LIMKOKWING UNIVERSITY OF CREATIVE TECHNOLOGY

BOTSWANA

BTECH3618 • Software Testing and Reliability

CHAPTER 5

Data Flow Testing

Partition Testing

- • Comparison with path testing techniques
  – Path testing techniques are based on control flow, not content sensitive.
  – SC/BC – weak criteria
  – Path coverage – too many paths, (approximation)
- • Cause data computation to be used.
- • Look for data flow anomalies

Anomalie – an illogical or useless sequence of data object state change;

Definitions

P: Function or procedure under test

Control Flow Graph (CFG) = (N, E, *nb, ne*)

Path, Subpath, initial path

subpath – A sequence of arcs through the CFG. (n1,n2,…,nk) where (ni,ni+1) ∈E for $1 \leq i \leq k$

Data Flow Terms

X: a variable in P

- Definition (def): gives a value to X;
- Undefinition(kill): X's value is no longer know with certainty;
- Use(use): value of X is accessed.
  - Computation-use(c-use): X is used in a computation
  - Predicate-use(p-use): X is used in a predicate

Def-clear Subpath

- Def-clear subpath with respect to X is a path
- (i, n1,n2,…,nk,j) where there are no defs or
- undefs of X in any of the nodes n1,n2,…,nk.

Example

Def-clear Path:

1-2-4-5

3-4-5

Not:

1-3-4-5

Data Flow Terms

- Global c-use: no definition of X preceding
- the use in the same block;
- • Local c-use: A def of X precedes the use in
- the same block
- A = B +C; C is a global c-uses
- E = A * A; A is a local uses
- • Global-def: def of X in a basic block is used
- in a predicate, or there is a global c-use in
- another basic block;

Data Flow Terms

- V – set of variables
- N – Set of nodes(Basic blocks)
- E – set of edges
- def(i) = { x ∈V | X has a global def in block i}
- c-use(i) ={x∈V | X has a global c-use in block i}
- p-use(i, j) ={x∈V|X has a p-use on edge (i,j)}

Data Flow Terms

- dcu(x,i) = { j ∈N | X ∈ c-use(j) and a def-clear path with respect to x from i to j}
- dpu(x,i) = {(j,k) ∈N | X ∈ p-use(j,k) and a def-clear path with respect to x from i to (j,k)}

Frankl and Weyuker's Data Flow Criteria

- All-defs requires that for each definition of a variable X in P, the set of paths Π executed by the test set T contains a definition-clear subpath from the definition to at least one c-use or one p-use of X.
- All-c-uses requires that for each definition of a variable X in P, and each c-use of X reachable from the definition, Π contains a definitionclear subpath from the definition to the c-use.
- All-p-uses requires that for each definition of a variable X in P, and each p-use of X reachable from the definition, Π contains a definitionclear subpath from the definition to the p-use.

Frankl and Weyuker's Seven Data Flow Criteria

- All-c-uses/some-p-uses that for each definition of X in p, if there exists
  at least one c-use of X reachable from the definition, Π contains a definition-clear subpath from the definition to at all reachable c-uses of X, otherwise, Π contains a definition-clear subpath from the definition to a reachable p-use of X.

- All-p-uses/some-c-uses that for each definition of X in p, if there exists
  at least one p-use of X reachable from the definition, Π contains a definition-clear subpath from the definition to at all reachable p-uses of X, otherwise, Π contains a definition-clear subpath from the definition to a reachable c-use of X.

Frankl and Weyuker's Seven Data Flow Criteria

- All-uses requires that for each definition of X in P, Π contains a
  definition-clear subpath from the definition to all reachable c-uses and
  p-uses of X.
- All-du-paths requires that for each definition of X in P, Π contains all
  definition-clear subpaths from the definition to all reachable c-uses and
  p-uses of X, such that each subpath contains no loops, or contains one
  complete loop.
- • All-paths requires that all paths through the program be executed.

