Finite Models (Structural)

SWE 261P

Structural Models vs. Functional Models

Prescriptive

Functional models represent the specification, in other words, the *intended* behavior

Descriptive

Structural models represent the code, as it is (even when it contains bugs, or is incomplete)

Terminology

- ***Intraprocedural***: Within a single procedure, function, or method (sometimes called intramethod)
- ***Interprocedural***: Across procedure boundaries, procedure call, shared globals, etc.
- ***Intraclass***: Within a single class
- ***Interclass***: Across class boundaries
- ***Intrasubsystem***: Involving one subsystem, which may consist of a number of methods, classes, etc.

(Intraprocedural) Control Flow Graph

- nodes = regions of source code (basic blocks)
  - Basic block = program region with a that is atomically executable
  - Often statements are grouped in single regions to get a compact model
  - Sometimes single statements are broken into more than one node to model control flow within the statement
- directed edges = possibility that program execution proceeds from the end of one region directly to the beginning of another

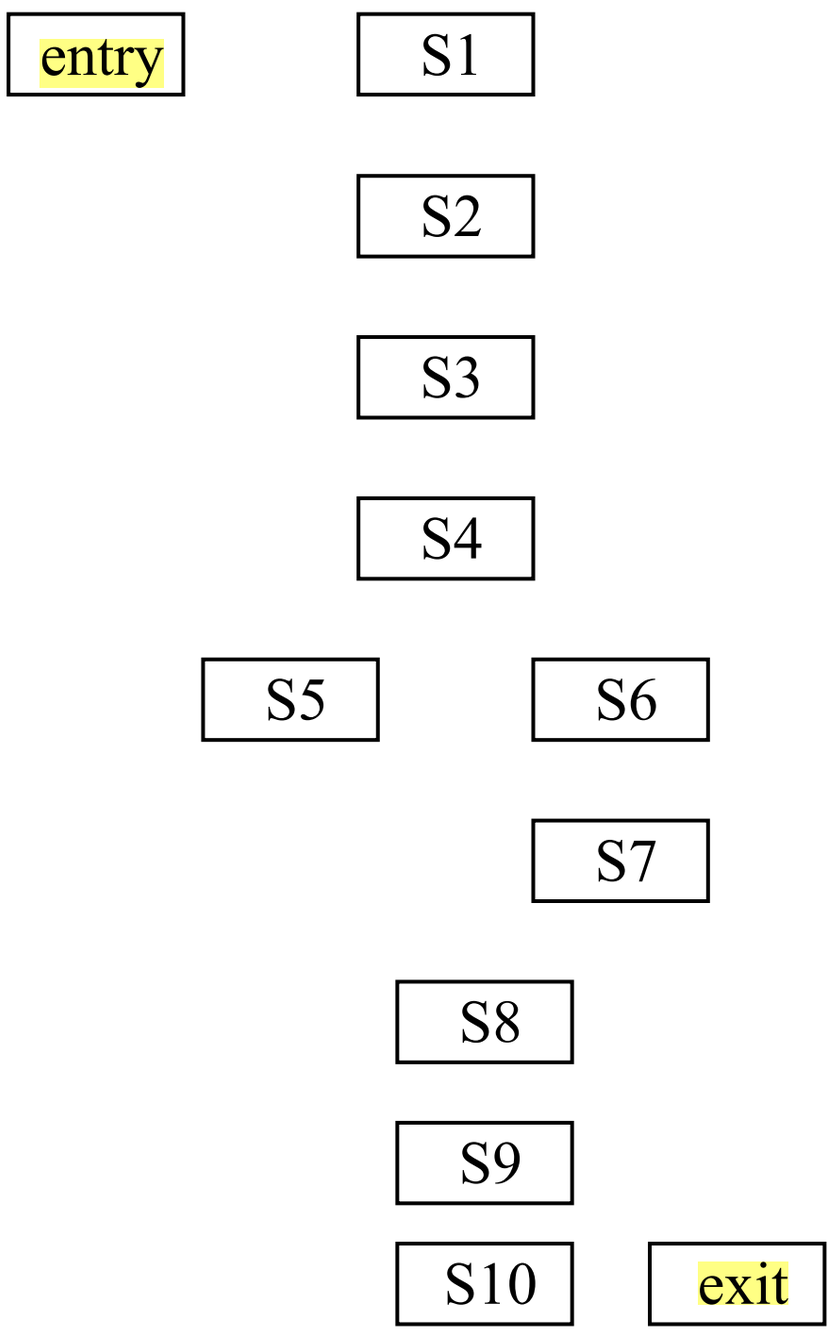
CF1. Computing Control Flow (an example)

```
Procedure AVG
S1    count = 0
S2    fread(fptr, n)
S3    while (not EOF) do
S4        if (n < 0)
S5            return (error)
          else
S6            nums[count] = n
S7            count ++
          endif
S8        fread(fptr, n)
        endwhile
S9    avg = mean(nums,count)
S10   return(avg)
```

CF1. Computing Control Flow (an example)

