{A}^{\mathcal{S}} = \emptyset$, $\mathcal{PC} = \emptyset$
  for all $o \in G$ do
    if $F_{o_S} \subseteq S$ then
      $\mathcal{A}^{\mathcal{S}} = \mathcal{A}^{\mathcal{S}} \cup \{o\}$
      $\mathcal{PC} = \mathcal{PC} \cup \{c_{o_S}\}$
      $count(c) = count(c) + 1$
    end if
  end for
  for all $c \in PC$ do
    $dr^S_{\mathbb{K}}(c) = \frac{count(c)}{count(A^S)}$
    $\mathcal{C} = \mathcal{C} \cup \{(c, r^S_{\mathbb{K}}(c))\}$
  end for
  return $\mathcal{C}$
```

**Example 10.** *The context $\mathbb{K} = (G, M, I)$ shown in Table 4.7 where $G = \{1, 2, 3, 4, 5\}$ and $M = \{a, b, c, d\}$. The applicable objects of observation $\{a, b, c\}$are $\{1, 2, 3, 4\}$. $dr(X) = 0.25, dr(Y) = 0.5$ and $dr(Z) = 0.25$.*

| ID | a | b | c | d | X | Y | Z |
|---|---|---|---|---|---|---|---|
| 1 | × | × |  |  | × |  |  |
| 2 | × | × |  |  |  | × |  |
| 3 |  |  | × |  |  | × |  |
| 4 | × | × | × |  |  |  | × |
| 5 |  |  | × | × |  |  | × |

Table 4.7: Context of subset searching

Compared with the previous approaches, there is no need to train the context with historical records using subset searching. The way to recommend component is using the idea of probability. Therefore it is easy to process each new incident in the computation point of view. But, when there are large data in the historical records, it also takes a longer time to go through the whole training records. Another disadvantage is for some component, there is strong related attributes but occurring very rare. Hence from probabilistic statistic, the right component will not be recommended with high degree of recommendation.

# Chapter 5

# Evaluation

In this chapter, we present experiments in every approach we showed in Chapter 4 to evaluate the different approaches on several sets of training data and test cases. We will compare the advantages and disadvantages of the approaches we evaluated.

We separate the experiments into two phases: training and predicting. In the training phase, the original many valued context is transformed into one value context, then an ET basis is generated from the training context. In the approach of subset searching, there is no need to generate an ET Basis first, since we process the whole training context and collect all candidate components for the observation. In the predicting phase, we first generate an ET Basis of the observation, which contains all predicting rules from an ET Basis we got in the training phase. Then we get all candidate components with ranking.

The given contexts from which we can generate an ET Basis are called *training data*. The incidents which need to be predicted (i.e. the responsible components should be found) are called *testing data*. We use part of the whole historical records as experiment data. We separate experiment data into training data and testing data. For the testing data actually we know the correct components but these components are not involved in either training or predicting phase. We use these correct components only for verifying whether the recommended components are right.

To evaluate the different approaches, we use several criteria: training time, predicting time, rate of positive classification and rate of correct classification. *Training time* is the time which was spent on generating the ET Basis from training data. *Predicting time* is the time which was spent on predicting the right component for one incident. An incident is *positively classified* if the correct component of the test incident is among the candidate components. An incident is *correctly classified* if the correct component ranks on the top one among the recommended components. We use # to indicate the amount of incidents, e.g. # Train means the amount of test incidents. T(train) means the training time. The *rate of positive classification* is the proportion of the amount of positively classified incidents among the whole test data. Analogously, the *rate of correct classification* is the proportion of the amount of correctly classified incidents among the whole test data.

We use several test series to evaluate every approach, which will be listed in tables and figures in the following sections. Each incident we use to train or test has six features and one component. The values we write in the examples are not real data from customer.

## 5.1 Closure Computation

In Section 4.1, we introduced the approach based on Luxenburger Basis. In this section, we present the experiments of this approach. Table 5.1 and Figure 5.1 show the training time to generate an ET Basis which is the union of Luxenburger Basis and Guiques-Duquenne Basis. It costs 0.7, 99, 1882 and 28800 seconds to train 10, 50, 100 and 250 incidents respectively. The curve in Figure 5.1 shows the time is increasing super linearly. We use 20 incidents as test data and the average predicting time is 1.3, 2.8, 3, 3.5 milliseconds respectively.

