Transitional Semantics-based Abstract Interpretation

Woosuk Lee

CSE 6049 Program Analysis

Hanyang University, Korea

Some slides are borrowed from http://ropas.snu.ac.kr/~kwang/4541.664A/21/8.pdf

Goal of This Lecture

- See how to instantiate abstract interpretation framework for languages based on a transitional semantics
- Examples: Sign & Interval analysis

Semantics Style: Compositional vs. Transitional

- Compositional semantics is defined by the semantics of sub-parts of a program.

$$[\![AB]\!] = \cdots [\![A]\!] \cdots [\![B]\!] \cdots$$

- For some realistic languages, even defining their compositional ("denotational") semantics is not obvious.

  - goto, exceptions, function pointers, dynamic method dispatches, …

Semantics Style: Compositional vs. Transitional

- Transitional-style (“operational”) semantics avoids the hurdle.

$$[\![AB]\!] = \{s_1 \to s_2 \to \cdots\}$$

- In the transitional style, all the *intermediate* states of program executions are exposed.

Roadmap

- Concrete semantics: a set of reachable states
- Abstract semantics: an abstract memory *at each program location (or labels)*
- *Worklist algorithm* for efficient fixpoint computation

Informal Overview: Concrete Interpretation (Standard Semantics)

```
1:  x := 0;
2:  y := 0;
3:  while (x < 10) {
4:    x := x + 1;
5:    y := y + 1;
6:  }
7:  skip
```

Integers uniquely assigned to every statement

Execution Trace :

$(\underline{\text{Labels}} \times (\text{Var} \rightarrow \mathbb{Z}))^{+}$

$(1, \{x \longmapsto 0\})$

$(2, \{x \longmapsto 0, y \longmapsto 0\})$

$(3, \{x \longmapsto 0, y \longmapsto 0\})$

$(4, \{x \longmapsto 1, y \longmapsto 0\})$

$(5, \{x \longmapsto 1, y \longmapsto 1\})$

$(3, \{x \longmapsto 1, y \longmapsto 1\})$

. . .

$(3, \{x \longmapsto 10, y \longmapsto 10\})$

$(7, \{x \longmapsto 10, y \longmapsto 10\})$

Informal Overview: Concrete Interpretation (Collecting Semantics)

```
1:  x := 0;
2:  y := 0;
3:  while (x < 10) {
4:    x := x + 1;
5:    y := y + 1;
6:  }
7:  skip
```

Partitioned Execution Traces:

Labels → $2^{\text{Var} \to \text{Z}}$

```
(1,{{x ⟼ 0}})
(2,{{x ⟼ 0, y ⟼ 0}})
(3,{{x ⟼ 0, y ⟼ 0},
    {x ⟼ 1, y ⟼ 1},
    . . .
    {x ⟼ 10, y ⟼ 10}})
. . .
(7,{x ⟼ 10, y ⟼ 10})
```

Informal Overview: Abstract Interpretation (Abstract Semantics)

```
1:   x := 0;
2:   y := 0;
3:   while (x < 10) {
4:     x := x + 1;
5:     y := y + 1;
6:   }
7:   skip
```

Abstract State:

: Labels → (Var → Interval)

(1, {x ⟼ [0, 0]})

(2, {x ⟼ [0, 0], y ⟼ [0, 0]})

(3, {x ⟼ [0, 10], y ⟼ [0, 10])

(4, {x ⟼ [1, 10], y ⟼ [0, 9])

. . .

The possible value of x ranges from 1 to 10 **after** executing line 4.

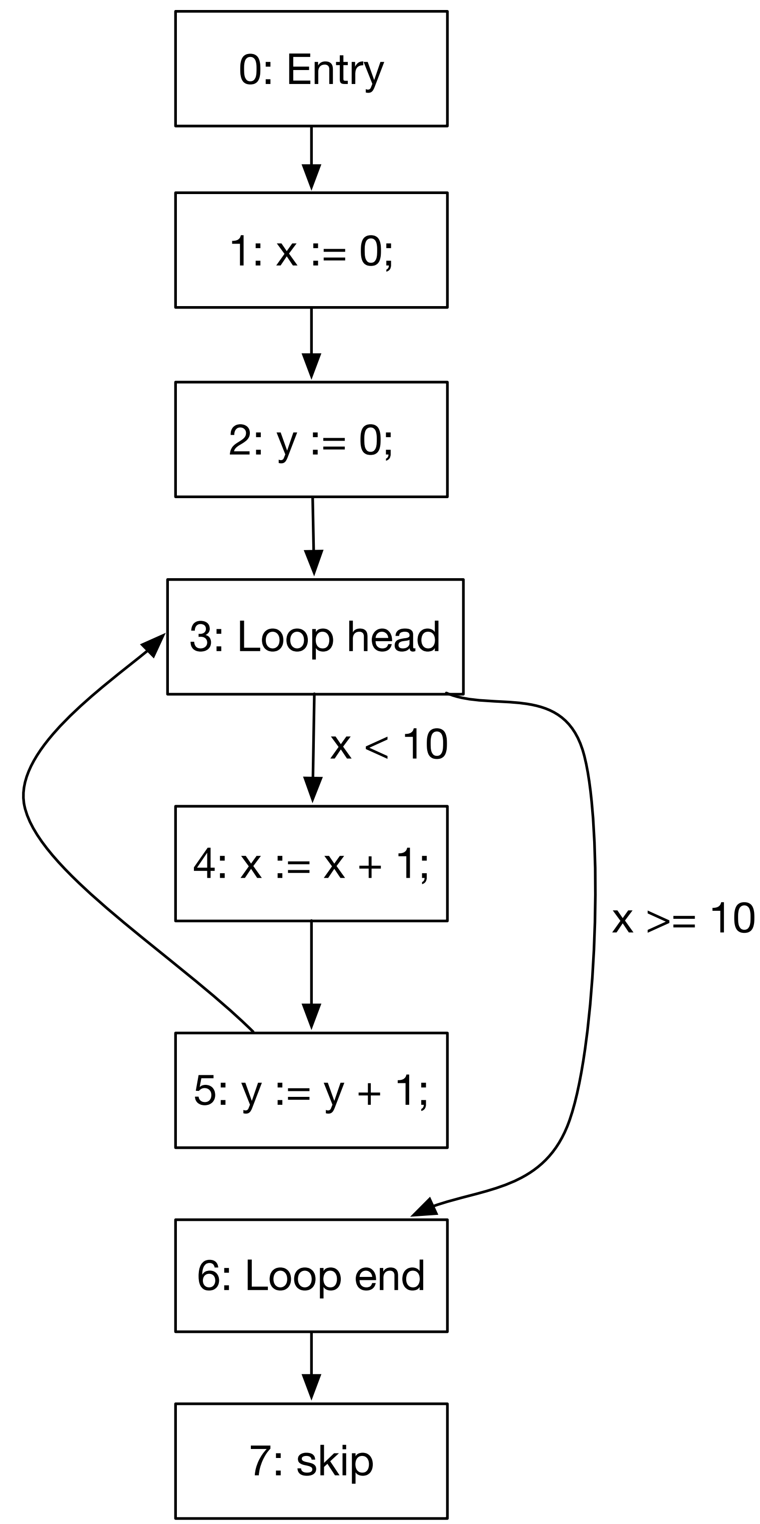
(7, {x ⟼ [10, 10], y ⟼ [10, 10]})

