Chapter 4 Using Program Slicing to Collect Slicing Profile For Crash Analysis

After we catch an exception, the next job of our system is to analyze the cause of the exception and conjecture which component may have caused it. This work is actually done in two phases. Before the execution of a Java application, we use program slicing to collect the slice information between statements from source files of the Java application. , The second phase is done after an exception is catched. In this phase, we use the slice information computed in first phase to conjecture the root cause of the crash.

Here, we describe the system functions for preparing the slice information by parsing the Java source in first phase. In this phase, we first use tool *programmer* to parse the Java source files. Java source parser is use for analyzing the Java source file using program slicing to prepare the information for diagnose exception.

In our system, we use Java bean to be a component. So, our system provides a tool called “component manager” to create a empty bean and then allow users to insert classes into a bean. These information will be used to compile these Java files

and build team into a JAR file. In our case, a JAR file contains the classes of a bean.

4.1 Using Programmar to parse Java source files

4.1.1 Information to construct component map

First, we need to create a component map that shows the relationship of all components of the program. Component includes class so the relation between components can be reduced to relation of classes across different components. There are several kinds of relationships between classes. They are Dependency, Association, Aggregation, Composition, Generalization, and Realization. In our system, we only use Association to show the relationship of class in component map. An example code in Figure 4-2 that the example graphic in Figure 4-3.

```
public class manager
{
        pmanager pm=new pmanager();
}
```

Figure 4-2 : An example code

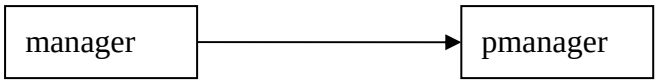

Figure 4-3 : Graphic that show the relation of class in Figure 4-2

The association relation can be obtained from the object's declaration in the class source. So, all the declaration statements in class and in method are the information we need to construct the component map.

4.1.2 Information to conjecture crash causes

As said in introduction, program slicing is useful for debugging, because it potentially allows one to ignore many statements in the process of localizing a bug. If a program computes an erroneous value for a variable *x*, only the statements in the slice w. r. t. *x* have (possibly) contributed to computation of that value; all statements which are not in the slice can safely be ignored. Before making a change to *s*, one could examine the slice *s*, indicating the program part affected by *s*. this may produce useful insights how the error may be corrected.

In our case, when a Java application crashes at statement *s*, we want to back trace the statements which may cause statement *s* to crash. Statement *s* may contains several variables which were pre-computed by other statements. To provide these

back-tracking information, we use program slicing to prepare these information and make them ready for exception diagnosis system to use.

4.2 Program slicing Java source

In this section, we describe how we slice a Java source files using Programmer. We first collect the variable defined in a class. Second, we find out the statements which use these variables. Starting from the declaration of the given variables, we follow the forward slice to get all the references of the given variables. In our system, we use forward slice to do program slicing with Java source file. Backward slicing would include the more statements that affect the variable indirectly. We do program slicing to each variable in each class by using forward slicing and use link to replace the other statement that backward slicing would include. Note that our approach is different from the pure forward slicing in the following way. If the statement of a reference is inside the flow control statement (e.g., if statement, for statement, while statement and etc.), we will slice the flow control statement as well. The algorithm is illustrated in Figure 4-4.

```
For each variable v
    For each statement s contains v
        If s is in flow control statement then
            Mark a link to the variables in the flow statement
        End if
        Get the line number l of the statements
        Keep the statement s and line number l
    Next
Next
```

Figure 4-4 : The algorithm to slice a java source file

For example, the forward slice in Figure 4-5 (A) shows exactly which statements are influenced by the initialization of variable *sum*. Our slice in Figure 4-5 (B) includes the statement. If.