| # Train | 10 | 50 | 100 | 250 |
|---|---|---|---|---|
| T(train) (s) | 0.7 | 99 | 1882 | 28800 |
| T(predict) (ms) | 1.3 | 3 | 2.8 | 3.5 |

Table 5.1: Training time in the approach of closure computation

The rates of positively and correctly classifications are shown in Figure 5.2. According to Figure 5.2, the more training data we use, the higher rates of classifications we get, although we only have small training data in the experiments. We set the minimal support is greater than 0, which means we just ignore those feature sets whose support values are 0. We set the minimal confidence 0.5 because we want to get those rules with candidate components which have more than 50

The main reason for negatively classification is that the training data is not large enough and the right components never appear in it. Although we want to add more training data to see the results, from Figure 5.1, it costs more than 8 hours to train 250 incidents, and we can imagine with this

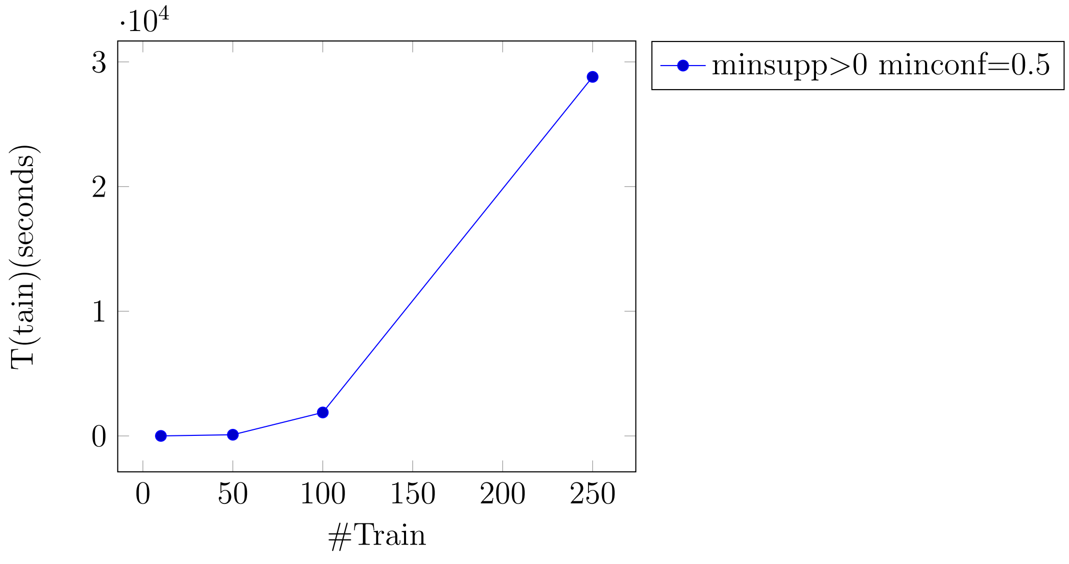

Figure 5.1: training time

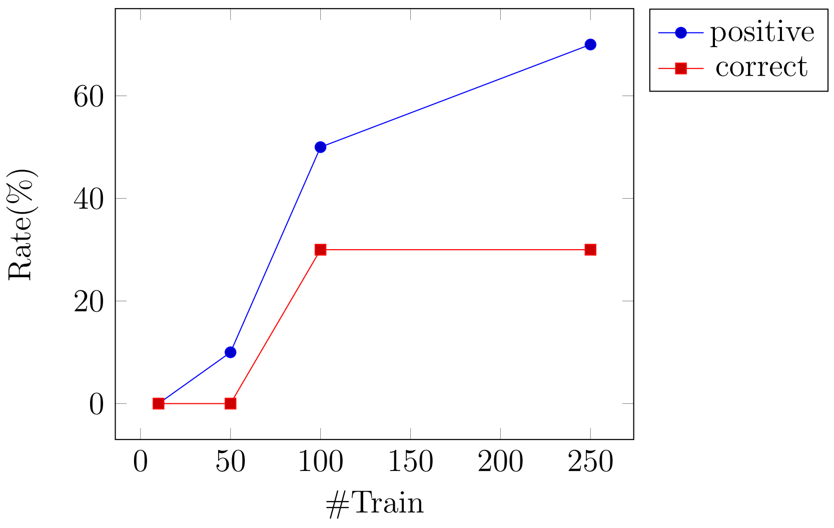

Figure 5.2: Rates in the approach of closure computation

minsupp and minconf it will cost much more time to train on larger data. Hence we stop investigating this approach.

