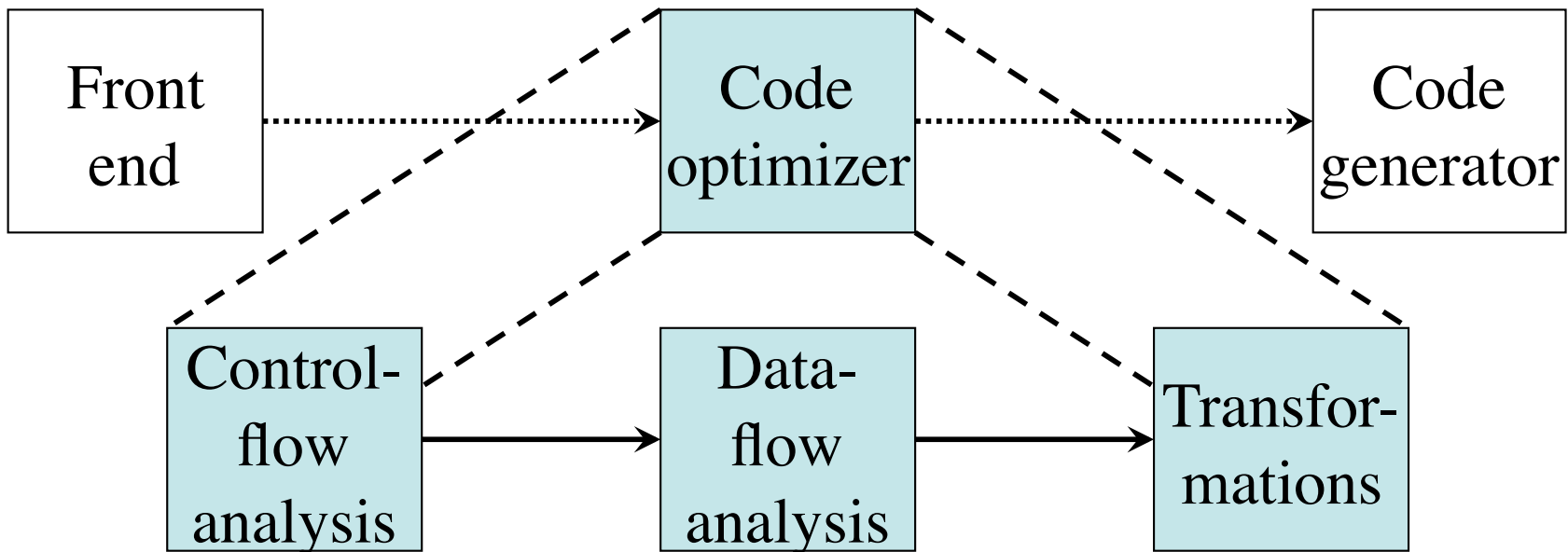


Code Optimization

Chapter 9
(1st ed. Ch.10)

The Code Optimizer

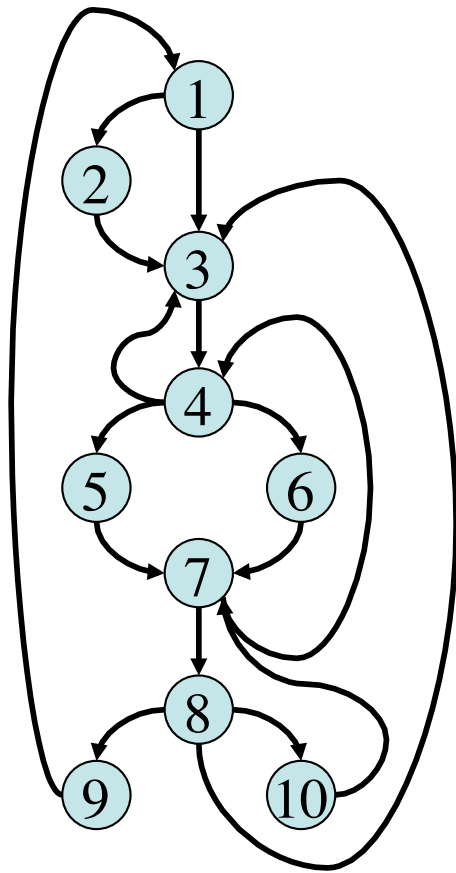
- Control flow analysis: control flow graph
- Data-flow analysis
- Transformations



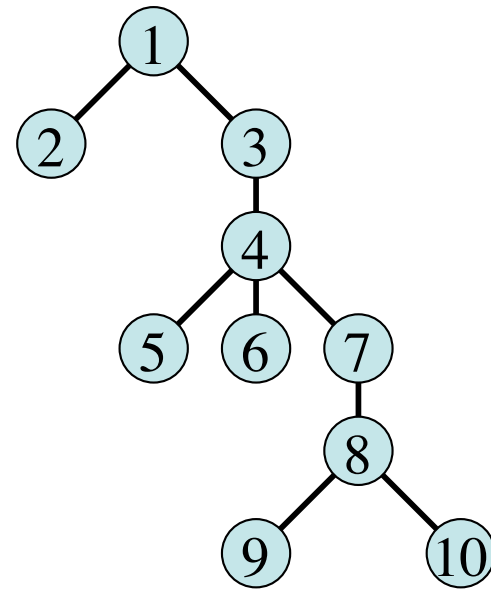
Determining Loops in Flow Graphs: Dominators

- Dominators: $d \text{ dom } n$
 - Node d of a CFG *dominates* node n if *every* path from the initial node of the CFG to n goes through d
 - The loop entry dominates all nodes in the loop
- The *immediate dominator* m of a node n is the last dominator on the path from the initial node to n
 - If $d \neq n$ and $d \text{ dom } n$ then $d \text{ dom } m$

