Data Flow Coverage 1

Conrad Hughes
School of Informatics

Slides thanks to Stuart Anderson
Why Consider Data Flow?

- **Control flow:**
  - Statement and branch coverage criteria are weak.
  - Condition coverage and path coverage are more costly and can become infeasible.

- **Data Flow:**
  - Base the coverage criterion on how variables are defined and used in the program.
  - Coverage is based on the idea that in principle for each statement in the program we should consider all possible ways of defining the variables used in the statement.

- **Data Flow Analysis** arose in the study of compiling - as well as suggesting coverage criteria it can also provide a means of statically checking variables are defined before use.
We introduce some standard naming conventions:
- **P** - code under test.
- **G(P)** - control flow graph of P, \( G(P) = (V,E,s,f) \) (Vertices, Edges, start node, finish node)
- Path is a sequence of vertices: \( v_0, v_1, \ldots, v_k \) where for each \( i \) (\( 1 < i < n+1 \)): \( (v_{i-1}, v_i) \) is a member of E.
- **x** is a variable of P
- If \( v \) is a vertex of the flow graph we define:
  - **defs(\( v \))**: the set of all variables that are defined at \( v \) (i.e. are on the LHS of an assignment or similar)
  - **undef(\( v \))**: the set of all variables whose value is undefined after executing the code corresponding to \( v \).
  - **c-use(\( v \))**: (c for computation) all variables that are used to define other variables in the code corresponding to \( v \)
  - **p-use(\( v, v' \))**: (p for predicate) all variables used in taking the \( (v,v') \) branch out of vertex \( v \)
  - \( v_0, v_1, \ldots, v_k \) is a **def-clear** path for \( x \), if \( x \) is not in **defs(\( v_i \))** for \( 0 < i < k \)
Example of a Def-Clear Path

- A, D, E is def-clear for x but not for y
- A, B, E is def-clear for y but not for x
Refinement

- We call a c-use of x global, if it is not preceded by a definition of x in the same basic block.
- We call a def of x global, if it is used in some other vertex in the flow graph.
- We refine our definitions only to take account of global uses and definitions (e.g. c-use(v) is the global c-uses in vertex v)
public int Segment(int t[], int l, int u) {
    // Assumes t is in ascending order, and l < u,
    // counts the length of the segment
    // of t with each element l < t[i] < u
    int k = 0;
    for(int i = 0; i < t.length && t[i] < u; i++) {
        if(t[i] > l) {
            k++;
        }
    }
    return k;
}
t, l, u defined
i = k = 0

\[ i < t.length \]

\[ t[i] < u \]

\[ t[i] > l \]

return k

k++
Data-flow Terminology

- \( \text{dcu}(x, v) = \{v' \in V \mid x \text{ is in } c\text{-use}(v') \text{ and there is a def-clear path for } x \text{ from } v \text{ to } v'\} \)
  - This is the set of vertices with c-uses of \( x \) that can potentially be influenced by the definition of \( x \) at \( v \)

- \( \text{dpu}(x, v) = \{(v', v'') \in E \mid x \text{ is in } p\text{-use}(v', v'') \text{ and there is a def clear path for } x \text{ from } v \text{ to } (v', v'')\} \)
  - This is the set of edges with p-uses of \( x \) that can potentially be influenced by the definition of \( x \) at \( v \).
Frankl and Weyuker’s data-flow coverage criteria

1. **All-defs** requires that for each definition of a variable $x$ in $P$, the set of paths $\Pi$ executed by the test set $T$ contains a def-clear path from the definition to at least one c-use or one p-use of $x$.

   ≈ all definitions get used.

2. **All-c-uses** requires that for each definition of a variable $x$ in $P$, and each c-use of $x$ reachable from the definition (see definition of $dcu(x,v)$), $\Pi$ contains a def-clear path from the definition to the c-use.

   ≈ all computations affected by each definition are exercised.

3. **All-p-uses** requires that for each definition of a variable $x$ in $P$, and each p-use of $x$ reachable from the definition (see definition of $dpu(x,v)$), $\Pi$ contains a def-clear path from the definition to the p-use.

   ≈ all branches affected by each definition are exercised.
Frankl and Weyuker’s data-flow coverage criteria

4. **All-c-uses/some-p-uses**: for each definition of $x$ in $P$ at $v$:
   - If $dcu(x,v)$ is not empty, the paths $\Pi$ executed by the test set $T$ contains a def-clear path from $v$ to each member of $dcu(x,v)$;
   - otherwise, the paths $\Pi$ executed by the test set $T$ contains a def-clear path from $v$ to an edge in $dpu(x,v)$.

   ≈ all definitions get used, and if they affect computations then all affected computations are exercised.

5. **All-p-uses/some-c-uses**: for each definition of $x$ in $P$ at $v$:
   - If $dpu(x,v)$ is not empty, the paths $\Pi$ executed by the test set $T$ contains a def-clear path from $v$ to each edge in $dpu(x,v)$;
   - otherwise, the paths $\Pi$ executed by the test set $T$ contains a def-clear path from $v$ to a member of $dcu(x,v)$.

   ≈ all definitions get used, and if they affect control flow then all affected branches are exercised.
Frankl and Weyuker’s data-flow coverage criteria

6. **All-uses** requires that for each definition of \( x \) at \( v \) in \( P \), the set of paths \( \Pi \) executed by the test set \( T \) contains a def-clear path from \( v \) to both \( dcu(x,v) \) and \( dpu(x,v) \).

   every computation and branch directly affected by a definition is exercised.

7. **All-du-paths** requires that for each definition of \( x \) at \( v \) in \( P \), the set of all paths \( \Pi \) executed by the test set \( T \) contains all def-clear paths from \( v \) to both \( dcu(x,v) \) and \( dpu(x,v) \), such that each path is loop free, or contains at most one loop of any loop on the path.

   all-uses, but requires exercise of all def-use paths, modulo looping.

8. **All-paths** requires that all paths through the program be executed.
Flow Graph, Revisited

t, l, u defined
i = k = 0

i++

i < t.length

f
t

f

return k

f

k++

f

t[i] < u

t

f

t[i] > l

t

i++
What is the **point** of all these distinctions?
Subsumption

- We say that test coverage criterion A *subsumes* test coverage criterion B if and only if, for every program P, every test set satisfying A with respect to P also satisfies B with respect to P.
- i.e. if any test set satisfying criterion A will (provably) always also satisfy B, then “A subsumes B”.
- Example: branch coverage subsumes statement coverage.
Subsumption relationships

All-paths

All-du-paths

All-uses

All-c-uses/Some p-uses

All-defs

All-p-uses

All-p-uses/Some-c-uses

Branch Coverage

Statement Coverage

Computation errors

Domain errors (wrong path)
Uses of Data Flow analysis

- We can use the analysis of definition and use to calculate optimistic and pessimistic estimates of whether variables are defined or not at particular vertices in the flow graph.
- We can use these to discover potential faults in the program.
- For example:
  - If a definition is only followed by definitions of the same variable - is it useful?
  - If we use a variable and it is not always preceded by a definition we might use it when it is undefined.
Summary

- Data-flow coverage criteria are claimed to provide a better measure of coverage than control flow because they track dependencies between variables in the flow graph.
- Frankl and Weyuker have done some empirical work on this (see references) and there is some justification for believing data-flow coverage is a good approach to structural testing.
- There are the usual issues of the computability of the exact relationships between definition and use but we are usually satisfied with approximations.
References for Coverage (available from Web page)


- Background reading