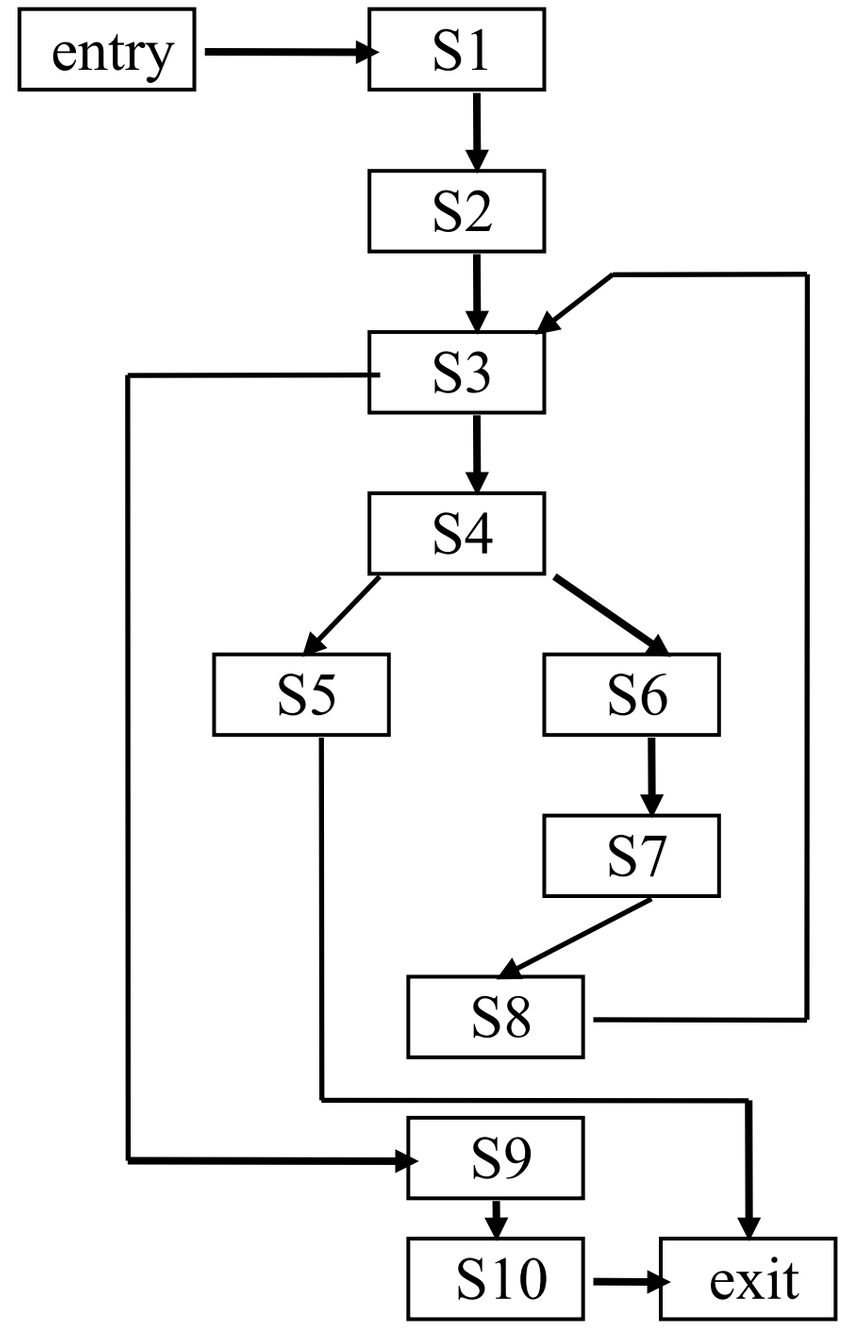
one entry,one exit

```
Procedure AVG
S1    count = 0
S2    fread(fptr, n)
S3    while (not EOF) do
S4        if (n < 0)
S5            return (error)
          else
S6            nums[count] = n
S7            count ++
          endif
S8        fread(fptr, n)
        endwhile
S9    avg = mean(nums,count)
S10   return(avg)
```



