CS510 Assignment Solutions#1

February 20, 2013

1 Control Graph, Dominator and Post-Domintor (25p)

(a) Construct the control flow graph for the below code snippet. Please also list the dominators and immediate post-dominators for 3, 5, 6, 7, 8, and 17.

```
1.    n=input();
2.    s=0;
3.    if (n>10)
4.        return;
5.    while (n>0) {
6.        if (s>10) {
7.            while (n>0) {
8.               s=s-n;
9.               n=n-1;
10.           }
11.           break;
12.       }
13.       s=s+2;
14.       n=n-1;
15.    }
16.    if (s>0 &&
17.        s\%2==0) {
18.            s=s+1;
19.    }
```

Solution:

DOM(3)={Start, 1, 2, 3}	IPDOM(3)={End}
DOM(5)={Start, 1, 2, 3, 5}	IPDOM (5)={16}
DOM(6)={Start, 1, 2, 3, 5, 6}	IPDOM (6)={16}
DOM(7)={Start, 1, 2, 3, 5, 6, 7}	IPDOM (7)={11}
DOM(8)={Start, 1, 2, 3, 5, 6, 8}	IPDOM (8)={9}
DOM(17)={Start, 1, 2, 3, 5, 16, 17}	IPDOM (17)={End}

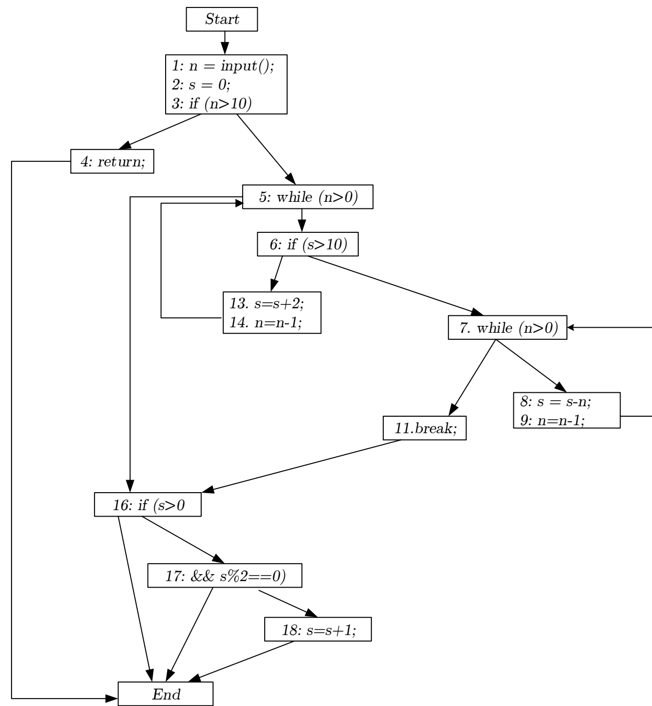


Figure 1: Control Flow Graph

(b) Prove that a statement has only one immediate post-dominator (8p).

Proof: Suppose a statement s has two IPDOM a and b. That means we have two paths: $s \rightarrow a \rightarrow ... \rightarrow b \rightarrow ... \rightarrow Exit$ and $s \rightarrow b \rightarrow ... \rightarrow a \rightarrow ... \rightarrow Exit$. Note that both a in b have to appear in both paths as they are post-dominators of s. The existence of the two paths implies we must also have a path $a \rightarrow ... \rightarrow Exit$ without passing b. Connecting this path with the edge $s \rightarrow a$ in path one get a path from a to $Exit$ without passing b.

Many students simply assumed an IPDOM is the closest PDOM along *any* paths to *Exit*. As a result, the property is trivially true. A proper understanding of IPDOM is that it is the closest PDOM along A path, but its property decides it is also the clostest PDOM for all other paths.

2 Program Dependence Graph (20p)

Build the program dependence graph for the code in problem 1. If the graph

is too crowded, you can separate it to two subgraphs: data dependence graph and control dependence graph.