## 5.2 Frequent Premises in Partitioned Context

In Section 4.2, we introduced the approach of frequent premises. In the experiment part, we use training data with 100, 250, 500, 1000 and 2000 incidents since the training time is much less than the previous approach. It only takes 2164 seconds to get an ET Basis from the training data with 2000 incidents. We use the same 20 test incidents as in the previous approach. And the predicting time is also increasing along with the number of training data. We show the training and predicting time in Table 5.2, Figure 5.3 and Figure 5.4.

| # Train | 100 | 250 | 500 | 1000 | 2000 |
|---|---|---|---|---|---|
| T(train) (s) | 5.4 | 41 | 217 | 899 | 2164 |
| T(predict) (ms) | 0.5 | 1.3 | 1.6 | 2.9 | 5.5 |

Table 5.2: Training and predicting time in the frequent premises approach

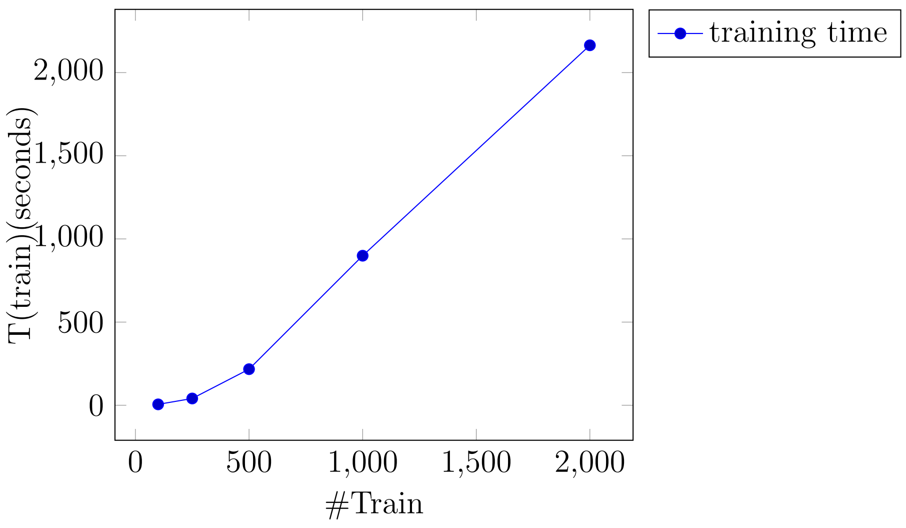

Figure 5.3: Training time in the approach of frequent premise

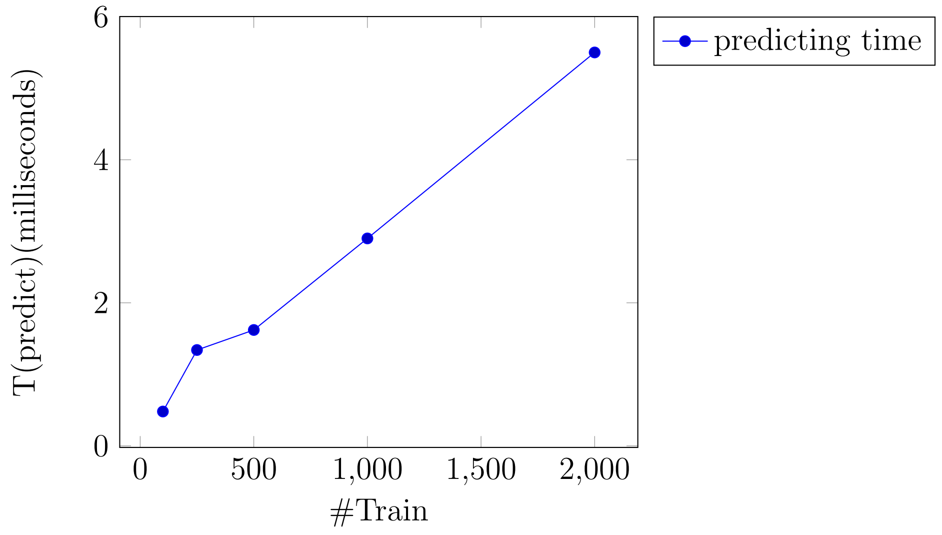

Figure 5.4: Predicting time in the approach of frequent premises

We show the rates of correct and positive classification in Figure 5.5 in the approach of frequent premises, where we can see we can see the rate of positive classification is above 80% but the rate of correct classification is 0, which indicates that the right component is among the candidates but the recommended degree of the right component is not high enough. The reason for the low rate of correct classification is that in the ET Basis, there is always some rules ranked top because one feature in these top rules appearing quite often in whole training data, thus rules containing such feature have very high support even if weight is added. Such rules always rank top but cannot predict the right components. To improve the rate of correct classification, we have to evaluate another approach.

