Moving Beyond Basic Blocks: Control Flow

- In order to perform global register allocation, we need to look beyond basic blocks.
- This requires additional information, specifically, about the *liveness* of names.
- This information:
  - Is not explicit in the program.
  - Must be calculated *statically* (i.e., at compile-time).
  - Must correctly characterize *all* dynamic (run-time) paths.
- Control flow makes it hard to extract this information.
  - Branches and loops in programs.
  - Different branches may be taken in different executions.
  - Different numbers of loop iterations may be executed in different executions.

Control Flow Graphs

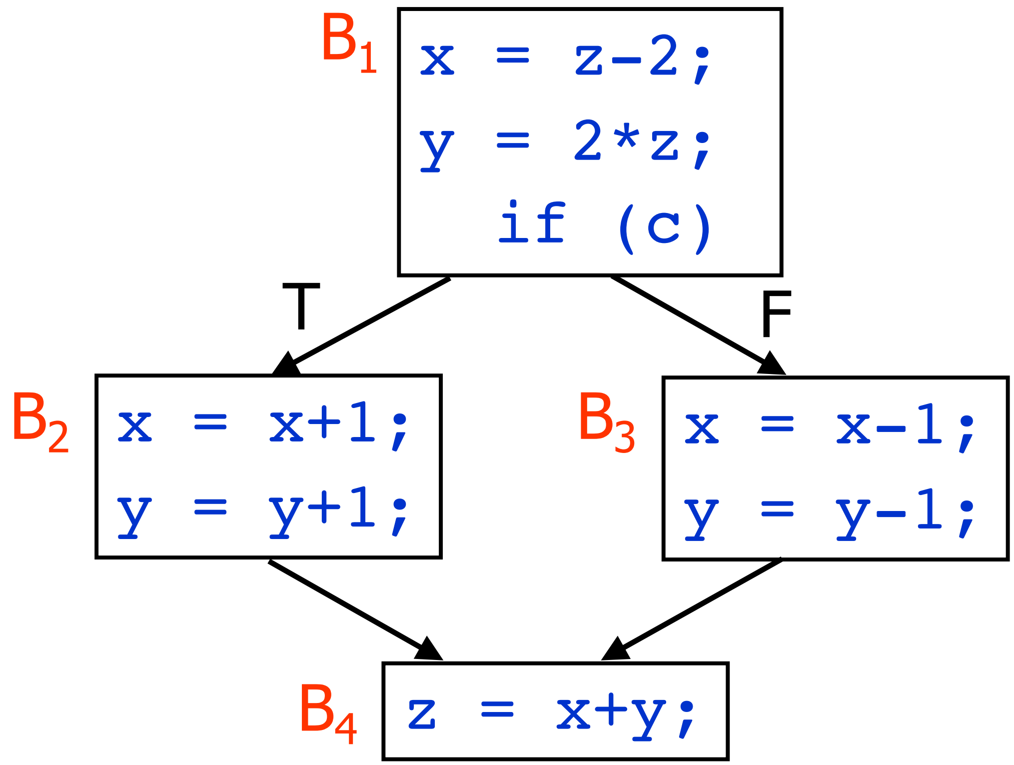
- The Control Flow Graph (CFG) is a graph representation of the computation and control flow in the program.
  - Provides a framework for static analysis of program control-flow.
- CFG nodes are basic blocks.
  - Recall: A basic block is a *maximal* straight-line, single-entry code sequence, with no branching except at the end of the sequence.
- CFG edges represent possible flow of control from the end of one basic block to the beginning of another basic block.
  - Edges are sometimes labeled with the Boolean value for which they are taken.
  - There may be multiple incoming/outgoing edges for a given basic block.
- A possible execution is a *consistent* path in the CFG.
  - There may be paths in the CFG that correspond to *infeasible* executions.

