Topic 10: Static Single Assignment

COS 320

Compiling Techniques

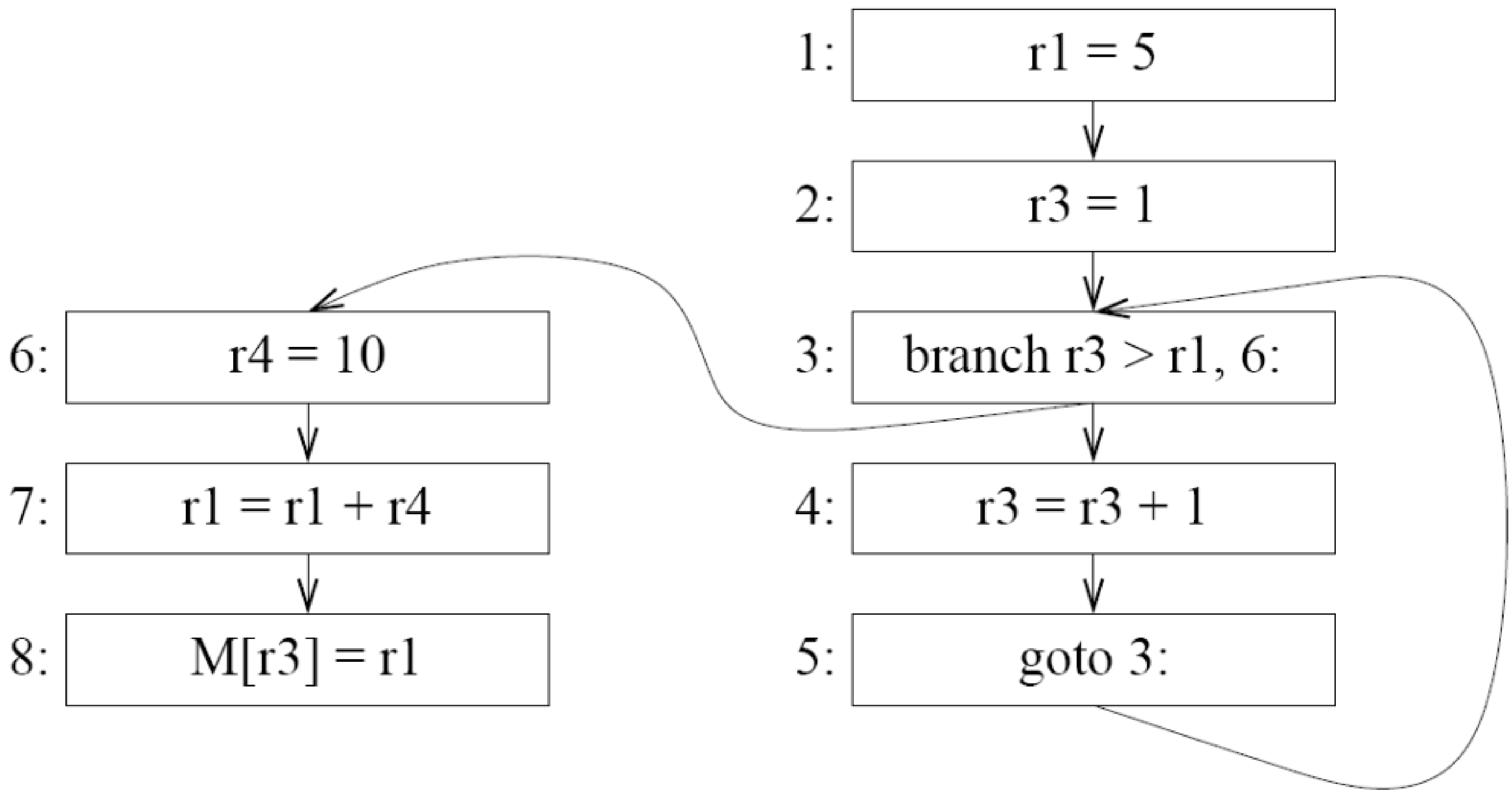
Princeton University
Spring 2015

Prof. David August

Def-Use Chains, Use-Def Chains

- Many optimizations need to find all use-sites for each definition, and all definition-sites for each use.
  - Constant propagation must refer to the definition-site of the unique reaching definition.
  - Copy propagation, reverse copy propagation, common sub-expression elimination...
- Information connecting all use-sites to corresponding definition-sites can be stored as *def-use chains* and/or *use-def chains*.
- *def-use chains*: for each definition d of r, list of pointers to all uses of r that d reaches.
- *use-def chains*: for each use u of r, list of pointers to all definitions of r that reach u.

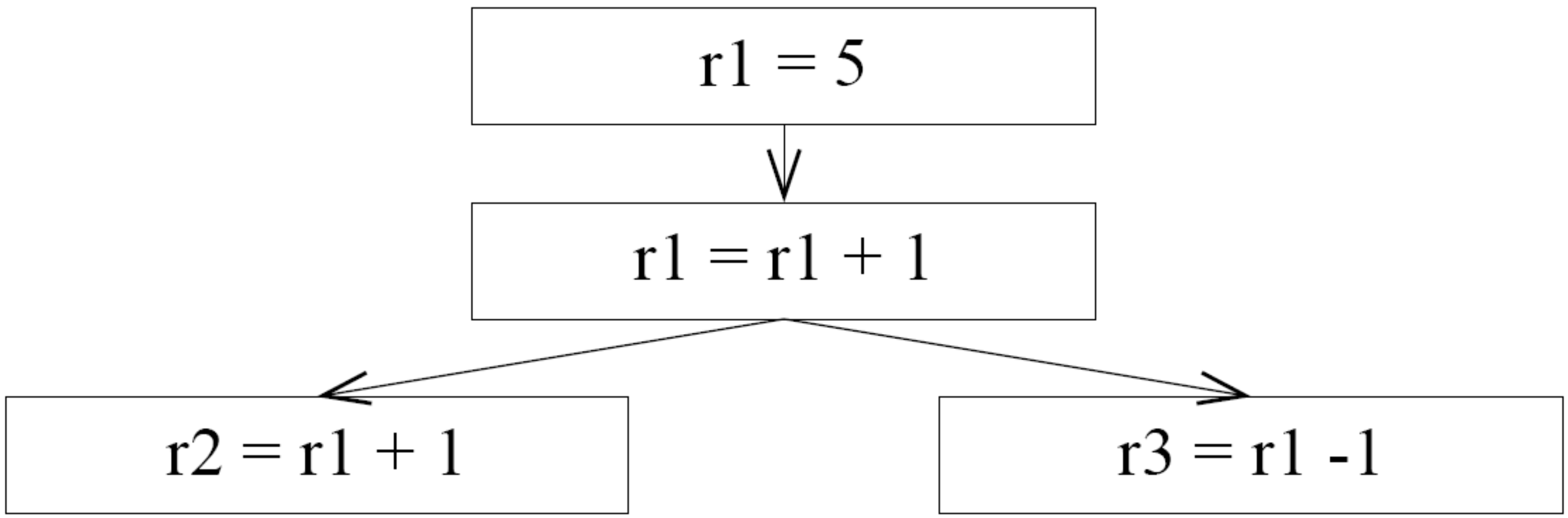
Use-Def Chains, Def-Use Chains



Static Single Assignment

Static Single Assignment (SSA):

- improvement on def-use chains
- each register has only one definition in program
- for each use u of r, only one definition of r reaches u



Why SSA?

