Analyzing Control Flow for Logic Errors

Thiri Nwe, Swe Zin Hlaing
thirixxo@gmail.com,swezinhlaingucsy@gmail.com

Abstract

Analyzing the program conditional control flow the important technique using in program execution. Software errors are often caused mostly due to the complexity of software, programming errors, time pressure and changing requirements. In order to ensure the quantity and integrity of a software product, need to do intensive quantity assurance. Analyzing involves operation of the program or application under controlled control flow, can easily known what control flow will be done during program execution. Control statements are used to control the flow of execution of the program. This execution order depends on the supplied data values and conditional logic. Control flow graph in computer science is a representation, using graph notation, of all parts that might be traversed through a program during its execution. This system describes what are the incorrect control flow logic errors are by comparing defined control flow in program source code.
Keywords: Analyzing, Conditional control flow, Branch (link) coverage, program node setting

1. Introduction

Control flow is the sequence of operation and control flow analysis is the flow of control within a procedure. This control flow construct programming rules more general than that from program source code and then applies to discover programming rules implicitly embedded in source code, called static analysis.

Control flow graph indicates for each point in a program the possible program points to be executed next. Analyzing control flow graph of a program is a standard practice in optimizing conditional flows for general purpose.

Writing a program exists software errors; classify into two categories: memory errors and logic errors. Logic errors refer to conditional control flow logic incorrectness (E.g., if, while, for) that may be embedded correct control in source code.

This paper structured with seven titles. They are introduction in section 1, theory background in section 2, overview of the system in section 3, flow diagram of the system in section 4, branch (link) coverage in section 5, control flow analysis in section 6, java control statement in section 7, the time taken between program nodes in section8. Section 9 describes the conclusion. Flow diagram of the system displays the details of system operations with step by step. Finally, Analyzing involves the overview of conditional control flow and detail of program node setting.

2. Theory Background

The theories of analysis produces a set of interaction fragments that precisely represents the control-flow behavior which affects the messages being sent by this method. These fragments encode all and only sequences of call statements that occur along all CFG control-flow paths. This section provides a high-level overview of the analysis; more details are available.

The analysis computes branch successors and loop successors for certain CFG nodes. This computation is based on the well-known notion of post-dominance. CFG node n2 post-dominates n1 if every path from *n*1 to an exit node contains n2. Node n2 immediately post-dominates n1 if n2 post dominates n1 and any other post-dominator of n1 is also a post-dominator of n2.

2.1. Branches and Branch Successors

A CFG node is a branch node if it has at least two outgoing edges. For some of these nodes the analysis creates all fragments. In this case it is necessary to determine which CFG nodes should be considered when building the contents of this new fragment.

Intuitively, we need to determine where the fragment "stops"; this stopping point is the start of the fragment that will follow the new alt fragment Consider a branch node n with outgoing edges (n, ni).

The branch successor of n is defined to be the lowest common ancestor of all ni in the post-dominance tree for the CFG. A common ancestor represents a merge point for all branches coming out of n. The lowest such ancestor is the merge point that is the "closest" to n.

The paths from n to this node define the CFG nodes that should be considered when building the contents of the alt fragment created for n. If n is inside a loop, the notion of a branch successor must be restricted to the flow of control that stays within the loop. For this, we define the notion of post-dominance inside a loop.

Consider nodes n1 and n2 in some loop L. Node n2 post-dominates n1 inside L if n2 belongs to every loop-only path from n1 to the loop header. This means that if n1 is reached during some iteration of L and subsequently the iteration completes successfully—i.e.,

the header of L is eventually reached—then n2 is reached after n1 as part of that same iteration.

If a node n has more than two successors in its enclosing loop L, the branch successor n is the lowest common ancestor of all such in the ostdominance tree for L. This definition can be easily generalized for nested loops.

2.2. Loops and Loop Successors

The notion of a reducible CFG is standard in program analysis research. A loop in a reducible CFG is a strongly connected Sub graphs L such that exactly one node n ∈ L has a predecessor that is not in the loop. Node n is the header node of L.

The control-flow features of Java ensure that the CFG of a method is reducible. CFG, because a loop fragment has only one entry point. Whenever the analysis encounters the header node of a loop, it creates a loop fragment.

In this case, the analysis needs to decide which nodes should be considered when building the contents of this loop fragment.

We determine a loop successor. In the case of an outermost loop, the loop successor is the lowest common ancestor for all targets of loop exit edges in the method-level post-dominance tree. This is the earliest common point for all possible executions after the loop terminates. The generalization for nested loops is presented.

