Concept-based Source Code Analysis for Software Maintenance

Hnin Pwint Phyu
University of Computer Studies, Yangon
ahpwint@gmail.com

Thi Thi Soe Nyunt
University of Computer Studies, Yangon
thithisn@gmail.com

Abstract

Designing, understanding and maintaining the source code is a crucial task in reengineering for software maintenance. Formal Concept Analysis (FCA) is a mathematical technique which has been applied in several fields and various problem domains of software research such as restructuring, correction of design defects, and object-oriented class identification tasks because it is useful to discover hierarchical groupings of objects having similar attribute. So, FCA is applied in the proposed system for analysis of software in order to support maintenance as it allows identifying concepts, or groupings of elements, that have common properties. Although FCA is composed of three parts- formal context, formal concept and concept lattice, this paper focuses to present detailed computing of formal concept in FCA with the proposed algorithm.

Keywords-- ***Formal Concept Analysis**, **Software Maintenance, Formal Concept Computing***

1. Introduction

A support of a system model from its code has become a standard part of modern analysis and design tools to satisfy the needs for reengineering, updating a model to the evolved code, and taking the legacy context into account. Moreover, understanding, designing and maintaining a software structure is a crucial task in reengineering. When an object-oriented software or model becomes bigger and bigger, it has duplicated elements start to appear, decreasing the readability and the maintainability of the software. A well designed class hierarchy makes the software easier to understand, maintain and reuse. So, Formal Concept Analysis is proposed to analyse software for ease of maintenance by developer (maintainer) before actual making of maintenance. FCA is a well-established technique for identifying groups of elements with common sets of properties. It can be successfully applied to complex software systems in order to automatically discover a variety of different kinds of implicit, recurring sets of dependencies amongst design artifacts.

FCA has been applied in the following problem domains of software engineering. FCA is applied in restructuring or redesigning class hierarchy [5][6][7][12], removal of code duplication [16], refactoring a bad design [10][13]. FCA is used in these fields to show in a way that is easier for the software engineer to see where the duplications take place, and also facilitate the possibility of applying the proposed refactoring. Also the manual searching for design defects in code is a tedious task. So, FCA based identification of design defects and their correction is used to solve this problem [2][9]. The idea of applying FCA to source code is not new. However, this work is intended to help the developer by doing software analysis with FCA for software maintenance support.

As a software maintenance point of view, a key difficulty in the maintenance and evolution of complex software systems is to recognize and understand the implicit dependencies that define contracts that must be respected by changes to the software. FCA guarantees the main properties of object oriented programming such as simplicity, understandability, extensibility, reusability, minimum redundancy, and subclass as specialization etc. The proposed system is related to the software maintenance activities as it is intended to help the developer by doing software analysis which save time and cost for ease of maintenance.

The paper starts with introduction of the problem domain and some other related work with overview (Section 2). Next, the detailed framework of FCA is discussed (Section 3), followed by the proposed algorithm for formal concept computing with case study is presented (Section 4). Finally, the paper is concluded (Section 5).

2. Related Work

Formal Concept Analysis provides a formal framework for identifying groups of elements sharing sets of properties. It focuses on the lattice structure induced by a binary relation between a pair of sets. FCA is composed of formal context, formal concept and concept lattice. A formal concept is a group of elements and their shared properties. Relationships among concepts lead to a concept lattice. Such concepts represent semantic properties of the underlying problem domain.

In 1993, work on the application of concept analysis in the area of program understanding and reengineering was initiated [11]. However, FCA is still applied in today's reengineering activities with some modification. The theory of FCA and concept lattice has mentioned in many research papers with various mathematical and algorithmic backgrounds. In this section, the only some relevant papers for reengineering from source code are mentioned. However, detailed concept computing algorithm or procedure is not found in these papers although they described general terms for formal concept.

The work investigated the application of FCA to the task of reverse engineering code inspection of individual java classes for the problem of understanding the structure of the class are found in [4]. This work considered the relation as their use of field and ignored the interaction of the class with other classes. Embedded call-graph (ECG) is also used in this paper to represent method-invocations (interaction between methods). The suggestion of a structure reading order for methods is also included. In this paper [11], explaining the concept lattice and discussion of the corresponding lattice offers insight not obvious from the original table is found. Concept analysis is applied in this to find modules in legacy code in C programs by analyzing the variable usage of module (relation) between procedures (objects) and global variables (attributes).

