Verifying the LLVM

Steve Zdancewic

DeepSpec Summer School 2017

Vminus Operational Semantics

- Only 5 kinds of instructions:
  - Binary arithmetic
  - Memory Load
  - Memory Store
  - Terminators
  - Phi nodes

- What is the state of a Vminus program?

Subtlety of Phi Nodes

- Phi-Nodes admit “cyclic” dependencies:

```
pred:
    ...

  br loop
```

```
loop:
  %x = ϕ[0;pred][y;loop]
  %y = ϕ[1;pred][x;loop]
  %b = %x ≤ %y
  br %b loop exit
```

Semantics of Phi Nodes

- The value of the RHS of a phi-defined uid is relative to the state at the entry to the block.

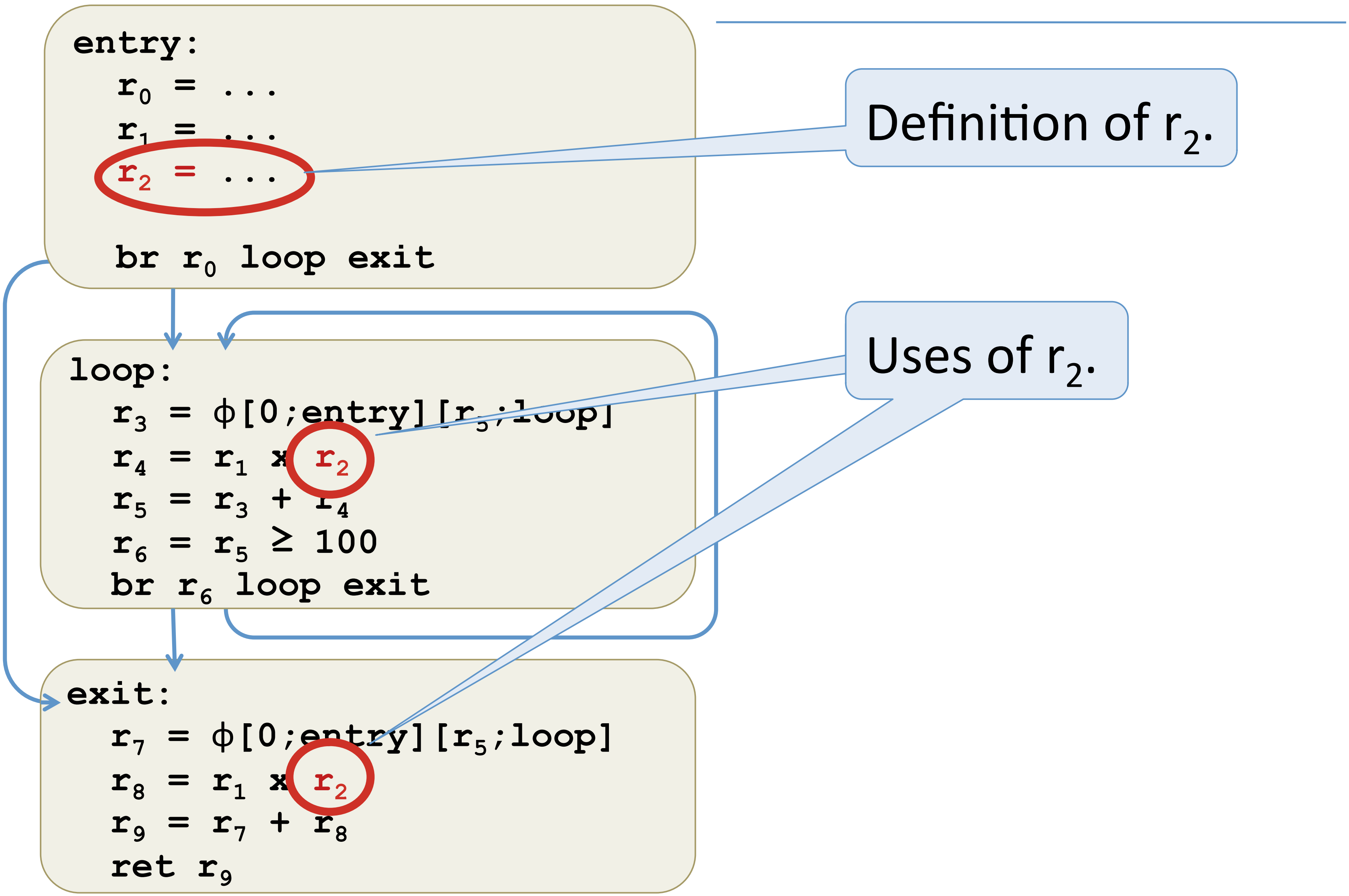
- Option 1:
  - Require all phi nodes to be at the beginning of the block
  - Execute them “atomically, in parallel”
  - (Original Vellvm followed this model)

- Option 2:
  - Keep track of the state upon entry to the block
  - Calculate the RHS of phi nodes relative to the entry state
  - (Vminus follows this model)