CFG Example

Program

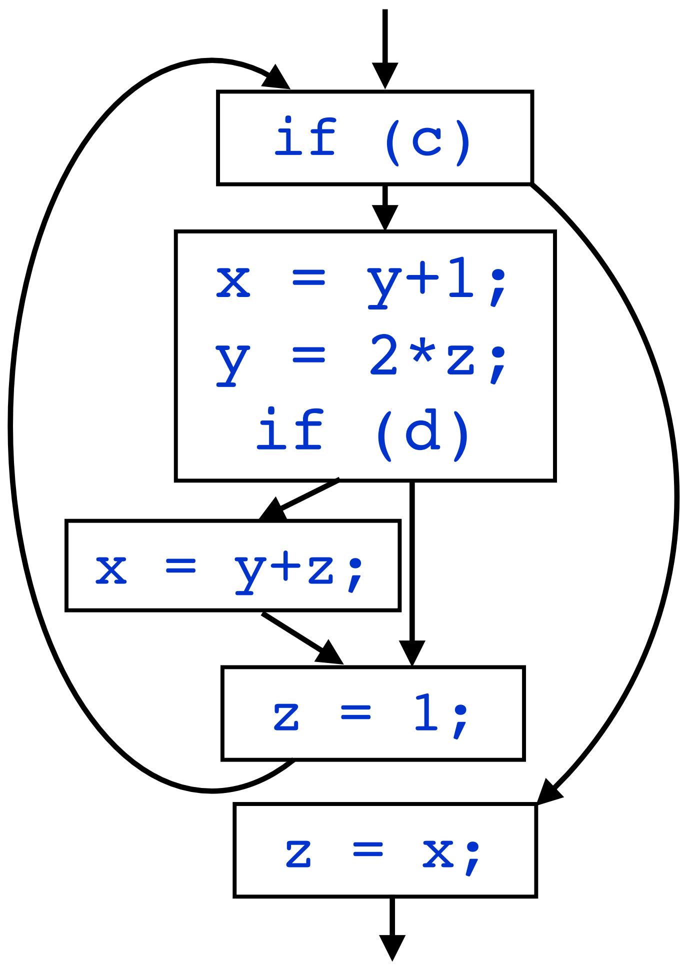
```
x = z-2;
y = 2*z;
if (c)  {
   x = x+1;
   y = y+1;
}
else {
   x = x-1;
   y = y-1;
}
z = x+y;
```

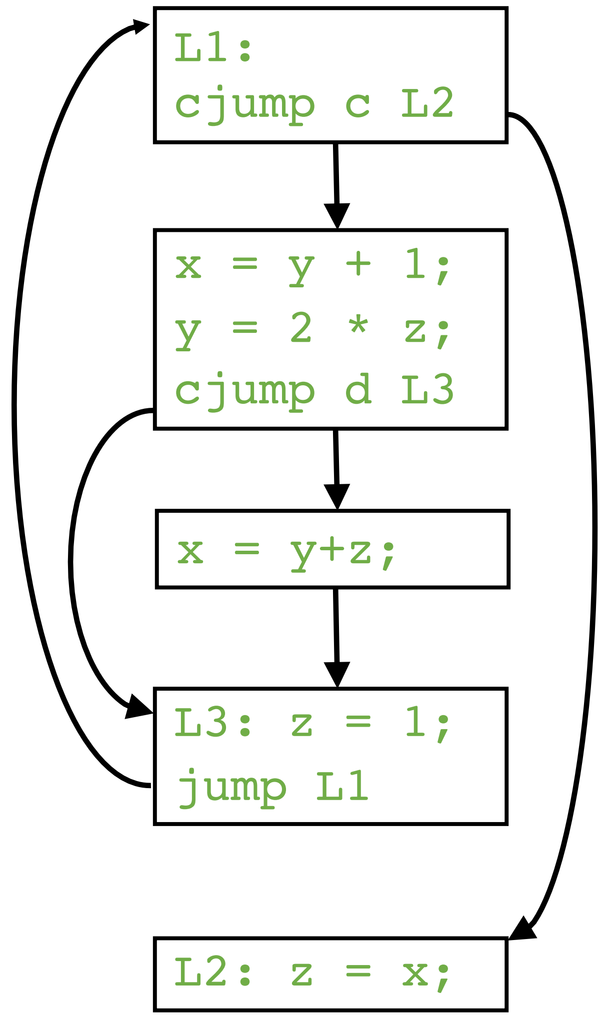
Control Flow Graph



A More Complicated CFG

```
while (c) {
   x = y+1;
   y = 2*z;
   if (d)
      x = y+z;
   z = 1;
}
z = x;
```





Using CFGs

- Use CFG to statically extract information about the program.
  - Reason at compile-time ...
  - ... about the run-time values of all variables and expressions across all program executions.

- Example of extracted information: live variables.

- Idea
  - Define program points in the CFG.
  - Reason statically about how information flows between these program points.

