Assignment 2

Flávio Cruz and Richard Veras

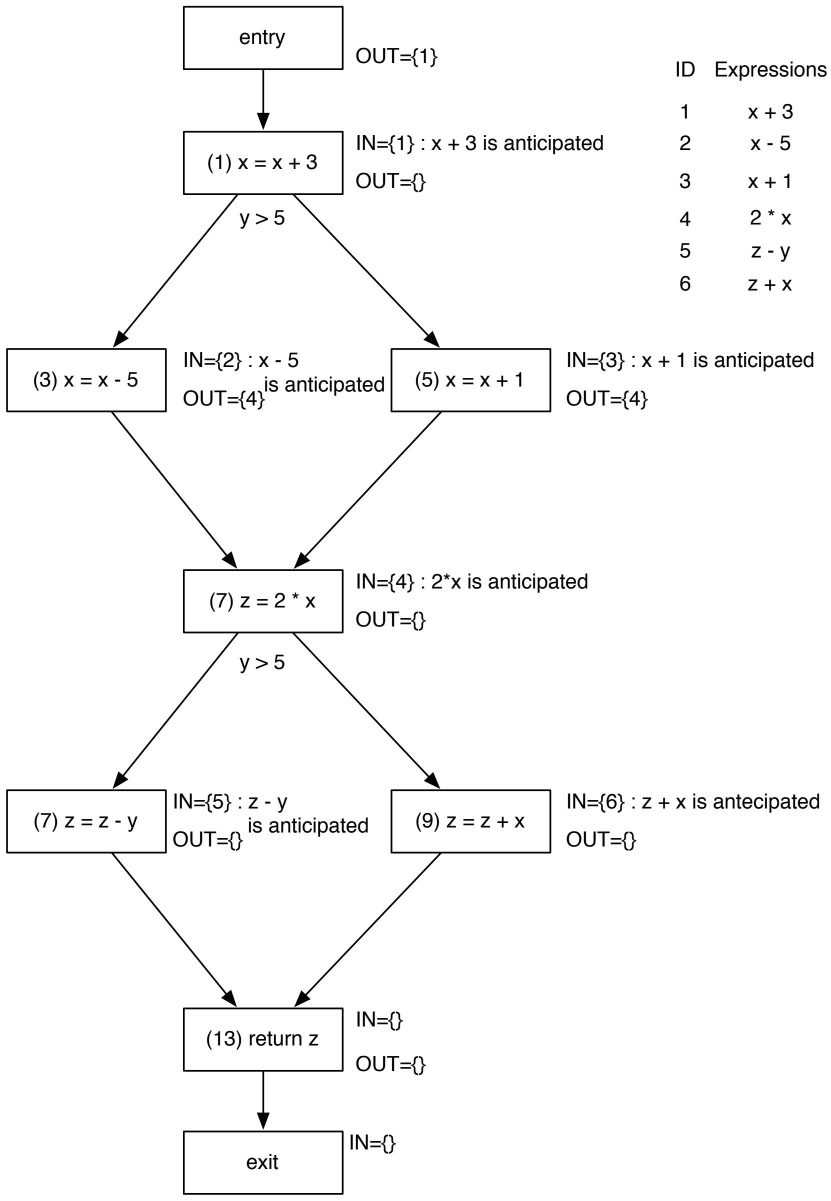
February 16, 2012

1 Questions

1.1 Lazy Code Motion

Item 1

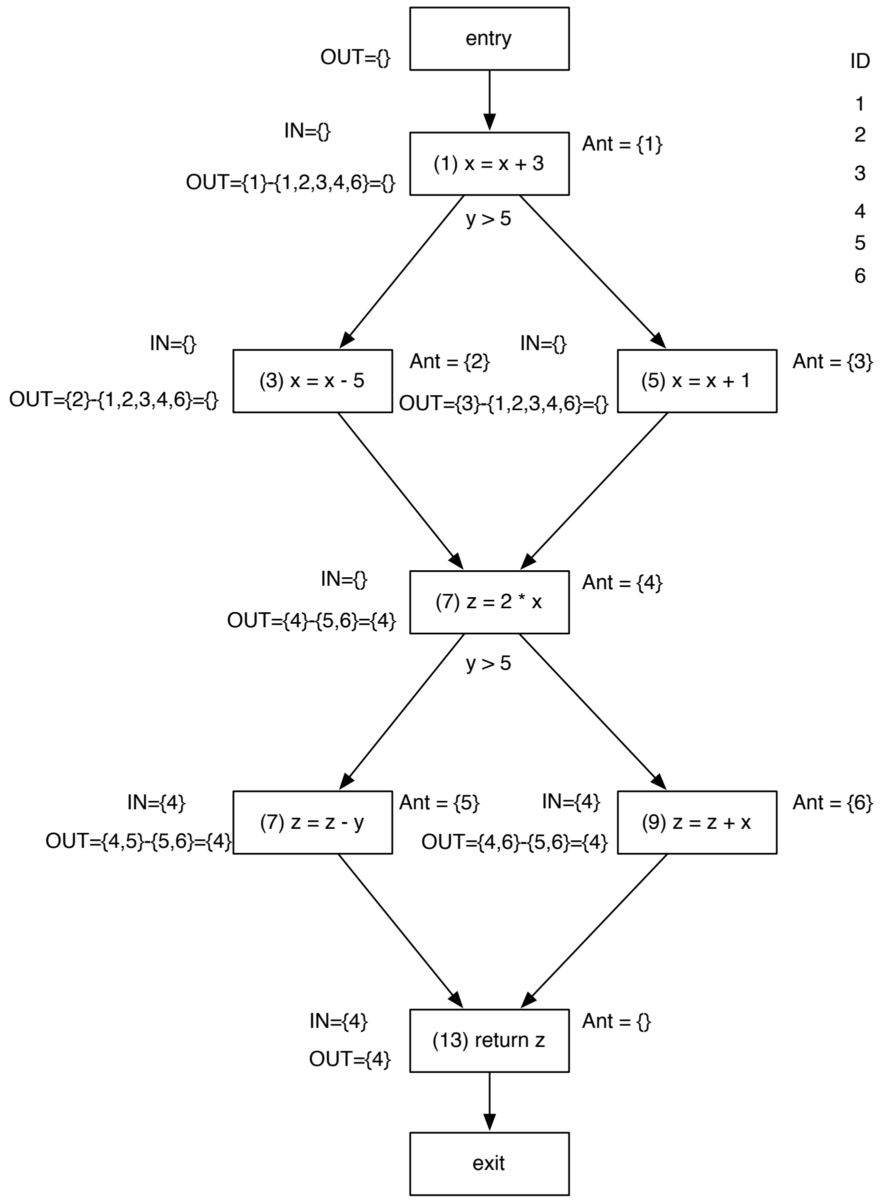
Figure 1: CFG after pass 1: anticipated expressions.


entry
OUT={1}
(1) x = x + 3
IN={1} : x + 3 is anticipated
OUT={}
y > 5
(3) x = x - 5
IN={2} : x - 5 is anticipated
OUT={4}
(5) x = x + 1
IN={3} : x + 1 is anticipated
OUT={4}
(7) z = 2 * x
IN={4} : 2*x is anticipated
OUT={}
y > 5
(7) z = z - y
IN={5} : z - y is anticipated
OUT={}
(9) z = z + x
IN={6} : z + x is antecipated
OUT={}
(13) return z
IN={}
OUT={}
exit
IN={}

ID	Expressions	Kill
1	x + 3	x
2	x - 5	x
3	x + 1	x
4	2 * x	x
5	z - y	z, y
6	z + x	z, x