## 5.3 Proper Premises in Partitioned Context

This section is related with the approach described in Section 4.3. In Table 5.3, we show a test incident. We list the first part of the rules in the ET Basis of training data on 2000 incidents and candidate components in Table 5.4. The rule with highest degree of recommendation is "paymentorder $\rightarrow$ Component- MOP", therefore component "MOP" is on the first rank.

We did experiments on training data with 100, 250, 400, 500, 700, 1000, 1500, 2000, 3000, and 4000 incidents. The subsumption relations between these data sets (ds) are:

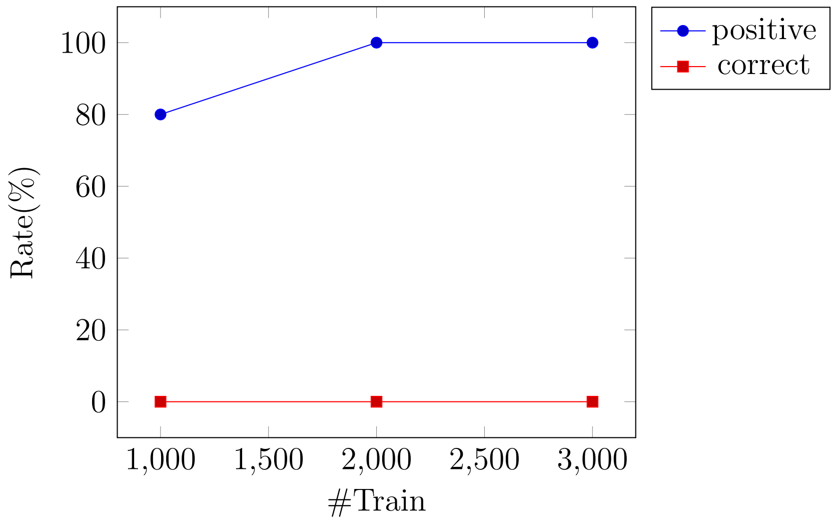

Figure 5.5: Rates in the approach of frequent premises

| feature | value |
|---|---|
| BusinessObject | “BSA” |
| EventName | “RELEASE-WITHOUT-CLEARING” |
| ProgramName | “CL-A1FOP” |
| FeaturePack | “4.0” |
| RuntimeError | “ASSERTION-FAILED” |
| UserInterface | “paymentorder” |

Table 5.3: A test incident with six features

| rule | dr |
|---|---|
| UserInterface “paymentorder”→ Component “MOP” | 100% |
| RuntimeError “ASSERTION-FAILED” → Component “XCR” | 11.4% |
| RuntimeError “ASSERTION-FAILED”→ Component “MDA” | 8.6% |
| . . . | . . . |
| FeaturePack “4.0”→ Component “MDA” | 0.6% |

Table 5.4: Candidate components

$\mathrm{ds}(100) \subseteq \mathrm{ds}(250) \subseteq \mathrm{ds}(400) \subseteq \mathrm{ds}(500) \subseteq \mathrm{ds}(1000) \subseteq \mathrm{ds}(2000)$;
$\mathrm{ds}(100) \subseteq \mathrm{ds}(700) \subseteq \mathrm{ds}(1500) \subseteq \mathrm{ds}(2000)$;
$\mathrm{ds}(2000) \subseteq \mathrm{ds}(3000) \subseteq \mathrm{ds}(4000)$.

The corresponding training time and predicting time is shown in Table 5.5, Figure 5.6 and Figure 5.7. We use the same 20 incidents as in the previous two approaches to test and the rates of correct classification reaches 95% and the rate of positive classification even reaches 100%. Hence we go on with large testing data to see the results.

