CS 6340 – Spring 2013 – Assignment 5
Assigned: January 30, 2013
Due: February 6, 2013

Name Sam Britt, Shriram Swaminathan,

Name and Sivaramachandran Ganesan

At the beginning of class on the due date, submit your neatly presented solution with this page stapled to the front (60 points).

NOTE: All work on this problem set is to be done with your partner and without solutions from other past or current students. Any violations will be dealt with according to the Georgia Tech Academic Honor Code and according to the College of Computing process for resolving academic honor code violations. All work must be done using some document creation tool. In addition, graphs must be drawn with a graph-drawing tool—no hand-drawn graphs will be accepted.

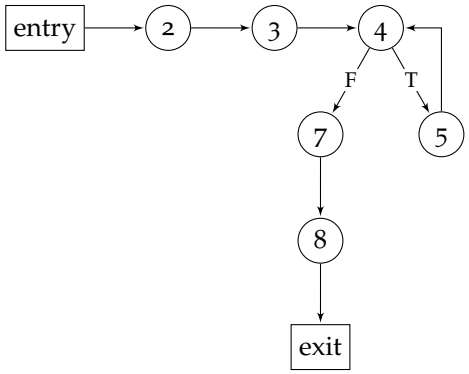
Given the following C program,

```
Program M
1.   begin M
2.     read i, j
3.     sum = 0
4.     while i <= 10 do
5.        call B
6.     endwhile
7.     print (i)
8.     call C
9.   end M

Procedure B
10.   begin B
11.     if sum > 10 then
12.        print (error)
13.     endif
14.     call C
15.     i = i + 1
16.  end B

Procedure C
17.  begin C
18.     if j >= 0 then
19.        sum = sum + j
20.        read j
21.     endif
22.  end C
```

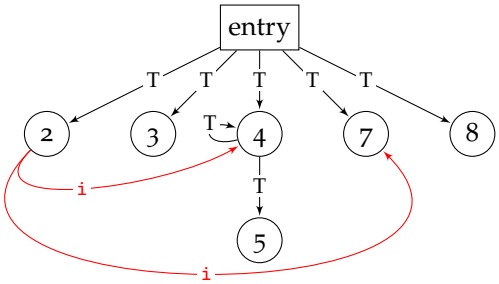
Sam Britt
Shriram Swaminathan
Sivaramachandran Ganesan

CS 6340
Assignment 5
Feb. 6, 2013

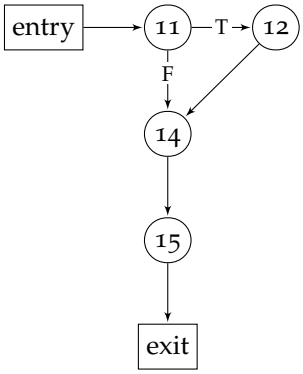
1. CFGs and PDGs for each of the three programs are shown below. Edges corresponding to data dependencies on the PDGs are shown in red (⟶), and are labeled by the variable they represent. Formal- and actual-in and out definition nodes are not included, so there are very few data dependencies.



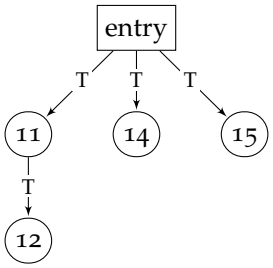
The CFG for program *M*.



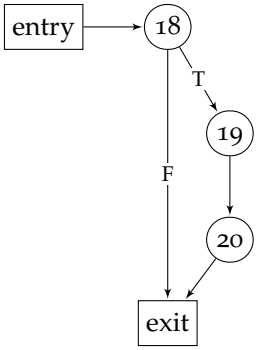
The PDG for program *M*.



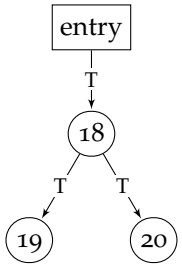
The CFG for procedure *B*.



The PDG for procedure *B*.