Dominator Trees



CFG



Dominator tree

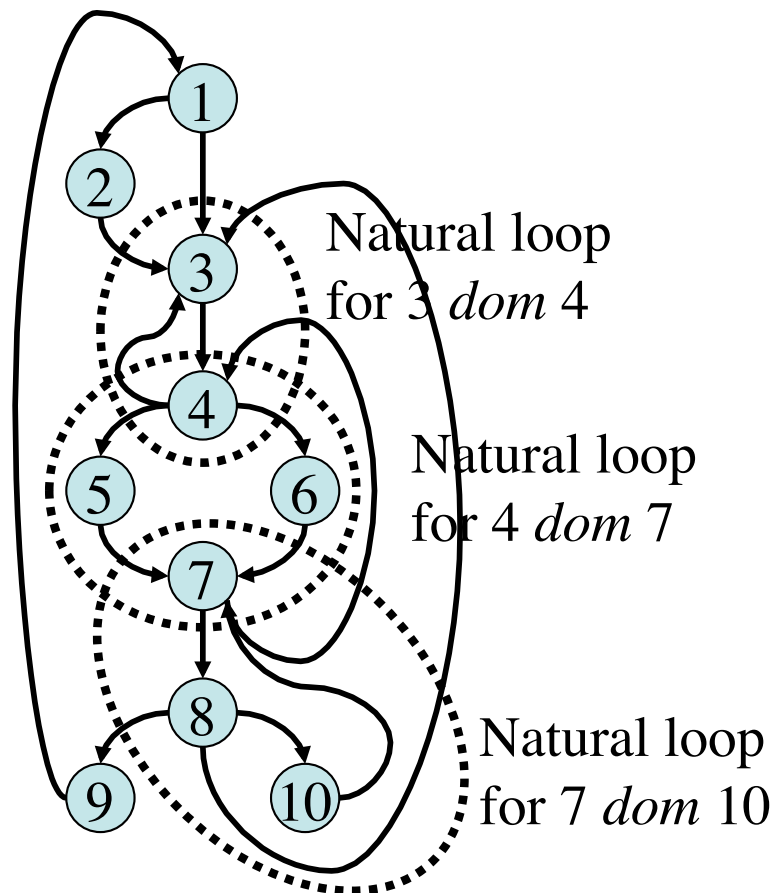
Natural Loops

- A *back edge* is an edge $a \rightarrow b$ whose head b dominates its tail a
- Given a back edge $n \rightarrow d$
 - The *natural loop* consists of d plus the nodes that can reach n without going through d
 - The *loop header* is node d
- In other words
 - A *natural loop* must have a single-entry node d
 - There must be a back edge that enters node d

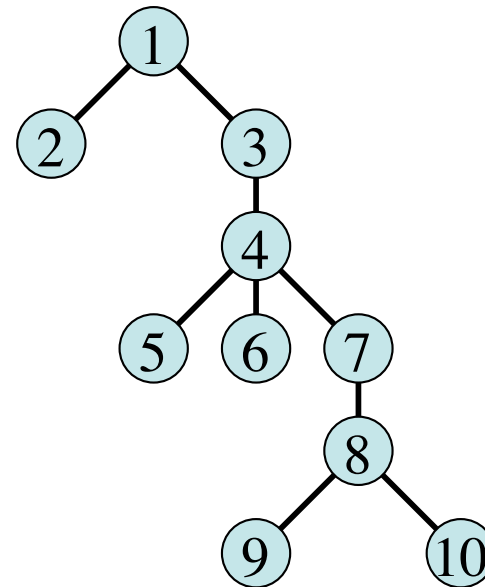
Natural Inner/Outer Loops

- Unless two loops have the same header, they are disjoint or one is nested within the other
- A nested loop is an *inner loop* if it contains no other loops
- A loop is an *outer loop* if it is not contained within another loop

