STRUCTURAL TESTING – PATH TESTING (PART II)

Daniel Sinnig, PhD
d_sinnig@cs.concordia.ca

Department for Computer Science
and Software Engineering

20. Jan-14

The baseline method

- Developed by McCabe (1987)
- Systematic approach to determine the set of basis paths.
- The method will return a minimal set of basis paths
- However, depending on the choice of the first 'baseline' path, this set may not be unique.
- Mathematical background:
  - A path p is a linear combination of paths $p_1, \dots, p_n$ iff there are integers $a_1, \dots, a_n$ such that $p = \sum_{i=1}^{n} a_i p_i$ (in the vector representation)
  - A set of paths is linearly independent iff no path in the set is a linear combination of any other paths in the set.

The baseline method (cont.)

Algorithm:

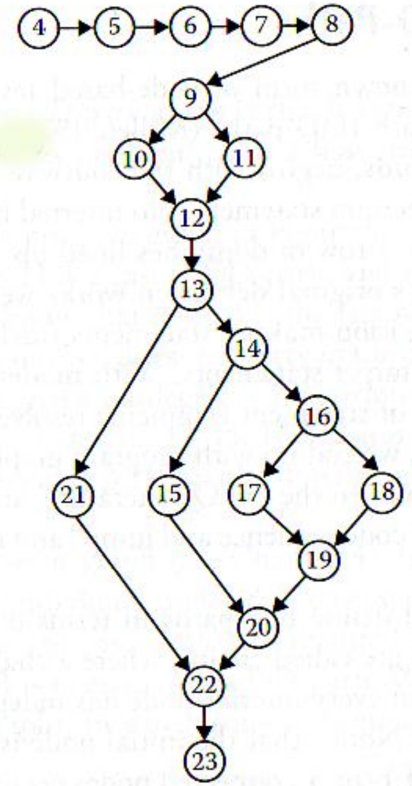
- Step 0: Initialize set of baseline paths B {}
- Step 1: Pick a functional "baseline" path (p1) through the program (a typical run through the program).
- Step 2: Add p1 to B
- Step 3: While there are 'unflipped' (binary) decision nodes do
  - Step 3.1: Pick path p from B
  - Step 3.2: Generate the next baseline path p_{next} by "flipping" the first decision node (n_d) of p. Should p_{next} rejoin p, it must follow it until the end.
  - Step 3.3: Add p_{next} to the set of basis paths. B
  - Step 3.4: Mark n_d as flipped

The baseline method (cont.)

- **Remarks**
  - Multi-way decisions (e.g., switch nodes) must be "flipped" to each of their decision outcomes
  - If the CFG only contains binary decision then the minimal number of basis paths can also be calculated as: Number of decision nodes + 1
  - Criticism:
    - May return infeasible paths due to data dependencies which conflict with the independency assumption of basis paths

Exercise (5-10min): Determine the set of basis paths for the following CFG. Are all paths feasible?

```
Program triangle2
Dim a,b,c As Integer
Dim IsATrinagle As Boolean
Output("Enter 3 integers which are sides of a triangle")
Input(a,b,c)
Output("Side A is", a)
Output("Side B is", b)
Output("Side C is", c)
If (a < b + c) AND (b < a + c) AND (c < a + b)
  Then IsATriangle = True
  Else IsATriangle = False
EndIf
If IsATriangle
  Then  If (a = b) AND (b = c)
         Then Output ("Equilateral")
         Else  If (a≠b) AND (a≠c) AND (b≠c)
                  Then Output ("Scalene")
                  Else Output ("Isosceles")
               EndIf
         EndIf
  Else  Output("Nota a Triangle")
EndIf
End triangle2
```



Path Testing Process

- Input:
  - Source code and a path selection criterion
- Process:
  - Generation of a CFG
  - Selection of Paths
  - Generation of Test Input Data
  - Feasibility Test of a Path
  - Evaluation of Program's Output for the Selected Test Cases

Minimum number of test cases

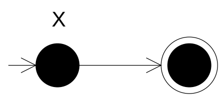
- Knowing the minimum number of test cases for a given coverage criteria required to estimate the effort required for testing.
- This requires some theory: Prime Decomposition Theorem

Prime Decomposition Theorem

Prime Decomposition Theorem: Any structured program graph can be uniquely decomposed into a hierarchy of sequencing and nesting *primes*.