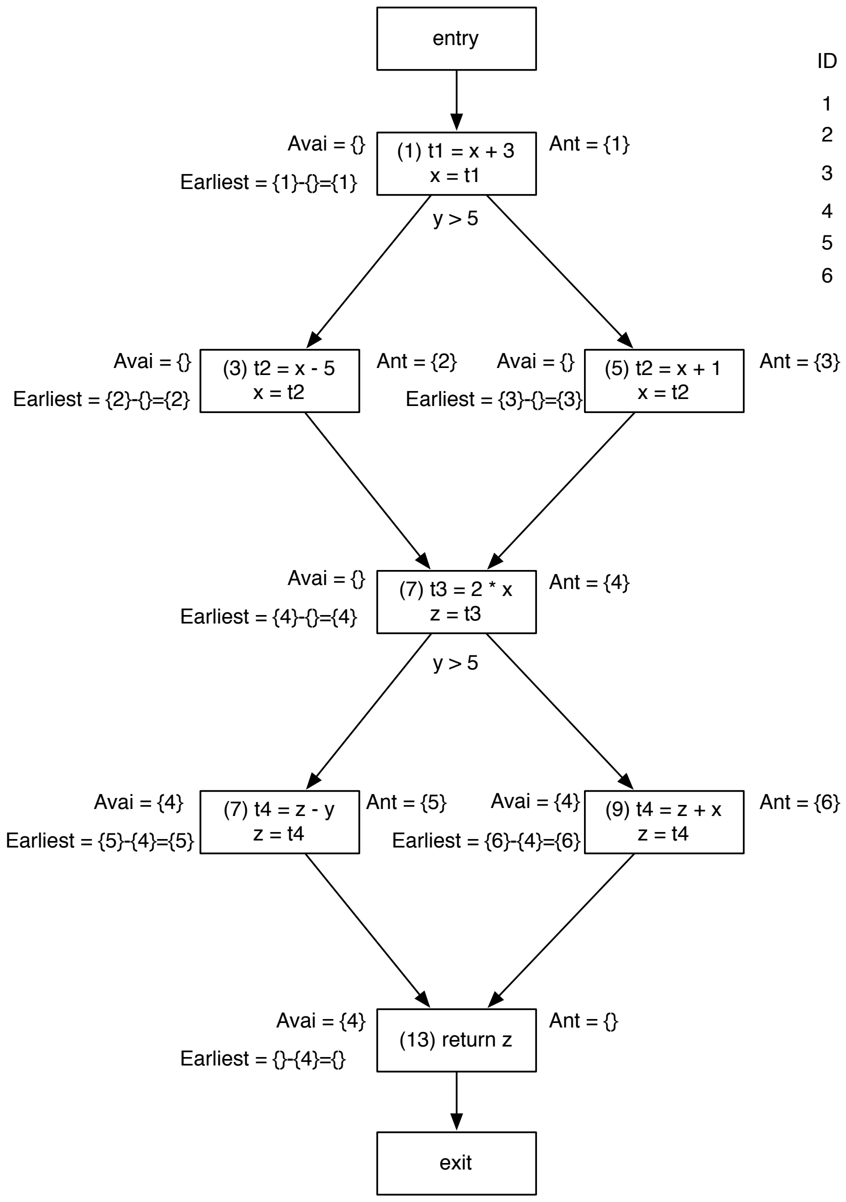
Item 2

Figure 2: CFG after pass 2: early placement.