The problem of inferring configuration structures from existing source code is analyzed and a tool is developed in [8]. Their configuration dependent on sources pieces (source code pieces as objects and procedure symbols as attributes) and used C preprocessor for configuration management. This paper emphasized on concept lattice and construction of context table for their objects and attributes and also considered data reduction.

Concept analysis based framework for detecting and remediating the design problems is presented in this paper [12]. This paper focused on redesigning and restructuring class hierarchies of C programs and analyzed the relation of the types of variables and class members. This paper obtained concept lattice by establishing a property of the connection between a table and its lattice. The theoretical background and various algorithms of FCA are described in [14]. The detailed explanation of FCA theory background, its applications are described and tracing of algorithms (TITANIC, Naïve approach) with examples are also explained in this.

Most papers described formal concept with concept forming operators or mathematical terms. However, the explanation on concept by concept forming of formal concepts is not found. For the beginners, FCA is not that easy to understand, hence step by step concept computing with case study is detail explained in this paper. And, java program as input source code is analysed in the proposed system.

3. The Theoretical Framework of Formal Concept Analysis

In this overview, we do not present the beautiful mathematical background and merely present explaining of formal concept for concept analysis.

The original motivation of formal concept analysis was the concrete representation of complete lattices and their properties by means of formal contexts (data tables) that represent binary relations between objects and attributes. Although FCA is composed formal context, formal concept and concept lattice, only the step by step formal concept computing with the proposed algorithm is emphasized on this paper. There are many considerations of relation for formal context from input source code and much more to say about concept lattices, their structure theory, related algorithms and methodology. However, the basic theorem suffices for the purpose of this paper.

3.1. Formal Context

A triple K= (G,M,I) is called a formal context if G and M are sets, and I $\subseteq$ G×M is a binary relation between G and M. Elements of G are called the objects, elements of M are called the attributes and I is called the incidence relation of the context K= (G,M,I).

Graphically, a formal context is represented as a cross-table. The rows of the table are headed by the object names and the columns are headed by the attribute names. So, a cross (×) in row g and column m means that the object g has the attribute m. A table entry containing a blank symbol (empty entry) indicates that the object does not have the attribute.

3.2. Formal Concept

In formal context, there is set of objects consisting which are closely connected to the set of attributes. This pair forms a formal concept. A tuple C= (A,B) is called a formal concept of the context (G,M,I), if A is a subset of G and B is a subset of M. B is the set of attributes common to all objects in A (intent) and A is the set of objects which have all attributes in B (extent).

A concept is constituted by its extension, comprising all objects which belong to the concept and its intension, including all attributes (properties, meanings) which apply to all objects of the extension. Concepts can only live in relationships with many other concepts where the subconcept-superconcept relation. In order to compute the infimum (greatest common subconcept) of two concepts, their extents must be intersected and their intents must be joined. Analogeously, the supremum (smallest common superconcept) of two concepts is computed by intersecting attributes and joining objects.

3.3. Concept Lattice

The ordered set of concepts is called a concept lattice of formal context (G, M, I). The concept lattice can be considered as a graph that is a relation. Different nodes of lattice are concept pairs (A, B). Consequently, the designated subcontext formed by those concept pairs has a concept lattice isomorphic to the concept lattice of the whole formal context. The family of the concepts obeys the mathematical axioms defining a lattice, and is called more formally a concept lattice. The lattice structure imposes a partial order on concepts and allows a labeling of the concept with object or attribute name.

4. The Proposed Algorithm for Computing of Formal Concept

In this section, the proposed algorithm is presented and process of algorithm is detail discussed with example case study. It consists of two steps for decomposing and composing. In this algorithm, the initial decomposing step is presented in Step 1 as initial decomposing is important in concept lattice. The result from the first step of the algorithm is directly used in the first level of concept lattice. And then, concept by concept computing of formal concept is achieved in Step 2. The step by step forming concepts are considered as the concepts. The algorithm iterates the computing of concepts for Step 2 until their extent must not be intersected upon given attributes their intents must not be joined upon given objects.