Static Single Assignment Advantages:

- Dataflow analysis and code optimization made simpler.
  - Variables have only one definition - no ambiguity.
  - Dominator information is encoded in the assignments.
- Less space required to represent def-use chains. For each variable, space is proportional to uses * defs.
- Eliminates unnecessary relationships:

```
for i = 1 to N do A[i] = 0
for i = 1 to M do B[i] = 1
```

  - No reason why both loops should be forced to use same register to hold index register.
  - SSA renames second `i` to new register which may lead to better register allocation/optimization.

(Dynamic Single Assignment is also proposed in the literature.)

Conversion to SSA Code

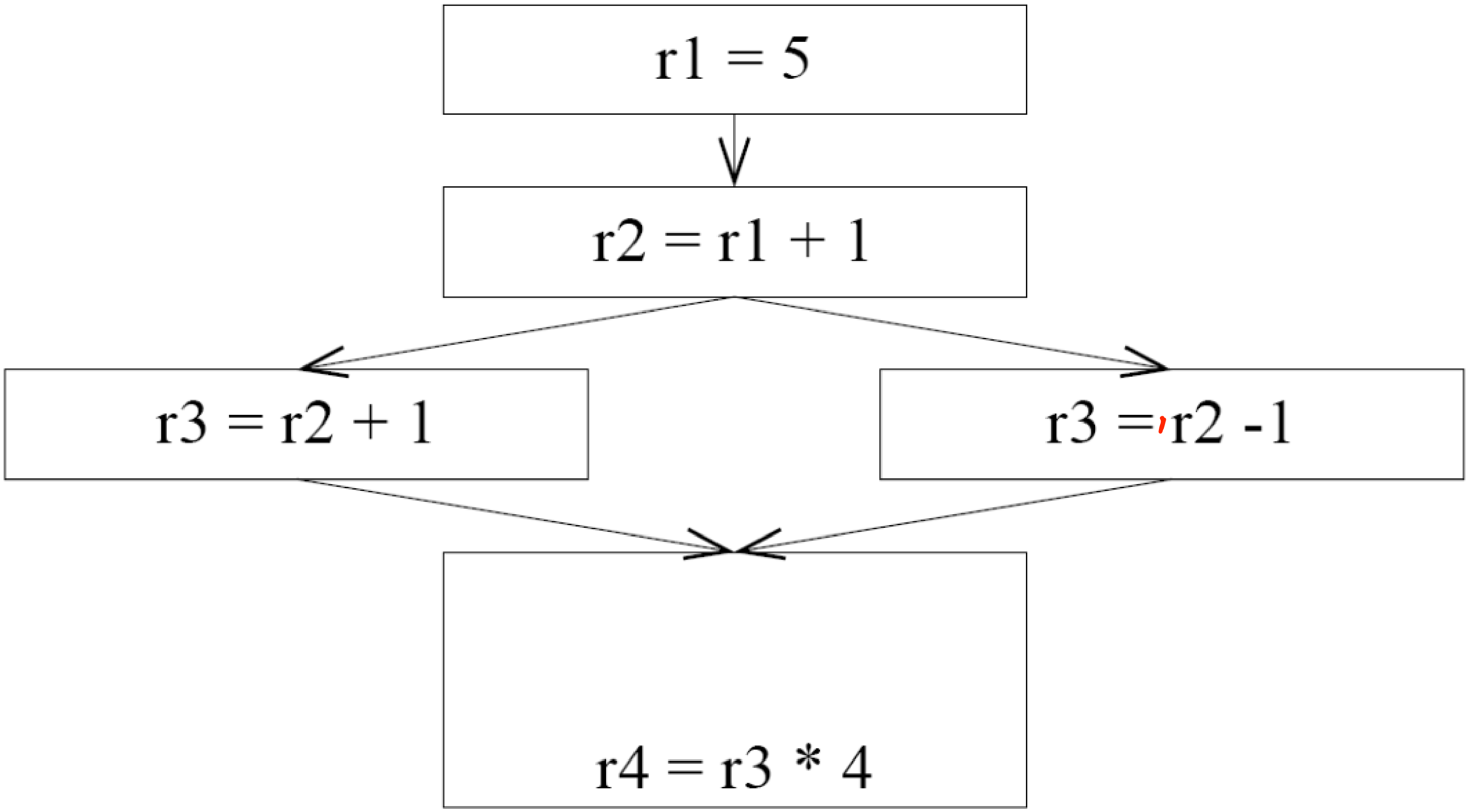
Easy to convert basic blocks into SSA form:

- Each definition modified to define brand-new register, instead of redefining old one.
- Each use of register modified to use most recently defined version.

```
r1 = r3 + r4

r2 = r1 - 1

r1 = r4 + r2

r2 = r5 * 4

r1 = r1 + r2
```

Control flow introduces problems.