Informal Overview: Performing Fixpoint Computation

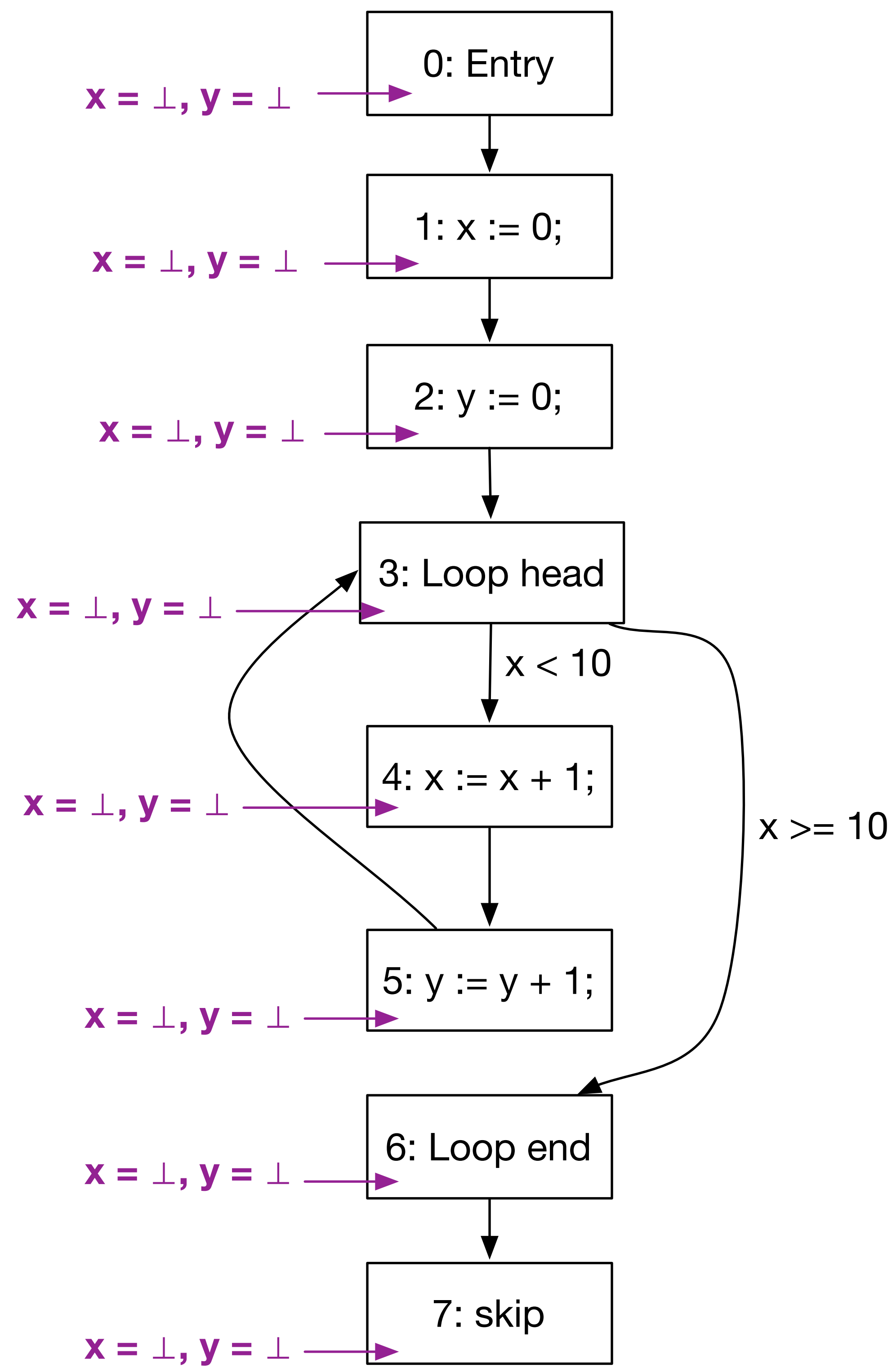
- Represent program as a *control flow graph*
- Compute abstract state at every program point
- Initialize all abstract states to $\bot$
- Repeat until no abstract state changes at any program point
  - For each program label `l`, compute an abstract state at entry to `l` by taking the join of `l`'s predecessors
  - Given the abstract state at entry to `l`, execute at each program point using abstract semantics

