2002

C++ class retrieval using formal concept analysis.

Hansoo. Ahn
University of Windsor

C++ CLASS RETRIEVAL
USING FORMAL CONCEPT ANALYSIS

By

Ahn, Hansoo

A Thesis
Submitted to the Faculty of Graduate Studies and Research
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

2002

962044

ABSTRACT

Type is used as a search key in component retrieval. In retrieving a class in an object-oriented language such as C++ or Java, the type information of a class can be represented as a set of types of functions and variables defined in the class. Formal Concept Analysis is applied to construct a structured class library in which search can be much faster than in an unstructured class library.

In this thesis, we present a method of representing a C++ class based on type information, and constructing a C++ class library using Formal Concept Analysis, and a retrieval process based on the structured library called a concept lattice. If the desired class that exactly matches with the query which the user provided does not exist, then more generic template classes are retrieved.

A prototype retrieval system for C++ classes is implemented based on the proposed retrieval method. This prototype system constructs a concept lattice of the classes and provides a user interface that supports the retrieval process.

ACKNOWLEDGEMENTS

I would like to thank Dr. Park, my supervisor, for his advice and guidance for my thesis work. He spent much time to help me to solve many problems in my thesis and inspires me.

I also would like to thank Dr. Li, Liu the internal reader in my thesis committee. He gave me many valuable suggestion and guidance.

I would specially like to thank Dr. Tsin, Peter who is a chair of my thesis defense and Dr. Suh, Sang who is an external reader from the department of economics.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 INTRODUCTION

1.1 Software Reuse

Software reuse is considered as the most effective technology for improving software quality and productivity [8,14,15,19,28]. This concept was introduced in late 1960's to solve "software crisis" which had been faced until now in developing software [25]. For many years of research, software reuse is being recognized as a reasonable way to satisfy currently increased demand of software development. Frakes and Gandel said in their paper [28] that a fundamental problem in software reuse is the lack of tools for representing, indexing, storing and retrieving reusable components. The major issue in software reuse is how to design a storage and reusable components, how to classify and index the components, how to store and search for the components, and how to evaluate the effectiveness of the system.

Generally, there are four processes in software reuse [18].

- Component insertion which inserts reusable components into a library for reuse [18].

- Component retrieval that retrieves suitable components from the library based on some criterion [18].

- Component composition that combines a set of components whose combined behavior satisfies the requirement. In some cases, the individual components in the library cannot satisfy the user's query, then the system will compose some components and see if the composition satisfies the query [18].

- Component adaptation which modifies the reusable components and adapt them into new application. This phase is applied only when it is very necessary, because modification dims the efficiency of software reusability. Programmers have to understand what the original components are supposed to do before they make any modification. In many cases, modifying components takes even more time and labor than starting from scratch. Since modifying a component is very expensive, it should be avoided or reduced to a minimum [18].

1.2 Semantic-based Software Retrieval

There have been several semantic software component retrieval

methods using types, execution and specification.

1.2.1 Type-based retrieval

In this method, type information is used as search key to retrieve reusable components [1,2,20,21,23]. Zaremski and Wing introduced signature matching as a solution for retrieving functions and modules. Signature matching is the process of determining when a library component matches a query [1]. The signature of a function is simply its type and the signature of a module is a multi-set of user-defined types and a multi-set of function signatures [2]. The deficiency of this approach is its low accuracy but it is very efficient compared to the execution-based or specification-based retrieval.

1.2.2 Execution-based retrieval

The basic idea of this approach is to send sample test data and component candidates to the search system and the search system executes and validates all component candidates, and finally returns the result to the user [16,18,27]. This is a better approach rather than the type-based retrieval because the type-based retrieval cannot distinguish functions if their types are the same. For example, the type of the function for adding two

integer type parameters is actually the same as that of the function for subtracting. But it has a deficiency such that the amount of time for retrieving is very long and non-terminated execution might occur in a real implementation.

1.2.3 Specification-based retrieval

This approach uses formal specification to represent reusable components [3,4,5,17]. In signature matching, signature represents the types of components, but formal specification describes the behavior of software components. Using this approach, we can increase the precision of the search result in the process of searching components in the library.

1.3 Formal Concept Analysis

1.3.1 Basics of Formal Concept Analysis

Formal Concept Analysis is a mathematical approach to data analysis based on the lattice theory [7]. It constructs a lattice to provide structured information from unstructured information. This can be summarized as follows [6,7,12,24]

Formal contexts

It is represented as a triple relation (O, A, R). In here, O is a

set of objects, A is a set of attributes, and R is a binary relation between objects and attributes. If a certain object O_1 has an attribute A_1, it can be expressed $(O_1, A_1) \in R$.

Formal concepts

A formal concept is a pair of (O_1, A_1) where O_1 is the object set of O and A_1 is the attribute set of A. In this pair, all objects in O_1 have all attributes in A_1. O_1 is called the extent and A_1 is called the intent of the concept (O_1, A_1).

Super-concept and Sub-concept

Let (O_1, A_1) and (O_2, A_2) be two concepts of a context. (O_1, A_1) is called a sub-concept of (O_2, A_2) and (O_2, A_2) is called a super-concept of (O_1, A_1) if the set of objects of O_1 is included in that of O_2 or the set of attributes of A_2 is included in that of A_1.

1.3.2 Component retrieval using Formal Concept Analysis

Park and Lindig used Formal Concept Analysis in retrieving components [9,10,26,27]. A function library is represented as a formal context using a relationship between function name and function type [26]. The name of each function can be represented by object and the type of each function can be

represented by attribute. From the context table that shows a relationship between objects and attributes, we can extract concepts and build a concept lattice. As explained above, a concept lattice shows us the relation between each concept. He argued that using this method in constructing a component library makes the average retrieval time faster compared with in an unstructured library.

1.4 Overview of the Thesis

1.4.1 Motivation

There are several approaches and implementation in function retrieval but not in class retrieval especially in the C++ language. The most significant thing in retrieving reusable components is that how to construct a component library to locate components we want to retrieve. Formal Concept Analysis is an effective way to construct a class library in that we can construct a lattice structure of a class library. Type-based retrieval that is a kind of semantic-based retrieval with execution-based retrieval and semantic-property based retrieval has some deficiency in searching a component because it just considers type not behavior of each component. But, type-based retrieval is an easy and efficient way to locate component

compared to execution-based retrieval and specification-based retrieval because the cost of latter two approaches is expensive and difficult to implement in real application. Park applied this method to the function retrieval in a functional language [26]. Tam presents how to retrieve functions in an imperative language such as C using the semantic-based approach [18].

1.4.2 The objective

The objective of this thesis is to apply the type-based retrieval approach in retrieving C++ clasess with Formal Concept Analysis. The prototype system that we developed computes a set of concepts from a context table and constructs a concept lattice. It is designed and implemented in java for retrieving C++ classes. When a user executes this program, it calculates all concepts and displays the concepts in java frame. A user selects a set of type in the frame, and then the results are sent back and displayed to the user.

1.4.3 Organization of the thesis

This thesis is organized into five chapters.

CHAPTER 1 gives a brief introduction of software reuse and

several semantic-based retrieval approaches. In addition to this the concept of Formal Concept Analysis is described.

CHAPTER 2 describes C++ component retrieval including function retrieval and class retrieval based on type information, and summarizes a C++ class retrieval method based on FCA.

CHAPTER 3 presents an implementation and usage in component retrieval with examples.

CHAPTER 4 gives conclusion and future work.

CHAPTER 2 C++ COMPONENT RETRIEVAL USING TYPE INFORMATION

2.1 Function Retrieval

C functions can be represented by input parameters and return type. Using this type information, we can locate the functions, which we want to retrieve from a function library. Tam presented this approach in [18]. He used type information as search key to retrieve functions. In this system, users provide type information and send a query to the system. The retrieval system searches the library and returns the function that matches the user's query type.

Example

```
int Multi_fun(int a,int b){
     int result=0;
     result=a*b;
     return result
}
bool is_true(int a){
     if(a>70)
          return true;
```

```
    else
        return false;
}
void what_is_myName(char *ch){
    printf(My name is %s\n",ch);
}
int what_is_double(int num){
    int double_of_value;
    double_of_value=num*2;
    return double_of_value;
}
```

In this example, we have four functions in our library. We can represent these four functions according to type of each function.

Multi_int: `(int,int)->int`
is_true: `(int) -> bool`
what_is_myName: `(char*)-> void`
what_is_double: `(int)->int`

Now, we construct our small library using type information, which consists of the argument types and the return type. When we retrieve the function, we make a query that consists of type

information. For example, we need a function that does summation for two integer variables and returns integer value. We make a query `(int,int)->int`. This means the number of argument is 2 and the types of two arguments are integer and the return type is also integer. This is done by the expectation of users. A user has to expect the types of the function that he or she wants to retrieve. After this retrieval system receives this query, it searches the library and finds that the function *Multi_int* is exactly matched with above query. The basic idea is simple in this system. But we have two problems in this retrieval system. First, for this retrieval system, every function is the same if they have the same type information. Second, the library is not structured. In this small example, it might have no problem, but if we have huge number of functions in our library, then it is not efficient for retrieving.

```
#include <iostream.h>
#include <conio.h>
template <class T>
class Somedata{
     protected:
          int x,y;
          T data;
     public:
```

```
        somedata(int ax,int ay,T adata){
            x=ax;
            y=ay;
            data=adata;
        }
    void outdata(void){
        cout << data;
     }
};
void main(){
    Somedata<int> int1(10,10,15);
    Somedata<char> char1(15,15,'A');
    Somedata<float> real1(35,10,2.78);
    int1.outdata();
    char1.outdata();
    real1.outdata();
}
```

This is a small example of a template class in C++. Class 'Somedata' is declared as a template class. In main method, we instantiate three objects of class '*Somedata*'. The object 'int1' is given type 'int' to class '*Somedata*'. By doing this, the parameter 'T' defined in a template class '*Somedata*' is considered to be an integer type. This concept gives us

convenience and efficiency because we do not need to create many classes whose behavior is the same but type is different. This concept is also utilized in retrieving a reusable component. In a function retrieval of functional languages [11,26], Park retrieved parametric polymorphic functions if there is no function that matches with the query. "Poly-type" means a set of types we can substitute with any type. Having a more general type component is very useful in the retrieval procedure because finding the component to match with query exactly is not easy and also not possible. Actually, we need some flexibility in comparing type information between a query and components in the component library.

2.2 Class Retrieval

C++ class consists of several functions and variables. In function retrieval, we just consider types of input parameters and return type in type-based approach. But, we have to consider each type of every function in the class that we want to retrieve.

One thing we need to consider in this step is the C++ language supports multiple inheritances. If class 'A' is subclass of class 'B' then the functions defined in the super-class 'B' can be

accessible from class 'A'. This means that the type of class includes the type of its super-class.

2.3 Formal Concept Analysis in Retrieving C++ Classes

In this thesis, we adopt Formal Concept Analysis for the retrieval method.

2.3.1 Building a formal context table

In this step, we build a formal context table from a sample class library. A set of objects consists of class names, and a set of attributes consists of the type of each function. The formal context represents a relation between objects and attributes. This means that it represents the relation between class names and the type of every function.

The type 'a' in the sample class library is a general type that can be substituted with any type. The classes in this sample class library are as follows:

Class A

int fun_1(int x, int y), void fun_2(), a fun_3(a x), bool fun_4()

Class B

int fun_5()

Class C

void fun_6()

Class D

void fun_7(a x), bool fun_8()

Class E

int fun_9(int x)

Class F

int fun_10(), void fun_11(a x), a fun_12(a x)

Class G

void fun_13(), void fun_14(a x), bool fun_15(), int fun_16(int x)

Then the set of types is as follows:

- T1: (int,int) -> int
- T2: () -> int
- T3: () -> void

- T4: (a) -> void
- T5: (a) -> a
- T6: () -> bool
- T7: (int) ->int

	T1	T2	T3	T4	T5	T6	T7
Class A	√		√		√	√	
Class B		√					
Class C			√				
Class D				√		√	
Class E							√
Class F		√		√	√		
Class G			√	√		√	√

Table 1: A formal context table

Table 1 is a formal context for this example. The symbol '√' means that there is a relation between corresponding object and attribute. In this formal context table, the column name means the set of attributes and the row name means the set of objects.

2.3.2 Computing all concepts

From the above context table, we compute all concepts. In computing concepts, we first get atomic concepts from the

context table and calculating concepts comparing with each concept [22]. The result of the computation is as follows:

- Concept 0 = (Bottom)({}, {T1,T2,T3,T4,T5,T6,T7})
- Concept 1 = ({Class A}, {T1,T3,T5,T6})
- Concept 2 = ({Class B, Class F}, {T2})
- Concept 3 = ({Class A, Class C, Class G}, {T3})
- Concept 4 = ({Class D, Class G}, {T4,T6})
- Concept 5 = ({Class E, Class G}, {T7})
- Concept 6 = ({Class F}, {T2,T4,T5})
- Concept 7 = ({Class G}, {T3,T4,T6,T7})
- Concept 8 = ({Class A, Class D, Class G}, {T6})
- Concept 9 = ({Class A, Class F}, {T5})
- Concept 10 = ({Class A, Class G}, {T3,T6})
- Concept 11 = ({Class D, Class F, Class G}, {T4})
- Concept 12 = (Top)={Class A, Class B, Class C, Class D, Class E, Class F, Class G}, {})

Totally, 12 concepts are computed including 'Top' and 'Bottom' concept.

2.3.3 Building a concept lattice

After computing all concepts, we build a concept lattice according to the relations between concepts. As mentioned in section 1.3, we can draw a concept lattice diagram using sub-concept and super-concept relation. For example, 'Concept8' is the super-concept of 'Concept1', 'Concept4', 'Concept7' and 'Concept10' because the set of attributes of 'Concept8' is included in the set of attributes of the above 4 concepts and the set of objects of the concepts is included in the set of objects of 'Concept8'.

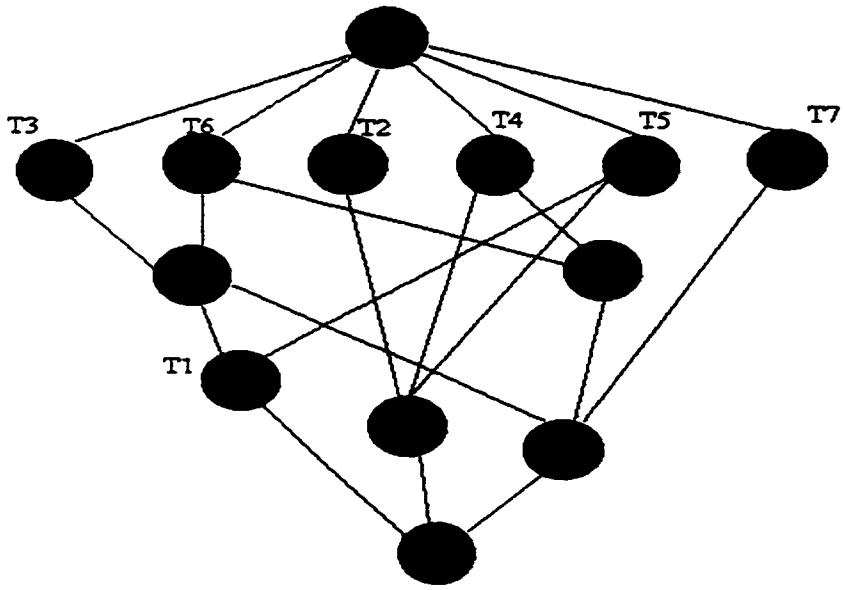

Figure 1: The concept lattice for Table 1

Figure 1 is the concept lattice for in this example. The edge

means that there exits the relation of super-concept and sub-concept between two concepts. From the above concept lattice, we have a structured reusable component library.

