Data flow testing
Learning objectives

• Understand why data flow criteria have been designed and used
• Recognize and distinguish basic DF criteria
  - All DU pairs, all DU paths, all definitions
• Understand how the infeasibility problem impacts data flow testing
• Appreciate limits and potential practical uses of data flow testing
**Motivation**

- **Middle ground in structural testing**
  - Node and edge coverage don’t test interactions
  - Path-based criteria require impractical number of test cases
    - And only a few paths uncover additional faults, anyway
  - Need to distinguish “important” paths

- **Intuition: Statements interact through data flow**
  - Value computed in one statement, used in another
  - Bad value computation revealed only when it is used
Data flow concept

- Value of x at 6 could be computed at 1 or at 4
- Bad computation at 1 or 4 could be revealed only if they are used at 6
- (1,6) and (4,6) are def-use (DU) pairs
  - defs at 1,4
  - use at 6
Terms

- DU pair: a pair of *definition* and *use* for some variable, such that at least one DU path exists from the definition to the use
  
  \[ x = \ldots \text{ is a definition of } x \]
  
  \[ = \ldots x \ldots \text{ is a use of } x \]

- DU path: a definition-clear path on the CFG starting from a definition to a use of a same variable
  
  - Definition clear: Value is not replaced on path
  - Note - loops could create infinite DU paths between a def and a use
Definition-clear path

- 1,2,3,5,6 is a definition-clear path from 1 to 6
  - x is not re-assigned between 1 and 6

- 1,2,4,5,6 is not a definition-clear path from 1 to 6
  - the value of x is "killed" (reassigned) at node 4

- (1,6) is a DU pair because 1,2,3,5,6 is a definition-clear path
Adequacy criteria

- All DU pairs: Each DU pair is exercised by at least one test case
- All DU paths: Each *simple* (non looping) DU path is exercised by at least one test case
- All definitions: For each definition, there is at least one test case which exercises a DU pair containing it
  - (Every computed value is used somewhere)

Corresponding coverage fractions can also be defined
Difficult cases

- \( x[i] = \cdots; \ \cdots; y = x[j] \)
  - DU pair (only) if \( i==j \)
- \( p = \&x; \ \cdots; *p = 99; \ \cdots; q = x \)
  - \( *p \) is an alias of \( x \)
- \( m.putFoo(\ldots); \ \cdots; y=n.getFoo(\ldots); \)
  - Are \( m \) and \( n \) the same object?
  - Do \( m \) and \( n \) share a “foo” field?

- Problem of *aliases*: Which references are (always or sometimes) the same?
Data flow coverage with complex structures

- Arrays and pointers are critical for data flow analysis
  - Under-estimation of aliases may fail to include some DU pairs
  - Over-estimation, on the other hand, may introduce unfeasible test obligations
- For testing, it may be preferrable to accept under-estimation of alias set rather than over-estimation or expensive analysis
  - Controversial: In other applications (e.g., compilers), a conservative over-estimation of aliases is usually required
  - Alias analysis may rely on external guidance or other global analysis to calculate good estimates
  - Undisciplined use of dynamic storage, pointer arithmetic, etc. may make the whole analysis infeasible
Infeasibility

- Suppose \( \text{cond} \) has not changed between 1 and 5
  - Or the conditions could be different, but the first implies the second

- Then (3,5) is not a (feasible) DU pair
  - But it is difficult or impossible to determine which pairs are infeasible

- Infeasible test obligations are a problem
  - No test case can cover them
Infeasibility

- The path-oriented nature of data flow analysis makes the infeasibility problem especially relevant
  - Combinations of elements matter!
  - Impossible to (infallibly) distinguish feasible from infeasible paths. More paths = more work to check manually.

- In practice, reasonable coverage is (often, not always) achievable
  - Number of paths is exponential in worst case, but often linear
  - All DU paths is more often impractical
Summary

• Data flow testing attempts to distinguish “important” paths: Interactions between statements
  • Intermediate between simple statement and branch coverage and more expensive path-based structural testing

• Cover Def-Use (DU) pairs: From computation of value to its use
  • Intuition: Bad computed value is revealed only when it is used
  • Levels: All DU pairs, all DU paths, all defs (some use)

• Limits: Aliases, infeasible paths
  • Worst case is bad (undecidable properties, exponential blowup of paths), so pragmatic compromises are required