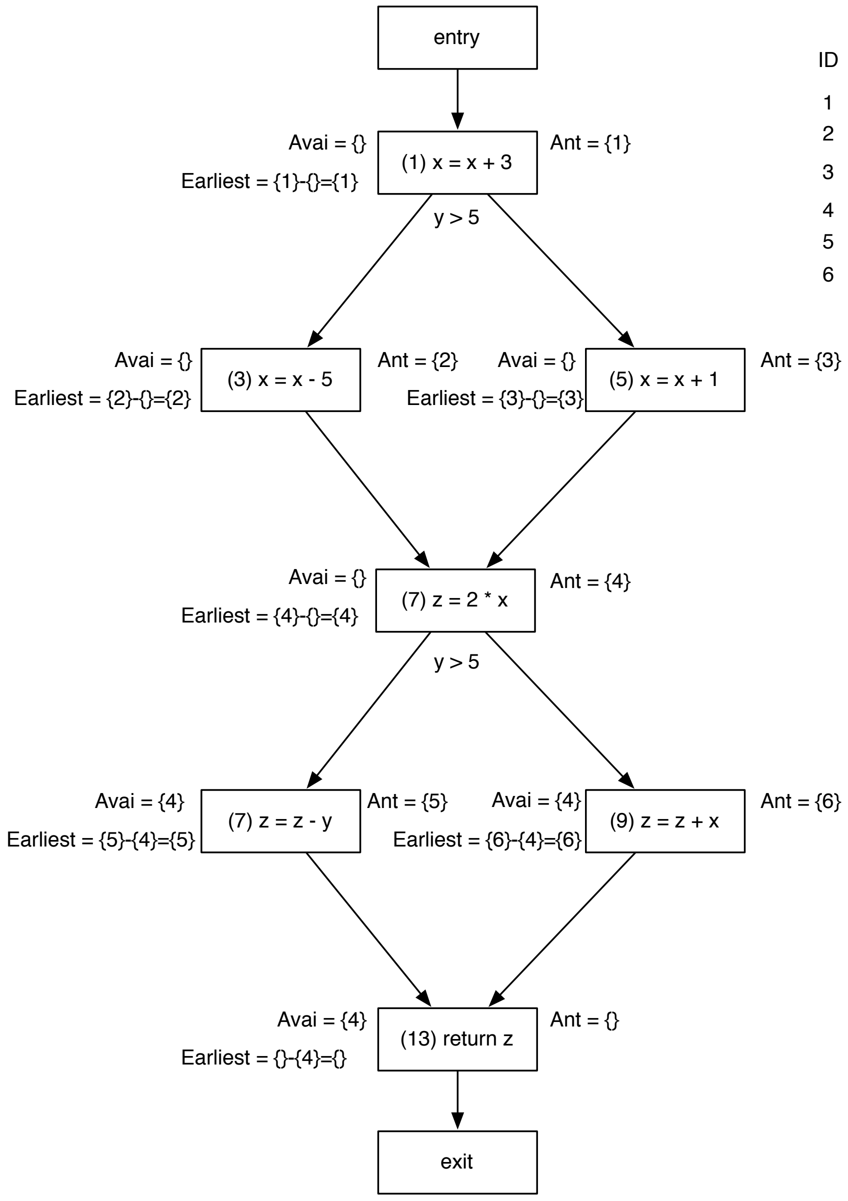
ID	Expressions	Kill
1	x + 3	x
2	x - 5	x
3	x + 1	x
4	2 * x	x
5	z - y	z, y
6	z + x	z, x

Figure 3: CFG after pass 2: computing `earliest` and adding instructions.



ID	Expressions	Kill
1	x + 3	x
2	x - 5	x
3	x + 1	x
4	2 * x	x
5	z - y	z, y
6	z + x	z, x

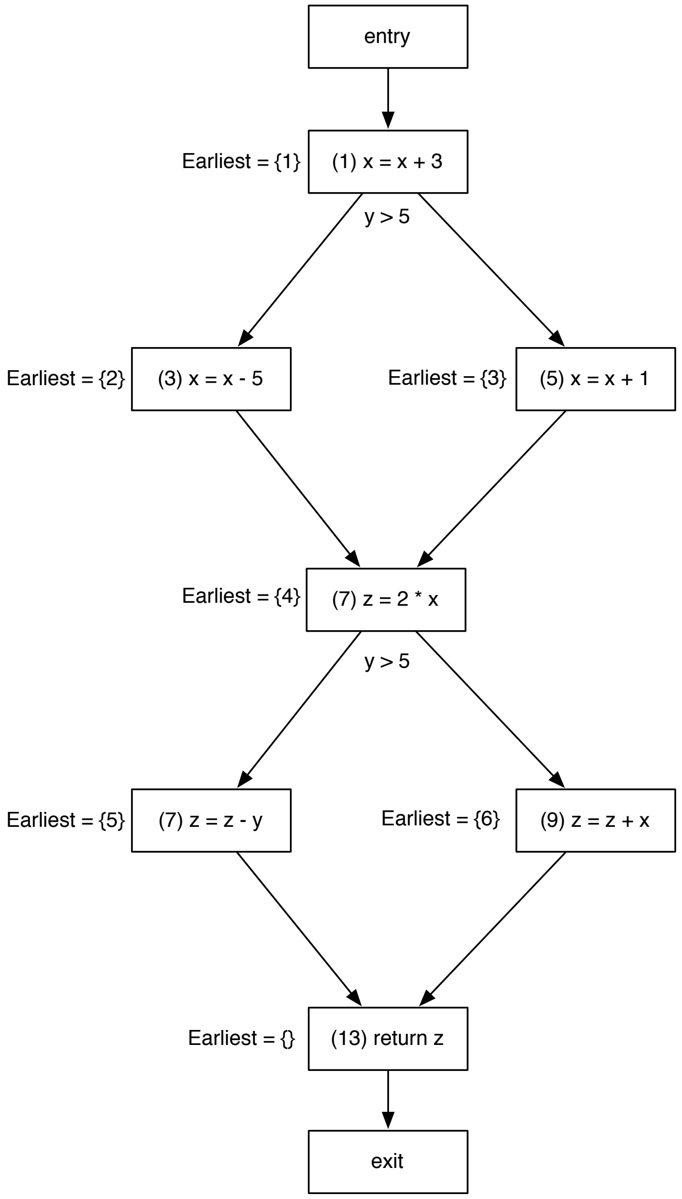
Figure 4: CFG after pass 2: constant folding.