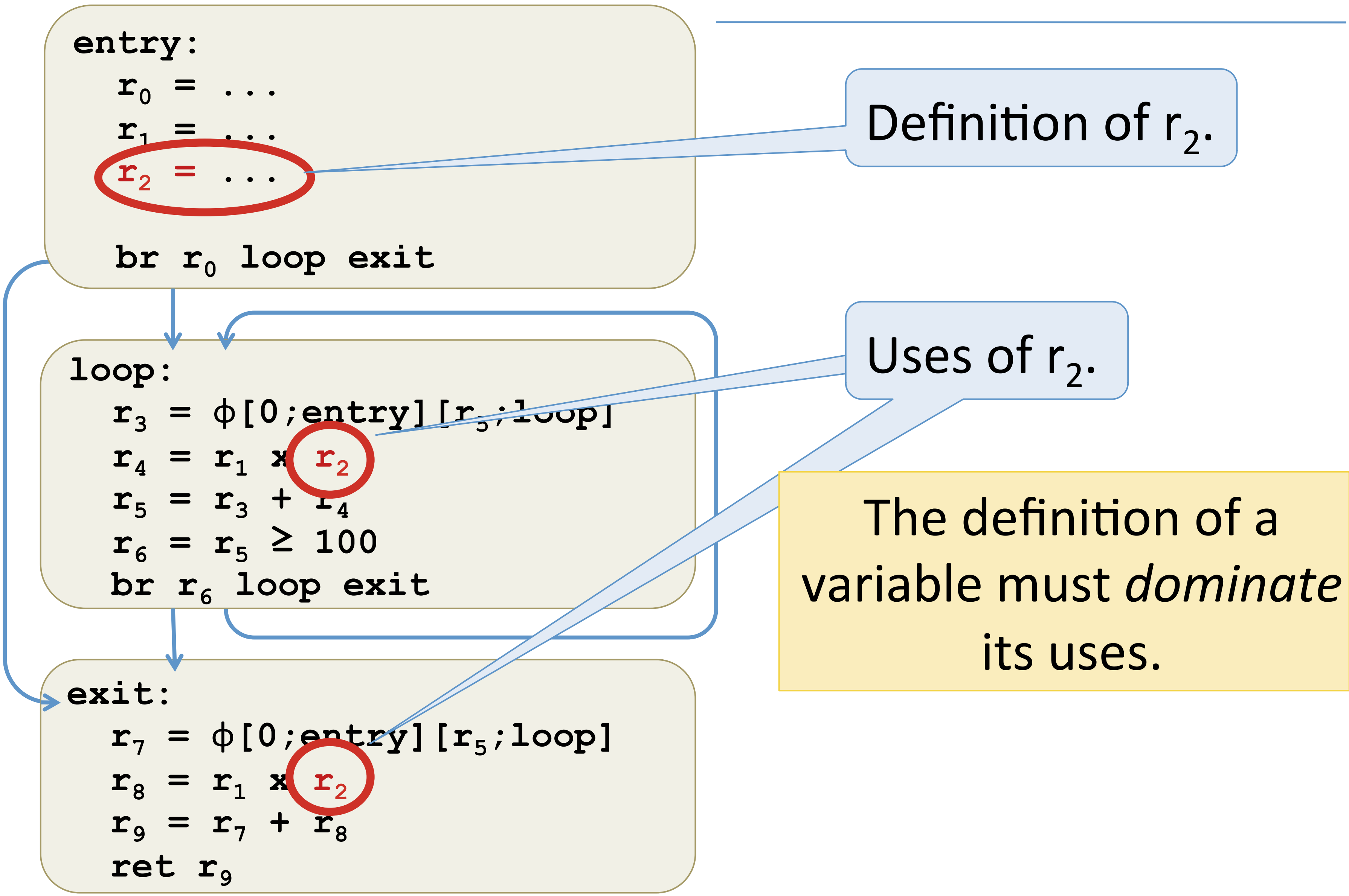
VminusOpsem.v

VMINUS OPERATIONAL SEMANTICS

Key SSA Invariant



Key SSA Invariant



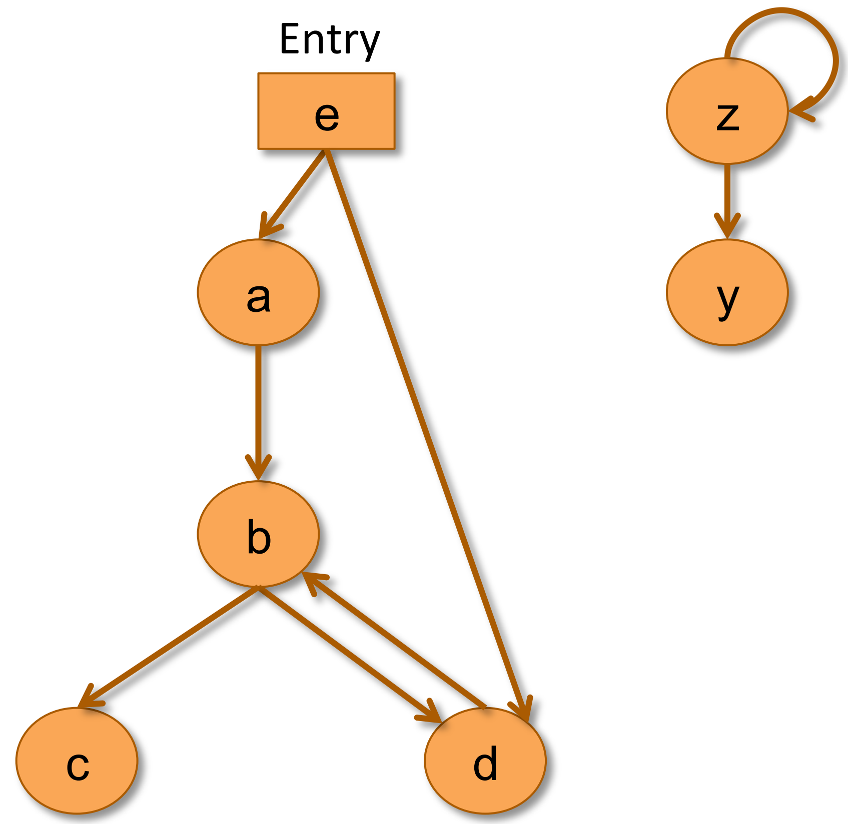
Defining SSA Variable Scope

Graph: g corresponds to a "fine grained" CFG

Nodes: program points
(maybe more than one per block)

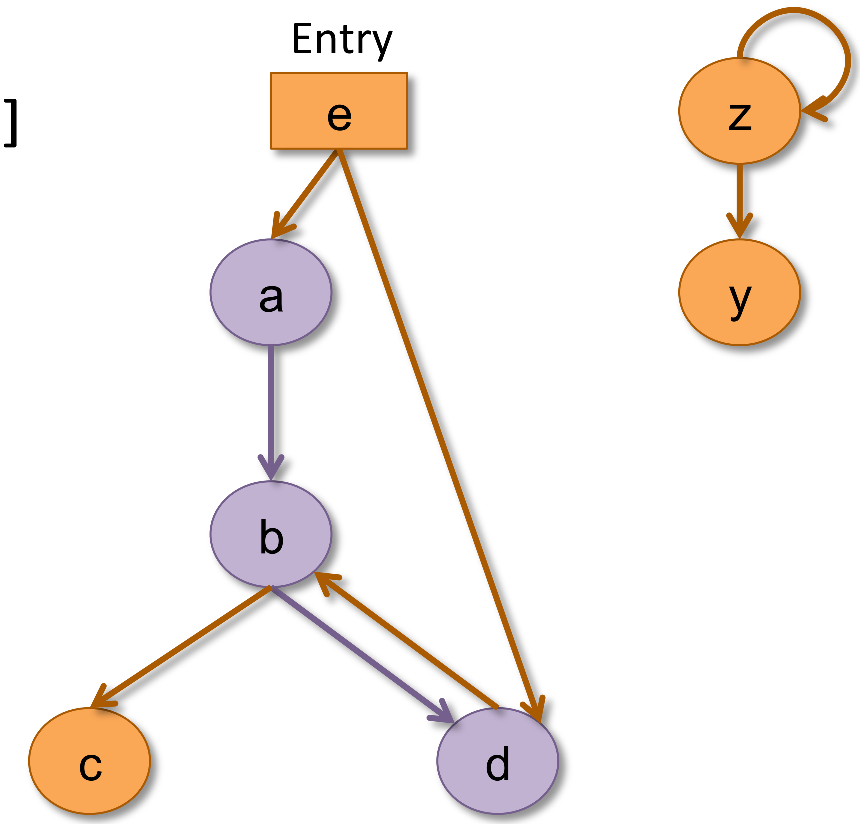
Edges: "fallthroughs", jump and branch instructions

