Revision: data flow based coverage — solutions

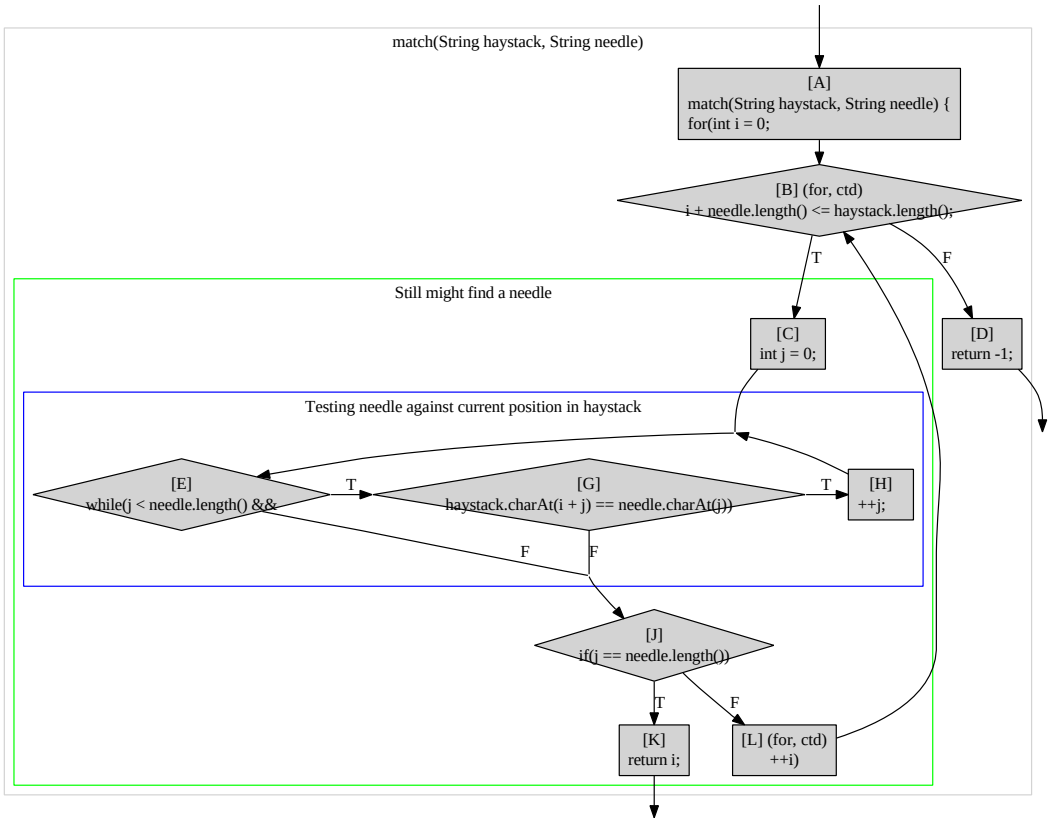
(code taken from 2008 exam, question 2)

Conrad Hughes

March 17, 2009

It's very important to know your coverage criteria definitions when attempting this kind of question. It's also worth doing one or two run-throughs with some simple inputs just to familiarise yourself with the code before you start to design your tests.

1. Control flow graph:



Note how the `for` loop is broken up over three nodes: **A**, **B** and **L**.

(In this case there are no nodes labelled **F** or **I** in order to avoid confusion with **F**alse branches, digit 1, letter l, etc.)

2. Annotations — note that the simplest thing is to write these on the graph itself, but *typesetting* that would be a lot of work.

Node	Defs/uses		
A	defs={haystack,needle,i}	G	p-uses={haystack,i,j,needle}
B	p-uses={i,needle,haystack}	H	defs={j}; c-uses={j}
C	defs={j}	J	p-uses={j,needle}
D	[none]	K	c-uses={i}
E	p-uses={j,needle}	L	defs={i}; c-uses={i}

3. Table: this is the same information as above, but now keyed by variable. This presentation makes it easier to identify all of the def-use pairs.