| # Train | 100 | 250 | 400 | 500 | 700 | 1000 | 1500 | 2000 | 3000 | 4000 |
|---|---|---|---|---|---|---|---|---|---|---|
| T(train) (s) | 12 | 13 | 412 | 657 | 1439 | 2938 | 6448 | 10006 | 21930 | 38108 |
| T(predict) (ms) | 1 | 1 | 2 | 3 | 4 | 6 | 9 | 11 | 16 | 25 |

Table 5.5: Training and predicting time in the approach of proper premises of partitioned context on subcontext with size of 3

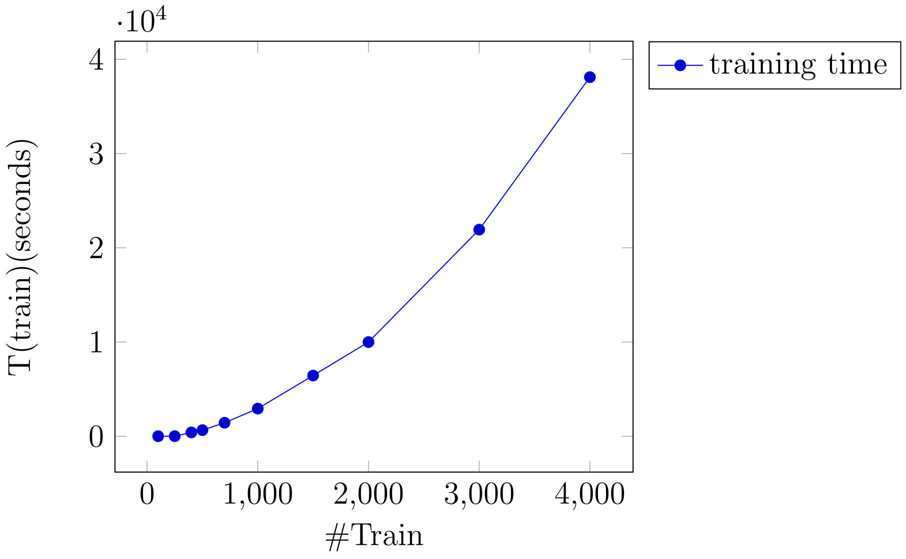

Figure 5.6: Training time in the approach of proper premises of partitioned context

Regarding the classification rates, we show two test cases on 2000 records of training data shown in Table 5.6. We use 457 and 951 test incidents for the first and second test series respectively. There are 438 out of 457 incidents which can be correctly classified and 454 out of 457 incidents which can be

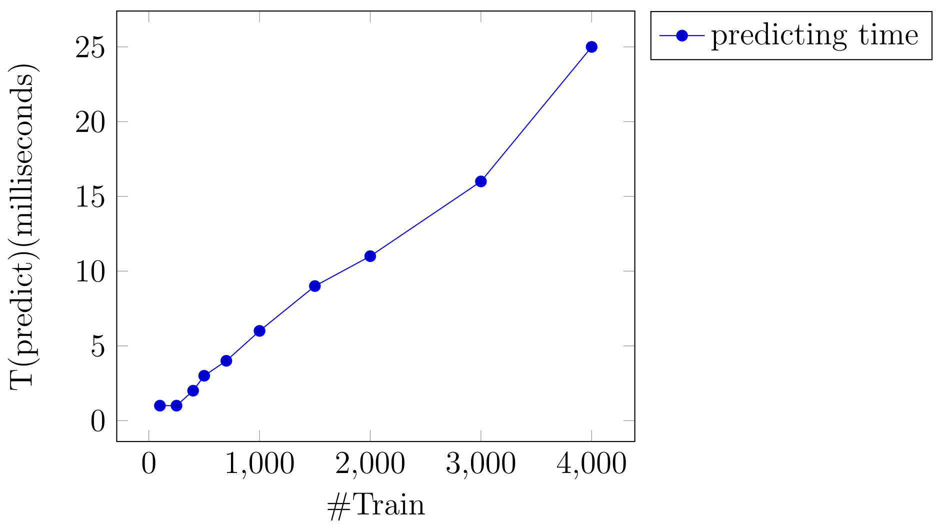

Figure 5.7: Predicting time in the approach of proper premises of partitioned context

positively classified, hence the rates of correct and positive classification is 96% and 99.8%. For the test series of 951 incidents, there are 766 out of 951 incidents which can be correctly classified and 886 out of 951 incidents which can be positively classified, hence the rates of correct and positive classification is 83% and 91%. In Figure 5.8 we show the curves of results when applying proper premises in partitioned context in the two test series. Compared with the approaches of closure computation and frequent premises, this approach of proper premises in partitioned context has much better result.