Conversion to SSA Form



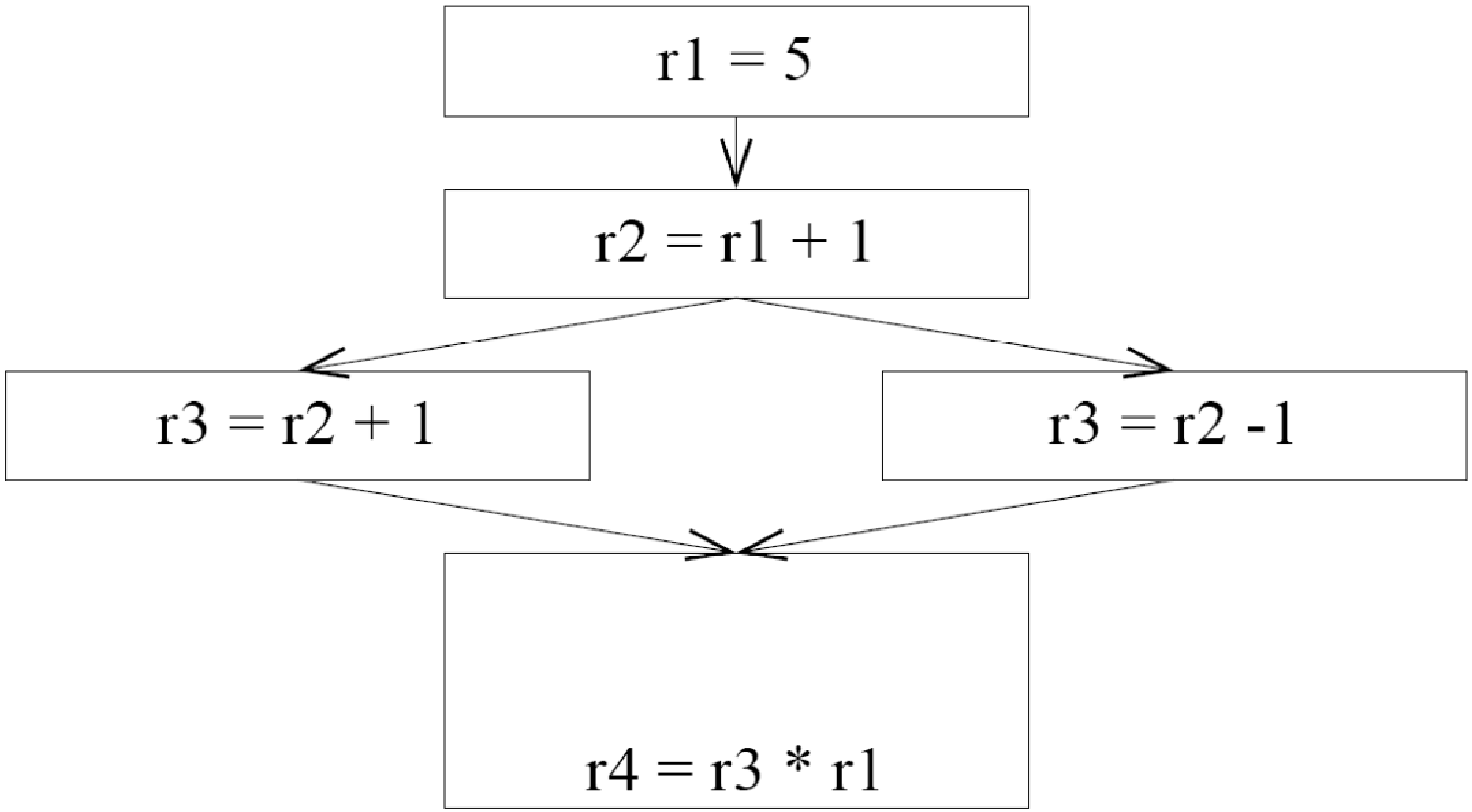
Use ϕ functions.

Conversion to SSA Form

- ϕ-functions enable the use of r3 to be reached by exactly one definition of r3.
- $r3'' = \phi(r3, r3')$:
  - $r3'' = r3$ if control enters from left
  - $r3'' = r3'$ if control enters from right
- Can implement ϕ-functions as set of move operations on each incoming edge.
- In practice, ϕ-functions are just used as notation.

Conversion to SSA Form

Can insert ϕ-functions for each register at each node with more than two predecessors.



We can do better...

Conversion to SSA Form

Path-Convergence Criterion: Insert a ϕ-function for a register r at node z of the flow graph if ALL of the following are true:

1. There is a block x containing a definition of r.
2. There is a block $y \neq x$ containing a definition of r.
3. There is a non-empty path P_{xz} of edges from x to z.
4. There is a non-empty path P_{yz} of edges from y to z.
5. Paths P_{xz} and P_{yz} do not have any node in common other than z.
6. The node z does not appear within both P_{xz} and P_{yz} prior to the end, though it may appear in one or the other.

Assume CFG entry node contains implicit definition of each register:

- r = actual parameter value
- r = undefined

ϕ-functions are counted as definitions.

Conversion to SSA Form

Solve path-convergence iteratively:

WHILE (there are nodes x, y, z satisfying conditions 1-6) &&
(z does not contain a phi-function for r) DO:
insert $r = \phi(r, r, ..., r)$ (one per predecessor) at node z.

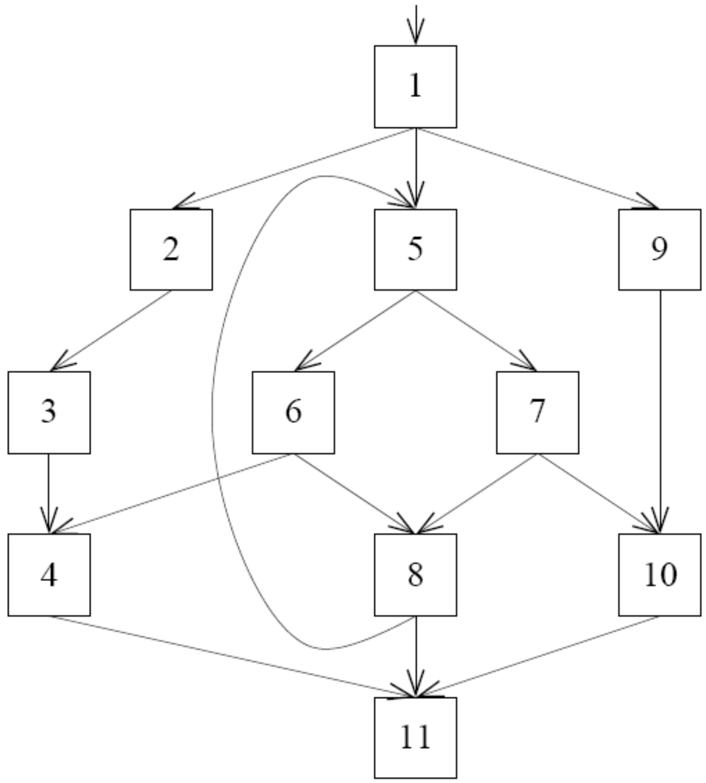
- Costly to compute.
- Since definitions dominate uses, use domination to simplify computation.

Use *Dominance Frontier...*

Dominance Frontier

Definitions:

- x *strictly dominates* w if x dominates w and $x \neq w$.
- *dominance frontier* of node x is set of all nodes w such that x dominates a predecessor of w, but does not strictly dominate w.



Dominance Frontier

- *Dominance Frontier Criterion:* Whenever node x contains definition of some register r, then need to insert ϕ-function for r in all nodes z in dominance frontier of x.
- *Iterated Dominance Frontier:* Need to repeatedly apply since ϕ-function counts as a definition.