2.3. Multiple CFG Exits

The algorithm becomes more complicated in the presence of multiple CFG exits. In particular, it becomes necessary to compute and use information about control dependencies between branch nodes and exit nodes, and to create return fragments for CFG sub graphs which lead to exit nodes. The general treatment of multiple exit nodes is described our implementation fully handles this general case.

3. Overview of the system

This system describes about analyzing control flow that the programmer writes. This system produces the control flow graph with flow graph timer. The system is analyzing the control flow that defined in programmer's source code with user's flow graph. This system uses the binary search program to analyze the control flow with programmer source code.

The system will be checked the conditional control flow that the user writes with source code control flow. The control flow must be written by java code. This system can be produced flow graph for any java program. But it can analyze only binary search program source code that is embedded in program.

4. Flow diagram of the System

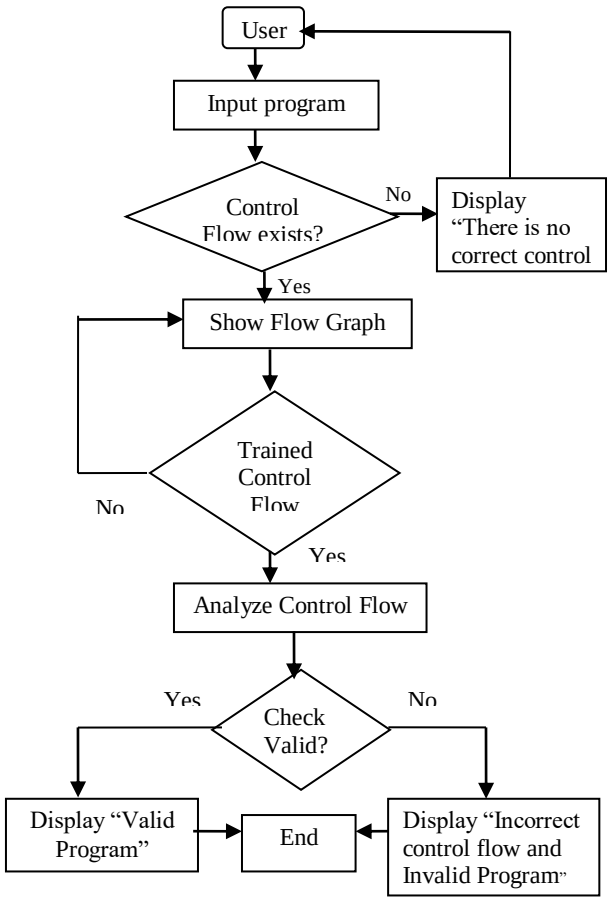

Figure1. System Flow for Analyzing Control Flow

When the user inputs the program, the system will be checked the program exists the control flow or not. If the program have the control flow, the system will be displayed the flow graph of the user's input program.

If the program hasn't the conditional control flow, the system will be displayed "There is no conditional control flow". If the program has been defined control flow, the system will be analyzed the control flow and check the control flow is valid or not.

Conditional control flow is not valid; the system will be displayed "Invalid program" and show the incorrect control flow. If the conditional control flow is valid, the system will be displayed "Valid program".

5. Branch (Link) Coverage

This system will select as many paths to cover all branches (links) at least once. Most test cases are required than statement coverage in terms of ability to detect more defects. It always contains statement coverage automatically by doing branch condition.

Node= sequence of action or procedure that has no branch

Link = sequence of execution of program

6. Control Flow analysis

Control flow analysis includes building basic blocks, building control flow graph, and loops. Data-flow is flow of data values from defs to users. Control –flow is sequence of operations. Example, Evaluation of then-code and else- code depends on if – test. This system uses a control flow graph (CFG).

Nodes represent statements and edges represent explicit flow of control. Control dependences in program dependences graph (PDG). Dependence on explicit statement in value dependence graph (VDG). Control flow information in a program may be static, as when the syntax of the program directly determines which parts of the program may be executed next.

Control flow may be dynamic, as when run time values and inputs of the program are required to determine which parts of the program may be executed next. A control flow analysis approximates the dynamic control flow information with conservative static control flow information.

Program Node setting

```
a ——→  public class Binary Search Tree {
b ——→  public void calculate tree Pos ( ) {
c ——→  if (num==sum) {
                    }                    }
d ——→  publicvoidbinarySearch () {
e ——→  while (first<last) {
f ——→  if (mid < first ) {
                    }
g ——→  else if (mid>first) {
                    }
h ——→  else {

                }
                }
                }
```

Example program for conditional control flow of node setting

In this program, node d, the analysis will create an alt fragment 3. The fragment, the alt in the loop's fragment sequence will be constructed starting from node d, which is the merge point of the branches coming out of node a, b and c. Node d is branch successor of node e.