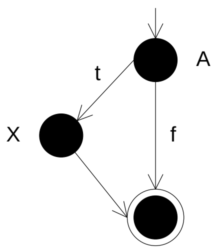
Primes and their Flowgraphs

P_1



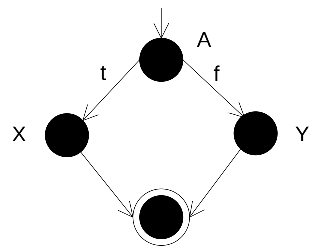
P_1(X) [X;]

D_0



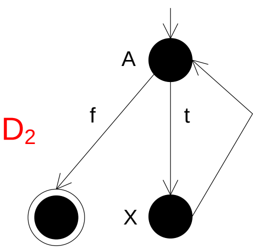
D_0(A,X) [if A then X]

D_1



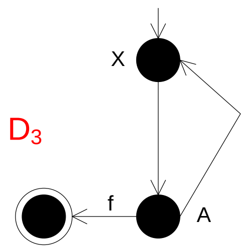
D_1(A,X,Y) [if A then X else Y]

D_2



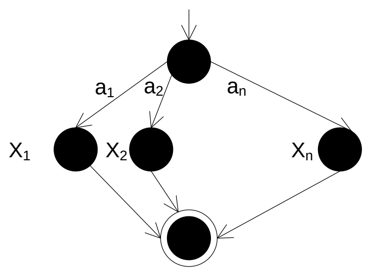
D_2(A, X) [while A do X]

D_3



D_3(A, X) [do X while A]

C_n



$C_n(A, X_1, \ldots, X_n)$
[
switch A
  case a1: X_1
  case a2: X_2
  …
  case an: X_n
]

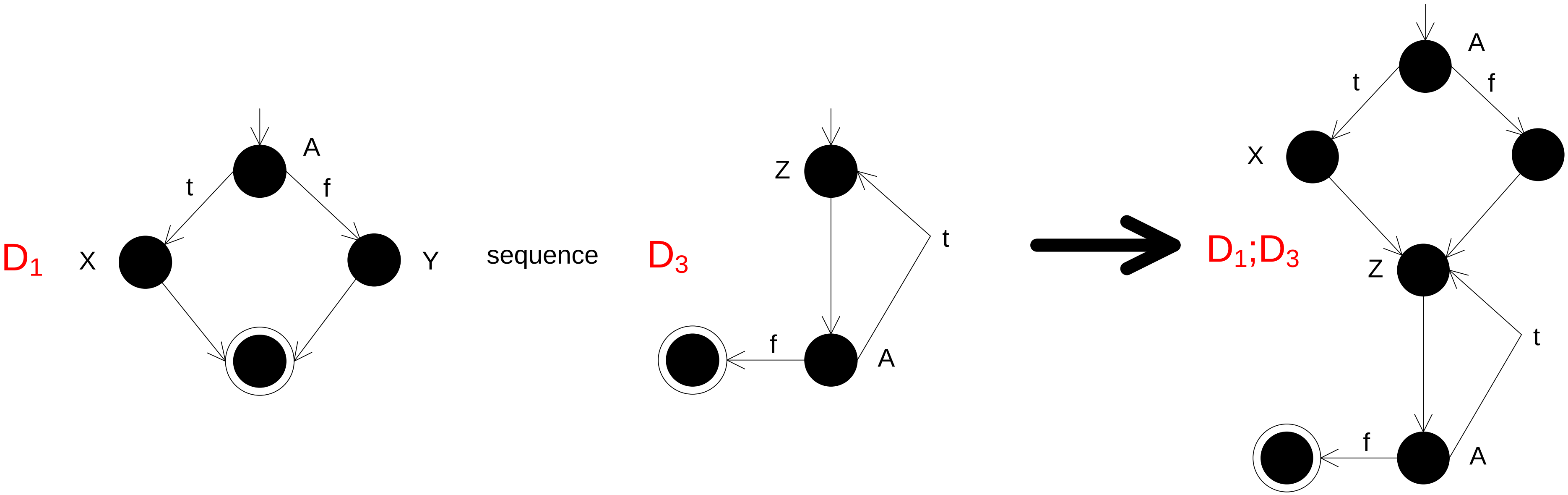
X, X1, …, Xn, Y are statements

Sequencing

Sequencing: Let F_1 and F_2 be two flowgraphs. Then, the sequence of F_1 and F_2 (shown by $F_1; F_2$ or $P_2(F_1, F_2)$) is a flowgraph formed by merging the terminal node of F_1 with the start node of F_2.