Distinguished entry



Paths

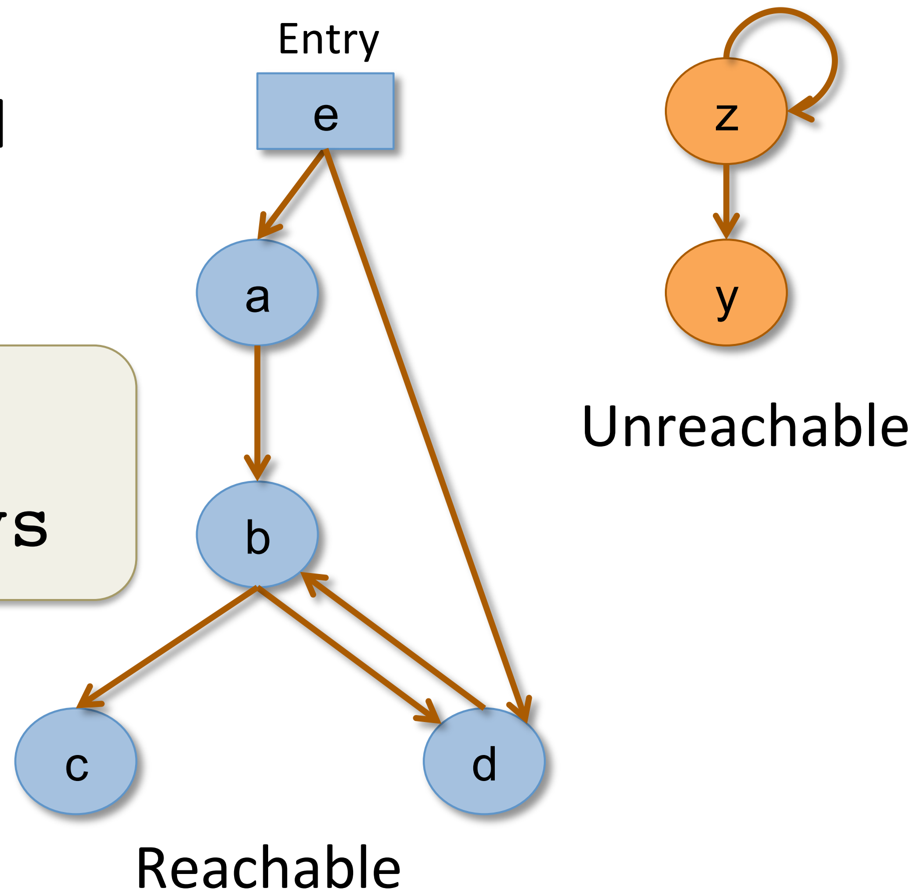
- Paths:
  `Path g a d [a;b;d]`



Reachability

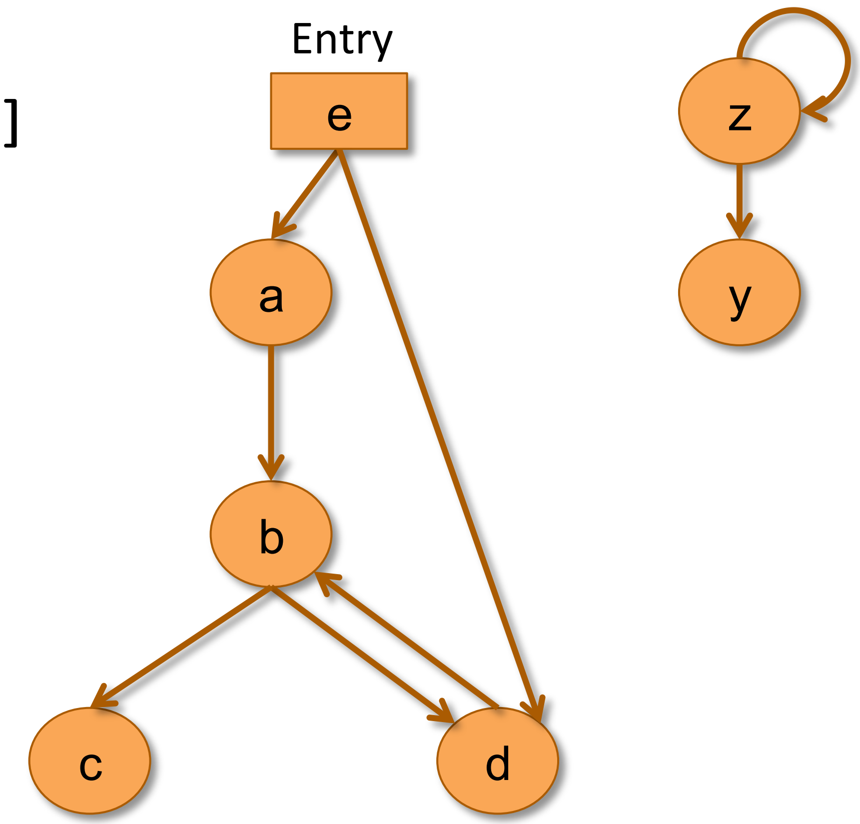
- Paths:
  `Path g a d [a;b;d]`
- Reachability:
  `Reachable g x`

iff
∃vs. `Path g e x vs`



Domination

- Paths:
  `Path g a d [a;b;d]`
- Reachability:
  `Reachable g x`
- Domination:
  `Dom g b c`

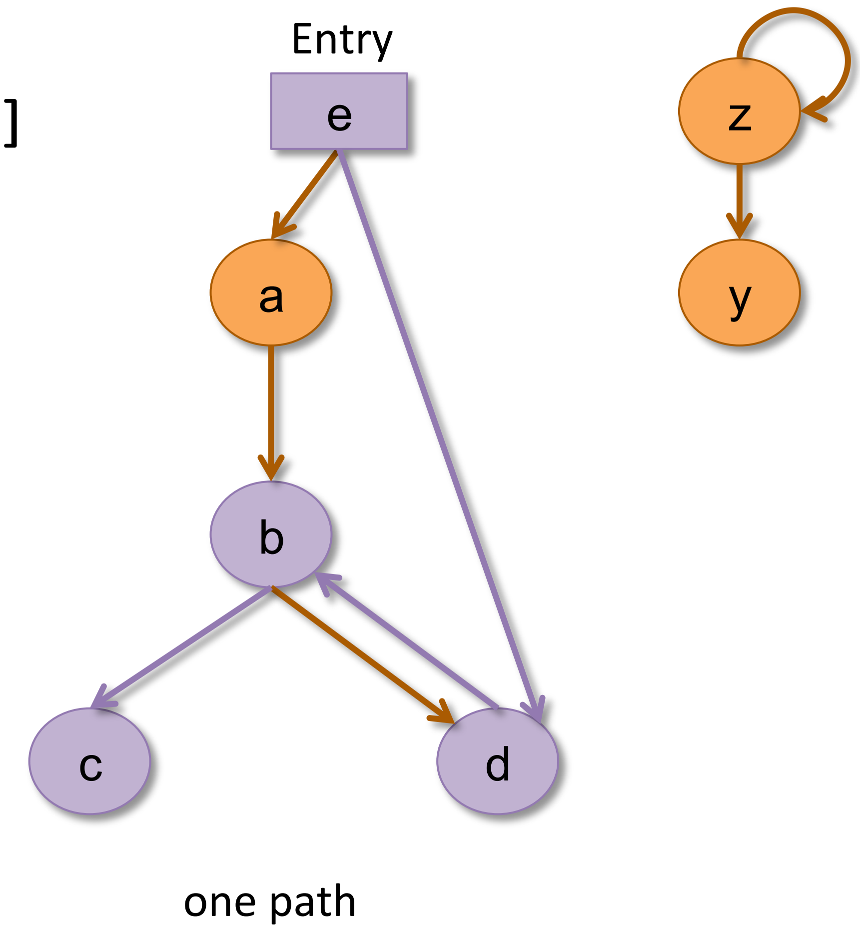


iff every path from e to c goes through b.

Domination

- Paths:
  `Path g a d [a;b;d]`
- Reachability:
  `Reachable g x`
- Domination:
  `Dom g b c`

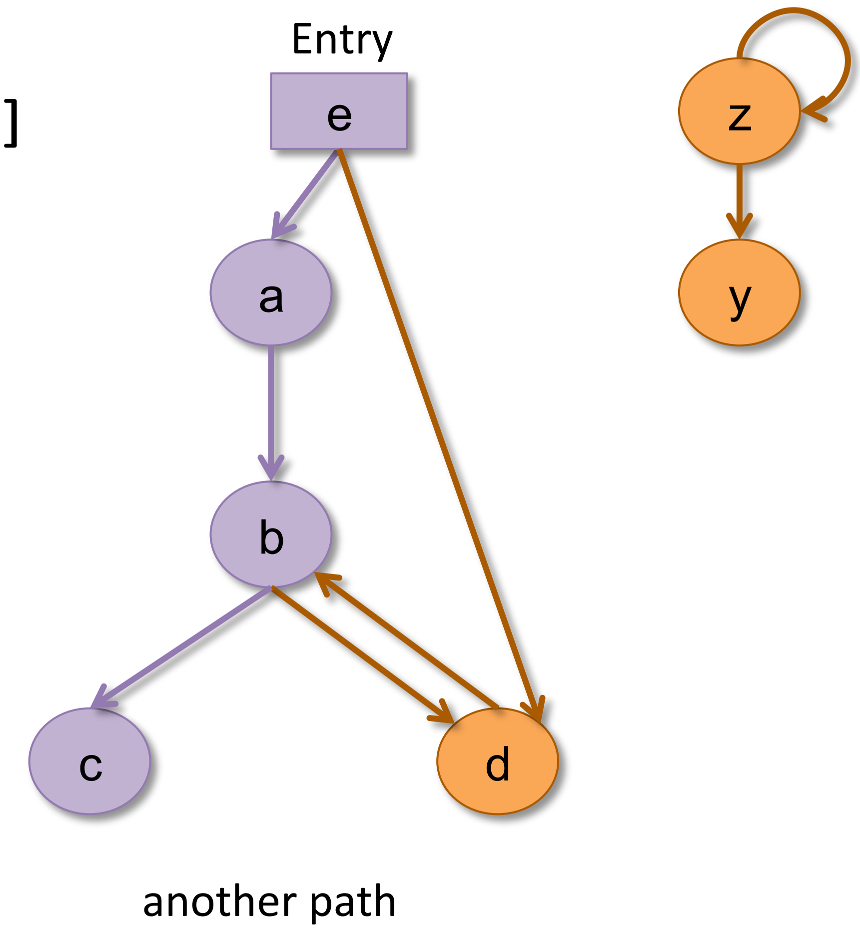
iff every path from e to c goes through b.



