Compiler Optimisation

4 – Dataflow Analysis

Hugh Leather
IF 1.18a
hleather@inf.ed.ac.uk

Institute for Computing Systems Architecture
School of Informatics
University of Edinburgh

2019

Introduction

This lecture:

- Data flow termination
- More data flow examples
- Dominance
- Static single-assignment form

Data flow termination

Fixed Point

A fixed point, x, of function $f : T \to T$ is when $f(x) = x$

Partial ordering

A binary relation, $\leq$, among elements of a set, S, that is:

$$\begin{array}{rlr} \forall a \in S, & a \leq a & \text{reflexive} \\ \forall a, b \in S, & a \leq b \wedge b \leq a \implies a = b & \text{antisymmetric} \\ \forall a, b, c \in S & a \leq b \wedge b \leq c \implies a \leq c & \text{transitive} \end{array}$$

Partially ordered set (poset)

A set with a partial order

Data flow termination

Join semi-lattice

Partially ordered set that has a join (a least upper bound) for any nonempty finite subset

$$\forall x, y, z \in V$$

$$x \wedge (y \wedge z) = (x \wedge y) \wedge z \qquad \text{Associativity}$$

$$x \wedge y = y \wedge x \qquad \text{Commutativity}$$

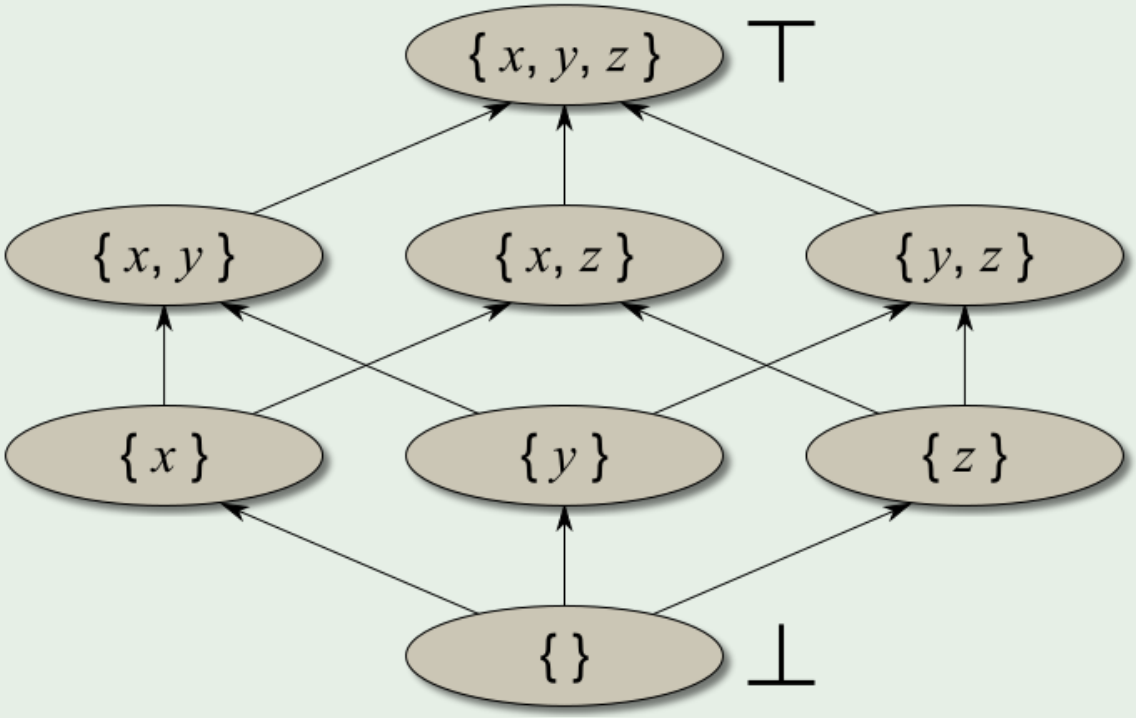
$$x \wedge x = x \qquad \text{Idempotency}$$

$$\top \wedge x = x \qquad \text{Top element}$$

Note, *meet semi-lattice* has $\bot$and *complete lattice* has both
Often just use 'semi-lattice'

Data flow termination
Lattices

Example lattice

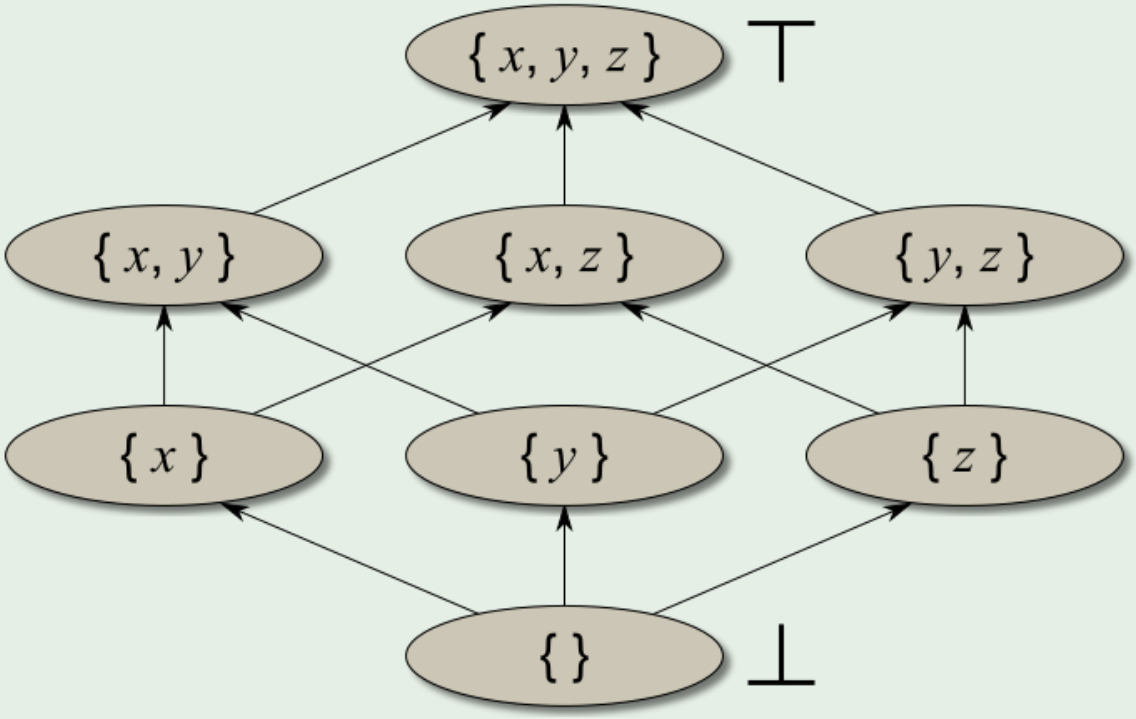


Hasse diagram for the set of all subsets of x, y, z, ordered by inclusion

Is it a lattice?