ID	Expressions	Kill
1	x + 3	x
2	x - 5	x
3	x + 1	x
4	2 * x	x
5	z - y	z, y
6	z + x	z, x

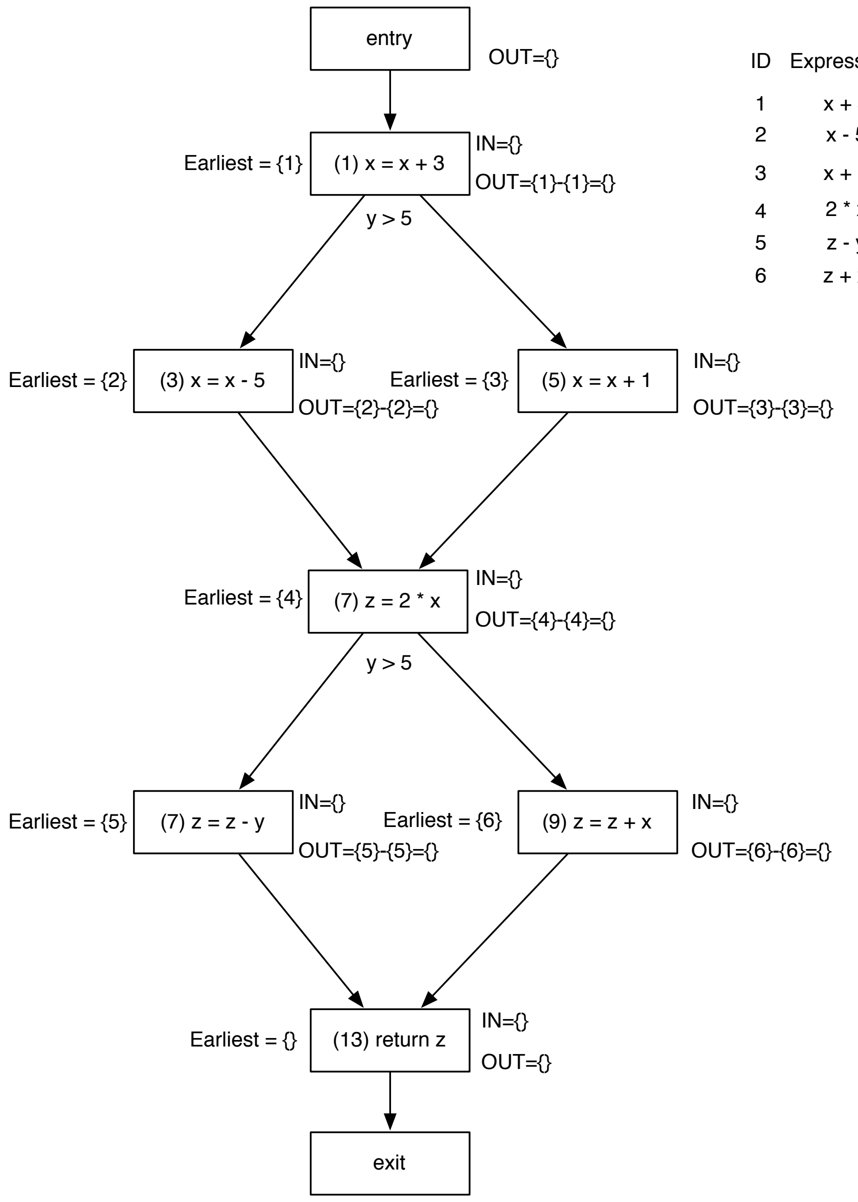
Item 3

Figure 5: CFG after pass 2: cleanup.