In this stage, we need to put label to each concept for our retrieval procedure.

Label	Labeled concept
T1: (int, int)->int	Concept1 = ({Class A}, {T1, T3, T5, T6})
T2: ( )->int	Concept2 = ({Class B, Class F},{T2})
T3: ( )->void	Concept3 = ({Class A, Class C, Class G}, {T3})
T4: (a)->void	Concept11= ({Class D, Class F, Class G}, {T4})
T5: (a)->a	Concept9 = ({Class A, Class F}, {T5})
T6: ( )->bool	Concept8 = ({Class A, Class D, Class G}, {T6})
T7: (int)->int	Concept5 = ({Class E, Class G}, {T7})

Table 2: The types and the labeled concepts

Table 2 shows the labeled concepts for each type. In a concept lattice, all sub-concepts of this labeled concept for the type have this type. This labeled concept has the largest number of extent for that type. So, when we try to find the classes corresponding to the query type, we locate the labeled concept

of that type because this concept shows us the largest number of classes. For example, 'Concept8' in Figure 1 is the labeled concept of the type 'T6'. The all sub-concepts of 'Concept8' – 'Concept1', 'Concept4', 'Concept7', and 'Concept10'- have the type 'T6' but 'Concept8' has the largest number of classes that has the type 'T6'.

2.3.4 Making a query

Now we can make a query for our retrieval. The retrieval algorithm is as follows: [27]

If the query type is defined in the class library

Then

- **Retrieve all extents of the labeled concepts for this query type.**
- **Change the query type with the general type and send the query again with this query type.**

Else

- **Change the query type with the general type instead of our original query type**

In the algorithm, the retrieval system receives the query from a user. The query is composed of the type that might be defined in the class that a user wants to retrieve. The retrieval system checks the query type to see if this type is defined in the class library. In the case that the type is defined, it just finds a labeled concept for the type in the concept lattice and returns the all extents of that concept to the user. In addition to this procedure, the retrieval system modifies the user's query type into the more general type. As explained in Chapter 2.1, the template function in the C++ can be used instead of the specific type of functions. The retrieval system also finds the labeled concept of this changed query type and returns the results to the user. In the case that the type that a user sends is not defined, the retrieval system changes the user's query type and the remaining procedure is the same as the previous case.

Example 1

Let us make a query '`()->int`' and send it to our retrieval system. The retrieval system searches the labeled concept for this type. Fortunately, the retrieval system finds the 'Concept 2' and returns all extents-'Class B', 'Class F'- to the user. The

next step is to change this query type with general type 'a'. So, the new query type is '() -> a'. And send this query again. This type is not defined in this sample library.

Example 2

Let us make a query "`(String)->String`" and send it to our retrieval system. At this time, this query type is not defined in our class library. But, we can use template function that has more general type instead of this query type.

First changed query is '`(a)->String`'. This type is not defined in the sample library.

Second changed query is '`(a)->a`'. This query type is defined in our class library. The labeled concept for this query type is 'Concept 9'. 'Class A' and 'Class F' are returned to the user.

The last changed query is '`(a)->b`'. 'b' is a poly-type like 'a'. The difference from second query type '`(a)->a`' is that its input type and the return type is not the same. This is more general type than the second changed query type.

CHAPTER 3 A PROTOTYPE SYSTEM

We have discussed class retrieval based on types using Formal Concept Analysis in previous chapters. We developed a prototype system based on this retrieval method to implement this retrieval procedure in java. The prototype system consists of two parts. The first part is a database system. The database system has information about the types of functions that each class has. This is like formal contexts in Formal Concept Analysis. The second part is an application part. It connects to database system and calculates concepts using the information stored in the database. More details in the prototype system are explained below.

3.1 Overview of the Prototype System

In our sample class library, 31 classes and 34 types are defined. The library consists of data structure class in C++ such as stack, queue, linked-list and tree etc. Several template classes are also included in this library to demonstrate how those template classes are usefully exploited for our retrieval process. The number of classes defined in this library is relatively small, but this is enough to demonstrate how our retrieval process works. The type `'(int,int)->int'` means that this function receive

two 'int' type arguments and returns 'int' type result. In the notation '(a)->a', 'a' is a poly-type. In template class, we have a flexible type that can be changed into any-type when we create this template class with specific type. Poly-type gives us a more general way to match with our query.

3.1.1 Implementation language

Java is chosen as an implementation program language. The features of Java are as follows: [13]

Platform independence

Once we write code and compile it, it runs everywhere regardless of any operating system environment such as win32, unix machine.

Information hiding

A programmer can know how to use the objects but the programmer has no knowledge of how it works internally. He can just sends a parameter and receives the results. "Implementation details are hidden within the objects themselves"[13].

Inheritance

This is a way of software reusability in which new classes are created from existing classes by extending their attributes and behaviors. The type of sub-class can be considered as that of super-class.

Garbage Collection

Garbage Collection can release memory when it is no longer used. Garbage Collector is a low-level priority thread. This makes a programmer to manage a memory without a problem such as memory leak.

3.1.2 Database tool

In this thesis, MySQL is chosen as a database tool. MySQL is a multi-user, multi-threaded SQL database server. MySQL is a client/server program in which a server daemon is mysqld.

3.1.3 JDBC

JDBC (Java Database Connectivity) is used to connect the implementation program to the Database server (MySql). Java supports package 'java.sql' that provides classes and interfaces for manipulating relational databases [13].

3.2 Implementation of the Prototype System

3.2.1 The class library

The number of classes in this sample library is 31. The classes in this library are as follows. Here, the signature of each function is shown.

Mammal

The functions defined in this class are:

```
void Speak()
void Sleep()
```

Cat

This class is subclass of class 'Mammal'. The function defined in this class is:

```
void Speak()
```

Dog

This class is subclass of class 'Mammal'. The functions defined in this class are:

```
void Tail()
void Food()
```

```
void Speak()
```

Horse

This class is subclass of class 'Mammal'. The function defined in this class is:

```
void Speak()const
```

Pig

This class is subclass of class 'Mammal'. The functions defined in this class are:

```
void Speak()const
```

Shape

The functions defined in this class are:

```
long Get_Area()
long Get_Perim()
void Draw()
```

Retangle

This class is subclass of class 'Shape'. The functions defined in this class are:

```
long Get_Area()
long Get_Perim()
int Get_Length()
```

```
int Get_Width()
void Draw();
```

Circle

This class is subclass of class 'Shape'. The functions defined in this class are:

```
long Get_Area()
long Get_Perim()
void Draw();
void setLength(int k)
```

Square

This class is subclass of class 'Rectangle'. The functions defined in this class are:

```
long Get_Perim()
void set_Length(int k)
```

Counter

The functions defined in this class are:

```
short GetVal()const
void SetVal(short x)
```

AATree

The functions defined in this class are:

```
void insert(Comparable x)->Insert x
void remove(Comparable x)->Remove x
Comparable find(Comparable x)->Return item that matches x
Comparable findMin()->Return smallest item
Comparable findMax()->Return largest item
boolean isEmpty()->Return true if empty
void makeEmpty()->Remove all items
void printTree()->Print tree in sorted order
```

AVLH

The functions defined in this class are:

```
void insert(int x)->Insert x
void remove(int x)->Remove x
int find(int x)->Return item that matches x
int findMin()->Return smallest item
int findMax()->Return largest item
boolean isEmpty()->Return true if empty
void makeEmpty()->Remove all items
void printTree()->Print tree
```

BinaryTree

The functions defined in this class are:

```
void insert(int x)->Insert x
void remove(int x)->Remove x
```

```
int find(int x)->Return item that matches x
int findMin()->Return smallest item
int findMax()->Return largest item
boolean isEmpty()->Return true if empty
void makeEmpty()->Remove all items
void printTree()->Print tree
```

DisSet

The functions defined in this class are:

```
int find(int x)
int find(int x)
void union(int root1, int root2)
```

IntCell

The functions defined in this class are:

```
int read( )
void write(int x)
```

List

The functions defined in this class are:

```
bool isEmpty()
void makeEmpty()
ListItr zeroth()
ListItr first()
```

```
void insert(const int & x, const ListItr & p)
ListItr find(const int & x)
ListItr findPrevious(const int & x)
void remove(const int & x);
```

ListItr

The functions defined in this class are:

```
bool is_End()
void advance()
const Object & retrieve()
```

MemoryCell

The functions defined in this class are:

```
const int & read() const
void write(const int & x)
```

PairHeap

The functions defined in this class are:

```
PairNode & insert(Comparable x)->Insert x
deleteMin(Comparable x)->Remove
Comparable findMin()->Return smallest item
bool isEmpty()->Return true if empty
bool isFull()->Return true if empty
void makeEmpty()->Remove all items
```

```
void decreaseKey(PairNode p, int Val)
```

Queue

The functions defined in this class

```
void enqueue(int x) --> Insert x
void dequeue( ) --> Return and remove item
int getFront( ) --> Return item
bool isEmpty( ) --> Return true if empty
bool isFull( ) --> Return true if full
void makeEmpty( ) --> Remove all items
```

Random

The functions defined in this class

```
int randomInt( )
int random1( )
int randomInt(int low, int high)
```

SplayTree

The functions defined in this class

```
void insert(Comparable x) --> Insert x
void remove(Comparable x) --> Remove x
Comparable find(Comparable x) --> Return item that matches x
Comparable findMin( ) --> Return smallest item
Comparable findMax( ) --> Return largest item
```

boolean isEmpty( ) --> Return true if empty

void makeEmpty( ) --> Remove all items

void printTree( ) --> Print tree

Stack

The functions defined in this class

void push(int x) --> Insert x

void pop( ) --> Remove most recently inserted item

int top( ) --> Return most recently inserted item

int topAndPop( ) --> Return and remove item

bool isEmpty( ) --> Return true if empty; else false

bool isFull( ) --> Return true if full; else false

void makeEmpty( ) --> Remove all items

TemplateAVL

The functions defined in this class

void insert(Comparable x) --> Insert x

void remove(Comparable x) --> Remove x

Comparable find(Comparable x) --> Return item that matches x

Comparable findMin( ) --> Return smallest item

Comparable findMax( ) --> Return largest item

boolean isEmpty( ) --> Return true if empty

void makeEmpty( ) --> Remove all items

void printTree( ) --> Print tree

TemplateBinary

The functions defined in this class

```
void insert(Comparable x) --> Insert x
void remove(Comparable x) --> Remove x
Comparable find(Comparable x) --> Return item that matches x
Comparable findMin( ) --> Return smallest item
Comparable findMax( ) --> Return largest item
boolean isEmpty( ) --> Return true if empty
void makeEmpty( ) --> Remove all items
void printTree( ) --> Print tree
```

TemplateHash

The functions defined in this class

```
void insert(Object x)--> Insert x
void remove(Object x) --> Remove x
Hashable find(Object x) --> Return item that matches x
void makeEmpty( ) --> Remove all items
```

TemplateHashTable

The functions defined in this class

```
void insert(Comparable x) --> Insert x
void remove(Comparable x) --> Remove x
Hashable find(Comparable x) --> Return item that matches x
```

```
void makeEmpty( ) --> Remove all items
```

TemplateLinkedList

The functions defined in this class

```
boolean isEmpty( ) --> Return true if empty
void makeEmpty( ) --> Remove all items
ListItr zeroth( ) --> Return position
ListItr first( ) --> Return first position
void insert(Object x) --> Insert x
void remove(Object x) --> Remove x
ListItr find(Object x) --> Return  x
ListItr findPrevious(Object x)
```

TemplateList

The functions defined in this class

```
bool isEmpty( )
void makeEmpty( )
ListItr<Object> zeroth( )
ListItr<Object> first( )
void insert(const Object & x, const ListItr<Object> & p)
ListItr<Object> find(const Object & x) const;
ListItr<Object> findPrevious(const Object & x) const;
void remove(const Object & x);
```

TemplateMemoryCell

The functions defined in this class

```
const Object read( ) const
void write(const Object x)
```

TemplateQueue

The functions defined in this class

```
void enqueue(Object x)
void dequeue( )
Object getFront( )
bool isEmpty( )
bool isFull( )
void makeEmpty( )
```

TemplateStackh

The functions defined in this class

```
void push(Object x)
void pop( )
Object top( )
Object topAndPop( )
bool isEmpty( )
bool isFull( )
void makeEmpty( )
```

3.2.2 Type information of all functions in the sample library

In this sample class library, 34 function types are defined. 'T' denotes type or signature of each function and following number is an index of types. In a database system, the formal context table is organized with the column name of this T_i(i denotes the index number of types). 'a' denotes a poly-type that can be substituted with any type. All type information in the sample class library is as follows:

```
T1:  ( ) -> void
T2:  ( ) -> long
T3:  int
T4:  ( ) ->int
T5:  (int,int) -> void
T6:  (int) -> void
T7:  USHORT
T8:  (short) -> void
T9:  ( ) -> short
T10: (a) ->void
T11: (a) -> a
T12: ( ) -> a
T13: ( ) -> bool
T14: *AANode
```

T15: a

T16: (a) -> int

T16: *AVLNode

T17: (int) -> int

T18: BinaryNode

T19: Vector

T20: ListNode

T21: ( ) -> ListItr

T22: (int&,ListItr&) -> void

T23: (int&) -> ListItr

T24: (int&) -> void

T25: ( ) -> Object&

T26: ( ) -> int&

T27: (a) -> PairNode

T28: (PairNode,int) -> void

T29: (int,int) ->int

T30: (a) -> Hashable

T31: HashedObj

T32: (a&,ListItr&) -> void

T33: (a&) -> ListItr

T34: a& -> void

3.2.3 The context table

From 3.2.1 and 3.2.2, we can build a context table for this example.