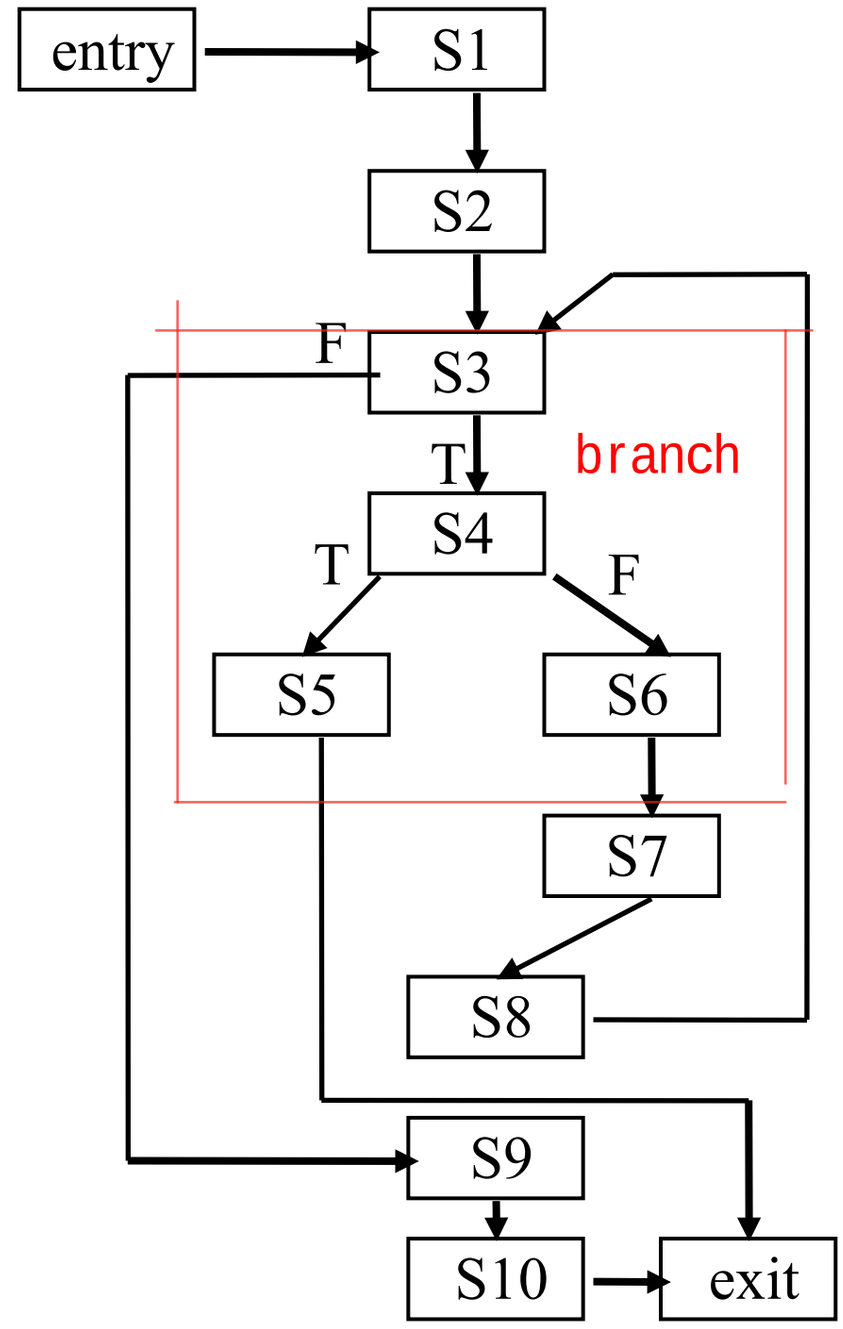
CF1. Computing Control Flow (an example)

```
Procedure AVG
S1    count = 0
S2    fread(fptr, n)
S3    while (not EOF) do
S4        if (n < 0)
S5            return (error)
          else
S6            nums[count] = n
S7            count ++
          endif
S8        fread(fptr, n)
        endwhile
S9    avg = mean(nums,count)
S10   return(avg)
```



CF1. Computing Control Flow (an example)

```
Procedure AVG
S1    count = 0
S2    fread(fptr, n)
S3    while (not EOF) do
S4        if (n < 0)
S5            return (error)
          else
S6            nums[count] = n
S7            count ++
          endif
S8        fread(fptr, n)
        endwhile
S9    avg = mean(nums,count)
S10   return(avg)
```

