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 \( d_1, d_2, \ldots \) 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?

\[ a \leftarrow 0 \]

\[ L_1 : b \leftarrow a + 1 \]

\[ c \leftarrow c + b \]

\[ a \leftarrow b \times 2 \]

if \( a < N \) goto \( L_1 \)

return \( c \)
• 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?

\[ a \leftarrow 0 \]

\[ L_1: \ b \leftarrow a + 1 \]

\[ c \leftarrow c + b \]

\[ a \leftarrow b \times 2 \]

if \( a < N \) goto \( L_1 \)

return \( c \)
Algorithm for computing reaching defs

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

- $\text{REACHin}(B)$: the set of defs reaching the entry of $B$
- $\text{REACHout}(B)$: the set of defs reaching the exit of $B$

To compute these, we use two pieces of information from $B$:

- $\text{GEN}(B)$: the set of defs which appear in $B$ and may reach the exit of $B$.
- $\text{KILL}(B)$: the set of defs (in the whole program scope being analyzed) whose variables are definitely redefined in $B$. 
Basic Equations

\[ \text{REACH}_{\text{out}}(B) = \text{GEN}(B) \cup (\text{REACH}_{\text{in}}(B) - \text{KILL}(B)) \]

(1)

\[ \text{REACH}_{\text{in}}(B) = \bigcup_{j \in \text{Pred}(B)} \text{REACH}_{\text{out}}(j) \]

(2)

• As in liveness analysis, we can solve this iteratively

• \text{GEN}(B) and \text{KILL}(B) are “constants”

• Initialize all \text{REACH}_{\text{in}} and \text{REACH}_{\text{out}} 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?

\[ a \leftarrow 0 \]

\[ L_1 : b \leftarrow a + 1 \]

\[ c \leftarrow c + b \]

\[ a \leftarrow b \times 2 \]

if \( a < N \) goto \( L_1 \)

return \( c \)
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?

Purdue University is an Equal Opportunity/Equal Access institution.
• If the basic blocks contain more than a single instruction

• Then the reachingdefs information within each basic block, B, can be easily computed “locally”, using information in \text{REACHin}(B).