Class Name	T1	T1-a	T1-b	T2	T2-a	T3	T3-a	T3-b	T4	T4-a
Mammal	✓	✓								
Cat	✓	✓								
Dog	✓	✓	✓							
Pig	✓	✓								
Horse	✓	✓								
Shape	✓			✓	✓					
Retangle	✓			✓	✓	✓	✓		✓	✓
Circle	✓			✓	✓	✓	✓	✓		
Square	✓			✓	✓	✓	✓		✓	✓
Counter										
AATree	✓	✓								
AVLH	✓	✓				✓			✓	✓
BinaryTree	✓	✓				✓			✓	✓
Disset										
IntCell						✓			✓	
List	✓									
ListItr	✓									
MemoryCell						✓				
PairHeap	✓									
Queueh	✓	✓				✓	✓	✓	✓	
Random						✓			✓	✓
SplayTree	✓	✓								
Stack	✓	✓				✓			✓	✓
TAVL	✓	✓								
Tbinary	✓	✓								
Thash	✓									
ThashTable	✓					✓				
Tlist	✓									
TmemoryCell										
Tqueue	✓	✓				✓	✓	✓		
Tstackh	✓	✓				✓				

Figure 2: The context table for the sample class library

Class Name	T5	T6	T6-a	T7	T8	T9	T10	T10-a	T11	T12
Mammal										
Cat										
Dog										
Pig										
Horse										
Shape										
Retangle										
Circle		✓								
Square	✓	✓								
Counter				✓	✓	✓				
AATree							✓	✓	✓	✓
AVLH		✓	✓							
BinaryTree		✓	✓							
Disset	✓									
IntCell		✓								
List										
ListItr										
MemoryCell										
PairHeap							✓			✓
Queueh		✓								
Random										
SplayTree							✓	✓	✓	✓
Stack		✓								
TAVL							✓	✓	✓	✓
Tbinary							✓	✓	✓	✓
Thash							✓	✓		
ThashTable							✓	✓		
Tlist										
TmemoryCell							✓			✓
Tqueue							✓			✓
Tstackh							✓			✓

Figure 2: The context table for the sample class library (continued)

Class Name	T12-a	T13	T13-a	T14	T15	T16	T17	T17-a	T18	T18-a
Mammal										
Cat										
Dog										
Pig										
Horse										
Shape										
Retangle										
Circle										
Square										
Counter										
AATree		✓		✓	✓					
AVLH		✓				✓	✓			
BinaryTree		✓					✓		✓	
Disset							✓	✓		
IntCell										
List		✓								
ListItr		✓								
MemoryCell										
PairHeap		✓	✓							
Queueh		✓	✓							
Random										
SplayTree	✓	✓			✓				✓	✓
Stack		✓	✓							
TAVL		✓			✓	✓				
Tbinary	✓	✓			✓				✓	
Thash										
ThashTable										
Tlist		✓								
TmemoryCell					✓					
Tqueue		✓	✓							
Tstackh	✓	✓	✓							

Figure 2: The context table for the sample class library (continued)

Class Name	T19	T20	T21	T21-a	T22	T23	T23-a	T24	T25	T26
Mammal										
Cat										
Dog										
Pig										
Horse										
Shape										
Retangle										
Circle										
Square										
Counter										
AATree										
AVLH										
BinaryTree										
Disset	✓									
IntCell										
List		✓	✓	✓	✓	✓	✓	✓		
ListItr		✓							✓	
MemoryCell								✓		✓
PairHeap										
Queueh										
Random										
SplayTree										
Stack	✓									
TAVL										
Tbinary										
Thash	✓									
ThashTable	✓									
Tlist		✓	✓	✓						
TmemoryCell										
Tqueue	✓									
Tstackh	✓									

Figure 2: The context table for the sample class library (continued)

Class Name	T27	T28	T29	T30	T31	T32	T33	T33a	T34
Mammal									
Cat									
Dog									
Pig									
Horse									
Shape									
Retangle									
Circle									
Square									
Counter									
AATree									
AVLH									
BinaryTree									
Disset									
IntCell									
List									
ListItr									
MemoryCell									
PairHeap	✓	✓							
Queueh									
Random			✓						
SplayTree									
Stack									
TAVL									
Tbinary									
Thash				✓	✓				
ThashTable				✓	✓				
Tlist						✓	✓	✓	✓
TmemoryCell									
Tqueue									
Tstackh									

Figure 2: The context table for the sample class library (continued)

Figure 2 shows the context table of the sample class library.

3.2.4 Concepts computed in the sample library

The concepts that are calculated in our sample library as follows:

- **Concept1** = Object(MAMMAL, CAT, DOG, HORSE, PIG, CIRCLE, AATREE, AVLH, BINARYTREE, QUEUEH, SPLAYTREE, STACK, TAVL, TBINARY, TQUEUE, TSTACK) Attribute(T1, T1a)
- **Concept2** = Object(DOG) Attribute(T1, T1a, T1b)
- **Concept3** = Object(SHAPE, RETANGLE, CIRCLE, SQUARE) Attribute(T1, T2, T2a)
- **Concept4** = Object(RETANGLE, SQUARE) Attribute(T1, T2, T2a, T3, T3a, T4, T4a)
- **Concept5** = Object(CIRCLE) Attribute(T1, T1a, T2, T2a, T3, T3a, T3b)
- **Concept6** = Object(SQUARE) Attribute(T1, T2, T2a, T3, T3a, T4, T4a, T5, T6)
- **Concept7** = Object(COUNTER) Attribute(T7, T8, T9)
- **Concept8** = Object(AATREE) Attribute(T1, T1a, T10, T10a, T11, T12, T13, T14, T15)

- **Concept9** = Object(AVLH) Attribute(T1, T1a, T3, T4, T4a, T6, T6a, T13, T16, T17)

- **Concept10** = Object(BINARYTREE) Attribute(T1, T1a, T3, T4, T4a, T6, T6a, T13, T17, T18)

- **Concept11** = Object(DISSET) Attribute(T5, T17, T17a, T19)

- **Concept12** = Object(SQUARE, AVLH, BINARYTREE, INTCELL, QUEUEH, STACK) Attribute(T3, T4, T6)

- **Concept13** = Object(LIST) Attribute(T1, T16, T20, T21, T21a, T22, T23, T23a, T24)

- **Concept14** = Object(LISTITR) Attribute(T1, T13, T20, T25)

- **Concept15** = Object(MEMORYCELL) Attribute(T3, T24, T26)

- **Concept16** = Object(PAIRHEAP) Attribute(T1, T3, T10, T12, T13, T13a, T27, T28)

- **Concept17** = Object(QUEUEH) Attribute(T1, T1a, T3, T3a, T3b, T4, T6, T13, T13a, T19)

- **Concept18** = Object(RANDOM) Attribute(T3, T4, T4a, T29)

- **Concept19** = Object(SPLAYTREE) Attribute(T1, T1a, T10, T10a, T11, T12, T12a, T13, T15, T18, T18a)

- **Concept20** = Object(STACK) Attribute(T1, T1a, T3, T4, T4a, T6, T13, T13a, T19)

- **Concept21** = Object(TAVL) Attribute(T1, T1a, T10, T10a, T11, T12, T12a, T13, T15, T16)

- **Concept22** = Object(SPLAYTREE, TBINARY) Attribute(T1, T1a, T10, T10a, T11, T12, T12a, T13, T15, T18)

- **Concept23** = Object(THASH, THASHTABLE) Attribute(T1, T10, T10a, T19, T30, T31)

- **Concept24** = Object(THASHTABLE) Attribute(T1, T3, T10, T10a, T19, T30, T31)

- **Concept25** = Object(TLIST) Attribute(T1, T13, T20, T21, T21a, T32, T33, T33a, T34)

- **Concept26** = Object(AATREE, SPLAYTREE, TAVL, TBINARY, TMEMORYCELL) Attribute(T10, T12, T15)

- **Concept27** = Object(TQUEUE) Attribute(T1, T1a, T3, T3a, T3b, T10, T12, T13, T13a, T19)

- **Concept28** = Object(TSTACK) Attribute(T1, T1a, T3, T10, T12, T12a, T13, T13a, T19)

- **Concept29** = Object(MAMMAL, CAT, DOG, HORSE, PIG, SHAPE, RETANGLE, CIRCLE, SQUARE, AATREE, AVLH, BINARYTREE, LIST, LISTITR, PAIRHEAP, QUEUEH, SPLAYTREE, STACK, TAVL, TBINARY, THASH, THASHTABLE, TLIST, TQUEUE, TSTACK) Attribute(T1)

- **Concept30** = Object(RETANGLE, CIRCLE, SQUARE) Attribute(T1, T2, T2a, T3, T3a)

- **Concept31** = Object(RETANGLE, SQUARE, AVLH, BINARYTREE, STACK) Attribute(T1, T3, T4, T4a)

- **Concept32** = Object(RETANGLE, SQUARE, AVLH, BINARYTREE, INTCELL, QUEUEH, RANDOM, STACK) Attribute(T3, T4)

- **Concept33** = Object(RETANGLE, CIRCLE, SQUARE, AVLH, BINARYTREE, INTCELL, MEMORYCELL, PAIRHEAP, QUEUEH, RANDOM, STACK, THASHTABLE, TQUEUE, TSTACK) Attribute(T3)

- **Concept34** = Object(RETANGLE, CIRCLE, SQUARE, AVLH, BINARYTREE, PAIRHEAP, QUEUEH, STACK, THASHTABLE, TQUEUE, TSTACK) Attribute(T1, T3)

- **Concept35** = Object(RETANGLE, SQUARE, QUEUEH) Attribute(T1, T3, T3a, T4)

- **Concept36** = Object(RETANGLE, SQUARE, AVLH, BINARYTREE, RANDOM, STACK) Attribute(T3, T4, T4a)

- **Concept37** = Object(RETANGLE, CIRCLE, SQUARE, QUEUEH, TQUEUE) Attribute(T1, T3, T3a)

- **Concept38** = Object(CIRCLE, AVLH, BINARYTREE, QUEUEH, STACK, TQUEUE, TSTACK) Attribute(T1, T1a, T3)

- **Concept39** = Object(CIRCLE, QUEUEH, TQUEUE) Attribute(T1, T1a, T3, T3a, T3b)

- **Concept40** = Object(SQUARE, AVLH, BINARYTREE, STACK) Attribute(T1, T3, T4, T4a, T6)

- **Concept41** = Object(SQUARE, DISSET) Attribute(T5)

- **Concept42** = Object(SQUARE, QUEUEH) Attribute(T1, T3, T3a, T4, T6)

- **Concept43** = Object(AATREE, AVLH, BINARYTREE, QUEUEH, SPLAYTREE, STACK, TAVL, TBINARY, TQUEUE, TSTACK) Attribute(T1, T1a, T13)

- **Concept44** = Object(AATREE, AVLH, BINARYTREE, LISTITR, PAIRHEAP, QUEUEH, SPLAYTREE, STACK, TAVL, TBINARY, TLIST, TQUEUE, TSTACK) Attribute(T1, T13)

- **Concept45** = Object(AATREE, PAIRHEAP, SPLAYTREE, TAVL, TBINARY, TQUEUE, TSTACK) Attribute(T1, T10, T12, T13)

- **Concept46** = Object(AATREE, SPLAYTREE, TAVL, TBINARY) Attribute(T1, T1a, T10, T10a, T11, T12, T13, T15)

- **Concept47** = Object(AATREE, SPLAYTREE, TAVL, TBINARY, THASH, THASHTABLE) Attribute(T1, T10, T10a)

- **Concept48** = Object(AATREE, SPLAYTREE, TAVL, TBINARY, TQUEUE, TSTACK) Attribute(T1, T1a, T10, T12, T13)

- **Concept49** = Object(AVLH, BINARYTREE) Attribute(T1, T1a, T3, T4, T4a, T6, T6a, T13, T17)

- **Concept50** = Object(AVLH, BINARYTREE, DISSET) Attribute(T17)

- **Concept51** = Object(AVLH, LIST, TAVL) Attribute(T1, T16)

- **Concept52** = Object(AVLH, BINARYTREE, PAIRHEAP, QUEUEH, STACK, TQUEUE, TSTACK) Attribute(T1, T3, T13)

- **Concept53** = Object(AVLH, BINARYTREE, QUEUEH, STACK) Attribute(T1, T1a, T3, T4, T6, T13)

- **Concept54** = Object(RETANGLE, SQUARE, AVLH, BINARYTREE, QUEUEH, STACK) Attribute(T1, T3, T4)

- **Concept55** = Object(SQUARE, AVLH, BINARYTREE, QUEUEH, STACK) Attribute(T1, T3, T4, T6)

- **Concept56** = Object(AVLH, BINARYTREE, STACK) Attribute(T1, T1a, T3, T4, T4a, T6, T13)

- **Concept57** = Object(AVLH, TAVL) Attribute(T1, T1a, T13, T16)

- **Concept58** = Object(AVLH, BINARYTREE, QUEUEH, STACK, TQUEUE, TSTACK) Attribute(T1, T1a, T3, T13)

- **Concept59** = Object(BINARYTREE, SPLAYTREE, TBINARY) Attribute(T1, T1a, T13, T18)

- **Concept60** = Object(DISSET, QUEUEH, STACK, THASH, THASHTABLE, TQUEUE, TSTACK) Attribute(T19)

- **Concept61** = Object(LIST, LISTITR, TLIST) Attribute(T1, T20)

- **Concept62** = Object(LIST, MEMORYCELL) Attribute(T24)

- **Concept63** = Object(LIST, TLIST) Attribute(T1, T20, T21, T21a)

- **Concept64** = Object(LISTITR, TLIST) Attribute(T1, T13, T20)

- **Concept65** = Object(PAIRHEAP, QUEUEH, STACK, TQUEUE, TSTACK) Attribute(T1, T3, T13, T13a)

- **Concept66** = Object(AATREE, PAIRHEAP, SPLAYTREE, TAVL, TBINARY, THASH, THASHTABLE, TQUEUE, TSTACK) Attribute(T1, T10)

- **Concept67** = Object(PAIRHEAP, THASHTABLE, TQUEUE, TSTACK) Attribute(T1, T3, T10)

- **Concept68** = Object(AATREE, PAIRHEAP, SPLAYTREE, TAVL, TBINARY, TMEMORYCELL, TQUEUE, TSTACK) Attribute(T10, T12)

- **Concept69** = Object(PAIRHEAP, TQUEUE, TSTACK) Attribute(T1, T3, T10, T12, T13, T13a)

- **Concept70** = Object(QUEUEH, STACK) Attribute(T1, T1a, T3, T4, T6, T13, T13a, T19)

- **Concept71** = Object(QUEUEH, STACK, THASH, THASHTABLE, TQUEUE, TSTACK) Attribute(T1, T19)

- **Concept72** = Object(QUEUEH, STACK, THASHTABLE, TQUEUE, TSTACK) Attribute(T1, T3, T19)

- **Concept73** = Object(QUEUEH, TQUEUE) Attribute(T1, T1a, T3, T3a, T3b, T13, T13a, T19)

- **Concept74** = Object(QUEUEH, STACK, TQUEUE, TSTACK) Attribute(T1, T1a, T3, T13, T13a, T19)

- **Concept75** = Object(SPLAYTREE TAVL, TBINARY) Attribute(T1, T1a, T10, T10a, T11, T12, T12a, T13, T15)

- **Concept76** = Object(SPLAYTREE, TAVL, TBINARY, TSTACK) Attribute(T1, T1a, T10, T12, T12a, T13)

- **Concept77** = Object(AATREE, PAIRHEAP, SPLAYTREE, TAVL, TBINARY, THASH, THASHTABLE, TMEMORYCELL, TQUEUE, TSTACK) Attribute(T10)

- **Concept78** = Object(THASH, THASHTABLE, TQUEUE, TSTACK) Attribute(T1, T10, T19)

- **Concept79** = Object(THASHTABLE, TQUEUE, TSTACK) Attribute(T1, T3, T10, T19)

- **Concept80** = Object(TQUEUE, TSTACK) Attribute(T1, T1a, T3, T10, T12, T13, T13a, T19)

When we need to denote a type more than once, we add 'a', 'b', 'c'. For example, in the expression 'Concept2 =({C3}, {T1, T1a, T1b})', we denote type 'T1' as 'T1a' and 'T1b'. This is because we need to distinguish the column name in our database. So, 'T1', 'T1a', 'T1b' are all the same type.

