Data-Flow Analysis

Global Data-Flow Analysis

- Knowledge about the behavior of a variable is essential for performing transformations.
- Control flow information is also required to do transformations across basic blocks.
- In addition, to apply global optimizations on basic blocks, data-flow information is collected by solving systems of data-flow equations

Data flow equations

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

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

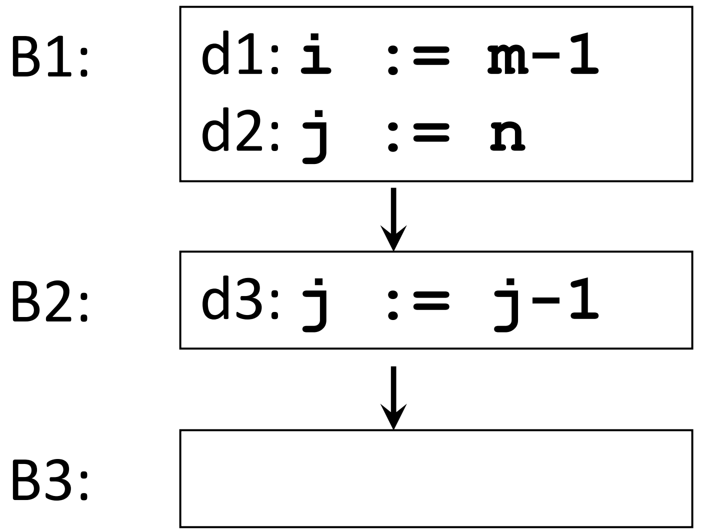
  - The information at the end of a statement S is either generated within the statement or enters at the beginning and is not killed as control flows through the statement.

Factors for setting up data-flow equations

- Notion of killing and generating depend on the desired information and on the data-flow analysis problem to be solved
  - Some problems out[S] need to be defined in terms of in[S] and for others in[S] need to be defined in terms of out[S]
- Data flow is interrupted by the control flow of the program.
  - Out[S] is based on the assumption that there is a unique end point
- Assignments through pointer variables, procedure calls, assignments to array variables influence the data flow

Point and Path

- Position between two adjacent statements and above the first and following the last is called as a point
  - B1 has 3 points
  - B2 has 2 points



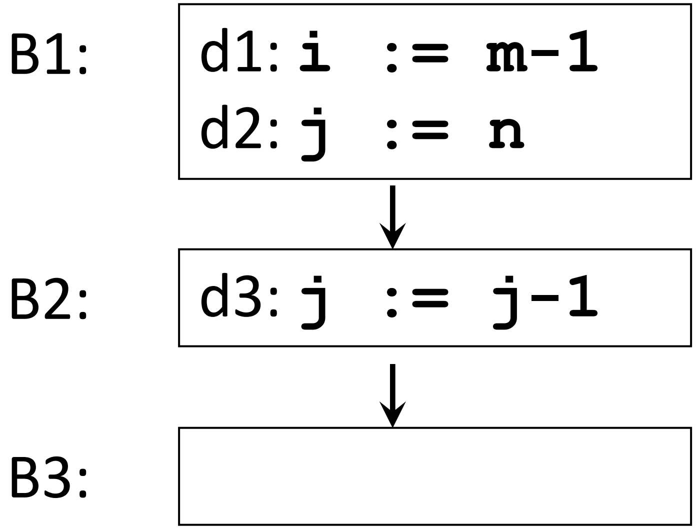
Points and Path

- Consider all the blocks and each will have many points. Merge the last point of a current block with the first point of its successor block.
- Path is the sequence of statements between any two points

Reaching Definitions

- A definition of a variable ‘x’ is a statement that assigns or may assign a value to ‘x’ – Unambiguous definition
- If x is used as a parameter of a procedure or through pointer. – Ambiguous definition
- ‘d’ reaches a point ‘p’ if there is a path from the point immediately following ‘d’ to ‘p’ and ‘d’ is not killed in that path
- ‘kill’ – b/w two points, where the variable is defined and its redefinition