| TestSeries | 1 |
|---|---|
| # Training | 2000 |
| # Testing | 457 |
| # Positively Classified | 454 |
| # Correctly Classified | 438 |
| Positive Rate | 99.8% |
| Correct Rate | 96% |

| TestSeries | 2 |
|---|---|
| # Training | 2000 |
| # Testing | 951 |
| # Positively Classified | 886 |
| # Correctly Classified | 766 |
| Positive Rate | 93% |
| Correct Rate | 81% |

Table 5.6: Two test series

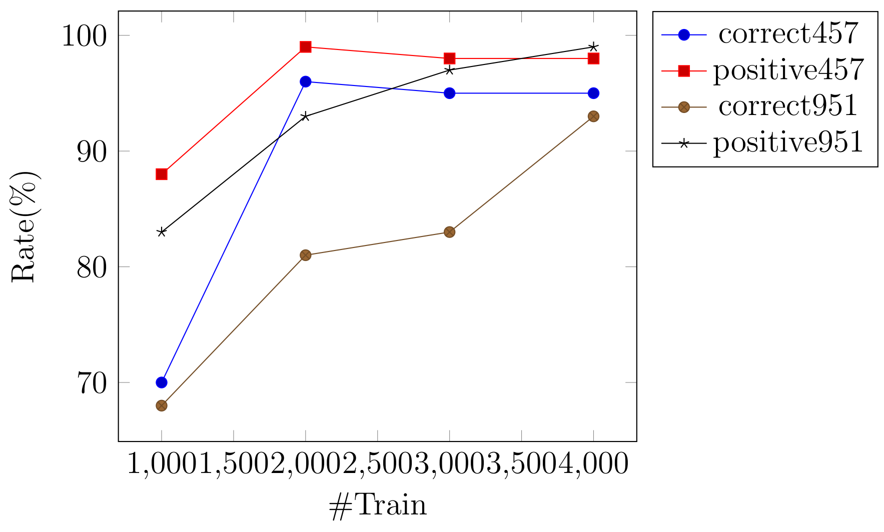

Figure 5.8: Rates in the approach of proper premises in partitioned context

## 5.4 Subset Searching

This section is corresponding to the approach in Section 4.4. Since there is no training phase in the approach of subset searching, for each new incident, the algorithm goes through the whole training data and searches for the candidate components. With our experiments we evaluated the predicting time as shown in Table 5.7 and Figure 5.9.

| # Train | 500 | 995 | 1990 | 2990 |
|---|---|---|---|---|
| T(predict) (ms) | 0.3 | 1.1 | 4 | 7 |

Table 5.7: Predicting time in the approach of subset searching

In Table 5.8 we show the features and values of one test incident for illustrating purposes. The right component for this incident is “AP-XCR”. The result of searching in the 2990 incidents of training data is shown in the Table 5.10: there are totally seven components are found as recommended components; component AP-XCR which is the right one we are looking for appears four times which is ranked top one.