3.2.5 A labeled concept for each type

Through the above procedure, we get labeled concepts for each type.

Type	Labeled Concept
T1: ( ) -> void	Concept 29
T2: ( ) -> long	Concept 3
T3: int	Concept 33
T4: ( ) -> int	Concept 32
T5: (int,int) -> void	Concept 41
T6: (int) -> void	Concept 12
T7: USHORT	Concept 7
T8: (short) -> void	Concept 7
T9: ( ) -> short	Concept 7
T10: (a) -> void	Concept 76
T11: (a) -> a	Concept 46
T12: ( ) -> a	Concept 67

Figure 3: The labeled concept for each type

T13: ( ) -> bool	Concept 44
T14: *AANode	Concept 8
T15: a	Concept 26
T16: *AVLNode	Concept 51
T17: (int) -> int	Concept 50
T18: BinaryNode	Concept 59
T19: Vector	Concept 60
T20: ListNode	Concept 61
T21: ( ) -> ListItr	Concept 63
T22: (int&,ListItr&) -> void	Concept 13
T23: (int&) -> ListItr	Concept 13
T24: (int&) -> void	Concept 70
T25: ( ) -> Object&	Concept 14
T26: ( ) -> int&	Concept 15
T27: (a) -> PairNode	Concept 16
T28: (PairNode,int) -> void	Concept16
T29: (int,int) -> int	Concept 18
T30: (a) -> Hashable	Concept 23
T31: Hashedobj	Concept 23
T32: (a&,ListItr&) -> void	Concept 25
T33: (a&) -> ListItr	Concept 25
T34: (a&) -> void	Concept 25

Figure 3: The labeled concept for each type (continued)

Figure 3 shows the labeled concept for each type. The labeled concept for a type has the largest number of extents that has this type. So, when a user send a query to the prototype system, the system finds a labeled concept and return all extents in the concept.

3.2.6 The concept lattice

After computing concepts, we get a concept lattice.

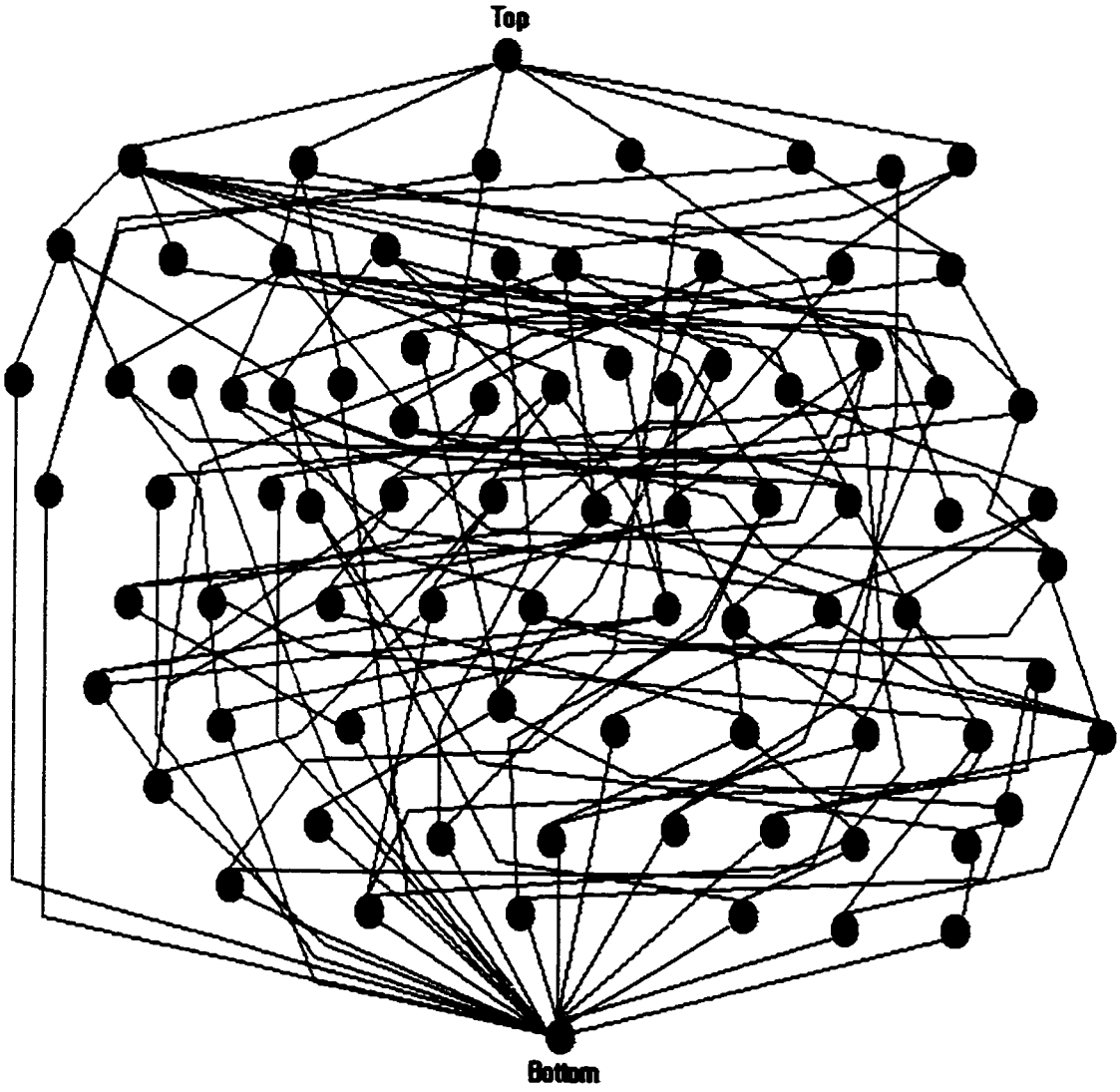

Figure 4: The concept lattice

Figure 4 shows the concept lattice for our sample class library.

3.3 Examples Using the Prototype System

3.3.1 Example 1

If the user wants to retrieve the class that has the types of '( ) -> void', 'int', '(String) -> String', '( ) -> long' then the retrieval steps are as follows.

- Step1: Sending a query that has the type to the retrieval system.
- Step2: The retrieval system finds a labeled concept of that type and returns all extents of that concept to the user.
- Step3: The retrieval system changes a query type with a general type - In this example, the general type is 'a' or 'b'- and send the query again.
- Step4: The user repeats the step from 1 to 3.

A user first sends the query '( ) -> void' to the prototype system and it shows the user all classes that have this type. Next, a user sends the second query for 'int' and the prototype system shows all classes that satisfy the second query 'int' and the first query '( ) -> void'. Theses steps are repeated until the user finishes queries. In this example, we have four query types. So, the prototype system shows the common classes that match

with all query types provided by the user. The detail implementation of the above example is as follows.

Making a query for the type '( ) -> void'

The first query in this example is '( ) -> void'. A user types the query type in the upper text field and then press the enter key.

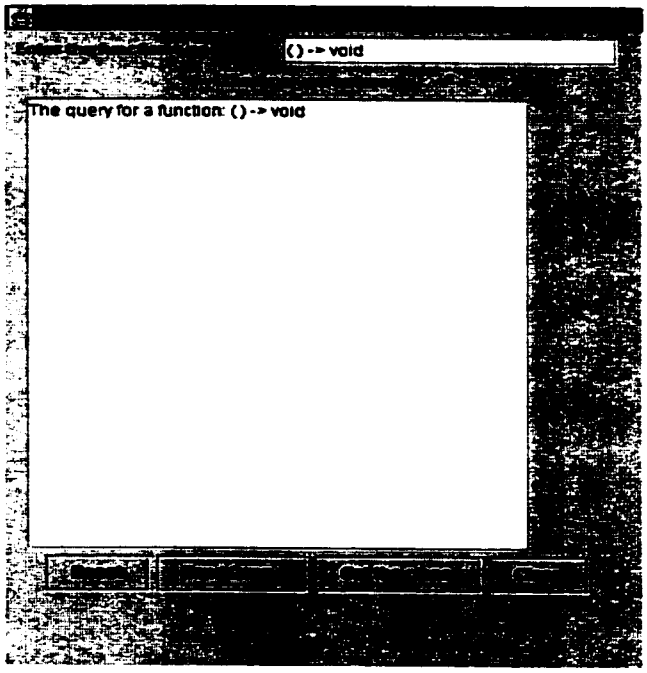

Figure 5: Making a query of '( ) -> void'

Figure 5 shows that the query type is '( ) -> void' in the text area.

Sending the first query to the system

When we press the button 'SendQuery', this action sends the above input type to the retrieval system to check if there are classes that have this type of function.

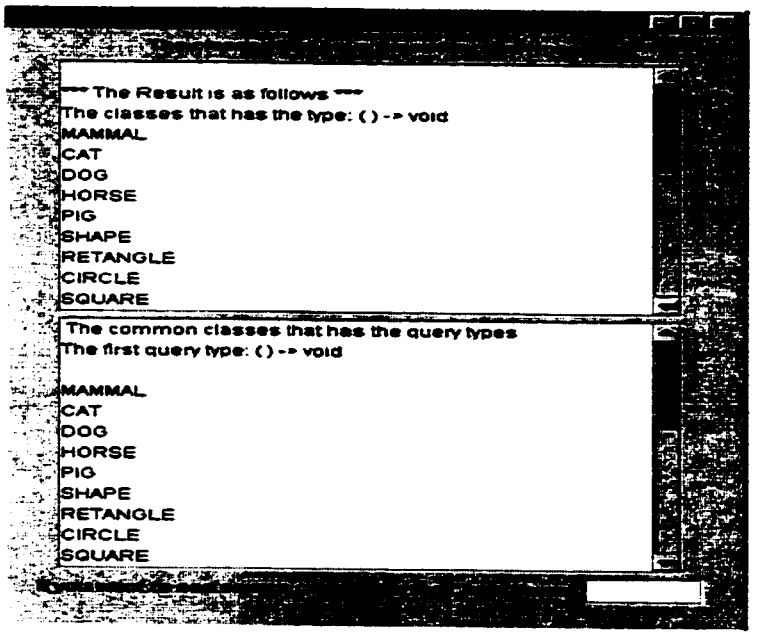

Figure 6: The classes that have the type '( ) -> void'

Figure 6 shows the result when we press the button 'SendQuery'.

The labeled concept for the type '( ) -> void' is the 'Concept29' in the concept lattice shown in 4.2.5. The prototype system returns all classes have the function type '( )

-> void'. The upper text area shows results for the query '( ) -> void'.

Sending the second query 'int' to the system

The second function query type is 'int'. A user enters this type and sends it to the retrieval system.

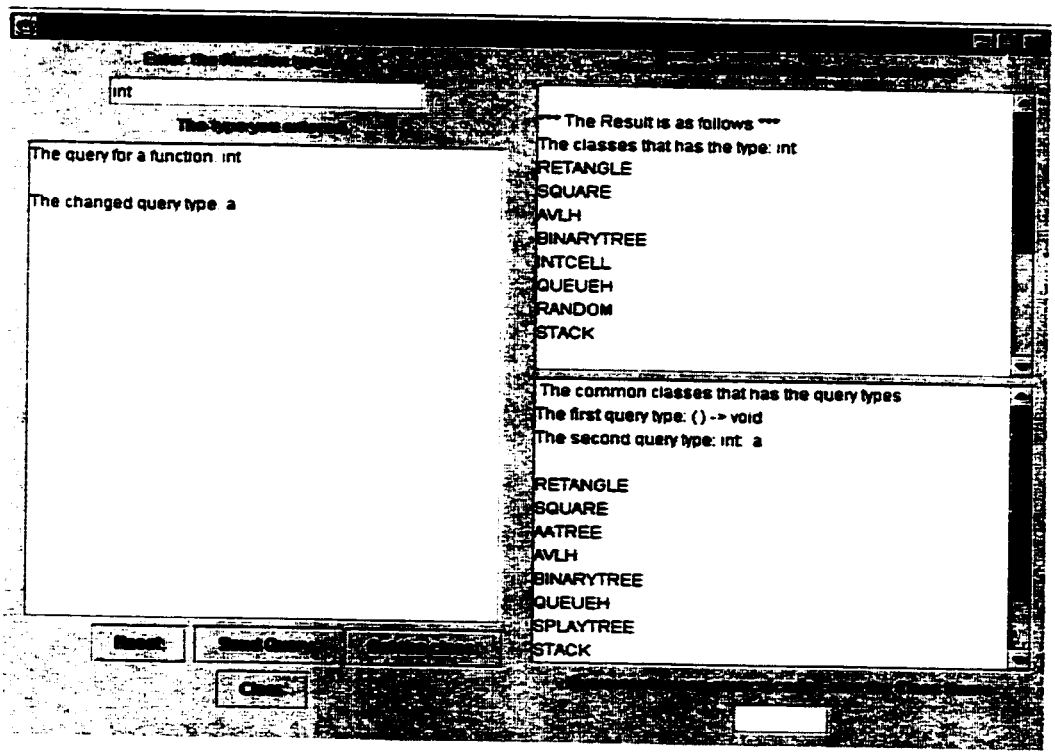

Figure 7: Sending the second query to the system.

Figure 7 shows the result of the second query. The difference from the first function query '( ) -> void' is that the retrieval system sends another query by changing the type 'int' to the general type 'a'.

As explained above, the retrieval system then shows all extents of the labeled concept for the type 'int' and for the type 'a' because the general type 'a' can be used for the type 'int'.

In the left text area, there are two query types. The first is 'int' that is provided by a user and the second is 'a' that is generated by the prototype system. As explained before, the general type 'a' can be used as a reusable component instead of the type 'int'. So, The right-upper text area shows the classes that have the type 'int' and the classes that has the type 'a'. The labeled concept of the type 'int' is the 'Concept33' and the labeled concept of the type 'a' is the 'Concept26' in the concept lattice of this sample class library. The right-bottom text area shows the classes that have the types '( ) -> void' and ('int' or 'a'). 10 classes are shown to the user for two queries in the test area.

Sending the third query '(String) -> String' to the system

The third query is '(String) -> String'. Like the above steps, a user sends this query to the system. In this case, we also can see that the system tries to change the user's query type to the more general type.

