An Applicable Family of Data Flow Testing Criteria

- Assumptions about the program
  - No goto statements
  - variant records
  - Functions having ‘var’ parameters
    - By reference
    - Procedural or functional parameters
  - Conformant arrays
    - size of an array parameter is not known to the called function until run-time
  - Every Boolean expression that determines the flow of control has at least one occurrence of a variable or a call to the function 'eof' or 'eoln'

Program Structure

- Program consists of 'blocks'
- Block
  - Sequence of statements
    - Whenever the first statement is executed, the remaining statements in the block are executed in the given order
  - Can be represented by a flow graph

Classifying each variable occurrence

- Definition
  - Value is stored in a memory location
- Use
  - Value is fetched from a memory location
- Undefinition
  - Value and location becomes unbound
- C-use
  - Use in a computation or output statement
    - Associated with each node
- P-use
  - Use in a predicate
    - Associated with each edge

Simple Statements

Assignment statement: \( v := \text{expr} \);

Node i has c-uses of each variable in expr followed by a definition of \( v \).

Input/Output statements:

\[
\begin{align*}
\text{read}(v_1, \ldots, v_n); & \\
\text{readln}(v_1, \ldots, v_n); & \\
\text{read}(f, v_1, \ldots, v_n); & \\
\text{readln}(v_1, \ldots, v_n); & \\
\end{align*}
\]

Node i has definitions of \( v_1, \ldots, v_n \).

If the file variable \( f \) is present then node i also has a c-use followed by a definition of \( f \).

\[
\begin{align*}
\text{write}(v_1, \ldots, v_n); & \\
\text{writeln}(v_1, \ldots, v_n); & \\
\text{write}(f, v_1, \ldots, v_n); & \\
\text{writeln}(f, v_1, \ldots, v_n); & \\
\end{align*}
\]

Node i has c-uses of each variable occurring in \( v_1, \ldots, v_n \).

If the file variable \( f \) is present then node i also has a definition followed by a c-use of \( f \).

Procedure call: \( P(e_1, \ldots, e_n) \);

Node j has c-uses of each variable occurring in the expressions \( e_1, \ldots, e_n \).

These are followed by definitions of each actual parameter which corresponds to an actual parameter.

Nodes i and k are included to assure that the procedure call has its own node.
Repetitive Statements

while statement: while B do S;

Let h be the entry node to subgraph S.
Edges (i,h) and (h,i) have p-uses of each variable in the boolean expression B.

Repetitive Statements

for statement:
for v->e1 to e2 do S;

Let tmp be a new variable.
Let f and g be the entry and exit nodes, respectively, of S. Node h has c-uses of each variable in e1.
followed by a definition of v and c-use of each variable in e2
followed by a definition of tmp.
Edges (i,f) and (h,g) have p-uses of v and tmp. Node g has a c-use followed by a def of v.

Repetitive Statements

repeat statement:
repeat S1,...,Sn until B;

Let i be the entry node of S1, and let j be the exit node of Sn.
Edges (i,j) and (j,i) have p-uses of each variable in the boolean expression B.

Conditional Statements

if-then-else statement
if B then S1;
if B then S1 else S2;

Let k and j be the entry nodes of S1 and S2, respectively.
Edges (i,k) and (j,i) have p-uses of each variable in the boolean expression B.
if there is no "else" part then subgraph S2 has a single node corresponding to an empty block.

Conditional Statements

case e1 of
label-xax : S1;
label-xbx : S2
end;

Let j1,...,jn be the entry nodes of S1,...,Sn, respectively.
Edges (i,j1),...,(i,jm) have p-uses of each variable in the expression e1.