Sequencing can be extended in a straightforward manner to n flowgraphs: $F_1; F_2; \dots; F_n$ or $P_n(F_1, F_2, \dots, F_n)$

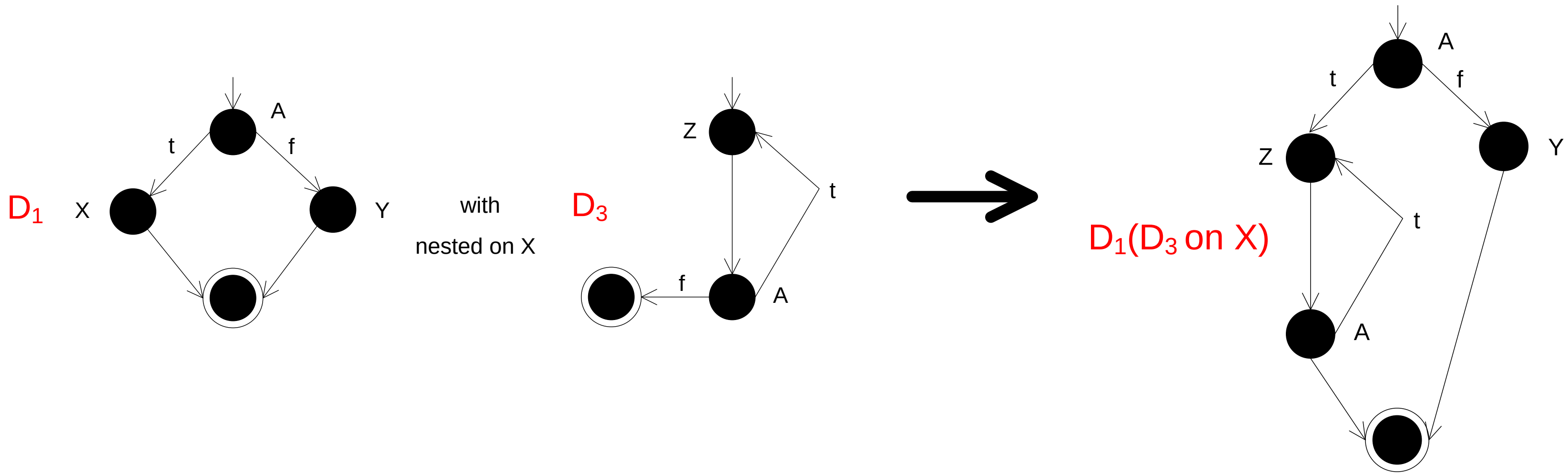
Sequencing (Example)



Nesting

Let F_1 and F_2 be two flowgraphs. Then, the nesting of F_2 onto F_1 at x shown by F_1 $(F_2 \; on \; x)$ is a flowgraph formed by replacing the edge from x with the entire flowgraph of F_2.

Nesting (Example)



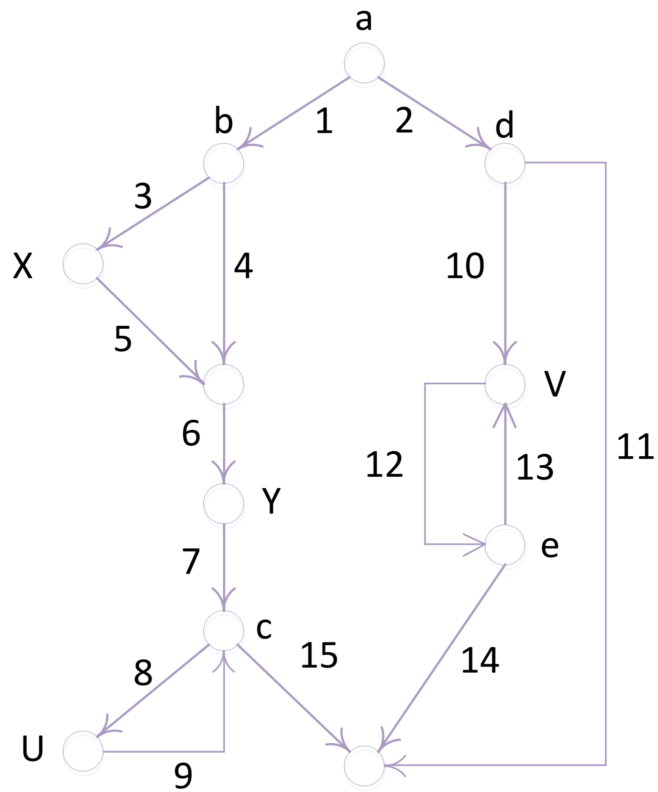
Prime Decomposition Tree

A prime decomposition tree is a tree where nodes represent prime flowgraphs (P_1, D_0, D_1, D_2, D_3, C_n) or sequencing of flowgraphs P_n.