Data flow termination
Lattices

Example lattice

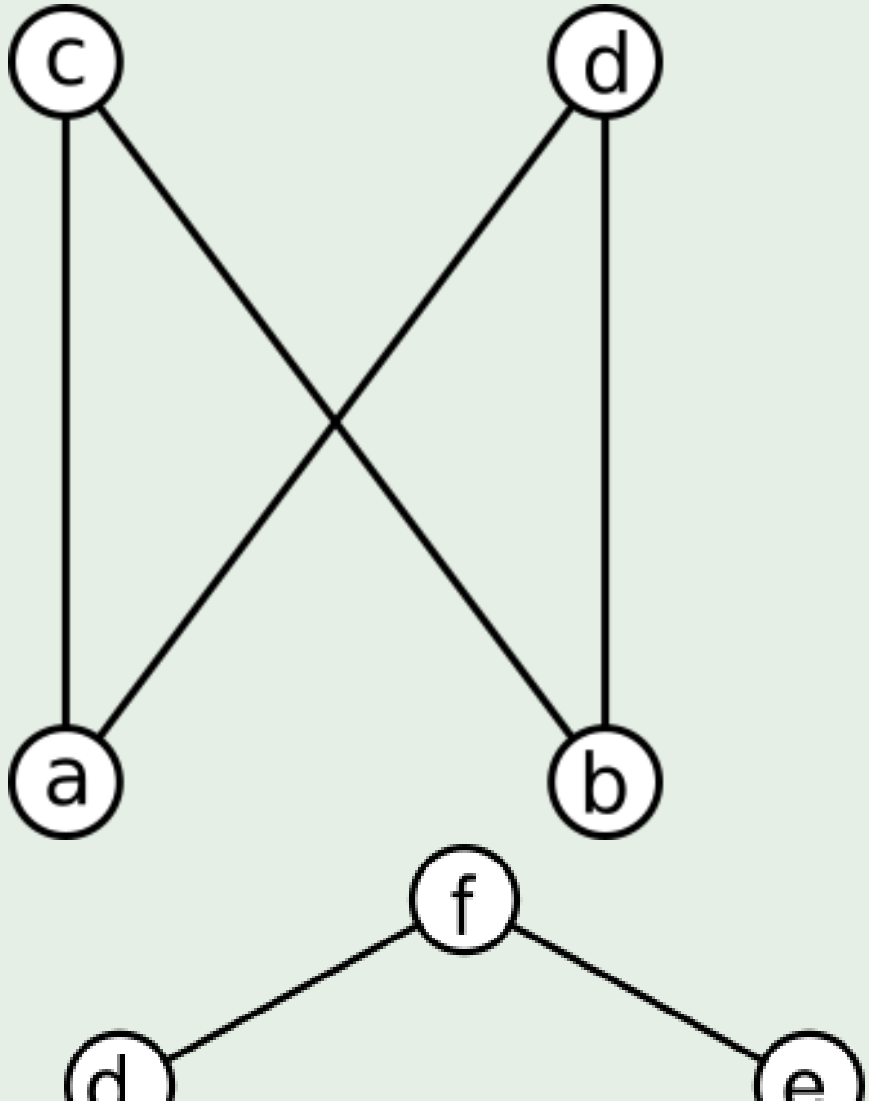


Hasse diagram for the set of all subsets of x, y, z, ordered by inclusion

Yes, and is lattice for LiveOut, with variables, x, y, z

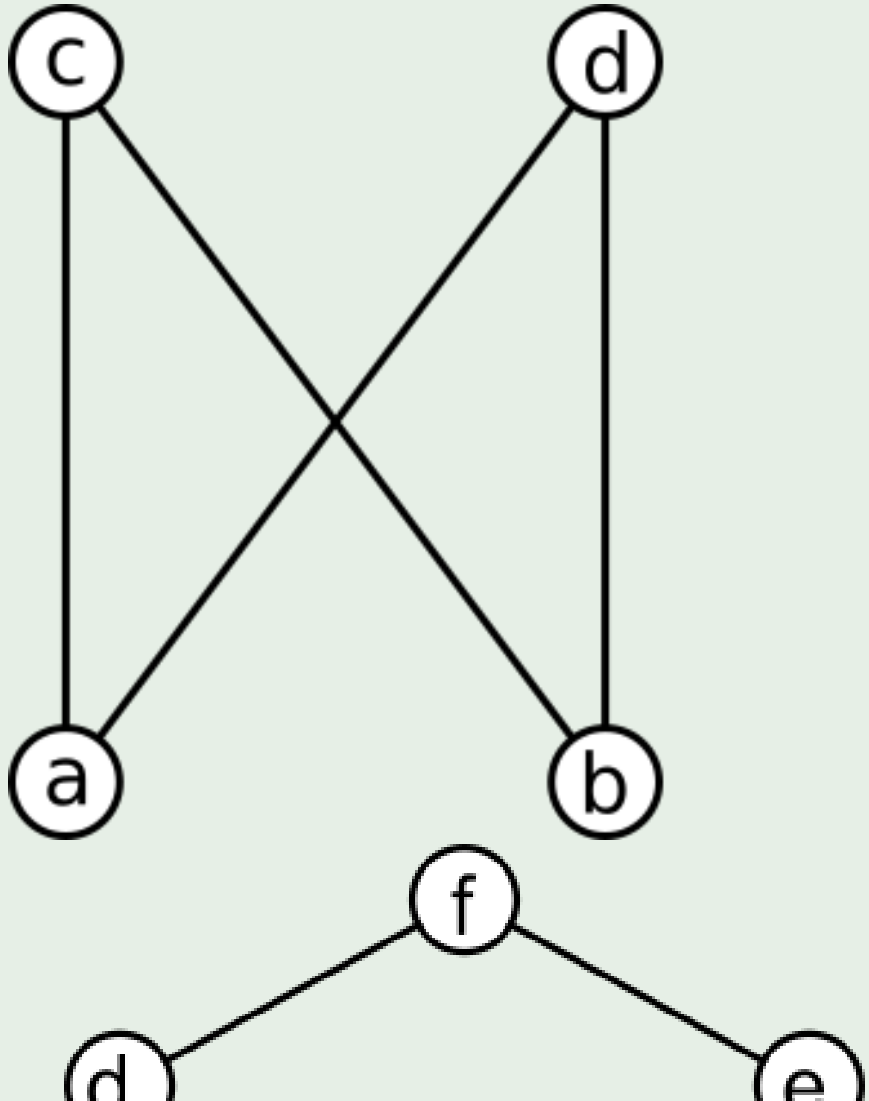
Data flow termination
Lattices

Example non-lattices



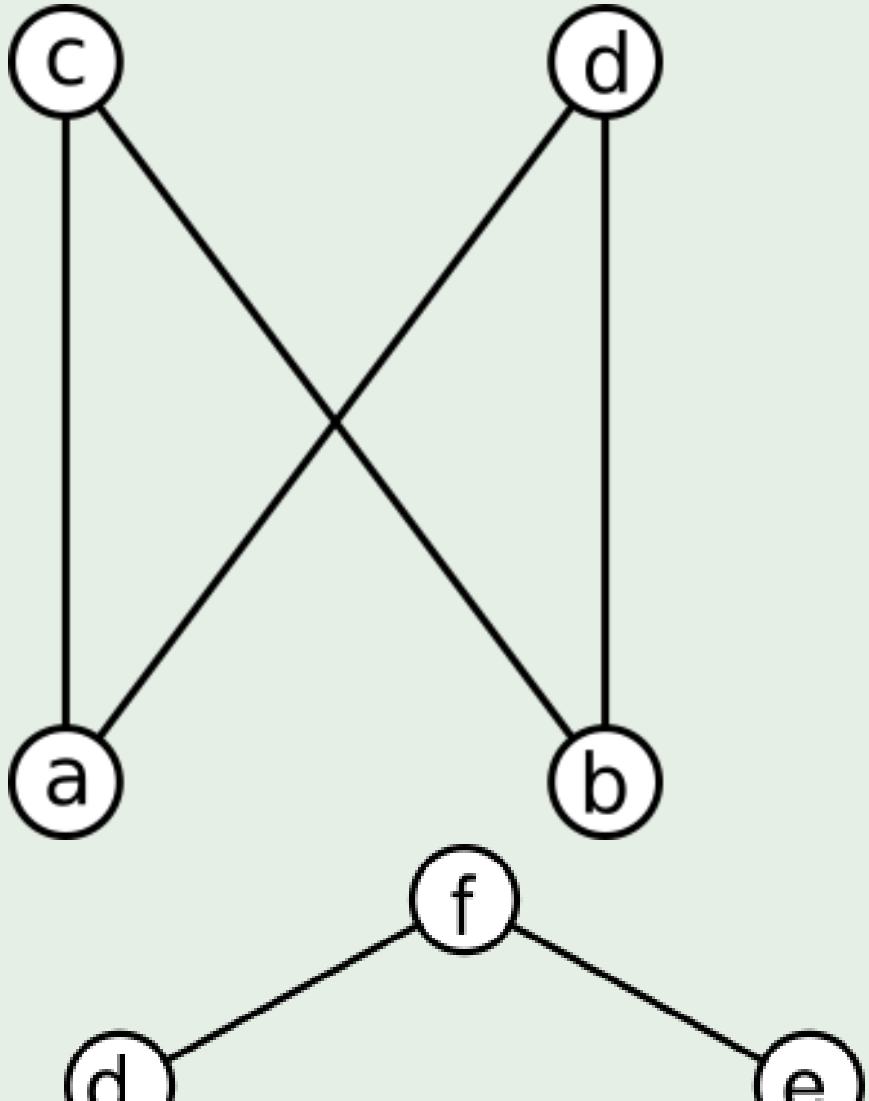
Data flow termination
Lattices

Example non-lattices



Data flow termination
Lattices

Example non-lattices



Data flow termination

Each block or statement has a particular function, e.g. based on *Kill* and *Gen* sets

Let F be the set of transfer functions

Sufficient termination constraints

V and $\wedge$ for a semi-lattice

F has identity: $I(x) = x, \forall x \in V$

F is closed under composition: $\forall f, g \in F, h = f \circ g \in F$

F is monotonic: $\forall f \in F, f(x \wedge y) \leq f(x) \wedge f(y)$

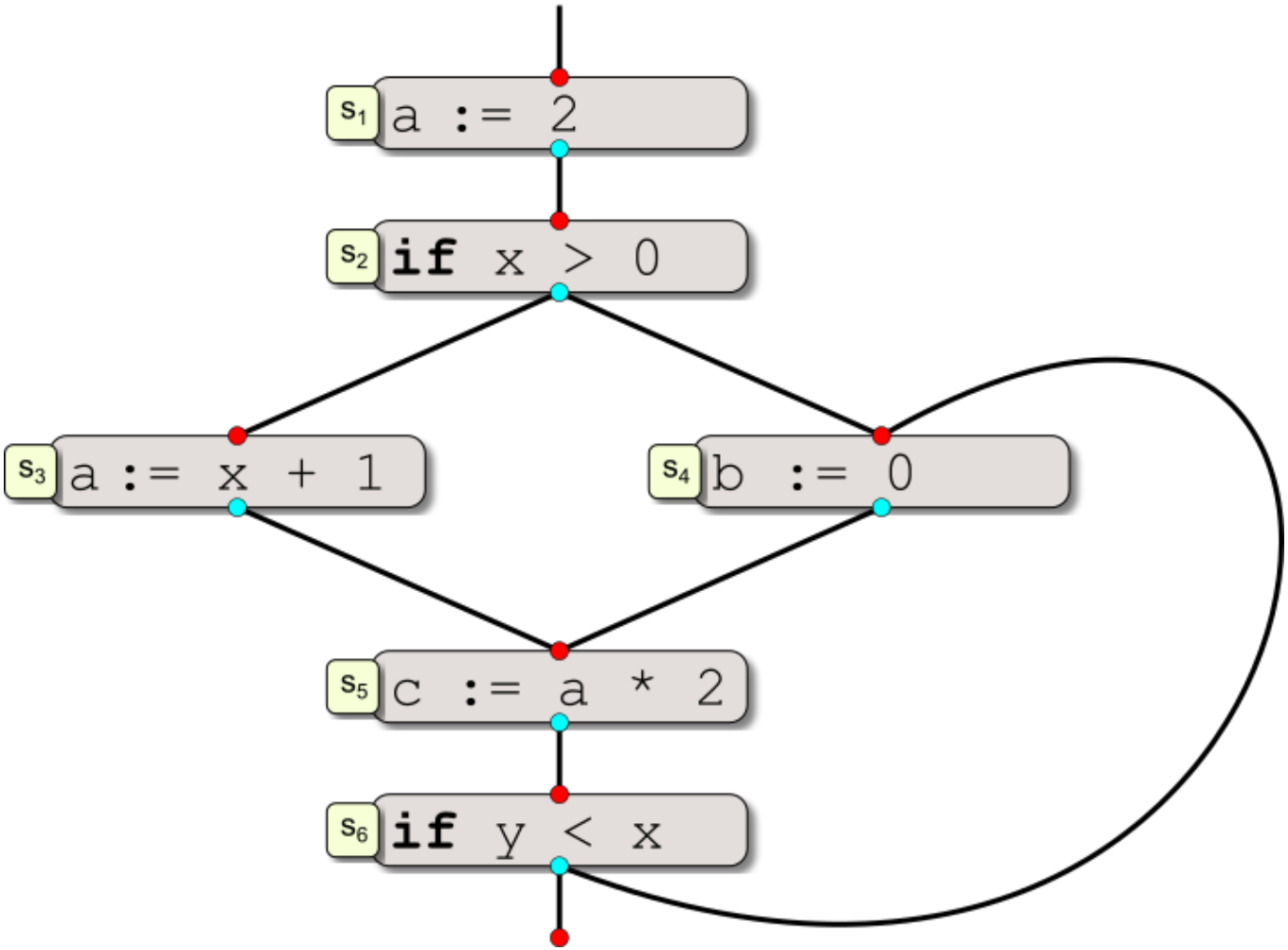
See 10.11

Liveness

- A variable $\mathtt{v}$ is **live-out** of statement $\mathtt{s}$ if $\mathtt{v}$ is used along some control path starting at $\mathtt{s}$
- Otherwise, we say that $\mathtt{v}$ is **dead**
- A variable is live if it holds a value that may be needed in the future

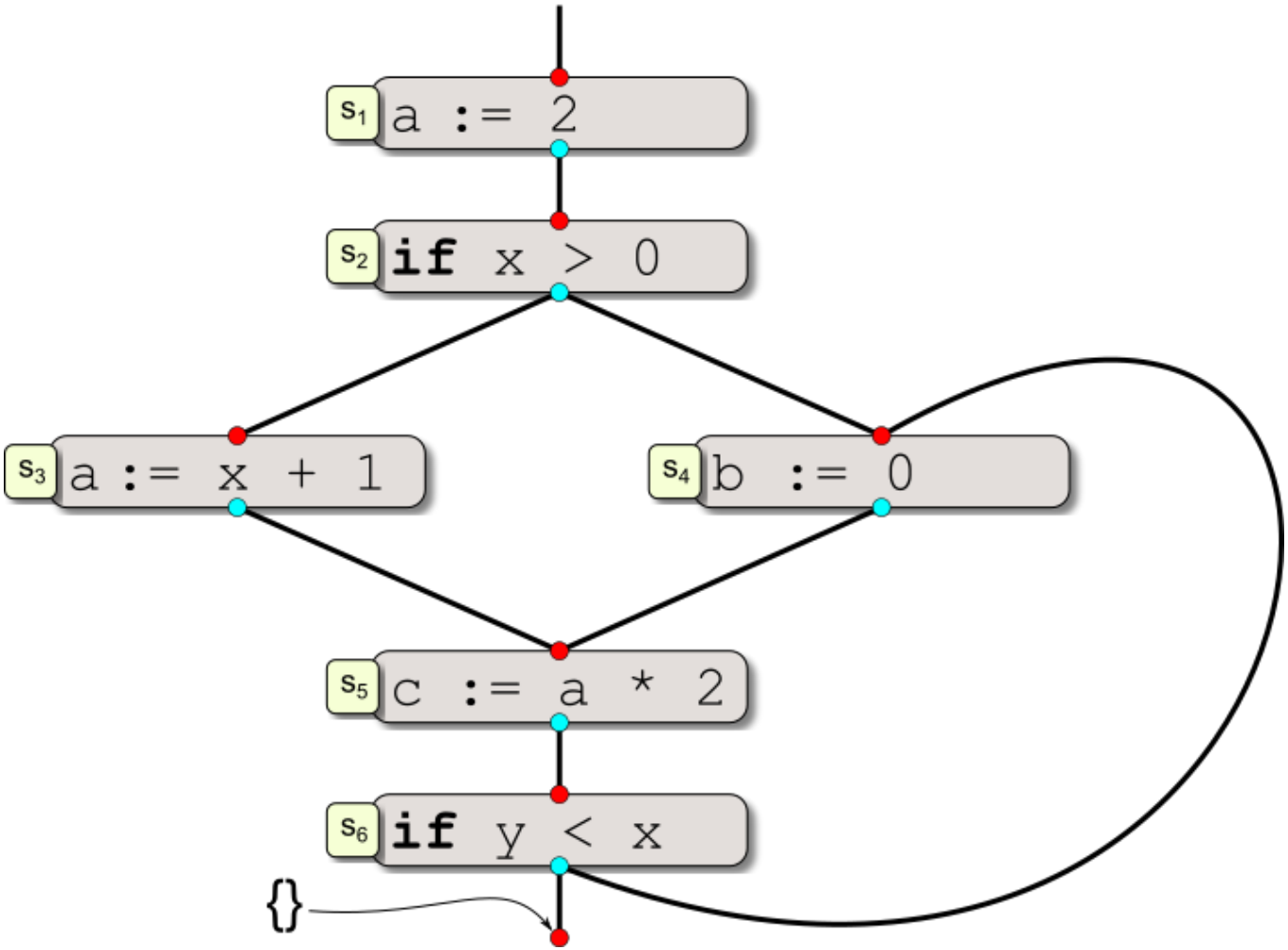
Information flows *backwards* from statement to predecessors
Liveness useful for optimisations (e.g. register allocation, store elimination, dead code...)

Liveness



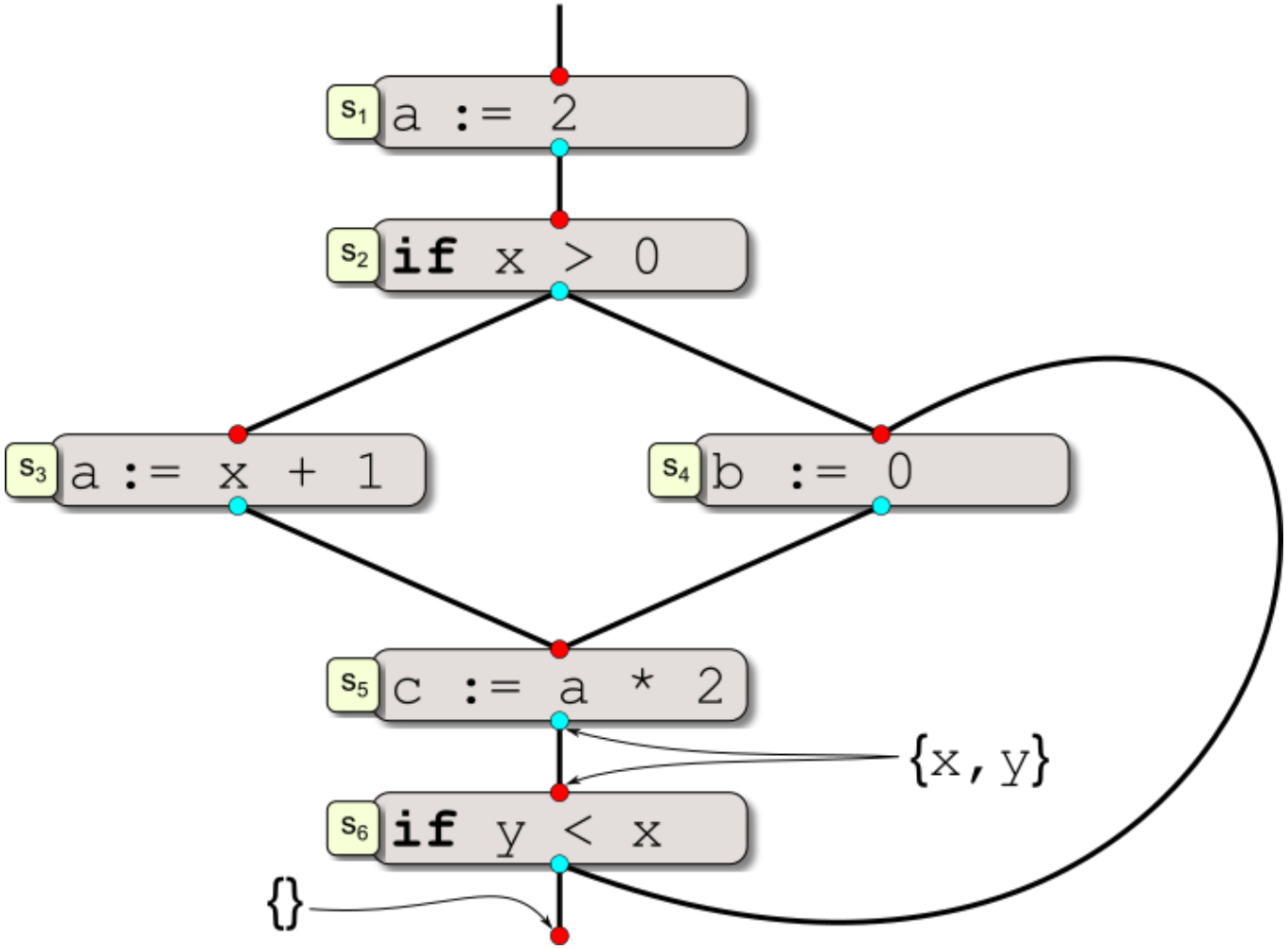
A variable v is live-out of statement s if v is used along some control path starting at s

