

Compiling Techniques

Lecture 16: Dataflow Analysis

Idea: Change Representation that makes def-use chains explicit

As a first step, we translate the nested AST representation into a graph representation:

AST

```
assign() {  
  id_expr() ["id" = "x"] } {  
    literal() ["value" = 1 : !i32] }  
  assign() {  
    id_expr() ["id" = "y"] } {  
      binary_expr() ["op" = "+" ] {  
        id_expr() ["id" = "x"] } {  
          literal() ["value" = 1 : !i32] } }  
    assign() {  
      id_expr() ["id" = "x"] } {  
        literal() ["value" = 2 : !i32] }  
      assign() {  
        id_expr() ["id" = "z"] } {  
          binary_expr() ["op" = "+" ] {  
            id_expr() ["id" = "x"] } {  
              literal() ["value" = 1 : !i32] } }  
          }  
        }  
      }  
    }  
  }  
}
```



Graph-based IR

```
%l0 : !int = literal() ["value" = 1 : !i32]  
assign(%x : !int, %l0 : !int)  
%l1 : !int = literal() ["value" = 1 : !i32]  
%t0 : !int = binary_expr(%x : !int, %l1 : !int) ["op" = "+"]  
assign(%y : !int, %t0 : !int)  
%l2 : !int = literal() ["value" = 2 : !i32]  
assign(%x : !int, %l2 : !int)  
%l3 : !int = literal() ["value" = 1 : !i32]  
%t1 : !int = binary_expr(%x : !int, %l3 : !int) ["op" = "+"]  
assign(%z : !int, %t1 : !int)
```

AST to Graph IR Translation Overview

- Recursively visit the AST nodes
- For each AST node without children create corresponding Graph node
- For each AST node with children create list of Graph nodes
- Replace nested regions representing AST children with Names (%x)
- Maintain context during translation that relates variable names in the AST with Names in the Graph (%x)

Types in the Translation

- In the Graph IR operations (can) have a result type
- To simplify the translation it helps to change the type checking to add the type of every expression as an attribute to the AST node
- Then the type of an expression in the AST is directly available when translating the AST node

Control-flow and Data-flow Analysis

Control-flow / data-flow analysis aim to understand the program's behaviour without executing it by analysing the possible different branches a program can take and where variables are accessed.

Analysis enables beneficial program transformations:

Optimizations = Analysis + Transformation

Data-flow Analysis

Data-flow analysis gathers information for *each program point* by analysing the static code approximating its **dynamic** behaviour

Examples:

- Reaching Definitions
- Initialised Variables
- Constant Propagation
- Sign Analysis
- Liveness of variables



```
1 int foo(int input) {  
2   int x,y,z;  
3   x = input;  
4   while (x > 1) {  
5     y = x / 2;  
6     if (y > 3) x = x - y;  
7     z = x - 4;  
8     if (z > 0) x = x / 2;  
9     z = z-1;  
10  }  
11  return x;  
12 }
```

Is z ever initialised?

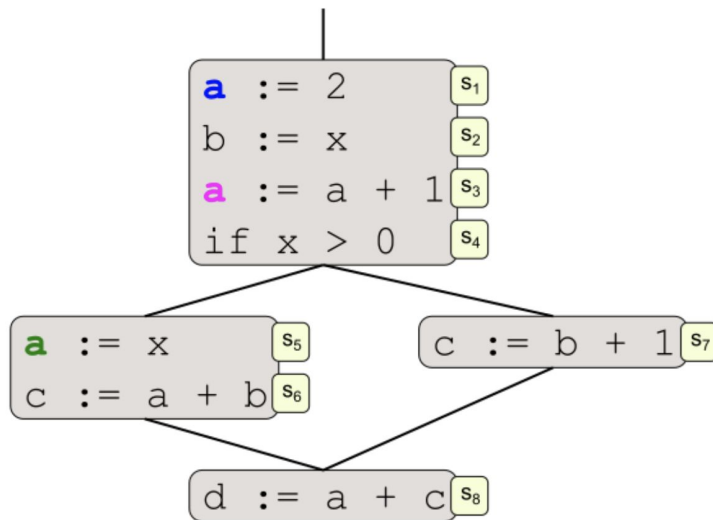
What values are possible for y here?

Is this computation ever used?

Reaching Analysis: Reaching definitions

*Definition of variable **v** at program point **d** **reaches** point **u** if there exists a control-flow path **p** from **d** to **u** such that no definition of **v** appears on that path.*

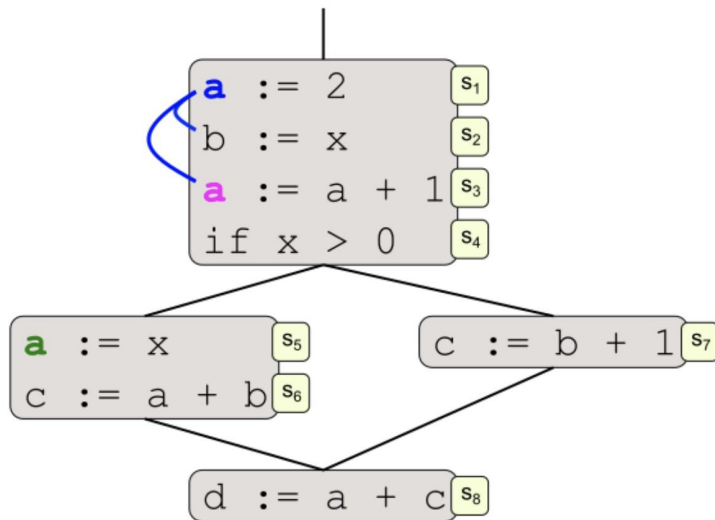
Reaching
definitions of **a**?



Reaching Analysis: Reaching definitions

*Definition of variable **v** at program point **d** **reaches** point **u** if there exists a control-flow path **p** from **d** to **u** such that no definition of **v** appears on that path.*

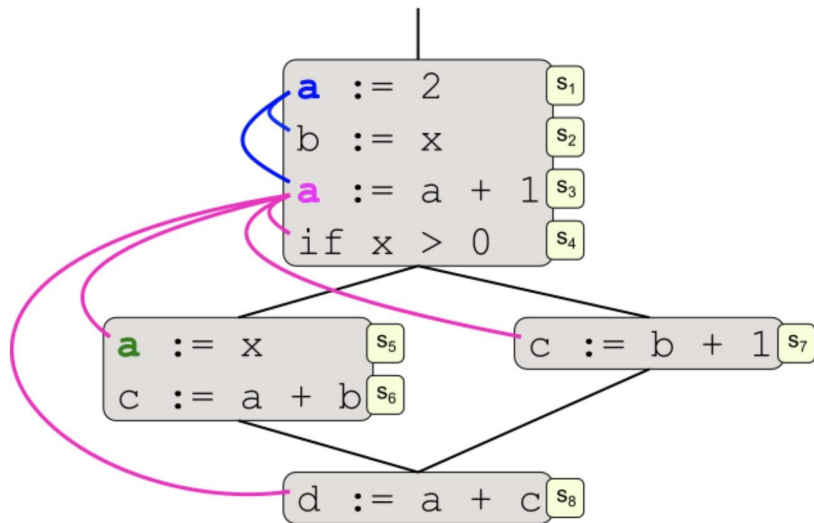
Reaching
definitions of **a**?



Reaching Analysis: Reaching definitions

*Definition of variable **v** at program point **d** **reaches** point **u** if there exists a control-flow path **p** from **d** to **u** such that no definition of **v** appears on that path.*

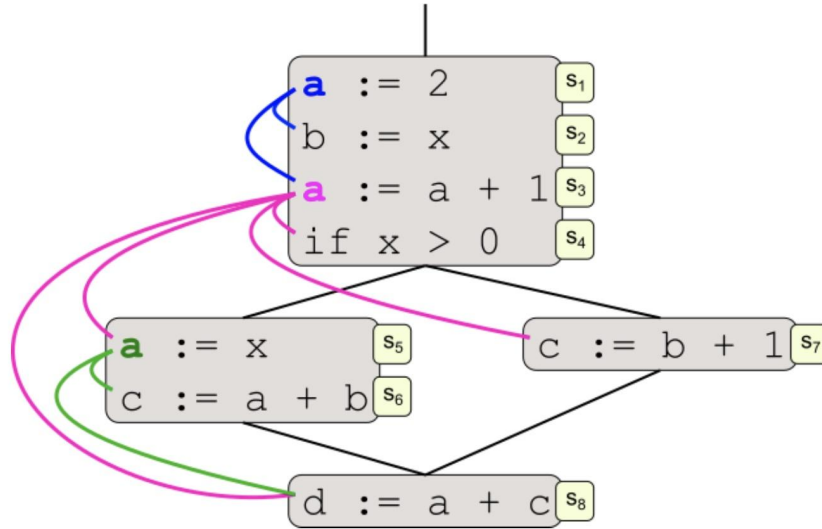
Reaching
definitions of **a**?



Reaching Analysis: Reaching definitions

*Definition of variable **v** at program point **d** **reaches** point **u** if there exists a control-flow path **p** from **d** to **u** such that no definition of **v** appears on that path.*

Reaching
definitions of **a**?



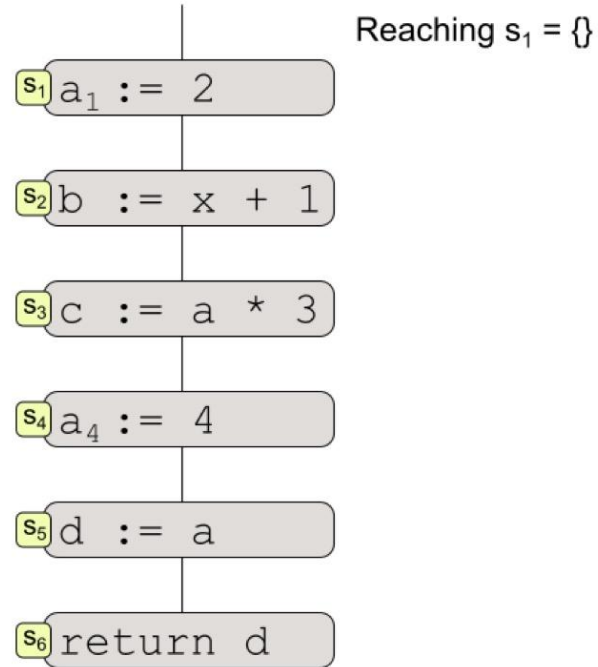
Local Reaching Analysis

A *local analysis* works only on a single basic block

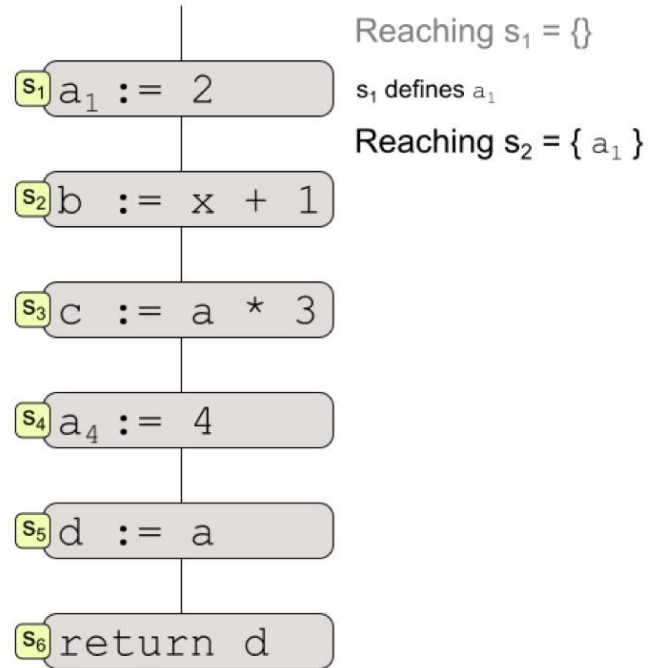
Local Reaching Analysis:

- Maintain a set of current reaching definitions
- Add subscripts to all variable definitions
- Go through all statements from start to end
- If assignment statement $\mathbf{x}_i := \dots$
 - For all $\mathbf{j}$ remove (kill) $\mathbf{x}_j$
 - Add $\mathbf{x}_i$ to the set
- Otherwise, the set remains unchanged