Domination

- Paths:
  `Path g a d [a;b;d]`
- Reachability:
  `Reachable g x`
- Domination:
  `Dom g b c`

iff every path from e to c goes through b.



Domination

- Paths:

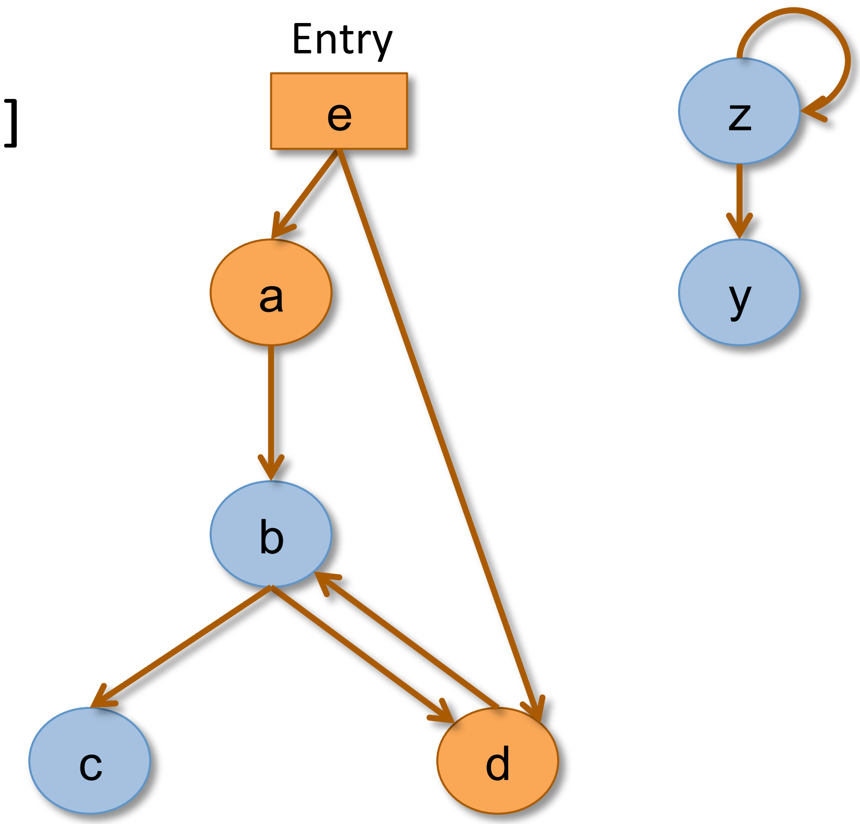
  `Path g a d [a;b;d]`

- Reachability:

  `Reachable g x`

- Domination:

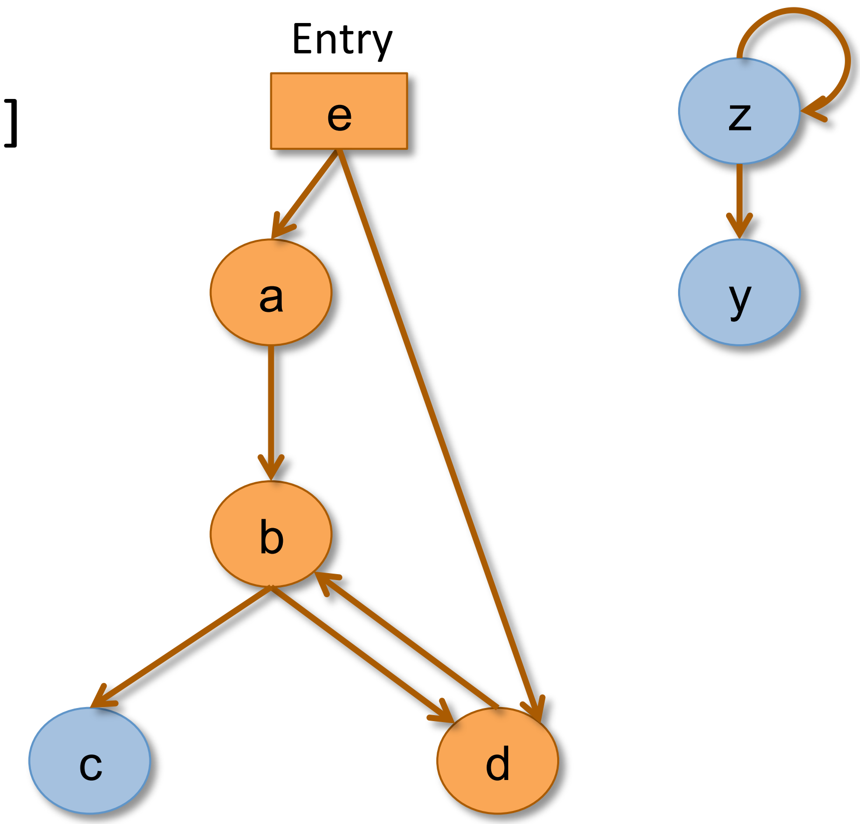
  `Dom g b c`



Nodes dominated by b.

Strict Domination

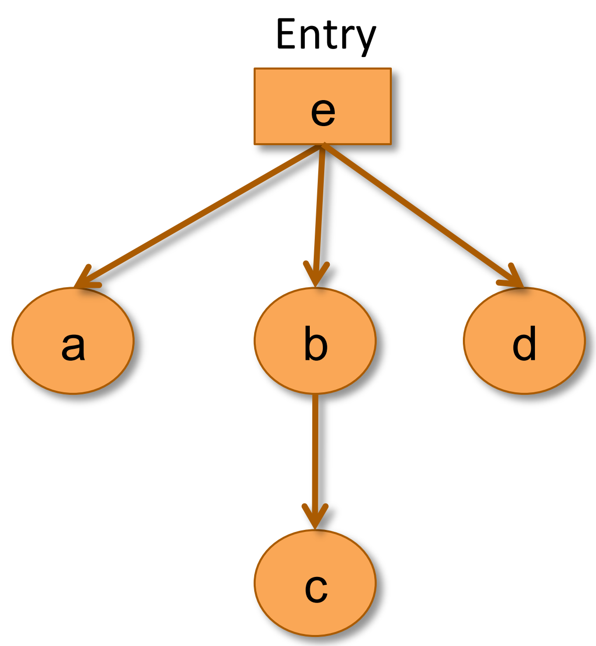
- Paths:
  `Path g a d [a;b;d]`
- Reachability:
  `Reachable g x`
- Domination:
  `Dom g b c`
- Strict Domination:
  `SDom g b c`



Nodes strictly dominated by b.