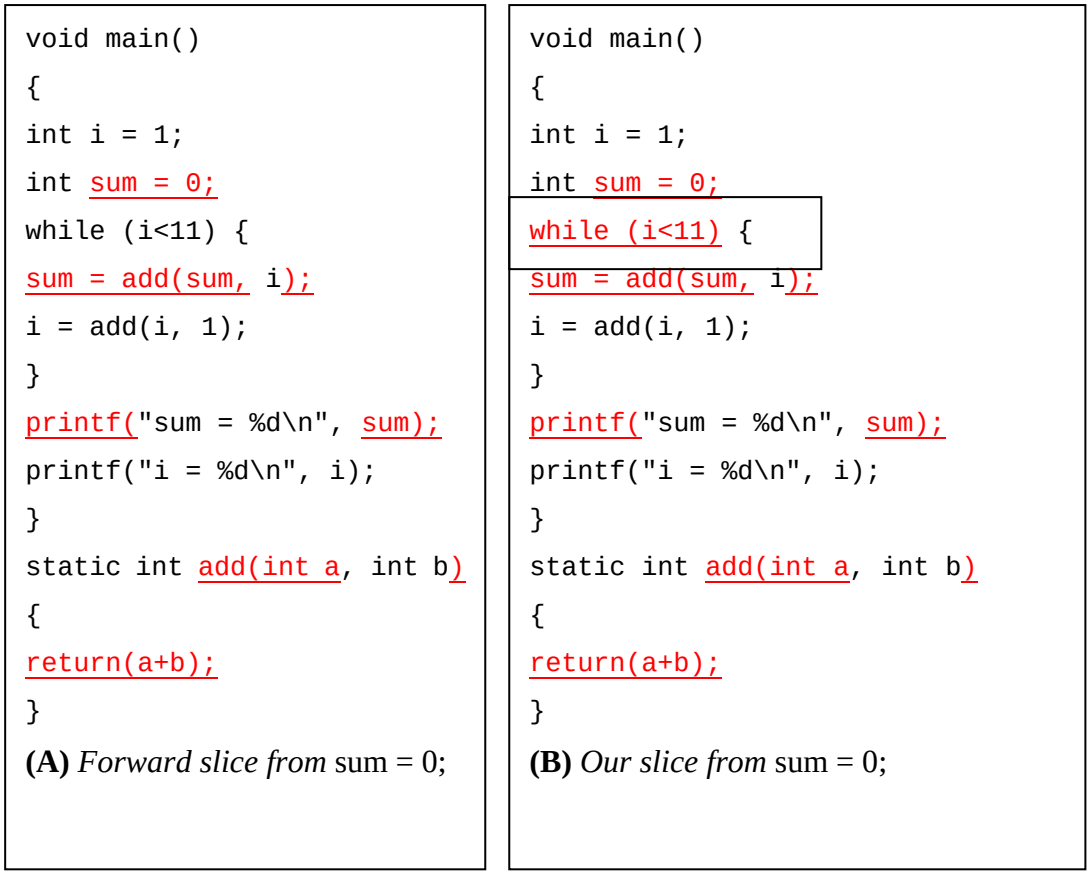

Figure 4-5 : The difference of forward slice and our slice

It's different from backward slice that the backward slice exactly which statements influence the output of variable *i*. The statement "`sum = add(sum, i);`" that would be effect by the flow statement "`while (i<11)`" so we mark it to link the slice profile of value *i* . The different from the backward slice show in the Figure 4-5. The slicing profile of the sum in the main is how in the Figure 4-5.

```
void main()
{
int i = 1;
int sum = 0;
while (i<11) {
sum = add(sum, i);
i = add(i, 1);
}
printf("sum = %d\n", sum);
printf("i = %d\n", i);
}
static int add(int a, int b)
{
return(a+b);
}
```

(A). Backward *slice for sum from*

```
printf("i = %d\n", i);
```

```
void main()
{
int i = 1;
int sum = 0;
while (i<11) {
sum = add(sum, i);
i = add(i, 1);
}
printf("sum = %d\n", sum);
printf("i = %d\n", i);
}
static int add(int a, int b)
{
return(a+b);
}
```

(B) *Our slice for sum from* sum = 0;

Figure 4-6 : The difference of backward slice and our slice

```
<ident_reference>
  <ident>
    <line>4</line>
    <statement> int sum = 0;</statement>
  </ident>
  <ident>
      <if><flowident>main_i</flowident></if>
      <method>add</ method >
      <line>4</line>
      <statement> sum = add(sum, i);</statement>
  </ident>
  <ident>
      <line>4</line>
      <statement> printf("sum = %d\n", sum);</statement>
  </ident>
</ident_reference>
```