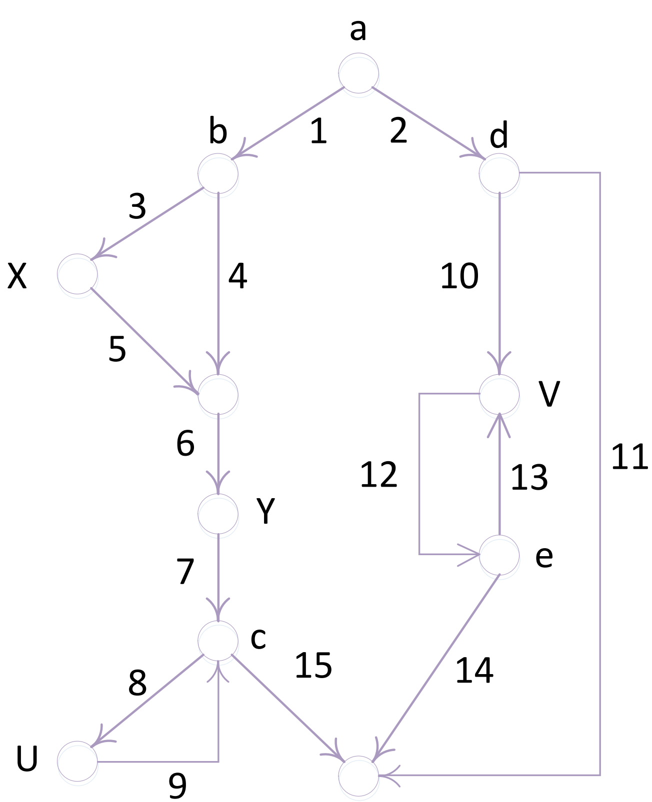
The arcs represent the nesting or sequential composition.

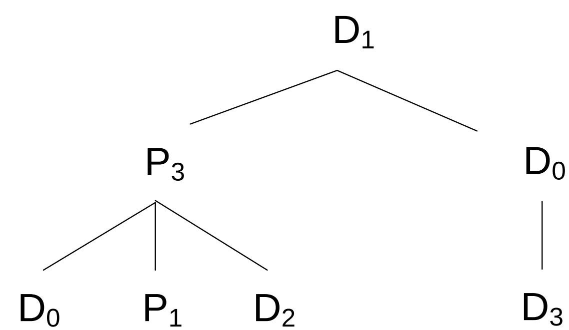
Exercise: Draw the corresponding prime decomposition tree

```
if (a) {
   if (b) {
     ... //statements X
   }
   ... //statements Y
   while (c) {
     ... //statements U
   }
} else {
   if (d) {
     do {
        ... //statements V
     }
   while (e);
   }
}
```



Exercise: Draw the corresponding prime decomposition tree





Prime Decomposition Tree

Essential complexity and structuredness

Essential complexity of a program P with flowgraph G is given by:

$$ev(G) = v(G) - m$$

where,

- $v(G)$ is the cyclomatic complexity of G ($v(g) = e - n + 2$)
- m is the number of sub-flowgraphs in G

Essential complexity of a flowgraph G corresponds to the cyclomatic complexity of the graph obtained by replacing all prime flowgraphs in G by a single node.

P is fully structured iff ev(G) = 1

Code coverage goal

- To test ***efficiently*****,** you must find the largest possible number of defects using the fewest resources
- To test ***effectively*****,** you must use coverage criteria that uncovers as many defects as possible.
- **Test metrics:**
  - Minimum number of test cases **[RE: efficiency]**
  - Test coverage **[RE: effectiveness]**

Minimum number of test cases

- **Basis path coverage:**
  - e – n + 2
  - # decision points + 1 (if graph entails only binary decision points)

- **Statement, branch, simple path, visit-each-loop, all paths coverage:**
  - Need to consider the prime decomposition tree (see next slides)

- **Multiple condition coverage:**
  - 2^n test cases for each decision with n compounds (to be discussed later)

Minimum number of test cases for primes

Test Strategy	P_1	D_0	D_1	D_2	D_3	C_n
Statement testing						
Branch testing						
Simple path testing						
Visit-each-loop-testing						
All path testing						