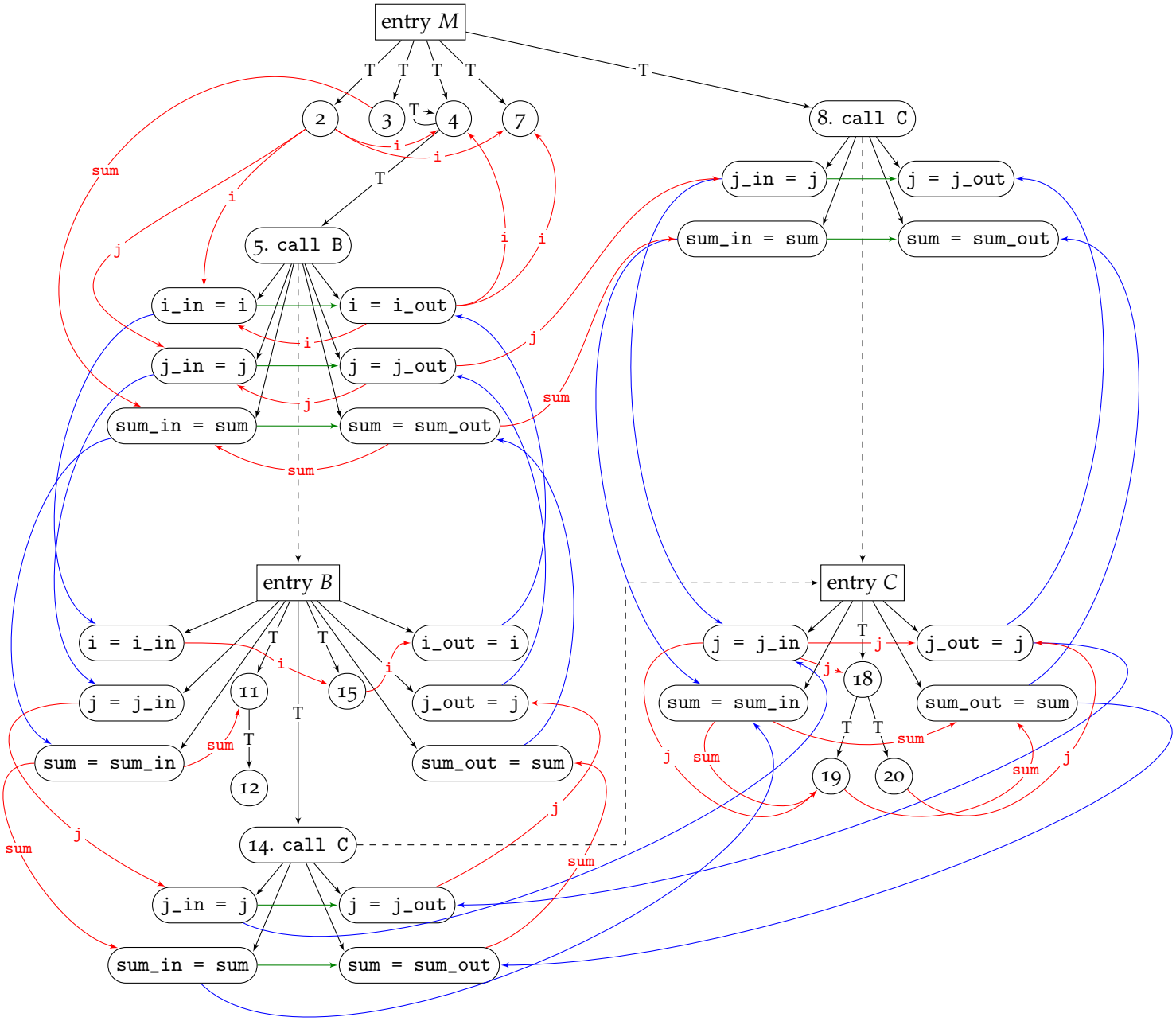


The CFG for procedure *C*.



