Loop-invariant code motion

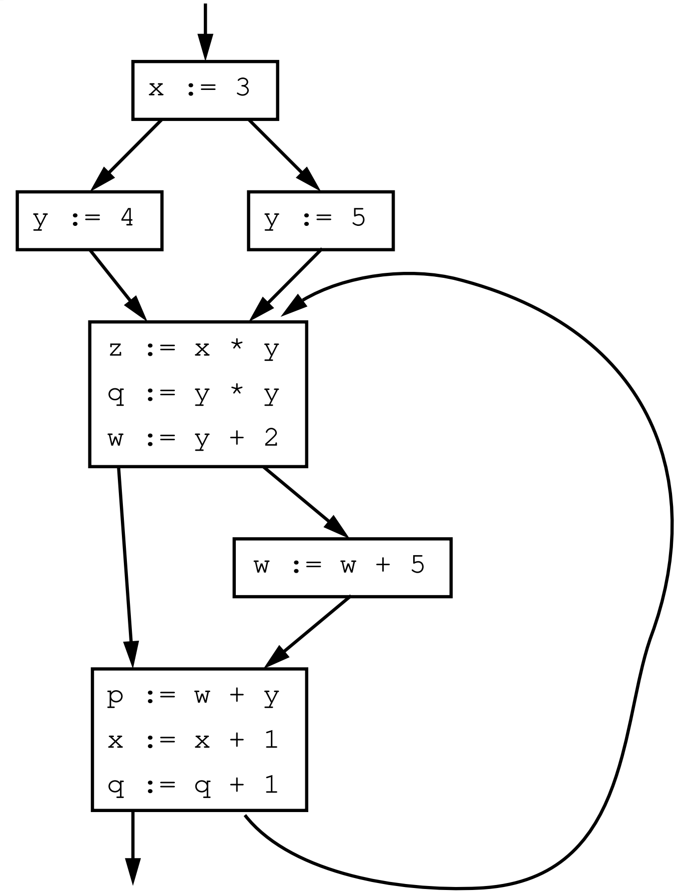
Step 1: find invariant computations in loop

- invariant: computes same result each time evaluated

Step 2: move them outside loop

- to top: **code hoisting**
- to bottom: **code sinking**

Example



Detecting loop-invariant expressions

An expression is invariant w.r.t. a loop *L* iff:

base cases:

- it's a constant
- it's a variable use, all of whose defs are outside *L*

inductive cases:

- it's an idempotent computation
  all of whose args are loop-invariant
- it's a variable use whose single reaching def is
  loop-invariant

A def is loop-invariant if its rhs is loop invariant

Computing loop-invariant expressions

Option 1:

- repeat iterative dfa
  until no more invariant expressions found

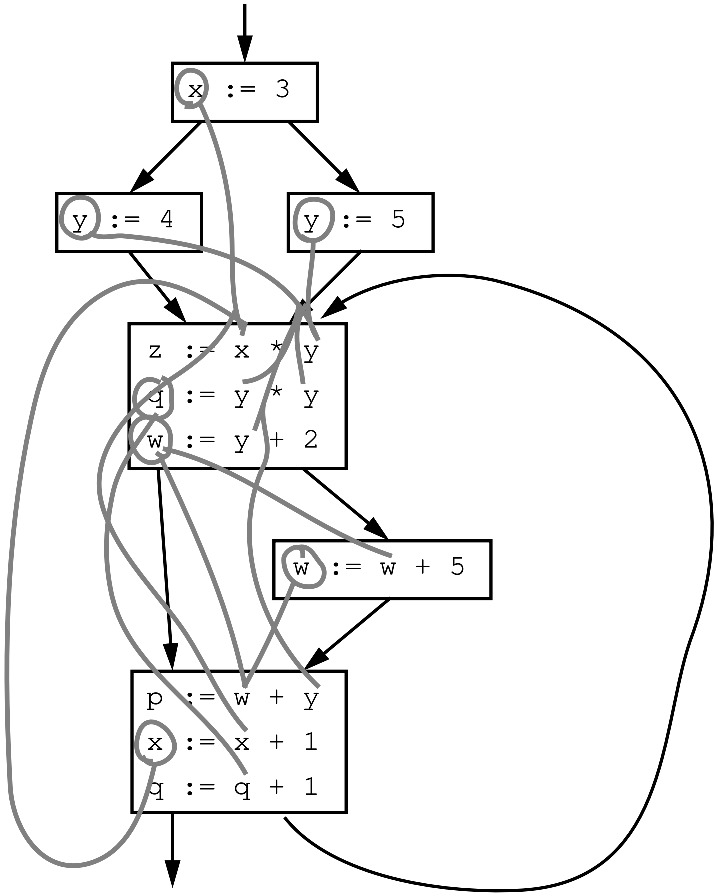
Option 2:

- build def/use chains,
  follow chains to identify & propagate
  invariant expressions

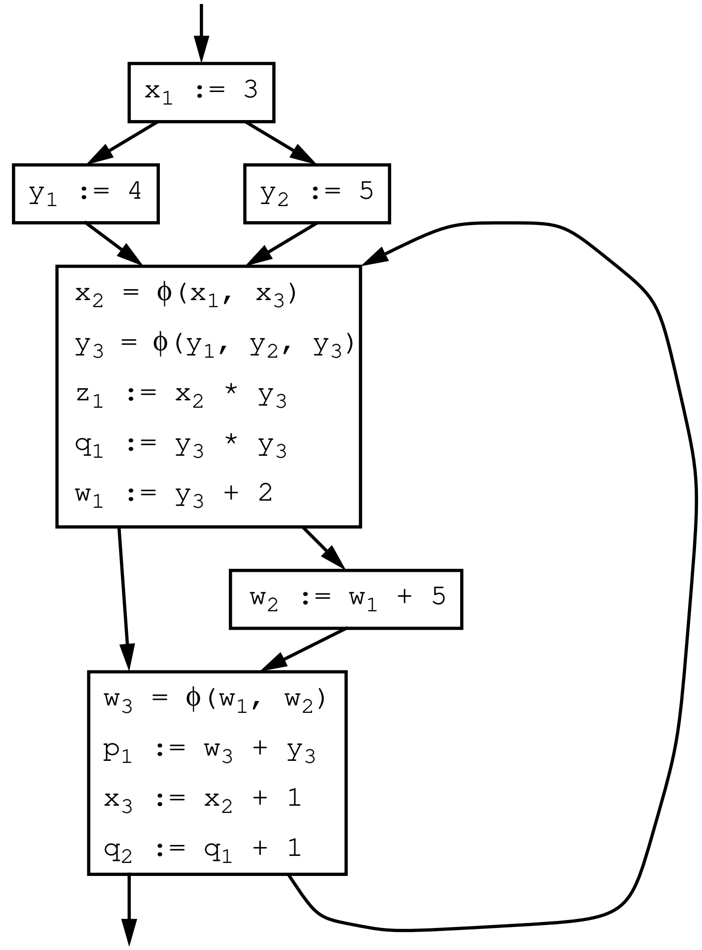
Option 3:

- convert to SSA form,
  then similar to def/use form

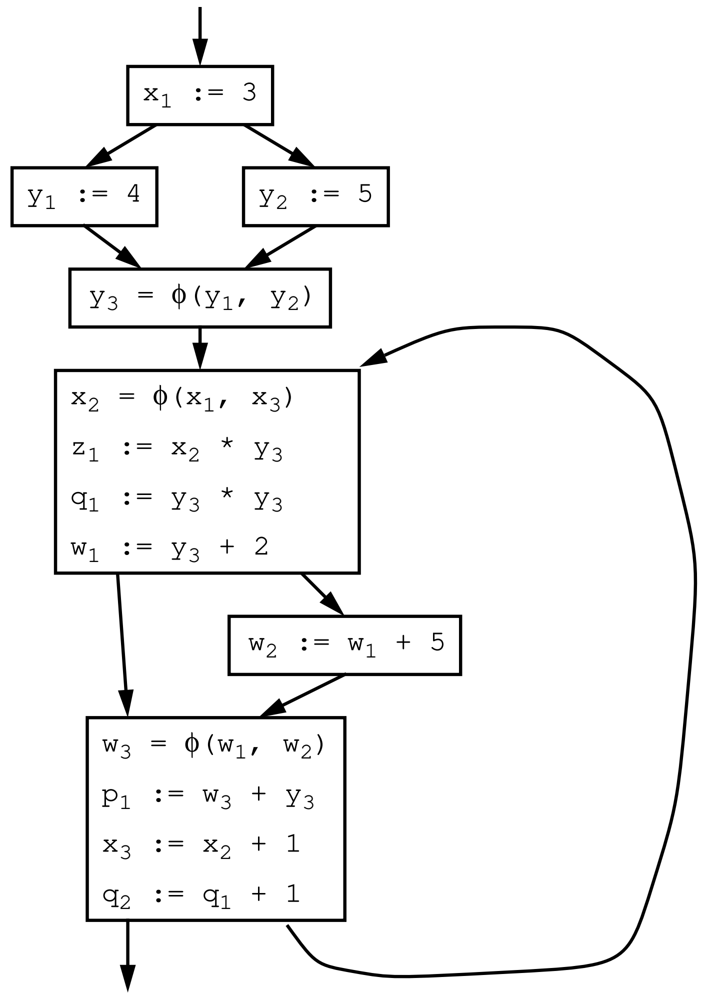
Example using def/use chains



Example using SSA form



Example using SSA form & preheader



Loop-invariant expression detection for SSA form

An expression is invariant w.r.t. a loop *L* iff:

- it's a constant
- it's a variable use, whose def is outside *L* or loop-invariant
- it's an idempotent (non-ϕ) computation
  all of whose args are loop-invariant

A def is loop-invariant if its rhs is loop invariant

Code motion

When find invariant computation *S:* `z := x op y`,
want to move it out of loop (to loop preheader)

When is this legal?

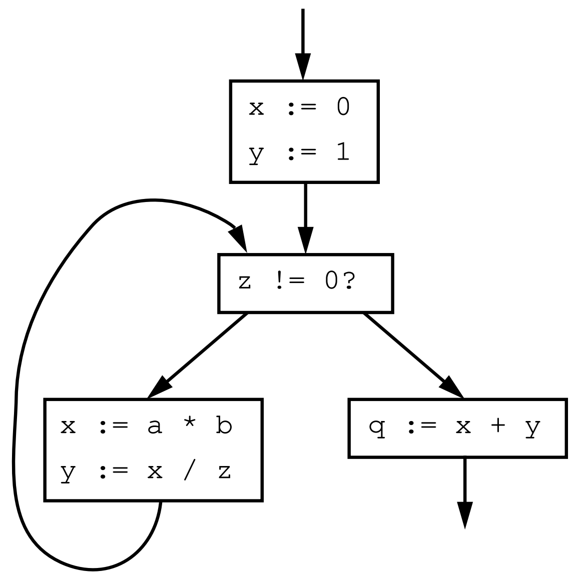
Sufficient conditions:

- *S* **dominates** all loop exits
  [*A* dominates *B* when
  all paths to *B* must first pass through *A*]
- *S* is only assignment to `z` in loop
- no use of `z` in loop is reached by any def other than *S*

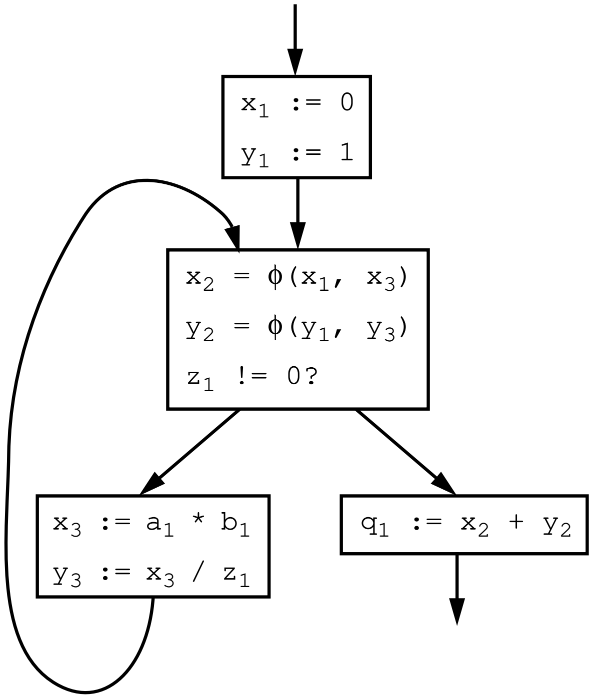
If met, then can move *S* to loop preheader

