# American International University-Bangladesh (AIUB)



# Final term Assignment of

# 

# Software Evolution and Maintenance

**Submitted By:**

|  |  |
| --- | --- |
| **Name** | **ID** |
| MD. RIFAT-UZ-ZAMAN | 22-92610-2 |

**Course:** SOFTWARE EVOLUTION AND MAINTENANCE

**Course Teacher:** DR. MD MEHEDI HASAN

**Section:** [MScCS] A

**Semester:** Spring 2022-2023

**Date of Submitting:** 12APRIL, 2023

**Question no 01:**

The above-mentioned paper talks about “transform slicing” and “direct slicing”.  Find a relevant paper that describes the two types of program slices. Describe the two types and their purpose. Which one can be automated? Give full reference to the paper(s) where you found the relevant information.

**Answer:**

Paper name:

Publication:

Paper Source:

**Question no 02:**

We mentioned two other types of slices in the class: *backward* and *forward*. Write the difference between *backward* and *forward* slicing with proper examples.

**Answer:**

Backward slicing and forward slicing are two techniques used in program slicing, which is a program analysis technique that helps developers identify the portions of code that contribute to a particular output or behavior. Here are the differences between backward and forward slicing:

|  |  |  |
| --- | --- | --- |
| **Type** | **Backward Slicing** | **Forward Slicing** |
| Definition: | Backward slicing is a technique that starts from the output of the program and identifies the parts of the program that contribute to that output. | Forward slicing is a technique that starts from a particular point in the program and identifies the parts of the program that are affected by that point. |
| Goal: | The goal of backward slicing is to reduce the code size by identifying the relevant portions of the program that contribute to the output. | The goal of forward slicing is to identify the code that is affected by a particular point in the program so that developers can understand the impact of changes to that point. |
| Example:  1. a = 2  2. b = a + 3  3. m= b \* 2 | Backward slicing: If we want to know which part of the program contributes to the value of “m”, we start from “m” and work backward. In this case, we would identify that y \* 2 contributes to the value of “m”, and then identify that y comes from a + 3, which in turn comes from a = 2. Therefore, the relevant portion of the program is lines | Forward slicing: If we want to know which parts of the program are affected by the value of “a”, we start from a = 2 and work forward. In this case, we would identify that y is affected by a, and that “m” is affected by “b”. Therefore, the relevant portion of the program is lines 1-3. |
| Techniques: | Backward slicing uses a technique called backward data-flow analysis, which involves tracing the data flow of the output back through the program to identify the relevant portions of the code. | Forward slicing uses a technique called forward data-flow analysis, which involves tracing the data flow of a variable or statement forward through the program to identify the portions of the code that are affected by it. |
| Applications: | Backward slicing can be used to identify the root cause of a bug or error in a program. By starting from the output and working backward, developers can isolate the portion of the code that is responsible for the error. | Forward slicing can be used to identify the impact of a change to a particular point in a program. By identifying the portions of the code that are affected by a variable or statement, developers can understand the potential consequences of modifying that variable or statement. |
| Limitations: | Backward slicing can be limited by the complexity of the data flow in a program. If the data flow is too complex, it may be difficult to trace the flow of data back to the relevant portions of the code. | Forward slicing can be limited by the number of variables or statements that are affected by a particular point in the program. If a large number of variables or statements are affected, the resulting slice may be too large to be useful. |
| Time complexity: | Backward slicing can be computationally expensive, as it involves tracing the data flow through the entire program. This can be time-consuming, especially for large programs. | Forward slicing is generally faster because it only involves tracing the data flow from a specific point in the code. |
| Code modification: | Backward slicing is useful for identifying the portions of the code that need modification to fix an error. | Forward slicing is useful for identifying the portions of the code that are impacted by a modification to ensure that the modification does not introduce new errors. |
| Static vs dynamic: | Backward slicing is usually performed statically | Forward slicing can be performed either statically or dynamically. |
| Modification: | Backward slicing is useful for identifying the portions of the code that need modification to fix an error. | Forward slicing is useful for identifying the portions of the code that are impacted by a modification to ensure that the modification does not introduce new errors. |
| Dependencies: | Backward slicing considers all the possible paths that lead to the output or error. | Forward slicing considers only the paths that lead to the specific statement or input. |

Example differences between backward and forward slicing:

Let's consider a simple code snippet as an example to illustrate the difference between backward and forward slicing.

Code:01

1. x = 5

2. y = x + 2

3. z = y \* 3

4. print(z)

In this code, we have three variables x, y, and z that are initialized with values, and then y and z are computed based on the value of x.

Now, let's say we want to use slicing to identify the portion of the code that contributes to the output z. We can use backward slicing by starting from the output z and tracing the data flow of z back through the code to identify the relevant portions of the program. In this case, we see that z is assigned the value of y \* 3, and y is assigned the value of x + 2. So, the relevant portion of the code in lines 2-3.

Code:02

2. y = x + 2

3. z = y \* 3

On the other hand, let's say we want to use slicing to identify the portions of the code that are affected by a change to the variable x. We can use forward slicing by starting from the statement that assigns the value to x and tracing the data flow of x forward through the code to identify the relevant portions of the program. In this case, we see that x is used to compute y and y is used to compute z. So, the relevant portion of the code in lines 2-3.

Code:03

2. y = x + 2

3. z = y \* 3

As we can see, both backward and forward slicing identify the same portion of the code in this example, but the starting points are different. Backward slicing starts from the output and identifies the relevant portions of the code, while forward slicing starts from a particular point and identifies the portions of the code that are affected by that point.

In summary, backward and forward slicing are two complementary techniques that can be used to analyze and understand programs. While backward slicing starts from the output and identifies the relevant portions of the code, forward slicing starts from a particular point and identifies the portions of the code that are affected by that point. Both techniques have their own strengths and limitations, and choosing the appropriate technique depends on the specific task at hand.