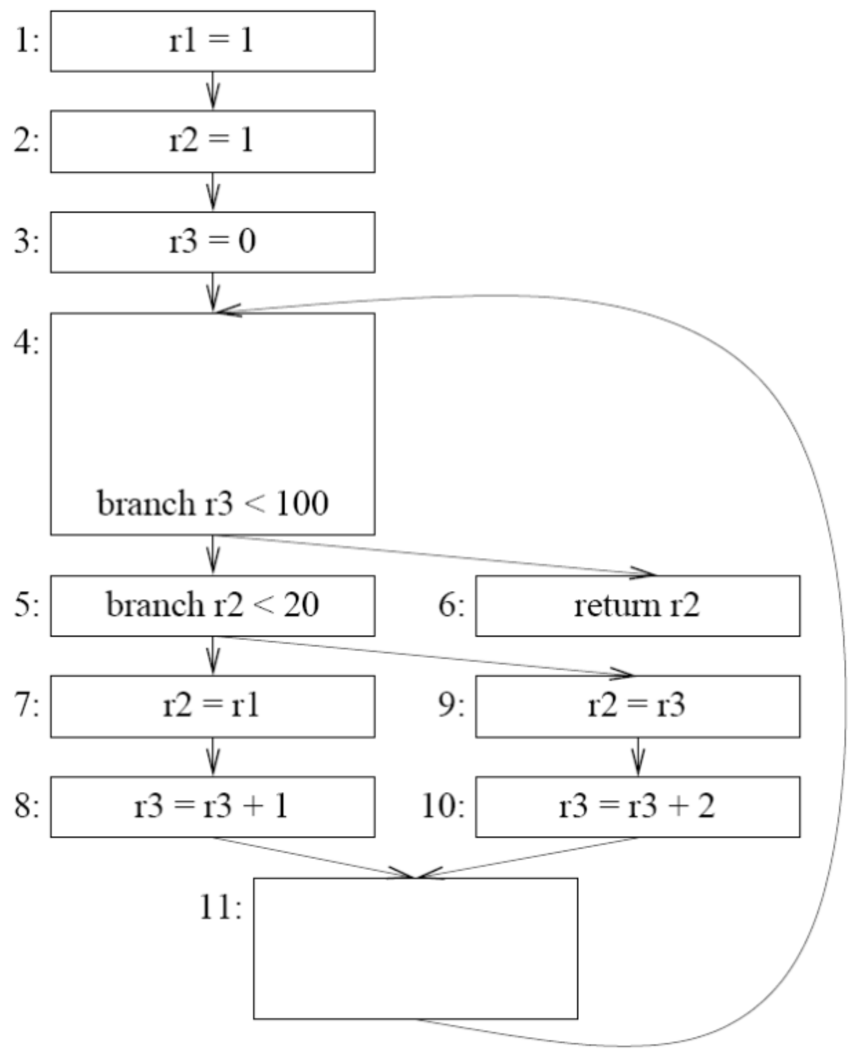
Dominance Frontier Computation

- Use dominator tree
- $DF[n]$: dominance frontier of n
- $DF_{local}[n]$: successors of n in CFG that are not strictly dominated by n
- $DF_{up}[c]$: nodes in dominance frontier of c that are not strictly dominated by c's immediate dominator

$$DF[n] = DF_{local}[n] \cup \left(\cup_{c \in children[n]} DF_{up}[c]\right)$$

- where $children[n]$ are the nodes whose idom is n.
- Work bottom up in dominator tree.

SSA Example



Node	$DOM[n]$	$IDOM[n]$
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		

Dominator Analysis

- If d dominates each of the p_i, then d dominates n.
- If d dominates n, then d dominates each of the p_i.
- $Dom[n]$ = set of nodes that dominate node n.
- N = set of all nodes.
- Computation:

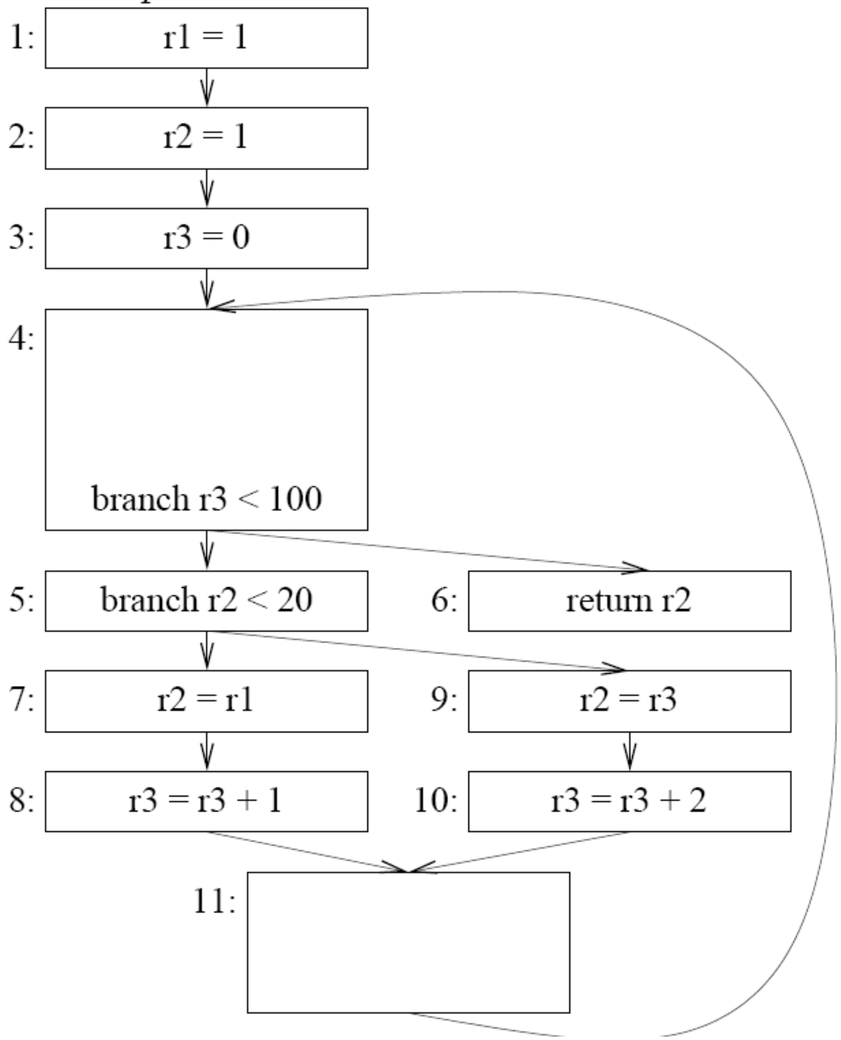
1. $Dom[s_0] = \{s_0\}$.
2. **for** $n \in N - \{s_0\}$ **do** $Dom[n] = N$
3. **while** (changes to any $Dom[n]$ occur) **do**
4. **for** $n \in N - \{s_0\}$ **do**
5. $Dom[n] = \{n\} \cup \left(\cap_{p \in pred[n]} Dom[p]\right)$.