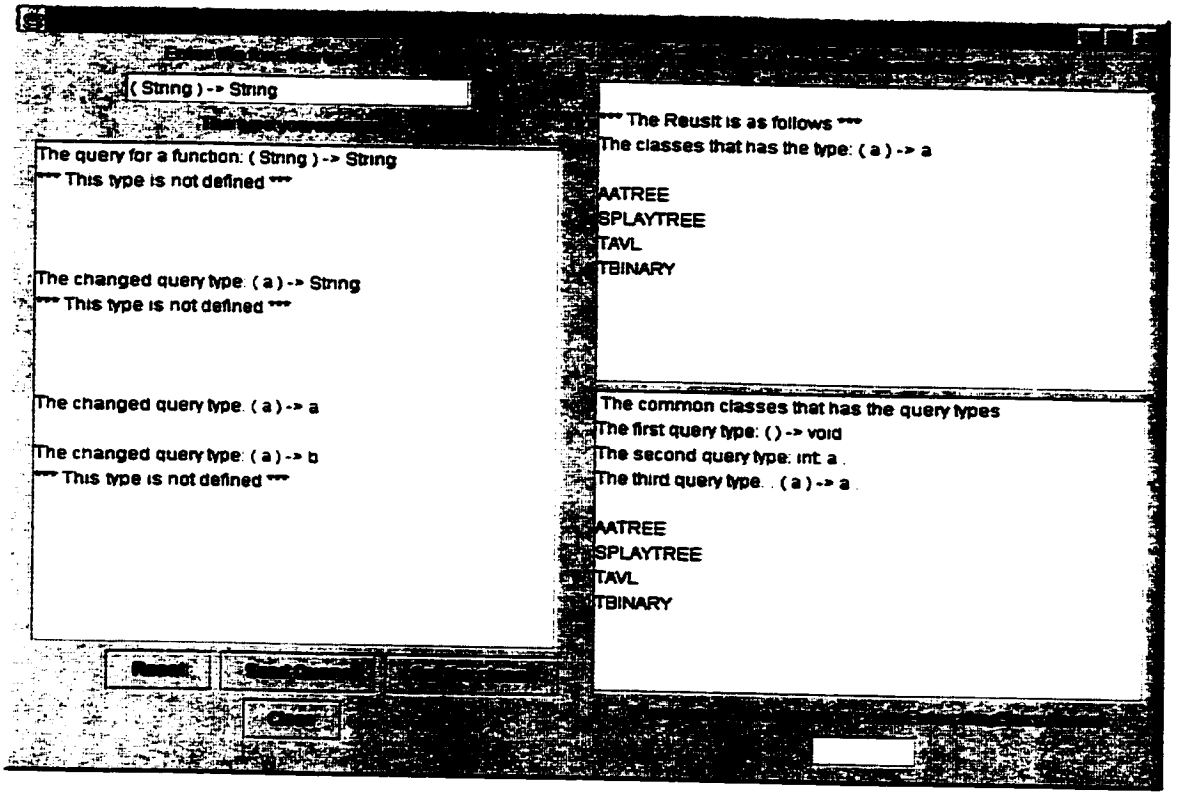

Figure 8: Sending the third query to the system

Figure 8 shows the result after we send a third query '(String) -> String'. In Figure 8, the type '(String) -> String is not defined in the sample class library. So, the retrieval system changes this type and tries to send new queries to the system.

In the left text area, the type '(a) -> a' is defined in the library. We can see the result in the right-upper text area that has the type '(a) –a'. Finally, the right-bottom text area in Figure 8 shows us the classes that have three types that we have sent as a query.

Sending the fourth query type '( ) -> long' to the system

Our last query type is '(short) -> void'.

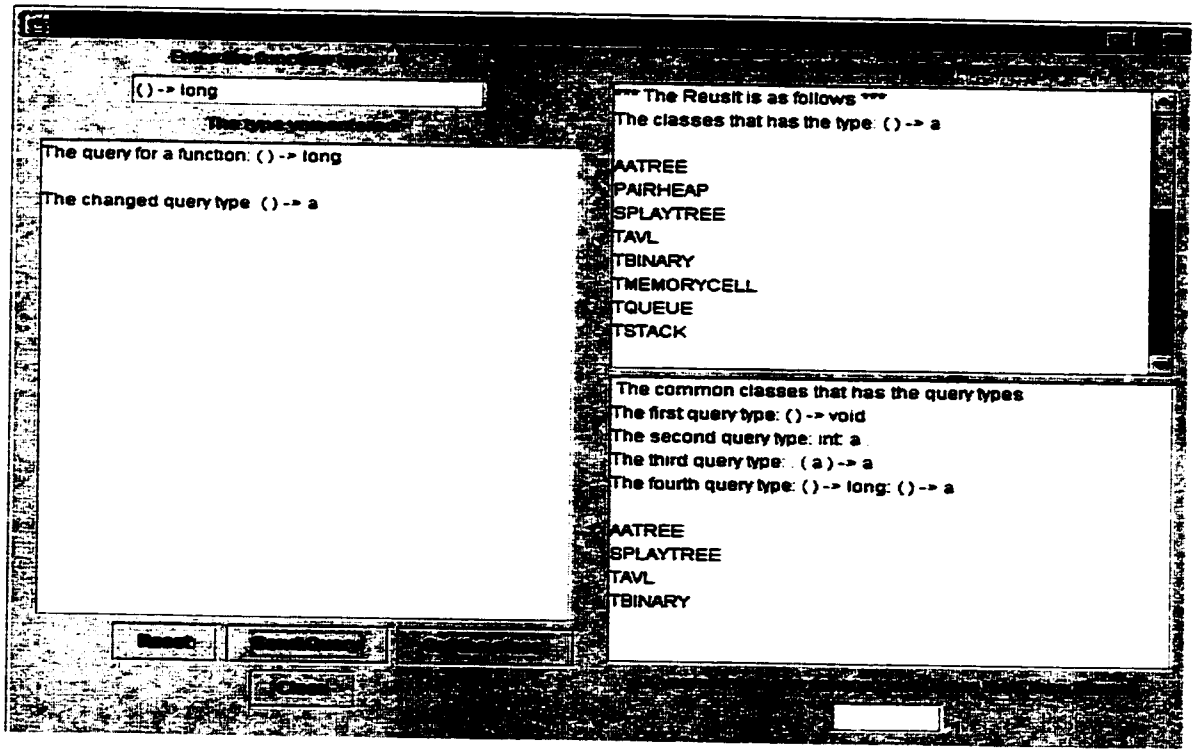

Figure 9: Sending the fourth query to the system

Figure 9 shows the result for this query. Like above queries, the system changes the type '( ) -> long' into '( ) -> a' and send a query again, we get the result in the right-bottom text area. The classes 'AATREE', 'SPLAYTREE', 'TAVL', 'TBINARY' can be used for our queries.

3.3.2 Example 2

In this example, a user wants to retrieve the classes that have the type 'int', '( ) -> int' and '( ) -> &Object'. A user sends a query one after another. The first query is the type 'int'.

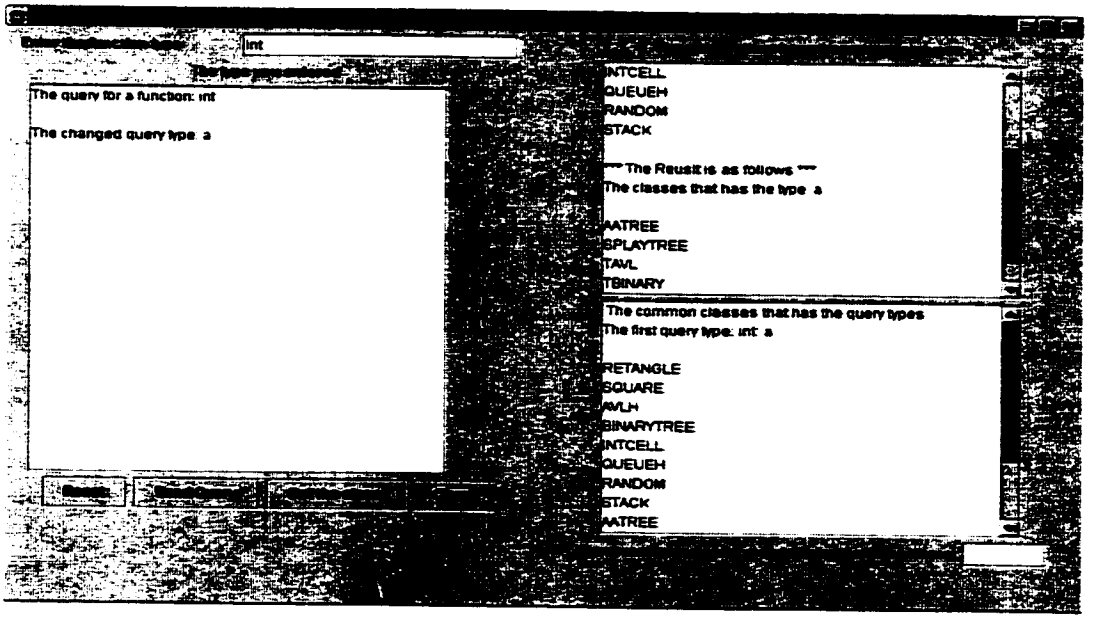

Figure 10: Sending the query of the type 'int'

Figure 10 shows the result of the query to the user. The labeled concept of this type is the 'Concept33'. And the changed query type made by the prototype system is the type 'a' that has the labeled concept 'Concept26'.

The prototype system shows the all extents of 'Concept33' and 'Concept26'.

The second query type is '( ) -> int'. This type is also changed into the general type '( ) -> a' by the prototype system after it finds all classes that has the type '( ) -> int'.

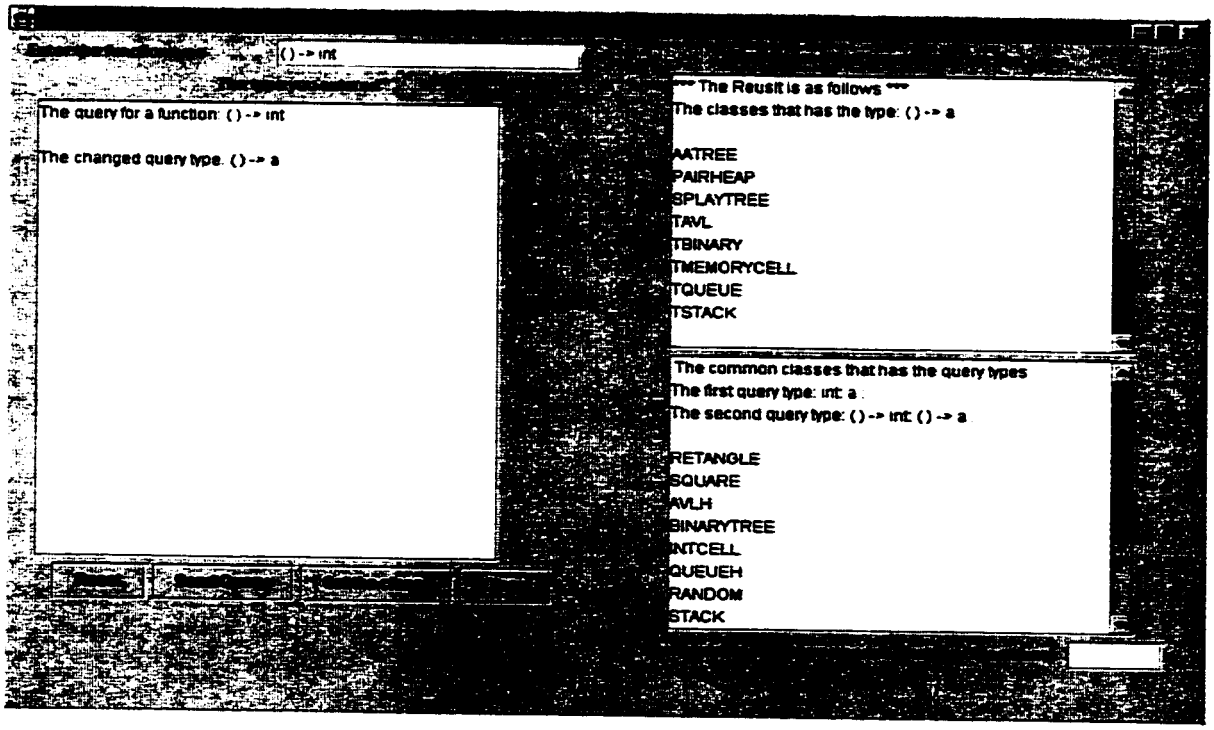

Figure 11: Sending the query of the type '( ) -> int'

Figure 11 shows the result of the query '( ) -> int'. The labeled concept of the type '( ) -> int' is the 'Concept32' and the labeled concept of the type '( ) -> a' is the 'Concept67'.

In the left-bottom text area, we can see the class names that have the first query type and the second query type.

The third query type is '( ) -> &Object'. The process is same as the previous ones.

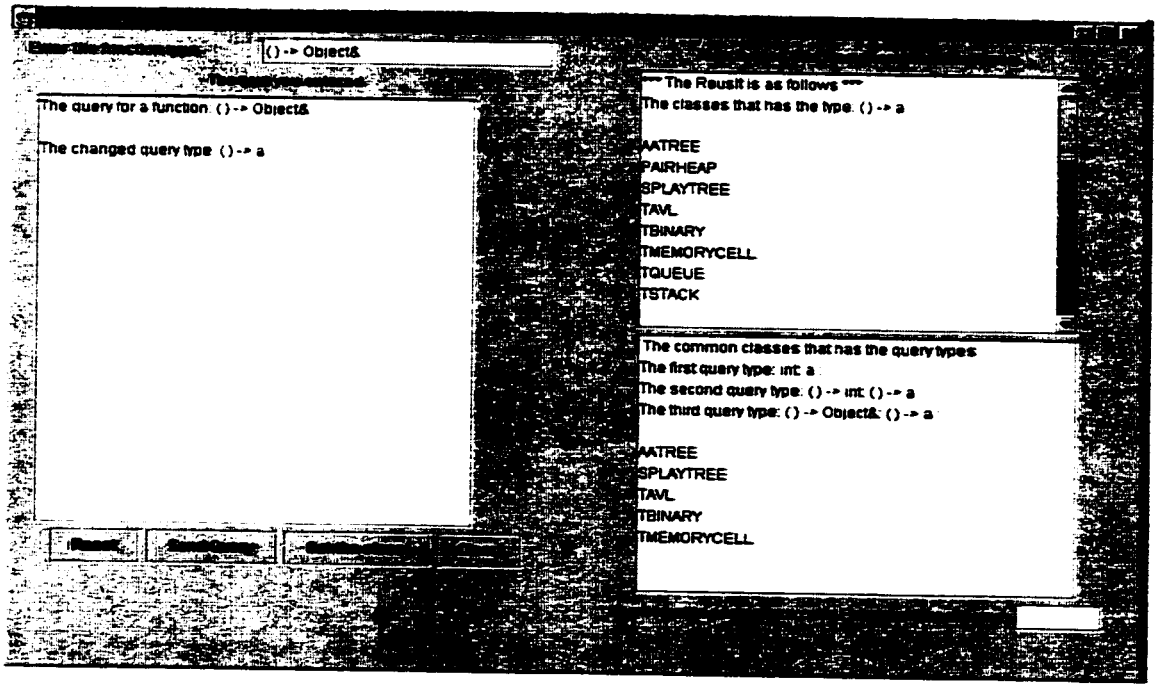

Figure 12: Sending the query of the type '( ) -> &Object'

Figure 12 shows the result of this query. The labeled concept of the type '( ) -> &Object is the 'Concept14' and the labeled concept of the changed query type '( ) -> a' is the 'Concept67'.