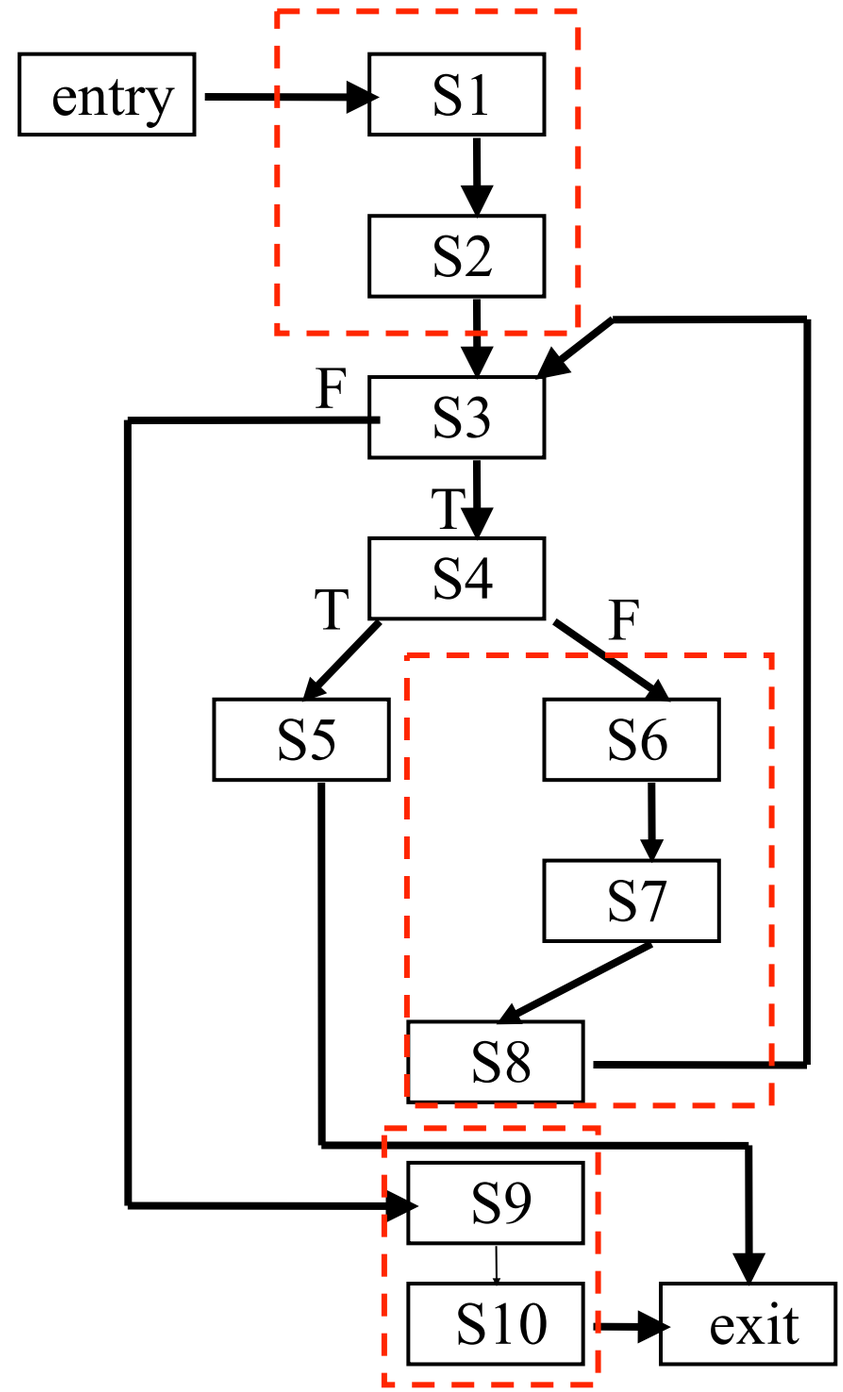


Basic Blocks

- A ***basic block*** is a sequence of consecutive statements in which flow of control enters at the beginning and leaves at the end without halt or possibility of branch, except at the end
- Single entry node, single exit node — may be multiple paths (edges) into entry, may be multiple paths (edges) out of exit — but each is at only one point
- A basic block may or may not be maximal, but in most cases, we mean "maximal basic blocks" when we say "basic blocks". In this course, we will always be referring to maximal basic blocks.

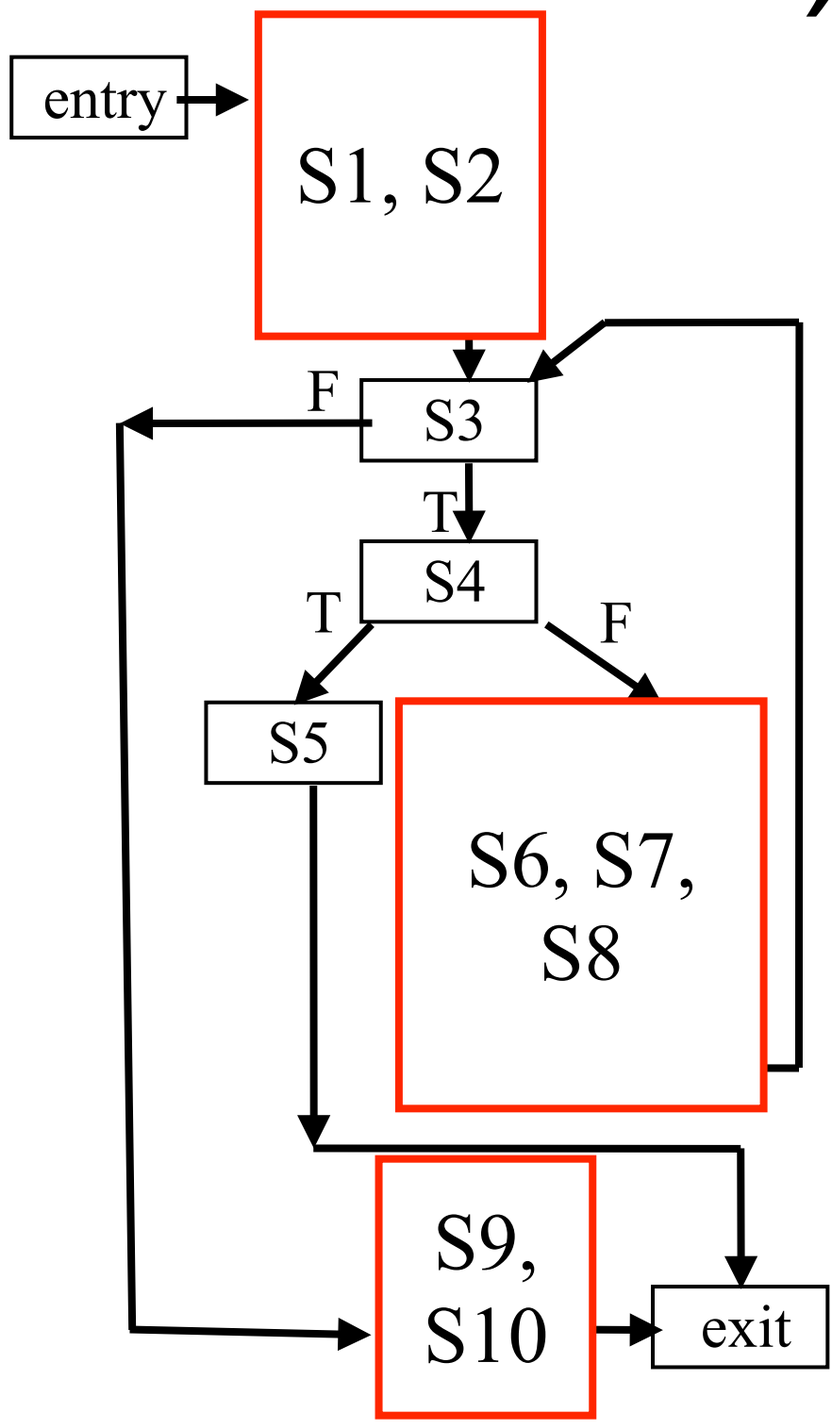
CF1. Computing Control Flow (cont'd) (w/maximal basic blocks)

```
Procedure AVG
S1    count = 0
S2    fread(fptr, n)
S3    while (not EOF) do
S4        if (n < 0)
S5            return (error)
          else
S6            nums[count] = n
S7            count ++
          endif
S8        fread(fptr, n)
        endwhile
S9    avg = mean(nums,count)
S10   return(avg)
```