Domination Tree

- Order the reachable nodes by (immediate) dominators, you get a tree:

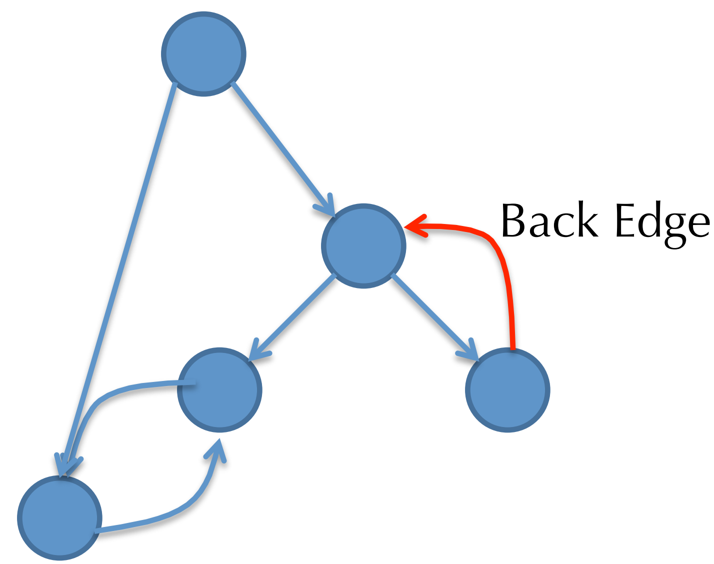


Control-flow Analysis

- Goal: Identify the loops and nesting structure of the CFG.

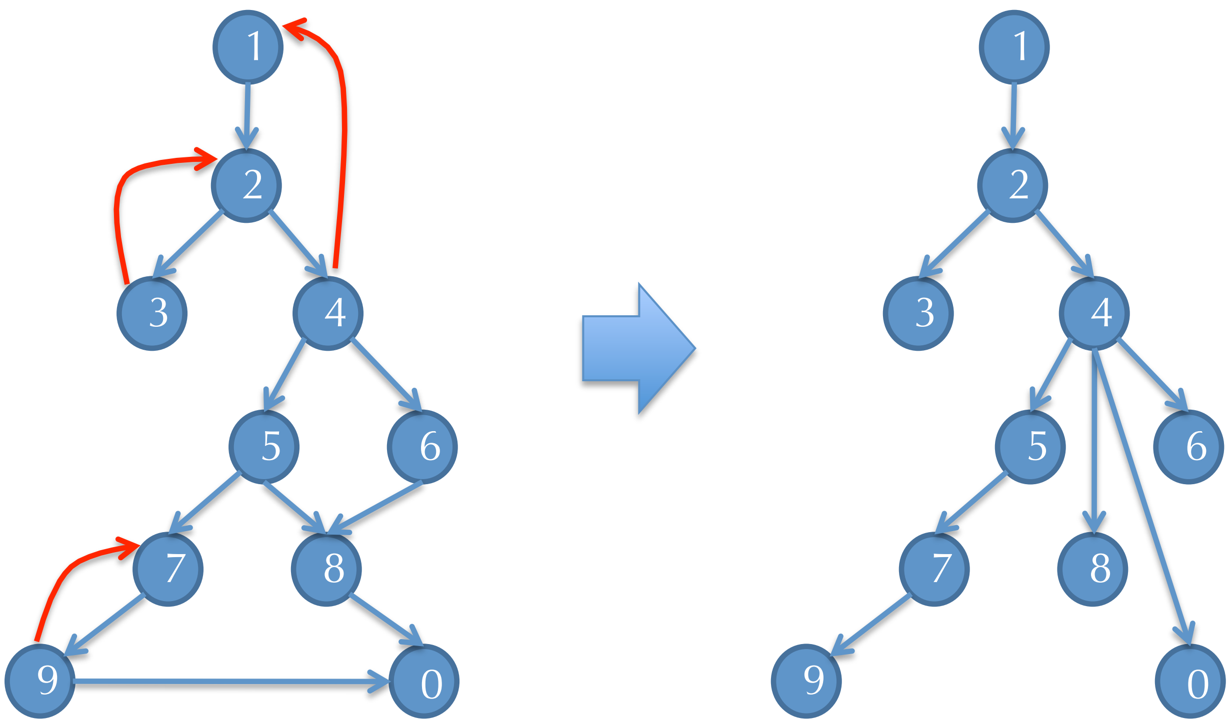
- An edge in the graph is a ***back edge*** if the target node dominates the source node.

- A loop contains at least one back edge.



Dominator Trees

- Domination is transitive:
  - if A dominates B and B dominates C then A dominates C
- Domination is anti-symmetric:
  - if A dominates B and B dominates A then A = B
- Every flow graph has a dominator tree
  - The Hasse diagram of the dominates relation

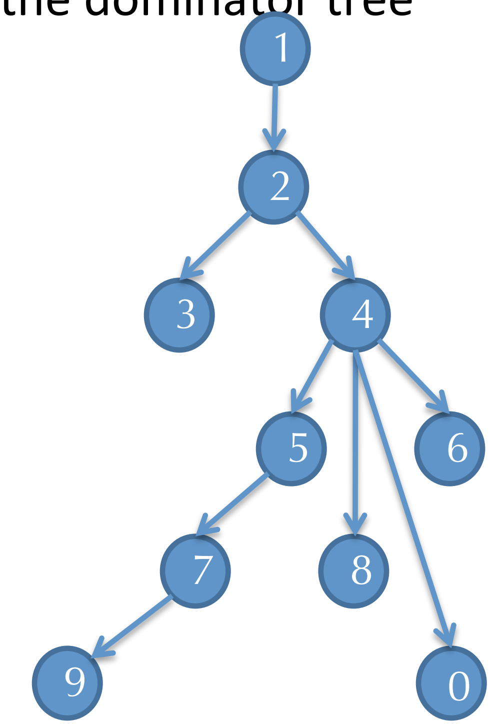


Dominator Dataflow Analysis

- Let Dom[n] = {m | m dominates n}
- We can define Dom[n] as a forward dataflow analysis.
  - Using the framework we saw earlier: Dom[n] = out[n] where:
- "A node B is dominated by another node A if A dominates *all* of the predecessors of B."
  - $in[n] := \bigcap_{n' \in pred[n]} out[n']$
- "Every node dominates itself."
  - $out[n] := in[n] \cup \{n\}$
- Formally: $\mathcal{L}$ = set of nodes ordered by $\subseteq$
  - $\top = \{\text{all nodes}\}$
  - $F_n(x) = x \cup \{n\}$
  - $\sqcap$ is $\cap$
- Easy to show monotonicity and that F_n distributes over meet.
  - Bounded set of variables: so algorithm terminates

Improving the Algorithm

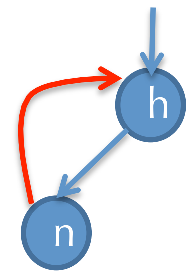
- Dom[b] contains just those nodes along the path in the dominator tree from the root to b:
  - e.g. Dom[8] = {1,2,4,8}, Dom[7] = {1,2,4,5,7}
  - There is a lot of sharing among the nodes
- More efficient way to represent Dom sets is to store the dominator *tree*.
  - doms[b] = immediate dominator of b
  - doms[8] = 4, doms[7] = 5
- To compute Dom[b] walk through doms[b]
- Need to efficiently compute intersections of Dom[a] and Dom[b]
  - Traverse up tree, looking for least common ancestor:
  - Dom[8] ∩Dom[7] = Dom[4]



- See: “A Simple, Fast Dominance Algorithm” Cooper, Harvey, and Kennedy

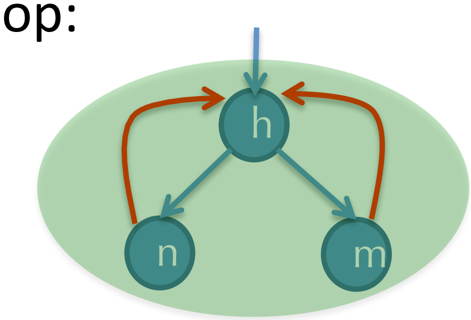
Completing Control-flow Analysis

- Dominator analysis identifies ***back edges***:
  - Edge n → h where h dominates n
- Each back edge has a ***natural loop***:
  - h is the header
  - All nodes reachable from h that also reach n without going through h