Natural Inner Loops Example



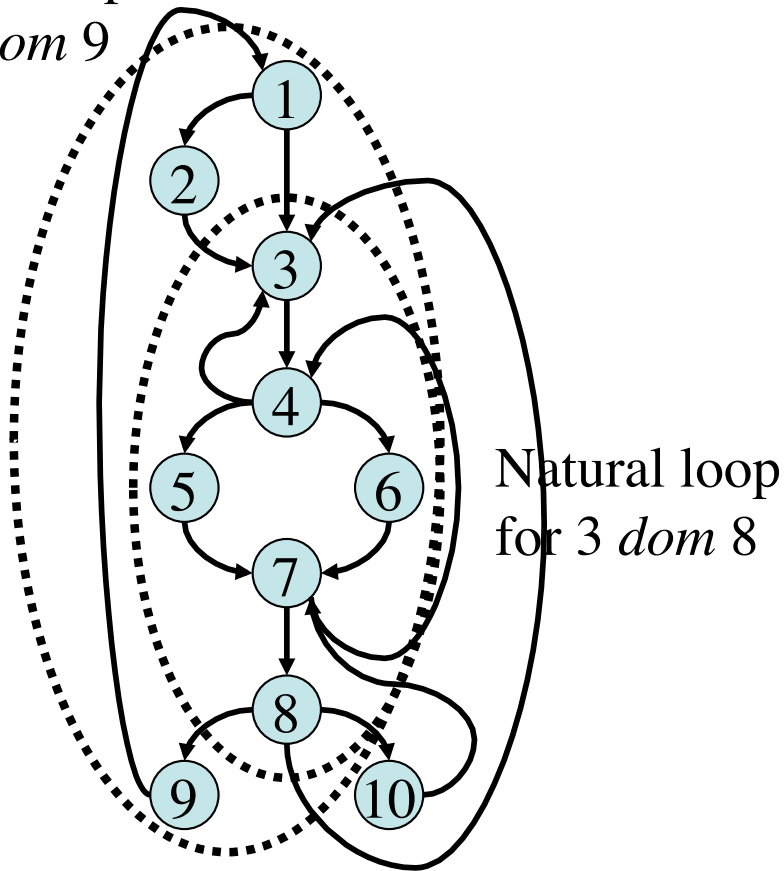
CFG



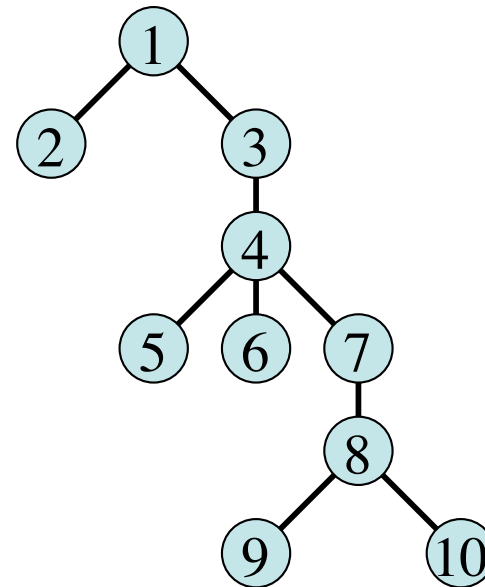
Dominator tree

Natural Outer Loops Example

Natural loop
for 1 *dom* 9



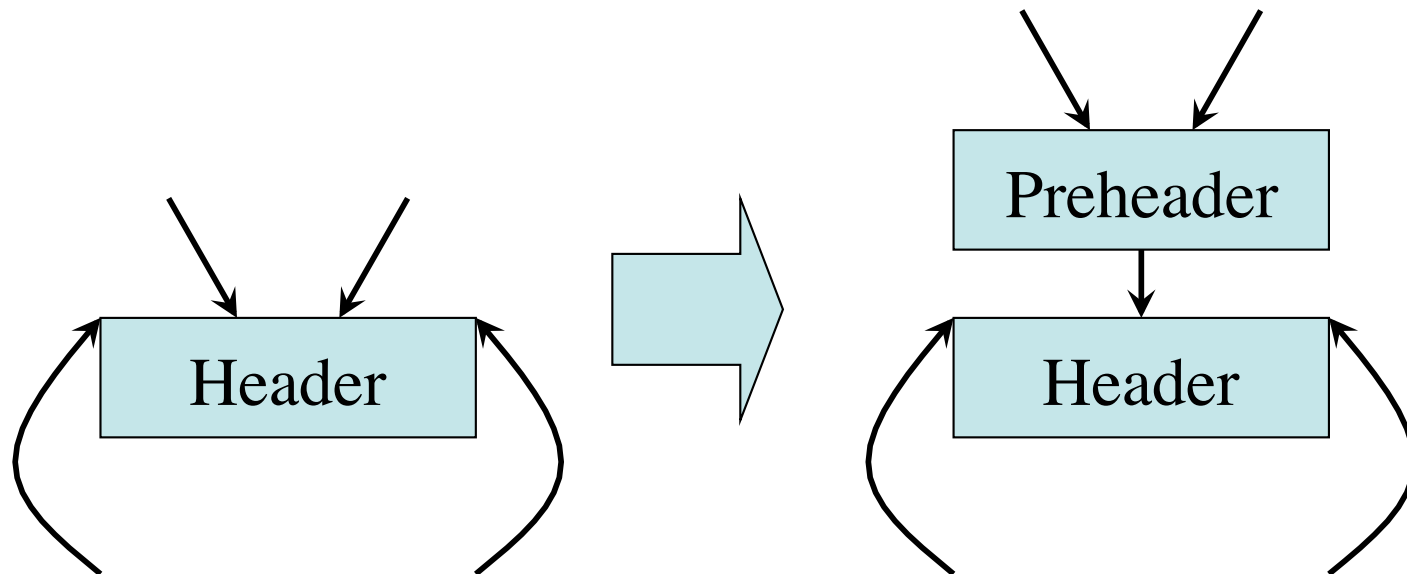
CFG



Dominator tree

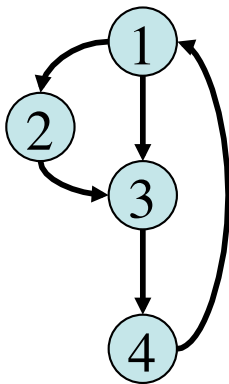
Pre-Headers

- To facilitate loop transformations, a compiler often adds a *preheader* to a loop
- Code motion, strength reduction, and other loop transformations populate the preheader

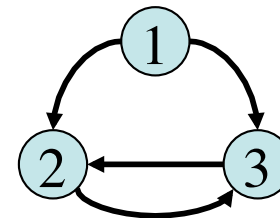


Reducible Flow Graphs

- *Reducible graph* = disjoint partition in forward and back edges such that the forward edges form an acyclic (sub)graph



Example of a
reducible CFG



Example of a
nonreducible CFG

Global Data-Flow Analysis

- To apply global optimizations on basic blocks, *data-flow information* is collected by solving systems of *data-flow equations*
- Suppose we need to determine the *live-variables* for a sequence of statements S

$$in[S] = use[S] \cup (out[S] - def[S])$$

B1: $\begin{array}{l} i := m-1 \\ j := n \end{array}$



B2: $j := j-1$



B3: $:= i+j$

Solution:

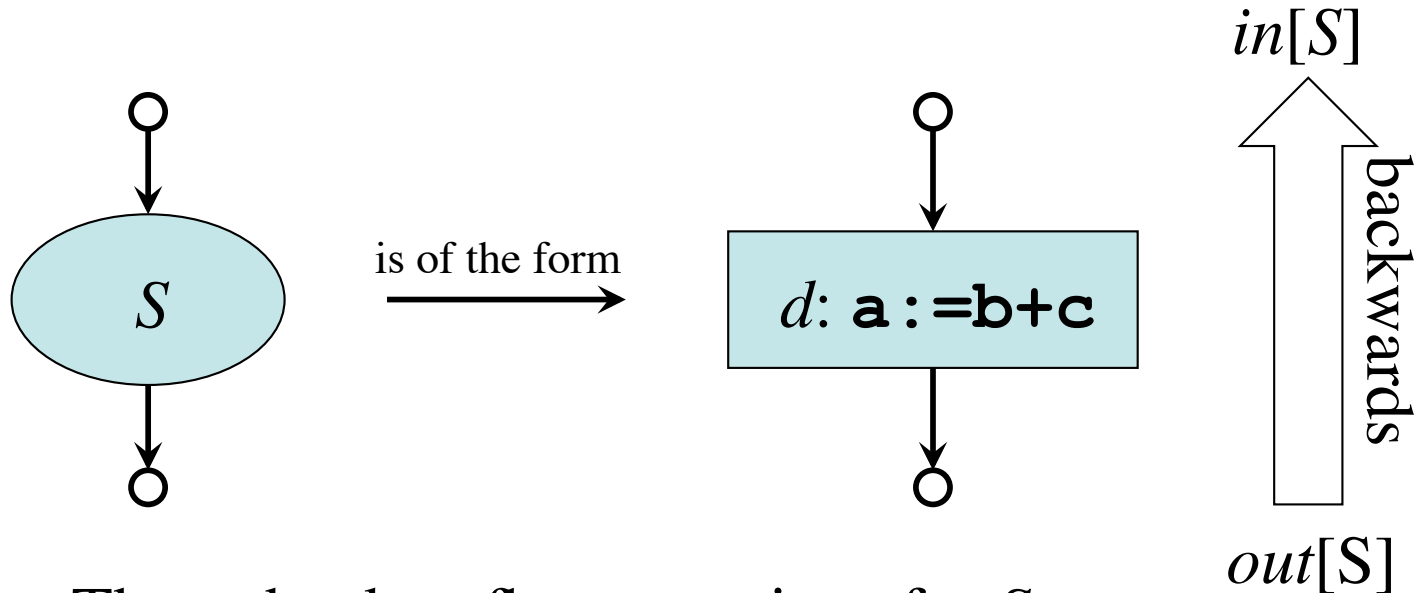
$$in[B1] = \{m, n\} \cup (\{i, j\} - \{i, j\}) = \{m, n\}$$

$$out[B1] = in[B2] = \{i, j\}$$

$$in[B2] = \{j\} \cup (\{i, j\} - \{j\}) = \{j\}$$

$$out[B2] = in[B3] = \{i, j\}$$

Live-Variable Analysis



applies
transfer function:

$$f_{[S]}(x) = use_{[S]} \cup (x - def_{[S]})$$

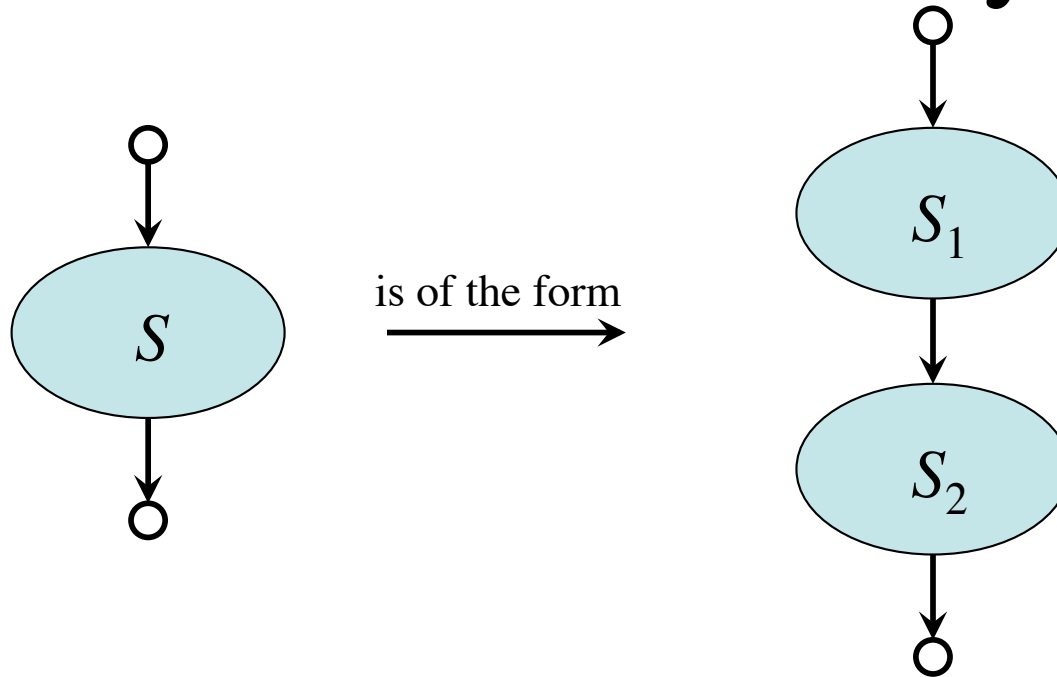
Then, the data-flow equations for S are:

$$use[S] = \{\mathbf{b}, \mathbf{c}\}$$

$$def[S] = \{\mathbf{a}\}$$

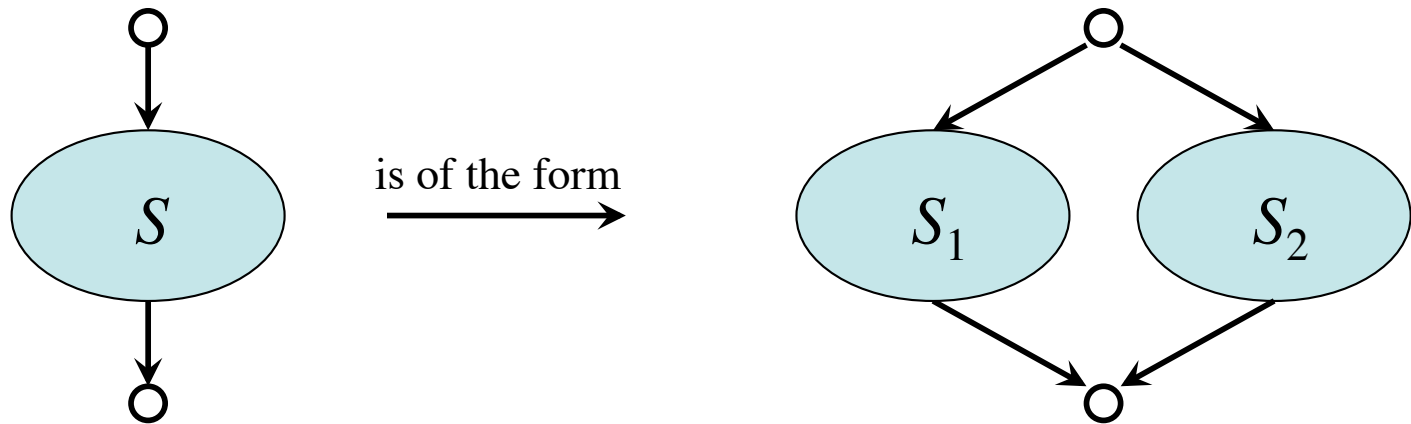
$$in[S] = use[S] \cup (out[S] - def[S])$$

Live-Variable Analysis



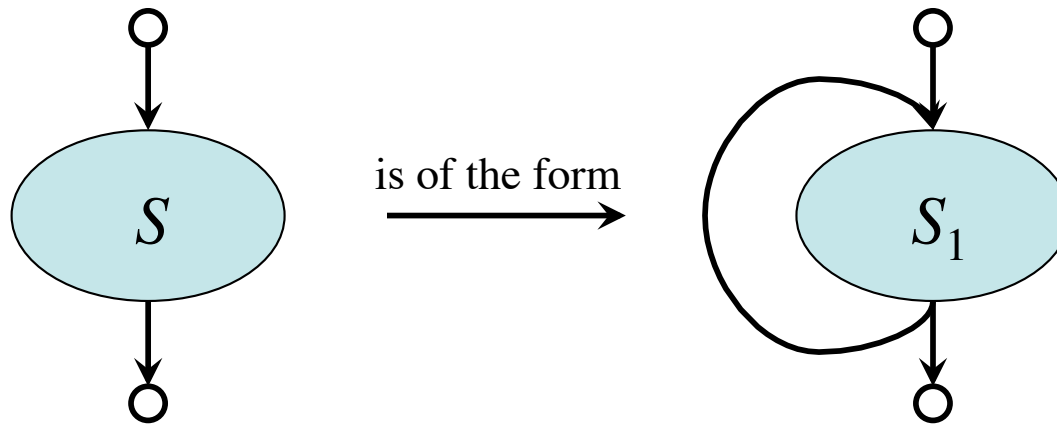
$$\begin{array}{l}
 in[S] \\
 in[S_1] \\
 out[S_1] \\
 in[S_2] \\
 out[S_2]
 \end{array}
 \quad
 \begin{array}{l}
 = in[S_1] \\
 = use[S_1] \cup (out[S_1] - def[S_1]) \\
 = in[S_2] \\
 = use[S_2] \cup (out[S_2] - def[S_2]) \\
 = out[S]
 \end{array}$$

Live-Variable Analysis



$$\begin{array}{l}
 in[S] \\
 in[S_1] \\
 in[S_2] \\
 out[S_1] \\
 out[S_2]
 \end{array}
 \quad
 \begin{array}{l}
 = in[S_1] \cup in[S_2] \\
 = use[S_1] \cup (out[S_1] - def[S_1]) \\
 = use[S_2] \cup (out[S_2] - def[S_2]) \\
 = out[S] \\
 = out[S]
 \end{array}$$

Live-Variable Analysis



$$\begin{array}{l}
 in[S] \\
 in[S_1] \\
 out[S_1]
 \end{array}
 \quad
 \begin{array}{l}
 = in[S_1] \\
 = use[S_1] \cup (out[S_1] - def[S_1]) \\
 = out[S] \cup use[S_1] \quad \text{non-iterative solution}
 \end{array}$$

Global Data-Flow Analysis

- Suppose we need to determine the *reaching definitions* for a sequence of statements S