For every input source codes, the first step of the algorithm finds independence of object sets (I) as initial concepts. If the (I) is equal to the whole object set, this concept is placed as top concept in concept lattice. If it is not equal to the whole object set, these concepts are placed as initial decomposing concepts in concept lattice. The second step of the algorithm computes iteratively the next concepts under each concept of the concepts of the first step. In this step, all attributes and only the object set of each concept in formal context are considered for further concepts. The algorithm terminates each concept' object set is not further divided or the number of object set is equal to one. By merging the resulted concepts in each step, the resulting concepts are considered as the last concepts but do not taking overlap.

```
Input: Formal context K= (G, M, I) where
        G - set of objects
        M- set of attributes
        I – relation between G and M
Output: Concept {A, B} where
          A- object set
          B- attribute set
Step 1: for all $B \in M$ do
          compute INDEPENDENCE (I).
          C $\leftarrow$ I {A, B};
          return $\bigcup_{i=1}^{n} c_i${A,B}$\in$ C
Step 2: i $\leftarrow$ 1;
          for all $c_i \in C$ do
            for all A, B $\in c_i$ do
               r $\leftarrow x_i \cap x_j$;
             while (r $\neq x_j$ or n(r) $\neq$1)
               $x_i$ $\leftarrow$ CLOSURE (A,B)$_i$ ;
               C $\leftarrow x_i$ ;
               return $\bigcup_{j=1}^{n} c_j${A,B}$\in$ C;
               i++;
             end while
           end
          end.
```

Figure 1. CONCEPT_COMPUTING Algorithm

The following procedures are the independence procedure and closure procedure. The independence procedure computes the uncommon and largest object set upon each attribute by comparing each extent with another for all attributes in the context table.

The closure procedure find the concept corresponds to a maximal rectangle in the context table upon the extent consisting of the objects that share the given attributes and the intent consisting of all attributes shared by given objects.

```
1.  for all $b_j \in M$ do
2.   $A_j \leftarrow$ Max (A);
3.  end
4.  for all $A_k \in M$ do
5.  for (k □ j) do
6.  if ($A_k \subseteq A_j$) then
7.   I $\leftarrow A_j$;
8.  go to line 16.
9.  else I $\leftarrow A_j$;
10.  M $\leftarrow$ M – $A_j$ ;
11.  $A_j \leftarrow A_k$ ;
12. go to line 4.
13. end if.
14. end
15. end
16. return I{A}.
```

Figure 2. Procedure of INDEPENDENCE (I)

```
1.  for all $B_i \in c_i$ do
2.    for all $A_i \in c_i$ do
3.     f $\leftarrow$ ( $B_i \times A_i$) $\cup A_i \cup$ B;
4.     g $\leftarrow$ (f $\cap$ ($A_i \times B_i'$ )) $\cup$ ($A_i \times$ B);
5.       CLOSURE (A, B) $\leftarrow$ g {A, B};
6.       return $c_j$ {A, B} $\in c_i$
7.    end
8.  end
```

Figure 3. Procedure of CLOSURE (A, B)

4.1. Example Case Study

In this section, the family class hierarchy from [3] is considered as sample source code for FCA.

Although there are many considerations of relations for formal context, only the (has a) relation of the class and the member data and function (including inheritance relation) is considered in this example. The class has the member data and function is only considered and the relation of class and member data will be discussed in the continued work. Assume the class and member data as the following notation.

name	- a
Person( )	- b
talk( )	- c
Baby( )	- d
vocabulary	- e
Toddler( )	- f
gender	- g
Teenager ( )	- h
Mother( )	- i
Father( )	- j
cyear	- k
yearOfB	- l
Grandad ( )	- m
Granny( )	- n

Person	- 1
Baby	- 2
Toddler	- 3
Teenager	- 4
Mother	- 5
Father	- 6
Grandad	- 7
Granny	- 8

A formal context is represented as a cross-table. In this example, the class names (except main class) are defined as the object set and the member data (variables and function) are defined as the attribute set. The object names are positioned in the rows of the table and the attribute names are positioned in the columns of the table.

The relation for the formal context in this example is considered as the class has the member data. A cross (×) means the class (object) g has the member data and function (attribute) m. A table entry containing a blank symbol (empty entry) indicates that the class (object) does not have the member data (attribute). In this example, a separator-free form for sets is used, e.g., 123 stand for {1, 2, 3}.

