

CMPSCI 521/621
Homework #4
November 20, 2005
Due: December 6, 2005

- 1) Finite State Machines (FSMs) to Petri Nets (PNs):
- a) [20pts for each solution] Not to be outdone by Amherst College, UMass has decided to place a lighted walkway for pedestrians north of the CS building and crossing Governors Drive. The goal is to warn oncoming cars of pedestrians in the crosswalk by flashing lights; with lights initiated (as they are near Amherst College) when a pedestrian pushes a walk button. To further the campus' drive for pedestrian safety, the system will photograph the license plates of any automobile entering a crosswalk when the lights are flashing. You are asked to specify a control system for this crosswalk by developing either [520] or both [620 or 521 extra credit] a FSM or/and a Petri net specification. In addition to the assumptions that follow below, state any other assumptions you make **clearly**, e.g. how are you going to handle time (clock, guards, etc.)

Assume:

- the crosswalk lights don't flash unless a pedestrian pushes "walk" button.
 - the crosswalk lights begin to flash 5 seconds following initiation
 - the crosswalk lights will flash for 30 seconds and any additional push of the walk button will be ignored **unless** it is within 5 seconds of the end of flashing (where it will reset to provide another 30 seconds)
 - cameras have been installed in the east and west directions of Governors Drive and will photograph the license plate of a car when signaled, i.e., you don't have to specify the camera system or how the images are stored or relayed to the campus police – just indicate a state in which the cameras are signaled.
 - cars entering the cross walk are sensed automatically and a signal is provided to your system
 - you must handle two sensors and two cameras
- b) [10 pts] Sketch a method for converting FSMs to PN's.
- c) [10 pts] Give an example illustrating why it is not always possible to convert PN's to FSMs.
- d) [5 pts] What if the number of tokens is bounded; can you define a method for converting PN's to FSMs?
- 2) [10pts] A dead definition is a definition of a variable that is **never** referenced. Show how to formulate the detection of dead definitions as a data flow problem. Specifically, indicate if it is an any or all paths problem, a forward or backward data flow problem, give the definitions of IN, OUT, GEN, and KILL, give initial values, and define how the result is computed.

3) Consider the following code fragment for a GCD algorithm:

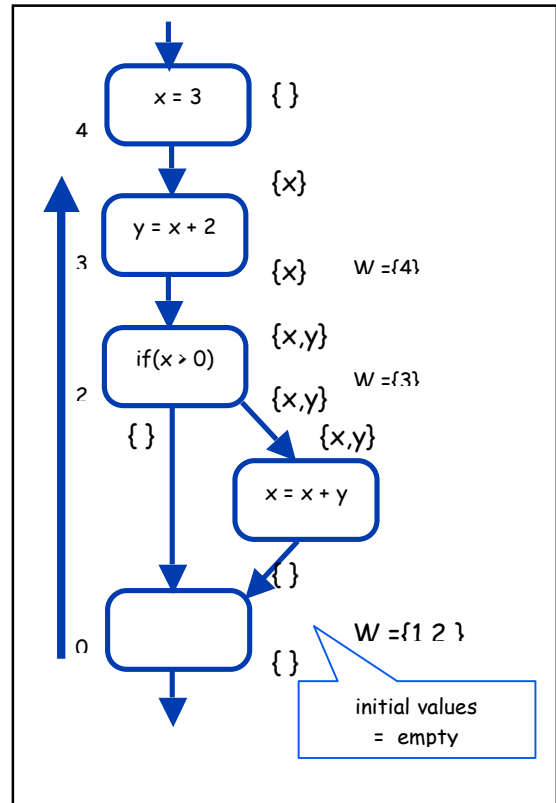
```
int F (int a, int b)
{
    int g = a, m = b;
    if (a < b) {g = b; m = a;}
    while (m)
    {
        int s = g;
        g = m;
        m = s % m;
    }
    return
    g;
}
```

- a. [10pts] Draw a control flow graph for this code fragment
- b. [10pts] Define “live variables” as used in data flow analysis.
 1. Is “live variable” analysis a forward or reverse flow problem?
 2. When would you use “all paths” and when would you use “any paths” in “live variable” analysis?
 3. Define IN & OUT functions and the GEN & KILL sets for “live variable” analysis
- c. [20pts] Using one of the attached **iterative** algorithms; perform a “live variable” analysis of the GCD algorithm for any path. Use a table to show your results.
- d. [5pts] Label the flow graph using the “worklist” algorithm attached.
- e. [10pts - **621 only or 521 extra credit**] Suppose the “if statement” in the box were deleted. Would the fragment compute the same result? Use symbolic execution to support your answer.

4) [10pts] BONUS: Find an error in the attached data flow algorithms.

Algorithm 1: (form of a worklist algorithm)

1. Start at initial node (entry for forward; exit for reverse), label IN_0 with pertinent "facts" (initial values)
2. Compute $OUT_0 = F(IN_0)$ (label OUT_0 with the computed facts)
3. Propagate OUT_0 to IN_i (label edge $N_0 \Rightarrow N_i$ with OUT_0) where N_i are successor nodes (forward) or predecessor nodes (reverse) of N_0
4. Compute $OUT_i = F(IN_i)$, place all N_i on a "worklist" W , and for all N_i label OUT_i with the computed facts.
5. While W is not empty,
 - a. pick N_i from W and propagate OUT_i to IN_k (label edges $N_i \Rightarrow N_k$ with OUT_i) where N_k are successor nodes (forward) or predecessor nodes (reverse) for N_i ; delete N_i from W
 - b. Compute $OUT_k = F(IN_k)$ for all N_k where $IN_k = \text{MERGE}$ all input edge labels ($\text{MERGE} = \cup$ for "some paths" and \cap for "all paths"), label OUT_k with the computed facts); and if for N_k , OUT_k changes put N_k on W'
6. If W' is not empty, then $W = W'$ and go to 5



Algorithm 2: (iteration)

1. For all nodes N_i , $IN_i(0) = \{ \}$, Compute $OUT_i(0) = F(IN_i(0))$
2. $k=1$
3. For $l = 0$ to n (N_0 is the entry node for forward; the exit node for reverse)
 - a. Compute $IN_i(k) = \text{MERGE } OUT_j(k)$ ($\text{MERGE} = \cup$ for "some paths" and \cap for "all paths"), where N_j is a successor (backward) or predecessor (forward) node of N_i .
 - b. Compute $OUT_i(k) = F(IN_i(k))$
 - a. if for all l , $OUT_i(k) = OUT_i(k-1)$ then exit else $k = k+1$ and go to 3

Algorithm 3: (iteration)

1. $IN_0(0) = \{ \}$, Compute $OUT_0(0) = F(IN_0(0))$ (N_0 is the entry node for forward; the exit node for reverse)
2. $k=0$
3. for $j = 1$ to n
 - a. For all N_i where N_i is a successor (backward) or predecessor (forward) node of N_j , compute $IN_i(k) = \text{MERGE } OUT_j(k)$ ($\text{MERGE} = \cup$ for "some paths" and \cap for "all paths")
 - b. Compute $OUT_j(k+1) = F(IN_j(k))$
4. if for all l , $OUT_i(k) = OUT_i(k-1)$ then exit else $k = k+1$ and go to 3