In Table 5.9 we show one test series on 2990 training incidents and 20 testing incidents which are the same as we first tested in the previous approaches. Among the 20 test incidents, there are 11 incidents which can be positively classified. Among these 11 incidents there are 10 incidents which

| feature | value |
|---|---|
| Interface | “UI5” |
| FeaturePack | “4.0” |
| BussinessObject | “Production” |
| EventName | “ProviderAccess” |
| Error | “TEXT” |
| ProgramName | “SAPLBOA” |
| Component | “AP-XCR” |

Table 5.8: One test incident in subset searching

| TestCase | 1 |
|---|---|
| # Training | 2990 |
| # Testing | 20 |
| # Positively Classified | 11 |
| # Correctly Classified | 10 |
| Positive Rate | 55% |
| Correct Rate | 50% |

Table 5.9: One test series in subset searching

| Component | count | dr |
|---|---|---|
| AP-XCR | 4 | 40% |
| AP-FMD-MAT | 1 | 10% |
| AP-DUE-TXR | 1 | 10% |
| AP-SLO | 1 | 10% |
| AP-TIM | 1 | 10% |
| BC-CCM-MON-TUN | 1 | 10% |
| AP-BP-EE | 1 | 10% |

Table 5.10: Candidate components subset searching

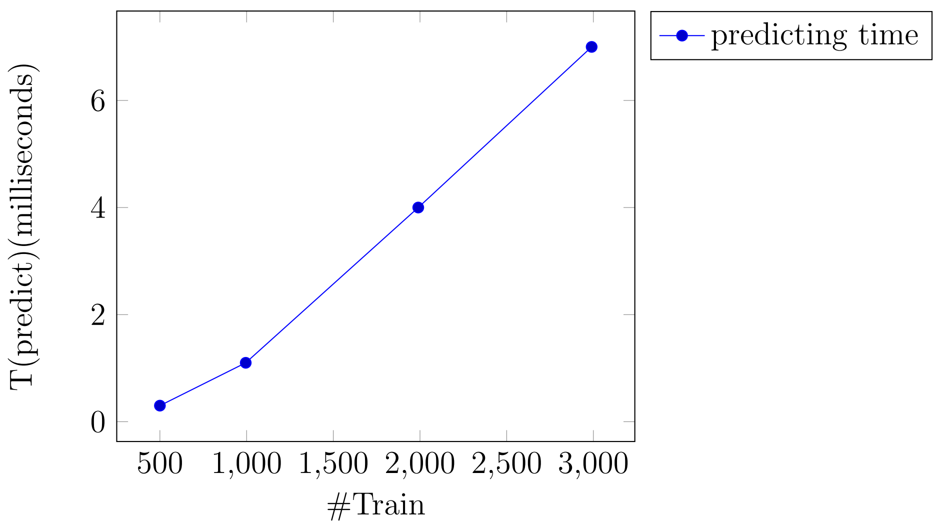

Figure 5.9: Predicting time in the approach of subset searching

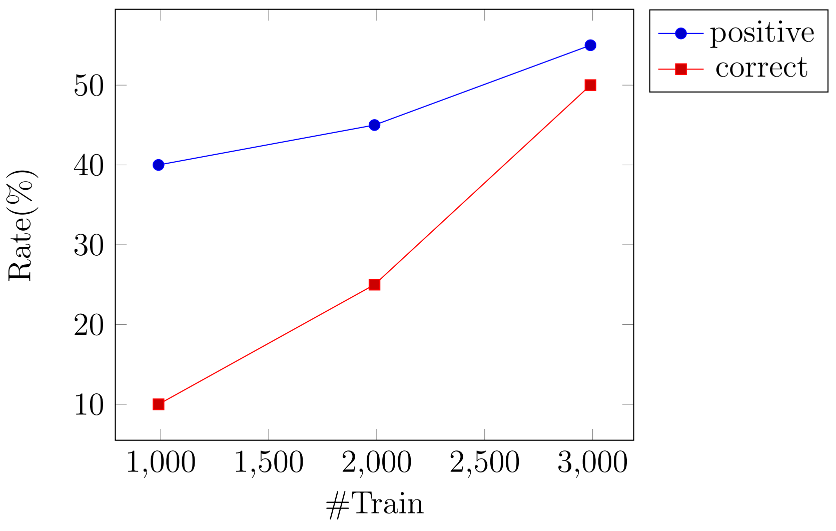

Figure 5.10: Rates in the approach of subset searching

can be correctly classified. The rates of positive and correct classification are shown in Figure 5.10. Compared with the previous approach, the result in this approach is not as good as approach of proper premises in partitioned context.

# Chapter 6

# Conclusions and Future Work

In this report, we introduced the bases of Formal Concept Analysis. We focus on two existing bases: basis of proper premises and Luxenburger Basis. But both bases are not suitable to deal with the erroneous data. To reach the goal of determining the right components for the newly reported issues, we make extensions of these bases. We defined the ET Basis and provided four approaches to generate the ET Basis. For each approach, we show the algorithm to generate an ET Basis and the way to define the degree of recommendation. Additionally, we make a comparison among the different approaches.

We implemented all algorithms and applied these bases to real software issues. In addition, we defined the rates of correct classification and positive classification, and implemented the algorithms that can calculate the rates. Furthermore, we designed several test cases and did experiments on real data to train and test.