Liveness



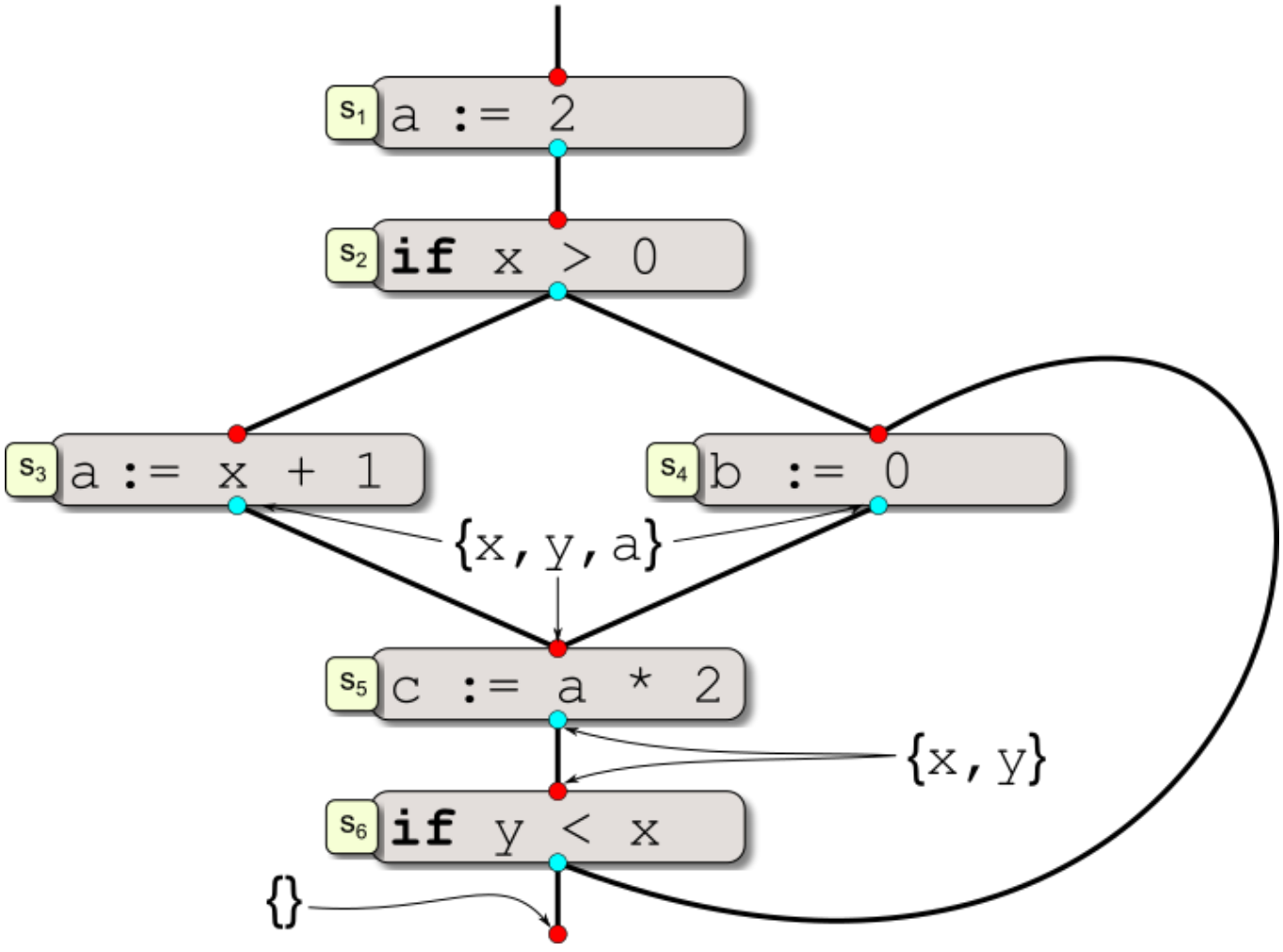
A variable v is live-out of statement s if v is used along some control path starting at s

Liveness



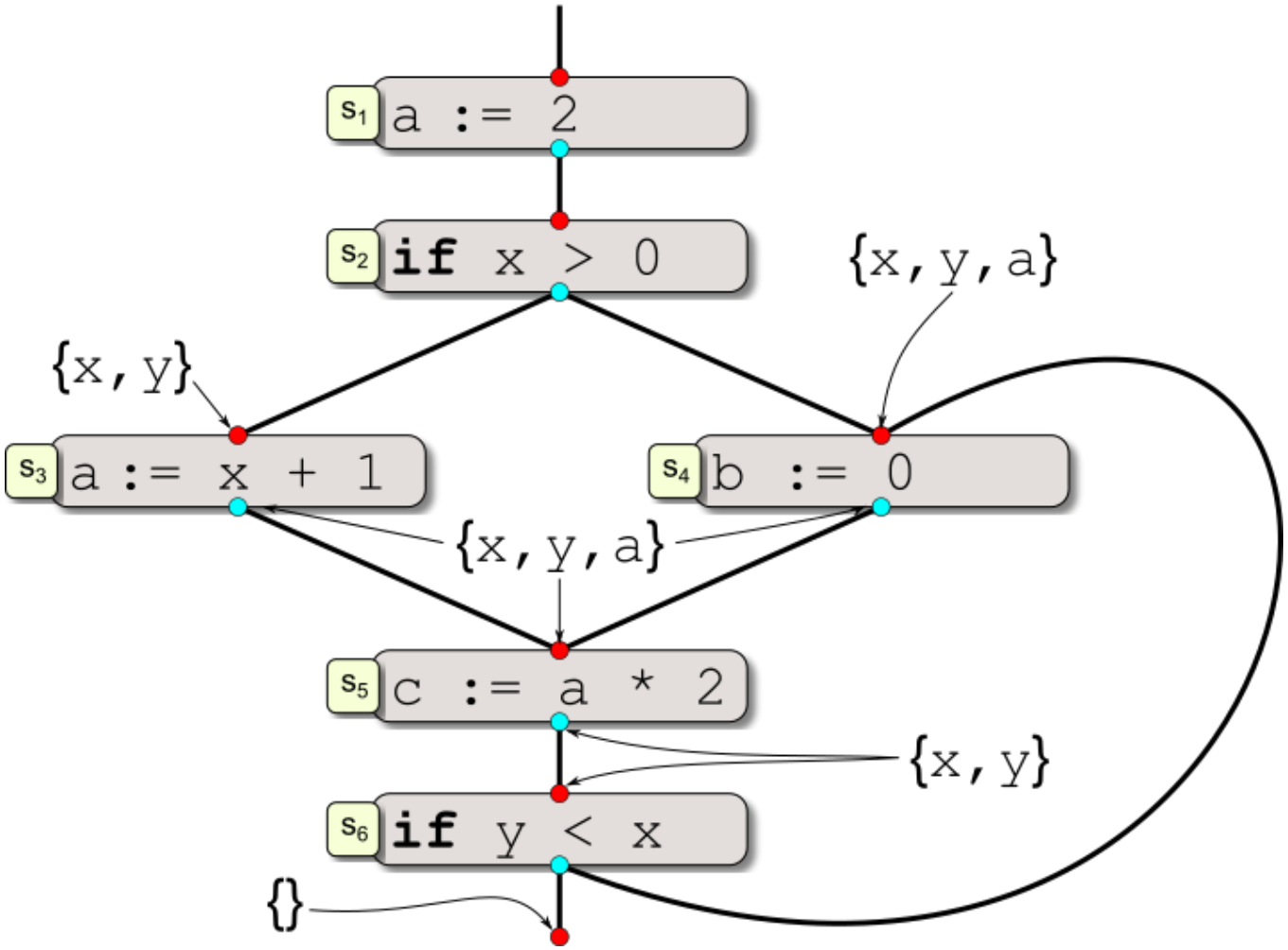
A variable v is live-out of statement s if v is used along some control path starting at s

Liveness



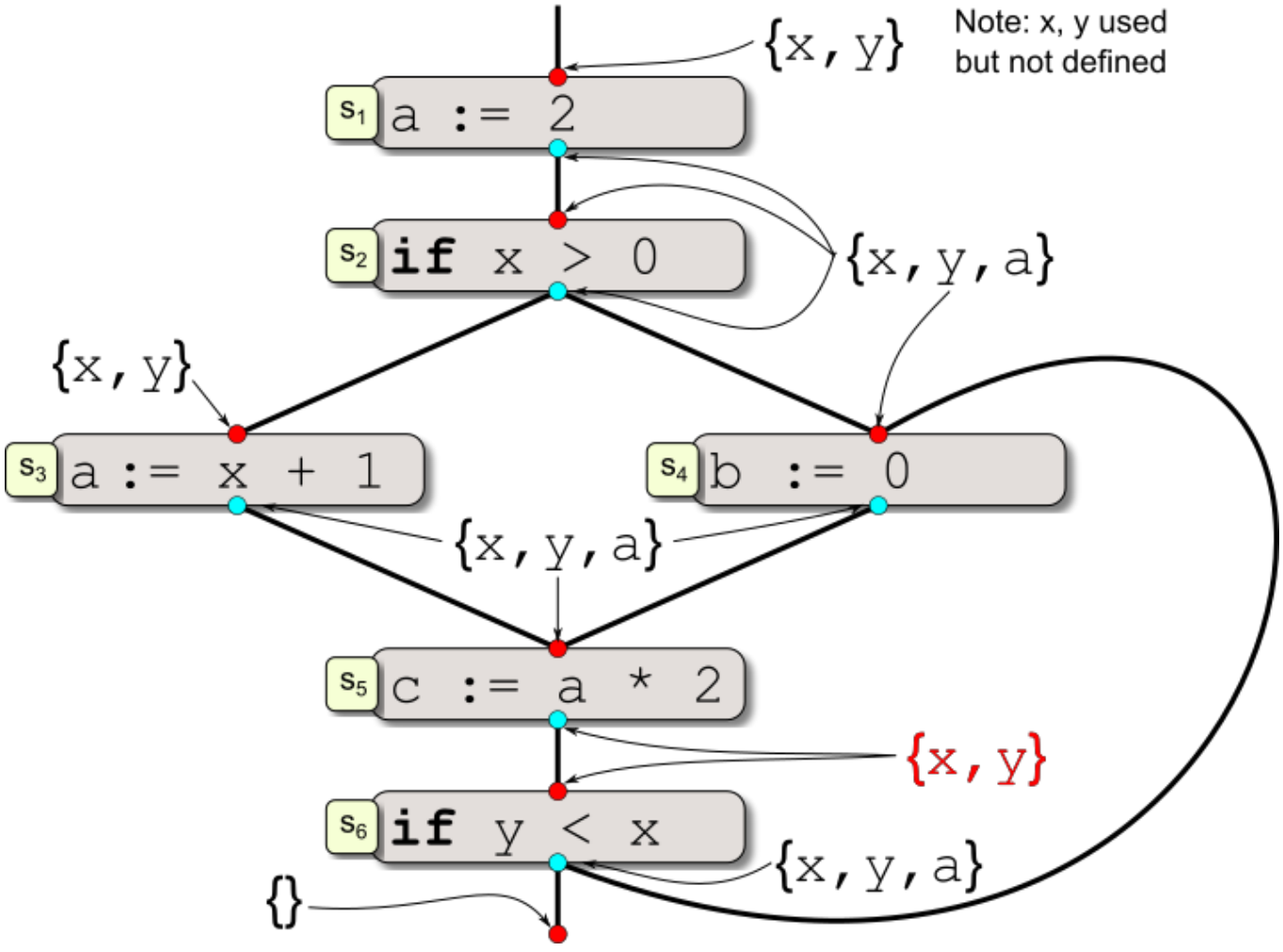
A variable v is live-out of statement s if v is used along some control path starting at s