Control Flow Graph Example (w/ maximal basic blocks)

```
Procedure AVG
S1    count = 0
S2    fread(fptr, n)
S3    while (not EOF) do
S4        if (n < 0)
S5            return (error)
          else
S6            nums[count] = n
S7            count ++
          endif
S8        fread(fptr, n)
        endwhile
S9    avg = mean(nums,count)
S10   return(avg)
```

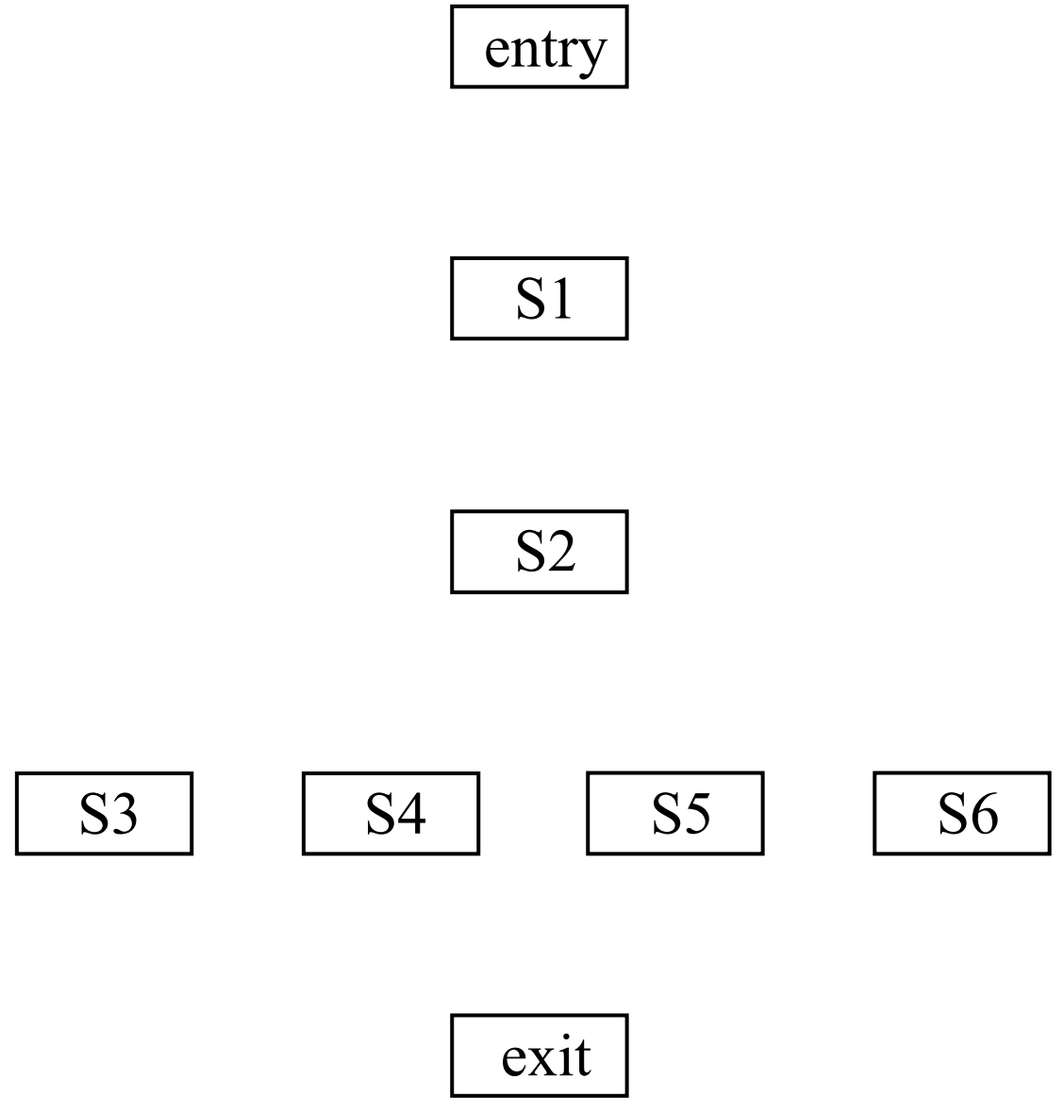


CF1. Computing Control Flow (cont'd) (another example)

```
Procedure Trivial
S1    read (n)
S2     switch (n)
         case 1:
S3         write (“one”)
           break
