Some pragmatics of dataflow analysis

When solving constraints, proceed topologically where possible

- straight-line constraints: **flow functions**
- flow functions often broken down into KILL set and GEN set:
  $F_{instr}(in) = in - KILL_{instr} \cup GEN_{instr}$
- can sometimes represent info/KILL/GEN sets as bit vectors
  (if can express abstractly as set of booleans)
- can summarize effect of basic block as single flow function
  (e.g. by combining KILL & GEN sets of instructions),
  for faster iteration

To speed iterative analysis, want to test guess ASAP

- ideal: proceed along shortest path from
  loop head to loop tail
- practical: don't follow loop exit until fixpoint reached

Another example: constant propagation

Goal: data flow analysis that implements constant propagation

What info computed for each program point?

Initial info?

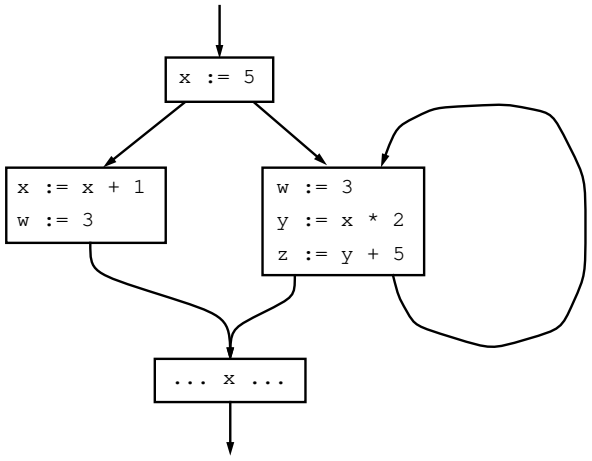
Effect of `x := y + z`?

Effect of `*p := *q + *r`?

Merge function?

Initial guess for back edge to start iterating?

Example



May vs. must info

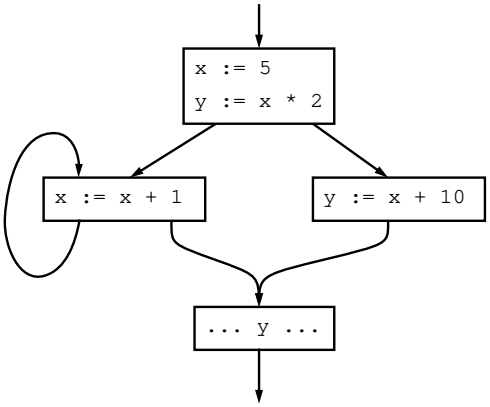
Some kinds of info imply possibilities: **may** info
Other kinds of info imply guarantees: **must** info

	May	Must
desired info	small set	big set
safe	overly big set	overly small set
GEN	add everything that might be true	add only if guaranteed true
KILL	remove only if guaranteed wrong	remove everything possibly wrong
MERGE	$\cup$	$\cap$
guess for back edge (optimistic)	$\varnothing$	universal set

Another example: live variables

Want the set of variables that are **live** at each pt. in program

- live: might be used later in the program

Supports dead assignment elimination, register allocation

Example