Figure 4-7 : The slicing profile of Figure 4-6(B)

We call program slices of a variable a *slicing profile*. We would create all the slicing profile for all variables that declared in the source. The flow statement is possible to affect the variables so we mark it to link the other variable's slice. For example, in Figure 4-6 (B) we show the slice profile of *sum.* It has a flow statement that will affect the sum. Figure 4-8 illustrates this. In our system, we need to prepare the information about source for diagnosing the exception. So, we got to do program slicing for all variables in source. If we use backward slicing to do this then the result of all the variables would cover each other. But we still need to find all the side effect statement that cause exception. So use the slicing could help find out all the side

effect statements and the result of program slicing do not cover each other.

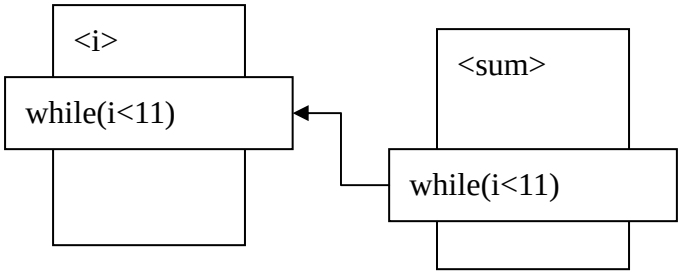

Figure 4-8 : slicing profile of sum has a flow statement link to slicing profile of i

4.2.1 Procedure of analyze the source

1. Find out all the global declaration and method in the class. It helps for collecting the relation among all class. We divide a java source body into four parts. Variable declaration, Method declaration, Constructor declaration, and class declaration. We get the Variable declaration and Method parts from a java source. For example, in Figure 4-9 we need to find out all the Variable declaration and Method declaration in it. The result of query from the programmer is shown in Figure 4-10.

```
public class FileLister extends Frame {
        private List list;
        private TextField infoarea;
        private Panel buttons;
        private Button parent, quit;
        private FilenameFilter filter;
        private File cwd;
        private String[] entries;
        public FileLister(String directory, FilenameFilter filter) throws
IOException
        {}
        public void list_directory(String directory) throws IOException {}
}
```

Figure 4-9 : The example code

```
        private List list;
        private TextField infoarea;
        private Panel buttons;
        private Button parent, quit;
        private FilenameFilter filter;
        private File cwd;
        private String[] entries;
        public void list_directory(String directory)
```

Figure 4-10: Find out all the Variable declaration

2. Implement the program slice that we introduced in 4.2 for each variable that we collect from step 1. Find out all the statements of each variable that has

reference in it and check those statements are in flow control statement or not. If the statement is in flow statement then find out what reference in the flow statement. The statement architecture is shown in the Figure 4-11. If a statement is found and it's parent is a flow control statement like if, for, while that we would mark the statement and get all the variable references in the flow control statement as well.

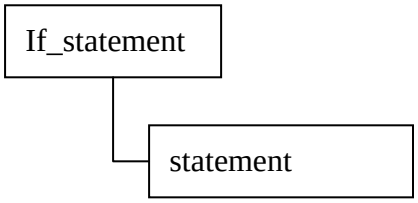

Figure 4-11 : A part of the parser tree about the flow statement

3. Find out all the variable declaration in each method. Like the step 1 in the function block.
4. Repeat the step 2 for each variable that is declared in the method

Our procedure repeats the four steps for all source files and save all the slices into several *XML* files. We use these XML file to record all the slicing information of a java source file. There are different kinds of xml file which are described in Figure

4-12. We use the class name, method name ,and value name to name the XML file. The project file is to keep the information of which classes are belonged to which bean (it is set up in a tool we previously describe).

File name	Description
Project.xml	Note what class and bean in the application and each bean contain what class in it
[class name].xml	Note all the variable declaration and method in the class
[class name]_[method name].xml	Note what variable declaration in the method of the class
[class name]_[value name].xml	Note all the reference of the value in the class
[class name]_[method name]_ [value name].xml	Note all the reference of the value in the method

Figure 4-12 : All kind of XML files that keeps the slicing information of the source file