Example



out[S] = gen[S] ∪ (in[S] - kill[S])

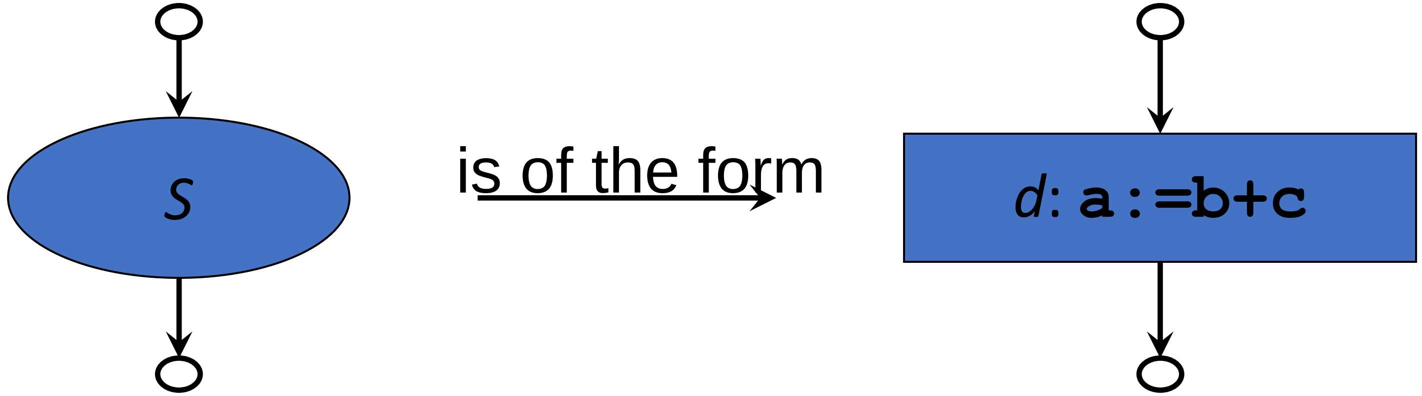
- *out*[B1] = *gen*[B1] = {d1, d2}
  out[B2] = *gen*[B2] ∪ {d1} = {d1, d3}
  d1 reaches B2 and B3 and
- d2 reaches B2, but not B3 because d2 is killed in B2

Data flow analysis – structured programs

- Assumption – single entry and single exit point
- S → id := E | S ; S | if E then S else S |

  do S while E
- E → id + id | id
- There is a unique header at which control begins

Reaching Definitions – S→ id:= E



Then, the data-flow equations for *S* are:

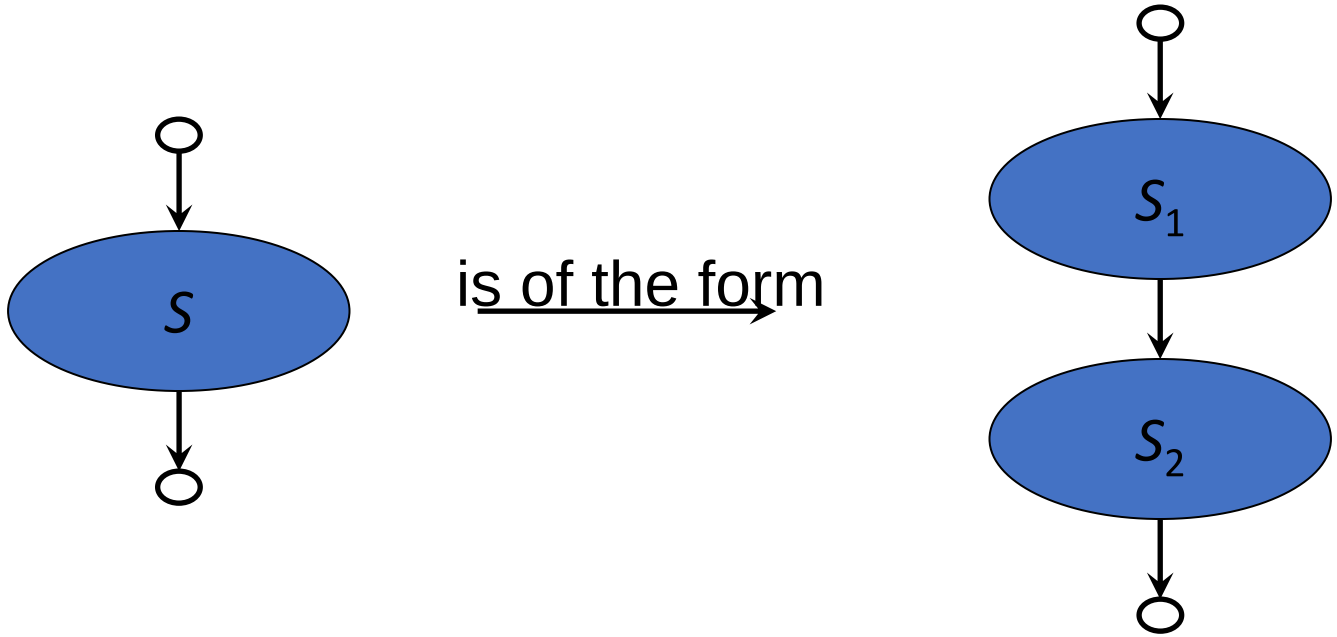
$gen[S]$ = $\{d\}$

$kill[S]$ = $D_{\mathbf{a}} - \{d\}$

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

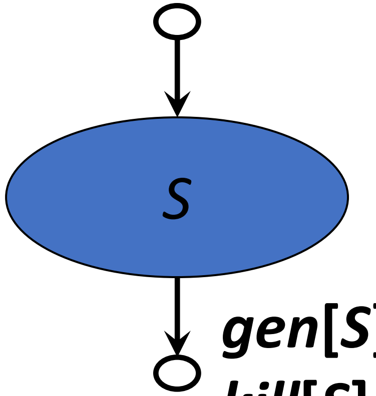
where $D_{\mathbf{a}}$ = all definitions of **a** in the region of code

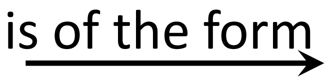
Reaching Definitions S → S1; S2

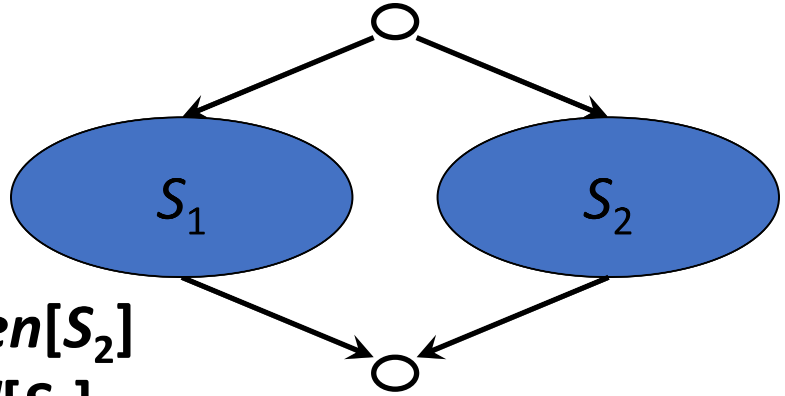


$$
\begin{aligned}
&\textbf{\textit{gen}}[S] && = \textbf{\textit{gen}}[S_2] \cup (\textbf{\textit{gen}}[S_1] - \textbf{\textit{kill}}[S_2]) \\
&\textbf{\textit{kill}}[S] && = \textbf{\textit{kill}}[S_2] \cup (\textbf{\textit{kill}}[S_1] - \textbf{\textit{gen}}[S_2]) \\
&\textbf{\textit{in}}[S_1] && = \textbf{\textit{in}}[S] \\
&\textbf{\textit{in}}[S_2] && = \textbf{\textit{out}}[S_1] \\
&\textbf{\textit{out}}[S] && = \textbf{\textit{out}}[S_2]
\end{aligned}
$$