Variable	defs	c-uses	p-uses
haystack	A		BC,BD,GH,GJ
needle	A		BC,BD,EG,EJ,GH,GJ,JK,JL
i	A,L	K,L	BC,BD,GH,GJ
j	C,H	H	EG,EJ,GH,GJ,JK,JL

4. Def-use pairs. Note here that variables i and j have more than one definition, so the issue of *def-clear paths* comes up. Consequently when designing paths to cover a du-pair (X,Y), we must be careful that the path between **X** and **Y** is def-clear for any relevant variables.

   all-defs: (A,B/G); (A,B/E/G/J); (A,K/L/B/G); (L,K/L/B/G); (C,H/E/G/J); (H,H/E/G/J)

   (X,Y/Z) means at least one of (X,Y) or (X,Z); note that I don't specify which branch ((A,BC) vs (A,BD)) because it doesn't matter if all we want is *at least one* — once we've reached **B** from **A**, any branch will satisfy all-defs, be it **BC** or **BD**.

   all-c-uses: (A,K); (A,L); (L,K); (L,L); (C,H); (H,H)

   all-p-uses: (A,BC); (A,BD); (A,GH); (A,GJ); (A,EG); (A,EJ); (A,JK); (A,JL); (L,BC); (L,BD); (L,GH); (L,GJ); (C,EG); (C,EJ); (C,GH); (C,GJ); (C,JK); (C,JL); (H,EG); (H,EJ); (H,GH); (H,GJ); (H,JK); (H,JL)

   all-uses: union of all-c-uses and all-p-uses.

5. Any the same? No, but notice that if you satisfy all-p-uses it's clear from the def-use table that you'll automatically satisfy all-defs. This isn't always true. Be aware of the graphs showing subsumption relationships in the slides.

6. Does an all-defs-adequate suite need to satisfy statement coverage? No: you could satisfy all-defs without ever reaching node **D** for example.

7. Test suites:

   all-defs: `match("xy", "xz")` $\rightarrow -1$ would cause the path **ABCEGHEGJLBD** to be executed. This satisfies all-defs.

all-c-uses: Two tests are needed:

- `match("x", "x")` → 0 would cause the path **ABCEGHEJK** to be executed. This covers pairs (A,K) and (C,H) from our path set, leaving (A,L), (L,K), (L,L) and (H,H) still to cover.
- To get (L,L) and (H,H) we need to see more than one iteration of both the i-loop and the j-loop. Adding `match("yyxx", "xx")` → 2 to the suite will execute the path **ABCEGJLBCEJLBCEGHEGHEJK**, which covers all of these remaining pairs.

Note that the first test is necessary since the second test doesn't contain a def-clear path for i from **A** to **K** since the second test's path contains node **L** (redefining i) between **A** and **K**.

all-p-uses: A much longer list of def-use pairs to cover...

- `match("", "x")` → −1 will execute path **ABD**, covering (A,BD).
- `match("x", "")` → 0 will execute path **ABCEJK**, covering (A,BC), (A,EJ), (A,JK), (C,EJ), and (C,JK).
- `match("y", "x")` → −1 will execute path **ABCEGJLBD**, covering (A,GJ), (A,EG), (A,JL), (L,BD), (C,EG), (C,GJ) and (C,JL).
- `match("yyyx", "yx")` → 2 will execute path **ABCEGHEGJLBCEGHEGJLBCEGHEGHEJK**, covering (A,GH), (L,BC), (L,GH), (L,GJ), (C,GH), (H,EG), (H,EJ), (H,GH), (H,GJ), (H,JK) and (H,JL).

The technique here is to see what you haven't covered, then add a test targetting some of the uncovered d-u pairs (or even a random test if you're stuck), then see what you still haven't covered, and so on.

all-uses: The above all-p-uses test actually covers all-c-uses too, so it's also all-uses-adequate.

8. To achieve all-du-paths: remember that the definition specifies that all paths between def-use pairs must be executed, modulo loops. So you'd need to ensure for example that for the (C,JK) pair with respect to variable j, that paths **CEJK**, **CEGJK**, **CEGHEJK**, and **CEGHEGJK** are covered. This is clearly a lot of work, and not something I'm about to ask you to do...