$$out[S] = gen[S] \cup (in[S] - kill[S])$$

B1: $d1: i := m-1$
 $d2: j := n$



B2: $d3: j := j-1$



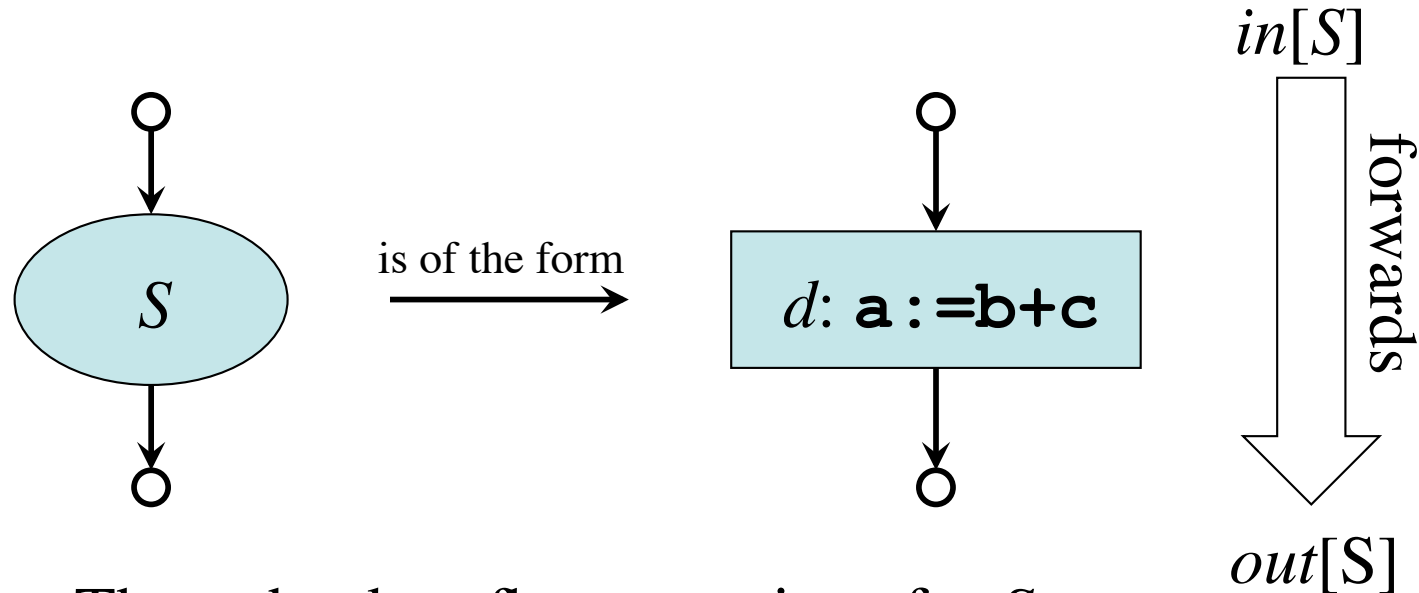
B3:

$$out[B1] = gen[B1] = \{d1, d2\}$$

$$out[B2] = gen[B2] \cup \{d1\} = \{d1, d3\}$$

$d1$ reaches B2 and B3 and
 $d2$ reaches B2, but not B3
 because $d2$ is killed in B2

Reaching Definitions



applies
transfer function:

$$f_{[S]}(x) = gen_{[S]} \cup (x - kill_{[S]})$$

Then, the data-flow equations for S are:

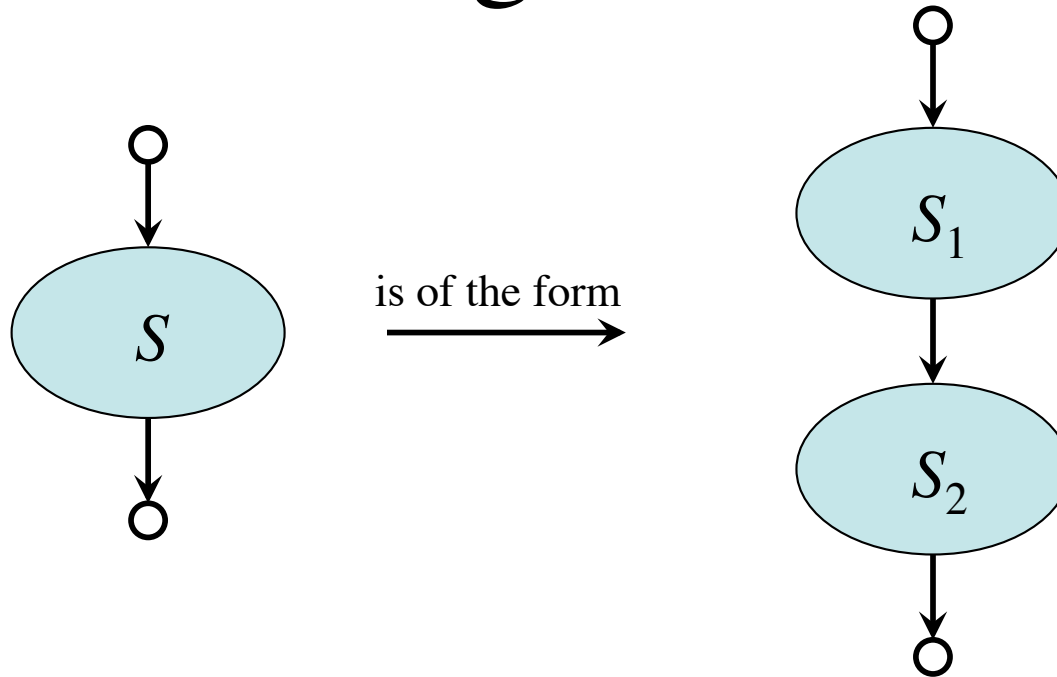
$$gen[S] = \{d\}$$

$$kill[S] = D_a - \{d\}$$

$$out[S] = gen[S] \cup (in[S] - kill[S])$$

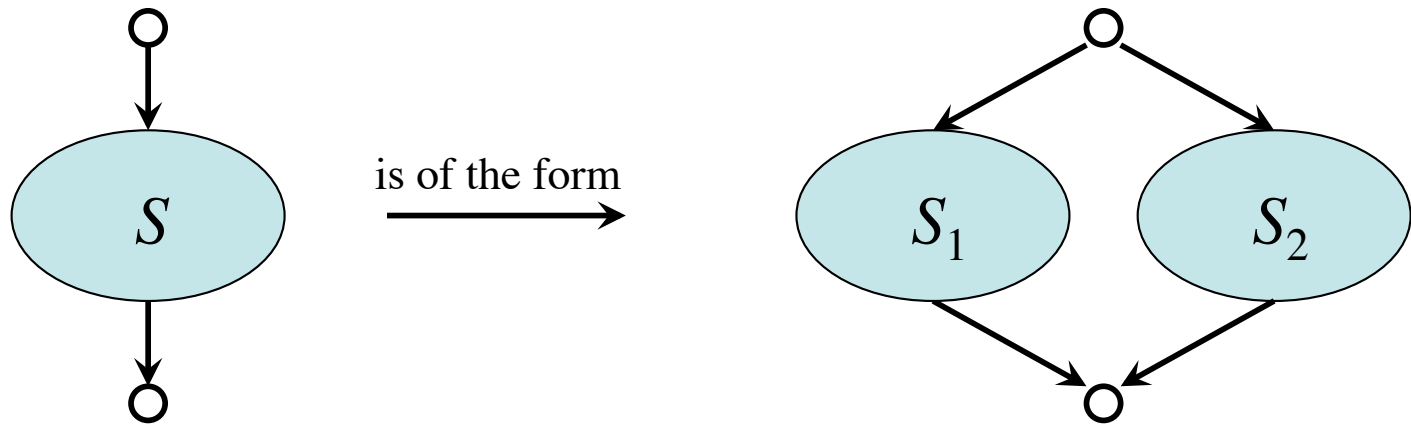
where $D_a =$ all definitions of \mathbf{a} in the region of code

