Converting a program into a Minimal SSA form

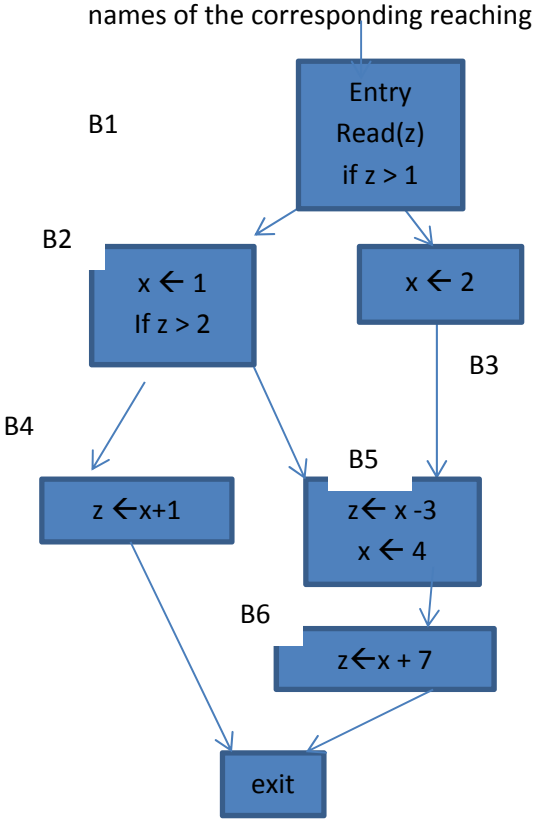
As we showed in the lecture, it is possible for the new definitions (for ϕ functions) inserted in the iterative dominance frontier to reach no uses of the defined variables. Such new definitions are wasted. A minimal SSA form only has ϕ functions inserted where the involved variables are live.

Converting a program into a minimal SSA for takes the following steps.

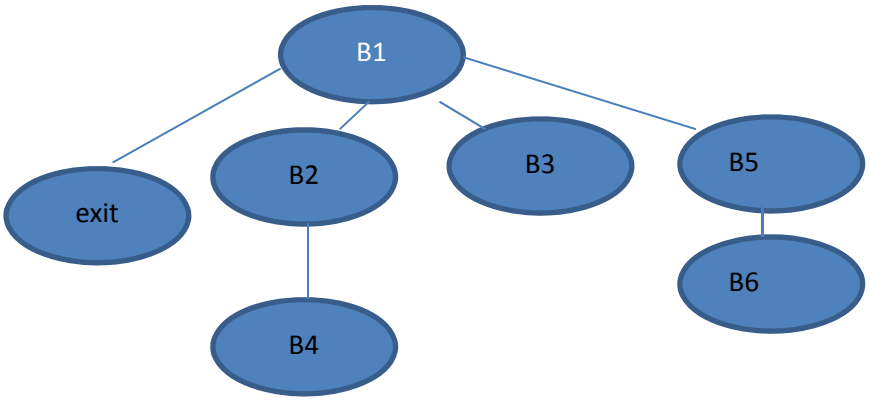
Step 1: rename the defined variables such that each definition is applied to a distinct name.

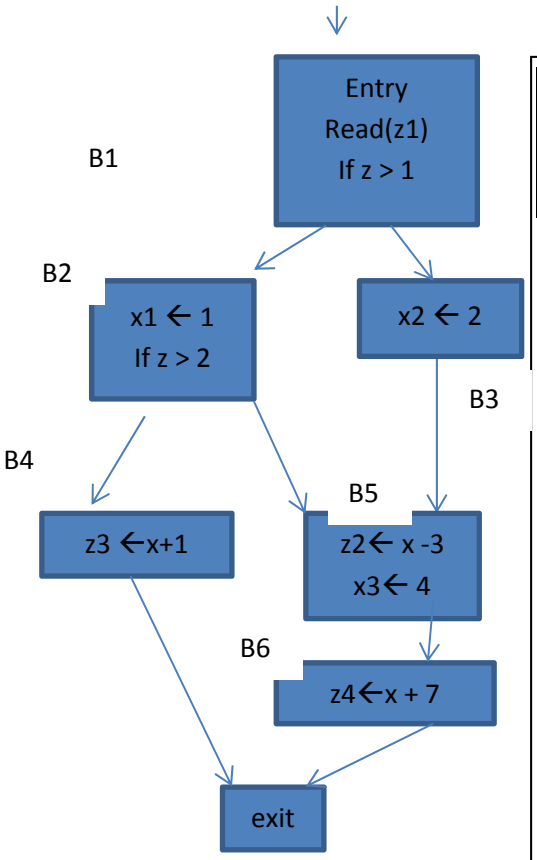
Step 2: For each renamed variable, x, let B1 be the block containing x. We compute the iterative dominance frontier of B1, denoted by DF+(B1). (The previous lecture notes gave a formal definition of iterative dominance frontiers.) For each member B2 in DF+(B1), if x is in Live_in(B2), then introduce a new name for x, say x' unless a ϕ function already exists in B2 for a renamed x. For x', we insert assignment x' = ϕ() and for now leaves the parameters in the ϕ function to be determined.