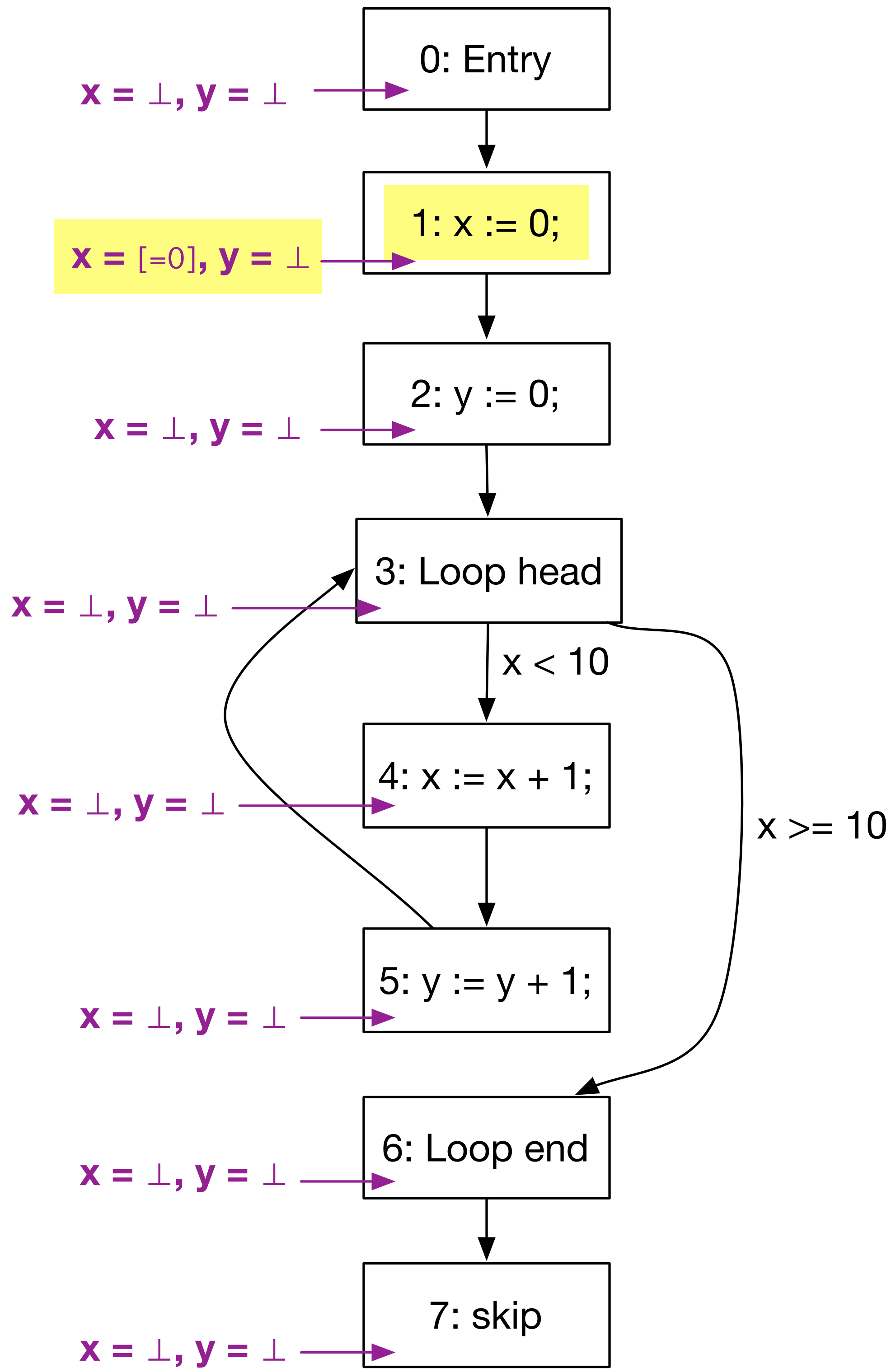
Program as a Graph

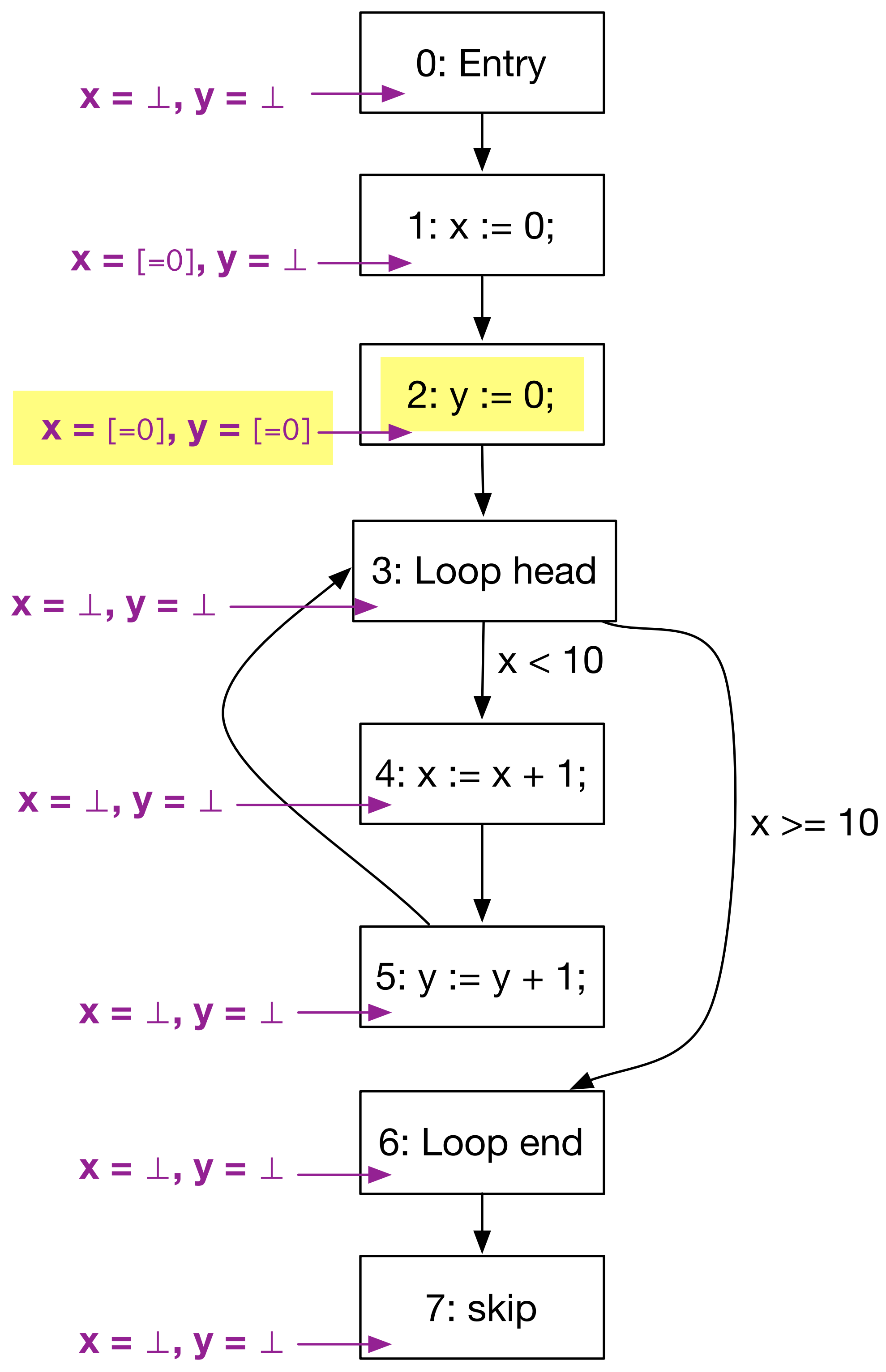
```
1:   x := 0;
2:   y := 0;
3:   while (x < 10) {
4:     x := x + 1;
5:     y := y + 1;
6:   }
7:   skip
```