Reaching Definitions



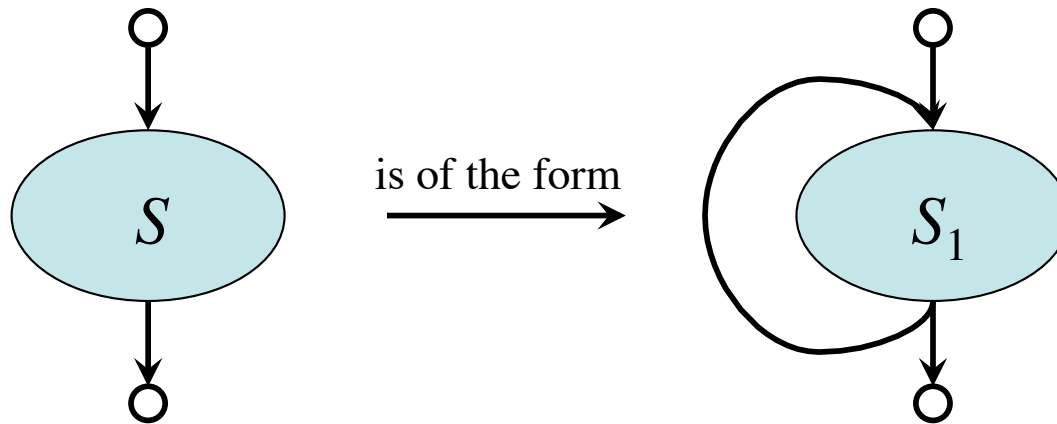
$$\begin{array}{l}
 \mathit{gen}[S] \\
 \mathit{kill}[S] \\
 \mathit{in}[S_1] \\
 \mathit{in}[S_2] \\
 \mathit{out}[S]
 \end{array}
 =
 \begin{array}{l}
 \mathit{gen}[S_2] \cup (\mathit{gen}[S_1] - \mathit{kill}[S_2]) \\
 \mathit{kill}[S_2] \cup (\mathit{kill}[S_1] - \mathit{gen}[S_2]) \\
 \mathit{in}[S] \\
 \mathit{out}[S_1] \\
 \mathit{out}[S_2]
 \end{array}$$

Reaching Definitions



$$\begin{array}{l}
 \mathit{gen}[S] \\
 \mathit{kill}[S] \\
 \mathit{in}[S_1] \\
 \mathit{in}[S_2] \\
 \mathit{out}[S]
 \end{array}
 =
 \begin{array}{l}
 \mathit{gen}[S_1] \cup \mathit{gen}[S_2] \\
 \mathit{kill}[S_1] \cap \mathit{kill}[S_2] \\
 \mathit{in}[S] \\
 \mathit{in}[S] \\
 \mathit{out}[S_1] \cup \mathit{out}[S_2]
 \end{array}$$