CHAPTER 4 CONCLUSION AND FUTURE WORK

Conclusion

In this thesis, we have proposed a method to retrieve classes in C++ based on type information. The type of a class is collection of type of each function and variables defined in the class. In our method, we have constructed the concept lattice for a class library with type information. This concept lattice reduces the search time. Based on the proposed method, we have developed a prototype system for C++ class retrieval. If the classes that match exactly with the query do not exist in the class library, then the prototype system changes the user's query to a more general type, and retrieves more generic template classes. Several examples for demonstrating our retrieval work are presented.

Future work

Type-based approach is very simple and retrieving time is very short compared to other retrieval methods. But, its accuracy is relatively low because it does not consider the meaning of components.

More practical retrieval methods to solve the above deficiency could be developed by incorporating other retrieval approaches such as execution-based retrieval or specification-based retrieval.

REFERENCES

[1] A. Zaremski and J. Wing, Signature Matching, *A Key to Reuse*, In Proceedings of ACM SIGSOFT Symposium on Foundation of Software Engineering, pp.182-190, 1993.

[2] A. Zaremski and J. Wing, *Signature matching, a tool for using software libraries*, In Proceedings of ACM Transactions on Software Engineering and Methodology (TOSEM), 4(2), 1995.

[3] A. Zaremski and J. Wing, *Specification matching of software components*, In Proceedings of the 3rd ACM SIGSOFT Symposium on the Foundations of Software Engineering, pp.333-369, 1995.

[4] B. Fischer, M. Kievernagel and G. Snelting, *Deduction-Based Software Component Retrieval*, In Proceedings of IJCAI-95 Workshop on Formal Approaches to the Reuse of Plans, Proofs, and Programs, Montreal, Canada, 1994.

[5] B. Fischer, *Specification-based browsing of software component libraries*, In Proceedings of the 13th ASE, pp.74-83, 1998.

[6] B. Ganter, R. Wille, Applied lattice theory, *Formal Concept Analysis*, http://www.math.tu-dresden.de/~ganter/concept.ps.

[7] B. Ganter and R. Wille, Formal Concept Analysis: Mathematical Foundations, Springer-Verlag, 1999.

[8] B. Barnes and T. B., Bollinger, *Making reuse cost-effective*, IEEE Software. pp.13-24, 1991.

[9] C. Lindig, *Concept-Based Component Retrieval*. In Proceedings of IJCAI-95 Workshop on Formal Approaches to the Reuse of Plans, Proofs, and Programs, pp.21-25, 1995.

[10] C. Lindig and G. Snelting, *Assessing Modular Structure of Legacy Code Based on Mathematical Concept Analysis*. In Proceedings of International Conference on Software Engineering, pp. 349-359, 1997.

[11] C. Runciman and L. Toyn, *Retrieving reusable software components by polymorphic type*. Journal of Functional programming, pp.191-211, 1991.

[12] G. Snelting and F. Tip, *Reengineering class hierarchies using concept analysis*, In Proceedings of Foundations of Software Engineering, pp.99-110, 1998.

[13] H. M. Deitel and P. J. Deitel, Java How to program, 3rd Edition, 2000.

[14] Isoda, *Experiences of a Software Reuse Project, Journal of Systems and Software*, pp.171-186, 1995.

[15] J. Baldo, J. Moore and D. Rine, *Software Reuse Standards*, Journal of StandardView 5(2), 1997.

[16] J. Jeng and B. Cheng, *A formal approach to using more general components*, In Proceedings of the 9th Knowledge-Based Software Engineering Conference, pp. 90-97, 1994.

[17] J. Jeng and B. Cheng, *Specification Matching for Software Reuse: A Foundation*, In Proceedings of the ACM Symposium on Software Reuse, pp.97-105, 1995.

[18] K. Tam, A semantic-based approach to retrieving imperative programs, Master's thesis, University of Windsor, 1997.

[19] M. L. Griss, *The Economics of Software Reuse*, In Proceedings of OOPSLA'91, pp.264-270, 1991.

[20] M. Rittri, *Retrieving library functions by unifying types modulo linear isomorphism*, In Programming Methodology Group Report 66, Chalmers University of Technology and University of Goteborg, Goteborg, Sweden, 1992.

[21] M. Rittri, *Retrieving library identifiers via equational matching of types*, In Proceedings of International Conference on Automated Deduction, pp.603-617, 1990.

[22] M. Siff and T. Reps, *Identifying modules via concept analysis*, In Proceedings of the International Conference on Software Maintenance, ICSM97, pp. 170-179, 1997.

[23] N. An and Y. Park, *A Structured Approach to Retrieving Functions by Types*, In Proceedings of ACM International Conference on Functional Programming (ICFP), pp.344, 1998.

[24] P. Funk, A. Lewien and G. Snelting, *Algorithms for Concept Lattice Decomposition and their Application*, Journal of Informatik -Bericht Rep.95-09, 1995.

[25] Q. An, Retrieving function component from a reuse library, Master's thesis, University of Windsor, 1998.

[26] Y. Park, *Polymorphic function retrieval via Formal Concept Analysis*, In Proceedings of the ACM/IACM International Conference on Computer Science (ICCS), pp. 165-170. 1999.

[27] Y. Park, *Software retrieval by samples using concept analysis*, Journal of Systems and Software, 54(3): pp. 179-183, 2000.

[28] W. B. Frakes, *Information retrieval and software reuse*, In Proceedings of the 12th Anniversary International ACMSIGIR Conference on Research and Development in Information Retrieval, pp. 251-256, 1989.

