Lecture 14 (Testing Metrics)
Analysis of Software Artifacts
Somesh Jha
Testing Terminology

- **white box testing:** tester has access to the code

- **black box testing:** tester does not have access to the code

- black box testing is useful for testing COTS based systems
Specification Based Testing

• let $I$ be the input domain

• let $O$ be the output domain

• a test-suite is a finite set of values from the set $I$

• let $i_1, \cdots, i_k$ be a test-suite of $k$ values
Specification Based Testing

• run the program on inputs $i_1, \cdots, i_k$

• check the output with the expected outputs

• expected outputs derived from the specification
How to Derive Tests

- use specifications
- partition the input domain $I$ into classes
- system behaves the same on each class
- pick at least one input from each class
- known as partition testing
Random Testing

• sample input domain $I$ randomly

• big debate on which is better
  – partition testing
  – random testing

• probably should use a combination
Other Forms of Testing

• **boundary Testing:** check exceptional conditions

• *example:* start the client process without starting the server

• **load Testing:** check how the system performs under exceptional load

• *example:* load a web-server with unusual traffic of transactions
Paper

Program as Graph

• We will view a program as a directed graph.
• Each node of the graph represents a basic block.
• Each edge of the graph represents a conditional, i.e., control flow.
• A basic block is a sequence of statements such that:
   – can only enter the basic block through the first statement.
   – can only leave the basic block after executing the last statement.
Classifying variable occurrence

- **Definition** (denoted as *def*)
  Variable is defined here, e.g., assignment or a read statement.

- **Use for computation** (denoted by *c-use*)
  Use the variable for computation, e.g., in an expression or print statement.

- **Use for control flow** (denoted by *p-use*)
  Use variable in the condition of an if statement.

- Each variable occurrence in the program is classified according to the criteria given above.
Definitions

- A *path* is a finite sequence of nodes $(n_1, \ldots, n_k)$ such that there is an edge from $n_i$ to $n_{i+1}$.

- A path is *simple* if all nodes, except possibly the first and the last, are distinct.

- A path is *loop-free* if all nodes are distinct.

- A *complete path* is a path whose initial node is the start node and whose final node is the exit node.
Definitions (Contd)

• A c-use of variable $x$ is a *global c-use* provided that there is no def of $x$ preceding the c-use within the block which it occurs.

• A path $(i, n_1, \cdots, n_m, j)$ from $i$ to $j$ is a *def-clear path* wrt $x$ iff $n_1, \cdots, n_m$ contain no defs of $x$.

  **Note:** wrt to is an abbreviation for with respect to.

• A path $(i, n_1, \cdots, n_m, j, k)$ is called a *def-clear path* wrt $x$ from node $i$ to edge $(j, k)$ if nodes $(n_1, \cdots, n_m, j)$ contain no defs of $x$. 
Definitions (Contd)

- A def of variable $x$ in node $i$ is called a *global def* if its the last def of $x$ occurring in the block associated with node $i$ and
  - there is a def-clear path wrt $x$ to node containing a global $c$-use of $x$ or
  - to an edge containing a $p$-use of $x$.

- Intuitively, a def is called global iff that definition can be used outside that basic block.

- A def that is not global is called a *local def*.
Definitions

• Assume that there is some path from start node to every global c-use or p-use of a variable which contains a def of that variable.
• $def(i)$ is the set of variables for which node $i$ contains a global def.
• $c$-use$(i)$ is the set of variables for which node $i$ contains a global c-use.
• $p$-use$(i,j)$ is the set of variables for which edge $(i, j)$ contains a p-use.
Definitions

- $dcu(x, i)$ is the set of all nodes $j$ such that $x \in c\text{-}\text{use}(j)$ and for which there is a def-clear path wrt $x$ from $i$ to $j$.

- $dpu(x, i)$ is the set of all edges $(j, k)$ such that $x \in p\text{-}\text{use}(j,k)$ and for which there is a def-clear path wrt $x$ from $i$ to edge $(j, k)$.
Definitions

A path \( (n_1, \cdots, n_j, n_k) \) is a *du-path* wrt \( x \) if \( n_1 \) has a global def of \( x \) and either:

- \( n_k \) has a c-use of \( x \) and \( (n_1, \cdots, n_j, n_k) \) is a def-clear simple path wrt \( x \)
- \( (n_j, n_k) \) has a p-use of \( x \) and \( (n_1, \cdots, n_j) \) is a def-clear loop-free path wrt \( x \).
Computing $dcu(x,i)$

- First we compute $\text{def-clear}(x,i)$, i.e., the set of nodes which have a def-clear path wrt $x$ from node $i$.