Entry and exit nodes

- Entry node
  - Has the definition of
    - Each parameter
    - Input buffer input

- Exit node has
  - An undefined of each local variable
  - A c-use of each variable parameter
  - A c-use of each non-local variable
  - A c-use of the input buffer input
**Arrays**

- It is impossible to determine the particular array element which is being used or defined in an occurrence of an array variable
  - `A[i+j]`
- Definition of `A[expr]`
  - A c-use of each variable in `expr`
  - Followed by a definition of `a`
- Use of `A[expr]`
  - c-uses of all the variables in `expr`
  - Followed by a use of `a`

**Pointers**

- Impossible to determine statically the memory location to which a pointer points
- Syntactic treatment
- If `p` is a pointer variable
  - Definition of `p`
    - C-use of `p`
    - Followed by a definition of `p`
  - Use of `p`
    - C-use of `p`
    - Followed by a c-use of `p`
  - Ignore definitions and uses of `p`

**Records & Files**

- Records
  - Each field is treated as an individual variable
  - Any unqualified occurrence of a record is treated as an occurrence of each field
- File variables
  - Considering the effect on the file buffer

**Simplifying Assumptions**

- No interprocedural dataflow analysis
- Ignore pointers
- Array reference simplification
- No aliasing/side-effects
- Consequences
  - Perhaps "less than perfect" test data

**Global Definition**

- Global c-use
  - A c-use of `x` in node `i` is global if `x` has been assigned in some block other than `i`
- Def-clear path wrt `x"from node `i to node `j" and "from node `i to edge `(n_m, `j)"
  - A path `(i, n_1, n_2, ..., n_m, `j)` containing no definitions or undefinitions of `x` in nodes `n_1, n_2, ..., n_m`
- Global definition of `x`
  - A node `i` has a global definition of a variable `x` if
    - it has a definition of `x` and
    - there is a def-clear path wrt `x` from node `i` to some node containing
      - a global c-use of `x`
      - edge containing a p-use of `x`

**Restricted Programs Class**

- Satisfying the following properties
  - NSUP
    - No-syntactic-undefined-p-use Property
      - For every p-use of a variable `x` on an edge `(i, `j)` in `P`, there is some path from the start node to edge `(i, `j)`, which contains a global definition of `x`
  - NSL
    - Non-straight-line property
      - `P` has at least one conditional or repetitive statement
        - At least one node in `P`’s flow-graph has more than one successor
        - At least one variable has a p-use in `P`
Def-use graph

- Obtained from the flow graph
- Associate with each node the sets
  - C-use(i)
    - Variables which have global c-uses in block i
  - Def(i)
    - Variables which have global definitions in block i
- Associate with each edge \((i,j)\)
  - P-use(i,j)
    - Variables which have p-uses on edge \((i,j)\)
- Define sets of nodes
  - dcu(x,i)
    - Nodes \(j\) such that \(x \in C\)-use\((j)\) and there is a def-clear path with respect to \(x\) from \(i\) to \(j\)
  - dpu(x,i)
    - Edges \((j,k)\) such that \(x \in P\)-use\((j,k)\) and there is a def-clear path with respect to \(x\) from \(i\) to \((j,k)\)

Definitions for def-use graph

- \(V\) = the set of variables
- \(N\) = the set of nodes
- \(E\) = the set of edges
- \(\text{def}(i)\) = \(\{x \in V \mid x\ has a global definition in block i\}\)
- \(\text{c-use}(i)\) = \(\{x \in V \mid x\ has a global c-use in block i\}\)
- \(\text{p-use}(i)\) = \(\{x \in V \mid x\ has a p-use in edge (i,j)\}\)
- \(\text{dcu}(x,i)\) = \(\{j \in N \mid x \in C\)-use\((j)\) and there is a def-clear path wrt \(x\) from \(i\) to \((j,k)\)\}
- \(\text{dpu}(x,i)\) = \(\{(j,k) \mid x \in P\)-use\((j,k)\) and there is a def-clear path wrt \(x\) from \(i\) to \((j,k)\)\}