APPENDIX: SOURCE CODE FOR THE PROTOTYPE SYSTEM

```
import java.util.*;


class Concept
{

  public Vector v1_Of_Concept;
  public  Vector v2_Of_Concept;
  private Vector common_attribute;
  private Vector temp,temp1;
  Vector available_types;
  Vector Type_Label;
  Vector remove_item,real_child;
  String Result="";
  private int index1,dummy,dummy1,num_of_ob;
  Vector to_Parent,to_Child,remove_parent;
  Vector real_parent;
  String Name_Of_Concept="";
  String[] tem_Child1,tem_Child2;
  int label=0;

  public Concept(Vector v1_Con,Vector v2_Con,int num)
   {

            v1_Of_Concept=new Vector();
            v2_Of_Concept=new Vector();

            v1_Of_Concept=v1_Con;
            v2_Of_Concept=v2_Con;
            num_of_ob=v1_Of_Concept.size();
            label=v2_Of_Concept.size();
            index1=num;


            to_Parent=new Vector();
            to_Child=new Vector();
            remove_item=new Vector();
            real_child=new Vector();
            available_types=new Vector();
            Name_Of_Concept="Concept"+num;
            Type_Label=new Vector();

   }


  public void get_available_types()
  {
           Concept c;
           for(int i=0;i<to_Child.size();i++)
           {
            c=(Concept)to_Child.elementAt(i);

            if(c.Type_Label.size() != 0)
              available_types.addElement(c.Type_Label);
            else
             dummy++;

           }

  }


public void toString2()
 {

  Result+="Concept"+index1+" = Object(";
          for(int i=0;i<v1_Of_Concept.size();i++)
```

```
            }
              Result+=(String)v1_Of_Concept.elementAt(i)+" ";
            }

            Result+=")  Attribute(";

            for(int i=0;i<v2_Of_Concept.size();i++)
            {
               Result+=(String)v2_Of_Concept.elementAt(i)+" ";
            }

            Result+=")";

            System.out.println(Result+" label= "+label+" num_of_object"+num_of_ob);

  }

 public boolean check_attribute(Vector v_check)
 {
    common_attribute=new Vector();
    boolean flag=false;

    for(int i=0;i<v_check.size();i++)
        {
            if (isIncluded((String)v_check.elementAt(i))==true)
             {
               flag=true;
               common_attribute.addElement(v_check.elementAt(i));
             }
            else
               dummy1++;

   }
    return flag;
 }

 public boolean isIncluded(String str)
 {
     boolean flag1=false;

     for(int i=0;i<v2_Of_Concept.size();i++)
       {
          if(str.equals((String)v2_Of_Concept.elementAt(i))==true)
            flag1=true;
          else
            dummy1++;
       }
   return flag1;
 }

 public boolean isIncluded1(String str)
 {
     boolean flag3=false;

     for(int i=0;i<v1_Of_Concept.size();i++)
      {
        if(str.equals((String)v1_Of_Concept.elementAt(i))==true)
            flag3=true;
        else
            dummy1++;
      }
  return flag3;
 }


public boolean isIncluded2(String str)
 {
     boolean flag3=false;

    for(int i=0;i<v2_Of_Concept.size();i++)
     {
```

```
        if(str.equals((String)v2_Of_Concept.elementAt(i))==true)
            flag3=true;
        else
            dummy1--;
      }
    return flag3;
  }

  public boolean check_con5(Vector v_s)
  {
      temp1=new Vector();
      temp1=v_s;
      boolean flag9=true;

      for(int i=0;i<temp1.size();i--)
       {
         if(isIncluded((String)temp1.elementAt(i))==true)
            dummy1--;
         else
            flag9=false;
       }
    return flag9;
  }


  public boolean is_Same5(Vector v_s1)
  {
     Vector tt=new Vector();
     tt=v_s1;
     boolean flag12;

     if((v2_Of_Concept.size())!=(tt.size()))
           return false;
     else
      {
            flag12=true;

            for(int i=0;i<tt.size();i--)
            {

               if(isIncluded((String)tt.elementAt(i))==true)
                     dummy1--;
               else flag12=false;

            }
      }

   return flag12;
  }


public Vector get_common_Attribute()
  {
    temp=common_attribute;
           common_attribute=null;
           return temp;
  }


  public boolean isParent(Concept c_a)
  {

     if(v1_Of_Concept.size()<=c_a.v1_Of_Concept.size())
           return false;

     else
      {
            boolean flag_a=true;
```

```
            for(int i=0;i<c_a.vl_Of_Concept.size();i++)
            {

              if(isIncluded1((String)(c_a.vl_Of_Concept.elementAt(i))))
               {
                 dummy1++;
               }
              else
               {
                 flag_a=false;
                 break;
               }

            }

            return flag_a;
        }
    }

  public void make_Parent(Concept c_b)
  {
    Concept this_Concept;
    this_Concept=c_b;
    to_Parent.addElement(this_Concept);
  }

  public void make_Child(Concept c_c)
  {
    Concept this1_Concept;
    this1_Concept=c_c;
    to_Child.addElement(this1_Concept);
  }


 public void distinguish_parent()
 {
     remove_parent=new Vector();

    for(int i=0;i<to_Parent.size();i++)
     {

      for(int j=0;j<to_Parent.size();j++)
       {
            Concept c1;
            Concept c2;

            c1=(Concept)to_Parent.elementAt(i);
            c2=(Concept)to_Parent.elementAt(j);

            if(c1.isParent(c2))
             remove_parent.addElement(c1);

            else
             dummy++;

       }
     }

  }

public void real_children()
{
    tem_Child1=new String[this.to_Child.size()];
    tem_Child2=new String[this.remove_item.size()];

    for(int i=0;i<to_Child.size();i++)
     {
       tem_Child1[i]=((Concept)to_Child.elementAt(i)).Name_Of_Concept;
     }
```

```
    for(int j=0;j<remove_item.size();j++)
     {
       tem_Child2[j]=((Concept)remove_item.elementAt(j)).Name_Of_Concept;
     }

    for(int i=0;i<tem_Child1.length;i++)
           for(int j=0;j<tem_Child2.length;j++)
           {
              if(tem_Child1[i].equals(tem_Child2[j]))
              tem_Child1[i]="no";
           }
 }


public void real_children1()
{
   for(int i=0;i<to_Child.size();i++)
    {
          Concept cs_1=(Concept)to_Child.elementAt(i);
          boolean flag=true;

          for(int j=0;j<remove_item.size();j++)
          {
            Concept cs_2=(Concept)remove_item.elementAt(j);

            System.out.println(i+"vs"+j);
            System.out.println(cs_1.Name_Of_Concept+"vs"+cs_2.Name_Of_Concept);


            if(cs_1.Name_Of_Concept.equals(cs_2.Name_Of_Concept))
             {
                    flag=false;
                    break;
             }
          }

          if(flag==true)
            real_child.addElement(cs_1);
    }
 }

public void print_remove_item()
{
   System.out.println(this.Name_Of_Concept);

   for(int i=0;i<remove_item.size();i++)
    {
      Concept c=(Concept)remove_item.elementAt(i);
      System.out.println(c.Name_Of_Concept);
    }
    System.out.println();
    System.out.println();
}


public void real_parent1()
{
   real_parent=new Vector();

  for(int i=0;i<to_Parent.size();i++)
   {
          for(int j=0;j<remove_parent.size();j++)
          {
           Concept c1;
           Concept c2;

           c1=(Concept)to_Parent.elementAt(i);
           c2=(Concept)remove_parent.elementAt(j);
```

```
            if(c1.Name_Of_Concept.equals(c2.Name_Of_Concept))
              break;
            else
            {
             if(j==remove_parent.size())
                        real_parent.addElement(c1);
             else
               dummy++;
            }
           }
  }
}


public void print_parent()
{
   System.out.println("This is new Parentset"+Name_Of_Concept);
   System.out.println(remove_parent.size());
   System.out.println(real_parent.size());

  for(int i=0;i<real_parent.size();i++)
   {
     System.out.print(((Concept)real_parent.elementAt(i)).Name_Of_Concept);

   }
}

public void who_parents()
{
    System.out.println();
    System.out.println("Parents of "+Name_Of_Concept);

    for(int i=0;i<to_Parent.size();i++)
      System.out.print(((Concept)to_Parent.elementAt(i)).Name_Of_Concept );
    System.out.println();
}

public void who_children()
{
   System.out.println();
   System.out.println("Children of "+Name_Of_Concept);

   for(int i=0;i<to_Child.size();i++)
     System.out.print(((Concept)to_Child.elementAt(i)).Name_Of_Concept );

  System.out.println();
}

public String who_parents1()
{
    String temp="";
    temp="\nParents\n";

   for(int i=0;i<to_Parent.size();i++)
    {
      temp+=((Concept)to_Parent.elementAt(i)).Name_Of_Concept+"\n";
    }
  return temp;
}

public String who_children1()
{
    String temp="";
    temp=this.Name_Of_Concept+"'s Children\n";

    for(int i=0;i<tem_Child1.length;i++)
     {
      if(!tem_Child1[i].equals("no"))
        {
          temp+=tem_Child1[i];
          temp+="\n";
```

```
        }
      }
    return temp;
  }

}


import java.util.*;

class Concept_Generator
{

  Vector v_cG;
  Vector each_Attribute;
  Vector each_Object;
  Vector Concept_Array;
  int dummy;
  private int index_of_concept=1;
  int count_1;

  public Concept_Generator(Vector v_concept_Generator)
  {

    v_cG=new Vector();
    v_cG=v_concept_Generator;
    each_Attribute=new Vector();
    Concept_Array=new Vector();

    for(int i=0;i<v_cG.size();i++)
     {

          each_Attribute=(Vector)v_cG.elementAt(i);
          each_Object=pass_to_Database(each_Attribute);


          if(check_itcanbe(each_Attribute)==true)
            {
              Concept c=new Concept(each_Object,each_Attribute,index_of_concept);
              Concept_Array.addElement(c);
              System.out.println("Concept"+index_of_concept+" is created1");
               index_of_concept++;
            }
          else
              dummy++;
     }

    for(int i=0;i<Concept_Array.size();i++)
     {
           Concept c;
          c=(Concept)Concept_Array.elementAt(i);

     }

     int range_of_att=Concept_Array.size();
     get_concept_between_concepts();

  }


public Vector pass_to_Database(Vector eA)
 {
    DatabaseConnectionfornewTable d=new DatabaseConnectionfornewTable(eA);
    return d.retrieve_object();
 }


public void get_concept_between_concepts()
{
```

```
for(int i=0;i<Concept_Array.size();i++)
 {
        Concept c0;
        c0=(Concept)Concept_Array.elementAt(i);

        for(int j=i+1;j<Concept_Array.size();j++)
         {

           Concept c1;
           c1=(Concept)Concept_Array.elementAt(j);

            Vector t=new Vector();
            if((c1.check_attribute(c0.v2_Of_Concept))==true)
              {
               t=c1.get_common_Attribute();/*return type is Vector*/
              each_Object=pass_to_Database(t);

              if(check_itcanbe(t)==true)
                {
                 Concept d=new Concept(each_Object,t,index_of_concept);
                 Concept_Array.addElement(d);
                 System.out.println("Concept"+index_of_concept+" is created ");
                 index_of_concept++;
                 System.out.println("Go to recursion");
                get_concept_between_concept1(i,d);
                }
               else
                    dummy++;;
              }
             else
               dummy++;
             }
         }

 }

public void get_concept_between_concept1(int k,Concept cp)
{
    for(int j=0;j<k;j++)
         {
            Concept c1;
            c1=(Concept)Concept_Array.elementAt(j);

            Vector t=new Vector();
            if((c1.check_attribute(cp.v2_Of_Concept))==true)
              {
               t=c1.get_common_Attribute();/*return type is Vector*/
               each_Object=pass_to_Database(t);

              if(check_itcanbe(t)==true)
               {
                   Concept d=new Concept(each_Object,t,index_of_concept);
                   Concept_Array.addElement(d);
                   System.out.println("Concept"+index_of_concept+" is created ");
                   index_of_concept++;
                    System.out.println("Go to recursion");
                    get_concept_between_concept1(j,d);
               }
              else dummy++;;

              }
             else
              dummy++;

            }
}


public boolean check_itcanbe(Vector t)
```

```
{
    boolean flag5=true;
    Vector t2;
    t2=new Vector();
    t2=t;
    Concept f;

    for(int i=0;i<Concept_Array.size();i++)
      {
          f=(Concept)Concept_Array.elementAt(i);
          if((t.is_Same5(t2))==true)
           {
              flag5=false;
              break;
           }
          else
            dummy++;
      }
      return  flag5;

 }

 public void output()
 {
   Concept z;

   for(int i=0;i<Concept_Array.size();i++)
    {
          z=(Concept)Concept_Array.elementAt(i);
          z.toString2();
    }
 }


 public void output1()
 {
   Concept z;

   for(int i=0;i<Concept_Array.size();i++)
    {
      z=(Concept)Concept_Array.elementAt(i);

      for(int j=0;j<z.Type_Label.size();j++)
       {
          System.out.println("The type label of "+z.Name_Of_Concept+" : "+(String)z.Type_Label.elementAt(j));
       }
      z.get_available_types();
    }
 }

public void output2()
{
   Concept z;

   for(int i=0;i<Concept_Array.size();i++)
    {
      z=(Concept)Concept_Array.elementAt(i);
      System.out.println("The available types of "+z.Name_Of_Concept);

      for(int j=0;j<z.available_types.size();j++)
       {
          Vector kr=(Vector)z.available_types.elementAt(j);
          for(int k=0;k<kr.size();k++)
            System.out.print((String)kr.elementAt(k)+" ");
       }

    System.out.println();
    }
 }

public void output3()
```

```
 {
   Concept d;
  String st;

    for(int i=0;i<Concept_Array.size();i++)
     {
       d=(Concept)Concept_Array.elementAt(i);
       st=d.who_children1();
       System.out.println(st);
     }
 }

 public void output4()
  {
    Concept d;
     for(int i=0;i<Concept_Array.size();i++)
      {
       d=(Concept)Concept_Array.elementAt(i);
       d.print_remove_item();
      }
  }

public void get_real_child()
{
    Concept d;
     for(int i=0;i<Concept_Array.size();i++)
      {
       d=(Concept)Concept_Array.elementAt(i);
       d.real_children();
      }
}



public String get_concept(String str2)
 {
    Concept c;
    String labeled_Concept="";
     int max_size=0;
     int temp_size=0;

     for(int i=0;i<Concept_Array.size();i++)
      {
              c=(Concept)Concept_Array.elementAt(i);
              if(c.isIncluded(str2) == true)
               {
                 temp_size=c.vl_Of_Concept.size();
                 if (temp_size>max_size)
                   {
                     max_size=temp_size;
                     labeled_Concept=c.Name_Of_Concept;
                   }
               }
              else
               dummy++;

     }

     for(int i=0;i<Concept_Array.size();i++)
      {
              c=(Concept)Concept_Array.elementAt(i);
              if (labeled_Concept.equals(c.Name_Of_Concept))
               {
                  c.Type_Label.addElement(str2);
               }
              else
                      dummy++;
            }

          count_1++;
       return labeled_Concept;
```

```
  }

public void get_all_labeled_concept()
  {
     String str="T";
     String str1="";
     String str2="";
     String str3="";

     for(int i=1;i<35;i++)
       {
            str1=String.valueOf(i);
            str2=str+str1;

            str3=get_concept(str2);
            if(i==16)
                  System.out.println("!!!!!!! "+str3);
           System.out.println("The labeled Concept of"+str2+" is "+str3);
       }
  }

}


import java.util.*;
import java.sql.*;


public class contentOfEachRow
{

  int num_Of_column;
  Vector cv;
  Vector cv1;
  String str="";
  Vector result_Of_attribute;
  int dummy=0;
  int count_of_Attribute=0;


public contentOfEachRow(Vector v1,Vector v2)
  {
    num_Of_column=v1.size();
    cv=new Vector();
    result_Of_attribute=new Vector();
    cv=v1;
    cv1=v2;

  }

public void toString1()
 {
    System.out.print("The Object : {"+(String)cv.elementAt(0)+"}\t");
    System.out.print("The Attributes is : {");

    for(int i=1;i<num_Of_column;i++)
      {
       str=(String)cv.elementAt(i);
       if (str.equals("TRUE"))
         {
           count_of_Attribute++;
           String s=(String)cv1.elementAt(i);
           System.out.print("\t"+s);
           result_Of_attribute.addElement(s);
         }
       else dummy++;
      }

      System.out.println("}");
      System.out.println();
```

```
  }

 public Vector getAttribute()
  {
    return result_Of_attribute;
  }

}



import java.sql.*;
import java.util.*;

class DatabaseConnectionfornewTable {

 private Vector vofAtt;
 private Vector vofObj;
 private int count_Of_Att;


 public DatabaseConnectionfornewTable(Vector v_1)
  {

      vofAtt=new Vector();
      vofObj=new Vector();
      vofAtt=v_1;
      count_Of_Att=vofAtt.size();

     try {
          Class.forName("org.gjt.mm.mysql.Driver");
          } catch (ClassNotFoundException e) {
            System.out.println(e.getMessage());
            }

    try {
      String url = "jdbc:mysql://127.0.0.1:3306/test";
      String conditionOfTable="";

      for(int i=0;i<count_Of_Att;i++)
       {
          if(i==(count_Of_Att-1))
            {
              conditionOfTable+=(String)vofAtt.elementAt(i)+"='TRUE'";
              break;
            }
          conditionOfTable+=(String)vofAtt.elementAt(i)+"='TRUE'&&";
       }

      Connection con = DriverManager.getConnection(url);
      Statement stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery("select Name from ST45 where "+ conditionOfTable );

      while(rs.next()) {
         String Year  = rs.getString(1);
         vofObj.addElement(Year);
         }
    stmt.close();
    con.close();
    } catch(java.lang.Exception ex) {
      ex.printStackTrace();
    }
}


public Vector retrieve_object()
 {
  return vofObj;
 }
```

```
}

import java.util.*;

class Make_lattice
 {

   Vector m_lattice;
   int dummy2=0;


  public Make_lattice(Vector h)
   {
     m_lattice=new Vector();
     m_lattice=h;
     Concept c_1;
     Concept c_2;

      for(int i=0;i<m_lattice.size();i++)
         for(int j=0;j<m_lattice.size();j++)
           {
              c_1=(Concept)m_lattice.elementAt(i);
              c_2=(Concept)m_lattice.elementAt(j);

              if(c_2.isParent(c_1))
                {
                  c_1.make_Parent(c_2);
                  c_2.make_Child(c_1);
                 }
              else
                dummy2++;
            }

    for(int i=0;i<m_lattice.size();i++)
     {
       ((Concept)m_lattice.elementAt(i)).who_parents();
      ((Concept)m_lattice.elementAt(i)).who_children();
     }
   }

}

import java.util.*;

class Make_lattice1
 {
  private Vector m_lattice;
  private int dummy2=0;
  private Vector temp_1,temp_2,temp_3;
  Concept c_1;
  Concept c_2;
  Concept c_3;
  Concept c_4;

 public Make_lattice1(Vector h)
  {
    m_lattice=new Vector();
    m_lattice=h;

    for(int i=0;i<m_lattice.size();i++)
     {
      c_1=(Concept)m_lattice.elementAt(i);
      temp_1=c_1.to_Child;

      for(int j=0;j<c_1.to_Child.size();j++)
       {
         Concept c_2=(Concept)c_1.to_Child.elementAt(j);
         temp_2=c_2.to_Child;
```

```
            make_new_lattice(temp_1,temp_2);
          }
        }


      for(int i=0;i<m_lattice.size();i++)
       {
         ((Concept)m_lattice.elementAt(i)).who_parents();
        ((Concept)m_lattice.elementAt(i)).who_children();
       }
  }


 public void make_new_lattice(Vector c1,Vector c2)
  {
    Vector v1,v2;
    v1=c1;
    v2=c2;

    for(int l=0;l<v2.size();l++)
     {
       c_3=(Concept)v2.elementAt(l);
       for(int k=0;k<v1.size();k++)
        {
          c_4=(Concept)v1.elementAt(k);
          if(c_3.Name_Of_Concept.equals(c_4.Name_Of_Concept))
          c_1.remove_item.addElement(c_3);
        }
      }
  }

}


import java.sql.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;

public class TestFCA extends Frame
{
   private Connection connection;
   private contentOfEachRow[] cOER;
   private int count;

   public TestFCA()
   {
     try {
          Class.forName("org.gjt.mm.mysql.Driver");
        } catch (ClassNotFoundException e) {
          System.out.println(e.getMessage());
        }

     try {
          String url = "jdbc:mysql://127.0.0.1:3306/test";
          connection = DriverManager.getConnection(url);
          System.out.println("Connection is succeed");
          }
           catch(java.lang.Exception ex){
           ex.printStackTrace();
          }

    get_content_of_Table();