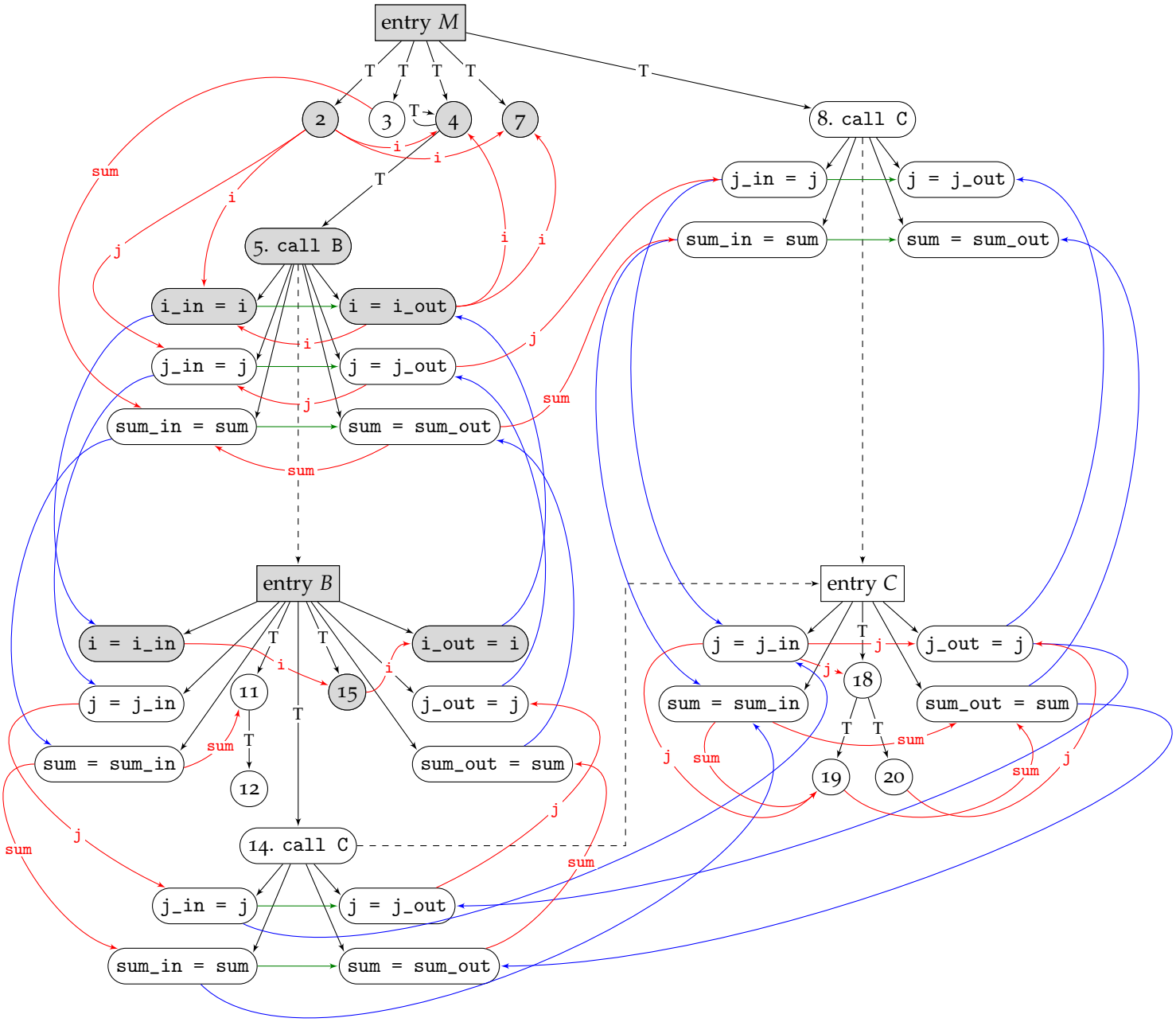
The PDG for procedure *C*.

2. In the following SDG, control dependencies are in solid black (⟶), call edges are in dashed black (⇢), data dependencies are in red (⟶), parameter in and out edges are in blue (⟶), and transitive flow dependence edges are in green (⟶).