Explanations

- If \(x \in \text{def}(i)\) and \(j \in \text{dcu}(x,i)\), then
  - \(x\) has a global definition in node \(i\) and
  - A c-use in node \(j\), and
  - There is a definition clear path with respect to \(x\) from \(i\) to \(j\)
- Hence
  - It may be possible for control to reach node \(j\) with the variable \(x\) having the value which was assigned to it in node \(i\)

More definitions

- Definition-c-use association
  - Triple \((i,j,x)\) where \(i\) is a node containing a global definition of \(x\) and \(j \in \text{dcu}(x,i)\)
- Definition-p-use association
  - Triple \((i,(j,k),x)\) where \(i\) is a node containing a global definition of \(x\) and \((j,k) \in \text{dpu}(x,i)\)
- A path \((n_1,n_2, \ldots, n_l, n_k)\) is a du-path wrt \(x\) if
  - \(n_l\) has a global c-use of \(x\) and \((n_1, \ldots, n_l, n_k)\) is a def-clear simple path wrt \(x\), and
  - \((n_j, n_k)\) has a p-use of \(x\) and \((n_1, \ldots, n_j)\) is a def-clear loop-free path wrt \(x\)
- An association is a definition-c-use association, a definition-p-use association, or a du-path

Yet more definitions

- Complete path
  - Path from the entry node to the exit node
- Covering
  - A complete path \(\pi\) covers a definition-c-use association \((i,j,x)\) if \(\pi\) has a definition clear subpath wrt \(x\) from \(i\) to \(j\)
  - A complete path \(\pi\) covers a definition-p-use association \((i,(j,k),x)\) if \(\pi\) has a definition clear subpath wrt \(x\) from \(i\) to \((j,k)\)
  - \(\pi\) covers a du-path \(\pi'\) if \(\pi'\) is a subpath of \(\pi\)
  - The set \(\Pi\) of paths covers an association if some element of the set does
  - A test set \(T\) covers an association if the elements of \(T\) cause the execution of the set of paths \(\Pi\), and \(\Pi\) covers the association

Finally, the criteria

- Intuitively
  - The family of DF testing criteria is based on requiring that
  - the test data execute definition-clear paths from each node containing a global definition of a variable to specified nodes containing
    - global c-uses and
    - edges containing p-uses of that variable
  - For each variable definition, the criteria require that
    - All/some definition-clear paths wrt that variable from the node containing the definition to all/some of the uses/c-uses/p-uses reachable by some such paths be executed
**All-defs criterion**

- If variable \( x \) has a global definition in node \( i \), the all-defs criterion requires the test data to exercise *some* path which goes from \( i \) to *some* node or edge at which the value assigned to \( x \) in node \( i \) is used.

**All-uses criterion**

- If variable \( x \) has a global definition in node \( i \), the all-uses criterion requires the test data to exercise *at least one* path which goes from \( i \) to *each* node and edge at which the value assigned to \( x \) in node \( i \) is used.

**All-DU-paths criterion**

- If variable \( x \) has a global definition in node \( i \), the all-DU-paths criterion requires the test data to exercise *all* paths which go from \( i \) to *each* node and edge at which the value assigned to \( x \) in node \( i \) is used.

**Other DF testing criteria**

- All-\( p \)-uses
- All-\( c \)-uses
- All-\( p \)-uses/some-\( c \)-uses
- All-\( c \)-uses/some-\( p \)-uses