   }

 private void get_content_of_Table()
  {
    Statement statement;
    ResultSet r_Set;
    try{
```

```
        String query="SELECT * FROM ST45";
        statement=connection.createStatement();
        r_set=statement.executeQuery(query);
       show_table(r_set);
       statement.close();
       }catch(SQLException sqlex){
           System.err.println("Error in getTable");
           sqlex.printStackTrace();}
}


private void show_table( ResultSet rs) throws SQLException
 {
  boolean more_item=rs.next();

  if(! more_item){
    System.out.println("No record");
    return;
   }

   Vector rows=new Vector();

   try{
         ResultSetMetaData rsmd=rs.getMetaData();
        Vector contentOfcolumn=new Vector();

        for(int i=1;i<=rsmd.getColumnCount();i++)
          contentOfcolumn.addElement(rsmd.getColumnName(i));

        do{
            rows.addElement(getNextRow(rs,rsmd));
           }while(rs.next());

            count=rows.size();
            cOER=new contentOfEachRow[count];
            Vector w=new Vector();

             for(int i=0;i<count;i++)
              {
               w=(Vector)rows.elementAt(i);
               cOER[i]=new contentOfEachRow(w,contentOfcolumn);
              }

             for(int i=0;i<count;i++)
              {
               cOER[i].toString1();
              }

               Vector para_to_concept_generator=new Vector();

               for(int i=0;i<count;i++)
                 {
                   para_to_concept_generator.addElement(cOER[i].getAttribute());
                 }

          Concept_Generator CG=new Concept_Generator(para_to_concept_generator);
          CG.output();
          CG.get_all_labeled_concept();
          System.out.println();
          System.out.println();

           Make_lattice m_L=new Make_lattice(CG.Concept_Array);
           Make_lattice1 m_l1=new Make_lattice1(CG.Concept_Array);
           CG.output1();
           CG.output2();
           CG.get_real_child();
           CG.output3();
           CG.output4();

           User_Interface ss=new User_Interface(CG.Concept_Array,contentOfcolumn);
           Store_concept_inFile SciF=new Store_concept_inFile(CG.Concept_Array);
```

```
            }catch(SQLException sqlex){
              System.err.println("Error in display");
              sqlex.printStackTrace();
            }
    }

private Vector getNextRow(ResultSet rs,ResultSetMetaData rsmd) throws SQLException
 {
   Vector currentRow=new Vector();

   for(int i=1;i<=rsmd.getColumnCount();++i)
     switch(rsmd.getColumnType(i)){
           case Types.VARCHAR:
              currentRow.addElement(rs.getString(i));
              break;
           case Types.INTEGER:
              currentRow.addElement(new Long(rs.getLong(i)));
              break;
           }
           return currentRow;
 }

public void shutDown()
 {
   try{
       connection.close();
      }
      catch(SQLException sqlex){
      sqlex.printStackTrace();
      }
}


 public static void main(String args[])
  {
     final TestFCA fca=new TestFCA();
     fca.addWindowListener(
            new WindowAdapter(){
             public void windowClosing(WindowEvent e)
                                 {
                                        fca.shutDown();
                                        System.exit(0);
                                 }
                          }
               );
           }
  }


import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;

class User_Interface extends JFrame
{
 private JPanel panel1,panel2;
 private Vector con_arr;
 private Vector column_content;
 private JLabel label1,label2,label3,label4,label5;
 private JTextArea area,area1,area2,area3;
 private JList list,list1,list2;
 private JButton btn1,btn2,btn3,btn4,btn5;
 private JTextField tF1,tF2;
 String sp="";
 String Fun_Type="";
 private String buff1="";
 private String buff2="";
 private String str10="";
```

```
private Vector v_type;
private Vector temp_Concept;
private int dummy;
private Vector tem_class;
String[] common_class;
int number_of_function;
int index;
int check_one=0;
int first_length=0;
DataInputStream input;
String temp_class="";
String query_types="";
boolean add_condition;
int start;

public User_Interface(Vector k,Vector k1)
{

  con_arr=k;
  column_content=k1;
  panel1=new JPanel();
  panel1.setLayout(new FlowLayout());
  panel2=new JPanel();
  panel2.setLayout(new FlowLayout());
  start=0;
 common_class=new String[100];

  for(int i=0;i<100;i++)
   {
    common_class[i]="0";
   }
  number_of_function=0;
  index=0;
  label1=new JLabel("Enter the function type              ");
  label2=new JLabel(" The type you entered    ");
  label3=new JLabel("The classes that have these function types  ");
  label4=new JLabel("If you want to see source code Enter the Class Name ");
  label5=new JLabel("              ");
  tF1=new JTextField(20);
  tF2=new JTextField(6);
  area1=new JTextArea(20,30);
  area2=new JTextArea(12,30);
  area3=new JTextArea(12,30);
  btn1=new JButton("Reset");
  btn2=new JButton("Send Query");
  btn3=new JButton("Get the class ");
  btn4=new JButton("Clear");

  tF1.addActionListener(
            new ActionListener()
            {
                        public void actionPerformed(ActionEvent e)
                        {
                                    String sp1="";
                                    sp1="The query for a function: ";
                                    sp=tF1.getText();

                                    buff1=sp1+sp;
                          area1.setText(buff1);
                                    check_type(sp);

                                    System.out.println("The Fun_Type is "+Fun_Type);
                        }
            }
  );


btn4.addActionListener(
            new ActionListener()
            {
                        public void actionPerformed(ActionEvent e)
```

```
                    ;


                                area3.setText("");
                                temp_Concept=null;
                                number_of_function=0;
                                check_one=0;
                                first_length=0;
                                query_types="";
                                index=0;
                                 for(int i=0;i<100;i++)
                        ;
                            common_class[i]="0";
                        ;
                            ;
                ;
    );



btn1.addActionListener(
            new ActionListener()
            ;
                    public void actionPerformed(ActionEvent e)
                    ;
                            number_of_function--;
                            start=index;
                            tF1.setText("");
                            area1.setText("");
                            area2.setText("");
                            temp_class="";

                    ;
            ;
    );



btn3.addActionListener(
            new ActionListener()
            ;
                    public void actionPerformed(ActionEvent e)
                    ;
                     System.out.println("Num_Of_Type is "+number_of_function);
                     for(int i=0;i<index;i++)
                            System.out.println(common_class[i]);
                     String buff3="";
                     int temp;
                     for(int i=0;i<first_length;i++)
                     ;
                            temp=0;
                      for(int j=first_length;j<index;j++)
                      ;
                            if (common_class[i].equals(common_class[j]))
                            ;
                                    temp--;
                              if (temp==number_of_function-1)
                                    ;
                                            buff3+=common_class[i];
                                      buff3+="\n";
                                      break;
                                    ;
                            ;
                            else
                            ;

                                    dummy--;
                            ;
                      ;
                     ;
```

```
                area3.setText(buff3);
                }
        }
);



btn2.addActionListener(
        new ActionListener()
        {
                public void actionPerformed(ActionEvent e)
                {

                 if(number_of_function == 0)
                        query_types+="The first query type: ";
                 if(number_of_function == 1)
                        query_types+="The second query type: ";
                 if(number_of_function == 2)
                  query_types+="The third query type: ";
                 if(number_of_function == 3)
                  query_types+="The fourth query type: ";


                 int num_of_tokens;

           if(Fun_Type.equals("xxx"))
                        send_new_query1();

                 else
                 {
                 temp_Concept=new Vector();
                 query_types+=sp;

                 for(int i=0;i<con_arr.size();i++)
                   {
                        Concept c1;
                        c1=(Concept)con_arr.elementAt(i);

                    if(c1.isIncluded2(Fun_Type)==true)
                        temp_Concept.addElement(c1);
                   else
                         dummy++;

           }
           Concept lower_bound=null;
           System.out.println("The length of temp_Concept is "+temp_Concept.size());
           lower_bound=what_is_lower_bound_concept(temp_Concept);
           System.out.println(lower_bound.num_of_ob);
           temp_class+="\n*** The Result is as follows ***";
           temp_class+="\n";
           temp_class+="The classes that has the type: "+sp+"\n";

           for(int i=0;i<lower_bound.num_of_ob;i++)
            {
              temp_class+=(String)(lower_bound.v1_Of_Concept.elementAt(i));
              temp_class+="\n";
              if(check_we_can_add_common_class((String)(lower_bound.v1_Of_Concept.elementAt(i))) != false)
                {
                  common_class[index++]=(String)(lower_bound.v1_Of_Concept.elementAt(i));
                }
            }

           if(number_of_function==0)
            {
             first_length=lower_bound.num_of_ob;
            }
           System.out.println("The concept name is "+lower_bound.Name_Of_Concept);
           System.out.println(temp_class);
```

```
                add_condition=false;
                send_new_query1();
                add_condition=true;
               }

              query_types+="\n";
              get_the_common_class();
             }
            }
        );

    panel1.add(label1);
    panel1.add(tF1);
    panel1.add(label2);
    panel1.add(new JScrollPane(area1));
    panel1.add(label5);
    panel1.add(btn1);
    panel1.add(btn2);
    panel1.add(btn3);
    panel1.add(btn4);
    panel2.add(label3);
    panel2.add(new JScrollPane(area2));
    panel2.add(new JScrollPane(area3));
    panel2.add(label4);
    panel2.add(tF2);

   Container c=getContentPane();
   c.setLayout(new GridLayout(1,2,5,5));
   c.add(panel1);
   c.add(panel2);
   setTitle("FCA Program AHN,HANSOO University of Windsor Computer Science");
   setSize(900,625);
   show();
 }

Concept  what_is_lower_bound_concept(Vector v)
 {
   int Max=0;
   Concept labe_Concept=null;

    for(int i=0;i<v.size();i++)
     {
        int temp;
        Concept c1;
        c1=(Concept)v.elementAt(i);
        temp=c1.num_of_ob;
        if (temp>Max)
          {
            Max=temp;
            labe_Concept=c1;
          }
         else
             dummy++;
       }
   return labe_Concept;
 }


 void send_new_query1()
  {
     int num_of_tokens;
     StringTokenizer tokens=new StringTokenizer(sp);
     num_of_tokens=tokens.countTokens();
     String[] temp=new String[num_of_tokens];

     for(int i=0;i<num_of_tokens;i++)
      {
        temp[i]=tokens.nextToken();
        System.out.println("*********");
        System.out.println(temp[i]);
        System.out.println("*********");
```

```
}

if( num_of_tokens == 1)
 {
   temp[0]="a";
   sp=temp[0];
   send_new_query();
 }
else if(num_of_tokens==4)
 {
   sp="";
   if(!temp[3].equals("void"))
    {
       temp[3]="a";
       for(int i=0;i<num_of_tokens;i++)
        {
                sp+=temp[i];
                if(i<num_of_tokens-1)
                  sp+=" ";
                else
                  dummy++;
        }
       send_new_query();
    }
   else
      area2.setText(temp_class);
   }
  else if(num_of_tokens==5)
   {
     if(!temp[1].equals("a"))
      {
        temp[1]="a";
        sp="";

        for(int i=0;i<num_of_tokens;i++)
         {
          sp+=temp[i];
          if(i<num_of_tokens-1)
            sp+=" ";
          else
            dummy++;
         }
        send_new_query();
        }
      if(!temp[4].equals("a"))
       {
         temp[4]="a";
         sp="";

         for(int i=0;i<num_of_tokens;i++)
          {
            sp+=temp[i];
            if(i<num_of_tokens-1)
               sp+=" ";
            else
               dummy++;
          }
         send_new_query();
         }
       temp[4]="b";
       sp="";
       for(int i=0;i<num_of_tokens;i++)
        {
          sp+=temp[i];
          if(i<num_of_tokens-1)
            sp+=" ";
          else
            dummy++;
        }
       send_new_query();
     }
```

```
    else if(num_of_tokens==7)
     {
        String str1="";
        String str2="";
        str1=temp[1];
        str2=temp[3];
        temp[1]="a";
        sp="";
        for(int i=0;i<num_of_tokens;i++)
         {
           sp+=temp[i];
           if(i<num_of_tokens-1)
             sp+=" ";
           else
            dummy++;
         }
         send_new_query();
         temp[1]=str1;
         temp[3]="a";
         sp="";

         for(int i=0;i<num_of_tokens;i++)
          {
           sp+=temp[i];
           if(i<num_of_tokens-1)
             sp+=" ";
           else
            dummy++;
          }
         send_new_query();
         temp[1]="a";
         sp="";

         for(int i=0;i<num_of_tokens;i++)
          {
           sp+=temp[i];
           if(i<num_of_tokens-1)
             sp+=" ";
           else
             dummy++;
          }
        send_new_query();
        temp[6]="a";
        sp="";

        for(int i=0;i<num_of_tokens;i++)
         {
           sp+=temp[i];
           if(i<num_of_tokens-1)
             sp+=" ";
           else
             dummy++;
         }
         send_new_query();
         temp[6]="b";
         sp="";

         for(int i=0;i<num_of_tokens;i++)
          {
             sp+=temp[i];
             if(i<num_of_tokens-1)
                  sp+=" ";
             else
                  dummy++;
           }
           send_new_query();
        }

else
   dummy++;
```

```
  }

boolean check_we_can_add_common_class(String str )
 {
   boolean flag=true;

   for(int i=start;i<index;i++)
    {
     if (str.equals(common_class[i]))
      {
         flag=false;
         break;
      }
     else
         dummy++;
    }
   return flag;
 }

void send_new_query()
 {
    System.out.println("The changed query type :"+sp);
    buff1+="\n\n";
    buff1+="The changed query type: ";
    buff1+=sp;
    area1.setText(buff1);
    check_type(sp);
    if(Fun_Type.equals("xxx")) return;
    System.out.println("The new Fun_Type is "+Fun_Type);
   query_types+=": "+sp+" :";
   temp_Concept=new Vector();

   for(int i=0;i<con_arr.size();i++)
    {
         Concept c1;
         c1=(Concept)con_arr.elementAt(i);
          if(c1.isIncluded2(Fun_Type)==true)
             temp_Concept.addElement(c1);
         else
             dummy++;
     }
     Concept lower_bound=null;
     System.out.println("The length of temp_Concept is "+temp_Concept.size());
     lower_bound=what_is_lower_bound_concept(temp_Concept);
     System.out.println(lower_bound.num_of_ob);
     temp_class+="\n*** The Reusit is as follows ***";
     temp_class+="\n";
     temp_class+="The classes that has the type: "+sp+"\n\n";

     for(int i=0;i<lower_bound.num_of_ob;i++)
      {
        temp_class+=(String)(lower_bound.vl_Of_Concept.elementAt(i));
        temp_class+="\n";
        if(check_we_can_add_common_class((String)(lower_bound.vl_Of_Concept.elementAt(i))) != false)
          {
           common_class[index++]=(String)(lower_bound.vl_Of_Concept.elementAt(i));
          }
      }
     if(number_of_function==0)
      {
        first_length+=lower_bound.num_of_ob;
      }
     System.out.println("The concept name is "+lower_bound.Name_Of_Concept);
     System.out.println(temp_class);
     area2.setText(temp_class);

}
```

```
void check_type(String str)
{
 if(str.equals("( ) -> void"))
  Fun_Type="T1";
        else if(str.equals("( ) -> long"))
  Fun_Type="T2";
        else if(str.equals("( ) -> int"))
         Fun_Type="T3";
        else if(str.equals("int"))
         Fun_Type="T4";
        else if(str.equals("( int , int ) -> void"))
         Fun_Type="T5";
        else if(str.equals("( int ) -> void"))
         Fun_Type="T6";
        else if(str.equals("USHORT"))
         Fun_Type="T7";
        else if(str.equals("( short ) -> void"))
         Fun_Type="T8";
        else if(str.equals("( ) -> short"))
         Fun_Type="T9";
        else if(str.equals("( a ) -> void"))
         Fun_Type="T10";
        else if(str.equals("( a ) -> a"))
         Fun_Type="T11";
        else if(str.equals("( ) -> a"))
         Fun_Type="T12";
        else if(str.equals("( ) -> bool"))
         Fun_Type="T13";
        else if(str.equals("*AANode"))
         Fun_Type="T14";
        else if(str.equals("a"))
         Fun_Type="T15";
        else if(str.equals("AVLNode"))
         Fun_Type="T16";
        else if(str.equals("( int ) -> int"))
         Fun_Type="T17";
        else if(str.equals("BinaryNode"))
         Fun_Type="T18";
        else if(str.equals("Vector"))
         Fun_Type="T19";
        else if(str.equals("ListNode"))
         Fun_Type="T20";
        else if(str.equals("( ) -> ListItr"))
         Fun_Type="T21";
        else if(str.equals("( int& , ListItr& ) -> void"))
         Fun_Type="T22";
        else if(str.equals("( int& ) -> ListItr"))
         Fun_Type="T23";
        else if(str.equals("( int& ) -> void"))
         Fun_Type="T24";
        else if(str.equals("( ) -> Object&"))
         Fun_Type="T25";
        else if(str.equals("( ) -> int&"))
         Fun_Type="T26";
        else if(str.equals("( a ) -> PairNode"))
         Fun_Type="T27";
        else if(str.equals("( PairNode , int ) -> void"))
         Fun_Type="T28";
        else if(str.equals("( int , int ) -> int"))
         Fun_Type="T29";
        else if(str.equals("( a ) -> Hashable"))
         Fun_Type="T30";
        else if(str.equals("HashedObj"))
         Fun_Type="T31";
        else if(str.equals("( a& , ListItr& ) -> void"))
         Fun_Type="T32";
        else if(str.equals("( a& ) -> ListItr"))
         Fun_Type="T33";
        else if(str.equals(" a& -> void"))
         Fun_Type="T34";
        else
```

```
            }
             Fun_Type="xxx";
             buff1+="\n*** This type is not defined ***";
             buff1+="\n\n";
             area1.setText(buff1);
            }



}


void get_the_common_class()
{
   System.out.println("Num_Of_Type is "+number_of_function);

  for(int i=0;i<index;i++)
    System.out.println(common_class[i]);

   String buff3=" The common classes that has the query types \n";
   buff3+=query_types;
   buff3+="\n";
  int temp;
  if (number_of_function == 0)
   {
    for(int i=0;i<first_length;i++)
     {
       buff3+=common_class[i];
       buff3+="\n";
     }
   }

  for(int i=0;i<first_length;i++)
   {
     temp=0;
    for(int j=first_length;j<index;j++)
     {
       if (common_class[i].equals(common_class[j]))
          {
            temp++;
            if (temp==number_of_function)
              {
               buff3+=common_class[i];
               buff3+="\n";
               break;
              }
          }
       else
          {
            dummy++;
          }
     }
   }
  System.out.println("The length of First is :"+first_length);
 area3.setText(buff3);
 buff3="";
}
}
```

VITA AUTORIS

Name : AHN, HANSOO

PLACE OF BIRTH : PUSAN, SOUTH KOREA

YEAR OF BIRTH : 1971

EDUCATION : MYUNG-GI HIGH SCOOL, SEOUL SOUTH KOREA
1987-1989
KOOK-MIN UNIVERSITY, SEOUL SOUTH KOREA
1990-1997
UNIVERSITY OF WINDSOR, WINDOSR ONTARIO, CANADA
1999-2002 M.Sc.