Liveness



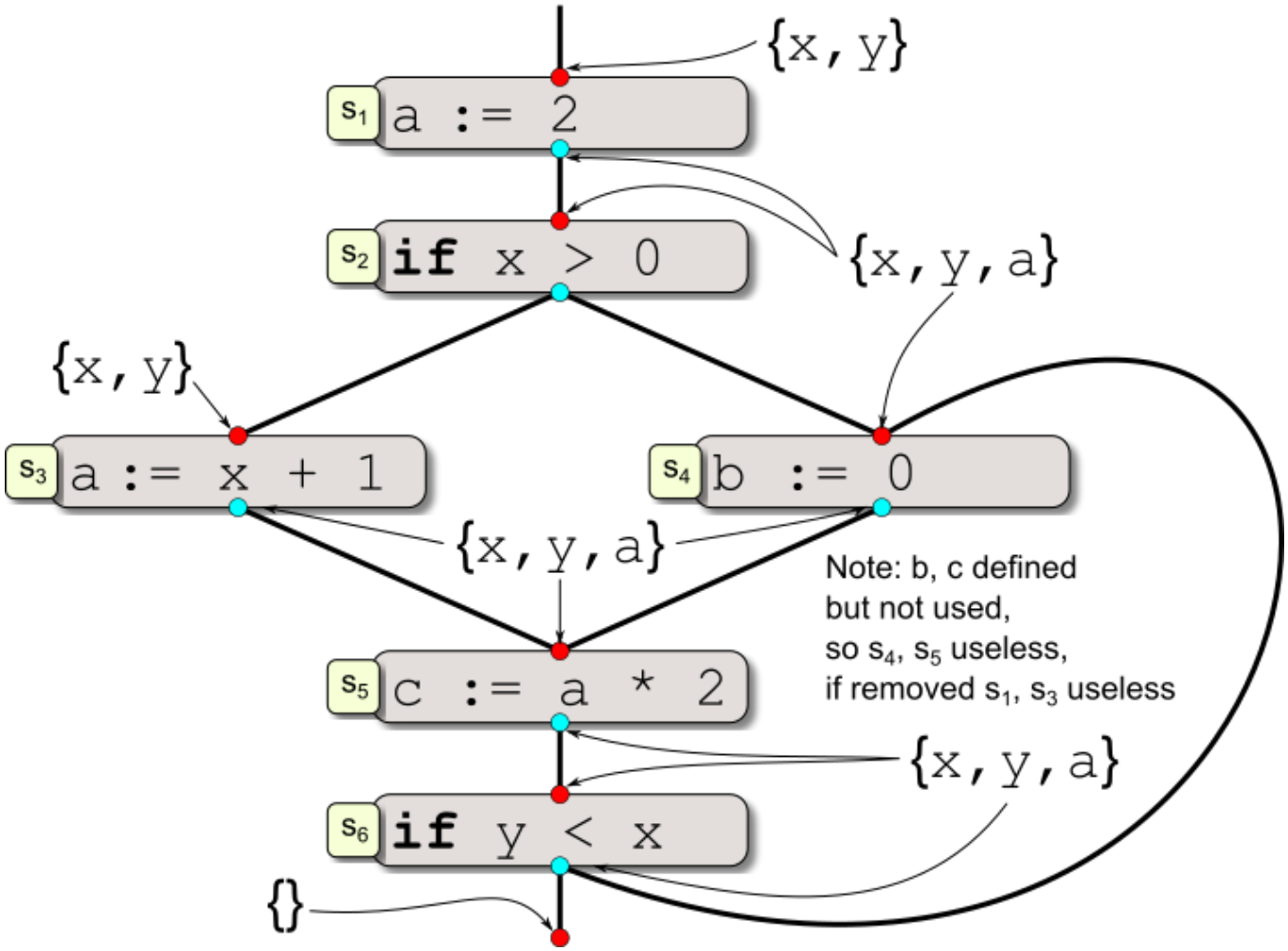
A variable v is live-out of statement s if v is used along some control path starting at s

Liveness



A variable v is live-out of statement s if v is used along some control path starting at s

Liveness



A variable v is live-out of statement s if v is used along some control path starting at s

Liveness

- Live variables come up from their successors using them
  $Out(s) = \bigcup_{\forall n \in Succ(s)} In(n)$
- Transfer back across the node
  $In(s) = Out(s) - Kill(s) \cup Gen(s)$
- Used variables are live
  $Gen(s) = \{u \text{ such that } u \text{ is used in } s\}$
- Defined but not used variables are killed
  $Kill(s) = \{d \text{ such that } d \text{ is defined in } s \text{ but not used in } s\}$
- If we don't know, start with empty
  $Init(s) = \varnothing$

Others

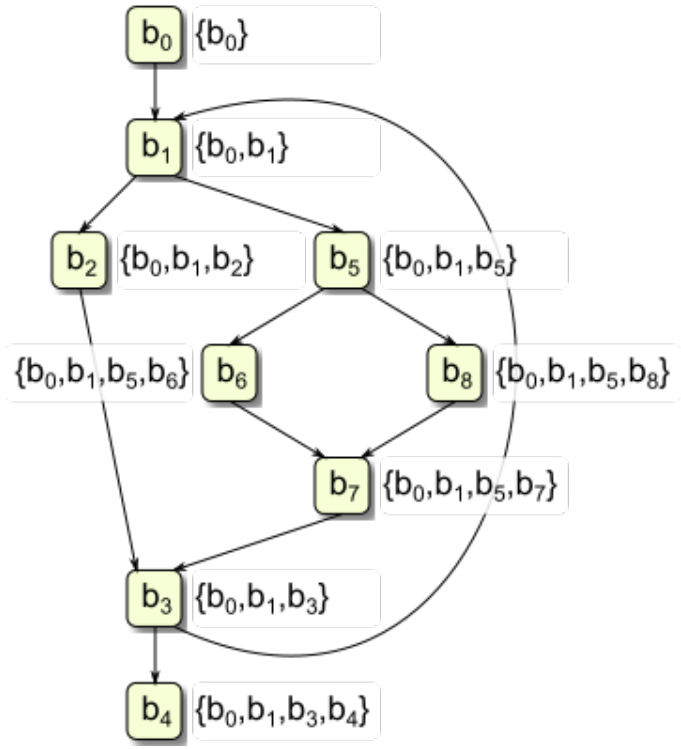
- Constant propagation - show variable has same constant value at some point
  - Strictly speaking does not compute expressions except `x := const`, or `x := y` and `y` is constant
  - Often combined with constant folding that computes expressions
- Copy propagation - show variable is copy of other variable
- Available expressions - set of expressions reaching by all paths
- Very busy expressions - expressions evaluated on all paths leaving block - for code hoisting
- Definite assignment - variable always assigned before use
- Redundant expressions, and partial redundant expressions
- Many more - read about them!

Dominators

CFG node b_i dominates b_j, written $b_i \gg b_j$,
iff every path from the start node to b_j goes through b_i

Design data flow equations to compute which nodes dominate each node

What direction?
What value set?
What transfer?
What Meet?
Initial values?

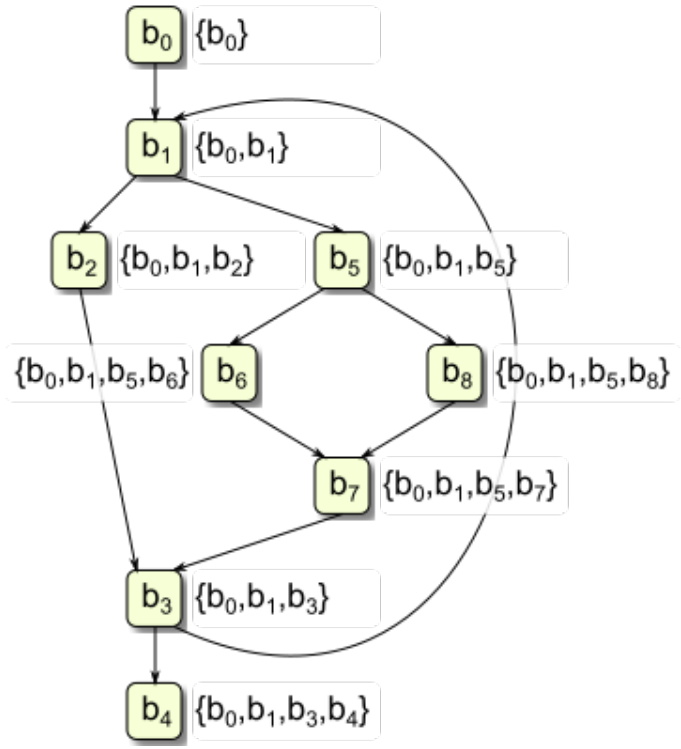


Dominators

CFG node b_i dominates b_j, written $b_i \gg b_j$,
iff every path from the start node to b_j goes through b_i

Design data flow equations to compute which nodes dominate each node

What direction?
What value set?
What transfer?
What Meet?
Initial values?

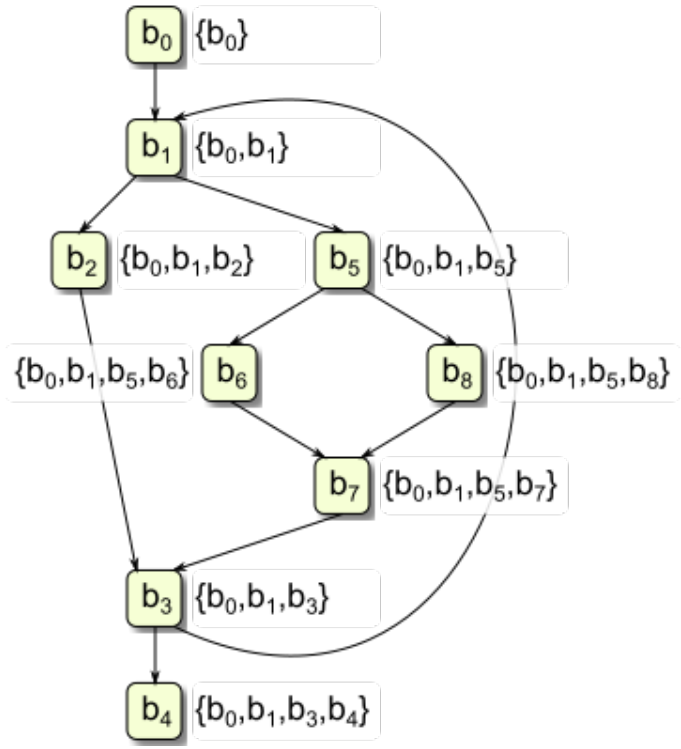


Dominators

CFG node b_i dominates b_j, written $b_i \gg b_j$,
iff every path from the start node to b_j goes through b_i

Design data flow equations to compute which nodes dominate each node

Direction: Forward
What value set?
What transfer?
What Meet?
Initial values?

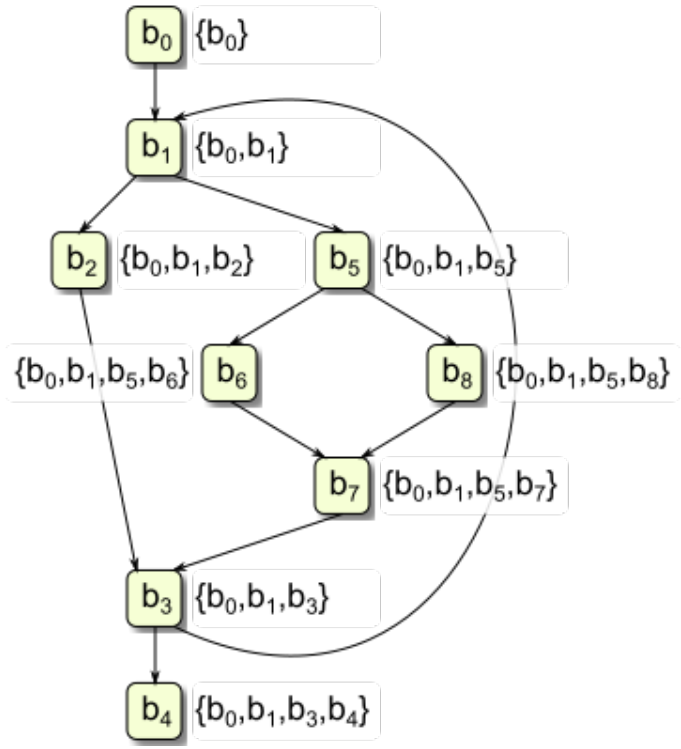


Dominators

CFG node b_i dominates b_j, written $b_i \gg b_j$,
iff every path from the start node to b_j goes through b_i

Design data flow equations to compute which nodes dominate each node

Direction: Forward
What value set?
What transfer?
What Meet?
Initial values?