Reaching Definitions - S → if E then S1 else S2

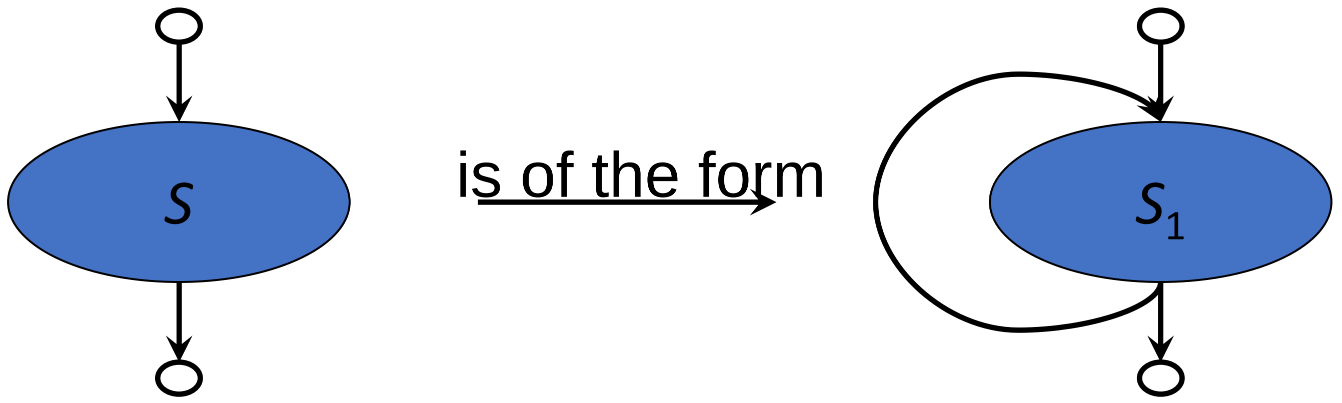






$$
\begin{aligned}
\textbf{\textit{gen}}[S] &= \textbf{\textit{gen}}[S_1] \cup \textbf{\textit{gen}}[S_2] \\
\textbf{\textit{kill}}[S] &= \textbf{\textit{kill}}[S_1] \cap \textbf{\textit{kill}}[S_2] \\
\textbf{\textit{in}}[S_1] &= \textbf{\textit{in}}[S] \\
\textbf{\textit{in}}[S_2] &= \textbf{\textit{in}}[S]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