Program Points

- Two program points for an instruction:
  - A program point *before* the instruction.
  - A program point *after* the instruction.

Point before → •

$$x = y+1$$

Point after → •

- Within a basic block, the program point after an instruction is the same as the program point before its successor instruction.
- Extend the definitions to basic blocks in the obvious manner.
  - The program point before a basic block is the program point before the first instruction in the basic block.
  - The program point after a basic block is the program point after the last instruction in the basic block.

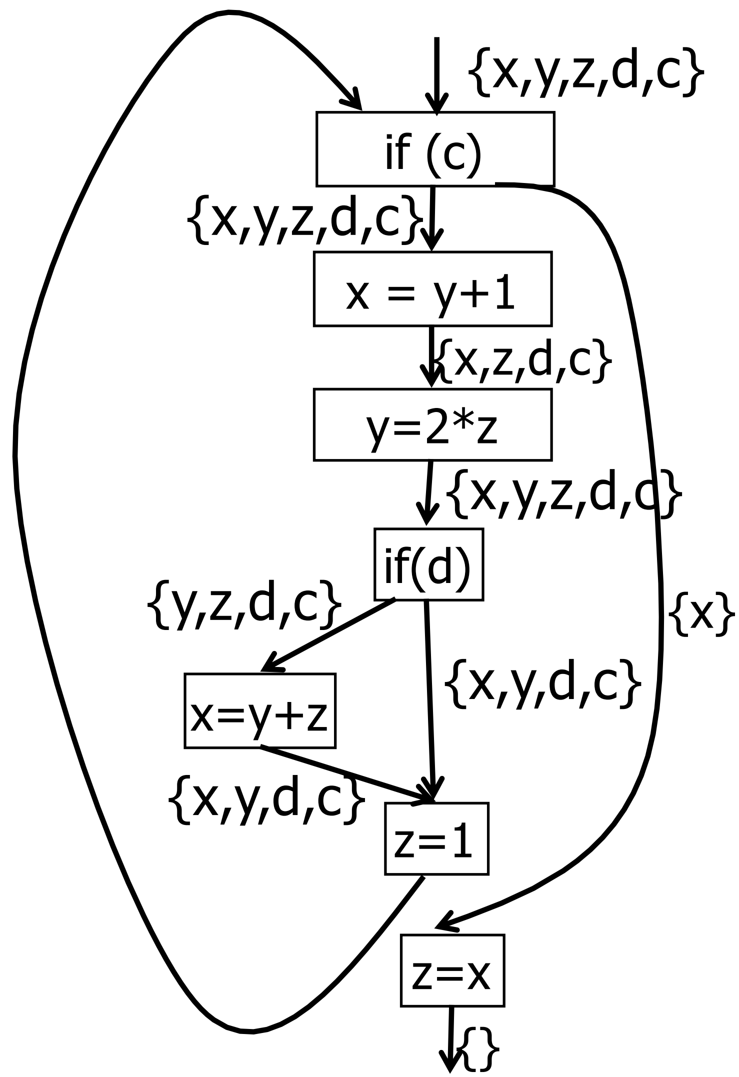
Flow of Extracted Information

- Question 1: How does the information flow between the program points before and after an instruction?
  - That is, what is the effect of instruction execution?
- Question 2: How does the information flow between successor and predecessor basic blocks?
  - That is, what is the effect of control flow?

- We will answer these questions for the live variables problem.

Live Variables

- A statement is a ***definition*** of a variable v if it may write to v.
- A statement is a ***use*** of variable v if it may read from v.
- A variable v is ***live*** at a point p in a CFG if there is a path from p to a use of v, and that path does not contain a (re-)definition of v.
- Computing liveness
  - Write down a system of equations that define live variable sets at each point in the CFG.
  - Solve the system iteratively.