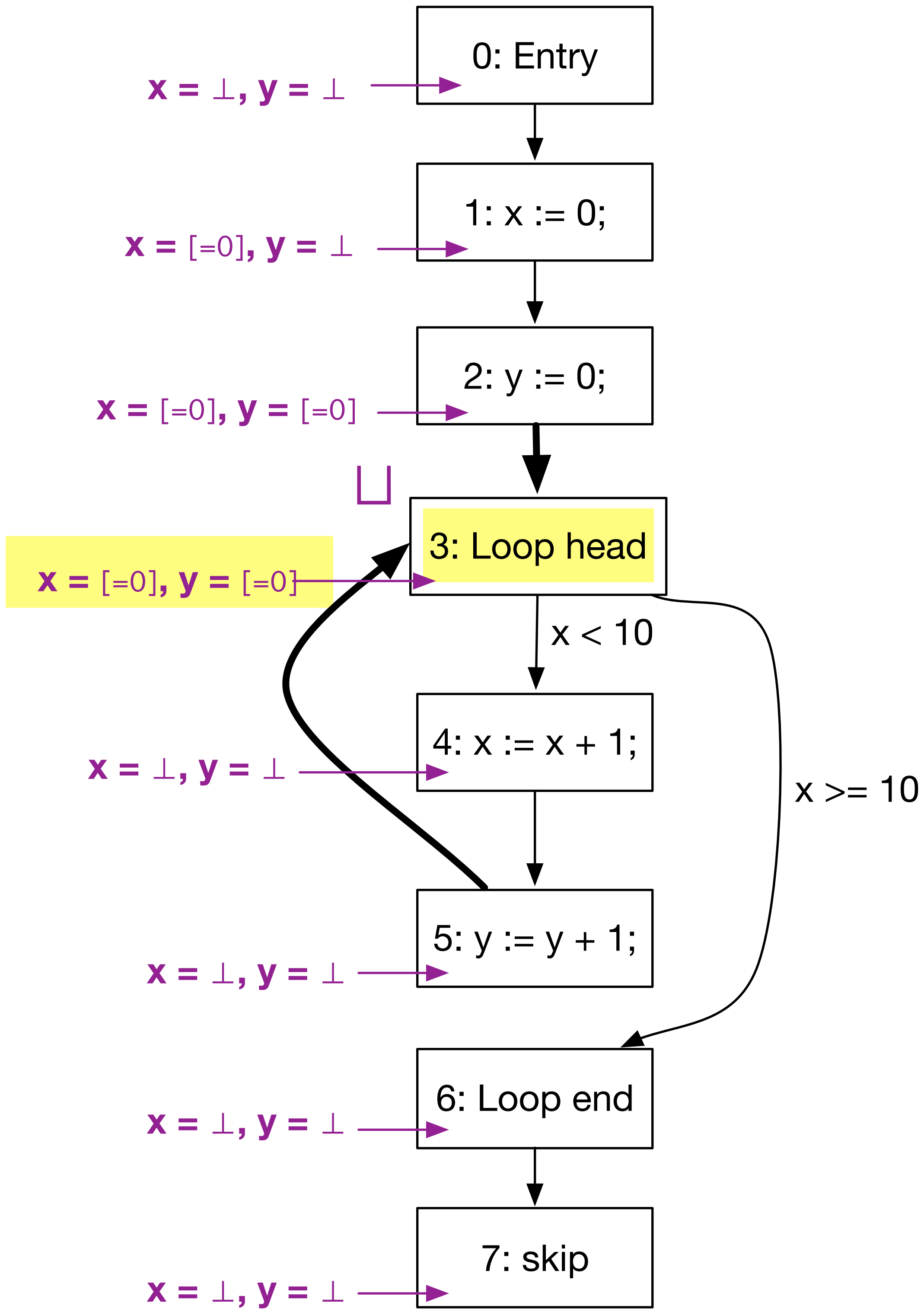
Sign Analysis



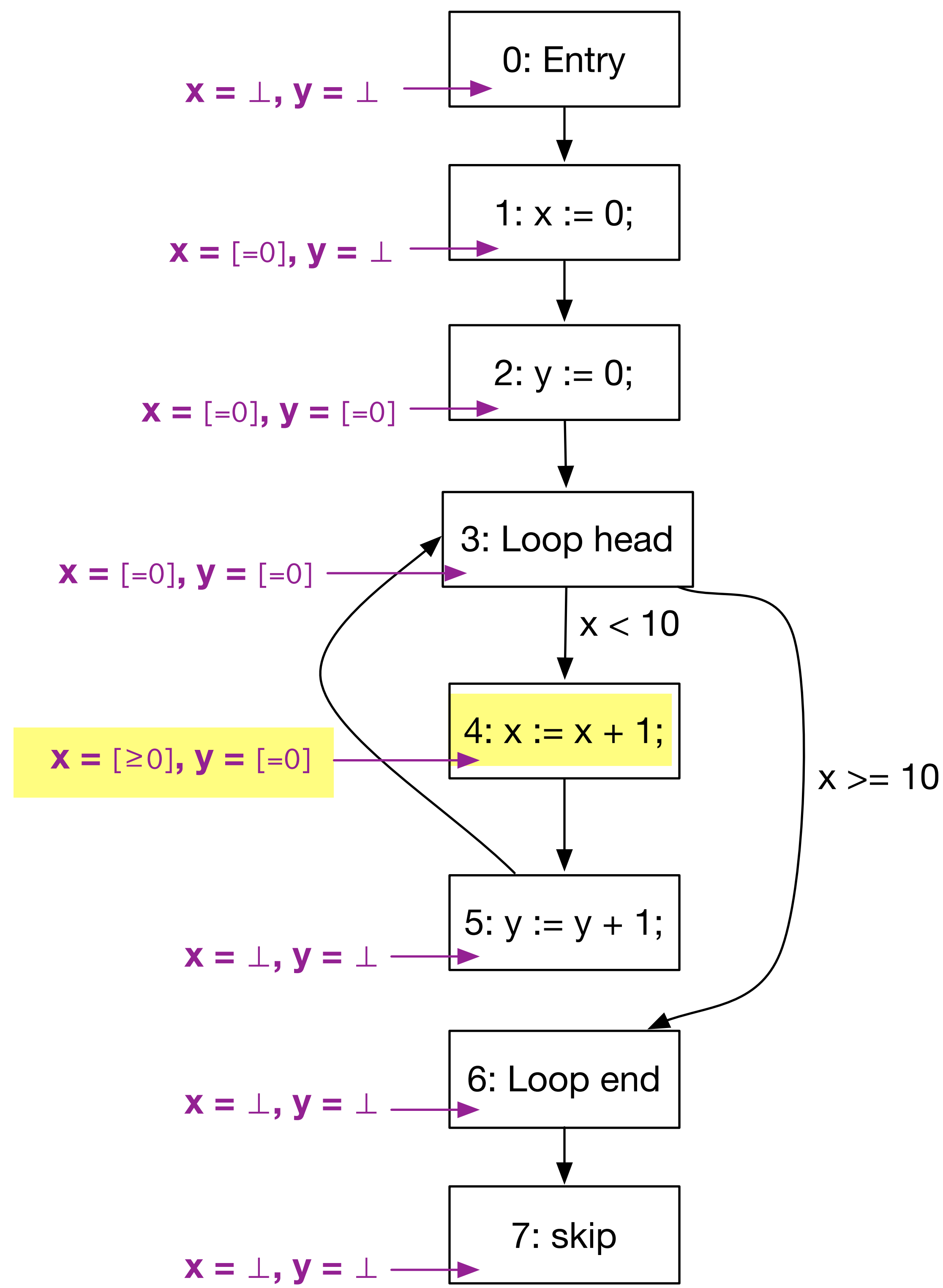

0: Entry
x = ⊥, y = ⊥
1: x := 0;
x = [=0], y = ⊥
2: y := 0;
x = ⊥, y = ⊥
3: Loop head
x = ⊥, y = ⊥
x < 10
4: x := x + 1;
x = ⊥, y = ⊥
x >= 10
5: y := y + 1;
x = ⊥, y = ⊥
6: Loop end
x = ⊥, y = ⊥
7: skip
x = ⊥, y = ⊥