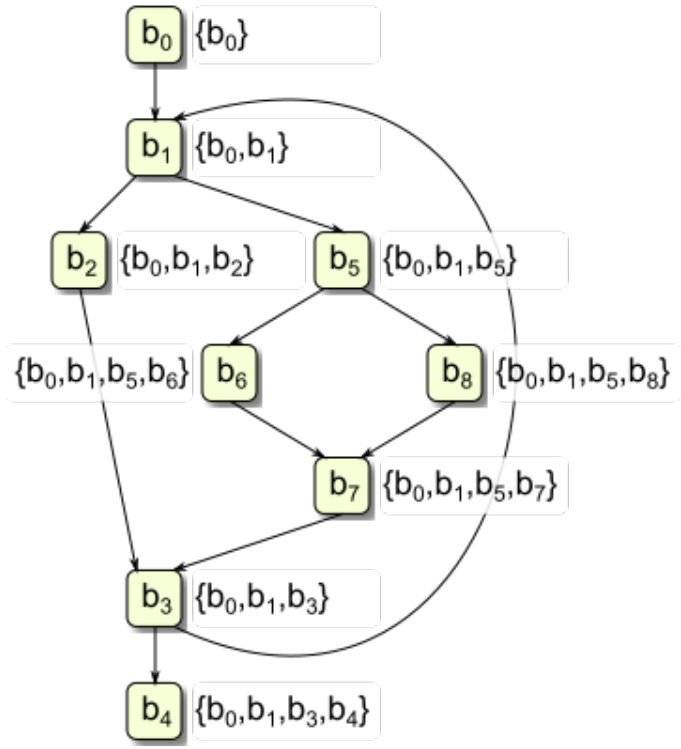


Dominators

CFG node b_i dominates b_j, written $b_i \gg b_j$,
iff every path from the start node to b_j goes through b_i

Design data flow equations to compute which nodes dominate each node

Direction: Forward
Values: Sets of nodes
What transfer?
What Meet?
Initial values?

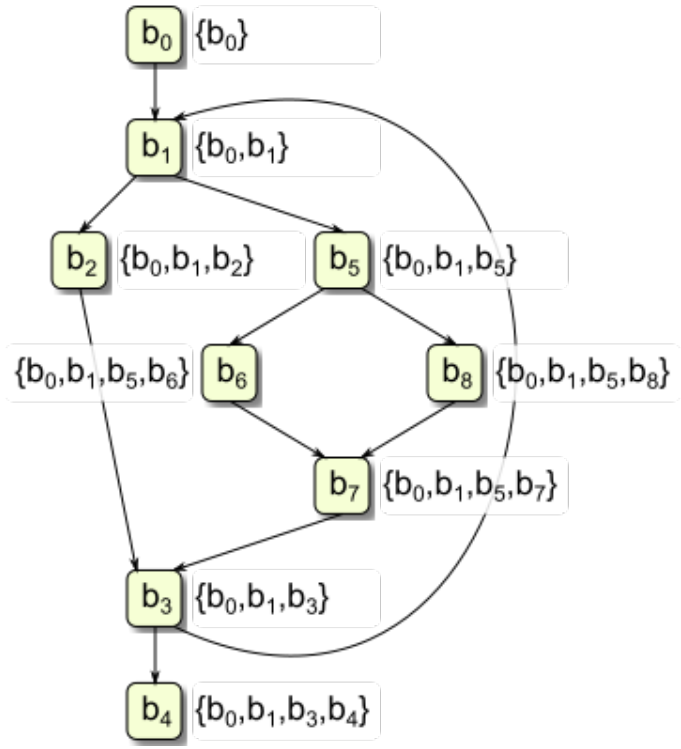


Dominators

CFG node b_i dominates b_j, written $b_i \gg b_j$,
iff every path from the start node to b_j goes through b_i

Design data flow equations to compute which nodes dominate each node

Direction: Forward
Values: Sets of nodes
What transfer?
What Meet?
Initial values?

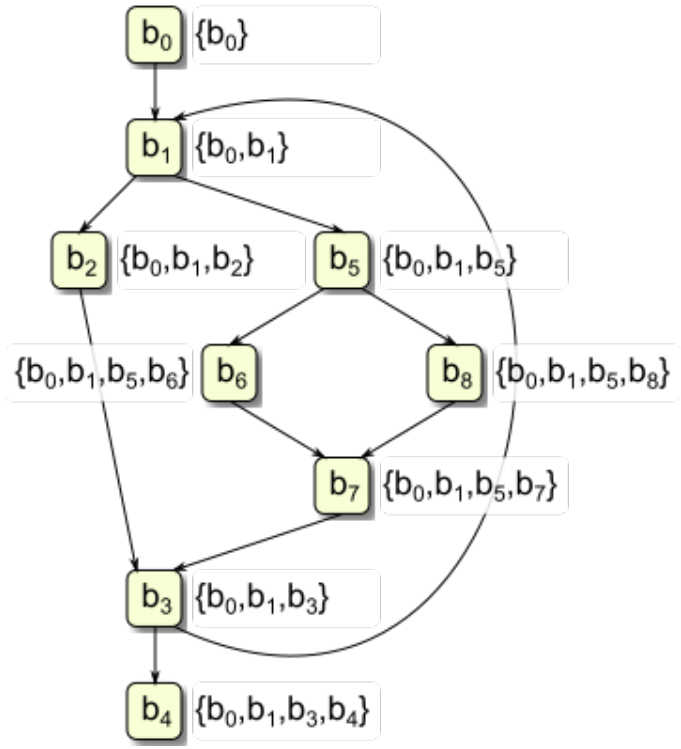


Dominators

CFG node b_i dominates b_j, written $b_i \gg b_j$,
iff every path from the start node to b_j goes through b_i

Design data flow equations to compute which nodes dominate each node

Direction: Forward
Values: Sets of nodes
Transfer: $Out(n) = In(n) \cup \{n\}$
What Meet?
Initial values?

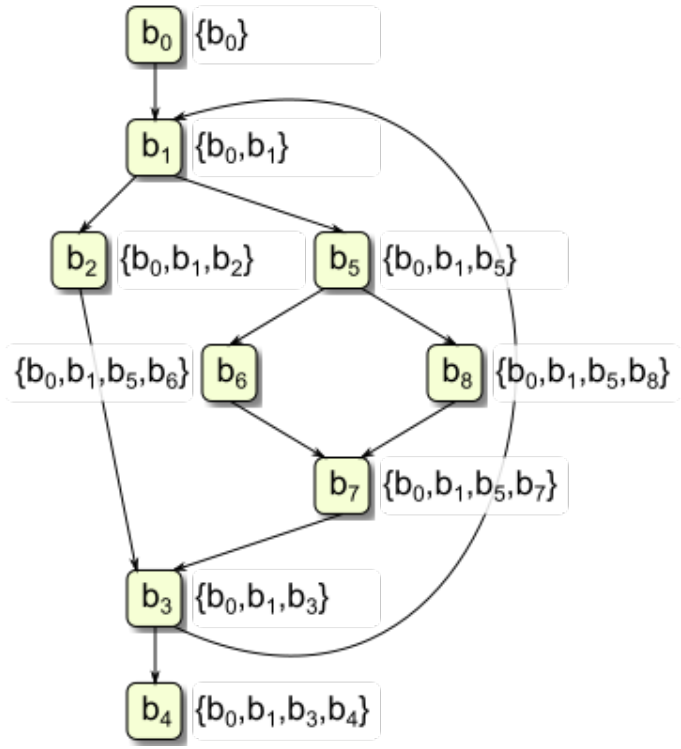


Dominators

CFG node b_i dominates b_j, written $b_i \gg b_j$,
iff every path from the start node to b_j goes through b_i

Design data flow equations to compute which nodes dominate each node

Direction: Forward
Values: Sets of nodes
Transfer: $Out(n) = In(n) \cup \{n\}$
What Meet?
Initial values?

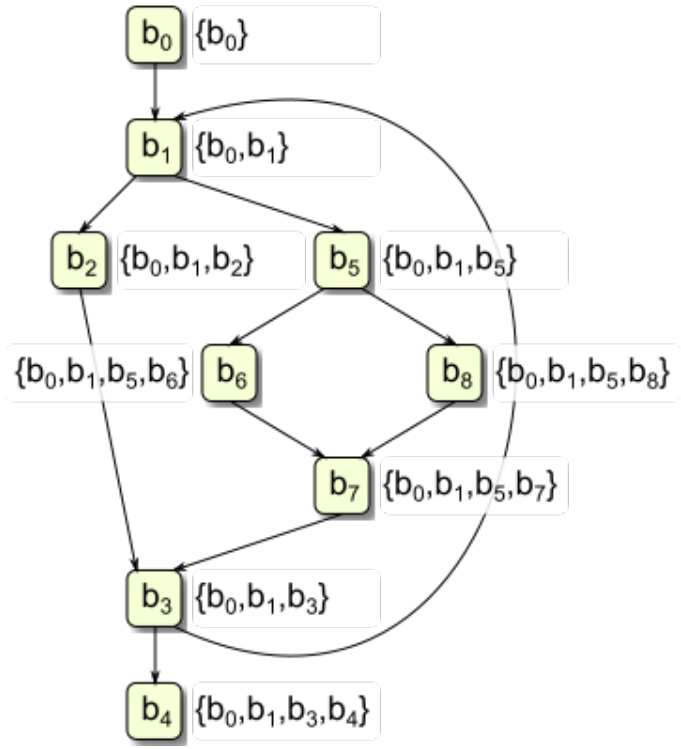


Dominators

CFG node b_i dominates b_j, written $b_i \gg b_j$,
iff every path from the start node to b_j goes through b_i

Design data flow equations to compute which nodes dominate each node

Direction: Forward
Values: Sets of nodes
Transfer: $Out(n) = In(n) \cup \{n\}$
Meet: $In(n) = \bigcap_{\forall n \in Pred(s)} Out(s)$
Initial values?



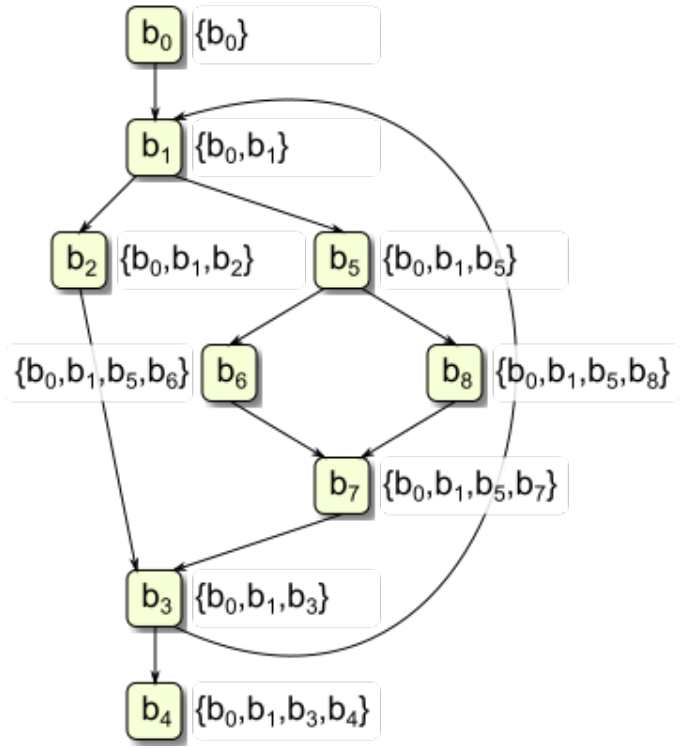
Dominators

CFG node b_i dominates b_j, written $b_i \gg b_j$,
iff every path from the start node to b_j goes through b_i

Design data flow equations to compute which nodes dominate each node

Direction: Forward
Values: Sets of nodes
Transfer: $Out(n) = In(n) \cup \{n\}$
Meet: $In(n) = \bigcap_{\forall n \in Pred(s)} Out(s)$

Initial values?

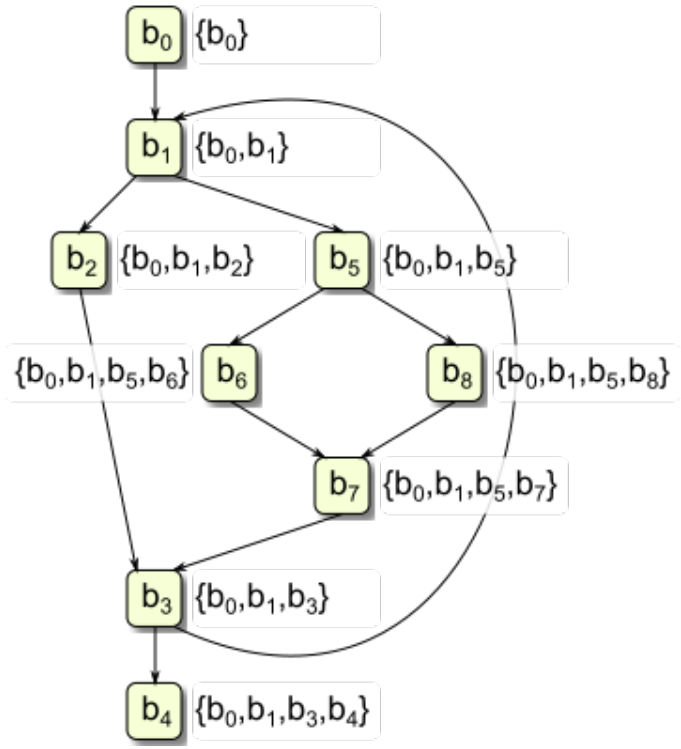


Dominators

CFG node b_i dominates b_j, written $b_i \gg b_j$,
iff every path from the start node to b_j goes through b_i

Design data flow equations to compute which nodes dominate each node

Direction: Forward
Values: Sets of nodes
Transfer: $Out(n) = In(n) \cup \{n\}$
Meet: $In(n) = \bigcap_{\forall n \in Pred(s)} Out(s)$
Initial: $Init(n_0) = \{n_0\}$; $Init(n) =$ all



Dominators

Post-dominator

Node z is said to post-dominate a node n if all paths to the exit node of the graph starting at n must go through z