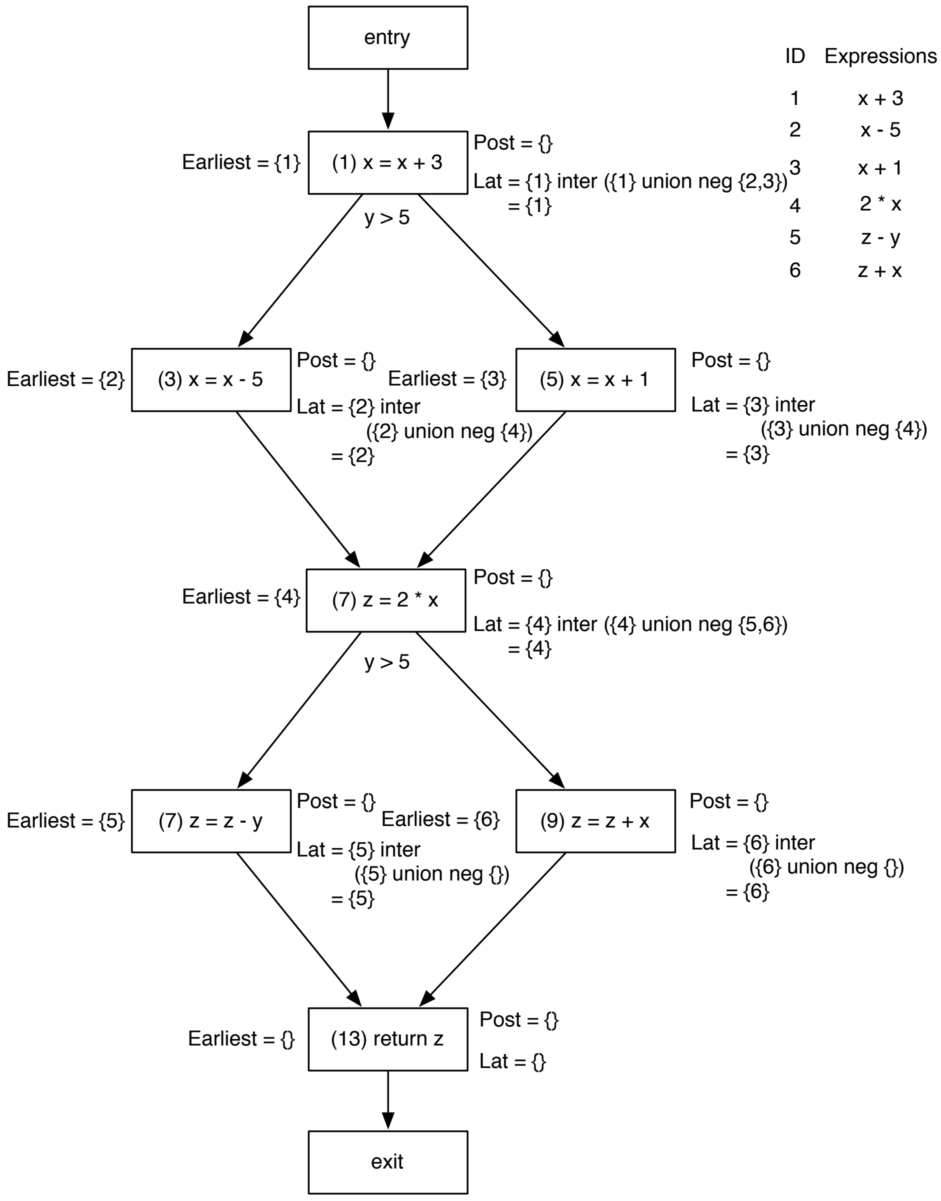
ID	Expressions	Kill
1	x + 3	x
2	x - 5	x
3	x + 1	x
4	2 * x	x
5	z - y	z, y
6	z + x	z, x

Figure 6: CFG after pass 3: lazy code motion.