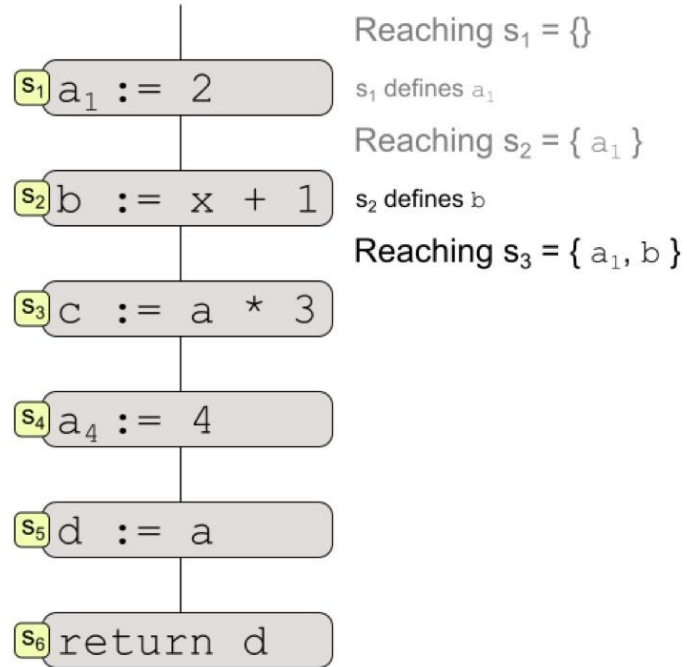
Local Reaching Analysis



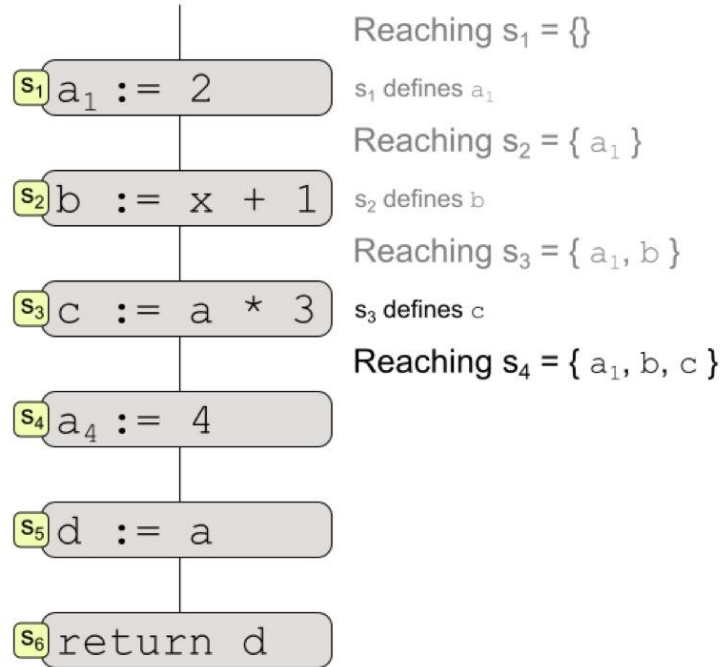
Local Reaching Analysis



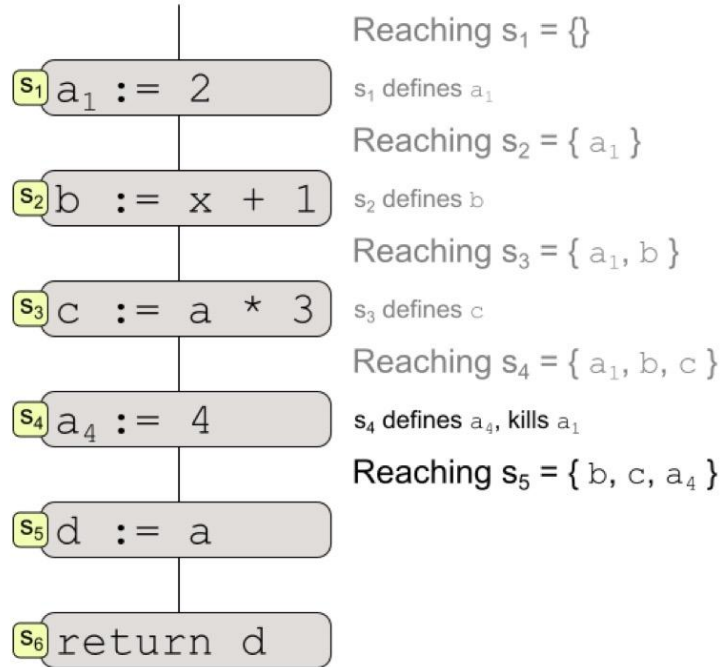
Local Reaching Analysis



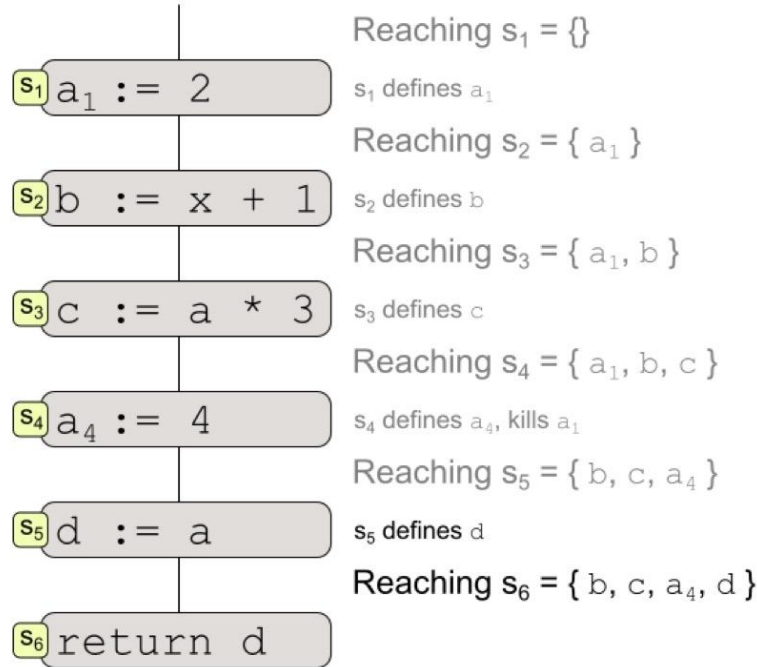
Local Reaching Analysis



Local Reaching Analysis



Local Reaching Analysis



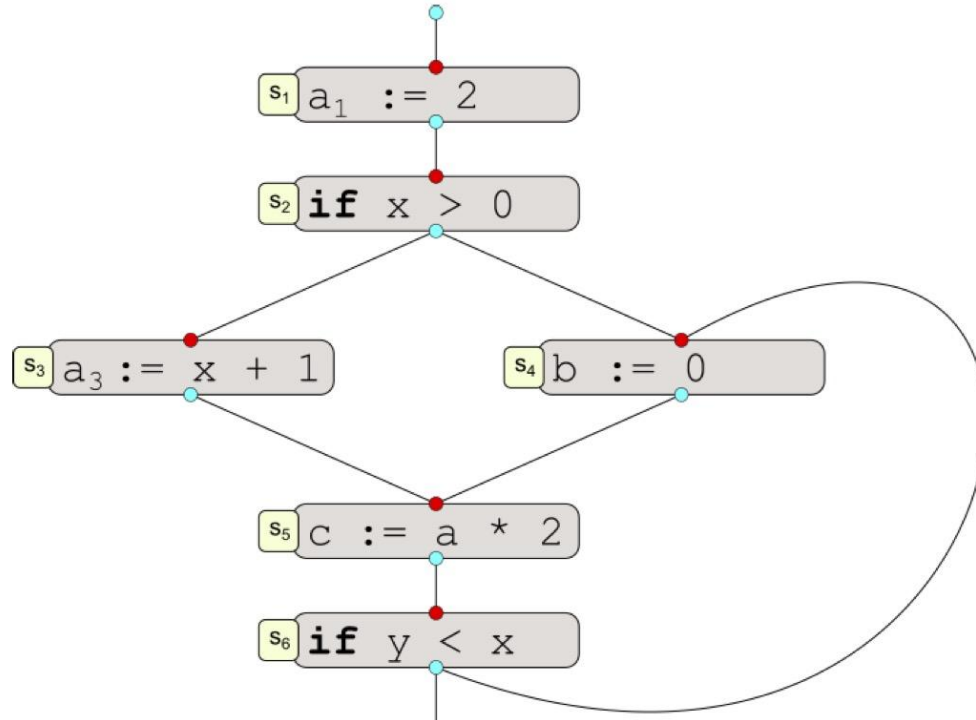
Global Reaching Analysis

Local Analysis is not enough, we must think about control flow!

- Control flow complicates matters
- Refine definition of program point:
 - *In* program point for a statement: Entering the statement
 - *Out* program point for a statement: Leaving the statement
- We will try the previous approach and see where it fails