Step 1:

	a	b	c	d	e	f	g	h	i	j	k	l	m	n
1	×	×	×											
2	×	×	×	×										
3	×	×	×		×	×								
4	×	×	×				×	×						
5	×	×	×						×					
6	×	×	×							×				
7	×	×	×								×	×	×	
8	×	×	×											×

Concept 1: {12345678, abc}

Step 2.1: concept 1 {12345678, abc}

	a	b	c	d	e	f	g	h	i	j	k	l	m	n
1	×	×	×											
2	×	×	×	×										
3	×	×	×		×	×								
4	×	×	×				×	×						
5	×	×	×						×					
6	×	×	×							×				
7	×	×	×								×	×	×	
8	×	×	×											×

Concept 2 {2, abcd}
Concept 3 {5, abci}
Concept 4 {6, abcj}
Concept 5 {8, abcn}
Concept 6 {3, abcef}
Concept 7 {4, abcgh}
Concept 8 {7, abcklm}

The concept 1 in this example is the first step of the proposed algorithm. The resulted concepts of Step 2.1 are the first iteration of the second step of the algorithm. In this second step, concept {1, abc} is not included because its intents must not be joined upon given objects and are equal to top concept. The algorithm terminates each concept' object set is not further divided or the number of object set is equal to one. So, the algorithm terminates with concepts 2 to 8 as their number of extents is equal to one or not further divided. If it can be further intersected, many concepts will be resulted under each concept and also further steps for step 2 (for each concept) will be found. Therefore, the resulting concepts are concept 1{12345678, abc}, concept 2 {2, abcd}, concept 3 {5, abci}, concept 4 {6, abcj}, concept 5 {8, abcn}, concept 6 {3, abcef}, concept 7 {4, abcgh}, concept 8 {7, abcklm}.

4.1.1. Concept Lattice Hierarchy

There are many considerations for concept lattice. However, this paper focuses on concept computing in FCA. So, concept lattice structure will be discussed later. In this concept lattice, a concept is constituted by its extension, comprising all objects which belong to the concept, and its intension, including all attributes which apply to all objects of the extension. Each concept is represented by a little circle so that its extension (intension) consists of all the objects (attributes) whose names can be reached by a descending (ascending) path from that circle.

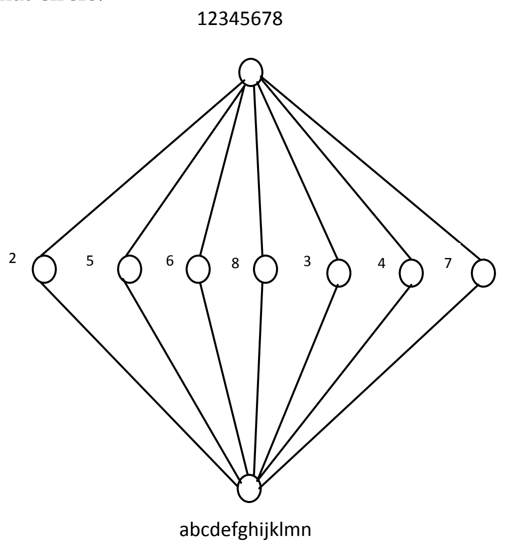

Figure 4. Concept Lattice for Family Hierarchy

Recently, all concepts under each concept number in each step is considered to draw relation with others by using hasse diagram**.** In the hasse diagram, the covering relations can be drawn, i.e. those relations where x<y and there does not exist z such that x<z<y.

For example, considering that concept x and concept y. These two relations imply concept x **<** concept y when R is a partial order. If concept x < concept y is not a covering relation, covering relation of concept x < concept z < concept y is considered. If the latter relation exists, an edge is not directly drawn between concept x and concept y. An edge from concept y to concept x via concept z can be drawn. If concept x < concept y is a covering relation, an edge can be drawn between concept x and concept y. In this case, "2<1" or "concept 2 has under concept 1" is investigated. If it is true, concept 2 is drawn under concept 1 in hasse diagram.

For “3<2”, it is not a covering relation or “concept 3 does not have under concept 2”. So concept 2 and concept 3 does not have relation and concept 3 is not drawn under concept 2. The relation within concepts in steps of case study is considered as this procedure. The resulting concept lattice for this case study is described in figure 4. In order to get such a lattice, it needs to remember that a subconcept of a concept has a smaller object set, but a larger attribute set. That is, the more we go down in the lattice, the more we get precise information about smaller object sets.