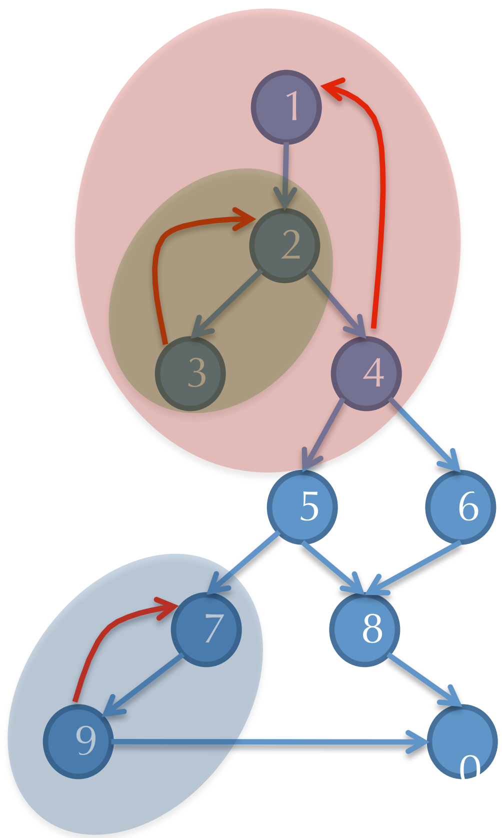
- For each back edge n → h, find the natural loop:
  - {n' | n is reachable from n' in G – {h}} ∪ {h}

- Two loops may share the same header: merge them



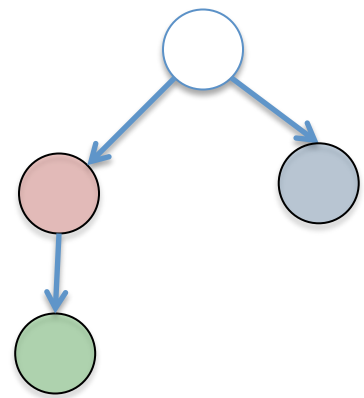
- Nesting structure of loops is determined by set inclusion
  - Can be used to build the control tree

Example Natural Loops



Natural Loops

Control Tree:



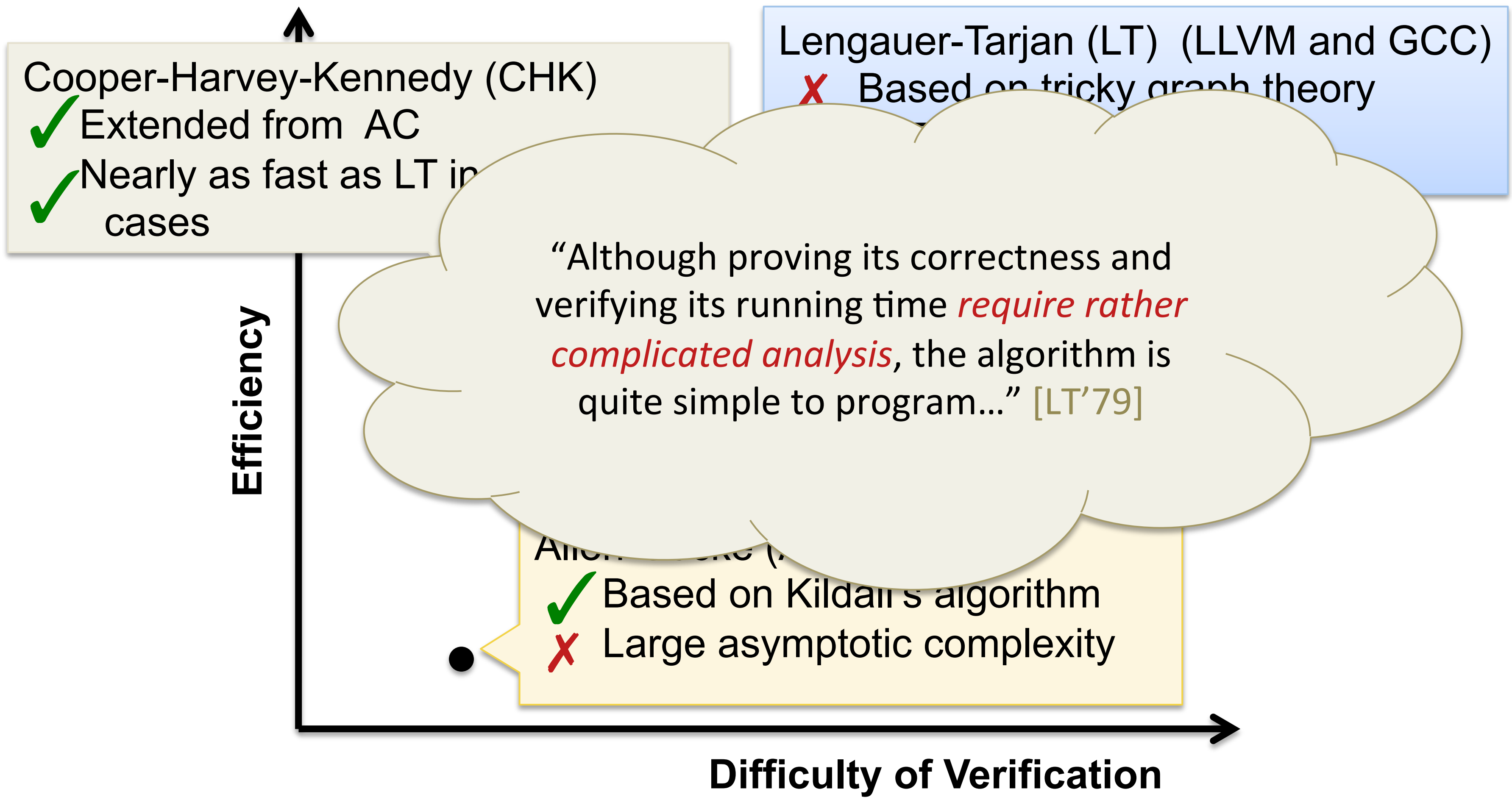
The control tree depicts the nesting structure of the program.