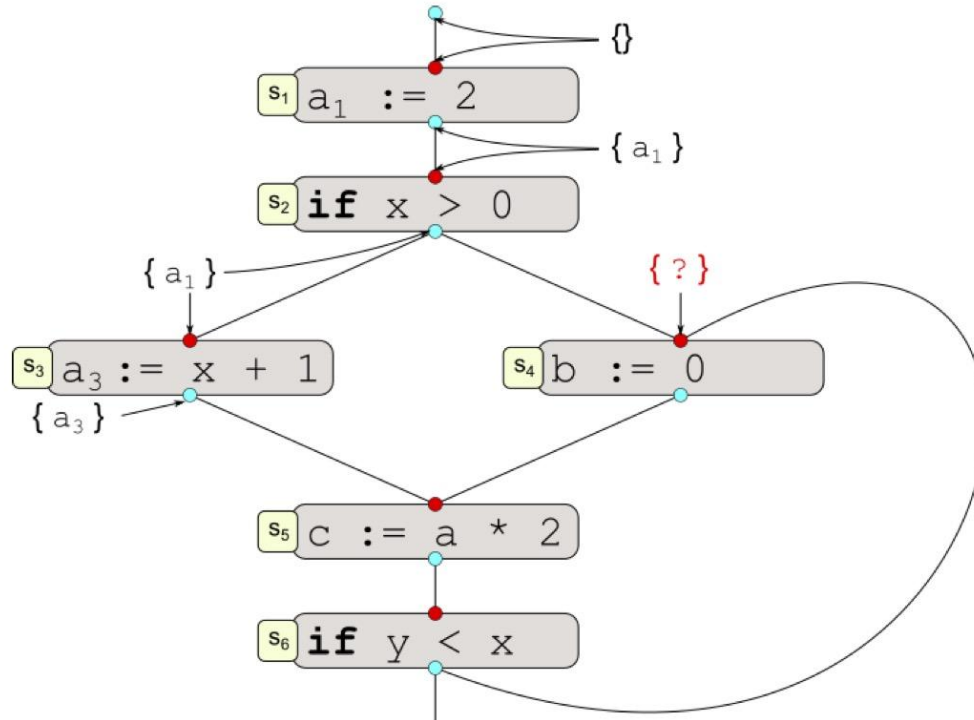
Global Reaching Analysis

Control flow example; try the previous approach



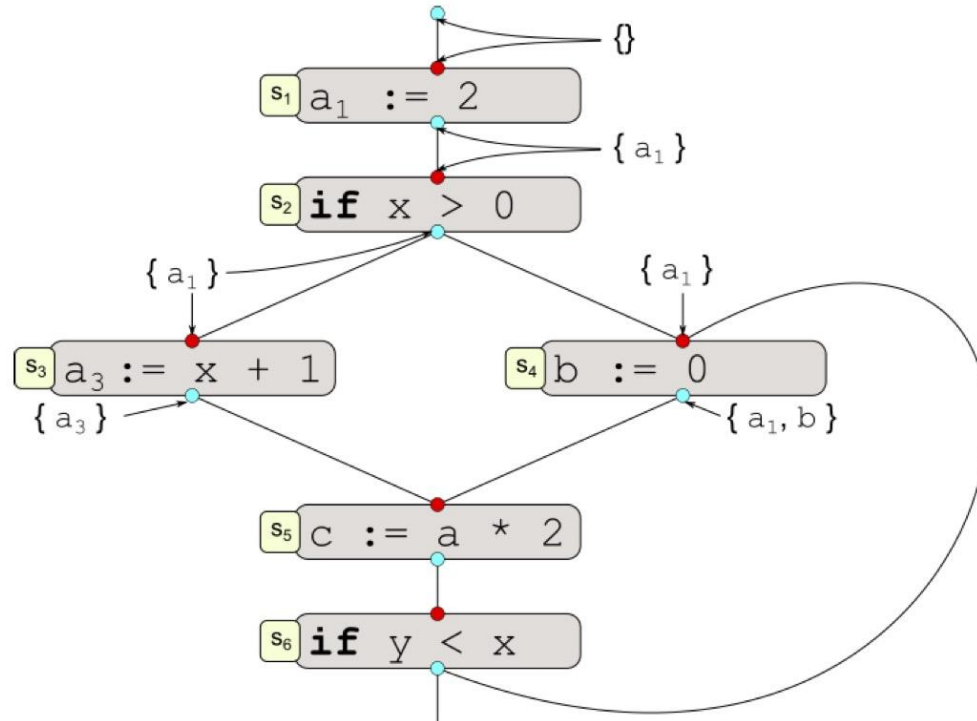
Global Reaching Analysis

s_4 has 2 predecessors; and we don't know $Out(s_6)$ yet



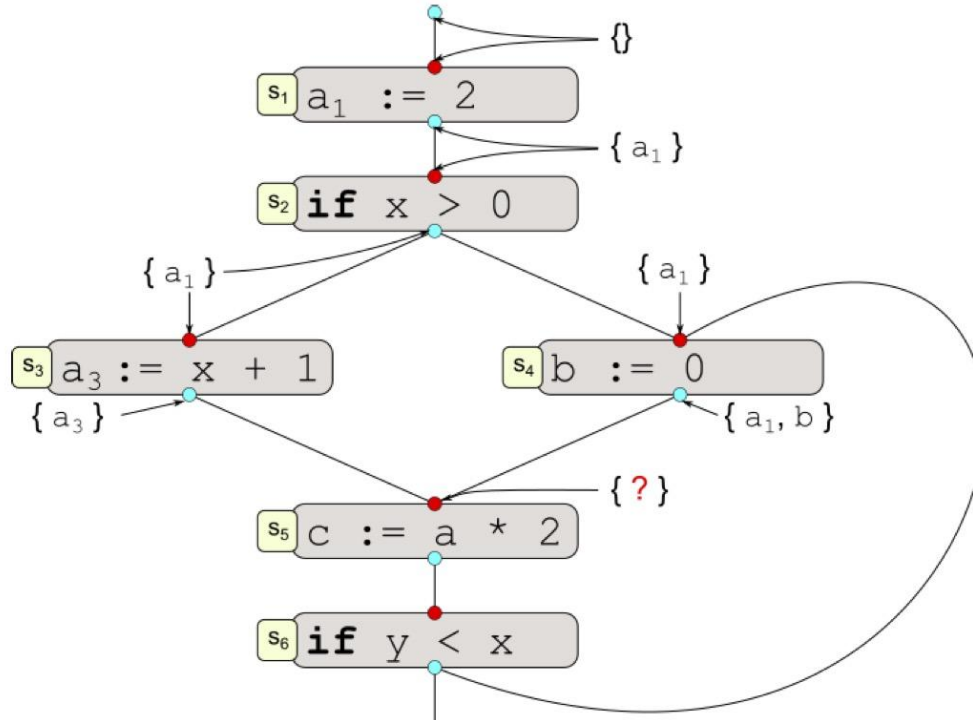
Global Reaching Analysis

But, we know at least that a_1 reaches s_4



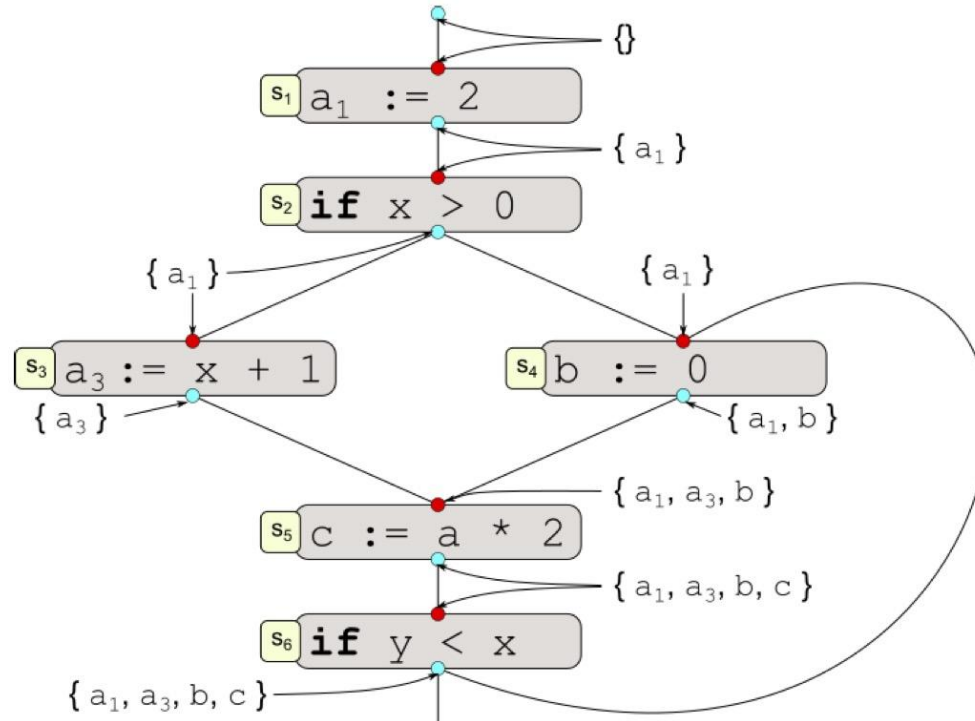
Global Reaching Analysis

s_5 has 2 predecessors



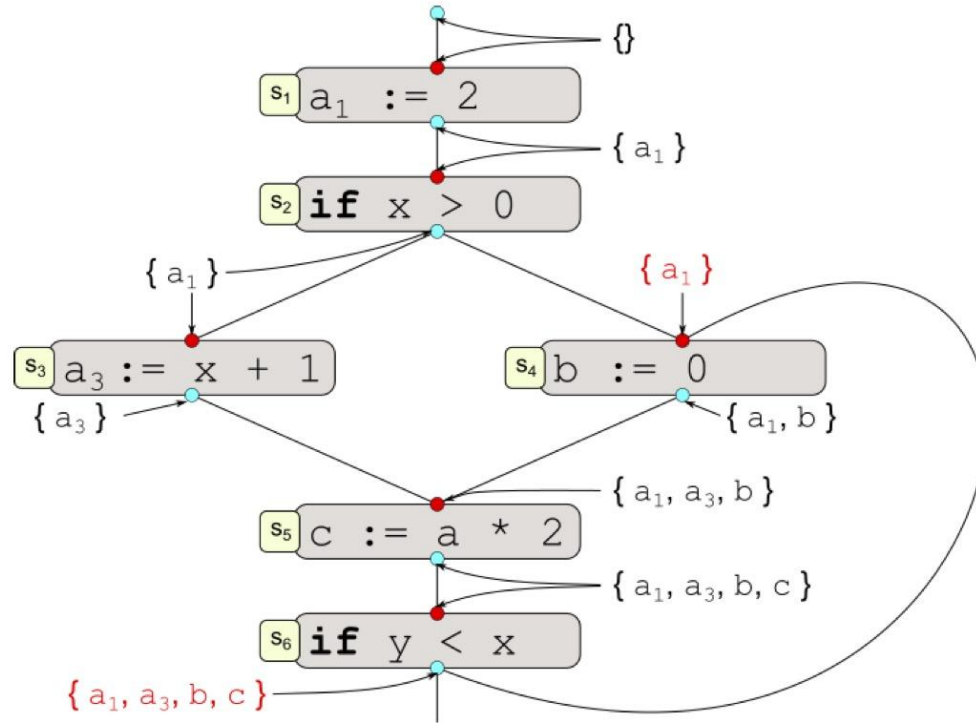
Global Reaching Analysis

All incoming definitions reach $\Rightarrow$ compute union of the two sets



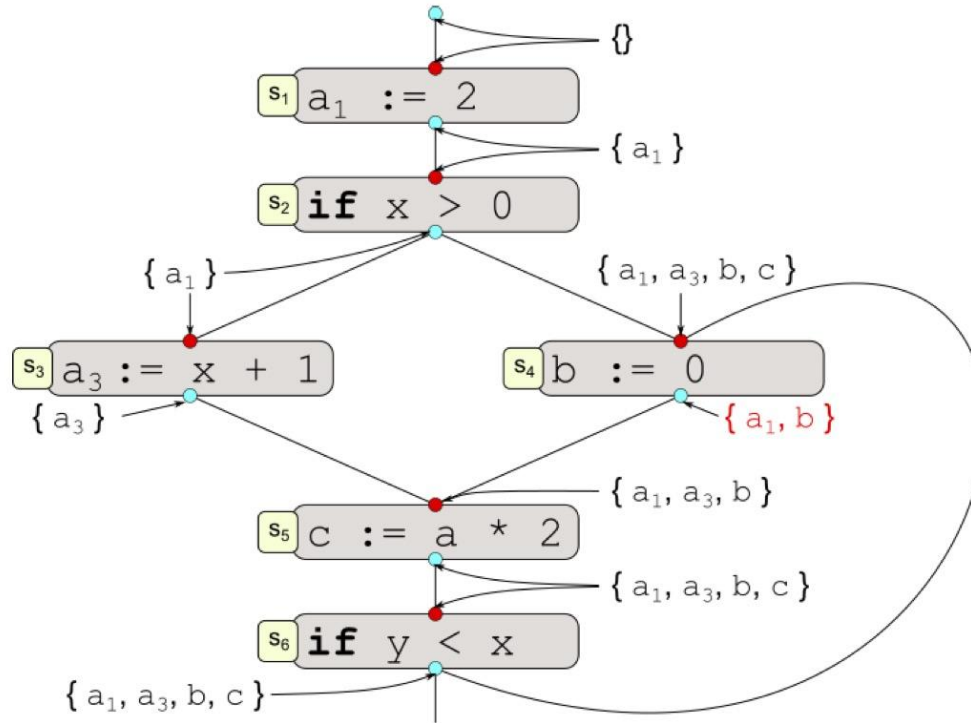
Global Reaching Analysis

Inconsistency, as we now know more about $Out(s_6)$



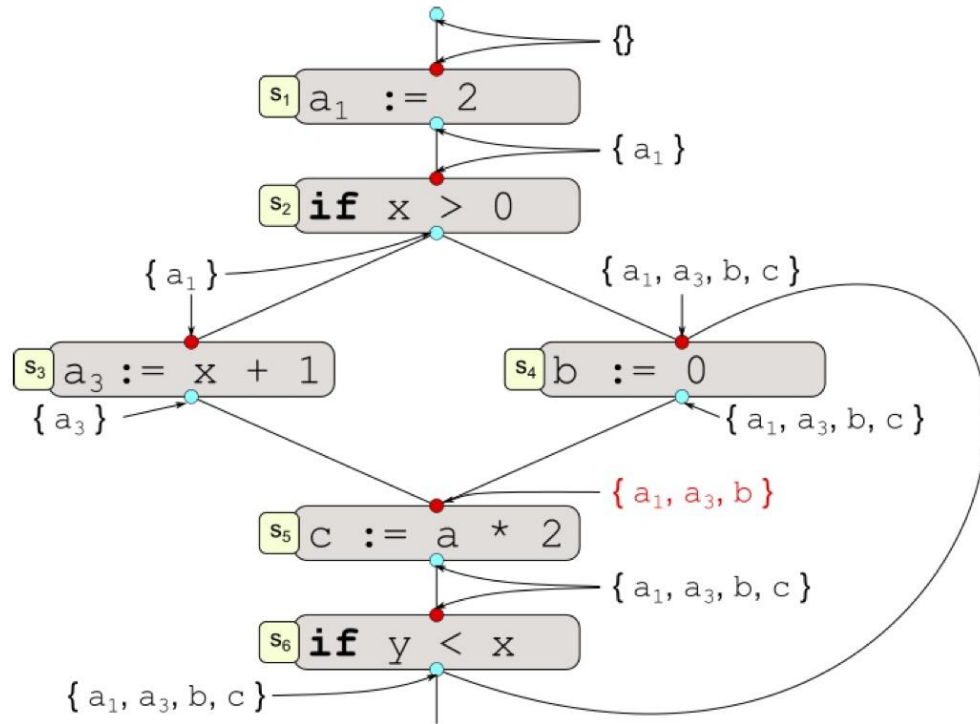
Global Reaching Analysis

All incoming definitions reach $\Rightarrow$ union $\Rightarrow$ inconsistency



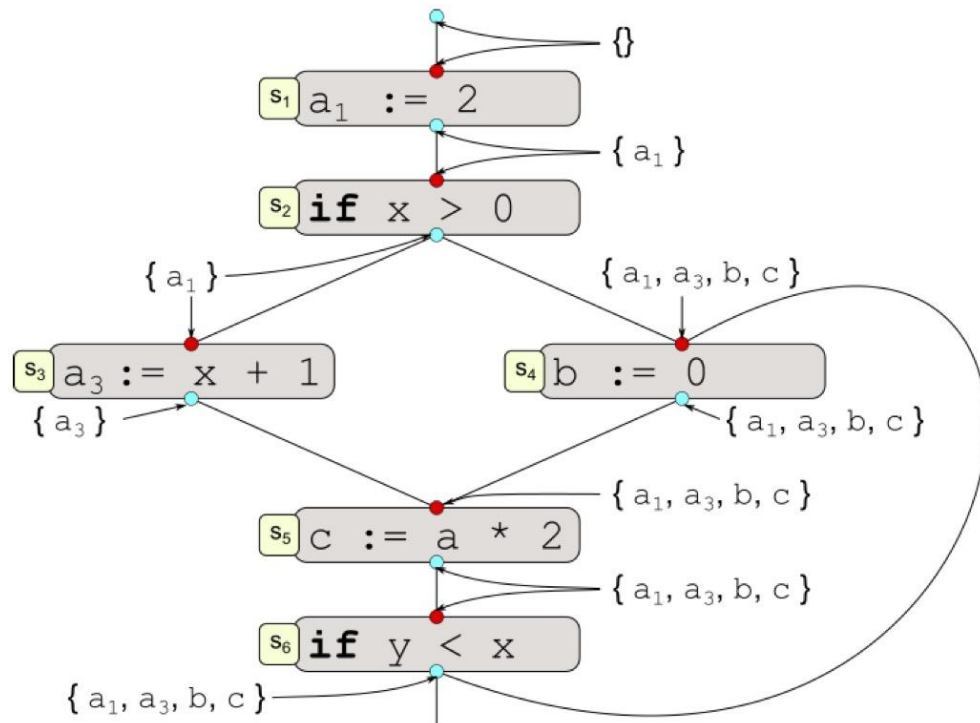
Global Reaching Analysis

Inconsistency



Global Reaching Analysis

Consistent state



Reaching Analysis: *Dataflow equations*

Let us formalise our intuition

- For each statement s , compute $Out(s)$ from $In(s)$

If s is an assignment to x , delete all definitions of x , and add new definitions:

$$Out(s : x_i := ...) = (In(s) - \{x_j; \forall j\}) \cup \{x_i\}$$

- Multiple incoming edges must merge to compute $In(s)$

$$In(s) = \bigcup_{\forall p \in Pred(s)} Out(p)$$

- We start with an empty set

$$Init(s) = \emptyset$$

Reaching Analysis: *Observations*

- Analysis assumes a control flow graph
- Start with a conservative approximation
- Refine the approximations
- Stop when consistent (there are no further changes)
- Information flows *forward* from a statement to its successors

General Dataflow Analysis

- **Direction** – *forward* or *backward*
- **Transfer function** – computes effect of statement
e.g. $Out(s) = (In(s) - Kill(s)) \cup Gen(s)$
- **Meet operator** – merges values from multiple incoming edges
e.g. $In(s) = \bigcup_{\forall p \in Pred(s)} Out(p)$
- **Value set** – the information being passed around
e.g. Sets of definitions
- **Initial values**
Should be most conservative value; Start node often a special case

Iterative Round-Robin Algorithm

```
for each node, start_node do
```

```
    Initialise start_node
```

```
while values changing do
```

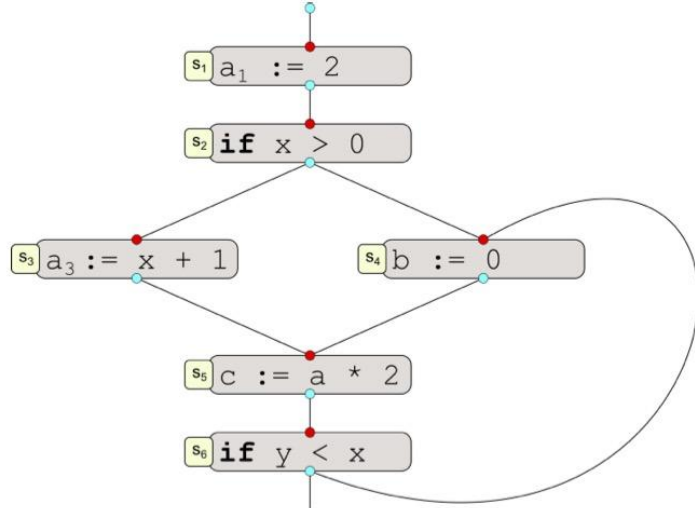
```
    for each node do
```

```
        Apply meet function           // compute  $In(s)$ 
```

```
        Apply transfer function       // compute  $Out(s)$ 
```

Reaching Analysis with Dataflow Iterative Algorithm

Reaching definitions control flow example - Calculate RD sets?



$$In(s) = \bigcup_{\forall p \in Pred(s)} Out(p)$$

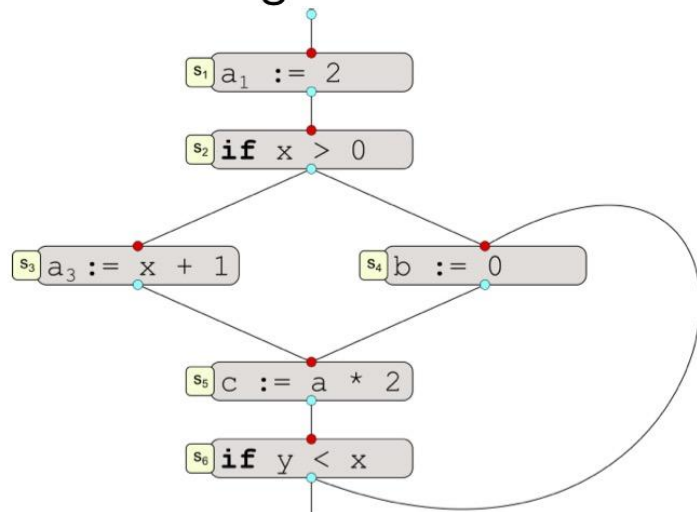
$$Out(s : d_i := \dots) = (In(s) - \{d_j; \forall j\}) \cup d_i$$

$$RD(s) = \bigcup_{\forall p: d_i = \dots \in Pred(s)} (RD(p) - \{d_j; \forall j\}) \cup \{d_i\}$$

Node	s ₁	s ₂	s ₃	s ₄	s ₅	s ₆
RD ⁴	∅	∅	∅	∅	∅	∅

Reaching Analysis with Dataflow Iterative Algorithm

Reaching definitions control flow example - Calculate RD sets?



$$In(s) = \bigcup_{\forall p \in Pred(s)} Out(p)$$

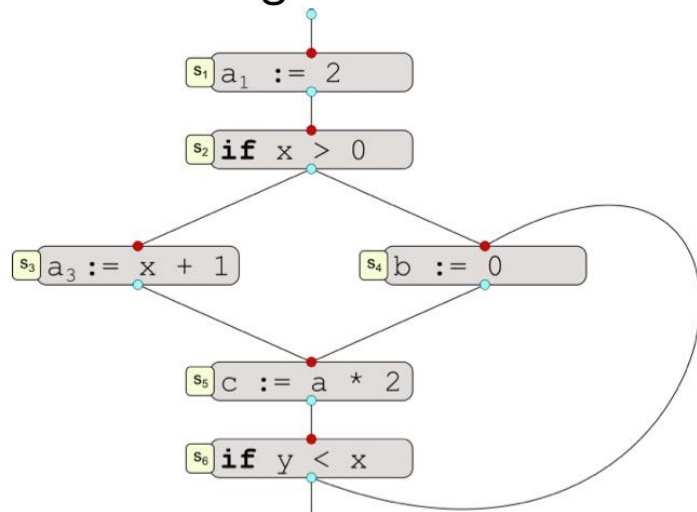
$$Out(s : d_i := \dots) = (In(s) - \{d_j; \forall j\}) \cup d_i$$

$$RD(s) = \bigcup_{\forall p: d_i = \dots \in Pred(s)} (RD(p) - \{d_j; \forall j\}) \cup \{d_i\}$$

Node	s ₁	s ₂	s ₃	s ₄	s ₅	s ₆
RD ⁴	∅	∅	∅	∅	∅	∅
	∅					

Reaching Analysis with Dataflow Iterative Algorithm

Reaching definitions control flow example - Calculate RD sets?



$$In(s) = \bigcup_{\forall p \in Pred(s)} Out(p)$$

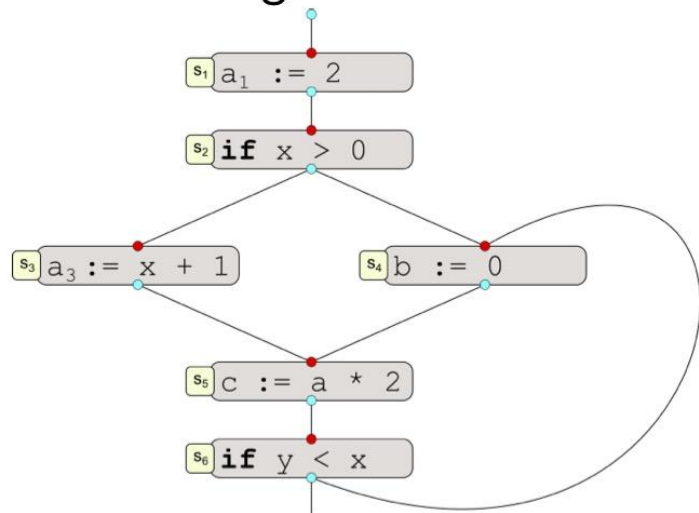
$$Out(s : d_i := ...) = (In(s) - \{d_j; \forall j\}) \cup d_i$$

$$RD(s) = \bigcup_{\forall p: d_i = \dots \in Pred(s)} (RD(p) - \{d_j; \forall j\}) \cup \{d_i\}$$

Node	s ₁	s ₂	s ₃	s ₄	s ₅	s ₆
RD ⁴	∅	∅	∅	∅	∅	∅
	∅	a ₁				

Reaching Analysis with Dataflow Iterative Algorithm

Reaching definitions control flow example - Calculate RD sets?



$$In(s) = \bigcup_{\forall p \in Pred(s)} Out(p)$$

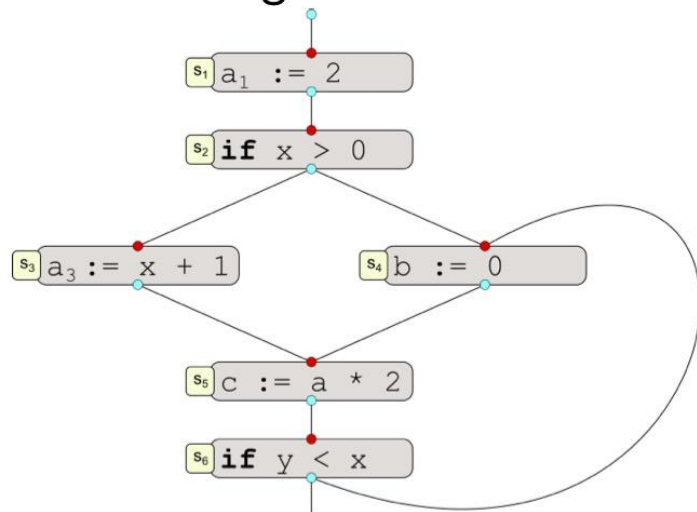
$$Out(s : d_i := \dots) = (In(s) - \{d_j; \forall j\}) \cup d_i$$

$$RD(s) = \bigcup_{\forall p: d_i = \dots \in Pred(s)} (RD(p) - \{d_j; \forall j\}) \cup \{d_i\}$$