**Definitions of DF criteria**

<table>
<thead>
<tr>
<th>CRITERION</th>
<th>ASSOCIATIONS REQUIRED</th>
</tr>
</thead>
<tbody>
<tr>
<td>All-defs</td>
<td>Some (( D_A ) ( x ), ( B_A ) ( i )) or (( D_A ) ( x ), ( B_A ) ( n ))</td>
</tr>
<tr>
<td>All-uses</td>
<td>All-( D_A ) ( x ), ( B_A ) ( i ) or All-( D_A ) ( x ), ( B_A ) ( n )</td>
</tr>
<tr>
<td>All-DU-paths</td>
<td>All ( D_A ) ( x ), ( B_A ) ( i ) and All ( D_A ) ( x ), ( B_A ) ( n )</td>
</tr>
<tr>
<td>All-( p )-uses</td>
<td>All ( D_A ) ( x ), ( B_A ) ( i ) and All ( D_A ) ( x ), ( B_A ) ( n )</td>
</tr>
<tr>
<td>All-( c )-uses</td>
<td>All ( D_A ) ( x ), ( B_A ) ( i ) and All ( D_A ) ( x ), ( B_A ) ( n )</td>
</tr>
<tr>
<td>All-( p )-uses/some-( c )-uses</td>
<td>All ( D_A ) ( x ), ( B_A ) ( i ) and All ( D_A ) ( x ), ( B_A ) ( n )</td>
</tr>
<tr>
<td>All-( c )-uses/some-( p )-uses</td>
<td>All ( D_A ) ( x ), ( B_A ) ( i ) and All ( D_A ) ( x ), ( B_A ) ( n )</td>
</tr>
</tbody>
</table>

**“includes”**

- Criterion \( C_1 \) includes criterion \( C_2 \) iff:
  - For every subprogram, any test set that satisfies \( C_1 \) also satisfies \( C_2 \).
- \( C_1 \) strictly includes \( C_2 \) iff:
  - denoted \( C_1 \Rightarrow \not C_2 \),
  - \( C_1 \) includes \( C_2 \) and for some subprogram \( P \) there is a test set that satisfies \( C_2 \) but does not satisfy \( C_1 \).
Includes relationship

Applicability
- It may be the case that no test set for program P satisfies criterion C
  - Infeasible paths
- Tailor the DF criteria so that they are applicable
  - Assumptions
    - All aliases are known
    - All side effects are known
    - No element of the test set causes the program to crash
      - Execution of entry node to exit node

Executable/Feasible Paths
- Recall
  - Complete path
    - Path from the entry node to the exit node
- Executable/feasible complete path
  - A complete path that is executed on some assignment of values to input variables
- Executable/feasible path
  - A subpath of an executable complete path

Executable Associations
- Definition
  - An association is executable if there is some executable complete path that covers it; otherwise it is unexecutable
  - \( fdcu(x,i) \in dcu(x,i) \)
    - Nodes \( j \) such that \( x \in c\text{-use}(j) \) and there is an executable definition clear path wrt \( x \) from \( i \) to \( j \)
  - \( fdpu(x,i) \in dpu(x,i) \)
    - Edges \( (j,k) \) such that \( x \in p\text{-use}(j,k) \) and there is an executable definition clear path wrt \( x \) from \( i \) to \( (j,k) \)

Executable/Feasible Paths
- Recall
  - Complete path
- Executable/feasible complete path
- Executable/feasible path
  - A subpath of an executable complete path

Executable Associations
- Definition
  - An association is executable if there is some executable complete path that covers it; otherwise it is unexecutable

Recall Definition
- Definition-c-use association
  - Triple \( (i,j,x) \) where \( i \) is a node containing a global definition of \( x \) and \( j \in dcu(x,i) \)
- Definition-p-use association
  - Triple \( (i,(j,k),x) \) where \( i \) is a node containing a global definition of \( x \) and \( (j,k) \in dpu(x,i) \)
  - A path \( (n_1, n_2, \ldots, n_i, n_k) \) is a du-path wrt \( x \) if \( n_i \) has a global definition of \( x \) and either
    - \( n_k \) has a global c-use of \( x \) and \( (n_1, \ldots, n_i, n_k) \) is a def-clear simple path wrt \( x \), and
    - \( (n_i, n_k) \) has a p-use of \( x \) and \( (n_1, \ldots, n_i) \) is a def-clear loop-free path wrt \( x \)
  - An association is a definition-c-use association, a definition-p-use association, or a du-path