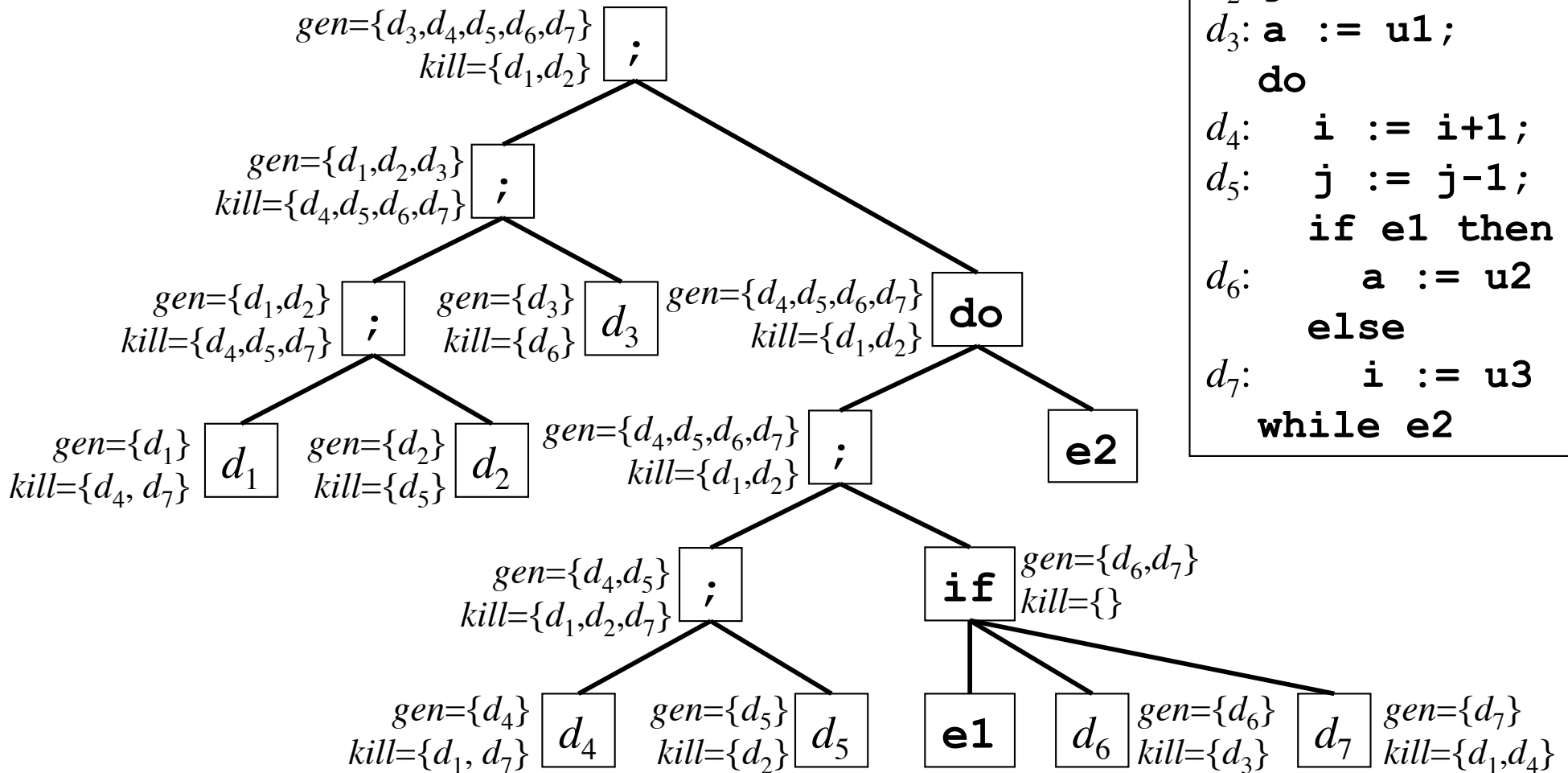
Reaching Definitions



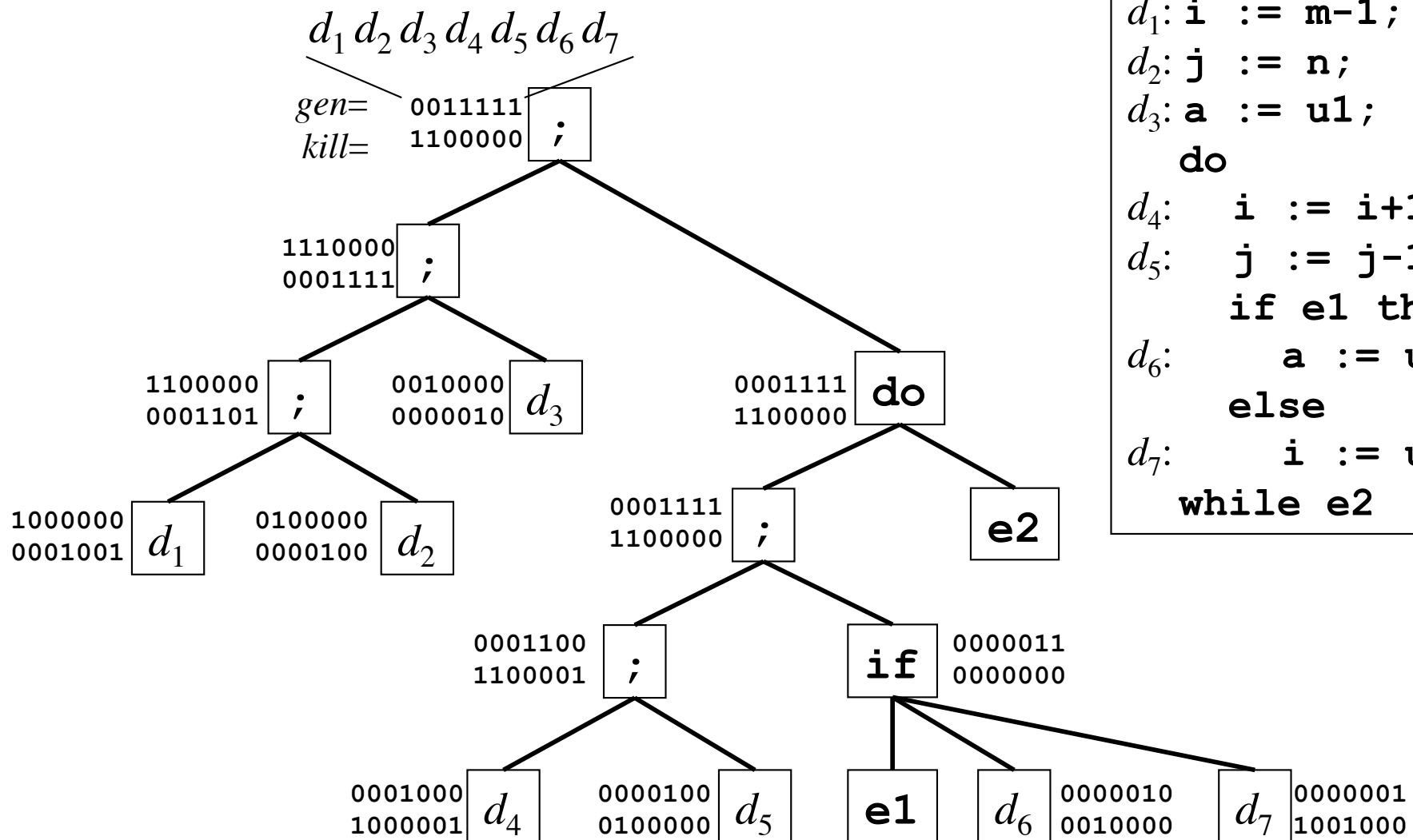
$$\begin{array}{ll}
 \mathit{gen}[S] & = \mathit{gen}[S_1] \\
 \mathit{kill}[S] & = \mathit{kill}[S_1] \\
 \mathit{in}[S_1] & = \mathit{in}[S] \cup \mathit{gen}[S_1] \\
 \mathit{out}[S] & = \mathit{out}[S_1]
 \end{array}
 \quad \text{non-iterative solution}$$

(see later)

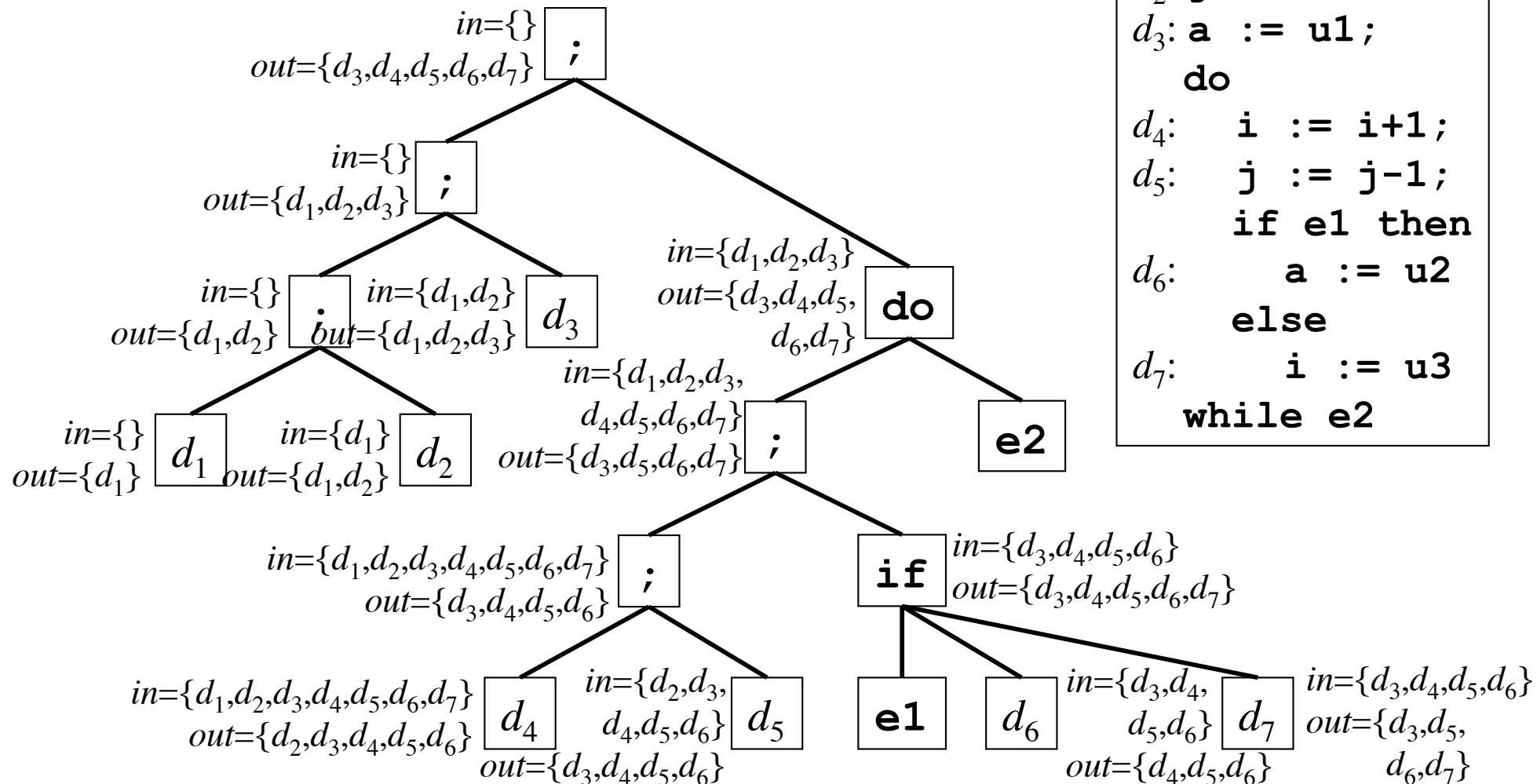
Reaching Definitions: Computing Gen/Kill