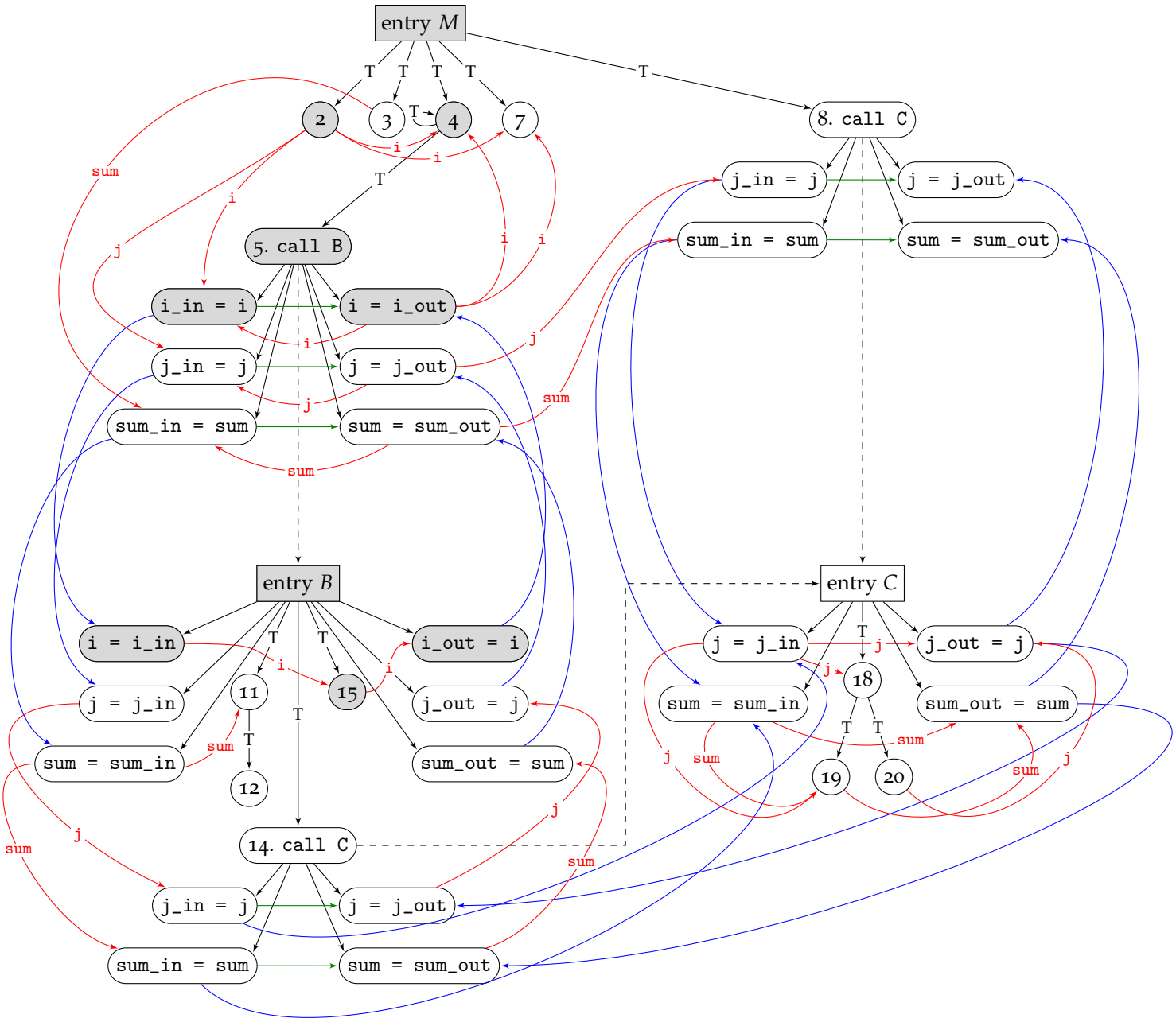


3. In the following, nodes included in the slice are shaded. Each slice was constructed using the two-phase method on the SDG as described in lecture. Although the slices were determined via the algorithm, intuitive justificiations for the inclusion of select nodes are added where appropriate.

   (a) The slice according to the criterion $\langle 7, \texttt{i} \rangle$. It is interesting that line 4 is included in the slice. While it does not directly affect the value of `i`, its execution does control whether the call to *B* on line 5 is executed, and procudre *B* does define `i`. Therefore, line 4 must be included.



(b) The slice according to the criterion $\langle 15, \texttt{i} \rangle$. It is perhaps not at first intuitive why the formal out definition of `i_out` in procedure B and the actual out definition of `i` in in the call to B on line 5 of program M are included in the slice. However, because B is called on line 5 inside a loop, the formal and actual out definitions of `i` in B and M eventually determine the evaluation of the loop condition in line 4 in M, which controls the further evaluation of B (and thus line 15) inside the loop. Therefore, the inclusion of the formal and actual out definitions is justified.



(c) The slice according to the criterion $\langle 18, \texttt{j} \rangle$. Again, it is perhaps not clear why the formal out defininition of `j_out` in procedure *C*, or indeed line 20, is included in the slice. But, since *C* is perhaps called many times (because *B* is called inside a loop on line 5, and *B* calls *C* on line 14), then the (potential) re-definition of `j` on line 20 and as a formal out parameter will affect the use of `j` on line 18 the next time *C* is called. Similarly, statements dealing with the definition of `i`, for example line 15, are included, because they control when *B* (and therefore *C*) gets called on line 5. However, the actual out definition of `j` on the call to *C* in line 8 of program *M* is *not* included in the slice, because by the time the call on line 8 returns, there is no way for *C* to be re-entered. Therefore, the use of `j` line 18 is not affected by the actual out definition of `j` on line 8.