Strict dominance

Node a strictly dominates b iff $a \gg b \wedge a \neq b$

Immediate dominator

$idom(n)$ strictly dominates n but not any other node that strictly dominates n

Dominator tree

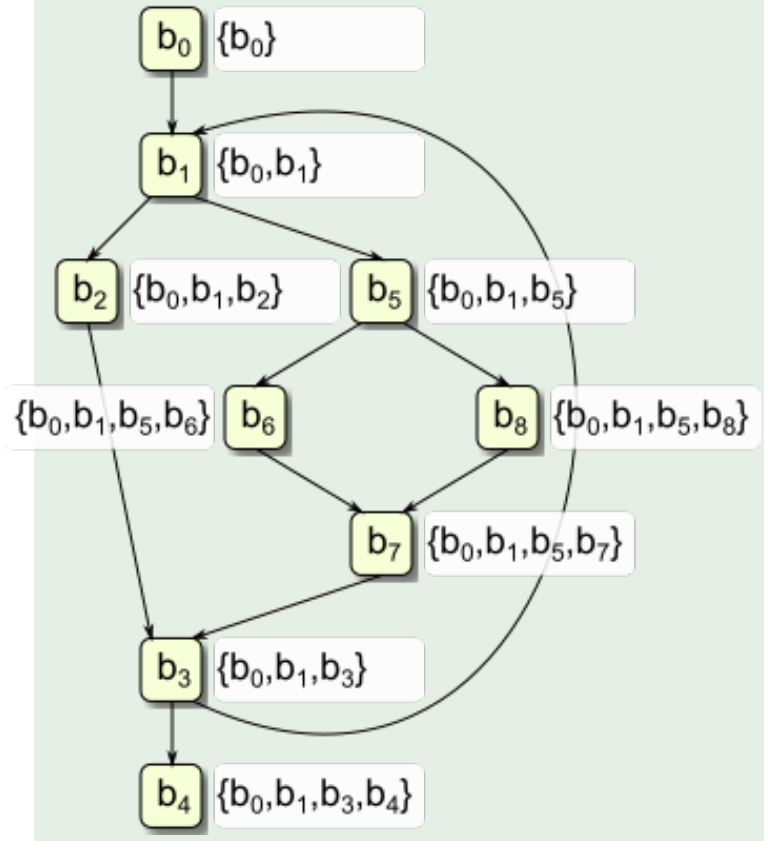
Tree where node's children are those it immediately dominates

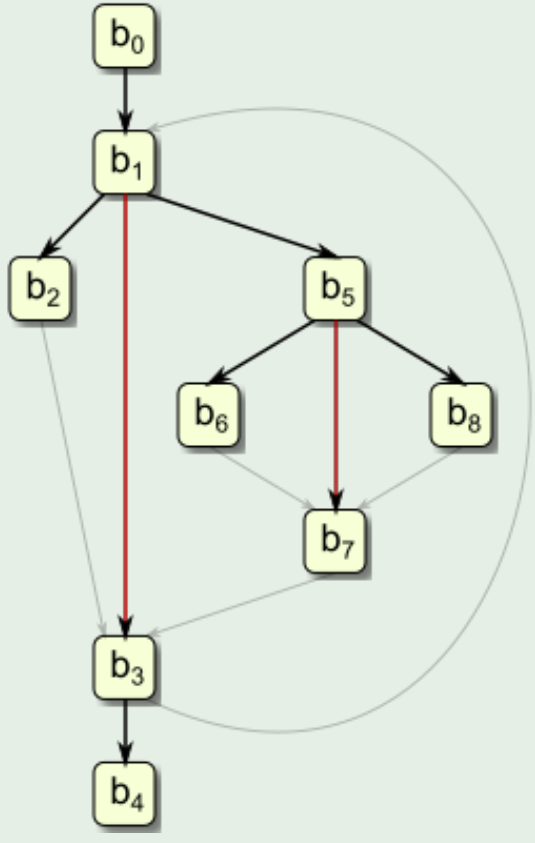
Dominance frontier

$DF(n)$ is set of nodes, d s.t. n dominates an immediate predecessor of d, but n does not strictly dominate d

Dominators

Example: Dominator tree

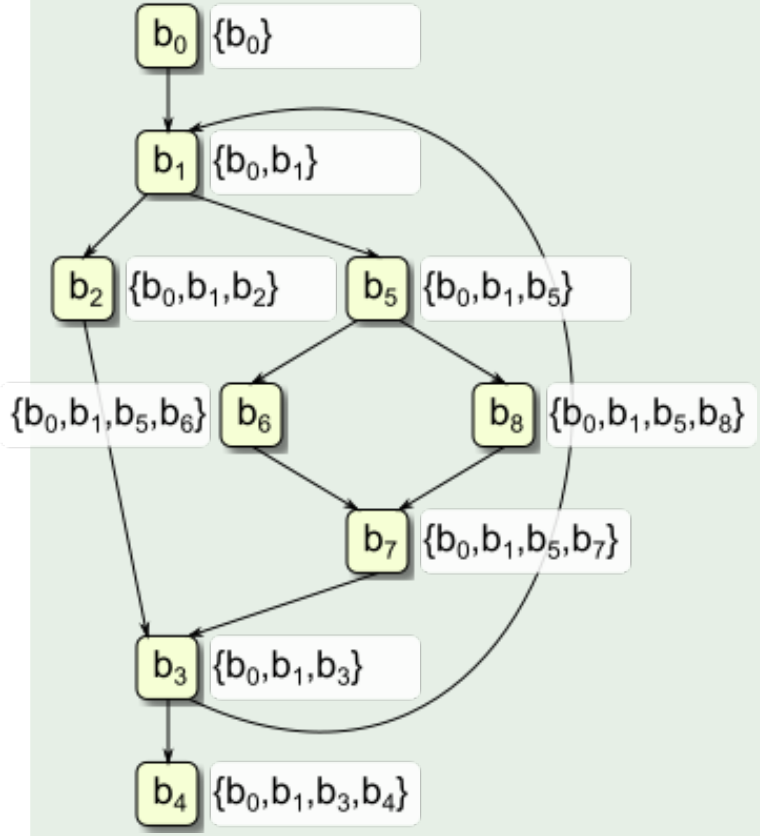


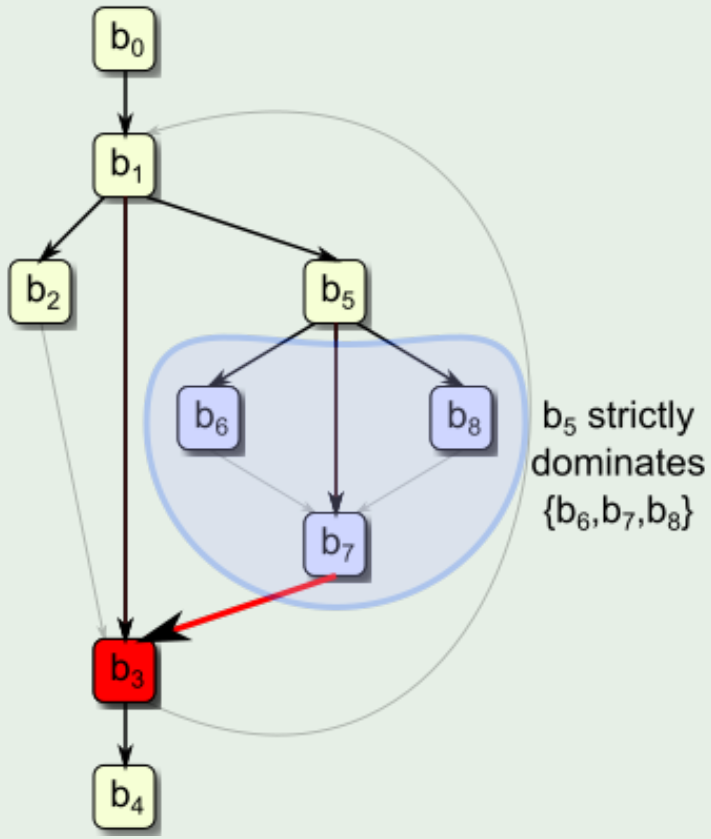


Where are dominance frontiers?

Dominators

Example: Dominator tree

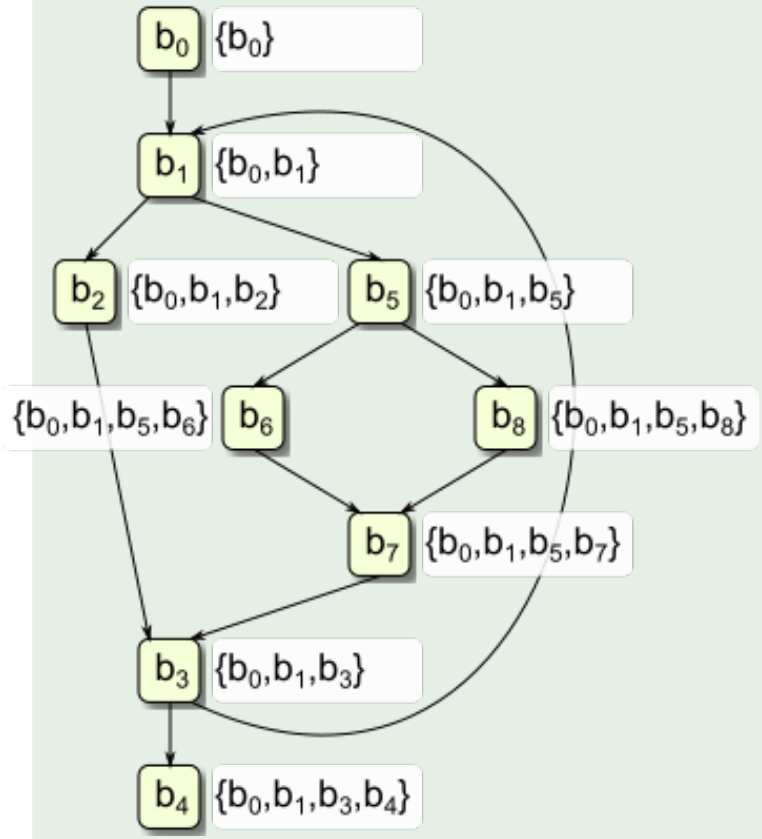


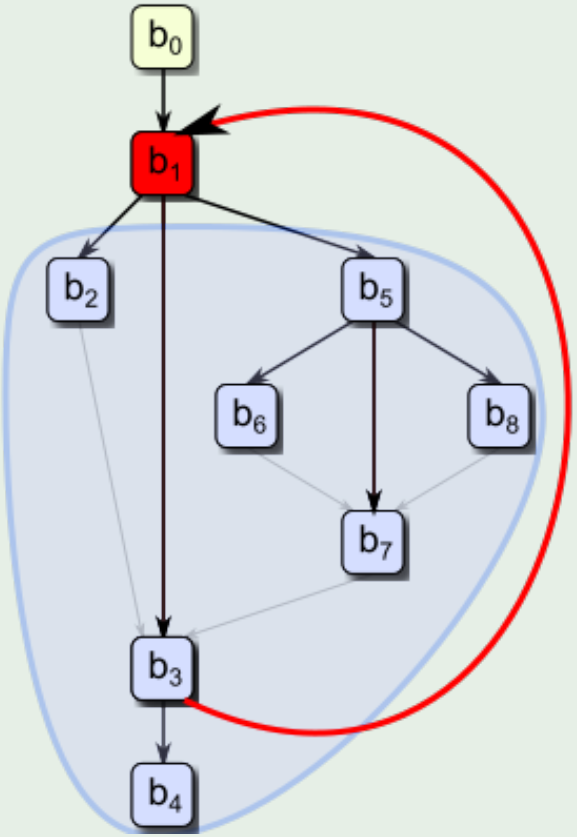


$$DF(b_5) = \{b_3\}$$

Dominators

Example: Dominator tree





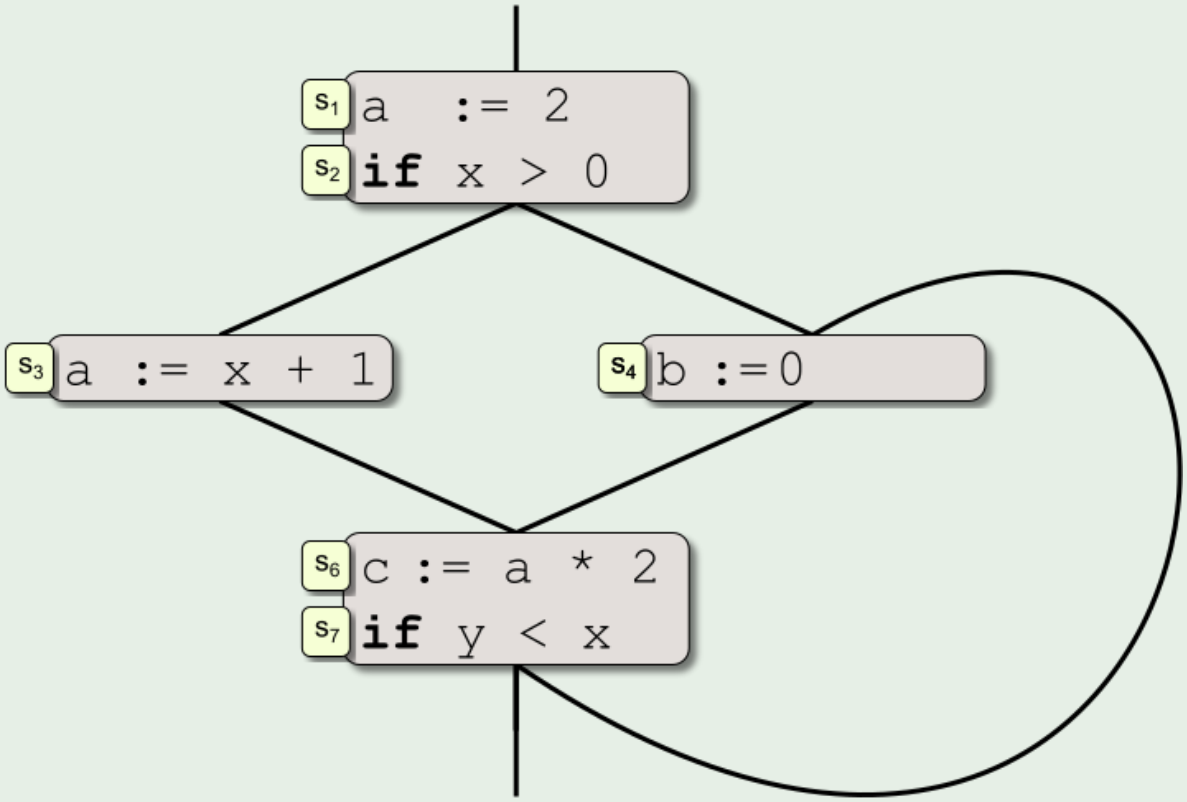
$$DF(b_1) = \{b_1\}$$

Static single-assignment form (SSA)

