Software Verification and Validation

Structural Testing

Part Two – Data Flow Graph

Data Flow Criteria

- Covering all elements of a control flow graph might be infeasible
  - Practical and feasible for simple elements (e.g., statement)
  - But, infeasible for path coverage
    - E.g., Number of paths (entry point to exit point) in a given program
- Generating test cases based on data flow and flow of data
- Based on the observation that computing the wrong value leads to a failure only iff the value is used
- Similar to control-flow criteria, data flow is based on static analysis
- Pairing variable definitions and their uses
  - To ensure each computed variable has been really used
  - Must propagates the results of erroneous computation to the point of observable failure

Data Flow Criteria – Basic Definitions

- *definition* (*def*, D) of a variable *x*, statements performing creation, initialization, and assignment of the variable x
  - Creation
  - Initialization
  - Assignment
  - Passing to a function that modify variable *x*
- *use*(U) of variable x, statements using x without changing its value
  - Two alternatives
    - p-use (predicate use): Use of variable x in a predicate (i.e., decision) in a conditional expression such as if or loop
    - c-use (computational use): Use of variable x in a computation

Data Flow Criteria – Example of Definition & (C and P) Use

```
1.   int main( void )
2.   {
3.      int row = 10;
4.      int column;
5.      while ( row >= 1 ) {
6.         column = 1;
7.         while ( column <= 10 ) {
8.            printf( "%s",
9.                     row % 2 ? "<": ">" );
10.           column++;
11.        }
12.        row--;
13.        printf( "\n" );
14.     }
15.     return 0;
16.  }
```

- *Defs*
  - row: lines 3, 12
  - column: lines 4, 6, 10
- *C-uses*
  - row: lines 12
  - column: lines 10
- *P-uses*
  - row: lines 9, 5
  - column: lines 7

Data Flow Criteria – Definition-Use (DU) Pair

- Two statements *i* and *j* are in DU relationship, if there is a variable *x* and a path *p* from statement *i* to statement *j* such that:
  - *i* is a definition for *x* and *j* is a use of *x*
  - There is no other definition of *x* within the path *p*
- At least one definition-use path is required for a DU pair
  - There might be several paths
- *DU path* – A particular definition-use path between a definition and a use

Data Flow Criteria – DU Criteria

- *All Definitions*
  - Pairing each definition with at least one use
  - A test suite T for a given program P satisfies all definitions adequacy criterion iff for each definition *def* of P, there exists at least one test case that exercises a DU pair that includes *def*.
- *All DU pairs*
  - Each DU pair to be exercised
  - A test suite T for a given program P satisfies the all DU pairs adequacy criterion iff foe each DU pair, there is at least one test case in T that exercise that path.
  - One DU pair may belong to may different execution paths.
- *All DU Paths*
  - An extension to DU pairs
  - Each simple (non looping) DU path to be traversed at least once
    - Exercising different ways of pairing Ds and Us
    - Good for revealing a set of faults that the definitions of variables are missed

Data Flow Criteria – DU Criteria

- *Def-use*
  - There is at least one test case that causes the path *p* between the *def* and the *use* is exercised
- *C-use*
  - The same as def-use, however the use is only restricted to C-use
- *P-use*
  - The same as def-use, however the use is only restrict
- *C-use + P-use = Def-use*

Data Flow Criteria – DU Criteria

```
1.   int main( void )
2.   {
3.      int row = 10;
4.      int column;
5.      while ( row >= 1 ) {
6.         column = 1;
7.         while ( column <= 10 ) {
8.            printf( "%s",
9.                     row % 2 ? "<": ">" );
10.           column++;
11.        }
12.        row--;
13.        printf( "\n" );
14.     }
15.     return 0;
16.  }
```

- C-use
  - For row
    - (3, 12),(12, 9)
  - For column
    - (6, 10)
- P-use
  - For row
    - (3, 9), (3, 5), (12, 5)
  - For column
    - (6, 7), (10, 7)

Data Dependency Graph (DDG)

- DU pairs shows direct data dependence
  - Can be representable in the form of a directed graph called Data Dependency Graph
- In DDG
  - Nodes represent statements
  - Edges represent the traces of dependencies between variables
- There exist various types of DDG.
  - We use the one used in the book written by Mauro Pezze and Michal Young, "Software Testing and Analysis"
    - Each node represents a statement or block
    - Each edge represents the trace of the data, labeled by the name of the variable

Data Dependency Graph (DDG) – The Example

```
1.     int main( void )
2.     {
3.        int row = 10;
4.        int column;
5.        while ( row >= 1 ) {
6.           column = 1;
7.           while ( column <= 10 ) {
8.              printf( "%s",
9.                              row % 2 ? "<": ">" );
10.             column++;
11.          }
12.          row--;
13.          printf( "\n" );
14.       }
15.       return 0;
16.    }
```