7. Java Control Statement

Control the order of execution in a java program, based on data values and conditional logic. The system will be produced the conditional control flow graph written by java program. There are three main categories of control flow statements.

Selection statements: if, if-else, and switch

Loop statements: while, do-while, and for

Transfer statements: break, continue, return, try-catch-finally and assert.

7.1. Selection statement

7.1.1. The if-statement

It executes a block of code only if the specified is true. If the value is false, then the if block is skipped and executions continues with the rest of program. It can either have a single statement or a block of code within an if- statement. Note that the conditional expressions:

If (<conditional statement>)

<Statement action >

7.1.2. If-else statement

It is an extinction of the if-statement. If the statement in the statement in the if –statements in the else block are executed. It can either have a single statement or a block of code within if-else blocks.

Following syntax:

```
If (< conditional expression >)
        <Statement action >
Else
        < Statement action >
```

Syntax: condition? Expression1: Expression2

7.2. Loop statements

A loop statement instructs the computer to repeat an Operation (the Loop Body) depending on a condition. Loops are less general than recursion but often easier to setup. There are different types of loops; they differ in how and when the loop condition is being checked. Loop must be guaranteed to terminate at some point.

7.2.1. The 'while' Loop

A while loop in java looks a bit like the if statement: While (condition) statement. Statement will be repeated as long as condition is true. The condition is checked before each execution. So statement may not be executed at all.

7.2.2. The ‘do-while’ Loop

Sometimes it needs the loop to be executed at least once. Therefore it would be better to check the condition after instead of before the loop body execution. The do-while loop in java does just that do statement while (condition); statement will be executed once, and after that as long as condition is true.

7.2.3. The ‘for’ Loop

In some cases it needs to execute a loop just n times. We will therefore need a counter to count to n. The ‘for loop’ in Java combines the while loop with a way to handle such a counter:

For (expression1; expression2; expression3) statement

8. Time Taken between Program nodes

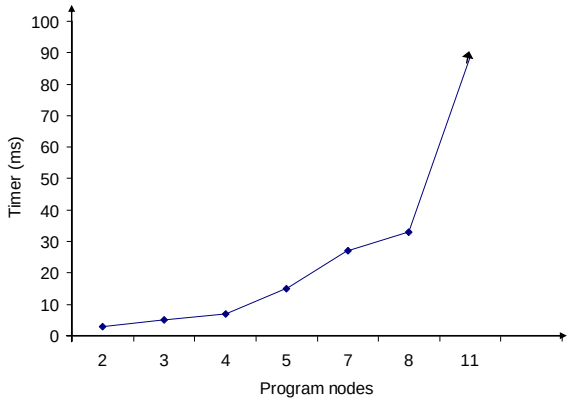

Figure2. Time taken between program nodes

Figure2 shows the time taken between program nodes. This system produces program nodes for user’s java program with timer. Timer will be increased depends on the number of nodes. Timer will be 3ms when number of nodes are 2, 15ms when number of nodes are 5, 33ms when number of nodes are 8, 89ms when number of nodes are 11 and etc.

9. Conclusion

This paper is building the control flow graph, the programmer will easily know the analyzing the conditional control flow. In the program, it can include some logic errors and mistake in conditional control flow. By analyzing the control flow, it can easily know how to control the conditional statement and find the logic errors. This system will be analyzed the binary search program with program source code and display the incorrect control flow if some control flow errors exists.

10. References

[1] D.Damian, “On static and dynamic Control Flow Information in Program analysis and Transformation”, Aug 2001.

[2] S. Geniam, F. Spoto, “Information Flow Analysis for Java byte code”, Dipartimento di Informatica, Universit`a di Verona Strada Le Grazie, 15, 37134 Verona, Italy.

[3] P. Haschka, C. Huitema, “Control Flow Graph Analysis for automatic fast path implementation”, Proceedings «Second IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems», Williamsburg, Virginia Sep. 1-3 1993.

[4] J.Krinke, S.Breu, “Control Flow Graph Based Aspect Mining”, Germany Breu. NASA Ames Research Center, USA, 2004.

[5] C.Liu, X.Yan, J. Han, “Mining Control Flow Abnormality for Logic Error Isolation”, Special Interest Group on Management of Data. Publisher, Pages: 872-881, Year of Publication: 2006, ISBN: 1-59593-339-5, 2006.

[6] N. Nystrom, “Interprocedural control flow analysis”, Lecture Notes in Computer Science, Pages: 20-39, Year of Publication: 1999, ISBN 3-540-65699-5.

[7] A. Rountev, O. Volgin, M. redoch,“Static Control-flow Analysis for Reverse Engineering of UML sequence Diagrams”, ISSN:0163-5948, Addison-Wesley Longman Publishing Co., Inc., Boston, Ma, 1996.