- Use fix-point equations. Let $DC(x,i)$ contain the node $i$ initially.
Computing $dcu(x, i)$ (Contd)

- Let $DU(x)$ be the set of nodes that have a definition of $x$ in it. Can compute this by just looking at the graph of the program.

- Update $DC(x, i)$ using the expression given below:
  \[ DC(x, i) \cup succ(DC(x, i) - DU(x)) \]
- Explain the equation given above.

- When the equations given above reach a fix-point the $DC(x, i)$ is equal to $def\-clear(x, i)$. 
Computing $d_{cu}(x,i)$ (Contd)

- Let $c\text{-use}(x)$ be the set of nodes that have a c-use of $x$.

- I claim that $d_{cu}(x,i)$ is given by the following expression:

$$c\text{-use}(x) \cap \text{def-clear}(x,i)$$

- Why?

- Computation of $d_{pu}(x,i)$ is analogous. We keep track of edges rather than nodes.
Sample Program

Figure 1: Sample Program
Sample Program (Contd)

- Compute dcu(pow,3) and dpu(pow,3).

- Compute dcu(pow,6) and dpu(pow,6).
Path Selection

- Let $G$ be the def/use graph of a program. We turn a program graph into a def/use graph by associating the usage information def, c-use with nodes and p-use with edges.

- **Goal:** To select a set of complete paths $P$ through the graph satisfying certain criteria.
Path Selection (Contd)

- **All-nodes**
  
  $P$ satisfies the *all-nodes* if every node of $G$ is included in $P$.

- **All-edges**
  
  $P$ satisfies the *all-edges* if every edge of $G$ is included in $P$.  

Path Selection (Contd)

- **All-defs**
  
  $P$ satisfies the *all-defs* criterion if for every node $i$ of $G$ and every $x \in \text{def}(i)$, $P$ includes a def-clear path wrt $x$ from $i$ to some element of $\text{dcu}(x,i)$ and $\text{dpu}(x,i)$.

- **All-p-uses**
  
  $P$ satisfies the *all-p-uses* criterion if for every node $i$ of $G$ and every $x \in \text{def}(i)$, $P$ includes a def-clear path wrt $x$ from $i$ to some element of $\text{dpu}(x,i)$.
Path Selection (Contd)

- **All-c-uses/some-p-uses**
  $P$ satisfies the *all-c-uses/some-p-uses* criterion if for every node $i$ and every $x \in \text{def}(i)$, $P$ includes some def-clear path wrt $x$ from $i$ to every node in $\text{dcu}(x,i)$; if $\text{dcu}(x,i)$ is empty, then $P$ must include a def-clear path wrt $x$ from $i$ to some edge contained in $\text{dpu}(x,i)$.

**Intent**: Checking every c-use of a variable and checking a p-use if there are no c-uses. Statements given precedence over conditionals, i.e., computation given preference over control.
Path Selection (Contd)

- **All-p-uses/some-c-uses**
  
  $P$ satisfies the *all-p-uses/some-c-uses* criterion if for every node $i$ and every $x \in \text{def}(i)$, $P$ includes some def-clear path wrt $x$ from $i$ to every node in $\text{dpu}(x,i)$; if $\text{dpu}(x,i)$ is empty, then $P$ must include a def-clear path wrt $x$ from $i$ to some edge contained in $\text{dcu}(x,i)$.

  **Intent:** Conditionals given preference over statements.
Path Selection (Contd)

• All-uses
  \( P \) satisfies the *all-uses* criterion if for every node \( i \) and every \( x \in \text{def}(i), P \) includes a def-clear path wrt \( x \) from \( i \) to all elements of \( \text{dcu}(x,i) \) and to all elements of \( \text{dpu}(x,i) \).

  **Intent:** Checks all possible uses of a variable \( x \). Very expensive.
Path Selection (Contd)

- **All-du-paths**
  
  $P$ satisfies the *all-du-paths* criterion if for every node $i$ and every $x \in \text{def}(i)$, $P$ includes every du-path with respect to $x$.  
  **Intent**: Checking every possible use of a variable $x$. Very expensive path selection criteria.

- **All-paths**
  
  $P$ satisfies the *all-paths* criterion if $P$ includes every complete path of $G$. In general, is this feasible?
Other Kind of analysis

- **Endless loop**
  An endless loop is a path \((n_1, \ldots, n_k)\), for \(k > 1\) and \(n_1 = n_k\), such that none of the blocks represented by the nodes on the path contain a conditional transfer statement whose target is either in a block which is not on the path or is a halt statement.

- **Using but no definition**
  A *def-clear* path from the start node to a use of variable \(x\) is a possible error/anamoly. Might be using a variable before defining it.