         case 2:
S4         write (“two”)
         case 3:
S5         write (“three”)
           break
         default
S6         write (“Other”)
       endswitch
end Trivial
```

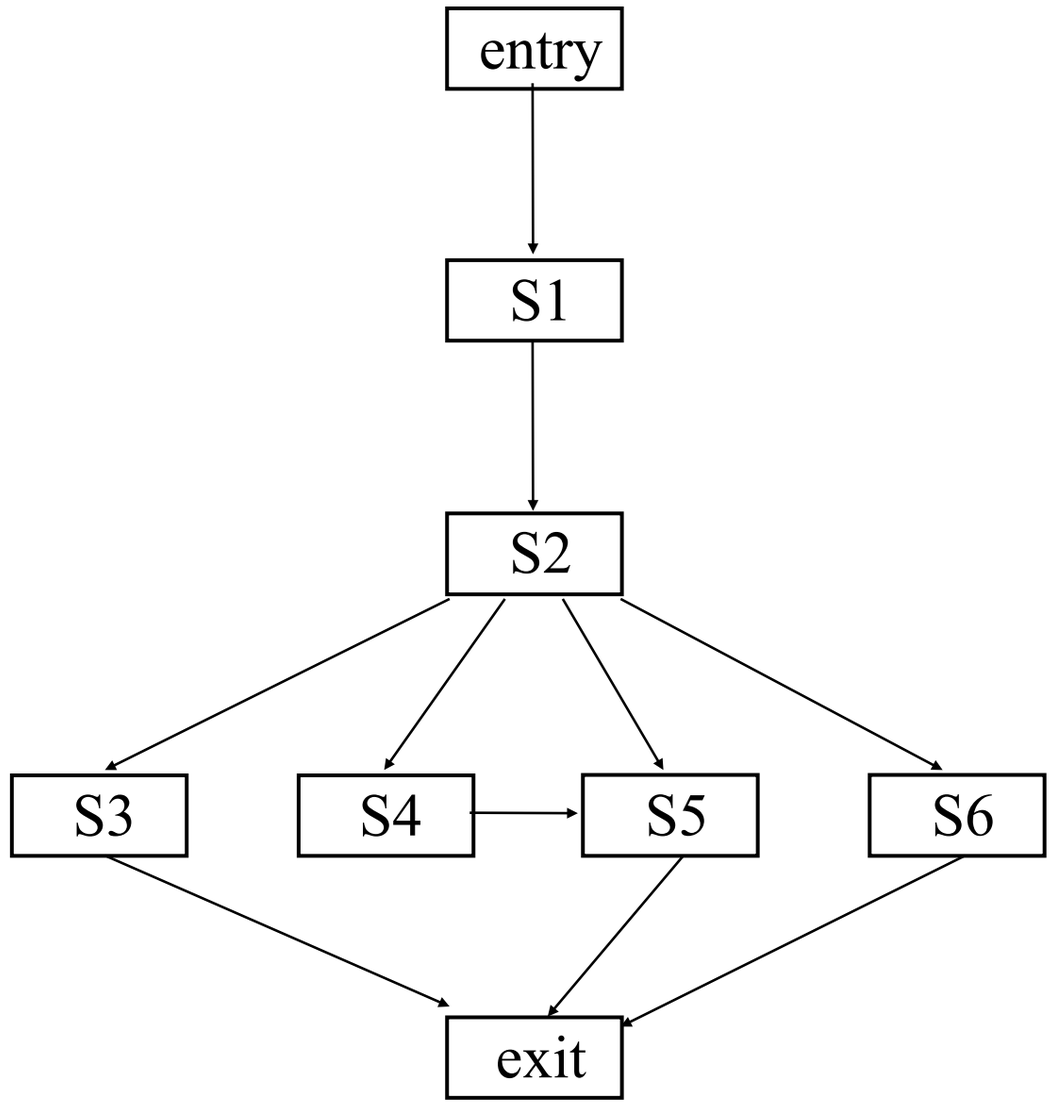
CF1. Computing Control Flow (cont'd) (another example)

```
Procedure Trivial
S1    read (n)
S2     switch (n)
         case 1:
S3         write ("one")
           break
         case 2:
S4         write ("two")
         case 3:
S5         write ("three")
           break
         default
S6         write ("Other")
       endswitch
end Trivial
```