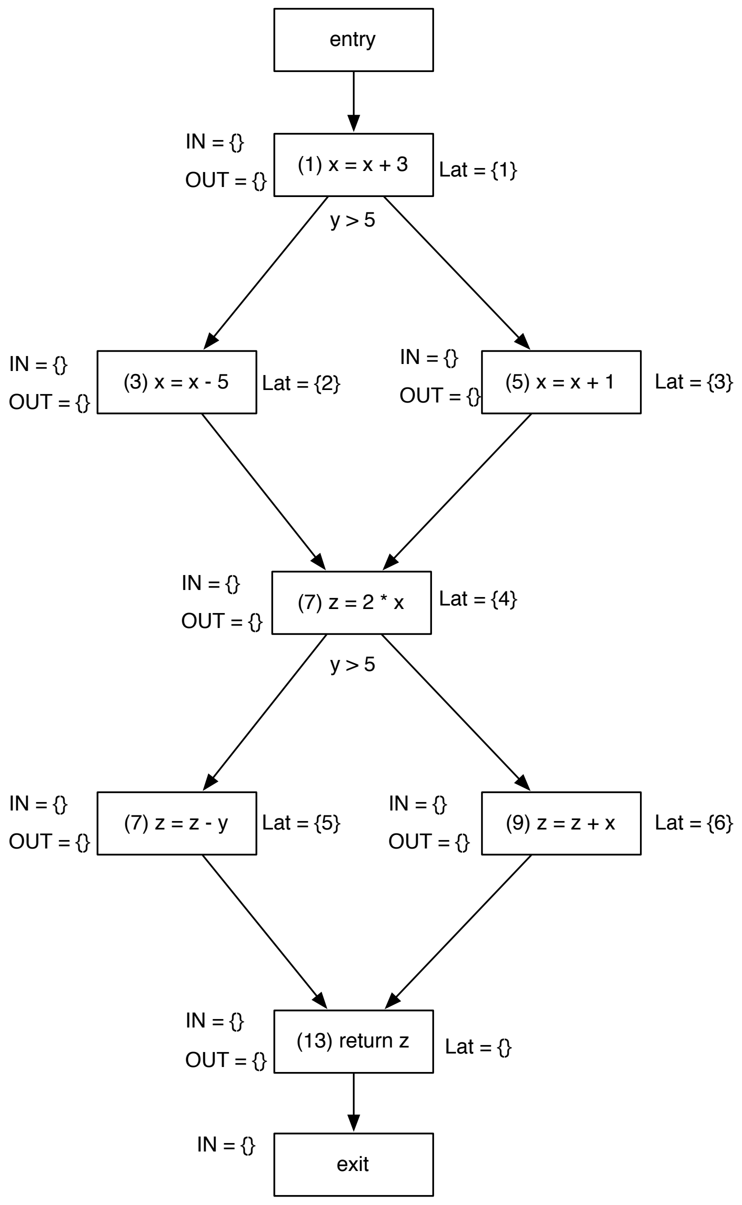
ID	Expressions	Kill
1	x + 3	x
2	x - 5	x
3	x + 1	x
4	2 * x	x
5	z - y	z, y
6	z + x	z, x

Figure 7: CFG after pass 3: computing latest.



ID	Expressions	Kill
1	x + 3	x
2	x - 5	x
3	x + 1	x
4	2 * x	x
5	z - y	z, y
6	z + x	z, x

Figure 8: CFG after pass 4: cleaning up.



ID	Expressions	Kill
1	x + 3	x
2	x - 5	x
3	x + 1	x
4	2 * x	x
5	z - y	z, y
6	z + x	z, x

1.2 LICM: Loop Invariant Code Motion

Loop invariant instructions:

- S2: `y = 5`
- S3: `q = 7`
- S9: `m = y + 7` (only one reaching definition: S2)
- S12: `r = q + 9` (only one reaching definition: S3)

S10 is not an invariant since g has two reaching definitions (`g = 3` and `g = 4`).

S6 can also be considered a loop invariant, but execution can skip this block and `print` needs to print an undefined x value. So if this instruction was to be moved to the pre-header, it would print 1.

Moved instructions by loop invariant code motion pass:

For each statement s previously listed defining x, we move s to preheader if:

- s is in a block that dominates all exists of the loop,
- x is not defined elsewhere in the loop, and
- s is in a block that dominates all uses of x in the loop.