Node	s ₁	s ₂	s ₃	s ₄	s ₅	s ₆
RD ⁴	∅	∅	∅	∅	∅	∅
	∅	a ₁	a ₁			

Reaching Analysis with Dataflow Iterative Algorithm

Reaching definitions control flow example - Calculate RD sets?



$$In(s) = \bigcup_{\forall p \in Pred(s)} Out(p)$$

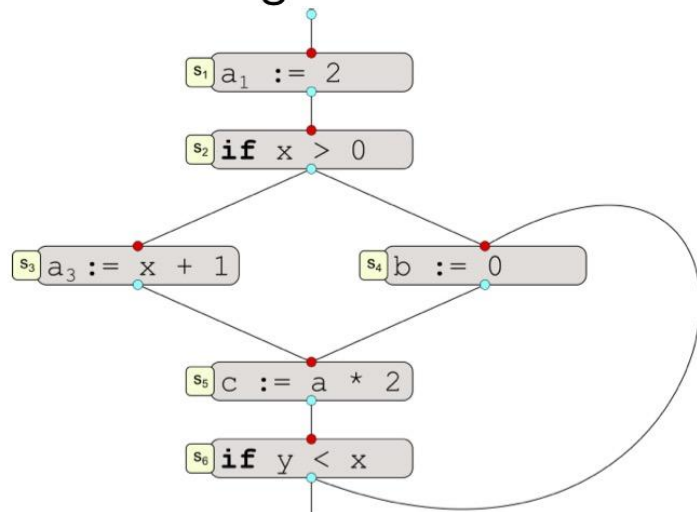
$$Out(s : d_i := \dots) = (In(s) - \{d_j; \forall j\}) \cup d_i$$

$$RD(s) = \bigcup_{\forall p: d_i = \dots \in Pred(s)} (RD(p) - \{d_j; \forall j\}) \cup \{d_i\}$$

Node	s ₁	s ₂	s ₃	s ₄	s ₅	s ₆
RD ⁴	∅	∅	∅	∅	∅	∅
	∅	a ₁	a ₁	a ₁		

Reaching Analysis with Dataflow Iterative Algorithm

Reaching definitions control flow example - Calculate RD sets?



$$In(s) = \bigcup_{\forall p \in Pred(s)} Out(p)$$

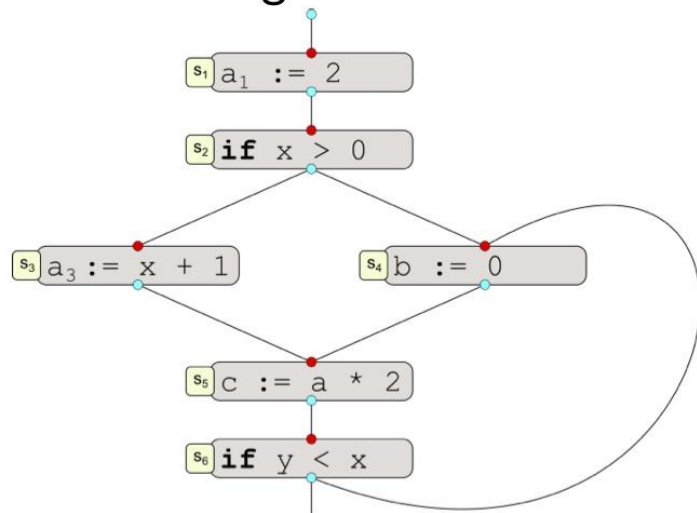
$$Out(s : d_i := \dots) = (In(s) - \{d_j; \forall j\}) \cup d_i$$

$$RD(s) = \bigcup_{\forall p: d_i = \dots \in Pred(s)} (RD(p) - \{d_j; \forall j\}) \cup \{d_i\}$$

Node	s ₁	s ₂	s ₃	s ₄	s ₅	s ₆
RD ⁴	∅	∅	∅	∅	∅	∅
	∅	a ₁	a ₁	a ₁	a ₁ , a ₃ , b	

Reaching Analysis with Dataflow Iterative Algorithm

Reaching definitions control flow example - Calculate RD sets?



$$In(s) = \bigcup_{\forall p \in Pred(s)} Out(p)$$

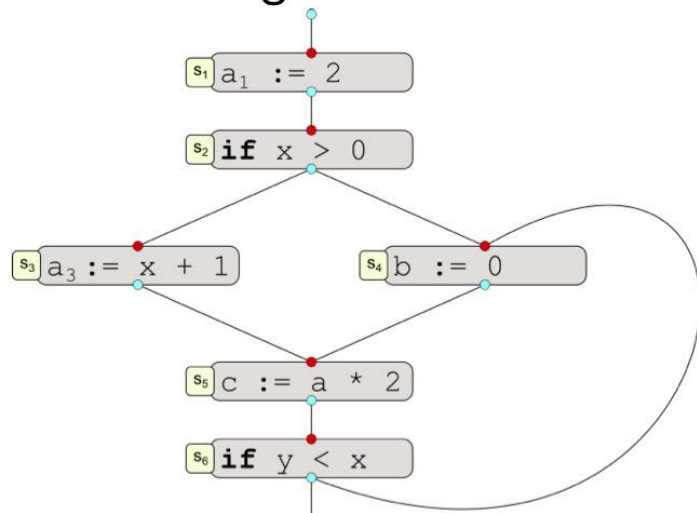
$$Out(s : d_i := \dots) = (In(s) - \{d_j; \forall j\}) \cup d_i$$

$$RD(s) = \bigcup_{\forall p: d_i = \dots \in Pred(s)} (RD(p) - \{d_j; \forall j\}) \cup \{d_i\}$$

Node	s ₁	s ₂	s ₃	s ₄	s ₅	s ₆
RD ⁴	∅	∅	∅	∅	∅	∅
	∅	a ₁	a ₁	a ₁	a ₁ , a ₃ , b	a ₁ , a ₃ , b, c

Reaching Analysis with Dataflow Iterative Algorithm

Reaching definitions control flow example - Calculate RD sets?



$$In(s) = \bigcup_{\forall p \in Pred(s)} Out(p)$$

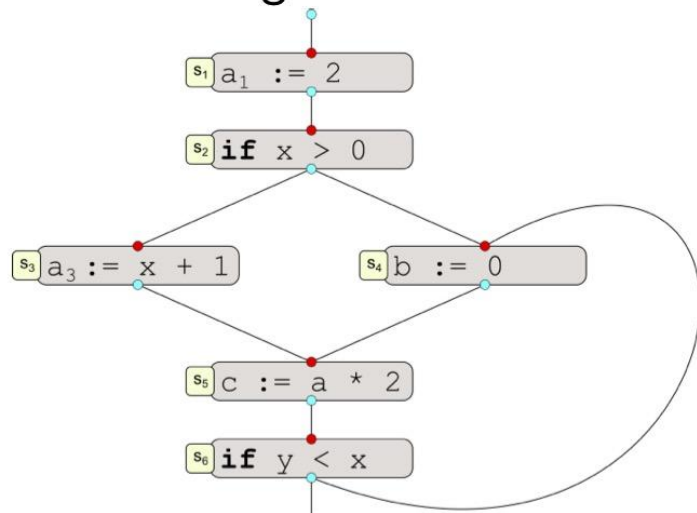
$$Out(s : d_i := \dots) = (In(s) - \{d_j; \forall j\}) \cup d_i$$

$$RD(s) = \bigcup_{\forall p: d_i = \dots \in Pred(s)} (RD(p) - \{d_j; \forall j\}) \cup \{d_i\}$$

Node	s ₁	s ₂	s ₃	s ₄	s ₅	s ₆
RD ⁴	∅	∅	∅	∅	∅	∅
	∅	a ₁	a ₁	a ₁	a ₁ , a ₃ , b	a ₁ , a ₃ , b, c
	∅					

Reaching Analysis with Dataflow Iterative Algorithm

Reaching definitions control flow example - Calculate RD sets?



$$In(s) = \bigcup_{\forall p \in Pred(s)} Out(p)$$

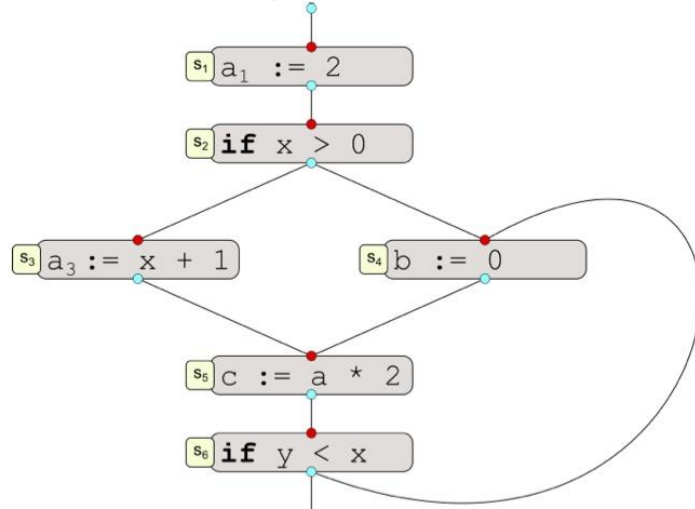
$$Out(s : d_i := \dots) = (In(s) - \{d_j; \forall j\}) \cup d_i$$

$$RD(s) = \bigcup_{\forall p: d_i = \dots \in Pred(s)} (RD(p) - \{d_j; \forall j\}) \cup \{d_i\}$$

Node	s ₁	s ₂	s ₃	s ₄	s ₅	s ₆
RD ⁴	∅	∅	∅	∅	∅	∅
	∅	a ₁	a ₁	a ₁	a ₁ , a ₃ , b	a ₁ , a ₃ , b, c
	∅	a ₁				

Reaching Analysis with Dataflow Iterative Algorithm

Reaching definitions control flow example - Calculate RD sets?



$$In(s) = \bigcup_{\forall p \in Pred(s)} Out(p)$$

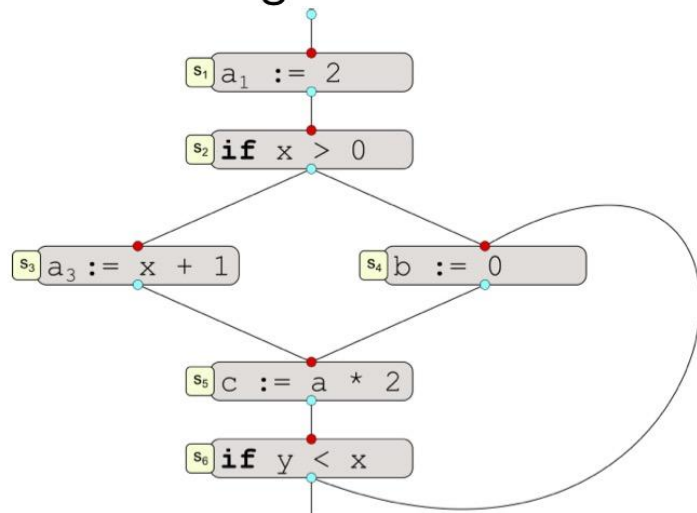
$$Out(s : d_i := \dots) = (In(s) - \{d_j; \forall j\}) \cup d_i$$

$$RD(s) = \bigcup_{\forall p: d_i = \dots \in Pred(s)} (RD(p) - \{d_j; \forall j\}) \cup \{d_i\}$$

Node	s ₁	s ₂	s ₃	s ₄	s ₅	s ₆
RD ⁴	∅	∅	∅	∅	∅	∅
	∅	a ₁	a ₁	a ₁	a ₁ , a ₃ , b	a ₁ , a ₃ , b, c
	∅	a ₁	a ₁			

Reaching Analysis with Dataflow Iterative Algorithm

Reaching definitions control flow example - Calculate RD sets?



$$In(s) = \bigcup_{\forall p \in Pred(s)} Out(p)$$

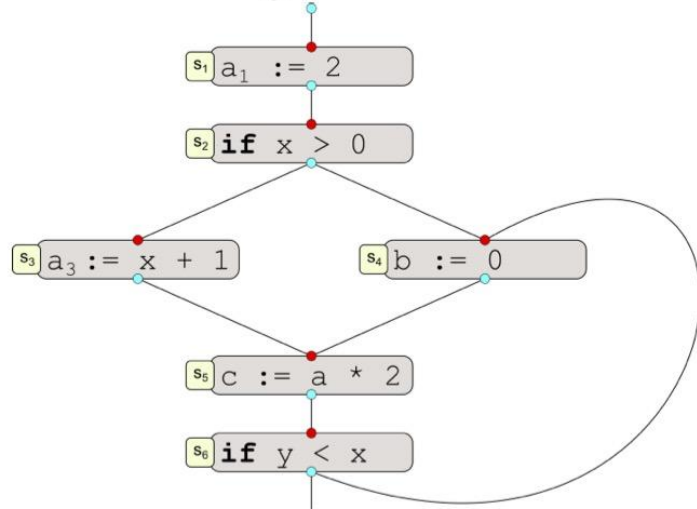
$$Out(s : d_i := \dots) = (In(s) - \{d_j; \forall j\}) \cup d_i$$

$$RD(s) = \bigcup_{\forall p: d_i = \dots \in Pred(s)} (RD(p) - \{d_j; \forall j\}) \cup \{d_i\}$$

Node	s ₁	s ₂	s ₃	s ₄	s ₅	s ₆
RD ⁴	∅	∅	∅	∅	∅	∅
	∅	a ₁	a ₁	a ₁	a ₁ , a ₃ , b	a ₁ , a ₃ , b, c
	∅	a ₁	a ₁	a ₁ , a ₃ , b, c		

Reaching Analysis with Dataflow Iterative Algorithm

Reaching definitions control flow example - Calculate RD sets?



$$In(s) = \bigcup_{\forall p \in Pred(s)} Out(p)$$

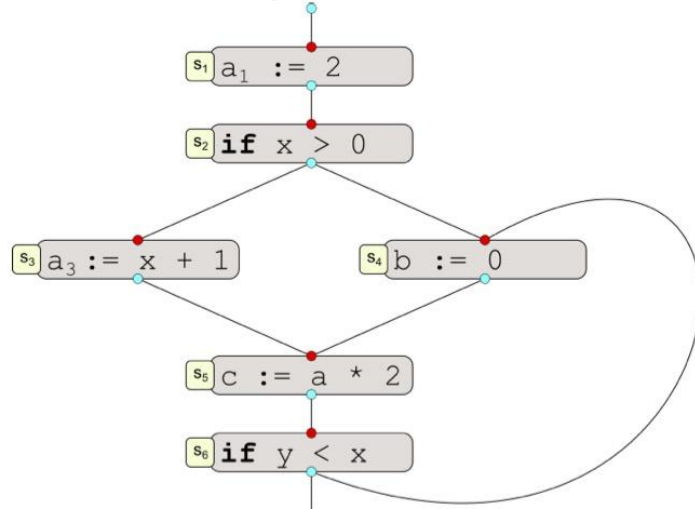
$$Out(s : d_i := \dots) = (In(s) - \{d_j; \forall j\}) \cup d_i$$

$$RD(s) = \bigcup_{\forall p: d_i = \dots \in Pred(s)} (RD(p) - \{d_j; \forall j\}) \cup \{d_i\}$$

Node	s ₁	s ₂	s ₃	s ₄	s ₅	s ₆
RD ⁴	∅	∅	∅	∅	∅	∅
	∅	a ₁	a ₁	a ₁	a ₁ , a ₃ , b	a ₁ , a ₃ , b, c
	∅	a ₁	a ₁	a ₁ , a ₃ , b, c	a ₁ , a ₃ , b, c	

Reaching Analysis with Dataflow Iterative Algorithm

Reaching definitions control flow example - Calculate RD sets?



$$In(s) = \bigcup_{\forall p \in Pred(s)} Out(p)$$

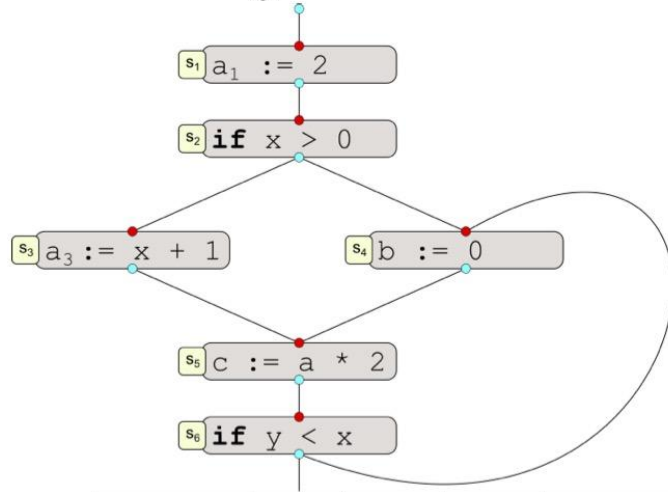
$$Out(s : d_i := \dots) = (In(s) - \{d_j; \forall j\}) \cup d_i$$

$$RD(s) = \bigcup_{\forall p: d_i = \dots \in Pred(s)} (RD(p) - \{d_j; \forall j\}) \cup \{d_i\}$$

Node	s ₁	s ₂	s ₃	s ₄	s ₅	s ₆
RD ⁴	∅	∅	∅	∅	∅	∅
	∅	a ₁	a ₁	a ₁	a ₁ , a ₃ , b	a ₁ , a ₃ , b, c
	∅	a ₁	a ₁	a ₁ , a ₃ , b, c	a ₁ , a ₃ , b, c	a ₁ , a ₃ , b, c

Reaching Analysis with Dataflow Iterative Algorithm

Reaching definitions control flow example - Calculate RD sets?



$$In(s) = \bigcup_{\forall p \in Pred(s)} Out(p)$$

$$Out(s : d_i := ...) = (In(s) - \{d_j; \forall j\}) \cup d_i$$

$$RD(s) = \bigcup_{\forall p: d_i = \dots \in Pred(s)} (RD(p) - \{d_j; \forall j\}) \cup \{d_i\}$$

Node	s ₁	s ₂	s ₃	s ₄	s ₅	s ₆
RD ⁴	∅	∅	∅	∅	∅	∅
	∅	a ₁	a ₁	a ₁	a ₁ , a ₃ , b	a ₁ , a ₃ , b, c
	∅	a ₁	a ₁	a ₁ , a ₃ , b, c	a ₁ , a ₃ , b, c	a ₁ , a ₃ , b, c
	∅	a ₁	a ₁	a ₁ , a ₃ , b, c	a ₁ , a ₃ , b, c	a ₁ , a ₃ , b, c

Iterative Algorithm *Termination*

Does the iterative round-robin is guaranteed to terminate?