The result of generating an ET Basis from each subcontext after partitioning the context seems to be the best. The rates of correct classification and positive classification have reached 80% and 90% respectively on large data set.

Comparing with current time support engineers spend on determining right component for one issue, it only costs milliseconds to determine the right component using our approaches. Furthermore, in the approach of proper premises in partitioned context, more than 90% of the testing incidents can be correctly classified when we have large training data.

There are several potential works that might be interesting to look into in future. Firstly, it would be also nice if we can evaluate the relation between correct classification and the size of subcontext in details. Secondly, it would be worth proving that the rates of classifications are statistical significant on even larger data. Thirdly, we are looking forward to applying these approaches to the real incidents newly created by customers. In addition, we can extend the number of attributes and it would be nice to have some measurement which can indicate that some feature is missing to correctly predict the component so that the right component can be found among those with high and similar degree of recommendations. For example, time frame is a potential attribute that may affect the rate of correct classification since newly processed incidents may have been processed on new components which replace the formerly used. Furthermore, we can also add several measurements. For example, the rate of negative classification, the reliability and so on.

To sum up, we provided several FCA approaches to solve a difficult issue in business software. And based on the experiment results, applying FCA to analyze large data is very promising in future.

# Appendix A

The experiments are performed using a laptop. The specification is as follows:

- Processor: Intel(R) Core(TM) i5 CPU
- RAM: 8.00 GB
- Operating System type: Windows 7 (64-bit)

We set up the environment for programming as follows:

- Java Virtue Machine: jdk 1.6.0_45
- Eclipse: eclipse-java-juno-SR1
- Plug-in of Eclipse: Clojure—Counterclockwise
- External Library: conexp-clj-0.0.7-alpha-SNAPSHOT-standalone.jar [Bor13]

# Bibliography

[BM10] Karell Bertet and Bernard Monjardet. The multiple facets of the canonical direct unit implicational basis. *Theor. Comput. Sci.*, 411(22-24):2155–2166, 2010.

[Bor13] Daniel Borchmann. conexp-clj library, 2013.

[GD86] J.-L. Guigues and V. Duquenne. Familles minimales d'implications informatives rsultant d'un tableau de donnes binaires. *Mathmatiques et Sciences Humaines*, 95:5–18, 1986.

[GW99] Bernhard Ganter and Rudolf Wille. *Formal concept analysis - mathematical foundations*. Springer, 1999.

[Hag07] Matthias Hagen. Lower bounds for three algorithms for the transversal hypergraph generation, 2007.

[JKZ09] Gaeul Jeong, Sunghun Kim, and Thomas Zimmermann. Improving bug triage with bug tossing graphs. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ESEC/FSE '09, pages 111–120, New York, NY, USA, 2009. ACM.

[Kuz04] Sergei O. Kuznetsov. On the intractability of computing the duquenne-guigues bas. *J. UCS*, 10(8):927–933, 2004.

[LS05] Lotfi Lakhal and Gerd Stumme. *Efficient Mining of Association Rules Based on Formal Concept Analysis*, volume 3626 of *LNAI*, pages 180–195. Springer, Heidelberg, 2005.

[Lux91] M. Luxenburger. Implications partielles dans un contexte. *Mathématiques, Informatique et Sciences Humaines*, 29(113):35–55, 1991.

[ORS06] Leon J. Osterweil, H. Dieter Rombach, and Mary Lou Soffa, editors. *28th International Conference on Software Engineering (ICSE 2006), Shanghai, China, May 20-28, 2006*. ACM, 2006.

[RDB11] Uwe Ryssel, Felix Distel, and Daniel Borchmann. Fast computation of proper premises. In Amedeo Napoli and Vilm Vychodil, editors, *CLA*, volume 959 of *CEUR Workshop Proceedings*, pages 100–113. CEUR-WS.org, 2011.

[STB$^+$01] G. Stumme, R. Taouil, Y. Bastide, N. Pasquier, and L. Lakhal. Intelligent structuring and reducing of association rules and with formal concept analysis. In F. Baader, G. Brewker, and T. Eiter, editors, *KI 2001: Advances in Artificial Intelligence. KI 2001*, volume 2174 of *LNAI*, pages 335–350, Heidelberg, 2001. Springer.

[Stu02] Gerd Stumme. Formal concept analysis, 2002.

[TSK05] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Addison-Wesley, 2005.

# Index