SSA Example

The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

SSA Example

Insert *phi*-functions:

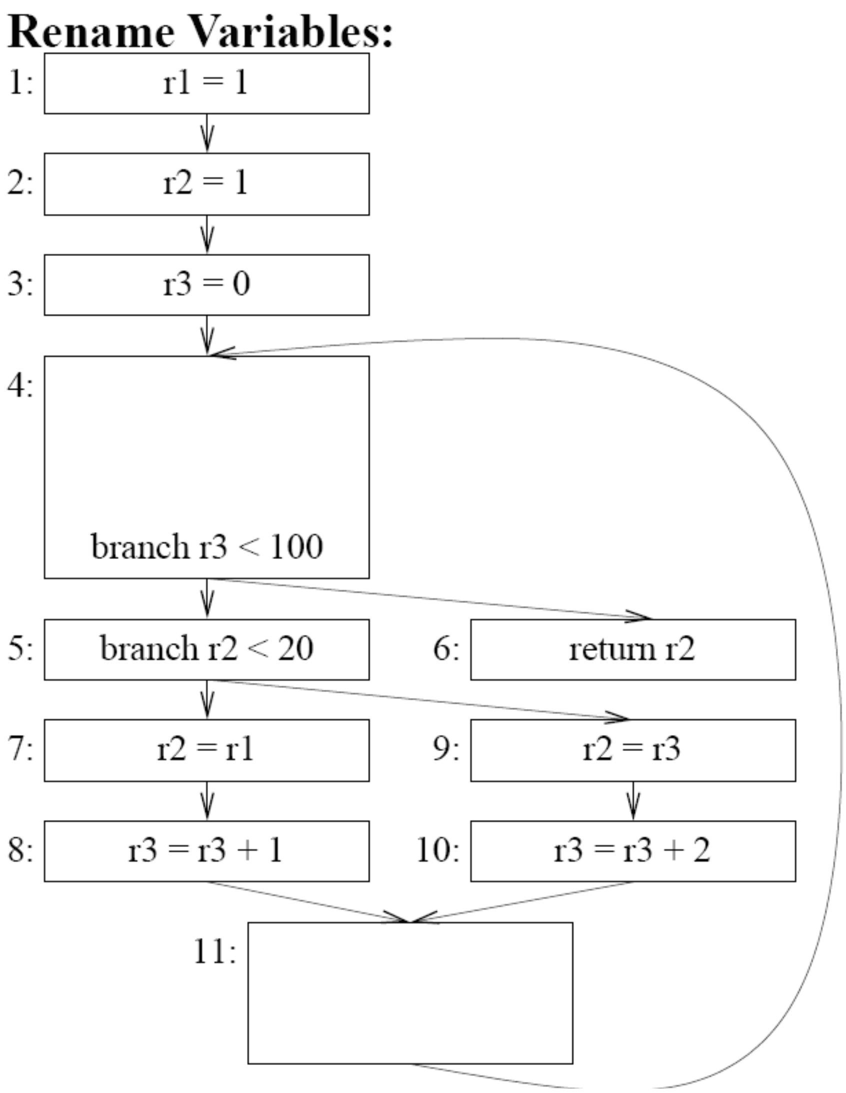


SSA Example

Rename Variables:

1. traverse dominator tree, renaming different definitions of r to $r_1, r_2, r_3 \ldots$
2. rename each regular use of r to most recent definition of r
3. rename ϕ-function arguments with each incoming edge's unique definition

SSA Example


Rename Variables:
1:
r1 = 1
2:
r2 = 1
3:
r3 = 0
4:
branch r3 < 100
5:
branch r2 < 20
6:
return r2
7:
r2 = r1
9:
r2 = r3
8:
r3 = r3 + 1
10:
r3 = r3 + 2
11:

Static Single Assignment

Static Single Assignment Advantages:

- Less space required to represent def-use chains. For each variable, space is proportional to uses * defs.
- Eliminates unnecessary relationships:

```
for i = 1 to N do A[i] = 0
for i = 1 to M do B[i] = 1
```

  – No reason why both loops should be forced to use same register to hold index register.

  – SSA renames second `i` to new register which may lead to better register allocation.

- SSA form make certain optimizations quick and easy → dominance property.

  – Variables have only one definition - no ambiguity.

  – Dominator information is encoded in the assignments.

SSA Dominance Property

Dominance property of SSA form: definitions dominate uses

- If x is i^{th} argument of ϕ-function in node n, then definition of x dominates i^{th} predecessor of n.
- If x is used in non-ϕ statement in node n, then definition of x dominates n.

SSA Dead Code Elimination

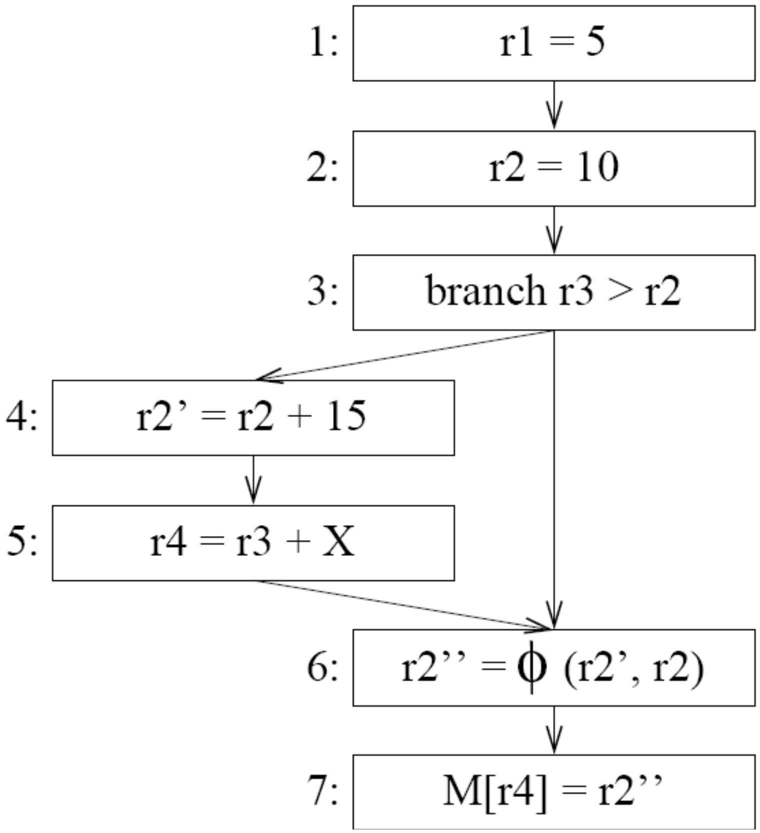
Given d: `t = x op y`

- `t` is live at end of node d if there exists path from end of d to use of `t` that does not go through definition of `t`.
- if program not in SSA form, need to perform liveness analysis to determine if `t` live at end of d.
- if program is in SSA form:
  - cannot be another definition of `t`
  - if there exists use of `t`, then path from end of d to use exists, since definitions dominate uses.
    - every use has a unique definition
    - `t` is live at end of node d if `t` is used at least once