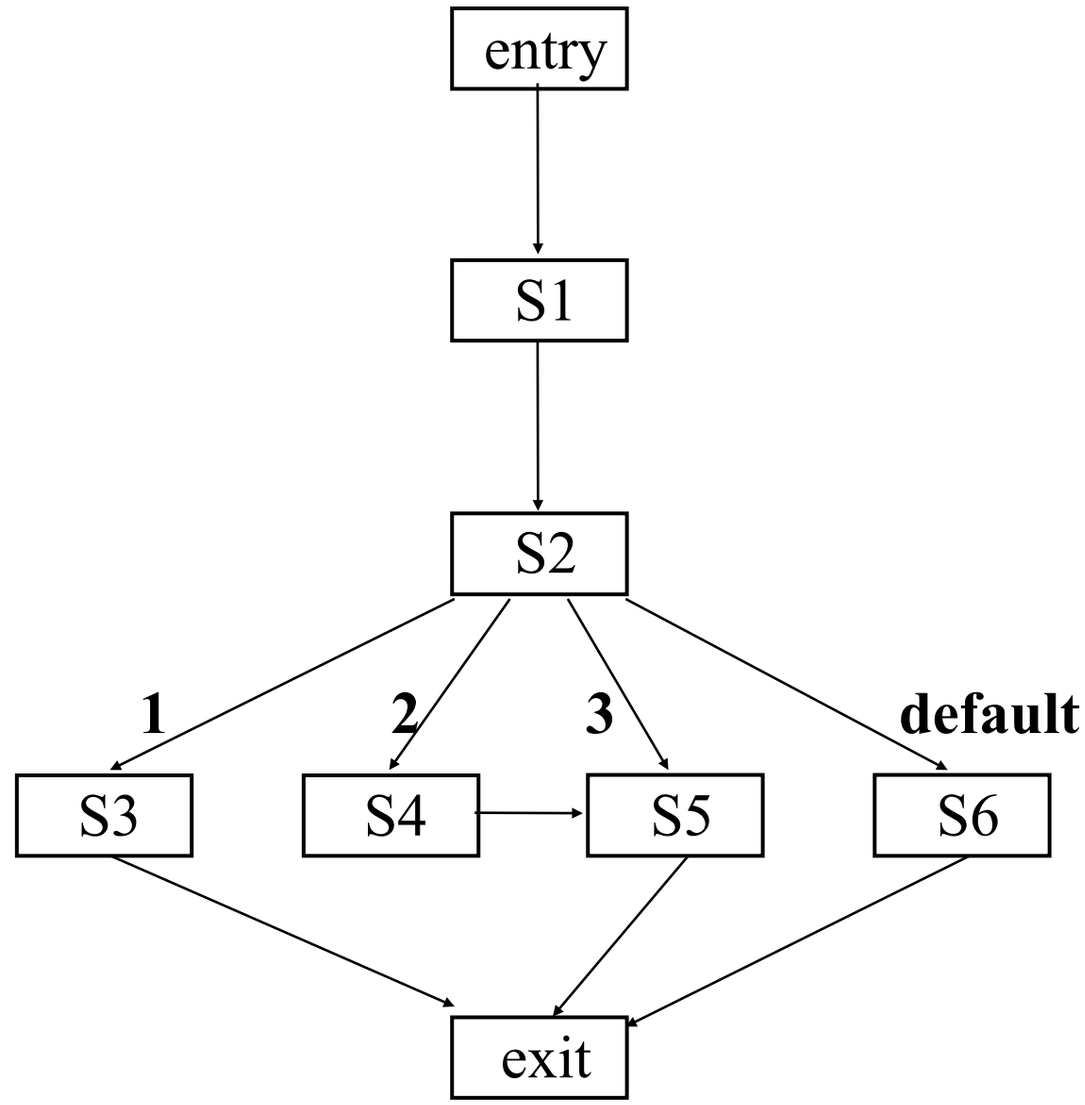
CF1. Computing Control Flow (cont'd) (another example)

```
Procedure Trivial
S1    read (n)
S2     switch (n)
         case 1:
S3         write (“one”)
           break
         case 2:
S4         write (“two”)
         case 3:
S5         write (“three”)
           break
         default
S6         write (“Other”)
       endswitch
end Trivial
```