Step 3: After ϕ locations and their corresponding new definitions are determined for all renamed variables. We recomputed reaching definitions such that (i) every use of variable, say x, in the original program can be renamed according to the unique reaching definition, and (ii) every ϕ function has the names of the corresponding reaching definitions inserted as its parameters.



For the program and its control flow graph shown to the left, we build the dominator tree to facilitate the computation of dominance frontiers. The DOM tree is shown below:





n	B4	B2	B3	B6	B5
DF_local	exit	B5	B5	exit	
DF_up		exit			exit
DF	exit	B5,exit	B5	exit	exit

Obviously DF(B1) = DF(exit) = empty . We skipped them when traversing the dom tree bottom up.

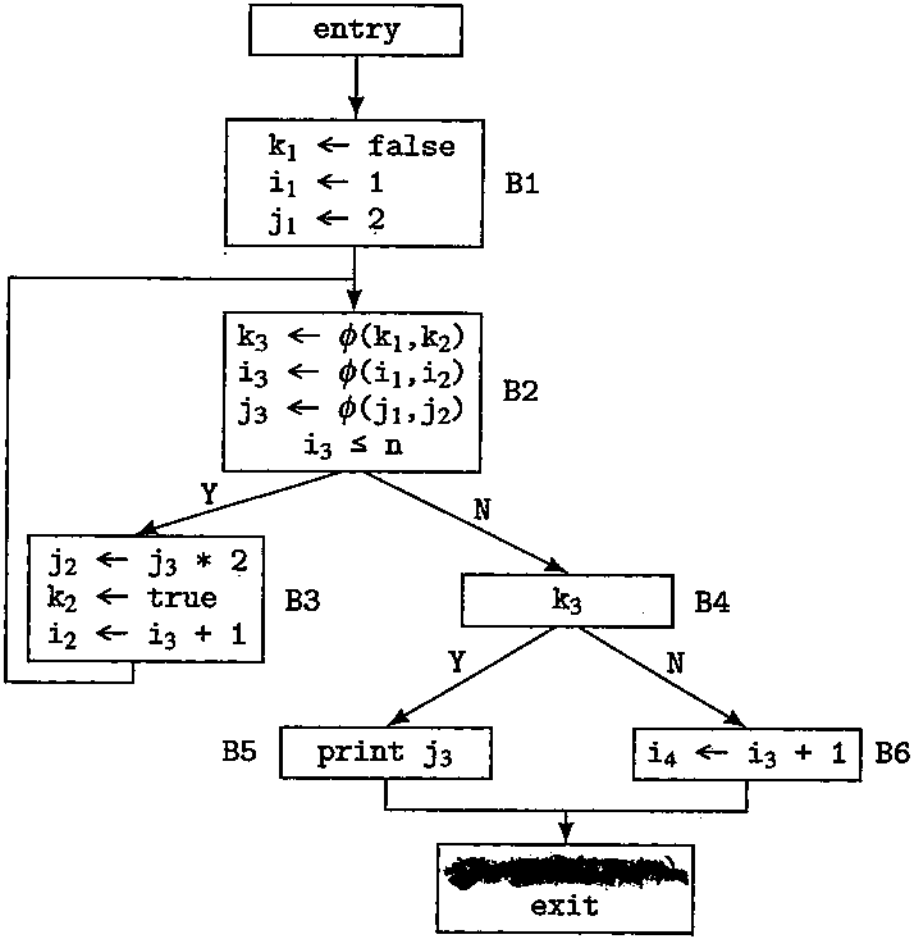
The program after renaming the defs is shown to the left.

For z1, DF(B1) is empty. For z2, z3 and z4, the blocks containing them all have the exit being the DF. But z is not in Live_in(exit) and no ϕ function is needed in exit.

For x1 and x2, we have DF(B2)=DF(B3)={B5,exit} which is also its iterative dominance frontier. Live_in(B5) contains x, and therefore we insert a new def x4 ← ϕ(?) in the beginning of B5. For x3, DF(B5) = exit whose Live_in is empty.

The only ϕ function inserted is x4←ϕ(x1,x2). The parameters x1 and x2 are determined by finding x1 and x2 reaching the top of B5. Reaching def also leads to the renaming of uses, as shown in the code to the left.

An example of minimal SSA transformation for a program with a loop is shown on the next page.


entry
k1 ← false
i1 ← 1
j1 ← 2
B1
k3 ← φ(k1,k2)
i3 ← φ(i1,i2)
j3 ← φ(j1,j2)
i3 ≤ n
B2
Y
N
j2 ← j3 * 2
k2 ← true
i2 ← i3 + 1
B3
k3
B4
Y
N
B5
print j3
i4 ← i3 + 1
B6
exit