SSA Dead Code Elimination

Algorithm:

WHILE (for each temporary `t` with no uses &&
statement defining `t` has no other side-effects) DO
delete statement definition `t`



SSA Simple Constant Propagation

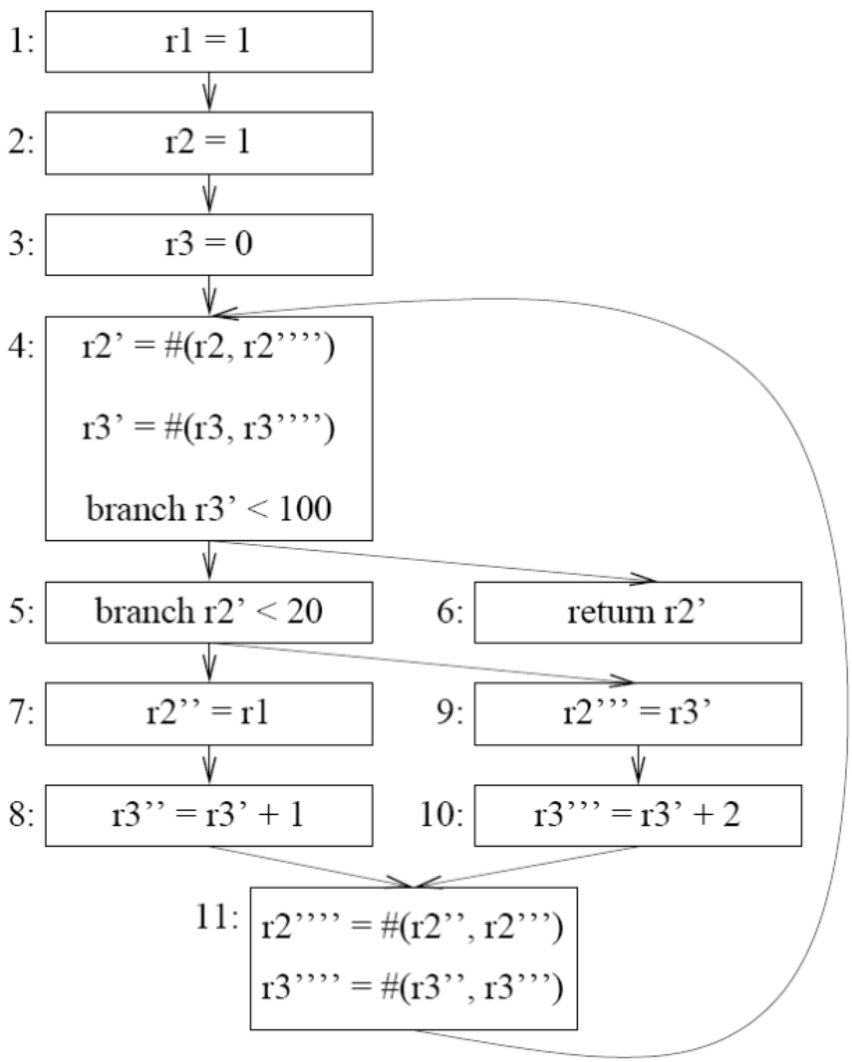
Given d: `t = c`, c is constant Given u: `x = t op b`

- if program not in SSA form:
  - need to perform reaching definition analysis
  - use of `t` in u may be replaced by `c` if d reaches u and no other definition of `t` reaches u
- if program is in SSA form:
  - d reaches u, since definitions dominate uses, and no other definition of `t` exists on path from d to u
  - d is only definition of `t` that reaches u, since it is the only definition of `t`.
    - any use of `t` can be replaced by `c`
    - any ϕ-function of form `v =` $\phi(c_1, c_2, ..., c_n)$, where $c_i = c$, can be replaced by `v = c`

SSA Simple Constant Propagation

The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

SSA Conditional Constant Propagation



- `r2` always has value of 1
- nodes 9, 10 never executed
- "simple" constant propagation algorithms assumes (through reaching definitions analysis) nodes 9, 10 may be executed.
- cannot optimize use of `r2` in node 5 since definitions 7 and 9 both reach 5.

SSA Conditional Constant Propagation

Much smarter than "simple" constant propagation:

- Does not assume a node can execute until evidence exists that it can be.
- Does not assume register is non-constant unless evidence exists that it is.

Track run-time value of each register `r` using *lattice* of values:

- $V[r] = \bot$ (bottom): compiler has seen no evidence that any assignment to `r` is ever executed.
- $V[r] = 4$: compiler has seen evidence that an assignment `r = 4` is executed, but has seen no evidence that `r` is ever assigned to another value.
- $V[r] = \top$ (top): compiler has seen evidence that `r` will have, at various times, two different values, or some value that is not predictable at compile-time.

Also:

- all registers start at bottom of lattice
- new information can only move registers up in lattice

SSA Conditional Constant Propagation

Track executability of each node in N:

- $E[N] =$ false: compiler has seen no evidence that node N can ever be executed.
- $E[N] =$ true: compiler has seen evidence that node N can be executed.

Initially:

- $V[r] = \bot$, for all registers r
- $E[s_0] =$ true, s_0 is CFG start node
- $E[N] =$ false, for all CFG nodes $N \neq s_0$

SSA Conditional Constant Propagation

Algorithm: apply following conditions until no more changes occur to E or V values:

1. Given: register `r` with no definition (formal parameter, uninitialized).
   Action: $V[r] = \top$
2. Given: executable node B with only one successor C
   Action: $E[C] = \text{true}$
3. Given: executable assignment `r = x op y`, $V[x] = c_1$ and $V[y] = c_2$
   Action: $V[r] = c_1 \text{op} c_2$
4. Given: executable assignment `r = x op y`, $V[x] = \top$ or $V[y] = \top$
   Action: $V[r] = \top$
5. Given: executable assignment `r =` $\phi(x_1, x_2, ..., x_n)$, $V[x_i] = c_1$, $V[x_j] = c_2$, and predecessors i and j are executable
   Action: $V[r] = \top$