Minimum number of test cases for primes

Test Strategy	P_1	D_0	D_1	D_2	D_3	C_n
Statement testing	1	1	2	1	1	n
Branch testing	1	2	2	1	1	n
Simple path testing	1	2	2	2	1	n
Visit-each-loop-testing	1	2	2	2	2	n
All path testing	1	2	2	∞	∞	n

Minimum number of test cases for Sequencing

Let $\mu(F_i)$ denote the minimum number of test cases for a given flowgraph F_i, then the minimum number of test cases for a flowgraph composed through sequencing can be computed as follows:

Test Strategy	**$P_n(F_1, F_2, \ldots, F_n)$**
Statement testing Branch testing	$\max\left(\mu(F_1),\ \mu(F_2), \ldots \mu(F_n)\right)$
Simple path testing Visit-each-loop-testing All path testing	$\prod_{i=1}^{n} \mu(F_n)$

Minimum number of test cases for Sequencing

Let $\mu(F_i)$ denote the minimum number of test cases for a given flowgraph F_i, then the minimum number of test cases for a flowgraph composed through nesting can be computed as follows:

Test Strategy	$D_0(F)$	$D_1(F_1, F_2)$	$D_2(F)$	$D_3(F)$	$C_n(F_1, \ldots, F_n)$
Statement testing	$\mu(F)$	$\mu(F_1) + \mu(F_2)$	1	1	$\sum_{i=1}^{n} \mu(F_i)$
Branch testing	$\mu(F)$ +1	$\mu(F_1) + \mu(F_2)$	1	1	$\sum_{i=1}^{n} \mu(F_i)$
Simple path testing	$\mu(F)$ +1	$\mu(F_1) + \mu(F_2)$	$\mu(F)$ +1	$\mu(F)$	$\sum_{i=1}^{n} \mu(F_i)$
Visit-each-loop-testing	$\mu(F)$ +1	$\mu(F_1) + \mu(F_2)$	$\mu(F)$ +1	$\mu(F) + \mu(F)^2$	$\sum_{i=1}^{n} \mu(F_i)$
All path testing	$\mu(F)$ +1	$\mu(F_1) + \mu(F_2)$	∞	∞	$\sum_{i=1}^{n} \mu(F_i)$

Multiple Condition Coverage Testing

- Assuming that predicate P1 is a compound predicate (i.e. A or B) then Multiple Condition Coverage Testing requires that each possible combination of truth values be tested for each decision.
- Example: "if (A or B)" requires 4 test cases:

  A = True, B = True

  A = True, B = False

  A = False, B = True

  A = False, B = False
- The problem: For n compounds, 2^n test cases are needed, and this grows exponentially with n
- Less test cases are needed if conditions are couples
  - Strong coupling: changing one condition <u>will</u> change the other
  - Weak coupling: changing one condition <u>may</u> change the other

Code coverage requirements

- Low code coverage indicates inadequate testing, but 100% coverage is impossible in practice **and generally** not cost effective
- Although 100% code coverage may appear like a best possible effort, even 100% code coverage is estimated to only expose about half the faults in a system.

- **Code coverage of 70-80% is a reasonable goal for system test of most projects with most coverage metrics.**
- **Empirical studies of real projects found that increasing code coverage above 70-80% is time consuming and therefore leads to a relatively slow bug detection rate**
- **Minimum code coverage for unit testing can be 10-20% higher than for system testing**

Minimum Acceptable Code Coverage Standards

- The aviation standard **DO-178B** requires **statement coverage, branch coverage and (modified) multiple decision coverage** for level A safety critical systems.
- The standard **IEC 61508:2010** "Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems" recommends **coverage of several metrics**, but the strenuousness of the recommendation relates to the criticality.
- The **IEEE Standard for Software Unit Testing** section 3.1.2 specifies **statement coverage** as a completeness requirement. Section A9 recommends **branch coverage for code that is critical** or has inadequate requirements specification.

White Box Testing Advantages

- Structural testing methods are very amenable to:
  - Rigorous definitions
    - control flow, objectives, coverage criteria, relation to programming language semantics
  - Mathematical analysis
    - Graphs, path analysis
  - Precise measurement
    - Metrics, coverage analysis

Problems with White-Box Testing

- Infeasible paths:
  program paths that cannot be executed for any input
- No white-box strategy on its own can guarantee adequate software testing
- Knowing the set of paths that satisfies a particular strategy doesn't tell you how to create test cases to match the paths.