Iterative Algorithm *Termination*

Does the iterative round-robin is guaranteed to terminate?

Yes!

- Each step of the iteration can only grow a set or leave unchanged
- Finite number of elements in each set, so finite number of times can change
- Each iteration either has a change or stops
- Therefore, must terminate!

Iterative Algorithm: *Improving Performance*

- Direction (forward vs. backward) can have a big impact on performance
- Round-Robin Algorithm is slow, may require many passes through nodes
- Can speed up by considering basic blocks, rather than individual nodes
- Only nodes which have inputs changed need to be processed, keep track with a work list

Liveness Analysis - What & why?

Intuition: A variable is *live* at a program point if its current value may be read during the remaining execution of the program; otherwise, the variable is *dead*.

Useful for *register allocation* and *dead code elimination*

```
1 int foo(int input) {  
2   int x,y,z;  
3   x = input;  
4   while (x > 1) {  
5     y = x / 2;  
6     if (y > 3) x = x - y;  
7     z = x - 4;  
8     if (z > 0) x = x / 2;  
9     z = z-1;  
10  }  
11  return x;  
12 }
```

Legal transformation
due to liveness information



```
1 int foo_opt(int input) {  
2   int x, yz;  
3   x = input;  
4   while (x > 1) {  
5     yz = x / 2;  
6     if (yz > 3) x = x - yz;  
7     yz = x-4;  
8     if (yz > 0) x = x / 2;  
9       
10  }  
11  return x;  
12 }
```

y and z can be stored
in the same register

Computation
was never used

Definition of Liveness

Definition

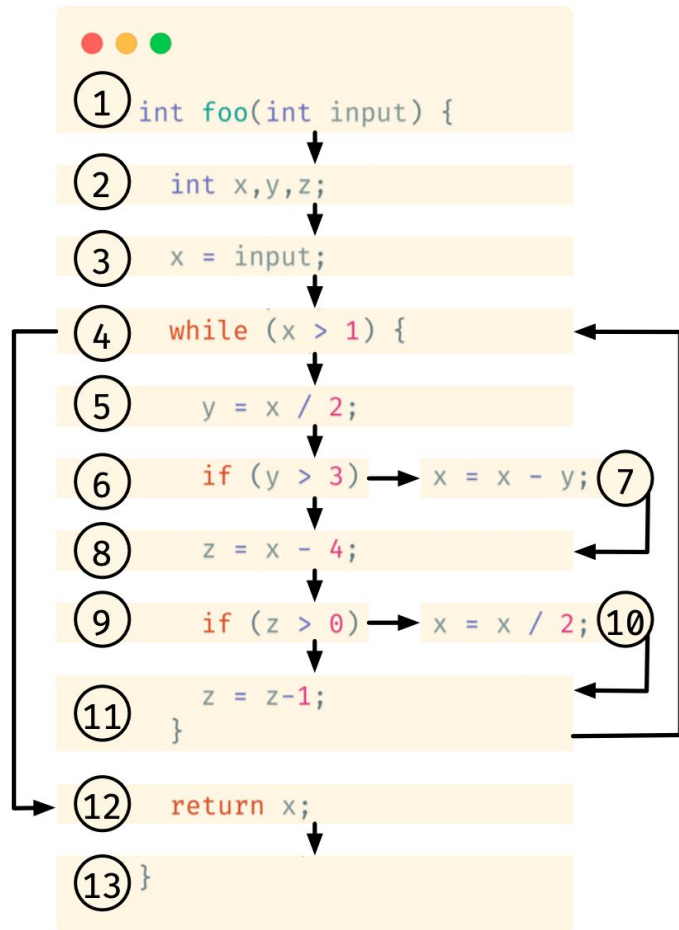
A variable v is live before a CFG node s if

1. $v \in use_{var}(s)$, or
2. $\exists$ a direct path from s to a node that uses v , and that path does not go through a node that defines (overrides) v .

Examples:

Is x live before $s = 5$? Yes, $x \in \{x\} = use_{var}(5)$

Is z live before $s = 5$? No, we first hit a def at 8



Backward Dataflow Analysis

- **Direction** – *backward*
- **Transfer function** – computes statement effect
$$In(s) = f_s(Out(s))$$
- **Meet operator** – merges values from multiple outcoming edges
$$Out(s) = \bigwedge_{\forall b \in Succ(s)} In(b)$$
- **Value set** – the information being passed around
e.g. Sets of variables
- **Initial values**
Should be most conservative value; Start node often a special case

Liveness as Dataflow Analysis

- **Direction** – *backward*
- **Transfer function** – computes statement effect
$$live(n) = (candidates(n) - def_{var}(n)) \cup use_{var}(n)$$
- **Meet operator** – merges values from multiple outcoming edges
$$candidates(n) = \bigcup_{\forall s \in Succ(n)} live(s)$$
- **Value set** – the information being passed around
Set of variables + Set of candidates
- **Initial values**
Empty sets

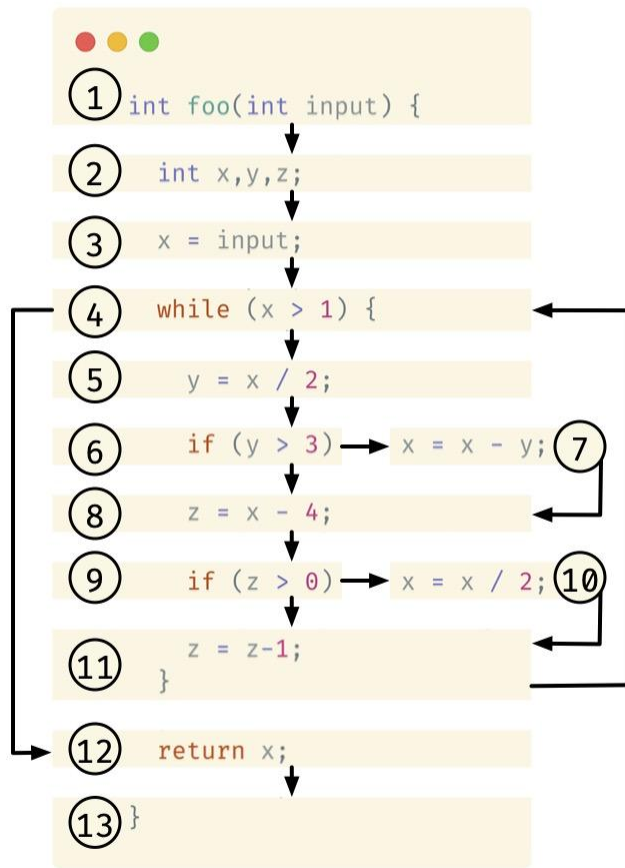
$Out(n)$	$= candidates(n)$
$In(n)$	$= live(n)$
$f_n(x)$	$= (x - def(n)) \cup use(n)$

Liveness Analysis Example

1st				
Node n	use _{var} (n)	def _{var} (n)	candidate[n]	live[n]
13	{ }	{ }		
12	{ x }	{ }		
11	{ z }	{ z }		
10	{ x }	{ x }		
9	{ z }	{ }		
8	{ x }	{ z }		
7	{ x, y }	{ x }		
6	{ y }	{ }		
5	{ x }	{ y }		
4	{ x }	{ }		
3	{ }	{ x }		
2	{ }	{ }		
1	{ }	{ }		

$$\text{candidates}[n] = \bigcup_{s \in \text{succ}(n)} \text{live}[s];$$

$$\text{live}[n] = \text{use}_{\text{var}}(n) \cup (\text{candidates}[n] - \text{def}_{\text{var}}(n));$$

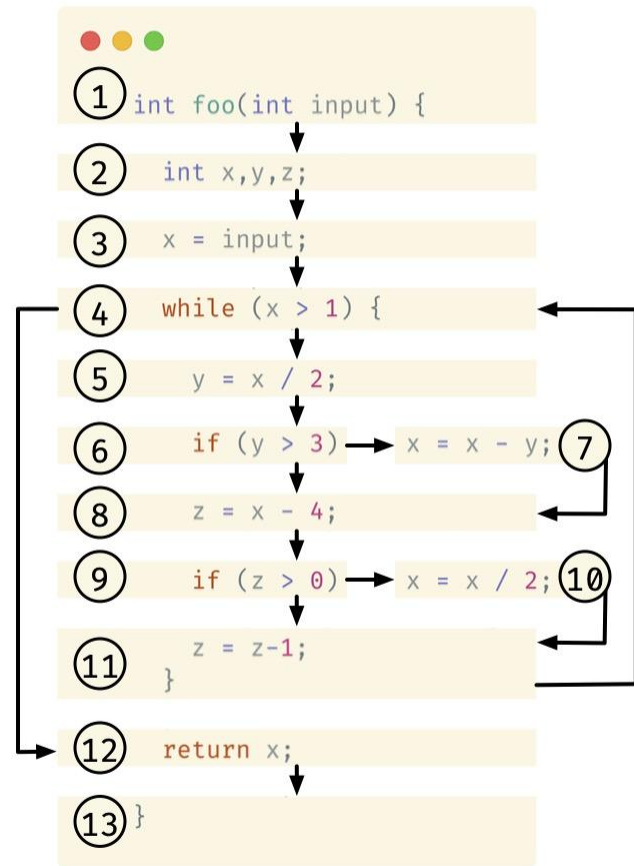


Liveness Analysis Example

1st				
Node n	use _{var} (n)	def _{var} (n)	candidate[n]	live[n]
13	{ }	{ }	{ }	{ }
12	{ x }	{ }	{ }	{ }
11	{ z }	{ z }	{ }	{ }
10	{ x }	{ x }	{ }	{ }
9	{ z }	{ }	{ }	{ }
8	{ x }	{ z }	{ }	{ }
7	{ x, y }	{ x }	{ }	{ }
6	{ y }	{ }	{ }	{ }
5	{ x }	{ y }	{ }	{ }
4	{ x }	{ }	{ }	{ }
3	{ }	{ x }	{ }	{ }
2	{ }	{ }	{ }	{ }
1	{ }	{ }	{ }	{ }

$$\text{candidates}[n] = \bigcup_{s \in \text{succ}(n)} \text{live}[s];$$

$$\text{live}[n] = \text{use}_{\text{var}}(n) \cup (\text{candidates}[n] - \text{def}_{\text{var}}(n));$$

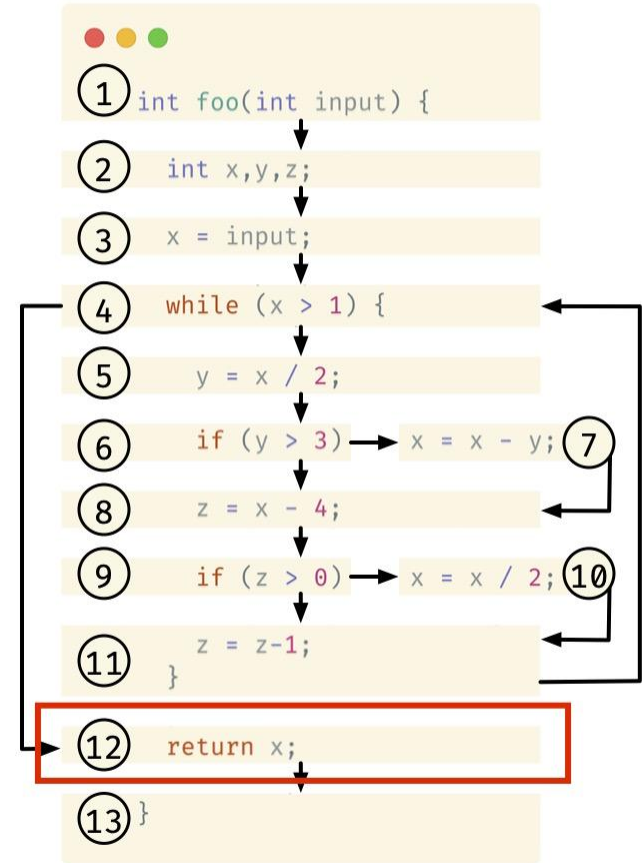


Liveness Analysis Example

1st				
Node n	use _{var} (n)	def _{var} (n)	candidate[n]	live[n]
13	{}	{}	{}	{}
12	{x}	{}	{}	{}
11	{z}	{z}	{}	{}
10	{x}	{x}	{}	{}
9	{z}	{}	{}	{}
8	{x}	{z}	{}	{}
7	{x, y}	{x}	{}	{}
6	{y}	{}	{}	{}
5	{x}	{y}	{}	{}
4	{x}	{}	{}	{}
3	{}	{x}	{}	{}
2	{}	{}	{}	{}
1	{}	{}	{}	{}

$$\text{candidates}[n] = \bigcup_{s \in \text{succ}(n)} \text{live}[s];$$

$$\text{live}[n] = \text{use}_{\text{var}}(n) \cup (\text{candidates}[n] - \text{def}_{\text{var}}(n));$$