SSA Conditional Constant Propagation

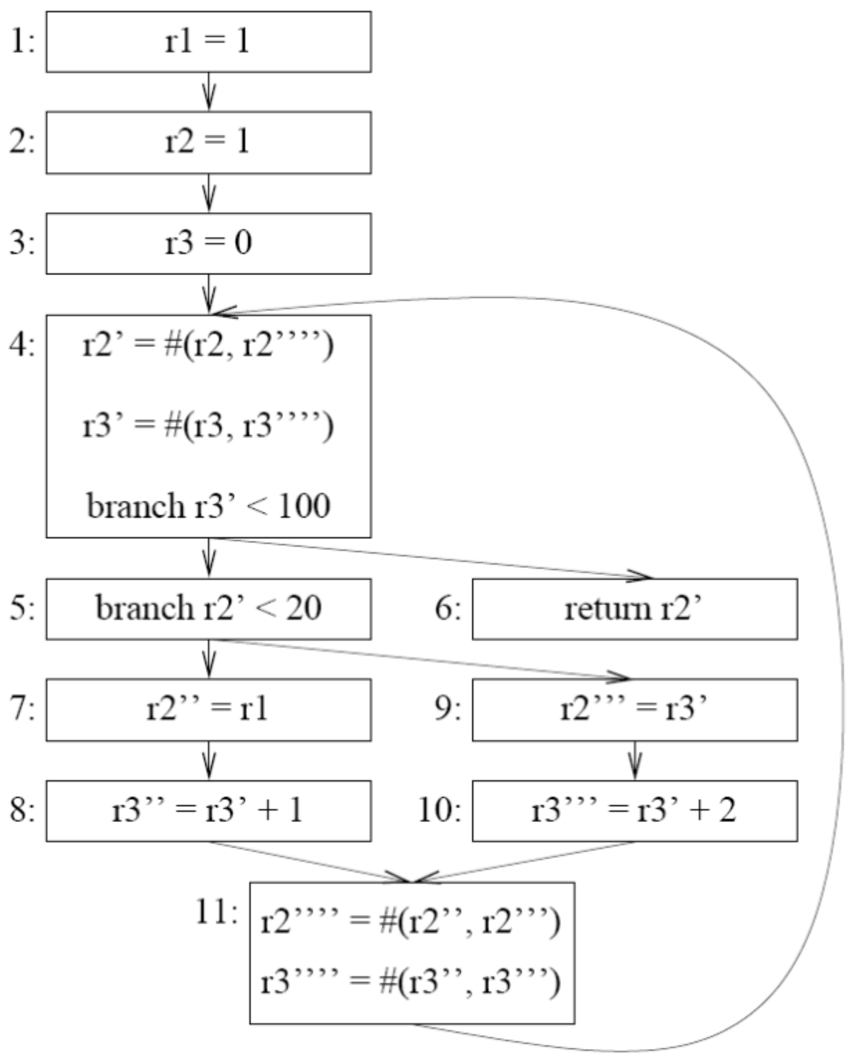
6. Given: executable assignment `r = M[..]` or `r = f(..)`
   Action: $V[r] = \top$
7. Given: executable assignment `r` $= \phi(x_1, x_2, ..., x_n)$, $V[x_i] = \top$, and predecessor i is executable
   Action: $V[r] = \top$
8. Given: executable assignment `r` $= \phi(x_1, x_2, ..., x_n)$, $V[x_i] = c_i$, and predecessor i is executable; and for all $j \neq i$ predecessor j is not executable, or $V[x_j] = \bot$, or $V[x_j] = c_i$
   Action: $V[r] = c_i$
9. Given: executable branch `branch x bop y, L1 (else L2)`, $V[x] = \top$ or $V[y] = \top$
   Action: $E[L1] =$ true, $E[L2] =$ true
10. Given: executable branch `branch x bop y, L1 (else L2)`, $V[x] = c_1$ and $V[y] = c_2$
    Action: $E[L1] =$ true *OR* $E[L2] =$ true depending on c_1 bop c_2.

SSA Conditional Constant Propagation

Given V, E values, program can be optimized as follows:

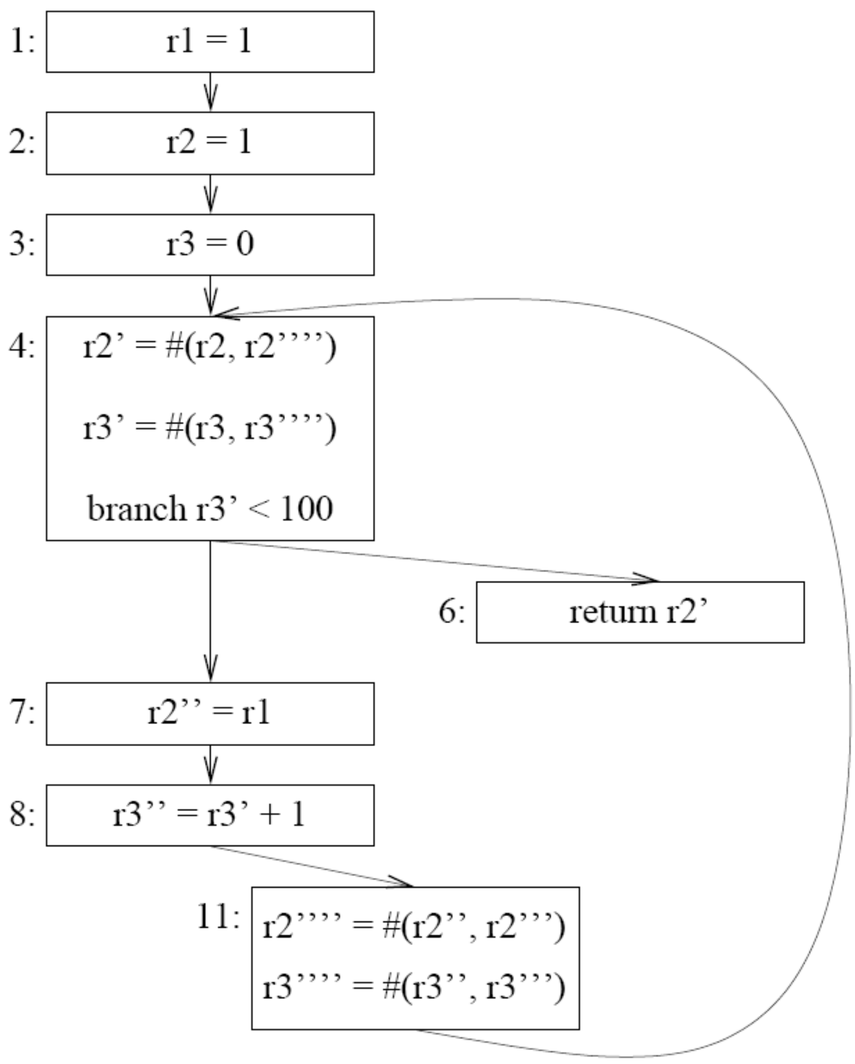
- if $E[B] =$ false, delete node B form CFG.
- if $V[r] = c$, replace each use of `r` by c, delete assignment to r.