Intuitively
- new criterion \( C^* \) for each DF criterion \( C \)
  - By selecting the required associations from \( fdcu(x,i) \) and \( fdpu(x,i) \) instead of from \( dcu(x,i) \) and \( dpu(x,i) \)
Feasible Data-flow Criteria (FDF)

<table>
<thead>
<tr>
<th>CRITERION</th>
<th>REQUIRED ASSOCIATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>(all-def)*</td>
<td>(l,k)(j,i) &amp; (p,k)(j,i) or some (l,k)((j,i)) or some (j,i)((l,k)) or some (p,k)((j,i)) or some (j,i)((p,k)) or some (l,k)((i,j)) or some (i,j)((l,k)) or some (p,k)((i,j)) or some (i,j)((p,k))</td>
</tr>
<tr>
<td>(all-c-use)*</td>
<td>(l,k)(j,i) &amp; (p,k)(j,i) or some (l,k)((j,i)) or some (j,i)((l,k)) or some (p,k)((j,i)) or some (j,i)((p,k)) or some (l,k)((i,j)) or some (i,j)((l,k)) or some (p,k)((i,j)) or some (i,j)((p,k))</td>
</tr>
<tr>
<td>(all-p-some-p-use)*</td>
<td>(l,k)(j,i) &amp; (p,k)(j,i) or some (l,k)((j,i)) or some (j,i)((l,k)) or some (p,k)((j,i)) or some (j,i)((p,k)) or some (l,k)((i,j)) or some (i,j)((l,k)) or some (p,k)((i,j)) or some (i,j)((p,k))</td>
</tr>
<tr>
<td>(all-use)*</td>
<td>(l,k)(j,i) &amp; (p,k)(j,i) or some (l,k)((j,i)) or some (j,i)((l,k)) or some (p,k)((j,i)) or some (j,i)((p,k)) or some (l,k)((i,j)) or some (i,j)((l,k)) or some (p,k)((i,j)) or some (i,j)((p,k))</td>
</tr>
<tr>
<td>(all-p-use)*</td>
<td>(l,k)(j,i) &amp; (p,k)(j,i) or some (l,k)((j,i)) or some (j,i)((l,k)) or some (p,k)((j,i)) or some (j,i)((p,k)) or some (l,k)((i,j)) or some (i,j)((l,k)) or some (p,k)((i,j)) or some (i,j)((p,k))</td>
</tr>
<tr>
<td>(all-def-path)*</td>
<td>(l,k)(j,i) &amp; (p,k)(j,i) or some (l,k)((j,i)) or some (j,i)((l,k)) or some (p,k)((j,i)) or some (j,i)((p,k)) or some (l,k)((i,j)) or some (i,j)((l,k)) or some (p,k)((i,j)) or some (i,j)((p,k))</td>
</tr>
</tbody>
</table>

Includes Relationships

Most DF testing methodologies deal with dependencies that exist within a procedure (i.e., intra-procedural).

Data dependencies also exist among procedures.

Requires analysis of the flow of data across procedure boundaries.

Calls and Returns

Direct dependencies (single call/return)

Indirect dependencies (multiple calls/returns)

The Def-uses

A test case

Execute and check

All def-use pairs are covered
module Max
declare
S is any 1..N of integer;
LMAX,MAX: integer;
begin
for I := 1 to N do read(S[I]);
getMax(S,MAX);  // write LMAX
end;

procedure GetMax;
input
P: integer;
M: reference integer;
declare M, MAX: integer;
begin
if P = M then PutMax(P);  // LMAX, MAX
else begin
M := P;  // LMAX, MAX
GetMax(P);  // P = MAX
end;
end;

procedure PutMax;
input
item: reference integer;
begin
if item > MAX then K := 1
else K := J;
end;

procedure DelMax;
input
K: integer;
begin
end.

Any missed def-uses?