- but move in order discovered,
  to preserve data flow among moved statements

Example of domination



Example of domination in SSA form



Avoiding domination restriction

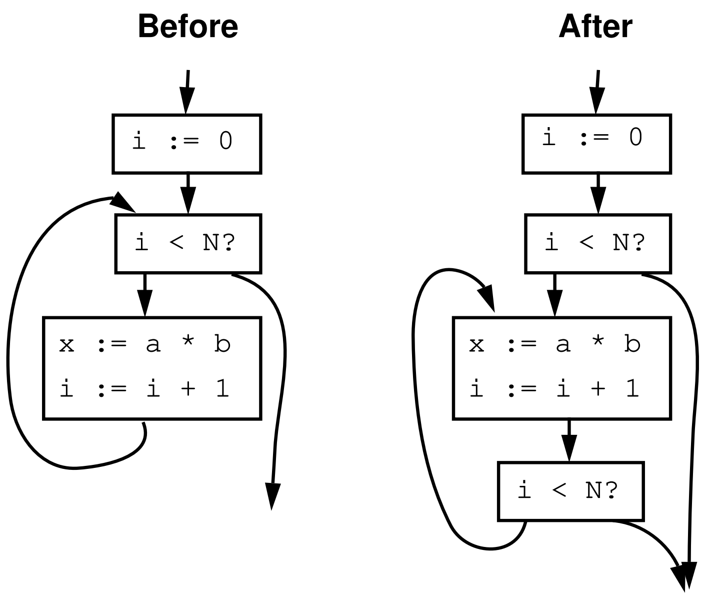
Requirement that invariant computation dominates exit is strict

- nothing in conditional branch can be moved
- nothing after loop exit test can be moved

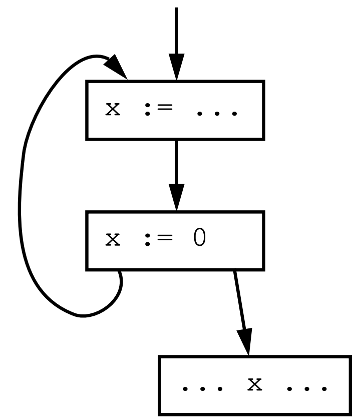
Can be circumvented through other transformations such as **loop canonicalization**

- move loop exit test to bottom of loop

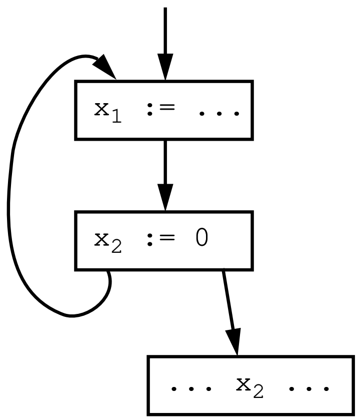
Example of loop canonicalization



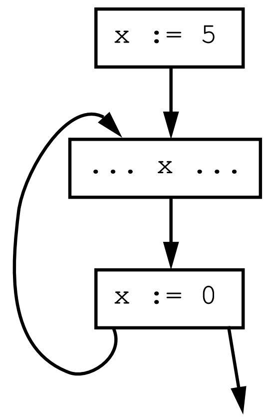
Example of “only assignment in loop” restriction



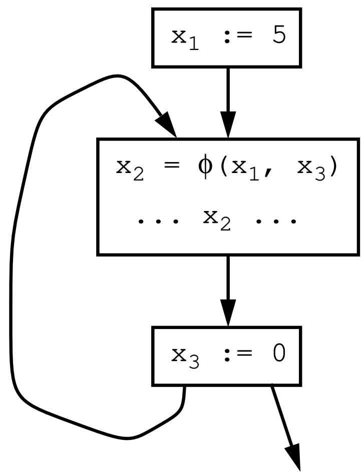
Example in SSA form



Example of “no other reaching defs” restriction



Example in SSA form



Loop-invariant code copying

Alternative to code motion:
copy instruction to loop header,
then do CSE & copy propagation to simplify in-loop version

Can always copy, unless instruction has side-effects

CSE & copy propagation will either eliminate in-loop instruction
or replace with copy instruction

Example