- Often allowing variable redefinition complicates analysis
- In SSA:
  - One variable per definition
  - Each use refers to one definition
  - Definitions merge with ϕ functions
  - Φ functions execute instantaneously in parallel
- Used by or simplifies many analyses

Static single-assignment form (SSA)

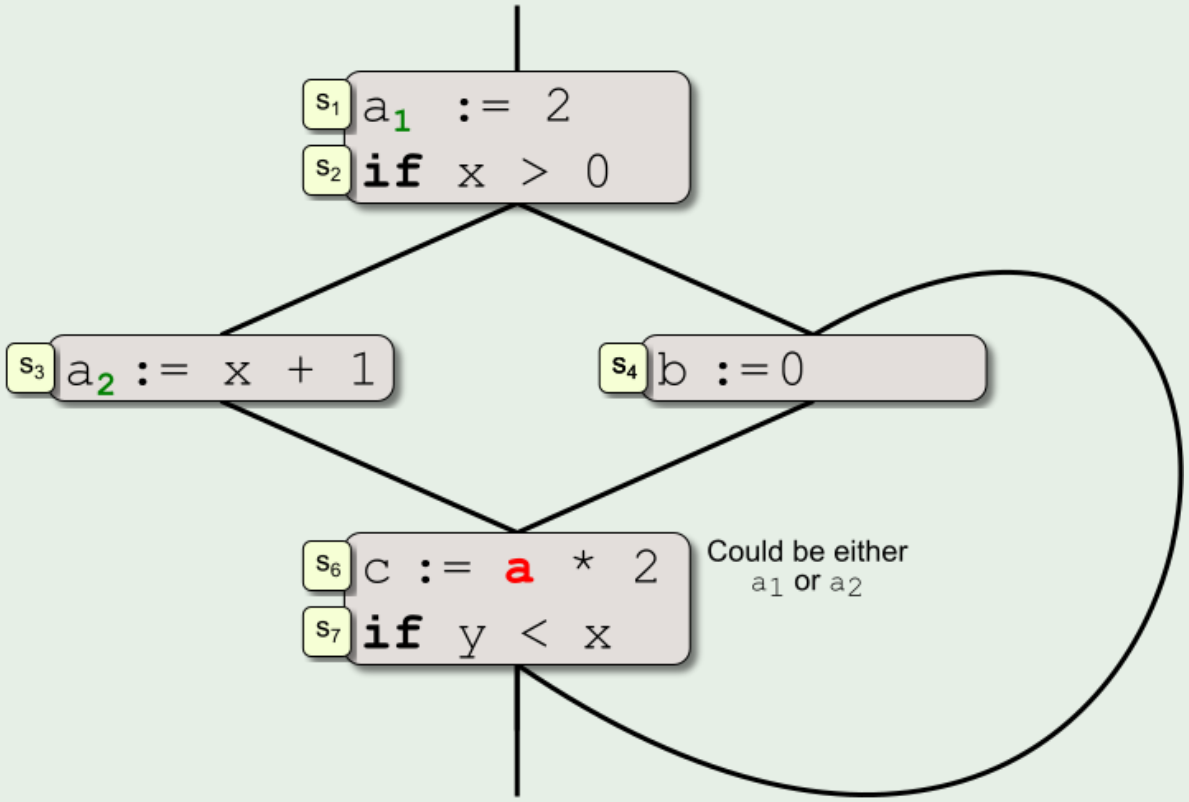
Example: Intuitive conversion to SSA



Original CFG

Static single-assignment form (SSA)

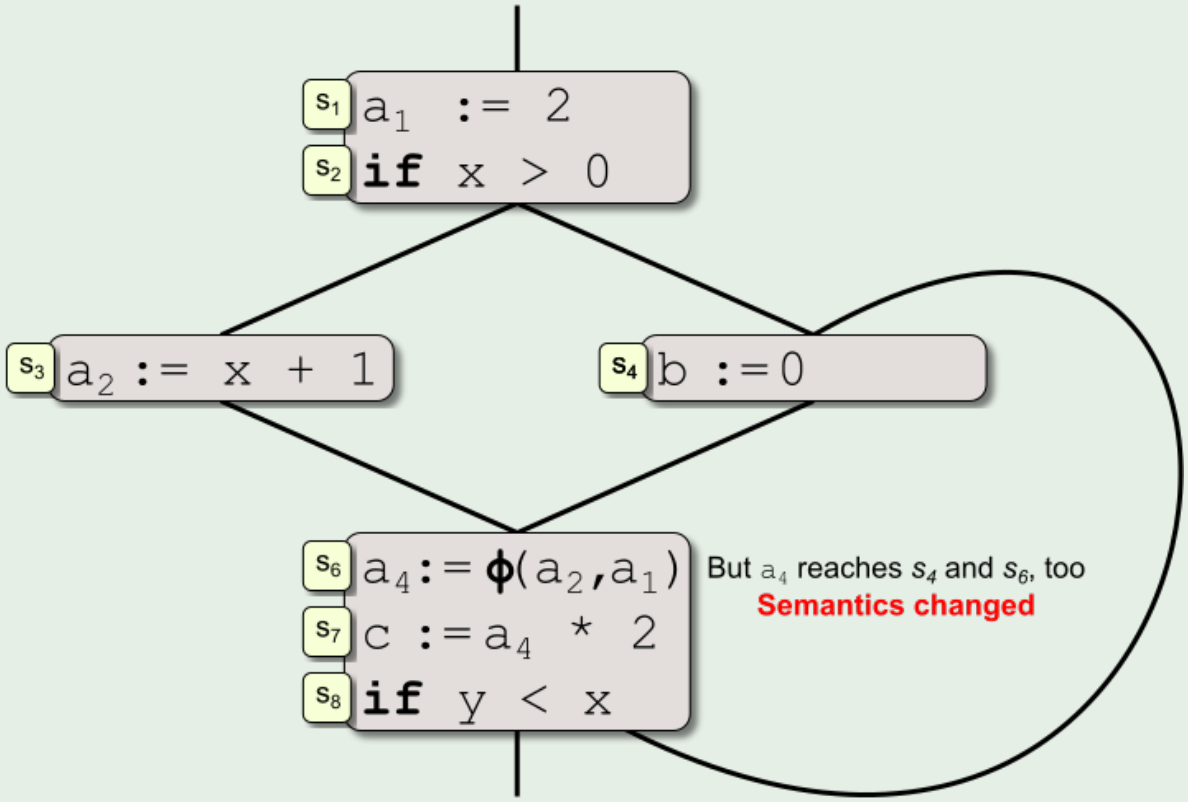
Example: Intuitive conversion to SSA



Rename multiple definitions of same variable

Static single-assignment form (SSA)

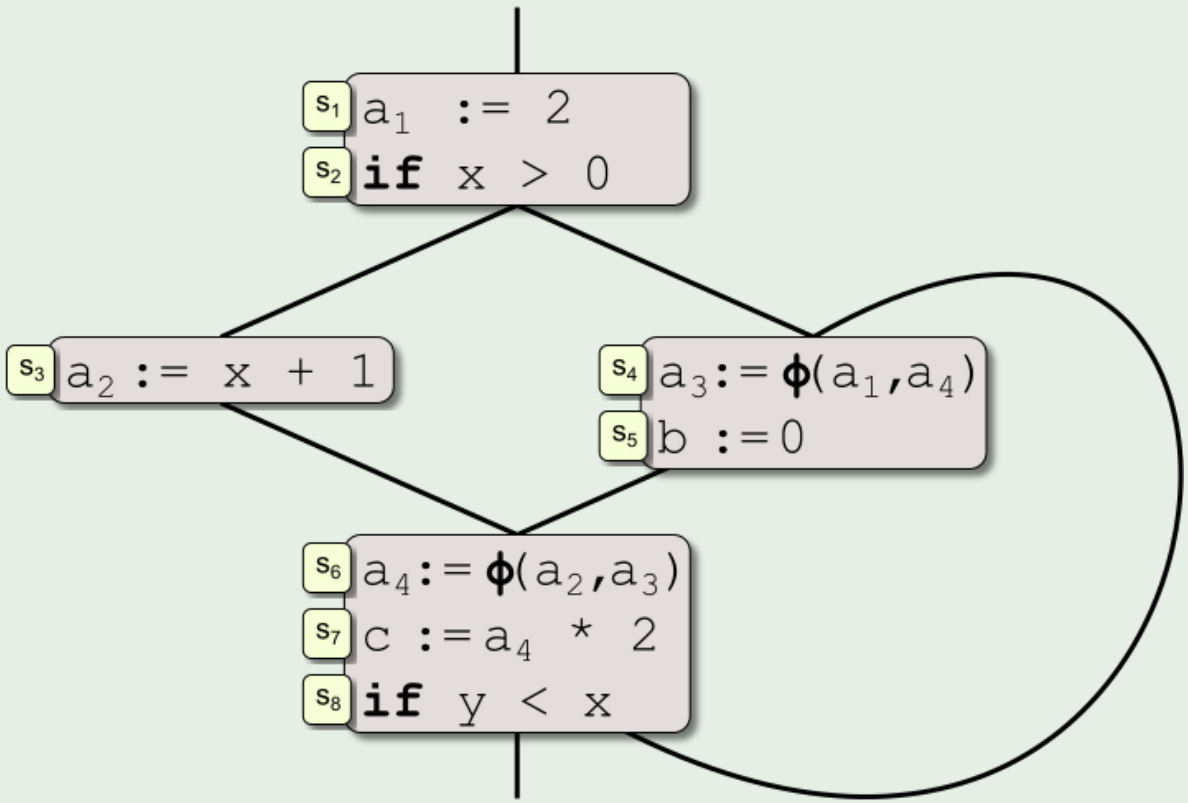
Example: Intuitive conversion to SSA



Repeatedly merge definitions with ϕ

Static single-assignment form (SSA)

Example: Intuitive conversion to SSA



Now in SSA form

Static single-assignment form (SSA)
Types of SSA

- **Maximal SSA** - Places ϕ node for variable $\mathtt{x}$ at every *join* block if block uses or defines $\mathtt{x}$
- **Minimal SSA** - Places ϕ node for variable $\mathtt{x}$ at every *join* block with $2+$ reaching definitions of $\mathtt{x}$
- **Semipruned SSA** - Eliminates ϕs not live across block boundaries
- **Pruned SSA** - Adds liveness test to avoid ϕs of dead definitions

Static single-assignment form (SSA)

Conversion to SSA sketch[2]

- For each definition[1] of `x` in block `b`, add ϕ for `x` in each block in $DF(b)$
- This introduces more definitions, so repeat
- Rename variables
- Can be done in $T(n) = O(n)$, if liveness cheap

[1] Different liveness tests (including none) here change SSA type

[2] See EaC 9.3.1-9.3.4

Static single-assignment form (SSA)

Conversion from SSA sketch[3]

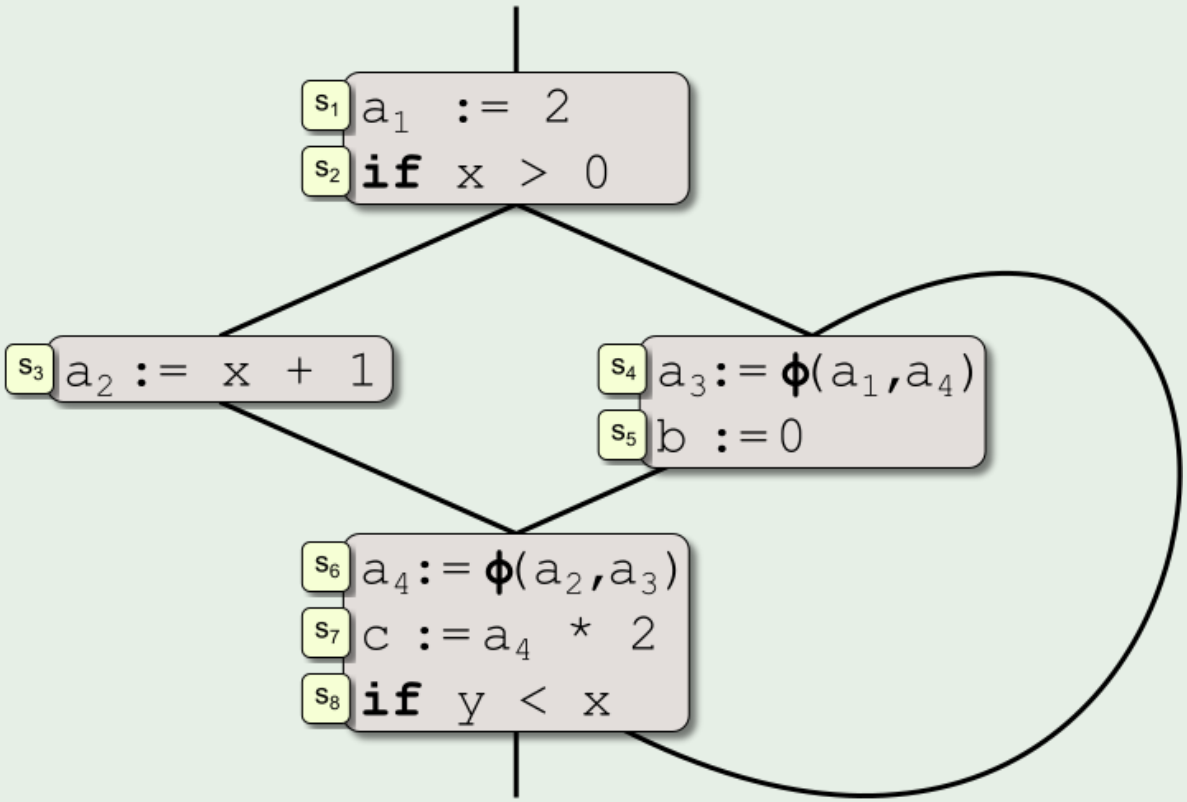
- Cannot just remove ϕ nodes; optimisations make this unsafe
- Place copy operations on incoming edges
- Split edges if necessary
- Delete ϕs
- Remove redundant copies afterwards

