Dataflow analysis

Discovering Global Live Ranges
of Variables

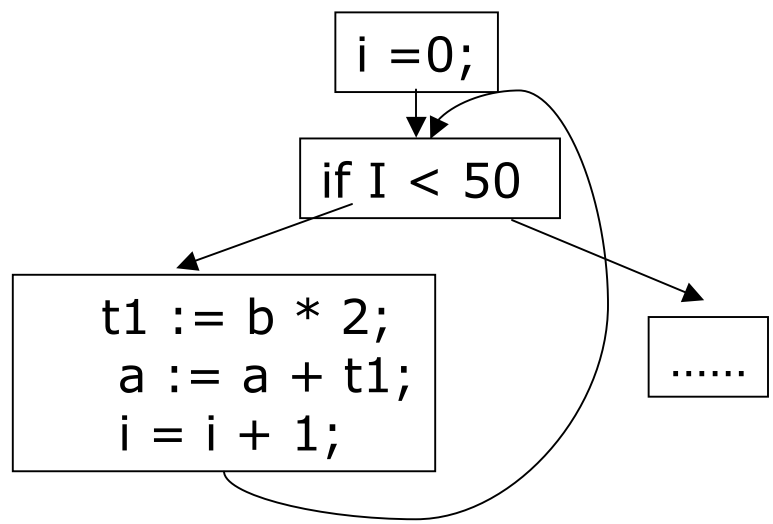
Optimization and analysis

- Requirement for optimizations
  - Correctness (safety)
    - must preserve the meaning of the input computation
  - Profitability
    - must improve code quality
- Program analysis
  - Statically examines input computation to ensure safety and profitability of optimizations
  - Compile-time reasoning of runtime program behavior
    - Undecidable in general due to external program input, complex control flow, and pointer/array references
    - Conservative approximation of program runtime behavior:
      may miss opportunities of applying optimization, but ensure all optimizations are correct
- Data-flow analysis
  - Reason about flow of values on control-flow graphs
  - Example: available expression analysis for global redundancy elimination
  - Can be used for program optimization or program understanding

Control-flow graph

- Graphical representation of runtime control-flow paths
  - Nodes of graph: basic blocks (straight-line computations)
  - Edges of graph: flows of control
- Useful for collecting information about computation
  - Detect loops, remove redundant computations, register allocation, instruction scheduling…
- Alternative CFG: Each node contains a single statement

```
……
    i = 0
   while (i < 50) {
     t1 = b * 2;
     a = a + t1;
     i = i + 1;
   }
….
```

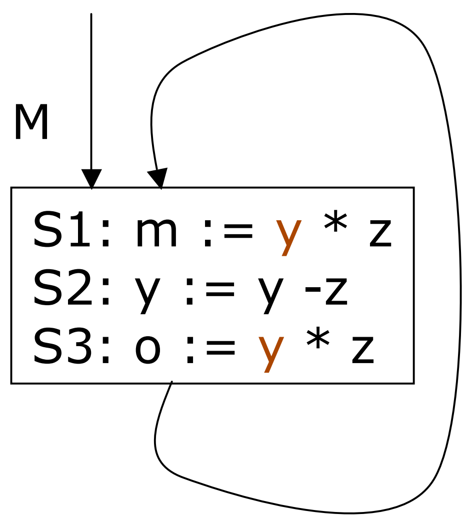


Live variable analysis

- A data-flow analysis problem
  - A variable v is live at CFG point p iff there is a path from p to a use of v along which v is not redefined
  - At any CFG point p, what variables are alive?
- Live variable analysis can be used in
  - Global register allocation
    - Dead variables no longer need to be in registers
  - Useless-store elimination
    - Dead variable don’t need to be stored back to memory
  - Uninitialized variable detection
    - No variable should be alive at program entry point