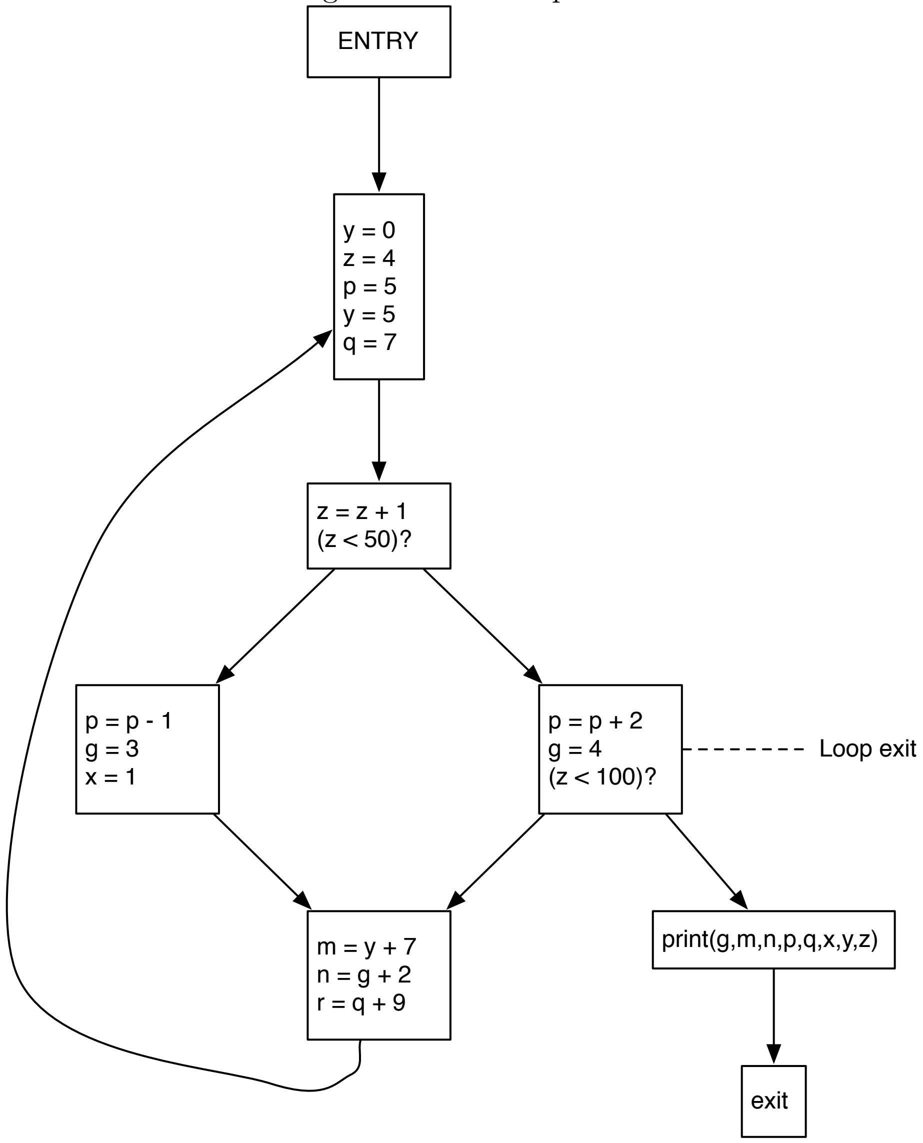
Since S11 is dead code, we removed this instruction and moved S2 (`y = 5`) to the pre-header.

We cannot move both S9 and S12, since both may not be executed at all, which would jeopardize the program logic when the owner block is not executed.

S3 can be moved without problems because the owner block is always be executed and satisfies all the conditions we listed previously.

The final CFG is shown in Fig. 9.

Figure 9: Final loop CFG.


ENTRY
y = 0
z = 4
p = 5
y = 5
q = 7
z = z + 1
(z < 50)?
p = p - 1
g = 3
x = 1
p = p + 2
g = 4
(z < 100)?
Loop exit
m = y + 7
n = g + 2
r = q + 9
print(g,m,n,p,q,x,y,z)
exit