Using Bit-Vectors to Compute Reaching Definitions



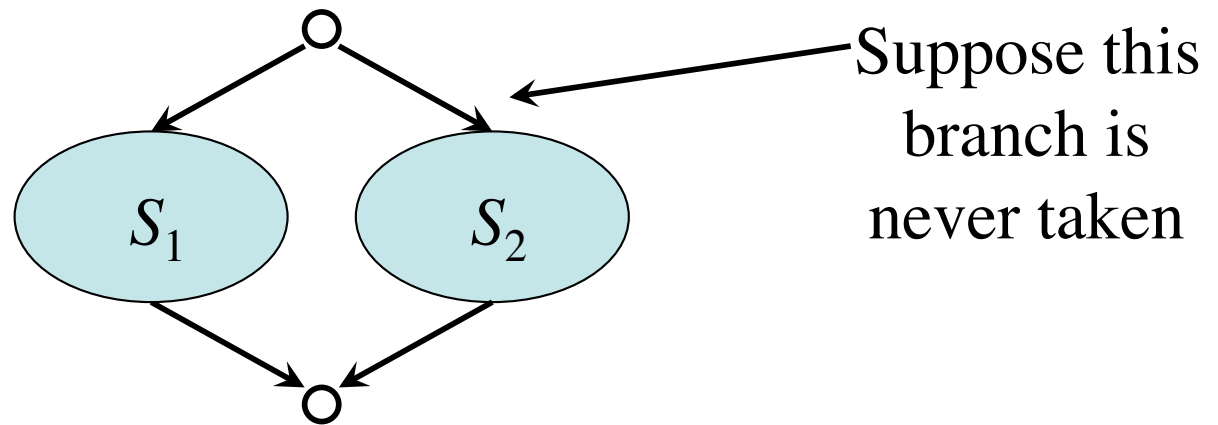
Reaching Definitions: Computing In/Out (non-iterative)



Accuracy, Safeness, and Conservative Estimations

- *Conservative*: refers to making safe assumptions when insufficient information is available at compile time, i.e. the compiler has to guarantee not to change the meaning of the optimized code
- *Safe*: refers to the fact that a superset of reaching definitions is safe (some may have been killed)
- *Accuracy*: more and better information enables more code optimizations

Reaching Definitions are a Conservative (Safe) Estimation



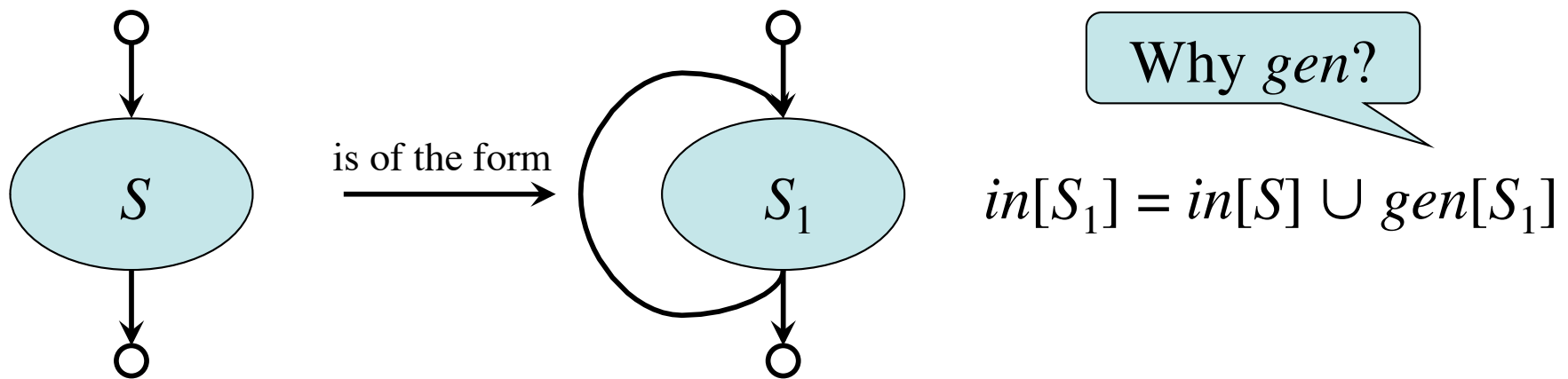
Estimation:

$$\begin{aligned} gen[S] &= gen[S_1] \cup gen[S_2] \\ kill[S] &= kill[S_1] \cap kill[S_2] \end{aligned}$$

Accurate:

$$\begin{aligned} gen'[S] &= gen[S_1] && \subseteq gen[S] \\ kill'[S] &= kill[S_1] && \supseteq kill[S] \end{aligned}$$

Reaching Definitions are a Conservative (Safe) Estimation



The problem is that

$$in[S_1] = in[S] \cup out[S_1]$$

makes more sense, but we cannot solve this directly (only iteratively), because $out[S_1]$ depends on $in[S_1]$

Reaching Definitions are a Conservative (Safe) Estimation

We have:

$$(1) \text{in}[S_1] = \text{in}[S] \cup \text{out}[S_1]$$

$$(2) \text{out}[S_1] = \text{gen}[S_1] \cup (\text{in}[S_1] - \text{kill}[S_1])$$

Solve $\text{in}[S_1]$ and $\text{out}[S_1]$ by estimating $\text{in}^1[S_1]$ using safe but approximate $\text{out}[S_1] = \emptyset$, then re-compute $\text{out}^1[S_1]$ using (2) to estimate $\text{in}^2[S_1]$, etc.

$$\text{in}^1[S_1] =_{(1)} \text{in}[S] \cup \text{out}[S_1] = \text{in}[S]$$

$$\text{out}^1[S_1] =_{(2)} \text{gen}[S_1] \cup (\text{in}^1[S_1] - \text{kill}[S_1]) = \text{gen}[S_1] \cup (\text{in}[S] - \text{kill}[S_1])$$

$$\text{in}^2[S_1] =_{(1)} \text{in}[S] \cup \text{out}^1[S_1] = \text{in}[S] \cup \text{gen}[S_1] \cup (\text{in}[S] - \text{kill}[S_1]) = \text{in}[S] \cup \text{gen}[S_1]$$

$$\begin{aligned} \text{out}^2[S_1] &=_{(2)} \text{gen}[S_1] \cup (\text{in}^2[S_1] - \text{kill}[S_1]) = \text{gen}[S_1] \cup (\text{in}[S] \cup \text{gen}[S_1] - \text{kill}[S_1]) \\ &= \text{gen}[S_1] \cup (\text{in}[S] - \text{kill}[S_1]) \end{aligned}$$

Because $\text{out}^1[S_1] = \text{out}^2[S_1]$, and therefore $\text{in}^3[S_1] = \text{in}^2[S_1]$, we conclude that

$$\text{in}[S_1] = \text{in}[S] \cup \text{gen}[S_1]$$