Solution:

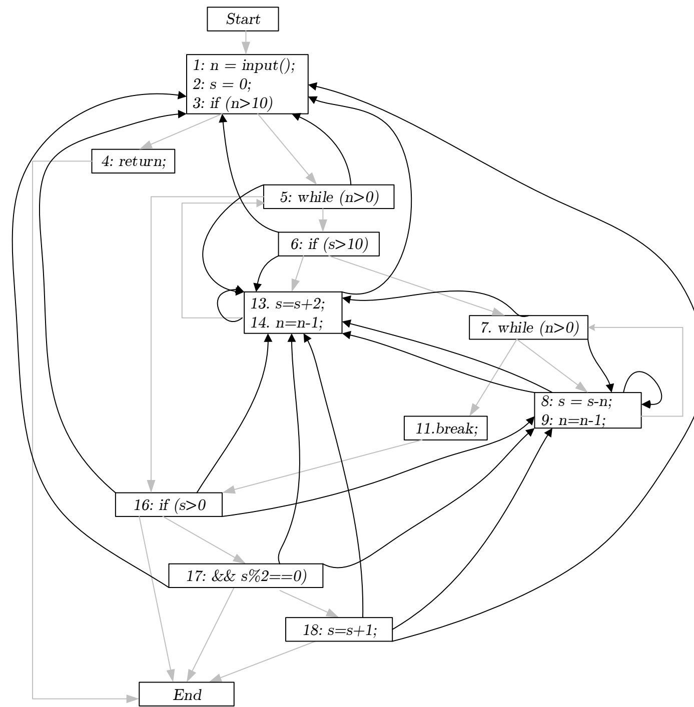


Figure 2: Data dependence graph

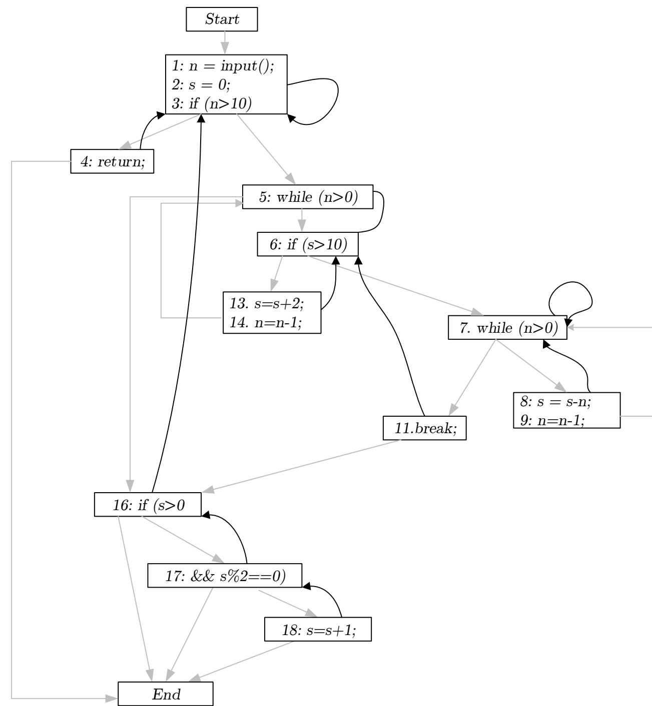


Figure 3: Control dependence graph

Program dependence graph = Data dependence graph + control dependnece graph.

3 Trace Compression (10p)

Let a plain text string be
a b a b c d c b a b c b.
Assume the initial lookup table is

Context	Prediction
ab	a
bc	a
cd	b

.

Use FCM-2 to compress the string. The final compressed string and the final lookup table are required. Intermediate steps are not required but encouraged.