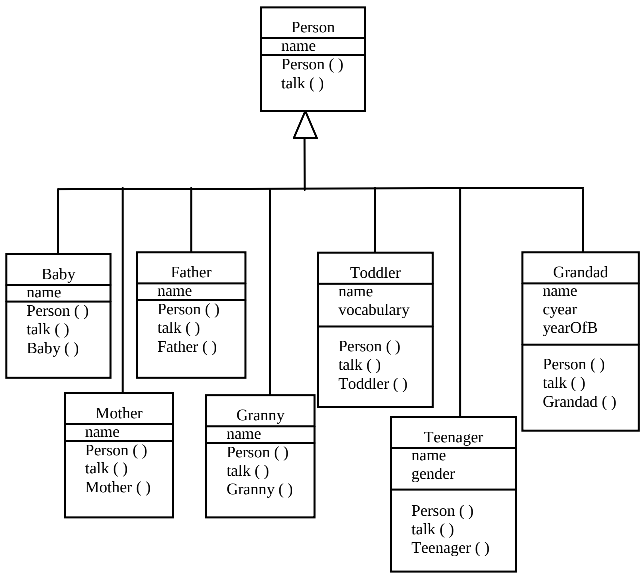

Figure 5. The Resulted Hierachy by FCA

In the resulting actual concept lattice (not considering with notations), the top concept is concept 1{Person Baby Toddler Teenager Mother Father Grandad Granny , name Person() talk ()} or { Person, name Person() talk ()}.The following concepts are concept 2{Baby, name Person() talk () Baby()}, concept 3{Mother, name Person() talk () Mother()}, concept 4{Father, name Person() talk () Father()}, concept 5{Granny, name Person() talk () Granny()}, concept 6{Toddler, name Person() talk () vocabulary Toddler( )}, concept 7{Teenager, name Person() talk () gender Teenager ( )}, concept 8{Grandad, name Person() talk () cyear yearOfB Grandad()}.

The resulted concept lattice may be viewed as the above figure by developer. The ordering between lattice elements may be viewed as inheritance relationship in the class hierarchy. Besides, the developer can see which part is closely related, which relation has in these classes and can know which part should be removed, which relation should be added, which part should not be changed and so on.

5. Conclusion

Formal Concept Analysis (FCA) is a strong tool which provides a framework for class hierarchy design and maintenance. FCA is still useful in today’s reengineering activities with some modification. The aim of the proposed system using FCA is to help the developer to decide right decision for doing the maintenance activities easily.

This analysis system can give the prototype of the software to the developer for making decision of maintenance should be done or not for that software. Besides, which part is closely related, which part should be removed, which part should be partitioned and which part should not be changed and so on can be decided. By analyzing software with concept-based method, it can save time and cost for maintenance activities. Only the relation of the class has the member data is considered for formal context in this paper, but the relation of formal context will be widely considered in the continued work. Also the detailed concept lattice structure will be presented in the continued work. As a future plan, the result of the proposed system based on lattice hierarchy will be evaluated with design metrics for maintainability.

References

[1] G. Arévalo, S.Ducasse, and O. Nierstrasz, “*Discovering Unanticipated Dependency Schemas in Class Hierarchies*”, 2010.
[2] S. Azhar, S. Samia, A. Anam, and A. Moein, “*Detection of Problems in Class Hierarchies and Their Correction through Formal Concept*”, in ICSM, 2010.
[3] C. Britton "*Object-Oriented System Development: A Gental Introduction*", 2000.
[4] U. Dekel, “*Applications of Concept Lattices to Code Inspection and Review*.”
[5] J. Falleri, M. Huchard, and C. Nebut, “*A generic approach for class model normalization*.”
[6] R. Godin, and P. Valtchev, *“Formal concept analysis-based class hierarchy design in object-oriented software development.”*
[7] M. Huchard, and H. Leblanc, “*From Java Classes to Java Interfaces through Galois Lattices*”, 1999.
[8] M. Krone, and G. Snelting, “*On the Inference of Configuration Structures from Source Code*.”
[9] N. Moha, J. Rezgui, Y. Gueheneuc, P. Valtchev, and G. E. Boussaidi, “*Using FCA to Suggest Refactorings to Correct Design Defects.”*
[10] P. S. Patil, “*Applying Formal Concept Analysis to Object Oriented Design and Refactoring.”*
[11] G. Snelting, “*Software Reengineering Based on Concept Lattices*.”
[12] G. Snelting, and F. Tip, “*Reengineering Class Hierarchies Using Concept Analysis*”, 1998.
[13] M. Streckenbach, and G. Snelting,” *Refactoring Class Hierarchies with KABA.”*
[14] G. Stumme, “*Tutorial Formal Concept Analysis*”, 2002.
[15] P. Valtchev, D. Grosser, C. Roume, and M.R. Hacene, “*Galicia: An Open Platform for Lattices*.”
[16] R. A. R. Torres, “*Source Code Mining for Code Duplication Refactorings with Formal Concept Analysis*”, 2004.