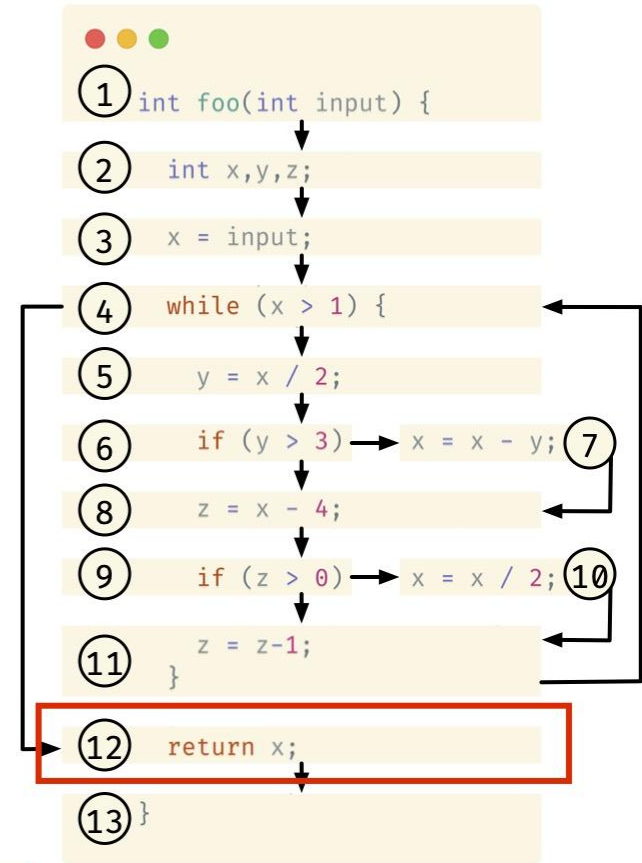


Liveness Analysis Example

1st				
Node n	use _{var} (n)	def _{var} (n)	candidate[n]	live[n]
13	{}	{}	{}	{}
12	{x}	{}	{} → {x}	
11	{z}	{z}	{}	{}
10	{x}	{x}	{}	{}
9	{z}	{}	{}	{}
8	{x}	{z}	{}	{}
7	{x, y}	{x}	{}	{}
6	{y}	{}	{}	{}
5	{x}	{y}	{}	{}
4	{x}	{}	{}	{}
3	{}	{x}	{}	{}
2	{}	{}	{}	{}
1	{}	{}	{}	{}

$$\text{candidates}[n] = \bigcup_{s \in \text{succ}(n)} \text{live}[s];$$

$$\text{live}[n] = \text{use}_{\text{var}}(n) \cup (\text{candidates}[n] - \text{def}_{\text{var}}(n));$$



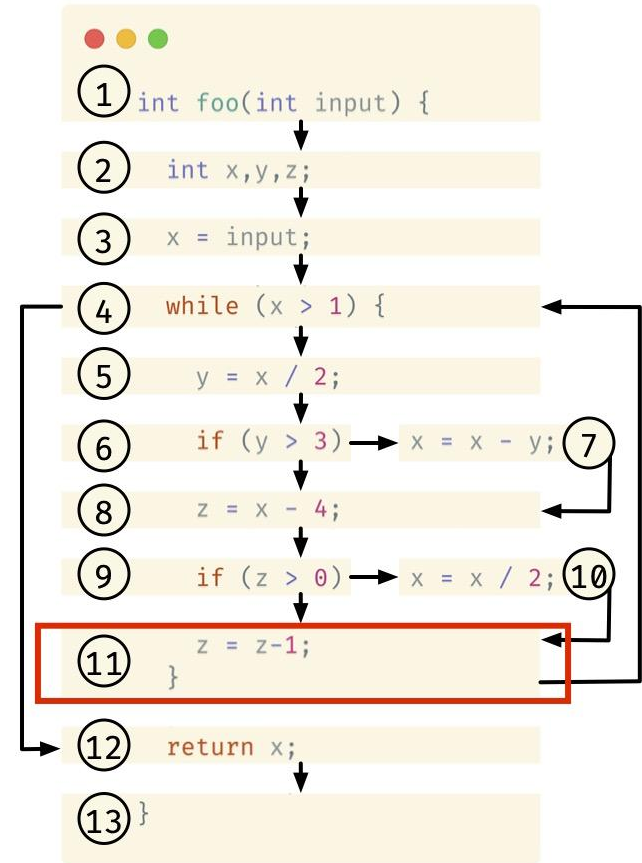
Liveness Analysis Example

1st

Node n	use _{var} (n)	def _{var} (n)	candidate[n]	live[n]
13	{}	{}	{}	{}
12	{x}	{}	{}	{x}
11	{z}	{z}	{}	{}
10	{x}	{x}	{}	{}
9	{z}	{}	{}	{}
8	{x}	{z}	{}	{}
7	{x, y}	{x}	{}	{}
6	{y}	{}	{}	{}
5	{x}	{y}	{}	{}
4	{x}	{}	{}	{}
3	{}	{x}	{}	{}
2	{}	{}	{}	{}
1	{}	{}	{}	{}

$$\text{candidates}[n] = \bigcup_{s \in \text{succ}(n)} \text{live}[s];$$

$$\text{live}[n] = \text{use}_{\text{var}}(n) \cup (\text{candidates}[n] - \text{def}_{\text{var}}(n));$$



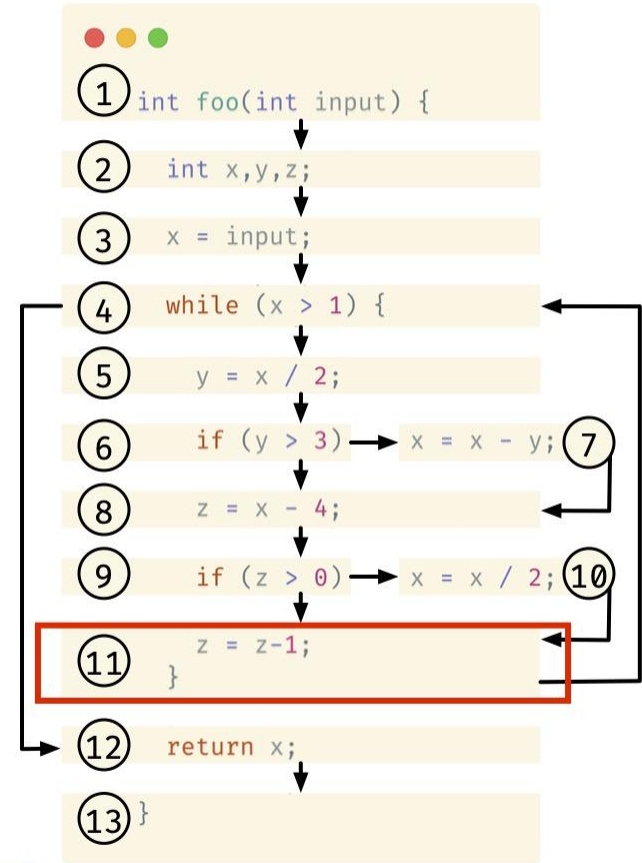
Liveness Analysis Example

1st

Node n	use _{var} (n)	def _{var} (n)	candidate[n]	live[n]
13	{}	{}	{}	{}
12	{x}	{}	{}	{x}
11	{z}	{z}	{}	{z}
10	{x}	{x}	{}	{}
9	{z}	{}	{}	{}
8	{x}	{z}	{}	{}
7	{x, y}	{x}	{}	{}
6	{y}	{}	{}	{}
5	{x}	{y}	{}	{}
4	{x}	{}	{}	{}
3	{}	{x}	{}	{}
2	{}	{}	{}	{}
1	{}	{}	{}	{}

$$\text{candidates}[n] = \bigcup_{s \in \text{succ}(n)} \text{live}[s];$$

$$\text{live}[n] = \text{use}_{\text{var}}(n) \cup (\text{candidates}[n] - \text{def}_{\text{var}}(n));$$

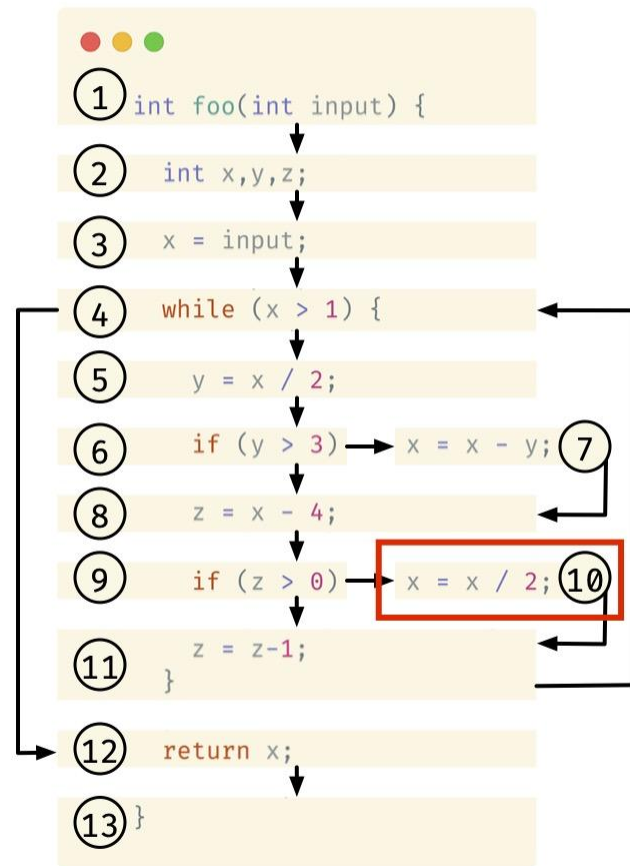


Liveness Analysis Example

1st				
Node n	use _{var} (n)	def _{var} (n)	candidate[n]	live[n]
13	{ }	{ }	{ }	{ }
12	{x}	{ }	{ }	{x}
11	{z}	{z}	{ }	{z}
10	{x}	{x}	{z}	{ }
9	{z}	{ }	{ }	{ }
8	{x}	{z}	{ }	{ }
7	{x,y}	{x}	{ }	{ }
6	{y}	{ }	{ }	{ }
5	{x}	{y}	{ }	{ }
4	{x}	{ }	{ }	{ }
3	{ }	{x}	{ }	{ }
2	{ }	{ }	{ }	{ }
1	{ }	{ }	{ }	{ }

$$\text{candidates}[n] = \bigcup_{s \in \text{succ}(n)} \text{live}[s];$$

$$\text{live}[n] = \text{use}_{\text{var}}(n) \cup (\text{candidates}[n] - \text{def}_{\text{var}}(n));$$



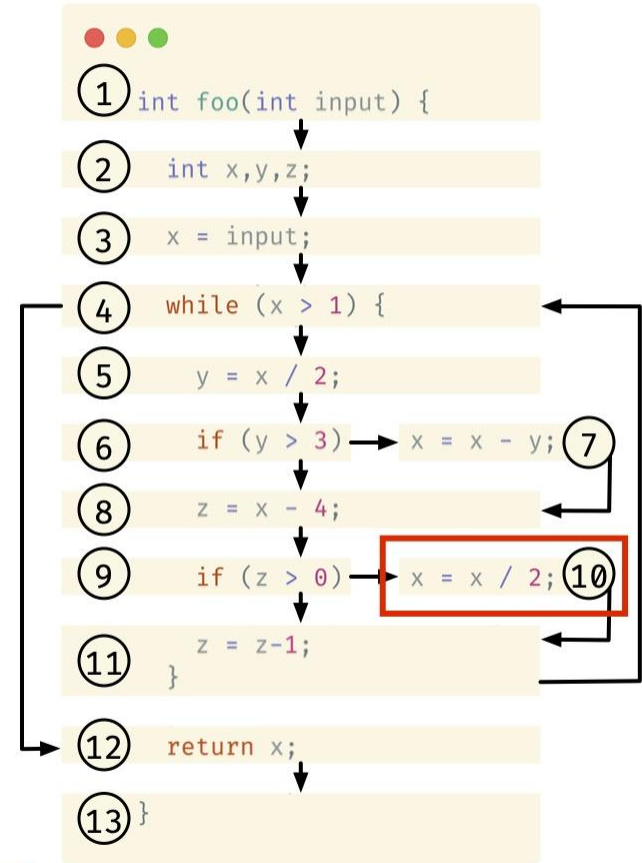
Liveness Analysis Example

1st

Node n	use _{var} (n)	def _{var} (n)	candidate[n]	live[n]
13	{}	{}	{}	{}
12	{x}	{}	{}	{x}
11	{z}	{z}	{}	{z}
10	{x}	{x}	{z} → {x, z}	
9	{z}	{}	{}	{}
8	{x}	{z}	{}	{}
7	{x, y}	{x}	{}	{}
6	{y}	{}	{}	{}
5	{x}	{y}	{}	{}
4	{x}	{}	{}	{}
3	{}	{x}	{}	{}
2	{}	{}	{}	{}
1	{}	{}	{}	{}

$$\text{candidates}[n] = \bigcup_{s \in \text{succ}(n)} \text{live}[s];$$

$$\text{live}[n] = \text{use}_{\text{var}}(n) \cup (\text{candidates}[n] - \text{def}_{\text{var}}(n));$$

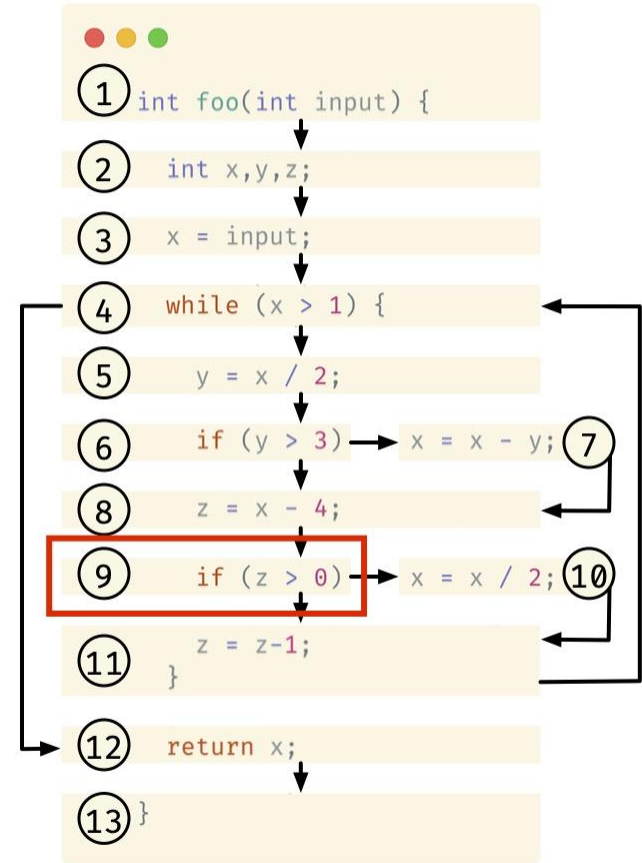


Liveness Analysis Example

1st				
Node n	use _{var} (n)	def _{var} (n)	candidate[n]	live[n]
13	{}	{}	{}	{}
12	{x}	{}	{}	{x}
11	{z}	{z}	{}	{z}
10	{x}	{x}	{z}	{x, z}
9	{z}	{}	{x, z}	{}
8	{x}	{z}	{}	{}
7	{x, y}	{x}	{}	{}
6	{y}	{}	{}	{}
5	{x}	{y}	{}	{}
4	{x}	{}	{}	{}
3	{}	{x}	{}	{}
2	{}	{}	{}	{}
1	{}	{}	{}	{}

$$\text{candidates}[n] = \bigcup_{s \in \text{succ}(n)} \text{live}[s];$$

$$\text{live}[n] = \text{use}_{\text{var}}(n) \cup (\text{candidates}[n] - \text{def}_{\text{var}}(n));$$

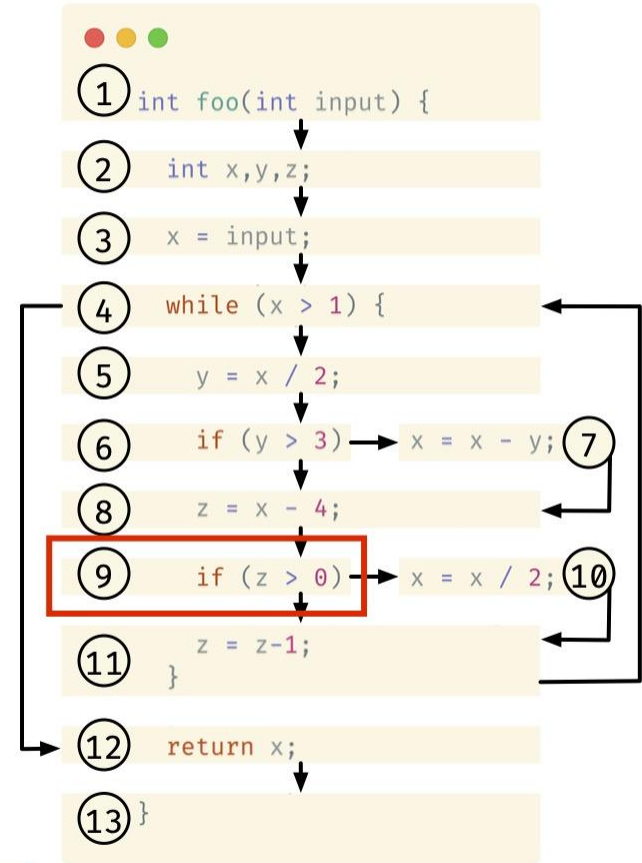


Liveness Analysis Example

1st				
Node n	use _{var} (n)	def _{var} (n)	candidate[n]	live[n]
13	{ }	{ }	{ }	{ }
12	{ x }	{ }	{ }	{ x }
11	{ z }	{ z }	{ }	{ z }
10	{ x }	{ x }	{ z }	{ x, z }
9	{ z }	{ }	{ x, z }	{ x, z }
8	{ x }	{ z }	{ }	{ }
7	{ x, y }	{ x }	{ }	{ }
6	{ y }	{ }	{ }	{ }
5	{ x }	{ y }	{ }	{ }
4	{ x }	{ }	{ }	{ }
3	{ }	{ x }	{ }	{ }
2	{ }	{ }	{ }	{ }
1	{ }	{ }	{ }	{ }

$$\text{candidates}[n] = \bigcup_{s \in \text{succ}(n)} \text{live}[s];$$

$$\text{live}[n] = \text{use}_{\text{var}}(n) \cup (\text{candidates}[n] - \text{def}_{\text{var}}(n));$$



