# Reaching Definition Analysis

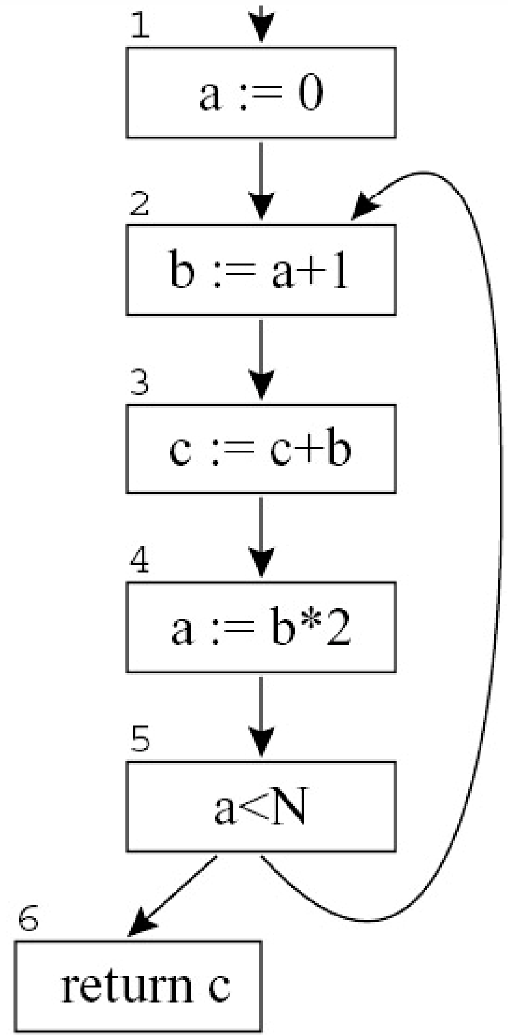
CS502

- The most fundamental dataflow information concerns the *definition* (or generation) and the *use* of values.
- When an operation modifies a variable at program point *p*, we say it defines the variable's value at *p*.
  - So, the operation is also called a *definition* of the variable.
  - We can number the definitions by d1, d2, … etc.
- Similarly, when an operation makes a read reference to a variable at program point *p*, we say it uses the variable's value at *p*.
  - The operation is also called a *use* of the variable.

- At run time, there is a unique write operation which generates the value used by a read operation.
- In the program, however, a read reference may represent many run-time read operations and therefore may use values defined by more than one write reference.
- Moreover, for different runs of the same program, a read reference may use values defined by different write references.

Example: which defs of a may provide values used by instruction 2?

$$
\begin{aligned}
& a \leftarrow 0 \\
L_1 :\ & b \leftarrow a + 1 \\
& c \leftarrow c + b \\
& a \leftarrow b * 2 \\
& \text{if } a < N \text{ goto } L_1 \\
& \text{return } c
\end{aligned}
$$



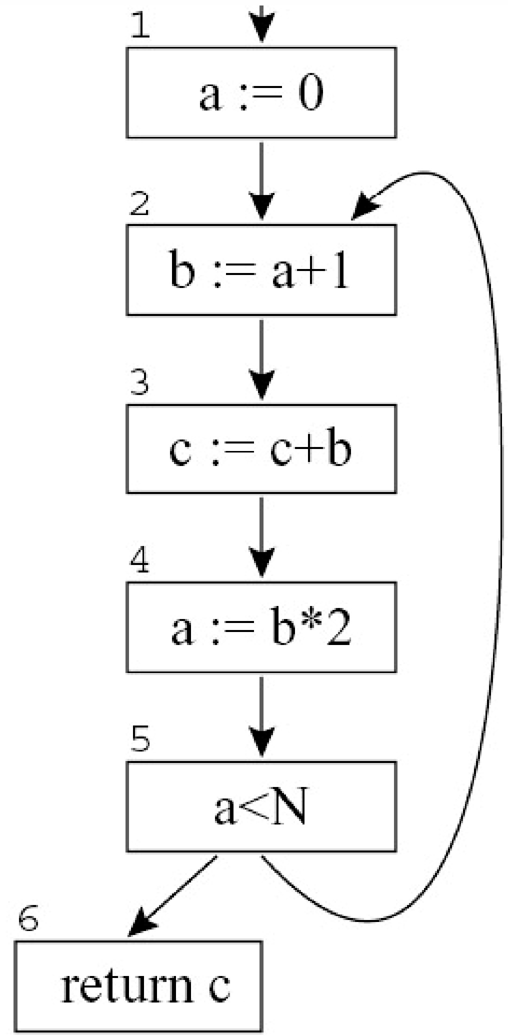
- If a value written by def1 is overwritten by def2, then def1 is *killed* by def2.
- If there exist an execution path from *def1* to a program point *p* such that *def1* is not killed by any defs, then we say def1 *reaches* p.
  - We also say that def1 belongs to the set of reaching definitions of p.
  - Similarly, we can define the reaching definitions of a basic block B.
  - And, we can define the reaching definitions of a use

# Conservative analysis

- If the compiler is uncertain whether a def is killed by another, then a “conservative” assumption needs to be made.
- For most purposes, the conservative assumption is to assume that the def is not killed.
- Therefore, a conservative analysis may over-estimate the set of reaching definitions.
- Let us inspect the previous example and find reaching definitions for each instruction.