Reaching Definitions – S → do S while E



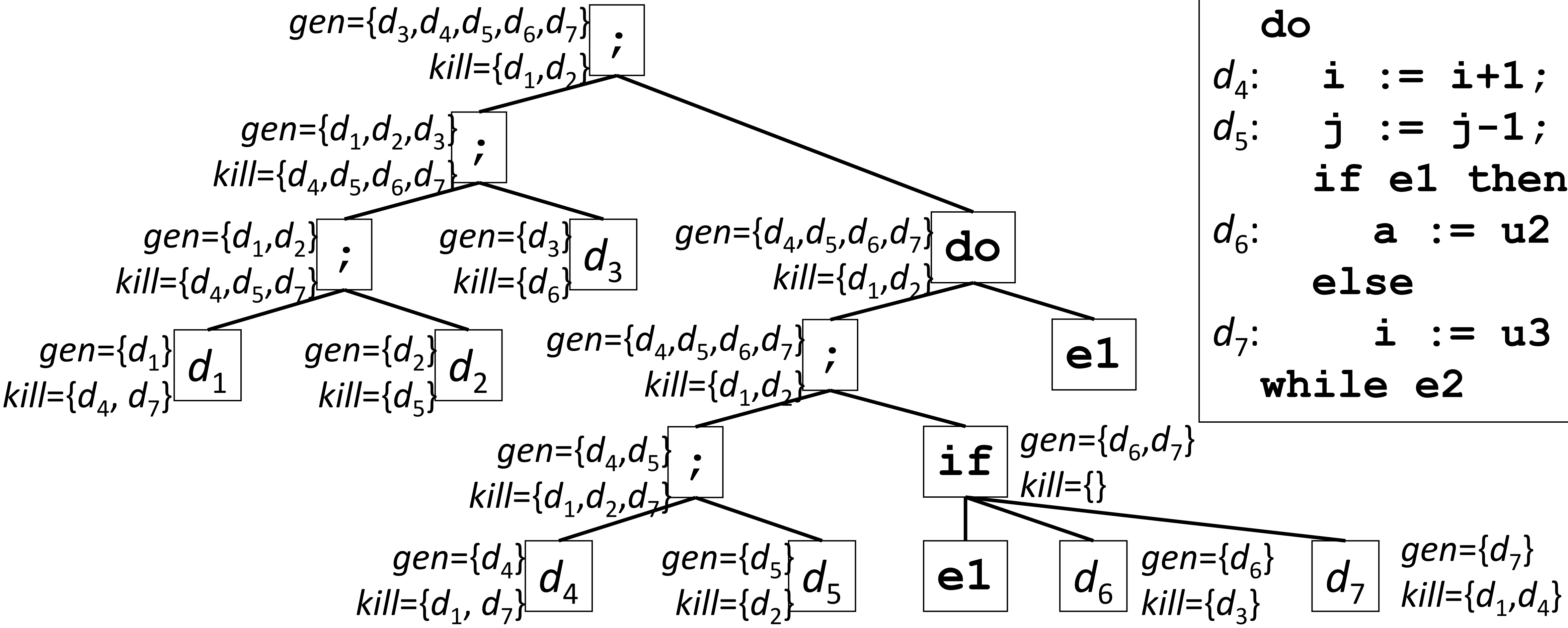
$$
\begin{aligned}
gen[S] &= gen[S_1] \\
kill[S] &= kill[S_1] \\
in[S_1] &= in[S] \cup gen[S_1] \\
out[S] &= out[S_1]
\end{aligned}
$$