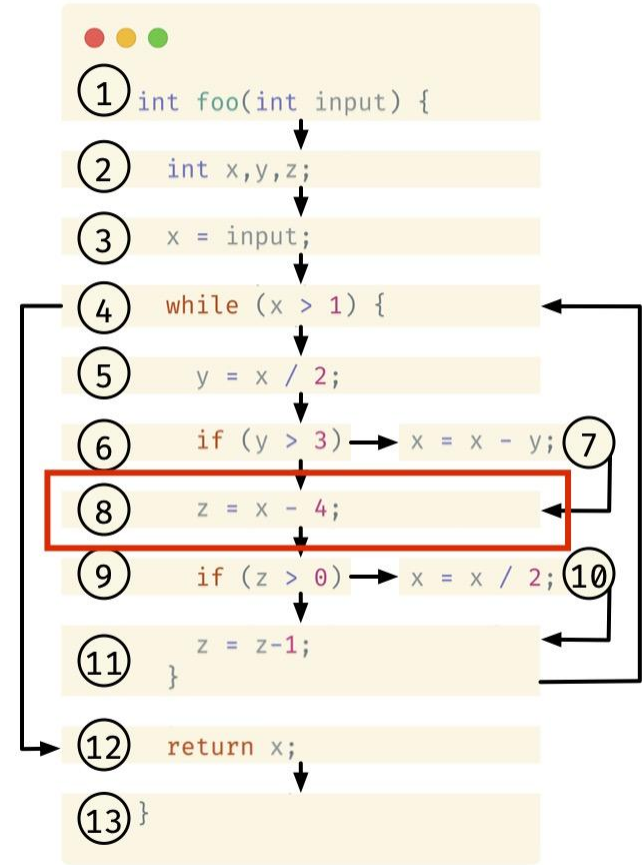
Liveness Analysis Example

1st

Node n	use _{var} (n)	def _{var} (n)	candidate[n]	live[n]
13	{}	{}	{}	{}
12	{x}	{}	{}	{x}
11	{z}	{z}	{}	{z}
10	{x}	{x}	{z}	{x, z}
9	{z}	{}	{x, z}	{x, z}
8	{x}	{z}	{x, z}	{}
7	{x, y}	{x}	{}	{}
6	{y}	{}	{}	{}
5	{x}	{y}	{}	{}
4	{x}	{}	{}	{}
3	{}	{x}	{}	{}
2	{}	{}	{}	{}
1	{}	{}	{}	{}

$$\text{candidates}[n] = \bigcup_{s \in \text{succ}(n)} \text{live}[s];$$

$$\text{live}[n] = \text{use}_{\text{var}}(n) \cup (\text{candidates}[n] - \text{def}_{\text{var}}(n));$$

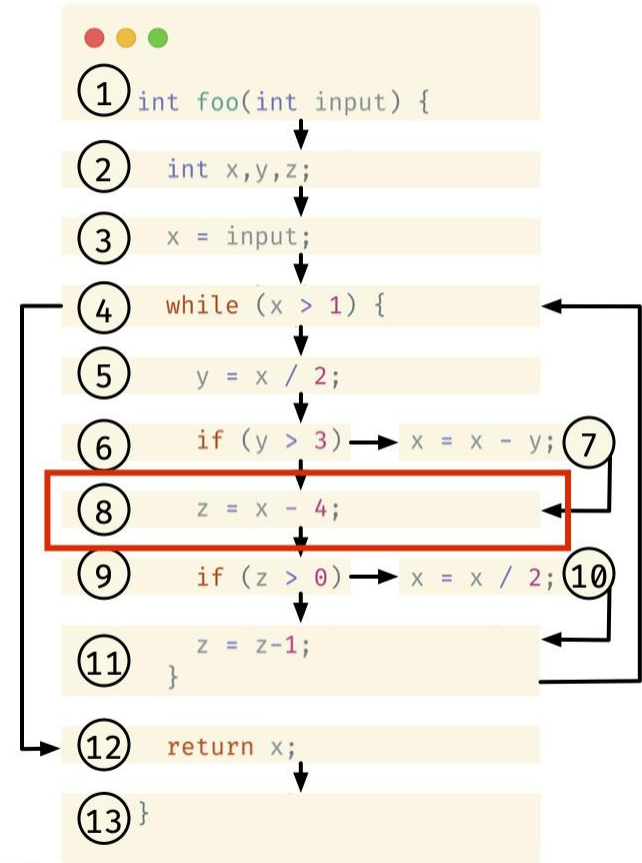


Liveness Analysis Example

1st				
Node n	use _{var} (n)	def _{var} (n)	candidate[n]	live[n]
13	{ }	{ }	{ }	{ }
12	{ x }	{ }	{ }	{ x }
11	{ z }	{ z }	{ }	{ z }
10	{ x }	{ x }	{ z }	{ x, z }
9	{ z }	{ }	{ x, z }	{ x, z }
8	{ x }	{ z }	{ x, z } → { x }	{ x }
7	{ x, y }	{ x }	{ }	{ }
6	{ y }	{ }	{ }	{ }
5	{ x }	{ y }	{ }	{ }
4	{ x }	{ }	{ }	{ }
3	{ }	{ x }	{ }	{ }
2	{ }	{ }	{ }	{ }
1	{ }	{ }	{ }	{ }

$$\text{candidates}[n] = \bigcup_{s \in \text{succ}(n)} \text{live}[s];$$

$$\text{live}[n] = \text{use}_{\text{var}}(n) \cup (\text{candidates}[n] - \text{def}_{\text{var}}(n));$$

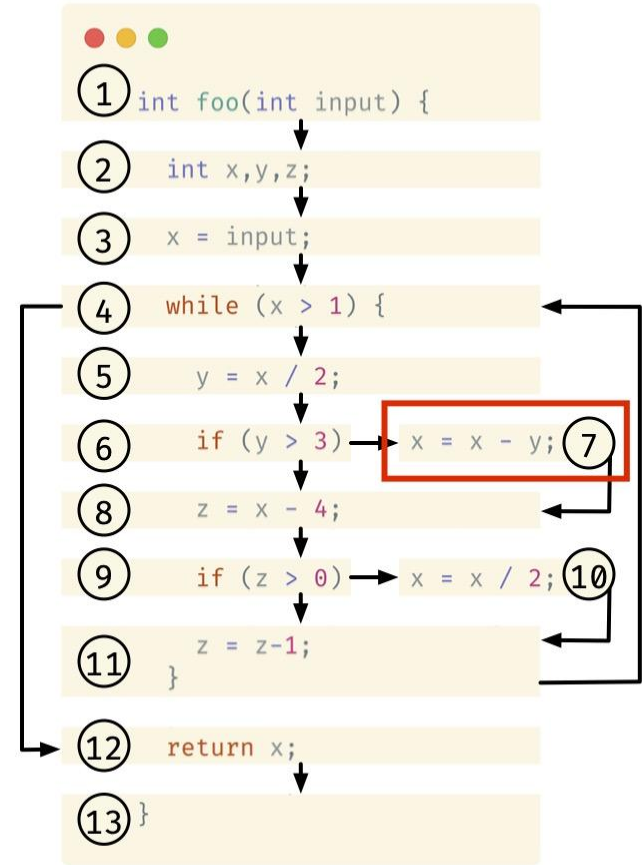


Liveness Analysis Example

1st				
Node n	use _{var} (n)	def _{var} (n)	candidate[n]	live[n]
13	{ }	{ }	{ }	{ }
12	{ x }	{ }	{ }	{ x }
11	{ z }	{ z }	{ }	{ z }
10	{ x }	{ x }	{ z }	{ x, z }
9	{ z }	{ }	{ x, z }	{ x, z }
8	{ x }	{ z }	{ x, z }	{ x }
7	{ x, y }	{ x }	{ x }	{ x, y }
6	{ y }	{ }	{ }	{ }
5	{ x }	{ y }	{ }	{ }
4	{ x }	{ }	{ }	{ }
3	{ }	{ x }	{ }	{ }
2	{ }	{ }	{ }	{ }
1	{ }	{ }	{ }	{ }

$$\text{candidates}[n] = \bigcup_{s \in \text{succ}(n)} \text{live}[s];$$

$$\text{live}[n] = \text{use}_{\text{var}}(n) \cup (\text{candidates}[n] - \text{def}_{\text{var}}(n));$$

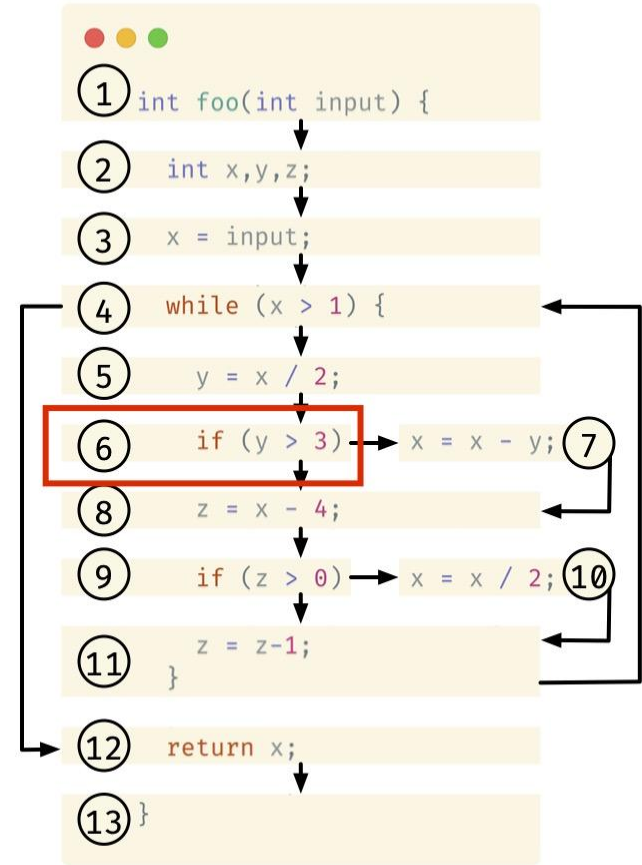


Liveness Analysis Example

1st				
Node n	use _{var} (n)	def _{var} (n)	candidate[n]	live[n]
13	{}	{}	{}	{}
12	{x}	{}	{}	{x}
11	{z}	{z}	{}	{z}
10	{x}	{x}	{z}	{x, z}
9	{z}	{}	{x, z}	{x, z}
8	{x}	{z}	{x, z}	{x}
7	{x, y}	{x}	{x}	{x, y}
6	{y}	{}	{x, y}	{x, y}
5	{x}	{y}	{}	{}
4	{x}	{}	{}	{}
3	{}	{x}	{}	{}
2	{}	{}	{}	{}
1	{}	{}	{}	{}

$$\text{candidates}[n] = \bigcup_{s \in \text{succ}(n)} \text{live}[s];$$

$$\text{live}[n] = \text{use}_{\text{var}}(n) \cup (\text{candidates}[n] - \text{def}_{\text{var}}(n));$$

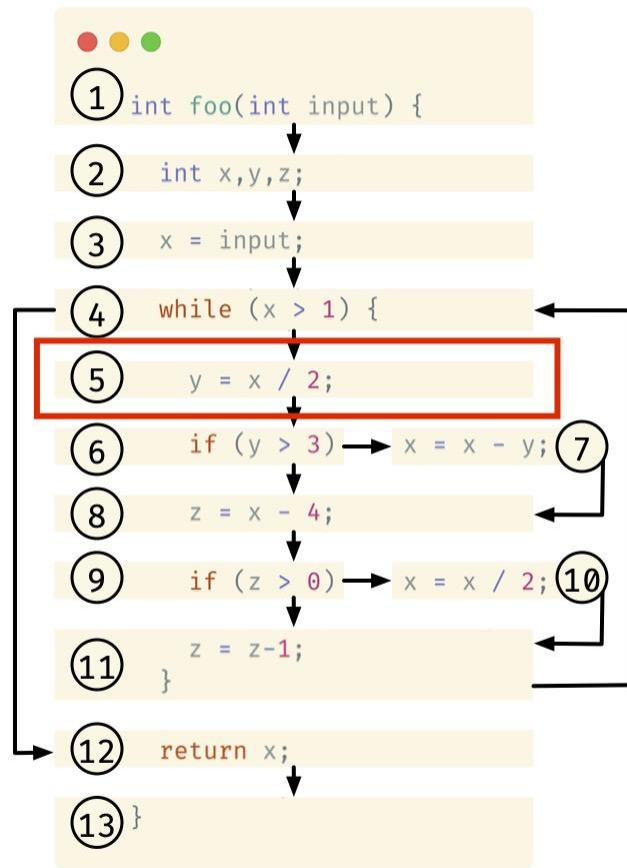


Liveness Analysis Example

1st				
Node n	use _{var} (n)	def _{var} (n)	candidate[n]	live[n]
13	{ }	{ }	{ }	{ }
12	{ x }	{ }	{ }	{ x }
11	{ z }	{ z }	{ }	{ z }
10	{ x }	{ x }	{ z }	{ x, z }
9	{ z }	{ }	{ x, z }	{ x, z }
8	{ x }	{ z }	{ x, z }	{ x }
7	{ x, y }	{ x }	{ x }	{ x, y }
6	{ y }	{ }	{ x, y }	{ x, y }
5	{ x }	{ y }	{ x, y }	{ x }
4	{ x }	{ }	{ }	{ }
3	{ }	{ x }	{ }	{ }
2	{ }	{ }	{ }	{ }
1	{ }	{ }	{ }	{ }

$$\text{candidates}[n] = \bigcup_{s \in \text{succ}(n)} \text{live}[s];$$

$$\text{live}[n] = \text{use}_{\text{var}}(n) \cup (\text{candidates}[n] - \text{def}_{\text{var}}(n));$$

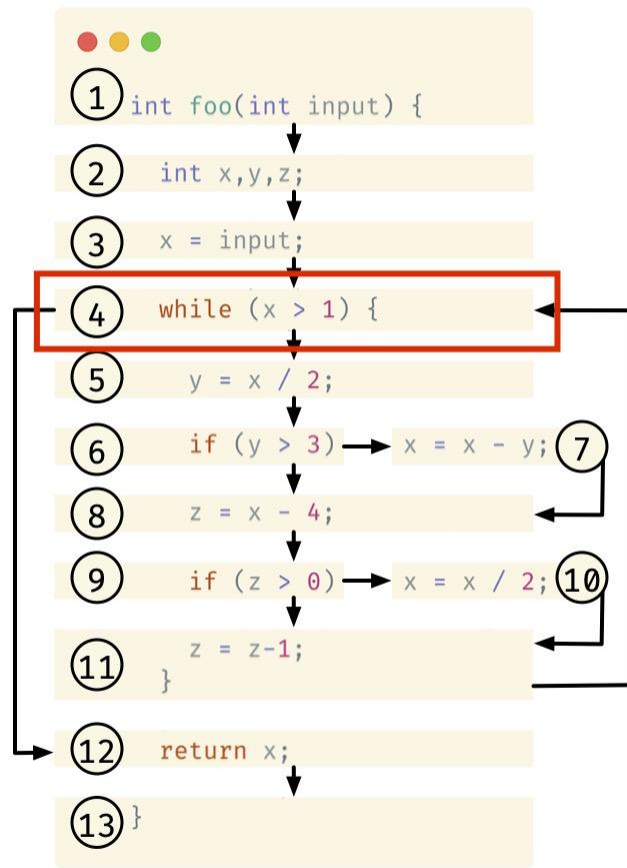


Liveness Analysis Example

1st				
Node n	use _{var} (n)	def _{var} (n)	candidate[n]	live[n]
13	{ }	{ }	{ }	{ }
12	{ x }	{ }	{ }	{ x }
11	{ z }	{ z }	{ }	{ z }
10	{ x }	{ x }	{ z }	{ x, z }
9	{ z }	{ }	{ x, z }	{ x, z }
8	{ x }	{ z }	{ x, z }	{ x }
7	{ x, y }	{ x }	{ x }	{ x, y }
6	{ y }	{ }	{ x, y }	{ x, y }
5	{ x }	{ y }	{ x, y }	{ x }
4	{ x }	{ }	{ x }	{ x }
3	{ }	{ x }	{ }	{ }
2	{ }	{ }	{ }	{ }
1	{ }	{ }	{ }	{ }

$$\text{candidates}[n] = \bigcup_{s \in \text{succ}(n)} \text{live}[s];$$

$$\text{live}[n] = \text{use}_{\text{var}}(n) \cup (\text{candidates}[n] - \text{def}_{\text{var}}(n));$$