[3] See EaC 9.3.5

Static single-assignment form (SSA)

Conversion from SSA

Example: Intuitive conversion from SSA

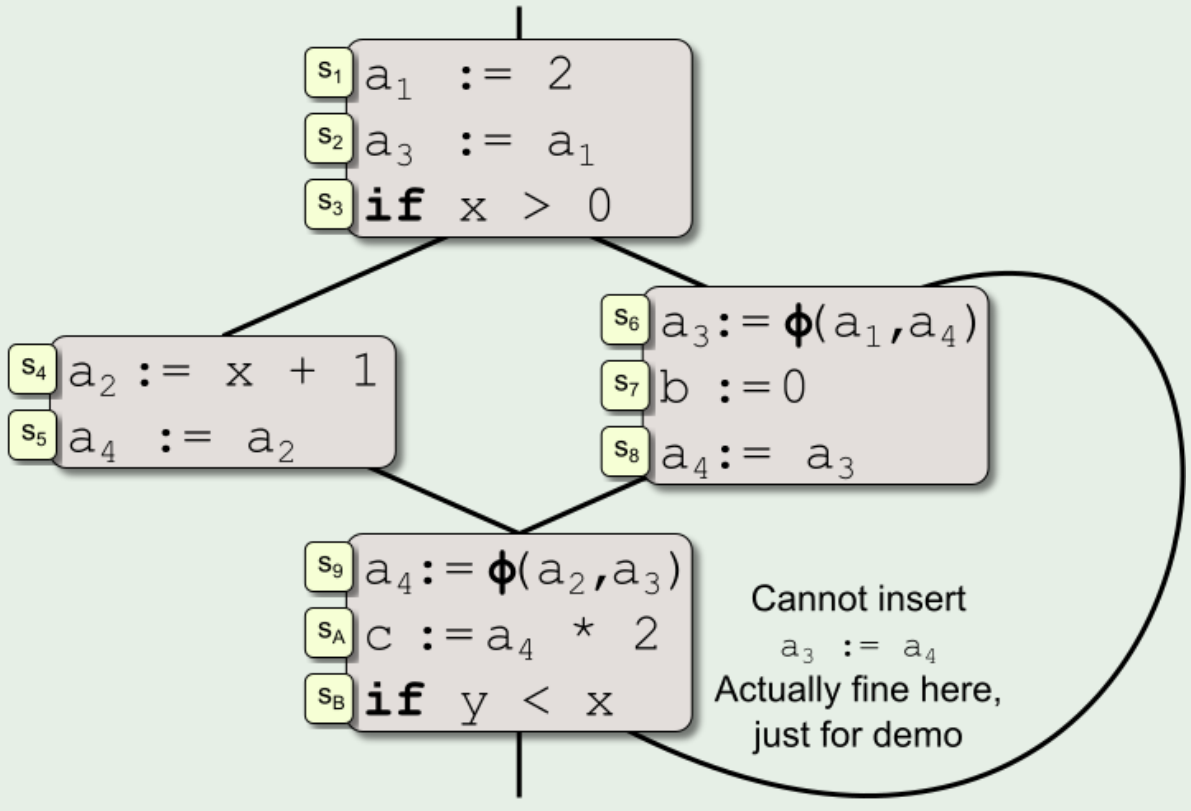


Original SSA CFG

Static single-assignment form (SSA)

Conversion from SSA

Example: Intuitive conversion from SSA

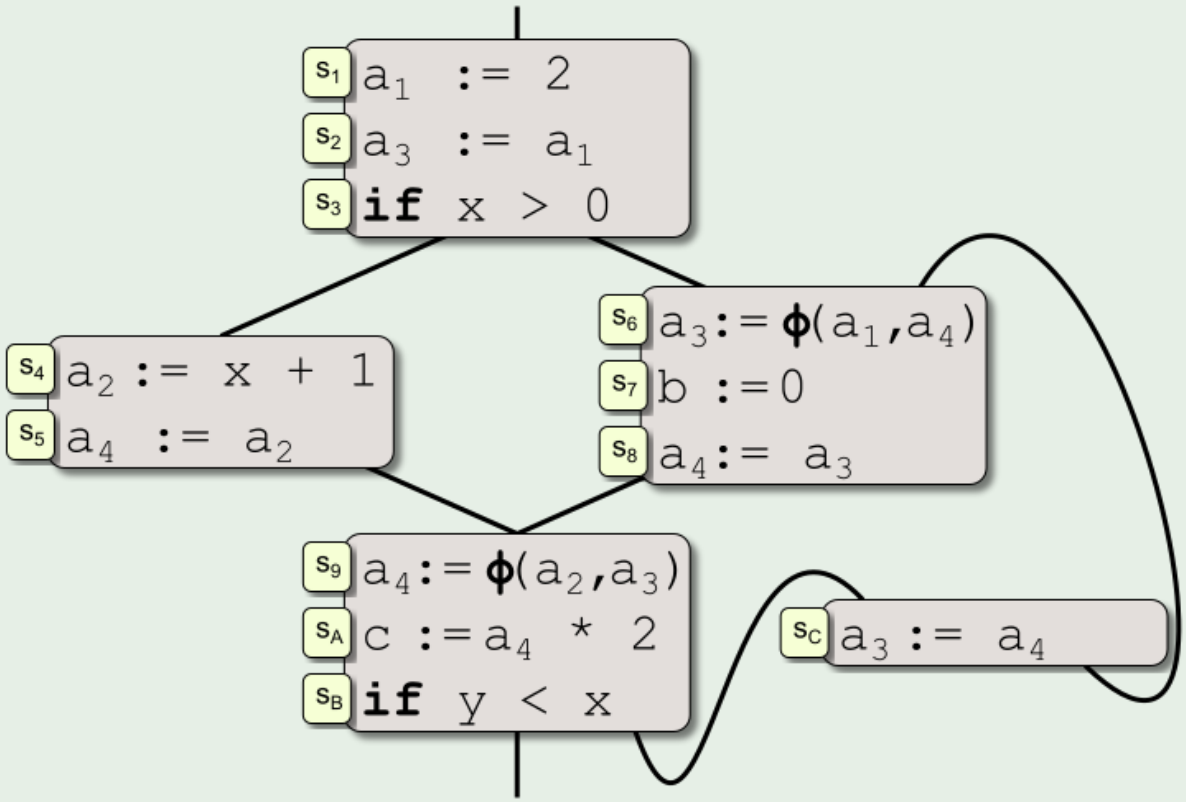


Place copies

Static single-assignment form (SSA)

Conversion from SSA

Example: Intuitive conversion from SSA

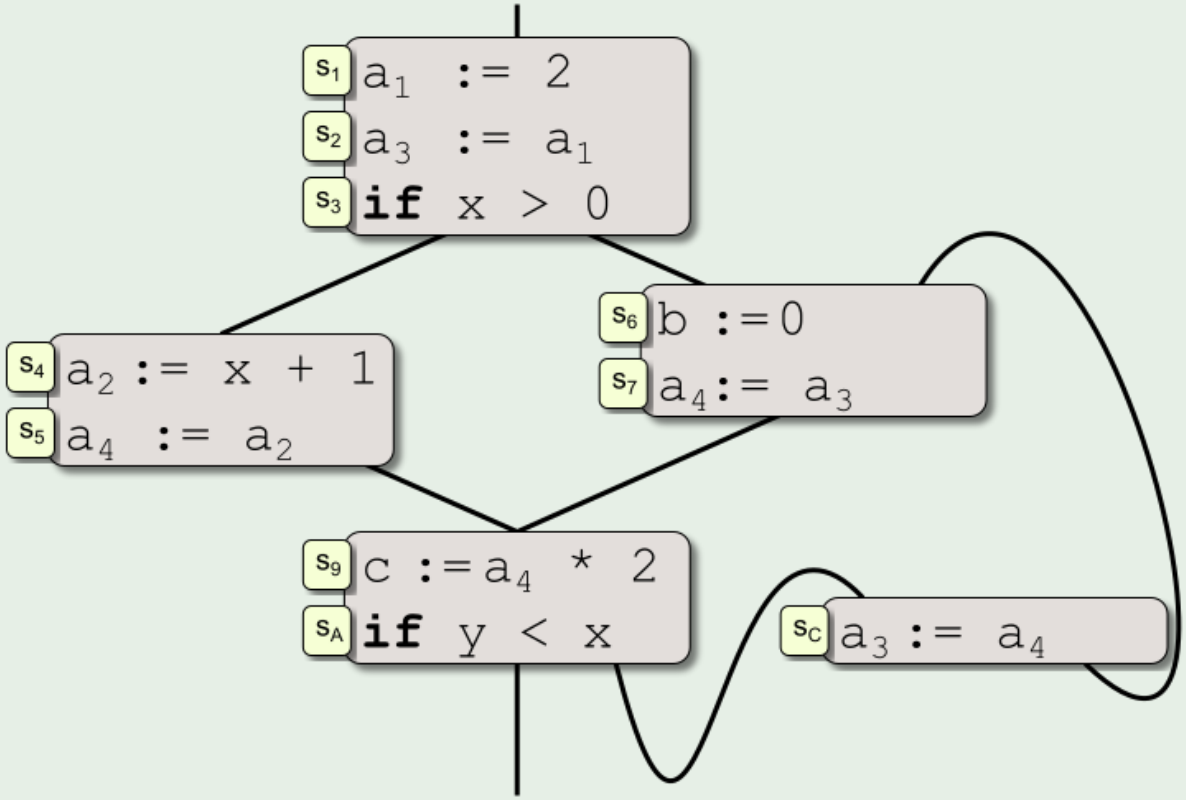


Split where necessary

Static single-assignment form (SSA)

Conversion from SSA

Example: Intuitive conversion from SSA



Remove ϕs

Static single-assignment form (SSA)

Extensions

- Dataflow assumes that all paths in the CFG are taken hence conservative approximations
  - Guarded SSA attempts to overcome this by having additional meet nodes γ, η and μ to carry conditional information around
- Arrays considered monolithic objects `A[1] = .., =A[2]` considered a def-use
  - Array based SSA models access patterns[4]
- Inter-procedural challenging. Pointers destroy analysis! Large research effort in points-to analysis.

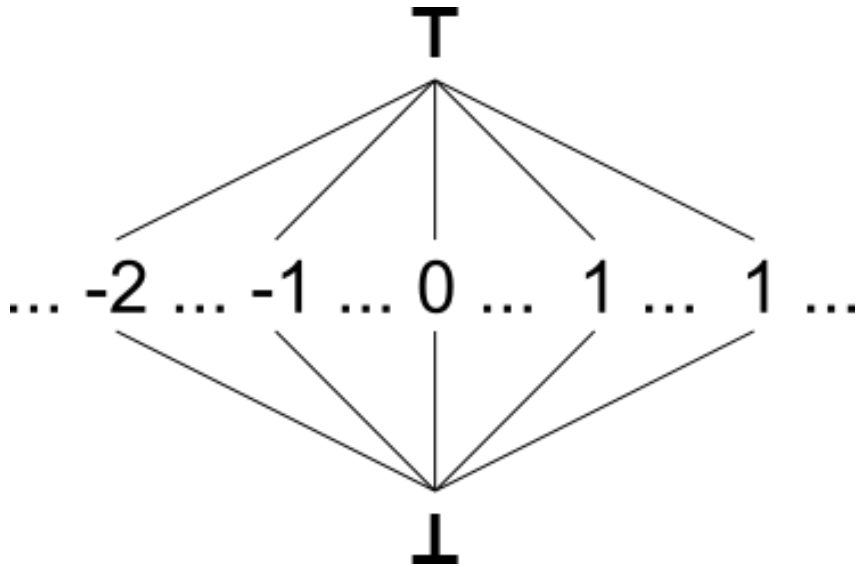
[4] Can be generalised using Presburger formula

Static single-assignment form (SSA)

Constant propagation

Three possible values per variable

$\top$	Not a constant
k	Constant value k
$\bot$	Not computed (maybe never)



Meet is $\top \wedge x = x, \bot \wedge x = \bot, c \wedge c = c, c \wedge d = \bot if\ \ c \neq d$

Transfer functions compute value if all inputs are constant

Summary

- Data flow termination
- More data flow examples
- Dominance
- Static single-assignment form

PPar CDT Advert