SSA Conditional Constant Propagation Example

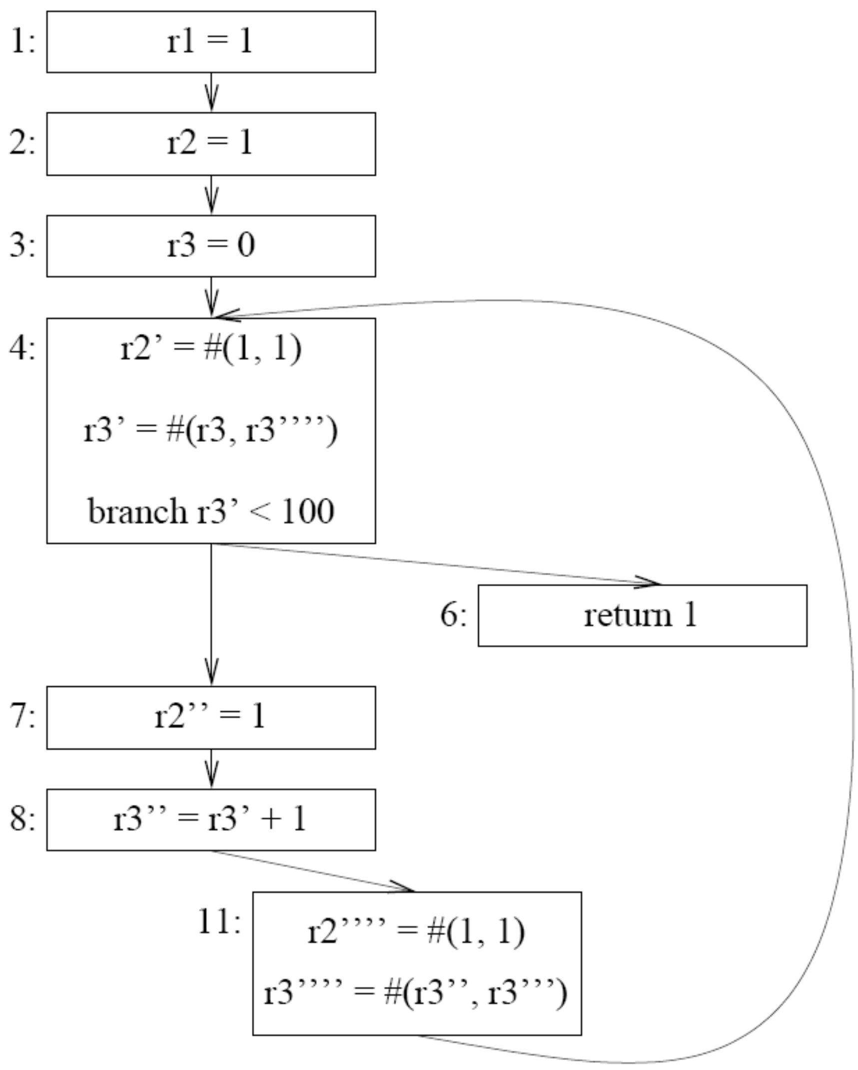


N	$E[N]$	r	$V[r]$
1	t	r1	$\perp$
2	f	r2	$\perp$
3	f	r2’	$\perp$
4	f	r2’’	$\perp$
5	f	r2’’’	$\perp$
6	f	r2’’’’	$\perp$
7	f	r3	$\perp$
8	f	r3’	$\perp$
9	f	r3’’	$\perp$
10	f	r3’’’	$\perp$
11	f	r3’’’’	$\perp$

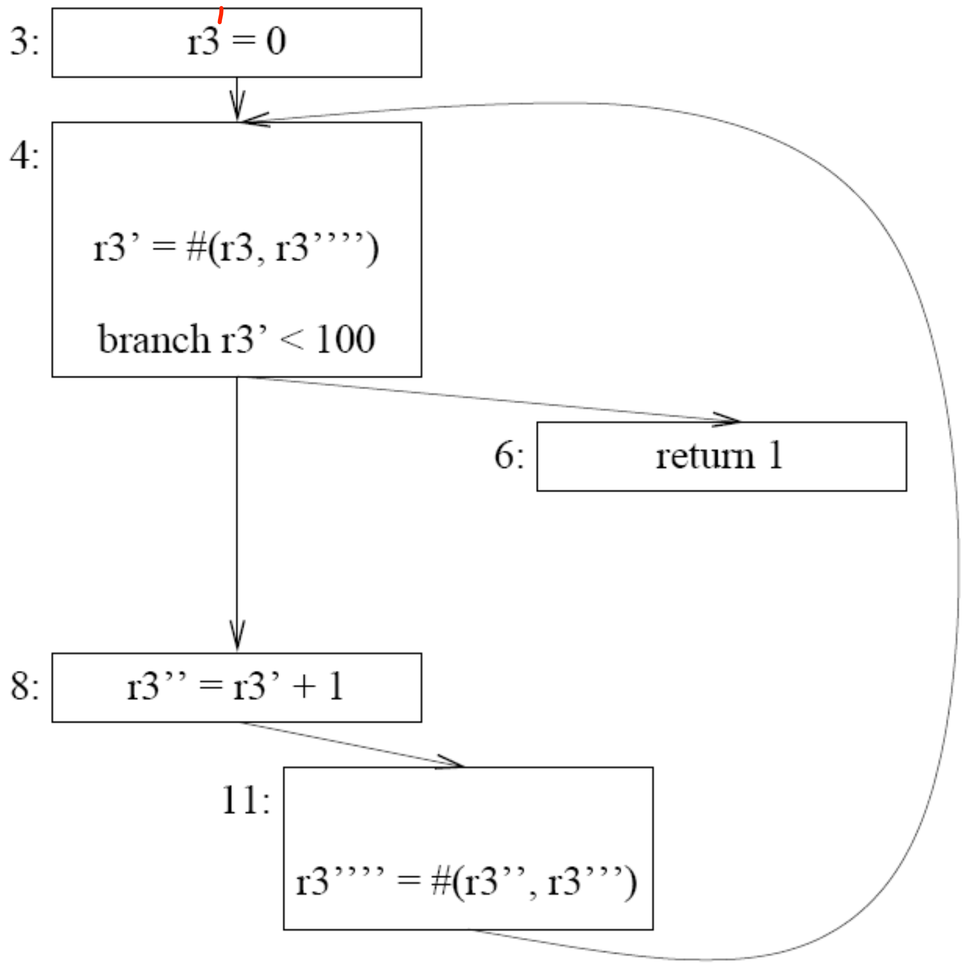
SSA Conditional Constant Propagation Example



SSA Conditional Constant Propagation Example



SSA Conditional Constant Propagation Example



SSA Conditional Constant Propagation Example