Live Variable Sets: Prerequisites

- Compute $\mathrm{use}(I)$ and $\mathrm{def}(I)$ sets for an instruction I based on its structure.
  - $I \rightarrow x = y$ **binop** z: $\mathrm{use}(I) = \{y, z\}$; $\mathrm{def}(I) = \{x\}$.
  - $I \rightarrow x =$ **unop** y: $\mathrm{use}(I) = \{y\}$; $\mathrm{def}(I) = \{x\}$.
  - $I \rightarrow x = y$: $\mathrm{use}(I) = \{y\}$; $\mathrm{def}(I) = \{x\}$.
  - $I \rightarrow x = f(y_1, \dots, y_n)$: $\mathrm{use}(I) = \{y_1, \dots, y_n\}$; $\mathrm{def}(I) = \{x\}$.
  - $I \rightarrow$ **if**(x): $\mathrm{use}(I) = \{x\}$; $\mathrm{def}(I) = \emptyset$.
  - $I \rightarrow$ **return** x: $\mathrm{use}(I) = \{x\}$; $\mathrm{def}(I) = \emptyset$.
- For instruction I, let:
  - $In(I)$ = the set of live variables at the program point before I.
  - $Out(I)$ = the set of live variables at the program point after I.
- For basic block B, let:
  - $In(B)$ = the set of live variables at the program point before B.
  - $Out(B)$ = the set of live variables at the program point after B.
- How are the In, Out, use, and def sets at various program points related to one another?

Live Variable Sets: Part 1

- Answer Question 1
  - What is the relation between sets of live variables before and after an instruction?
- Examples

conclude	in[I] = {y,z}	in[I] = {y,z,t}	in[I] = {x,t}
	x = y+z;	x = y+z;	x = x+1;
assume	out[I] = {z}	out[I] = {x,t}	out[I] = {x,t}

- General rule

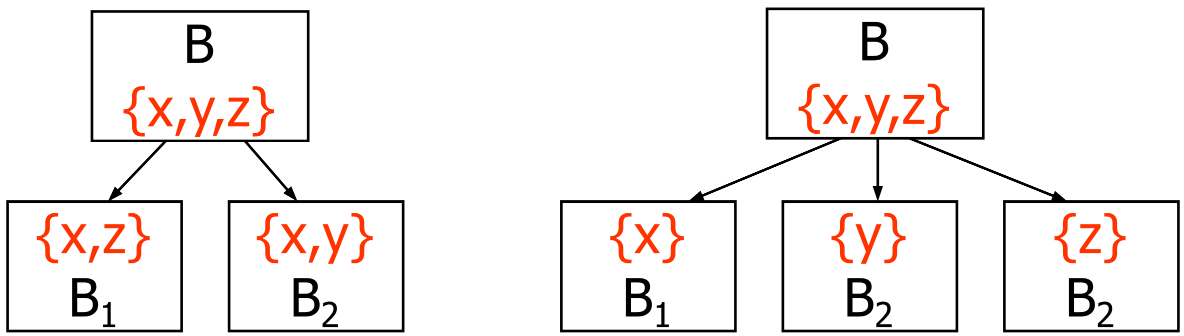
$$In(I) = \left(Out(I) \setminus \mathrm{def}(I)\right) \cup \mathrm{use}(I)$$

- Backward flow of information.
  - Given $Out(B)$, can compute $In(B)$.

```
In(B)
x = y+1
y =2*z
if (d)
Out(B)
```

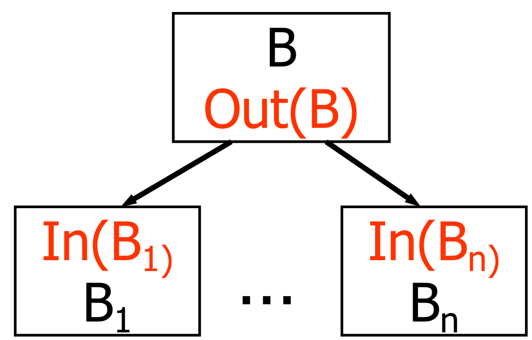
Live Variable Sets: Part 2

- Answer Question 2
  - For basic block B with successor blocks $B_1, \ldots, B_n$, what is the relation between $Out(B)$ and $In(B_1), \ldots, In(B_n)$?
- Examples