Uses of Control-flow Information

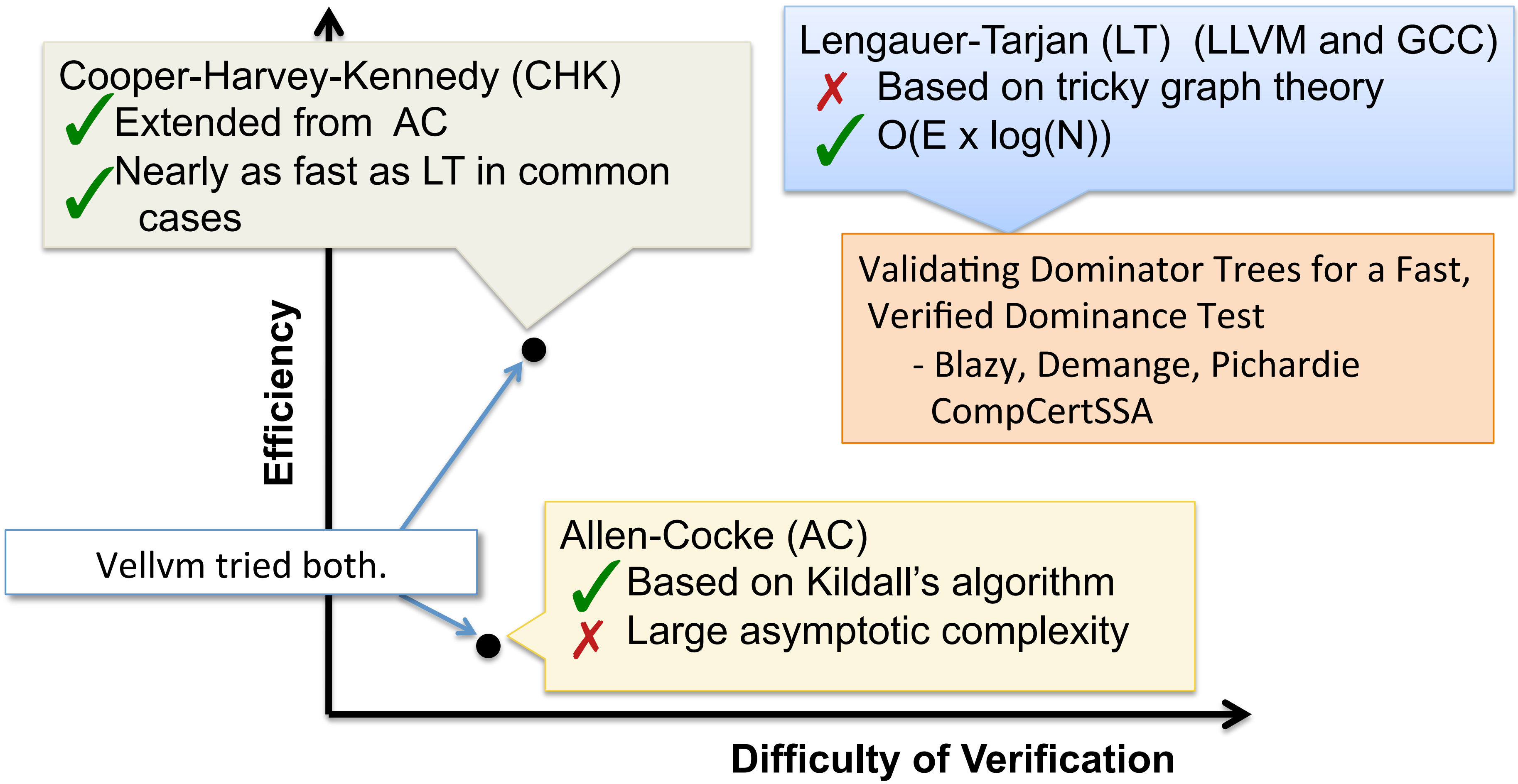
- Loop nesting depth plays an important role in optimization heuristics.
  - Deeply nested loops pay off the most for optimization.

- Need to know loop headers / back edges for doing
  - loop invariant code motion
  - loop unrolling

Dominator Algorithm Tradeoffs



Dominator Algorithm Tradeoffs



Dom.v

Kildall.v

DomKildall.v

DOMINATORS

Safety Properties

- A well-formed program never accesses undefined variables.

  If $\vdash f$ and $f \vdash \sigma_0 \longmapsto^* \sigma$ then σ is not stuck.

  $\vdash f$ program f is well formed

  σ program state

  $f \vdash \sigma \longmapsto^* \sigma$ evaluation of f

- *Initialization*:

  If $\vdash f$ then $wf(f, \sigma_0)$.

- *Preservation*:

  If $\vdash f$ and $f \vdash \sigma \longmapsto \sigma'$ and $wf(f, \sigma)$ then $wf(f, \sigma')$

- *Progress*:

  If $\vdash f$ and $wf(f, \sigma)$ then $f \vdash \sigma \longmapsto \sigma'$

Safety Properties

- A well-formed program never accesses undefined variables.

If $\vdash f$ and $f \vdash \sigma_0 \longmapsto^* \sigma$ then σ is not stuck.

$\vdash f$	program f is well formed
σ	program state
$f \vdash \sigma \longmapsto^* \sigma$	evaluation of f

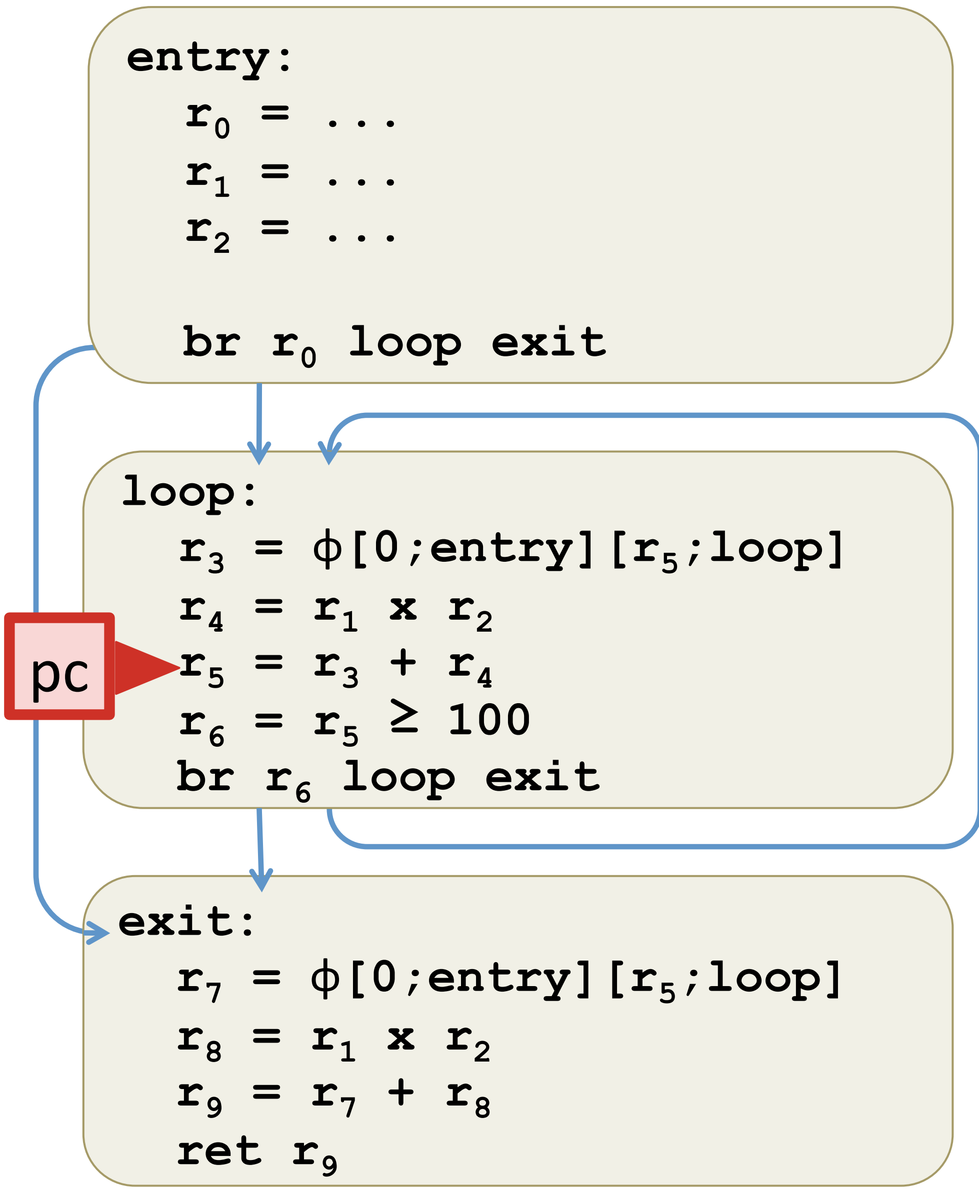
- *Initialization*: If $\vdash f$ then $wf(f, \sigma_0)$.
- *Preservation*:

If $\vdash f$ and $f \vdash \sigma \longmapsto \sigma'$ and $wf(f, \sigma)$ then $wf(f, \sigma')$

- *Progress*:

If $\vdash f$ and $wf(f, \sigma)$ then done(f,σ) or stuck(f,σ) or $f \vdash \sigma \longmapsto \sigma'$