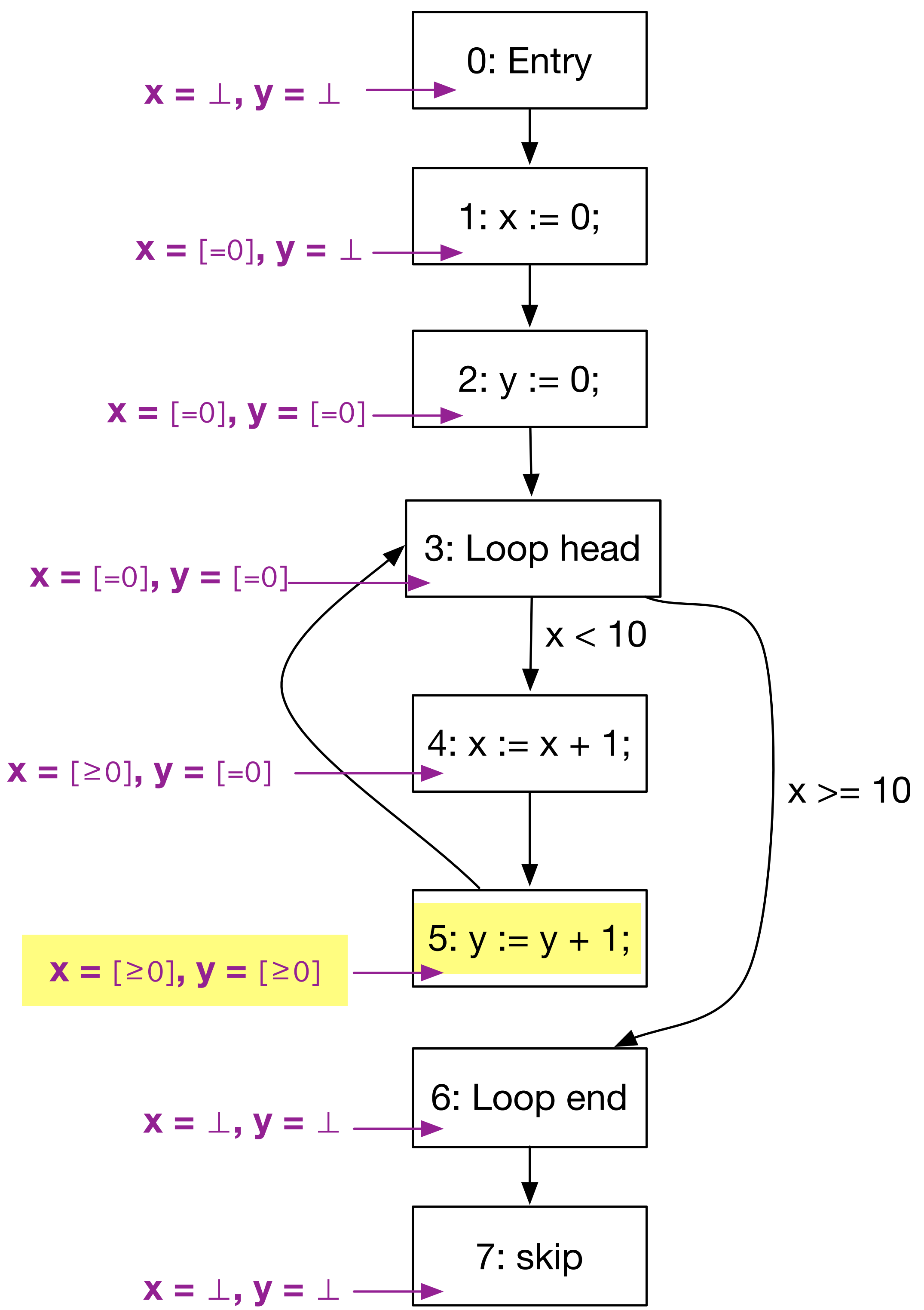


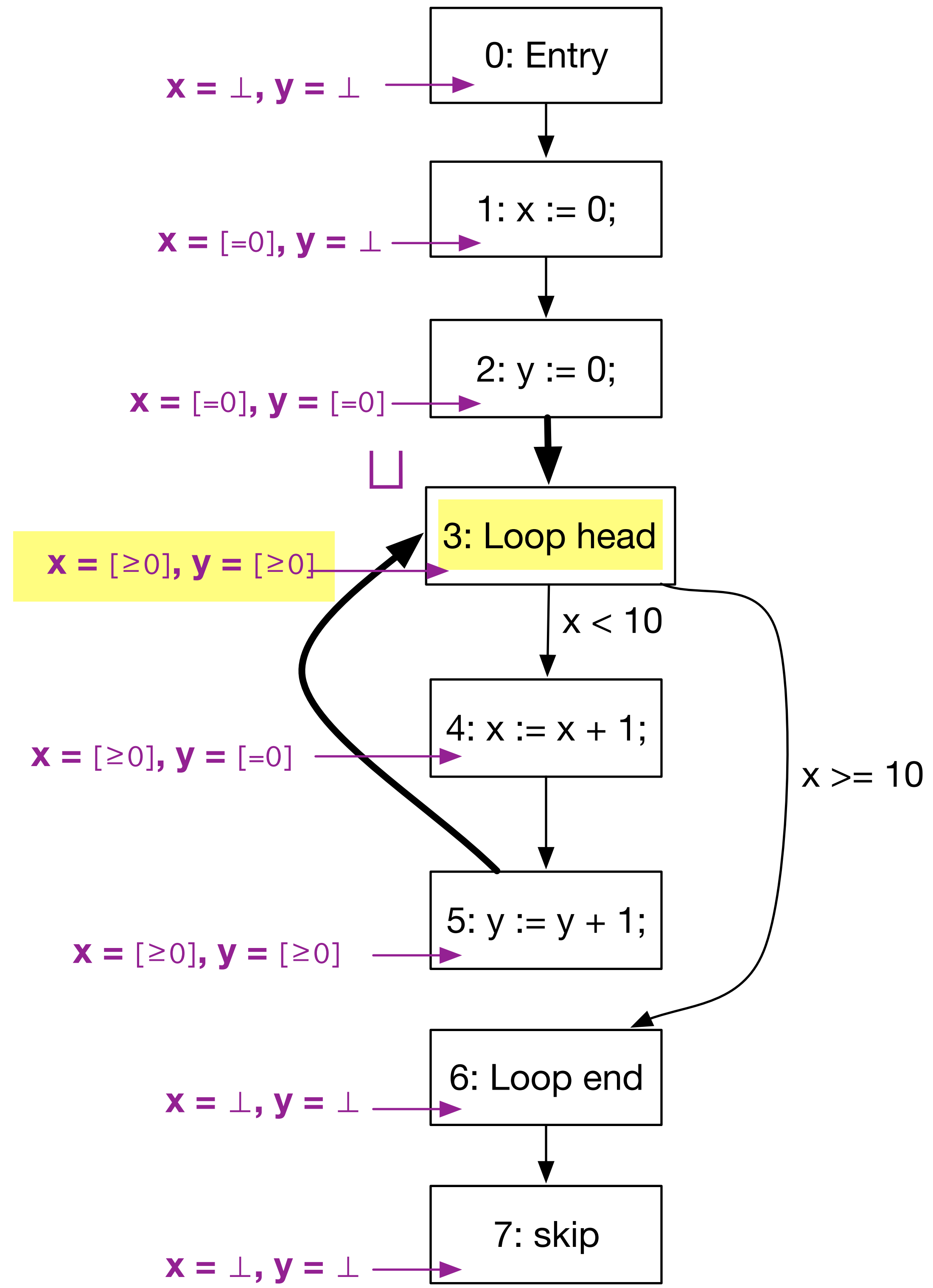
0: Entry
x = ⊥, y = ⊥
1: x := 0;
x = [=0], y = ⊥
2: y := 0;
x = [=0], y = [=0]
3: Loop head
x = ⊥, y = ⊥
x < 10
4: x := x + 1;
x = ⊥, y = ⊥
x >= 10
5: y := y + 1;
x = ⊥, y = ⊥
6: Loop end
x = ⊥, y = ⊥
7: skip
x = ⊥, y = ⊥