Computing live variables

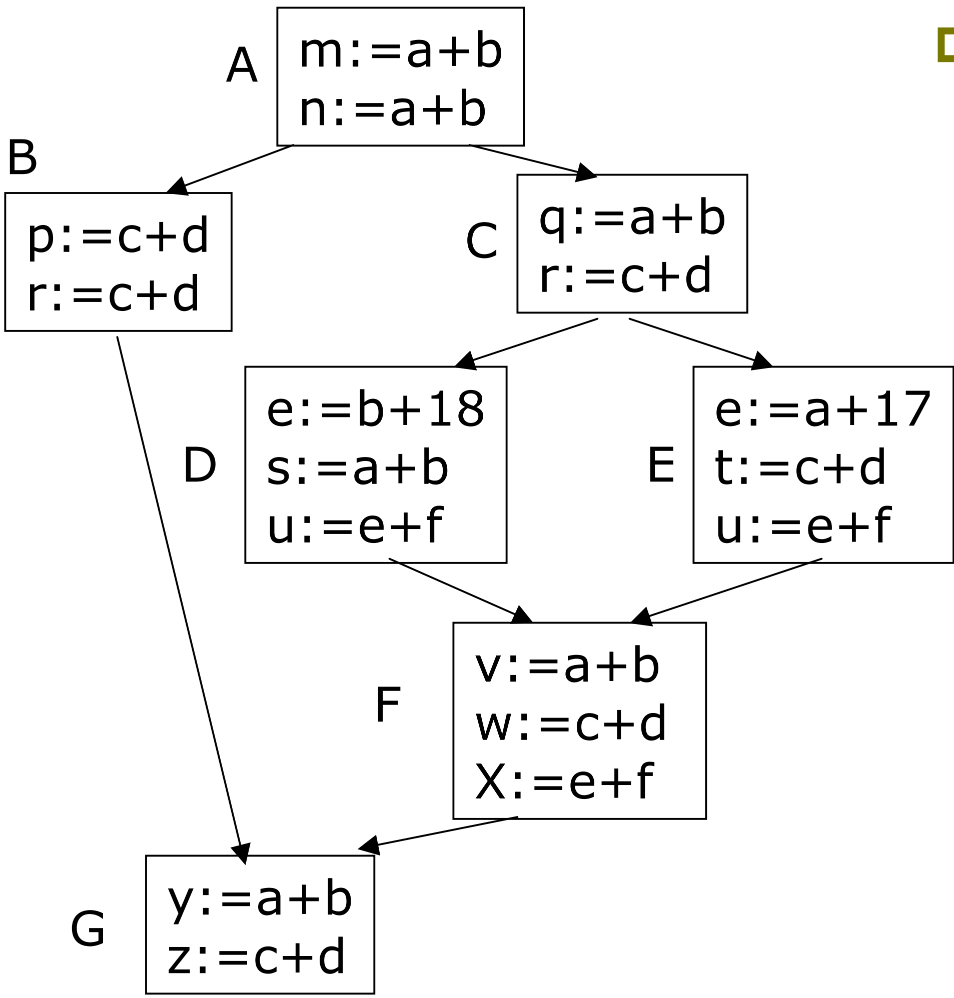
- For each basic block n, let
  - UEVar(n)=variables used before any definition in n
  - VarKill(n)=variables defined (modified) in n (killed by n)



for each basic block n:S1;S2;S3;…;Sk

```
VarKill := ∅
UEVar(n) := ∅
for i = 1 to k
   suppose Si is "x := y op z"
   if y ∉ VarKill
      UEVar(n) = UEVar(n) ∪ {y}
   if z ∉ VarKill
      UEVar(n) = UEVar(n) ∪ {z}
   VarKill = VarKill ∪ {x}
```

Computing live variables



- For each basic block n, let
  - UEVar(n)
    vars used before defined
  - VarKill(n)
    vars defined (killed by n)

Goal: evaluate vars alive on exit from n

LiveOut(n)= $\bigcup_{m \in succ(n)}$
(UEVar(m) $\cup$
(LiveOut(m)-VarKill(m))

Algorithm: computing live variables

- For each basic block n, let
  - UEVar(n)=variables used before any definition in n
  - VarKill(n)=variables defined (modified) in n (killed by n)