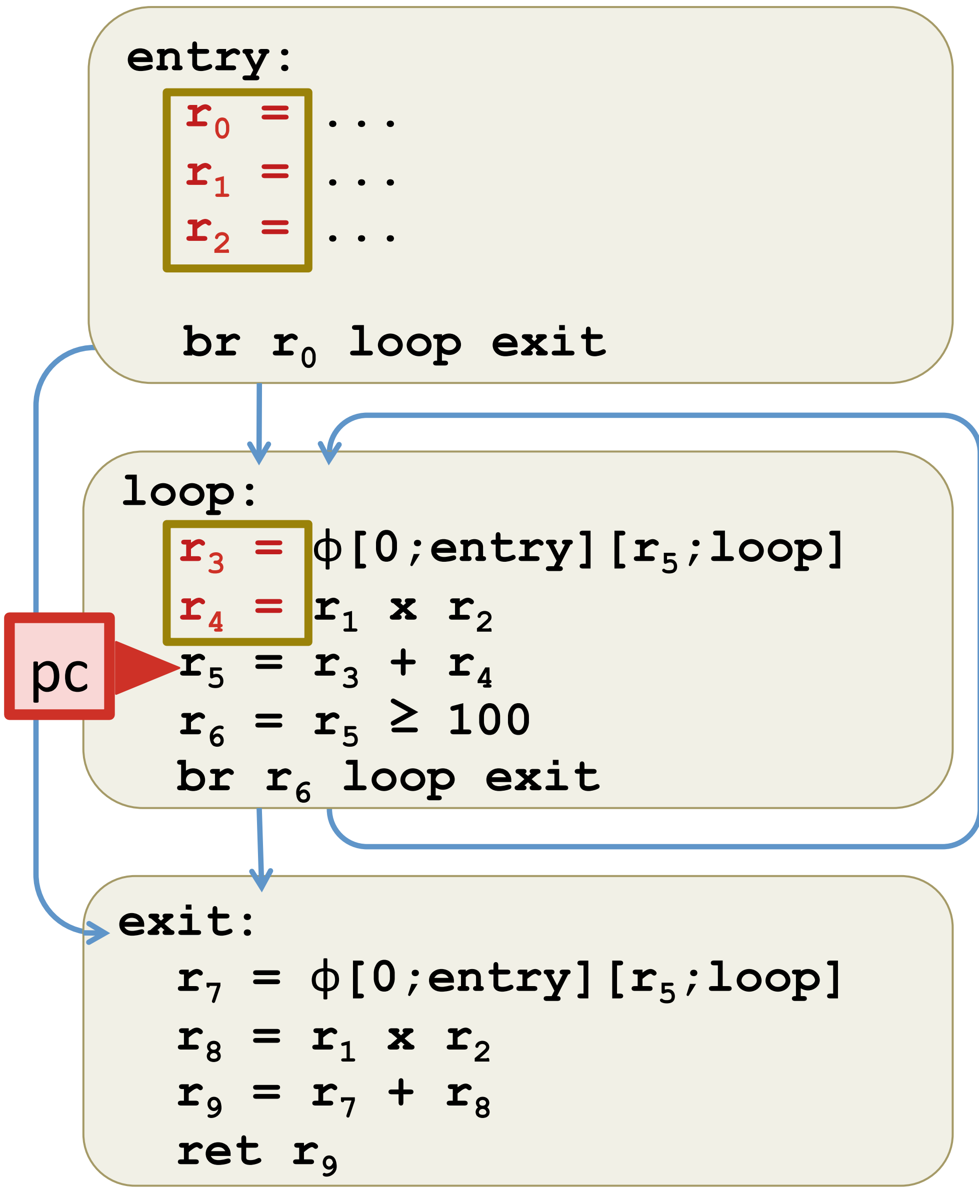
Well-formed States



State σ is:

- pc = program counter
- δ = local values

Well-formed States (Roughly)



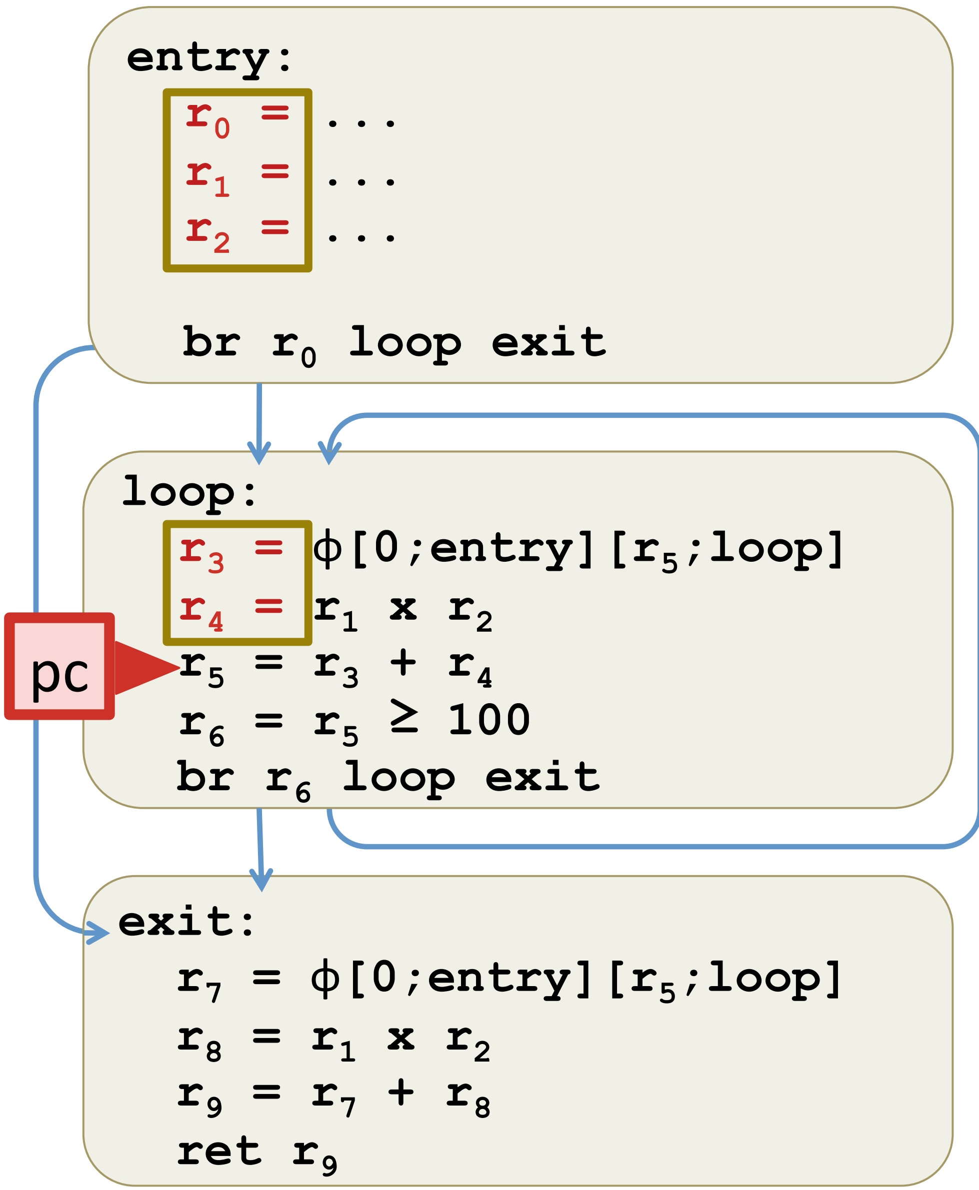
State σ is:

pc = program counter

δ = local values

sdom(f,pc) = variable defns. that *strictly dominate* pc.

Well-formed States (Roughly)



State σ contains:

- pc = program counter
- δ = local values

mem = memory

sdom(f,pc) = variable defns. that *strictly dominate* pc.

wf(g,σ) = $\forall r \in \text{sdom}(f,pc).\ \exists v.\ \delta(r) = \lfloor v \rfloor$

"All variables in scope are initialized."

VminusStatics.v

VMINUS STATIC SEMANTICS