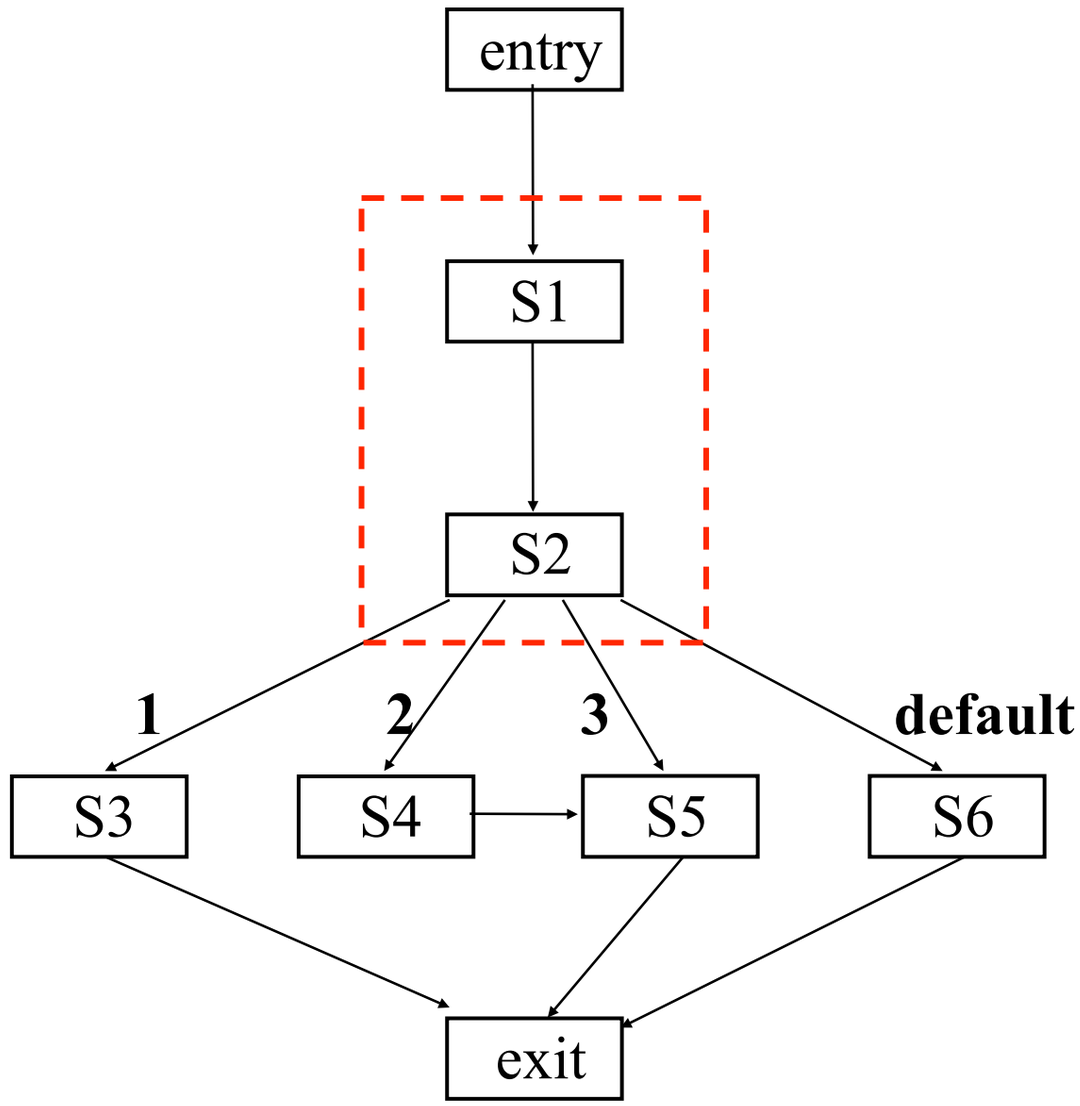
CF1. Computing Control Flow (cont'd) (another example)

```
Procedure Trivial
S1    read (n)
S2     switch (n)
         case 1:
S3         write (“one”)
           break
         case 2:
S4         write (“two”)
         case 3:
S5         write (“three”)
           break
         default
S6         write (“Other”)
       endswitch
end Trivial
```



CF1. Computing Control Flow (cont'd) (w/maximal basic blocks)

```
Procedure Trivial
S1    read (n)
S2     switch (n)
         case 1:
S3         write (“one”)
           break
         case 2:
S4         write (“two”)
         case 3:
S5         write (“three”)
           break
         default
S6         write (“Other”)
       endswitch
end Trivial
```



Interesting Cases of Control Flow

- `for` loops have 3 sub-statements: the initializer, the predicate, and the increment

```
for (int i=0; i < 10; i++)
```

Should be modeled with 3 nodes

- Ternary operators contain control flow within a single line that includes a predicate and a branch

```
i = (j>=0) ? 100 : -100;
```

Should be modeled with proper nodes, edges, and control flow

- `continue` and `break` instructions jump directly to the predicate or exits the loop, respectively

(caveat: in a `for` loop, `continue` jumps to the increment immediately before the predicate)

Should be modeled with proper edges and control flow

Control Flow Graph Exercises

Why?

Why?

- ~~Capture specs~~ (unlike finite functional models)
- Inform what test cases can and should be written
- Inform a “criterion” for what is “enough”
- Allow for analyses and automated assistance (those details are forthcoming...)