Example: who are the reaching defs for each instruction?

$$
\begin{aligned}
& a \leftarrow 0 \\
L_1 :\ & b \leftarrow a + 1 \\
& c \leftarrow c + b \\
& a \leftarrow b * 2 \\
& \text{if } a < N \text{ goto } L_1 \\
& \text{return } c
\end{aligned}
$$



# Algorithm for computing reaching defs

For each basic block *B*, we want to determine

- REACHin(B): the set of defs reaching the entry of B
- REACHout(B): the set of defs reaching the exit of B
- To compute these, we use two pieces of information from *B*:
  - GEN(B): the set of defs which appear in B and may reach the exit of B.
  - KILL(B): the set of defs (in the whole program scope being analyzed) whose variables are definitely redefined in B.

# Basic Equations

$$REACHout(B) = GEN(B) \cup (REACHin(B) - KILL(B)) \qquad (1)$$

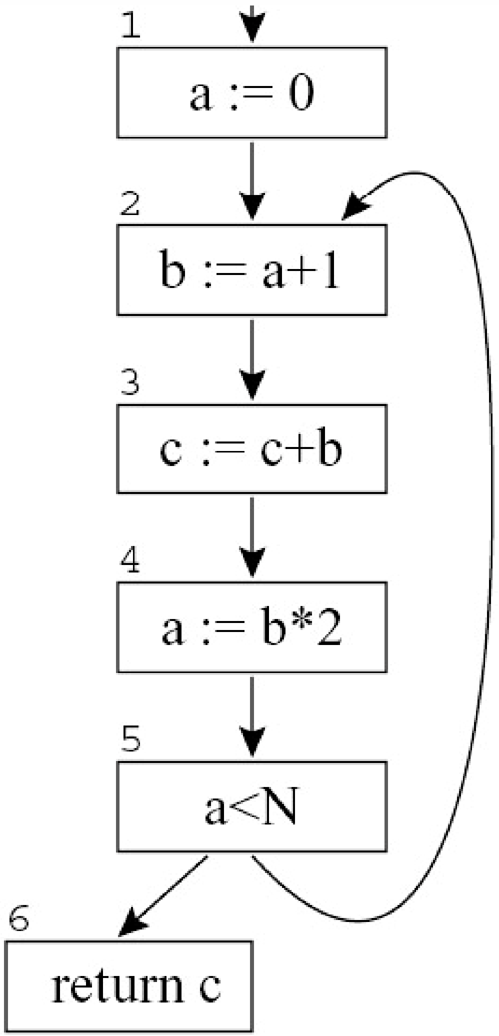
$$REACHin(B) = \bigcup_{j \in Pred(B)} REACHout(j) \qquad (2)$$

- As in liveness analysis, we can solve this iteratively
- GEN(B) and KILL(B) are “constants”
- Initialize all REACHin and REACHout variables to empty sets.

- The two equations become two operators (or two filters) that are monotonically non-decreasing for each B
- The variables are bounded from above ➔ The iterations must converge, reaching the “fixed point”, regardless in which order to apply the operators in each iteration
- However, some order makes the convergence faster

Example: who are the reaching defs for each instruction?

$$
\begin{aligned}
& a \leftarrow 0 \\
L_1 :\ & b \leftarrow a + 1 \\
& c \leftarrow c + b \\
& a \leftarrow b * 2 \\
& \text{if } a < N \text{ goto } L_1 \\
& \text{return } c
\end{aligned}
$$



# Algorithm Using a Work List

- In the naïve implementation, the algorithm iterates over all basic blocks until all blocks see no changes.
- A better implementation uses a worklist
- Initially include all basic blocks in the worklist.
- Until the worklist becomes empty, remove a basic block, B, from the list and recompute $REACH_{in}$ and $REACH_{out}$.
  - If $REACH_{out}$ changes, then for each successor, S, of B
    - If S is not yet in the worklist, append S to the worklist
- We can redo the examples using this implementation
- Worst case complexity?

- If the basic blocks contain more than a single instruction
- Then the reaching defs information within each basic block, B, can be easily computed "locally", using information in $\mathrm{REACH}_{in}(B)$.