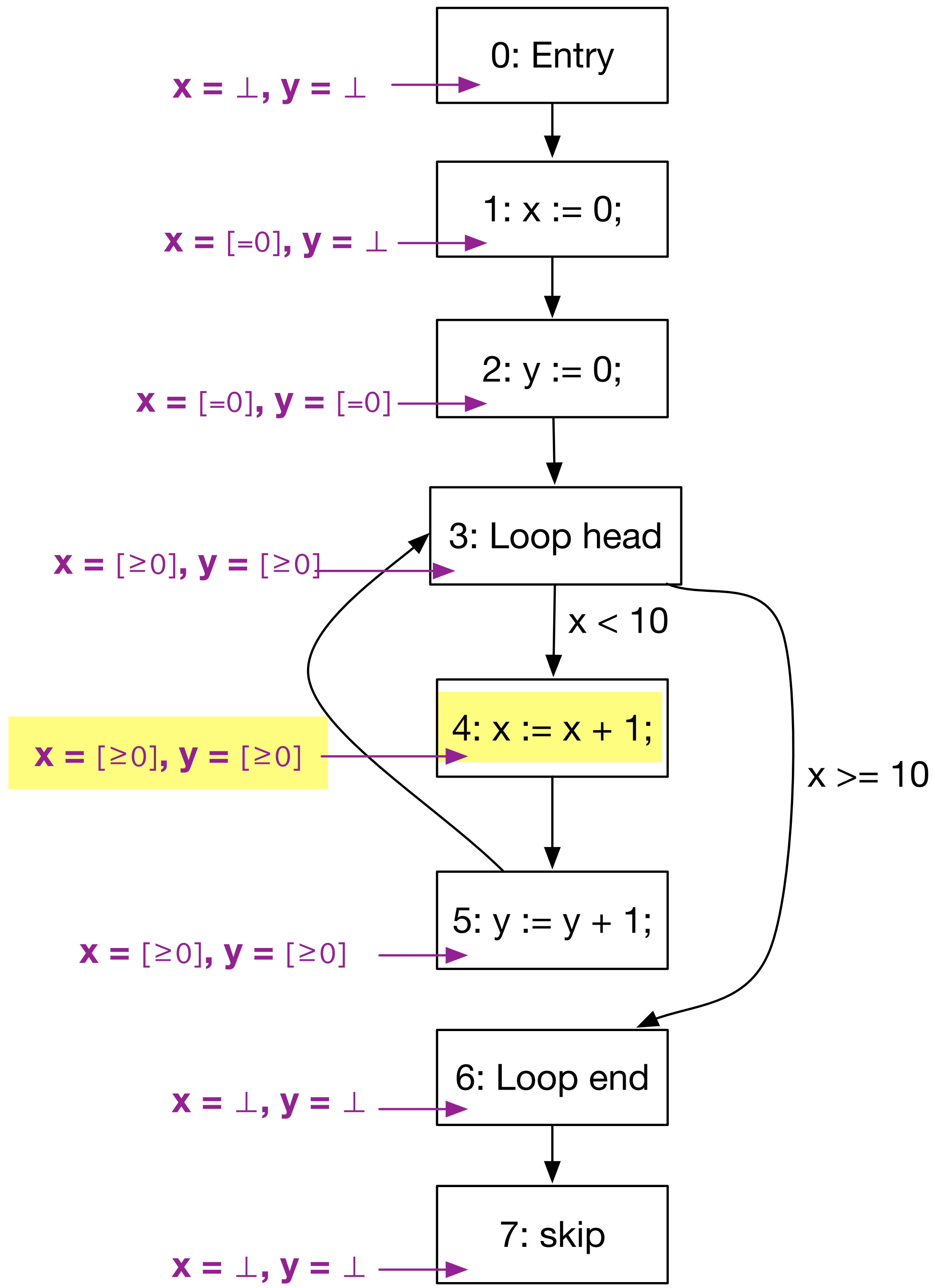


0: Entry
1: x := 0;
2: y := 0;
3: Loop head
4: x := x + 1;
5: y := y + 1;
6: Loop end
7: skip
x < 10
x >= 10
x = ⊥, y = ⊥
x = [=0], y = ⊥
x = [=0], y = [=0]
⊔
x = [=0], y = [=0]
x = ⊥, y = ⊥
x = ⊥, y = ⊥
x = ⊥, y = ⊥
x = ⊥, y = ⊥