Example

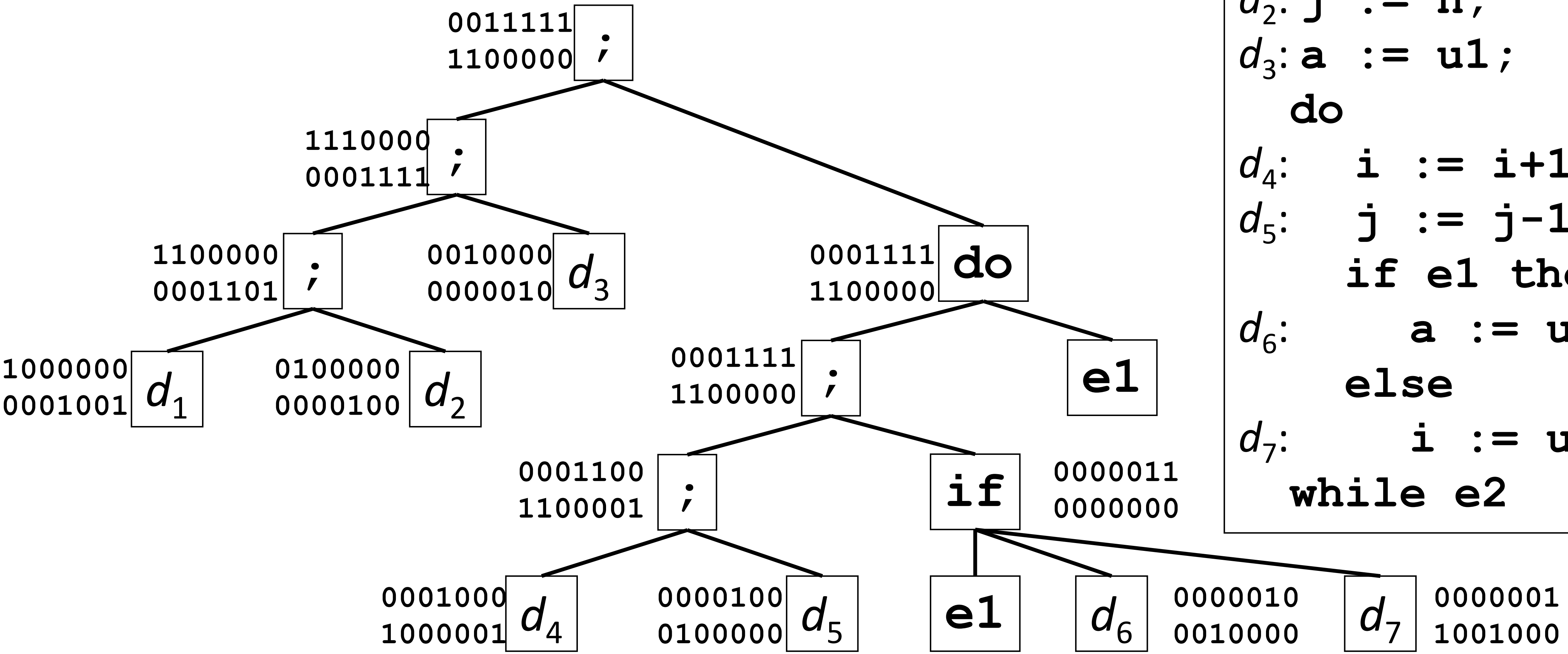
```
d1: i := m-1;
d2: j := n;
d3: a := u1;
    do
d4:   i := i+1;
d5:   j := j-1;
      if e1 then
d6:     a := u2
      else
d7:     i := u3
    while e2
```

Example Reaching Definitions

```
d1: i := m-1;
d2: j := n;
d3: a := u1;
  do
d4:   i := i+1;
d5:   j := j-1;
      if e1 then
d6:      a := u2
      else
d7:      i := u3
  while e2
```



Using Bit-Vectors to Compute Reaching Definitions



d_1: **`i := m-1;`**
d_2: **`j := n;`**
d_3: **`a := u1;`**
  `do`
d_4: **`i := i+1;`**
d_5: **`j := j-1;`**
    `if e1 then`
d_6: **`a := u2`**
    `else`
d_7: **`i := u3`**
  `while e2`

Accuracy, Safeness, and Conservative Estimations

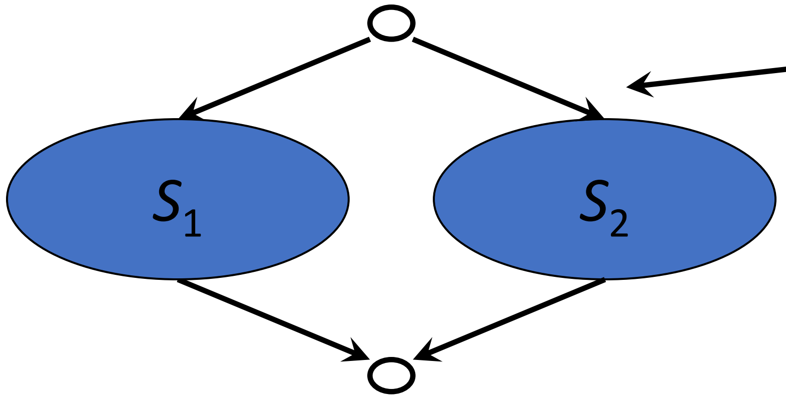
- Accuracy: the larger the superset of reaching definitions, the less information we have to apply code optimizations
- Safe: refers to the fact that a superset of reaching definitions is safe (some may be have been killed)
- Conservative: refers to making safe assumptions when insufficient information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