Goal: evaluate names of variables alive on exit from n

- $\text{LiveOut}(n) = \bigcup_{m \in \text{succ}(n)} (\text{UEVar}(m) \cup (\text{LiveOut}(m) - \text{VarKill}(m))$

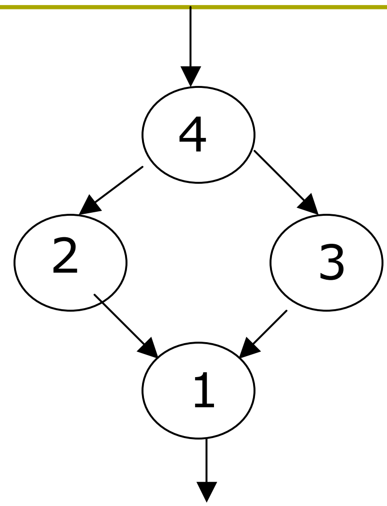
```
for each basic block bi
    compute UEVar(bi) and VarKill(bi)
    LiveOut(bi) := ∅
for (changed := true; changed; )
    changed = false
    for each basic block bi
        old = LiveOut(bi)
        LiveOut(bi)= ∪ (UEVar(m) ∪ (LiveOut(m) - VarKill(m))
                 m∈succ(bi)
        if (LiveOut(bi) != old) changed := true
```

Iterative dataflow algorithm

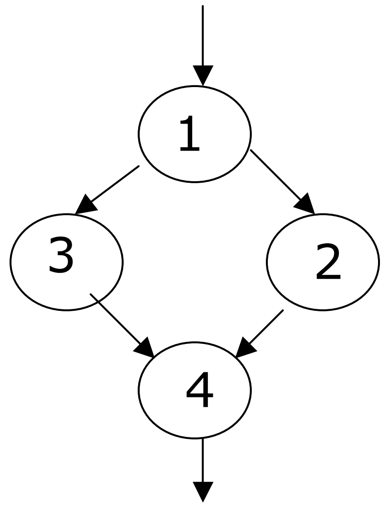
```
for each basic block bi
    compute Gen(bi) and Kill(bi)
    Result(bi) := ∅
for (changed := true; changed; )
    changed = false
    for each basic block bi
        old = Result(bi)
        Result(bi)=
        ∩ or ∪
    [m∈pred(bi) or succ(bi)]
  (Gen(m) ∪ (Result(m)-Kill(m))
        if (Result(bi) != old)
            changed := true
```

- Iterative evaluation of result sets until a fixed point is reached
  - Does the algorithm always terminate?
    - If the result sets are bounded and grow monotonically, then yes; Otherwise, no.
    - Fixed-point solution is independent of evaluation order
  - What answer does the algorithm compute?
    - Unique fixed-point solution
    - The meet-over-all-paths solution
  - How long does it take the algorithm to terminate?
    - Depends on traversing order of basic blocks

Traversing order of basic blocks



postorder



Reverse postorder

- Facilitate fast convergence to the fixed point
- Postorder traversal
  - Visits as many of a nodes successors as possible before visiting the node
  - Used in backward data-flow analysis
- Reverse postorder traversal
  - Visits as many of a node's predecessors as possible before visiting the node
  - Used in forward data-flow analysis

More about dataflow analysis

- Sources of imprecision
  - Unreachable control flow edges, array and pointer references, precedure calls
- Other data-flow programs
  - Reaching definition analysis
    - A definition point d of variable v reaches CFG point p iff there is a path from d to p along which v is not redefined
    - At any CFG point p, what definition points can reach p?
  - Very busy expression analysis
    - An expression e is very busy at a CFG point p if it is evaluated on every path leaving p, and evaluating e at p yields the same result.
    - At any CFG point p, what expressions are very busy?
  - Constant propagation analysis
    - A variable-value pair (v,c) is valid at a CFG point p if on every path from procedure entry to p, variable v has value c
    - At any CFG point p, what variables have constants?