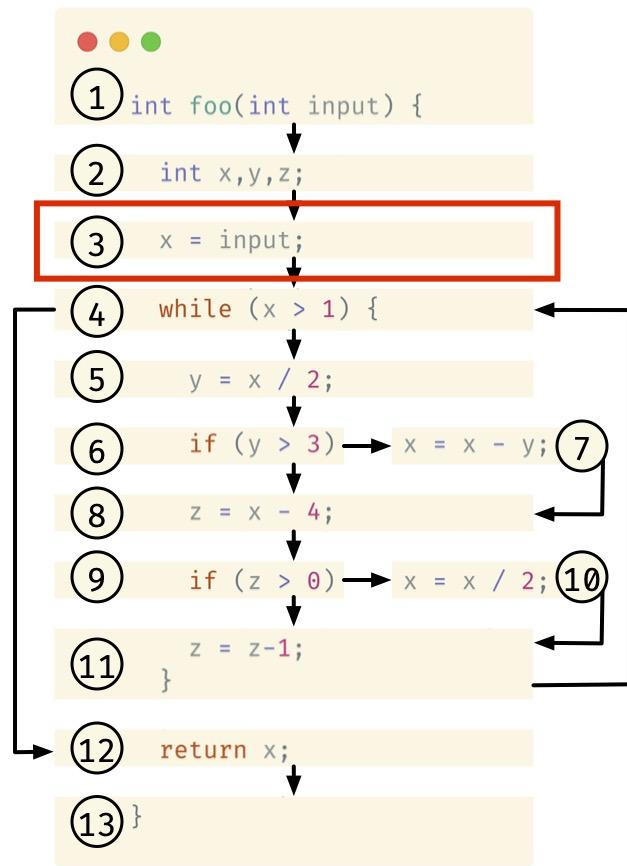


Liveness Analysis Example

1st				
Node n	use _{var} (n)	def _{var} (n)	candidate[n]	live[n]
13	{ }	{ }	{ }	{ }
12	{ x }	{ }	{ }	{ x }
11	{ z }	{ z }	{ }	{ z }
10	{ x }	{ x }	{ z }	{ x, z }
9	{ z }	{ }	{ x, z }	{ x, z }
8	{ x }	{ z }	{ x, z }	{ x }
7	{ x, y }	{ x }	{ x }	{ x, y }
6	{ y }	{ }	{ x, y }	{ x, y }
5	{ x }	{ y }	{ x, y }	{ x }
4	{ x }	{ }	{ x }	{ x }
3	{ }	{ x }	{ x }	{ }
2	{ }	{ }	{ }	{ }
1	{ }	{ }	{ }	{ }

$$\text{candidates}[n] = \bigcup_{s \in \text{succ}(n)} \text{live}[s];$$

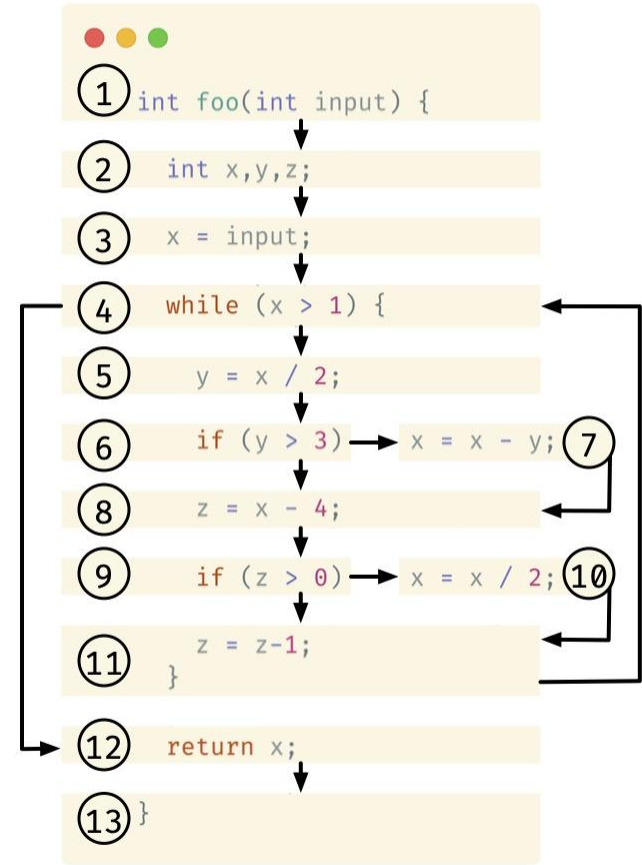
$$\text{live}[n] = \text{use}_{\text{var}}(n) \cup (\text{candidates}[n] - \text{def}_{\text{var}}(n));$$



Liveness Analysis Example

1st				
Node n	use _{var} (n)	def _{var} (n)	candidate[n]	live[n]
13	{ }	{ }	{ }	{ }
12	{ x }	{ }	{ }	{ x }
11	{ z }	{ z }	{ }	{ z }
10	{ x }	{ x }	{ z }	{ x, z }
9	{ z }	{ }	{ x, z }	{ x, z }
8	{ x }	{ z }	{ x, z }	{ x }
7	{ x, y }	{ x }	{ x }	{ x, y }
6	{ y }	{ }	{ x, y }	{ x, y }
5	{ x }	{ y }	{ x, y }	{ x }
4	{ x }	{ }	{ x }	{ x }
3	{ }	{ x }	{ x }	{ }
2	{ }	{ }	{ }	{ }
1	{ }	{ }	{ }	{ }

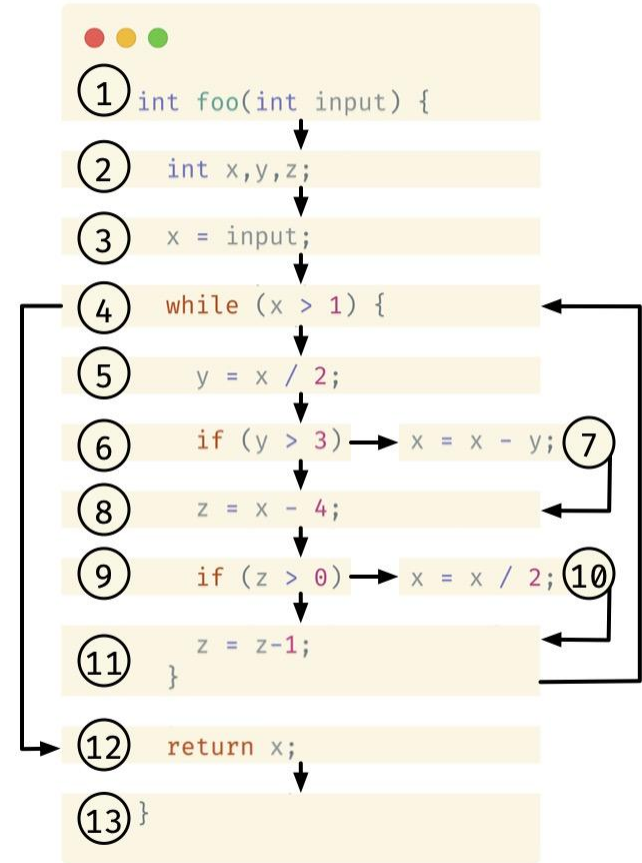
Completed fist iteration



Liveness Analysis Example

Node n	use _{var} (n)	def _{var} (n)	1st		2nd	
			candidate[n]	live[n]	candidate[n]	live[n]
13	{ }	{ }	{ }	{ }	{ }	{ }
12	{ x }	{ }	{ }	{ x }	{ }	{ x }
11	{ z }	{ z }	{ }	{ z }	{ x }	{ x , z }
10	{ x }	{ x }	{ z }	{ x, z }	{ x, z }	{ x, z }
9	{ z }	{ }	{ x, z }	{ x, z }	{ x, z }	{ x, z }
8	{ x }	{ z }	{ x, z }	{ x }	{ x, z }	{ x }
7	{ x, y }	{ x }	{ x }	{ x, y }	{ x }	{ x, y }
6	{ y }	{ }	{ x, y }	{ x, y }	{ x, y }	{ x, y }
5	{ x }	{ y }	{ x, y }	{ x }	{ x, y }	{ x }
4	{ x }	{ }	{ x }	{ x }	{ x }	{ x }
3	{ }	{ x }	{ x }	{ }	{ x }	{ }
2	{ }	{ }	{ }	{ }	{ }	{ }
1	{ }	{ }	{ }	{ }	{ }	{ }

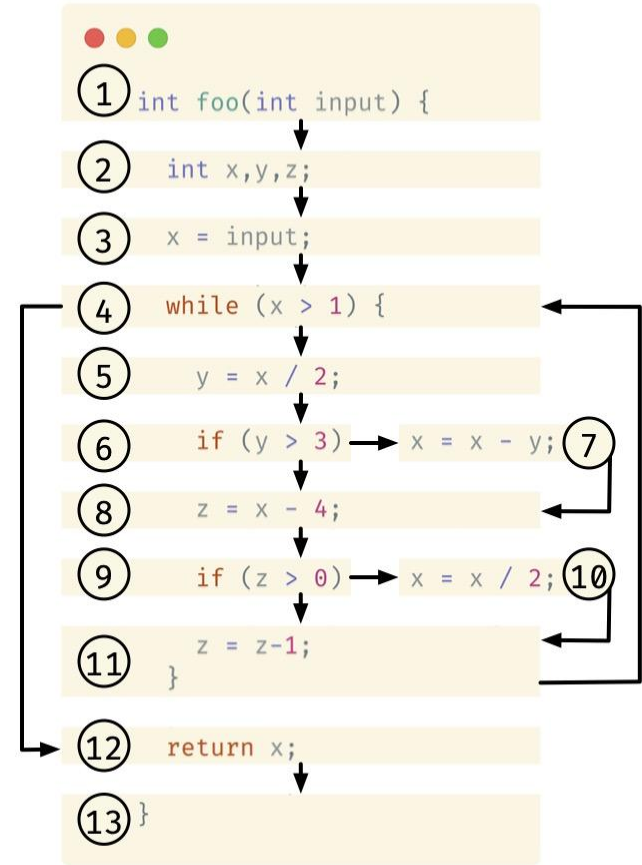
Completed second iteration



Liveness Analysis Example

Node n	use _{var} (n)	def _{var} (n)	1st		2nd		3rd	
			candidate[n]	live[n]	candidate[n]	live[n]	candidate[n]	live[n]
13	{ }	{ }	{ }	{ }	{ }	{ }	{ }	{ }
12	{ x }	{ }	{ }	{ x }	{ }	{ x }	{ }	{ x }
11	{ z }	{ z }	{ }	{ z }	{ x }	{ x, z }	{ x }	{ x, z }
10	{ x }	{ x }	{ z }	{ x, z }	{ x, z }	{ x, z }	{ x, z }	{ x, z }
9	{ z }	{ }	{ x, z }	{ x, z }	{ x, z }	{ x, z }	{ x, z }	{ x, z }
8	{ x }	{ z }	{ x, z }	{ x }	{ x, z }	{ x }	{ x, z }	{ x }
7	{ x, y }	{ x }	{ x }	{ x, y }	{ x }	{ x, y }	{ x }	{ x, y }
6	{ y }	{ }	{ x, y }	{ x, y }	{ x, y }	{ x, y }	{ x, y }	{ x, y }
5	{ x }	{ y }	{ x, y }	{ x }	{ x, y }	{ x }	{ x, y }	{ x }
4	{ x }	{ }	{ x }	{ x }	{ x }	{ x }	{ x }	{ x }
3	{ }	{ x }	{ x }	{ }	{ x }	{ }	{ x }	{ }
2	{ }	{ }	{ }	{ }	{ }	{ }	{ }	{ }
1	{ }	{ }	{ }	{ }	{ }	{ }	{ }	{ }

No changes: fixpoint reached

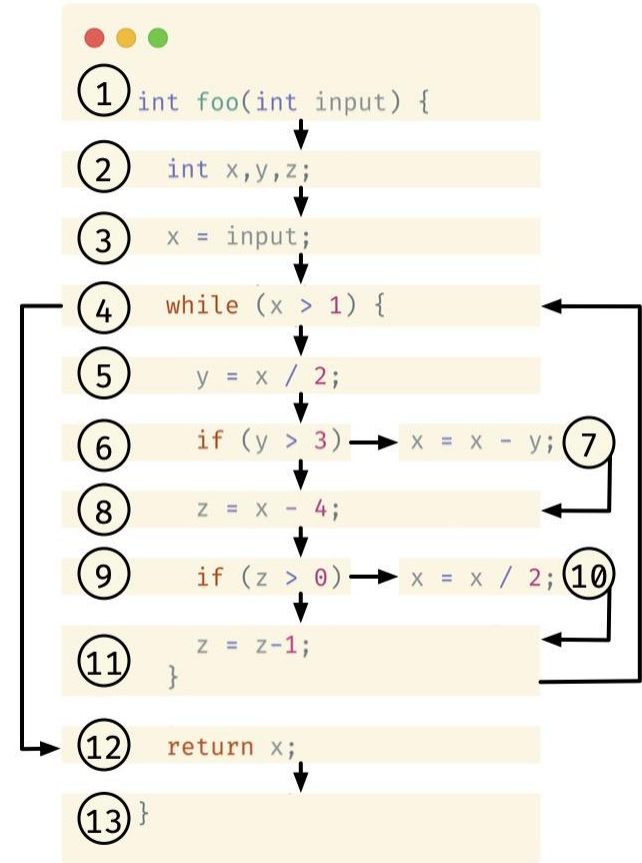


Liveness Analysis Example

Node n	use _{var} (n)	def _{var} (n)	1st		2nd		3rd	
			candidate[n]	live[n]	candidate[n]	live[n]	candidate[n]	live[n]
13	{}	{}	{}	{}	{}	{}	{}	{}
12	{x}	{}	{}	{x}	{}	{x}	{}	{x}
11	{z}	{z}	{}	{z}	{x}	{x, z}	{x}	{x, z}
10	{x}	{x}	{z}	{x, z}	{x, z}	{x, z}	{x, z}	{x, z}
9	{z}	{}	{x, z}	{x, z}	{x, z}	{x, z}	{x, z}	{x, z}
8	{x}	{z}	{x, z}	{x}	{x, z}	{x}	{x, z}	{x}
7	{x, y}	{x}	{x}	{x, y}	{x}	{x, y}	{x}	{x, y}
6	{y}	{}	{x, y}	{x, y}	{x, y}	{x, y}	{x, y}	{x, y}
5	{x}	{y}	{x, y}	{x}	{x, y}	{x}	{x, y}	{x}
4	{x}	{}	{x}	{x}	{x}	{x}	{x}	{x}
3	{}	{x}	{x}	{}	{x}	{}	{x}	{}
2	{}	{}	{}	{}	{}	{}	{}	{}
1	{}	{}	{}	{}	{}	{}	{}	{}

No changes: fixpoint reached

y and z
are never
live together



Data flow Analysis *Limitations*

Data flow analysis has some limitations:

- Static analysis may be (very) conservative
- CFG is only a static approximation of the dynamic control flow
- Pointers introduce aliases:
 - E.g. `*x = 10`; Does x point to another variable, y or z?
That would give a definition of y or z. May not know at compile time which ...
 - Precise alias analysis still an open problem
- Array access; generally cannot tell which indices are used
- Reasoning across function calls ...