Solution:

Output: 1 a 1 b 0 1 b 1 c 1 d 1 c 1 b 1 a 0 0 1 b

Context	Prediction
ab	c
bc	b
cd	c
ba	b
dc	b
cb	a

Note: Few students forgot to put bit 0 before a mis-predicted value. The bit is needed because with bit representation, a letter might start with a bit 0 or 1. In order to distinguish such bits from the bits that indicate prediction results, prediction result bit 0/1 need to put into the stream all the time. The rule becomes, if the algorithm sees prediction bit 1, it looks at the next bit; if the algorithm sees prediction bit 0, it extracts the next 8 bits as the value, and then looks at the next bit.

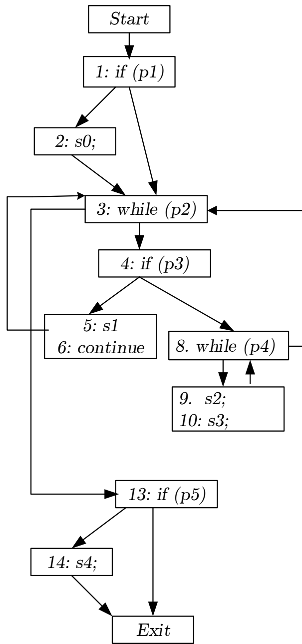
4 Path Profiling (25p)

```
1.    if (p1)
2.       s0;
3.    while (p2) {
4.       if (p3) {
5.          s1;
6.          continue;
7.       }
8.       while (p4) {
9.          s2;
10.         s3;
11.      }
12.   }
13.   if (p5)
14.       s4;
```

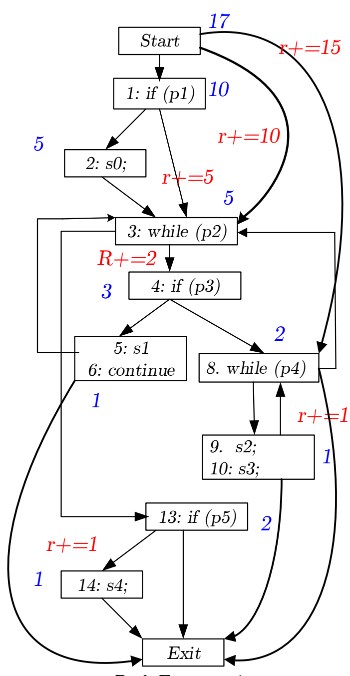
(a) Construct the path enumeration graph for the above program. Show the path encoding.

(b) Show the final instrumented program, executing which collects the path profile.

Solution:



Control Flow Graph



Path Enumeration Graph

```
      r=0;
1.    if (p1) {
2.        s0;
      } else {
        r+=5;
3.    while (p2) {
          r+=2;
4.        If (p3) {
5.            s1;
              counter[r]++;
              r=10;
6.            continue;
          }
8.        while (p4) {
              r+=1;
9.            s2;
10.           s3;
              counter[r]++;
              r=15;
      }
13.   if (p5) {
          r+=1;
14.       s4;
      }
      counter[r]++;
```

Intrumentation

5 Predicate Tracing (20p)

Predicate tracing is a control flow tracing technique that records the branch outcomes of predicates. For example,

```
1.    if (...)
2.       if (...)
3.         s0;
4.    if (...)
5.       s1
6.   s2;
```

The trace 1 2 3 4 6 for the above program can be represented as T T F. Three bits are needed.

(a) Please list the challenges for making the above idea work on real world programs. You can assume C or Java languages.

(b) Sketch solutions to such challenges.

Using examples is encouraged.

Solution:

Assume C language.

(1) One challenge is that when a function has more than one callers. The trace does not directly tell you where to return. The solution includes recording the program point that is returned to. Or the decoding algorithm needs to maintain a call stack.

(2) When a method is called through a function pointer, the target needs to be traced.

(3) For switch-case statments, the algorithm needs to trace which case is taken.

(4) For long jumps and set jumps, the correspondence needs to be traced.

Getting 3 out of 4 should be sufficient.

Assume Java language.

(1) Due to class inheritance, the dynmic method that is being called needs to be traced.

(2) The places that an exception is thrown and handled need to be traced.

(3) The above (1).

(4) The above (3).

Getting 3 out of 4 should be sufficient.