Reaching Definitions - Conservative (Safe) Estimation

- Assumption is that conditional expressions are uninterrupted, they will have one branch or the other
- The path of the flow graph is also the execution path

Reaching Definitions - Conservative (Safe) Estimation

If E is always true then S2 is not taken



Suppose this branch is never taken

Estimation:

$gen[S]$ $= gen[S_1] \cup gen[S_2]$ – may not be contributed by S2

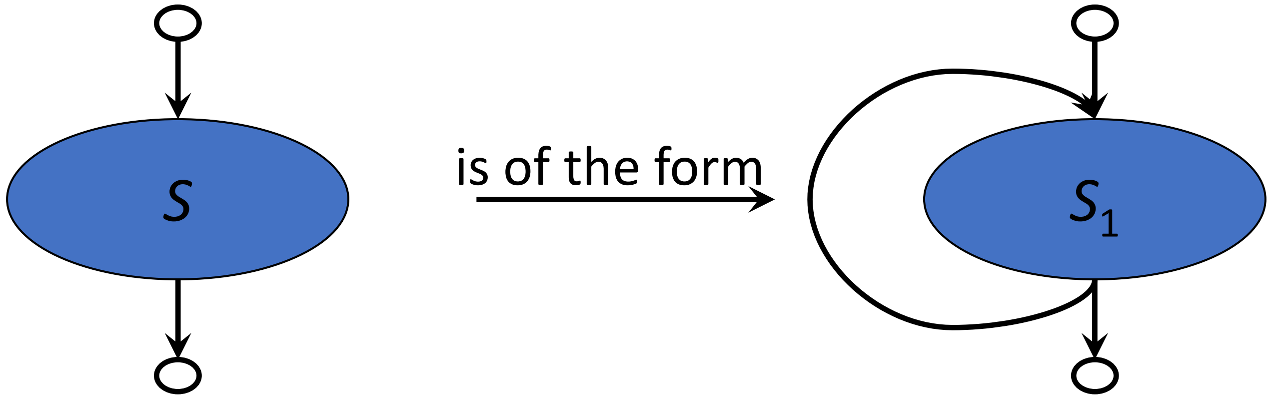
$kill[S]$ $= kill[S_1] \cap kill[S_2]$ – may not be contributed by S2

Accurate:

$gen'[S]$ $= gen[S_1]$ $\subseteq gen[S]$

$kill'[S]$ $= kill[S_1]$ $\supseteq kill[S]$

Reaching Definitions - Conservative (Safe) Estimation

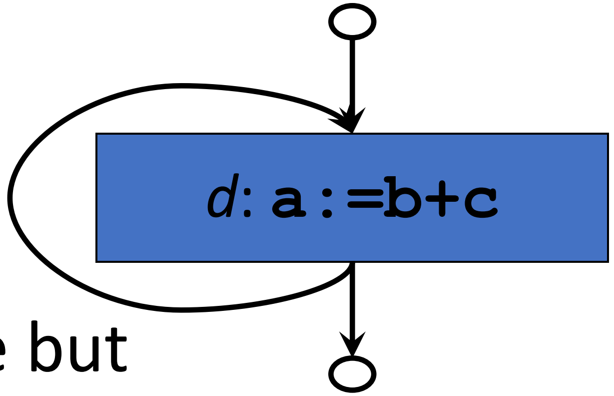


$in[S_1] = in[S] \cup gen[S_1]$

The problem is that
$in[S_1] = in[S] \cup out[S_1]$

but we cannot solve this directly, because $out[S_1]$ depends on $in[S_1]$

Reaching Definitions are a Conservative (Safe) Estimation



We have:

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

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

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

Reaching Definitions are a Conservative (Safe) Estimation

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

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

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