Data Flow Anomalies

- dd – A def followed by a def
  Missing use or incorrect def
- dk – def than killed
  Missing use or incorrect def or undefinition
- du / kd – normal
- kk – killed then killed
- ku – kill then use (fault)
- ud,uk,uu - normal

Data Flow Anomalies

- means nothing happens with X before or after current reference

-k: nothing to kill

-d: Normal

-u: fault (unless X is global)

k-: Normal

d-: definition not used (unless X is global)

u-: normal (unless X should be killed)

Example

```
A=0;
While (A>0) {
if (A>10)
A = A – B;
else
B = B + 1;
}
print A,B
```

Data Flow Anomalies

- mindef (i) = set of variables that will have definitions
  at node n;
  mindef(i) = ∩ [mindef(p(i)) ∪ def(p(i))]; p(i) – parents of i
  initial value of mindef of all nodes = {all variables}
- Defclear(i) =all variables that have def-clear paths,
  (t, n1,n2, …, nk, i);
  defclear(i) = ∪ [defclear(p(i)) ∪ def(p(i))] – use(i))
  initial value of defclear of all nodes= {}

Example

```
A=0;
While (A>0)
{
if (A>10)
A = A – B;
else
B = B + 1;
}
print A,B
```

Example – Step (8)

• Step 0:Derive all the def, use set
• Step 1:Initialization, make all mindef(i) = {ALL}, defclear(I)={}.
• Step 2:mindef(start) = ∩ {} = {}
defclear(start) = ∪ {} – use(start) = {}
• Step 3:mindef(1)= [mindef(start) ∪def(start)] = {}
defclear(1)= [defclear(start) ∪def(start)] – use(1) = {}
• Step 4:mindef(2)= [mindef(1) ∪def(1)] ∩ [mindef(4) ∪def(4)] ∩ [mindef(5) ∪def(5)]
={A} ∩ {ALL} ∩ {ALL} = {A}
defclear(2)=[defclear(1)∪def(1)]∪[defclear(4)∪def(4)]∪[defclear(5) ∪def(5)]- use(2)
= {A} ∪ {A} ∪ {B} – {A} = {B}
• Step 5:mindef(3) = [mindef(2) ∪ def(2)] = {A}
defclear(3)= [defclear(2) ∪ def(2)] – use(3) = {B} – {A} = {B}
• Step 6:mindef(4) = [mindef(3) ∪ def(3)] = {A}
defclear(4)= [defclear(3) ∪ def(3)] – use(4) = {B} – {A, B} = {}
• Step 7:mindef(5) = [mindef(3) ∪ def(3)] = {A}
defclear(5)= [defclear(3) ∪ def(3)] – use(5) = {B} – {B} = {}
• Step 8:mindef(6) = [mindef(2) ∪ def(2)] = {A}
defclear(6)= [defclear(2) ∪ def(2)] – use(6) = {B} – {A,B} = {}
mindef(end) = [mindef(6) ∪ def(6)] = {A}
defclear(end)= [defclear(6) ∪ def(6)] – use(end) = {} – {} = {}
When trying to calculate for the second time, all sets remain the same, stop.

Identify Faults and Data Flow Anomalies

d- : defclear(exitnode) <> {}

dd: defclear(i) ∩ def(i) <> {}

-u: use(i) - mindef(i) <> {}

Identify Faults and data Flow Anomalies

- NODE 1 dd: defclear(1) ∩ def(1) = {} ∩ {A} = {} √

-u: use(i) - mindef(i) = {} – {} = {} √

- NODE 2 dd: defclear(2) ∩ def(2) = {A} ∩ {} = {} √

-u: use(i) - mindef(i) = {A} – {A} = {} √

- NODE 3 dd: defclear(3) ∩ def(3) = {} ∩ {} = {} √

-u: use(i) - mindef(i) = {A} – {A} = {} √

- NODE 4 dd: defclear(4) ∩ def(4) = {} ∩ {A} = {} √

-u: use(i) - mindef(i) = {A,B} – {A} = {B} X -u type Fault

- NODE 5 dd: defclear(5) ∩ def(5) = {A} ∩ {B} = {} √

-u: use(i) - mindef(i) = {B} – {A} = {B} X -u type Fault

- NODE 6 dd: defclear(6) ∩ def(6) = {} ∩ {} = {} √

-u: use(i) - mindef(i) = {A,B} – {A} = {B} X -u type Fault

- EndNode d-: delclear(end) = {} √

Discussion

- Unachievable d-u pair
- Array, pointer
- Inter – procedure data-flow analysis