- General rule

$$Out(B) = \bigcup_{B' \in \mathrm{succ}(B)} In(B')$$



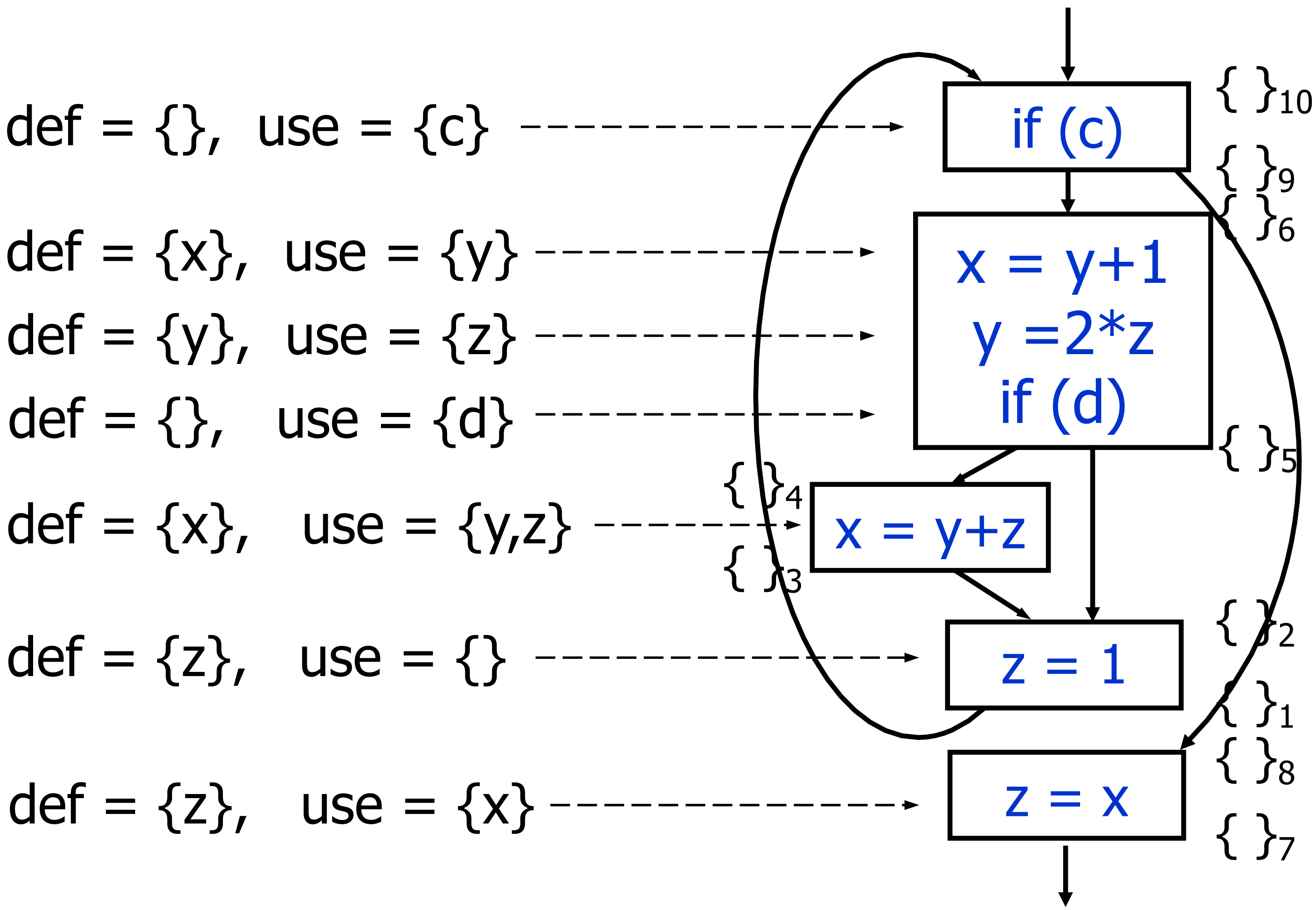
- Backward flow of information.
  - Given all the $In(B')$s, can compute $Out(B)$.

Liveness Analysis Algorithm

- Given the CFG, assemble the full system of equations.
  - Compute use and def sets for each instruction I and each basic block B.
  - For each instruction I:
  $$In(I) = \big(Out(I) \setminus \mathrm{def}(I)\big) \cup \mathrm{use}(I).$$
  - For each basic block B:
  $$Out(B) = \bigcup_{B' \in \mathrm{succ}(B)} In(B').$$

- Now solve this system iteratively.
  - Initialize all live variable sets to $\emptyset$.
  - Apply the constraints to calculate new values for sets.
    - That is, select an instruction I or basic block B for which the corresponding equation is not satisfied, and update $In(I)$ or $Out(B)$ accordingly.
  - Stop when we reach a fixed point (i.e., sets don't change any more, i.e., all equations have been satisfied).

Example: Initial Configuration



Example: Configuration At Convergence