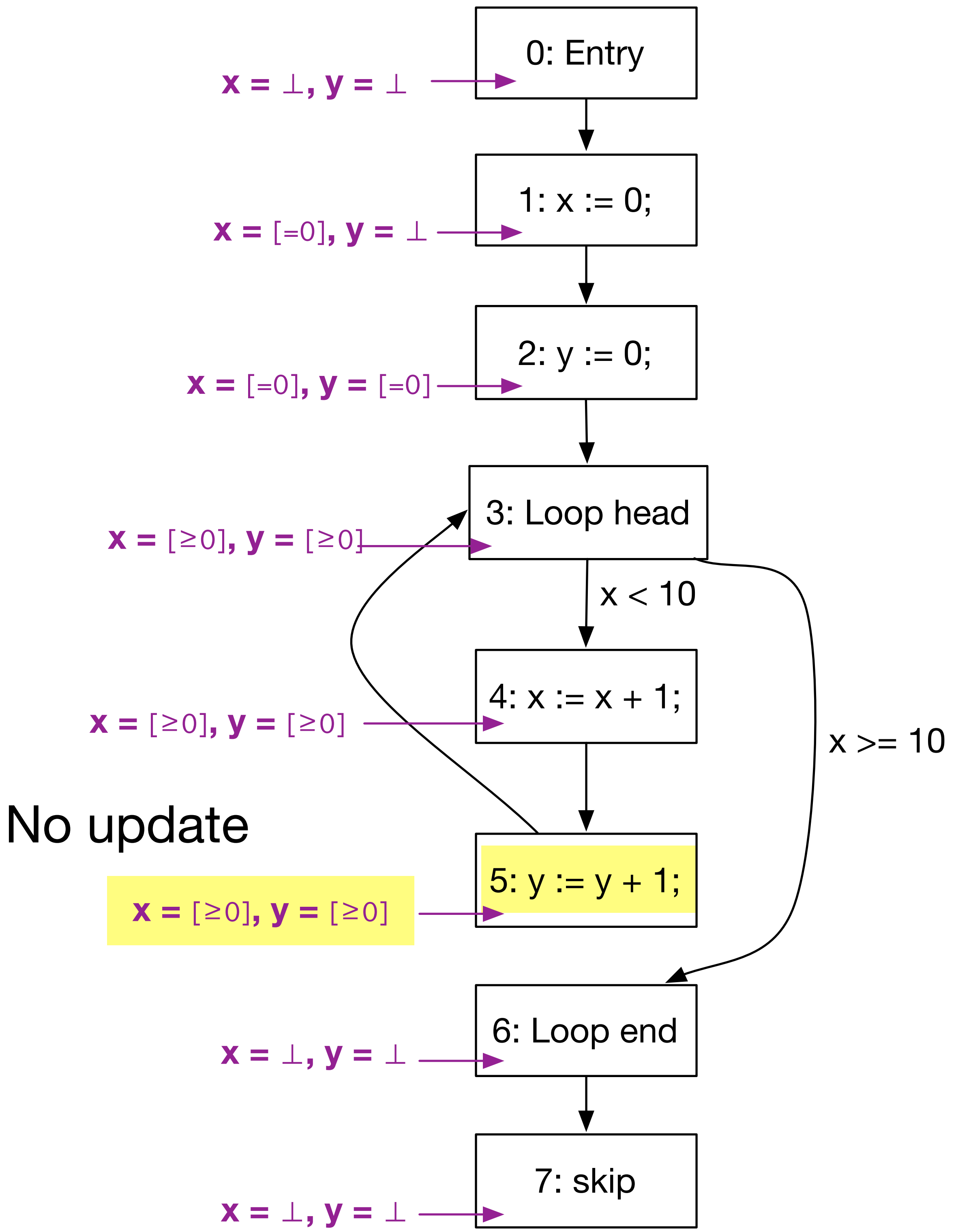

0: Entry
1: x := 0;
2: y := 0;
3: Loop head
4: x := x + 1;
5: y := y + 1;
6: Loop end
7: skip
x < 10
x >= 10
x = ⊥, y = ⊥
x = [=0], y = ⊥
x = [=0], y = [=0]
x = [=0], y = [=0]
x = [≥0], y = [=0]
x = ⊥, y = ⊥
x = ⊥, y = ⊥
x = ⊥, y = ⊥

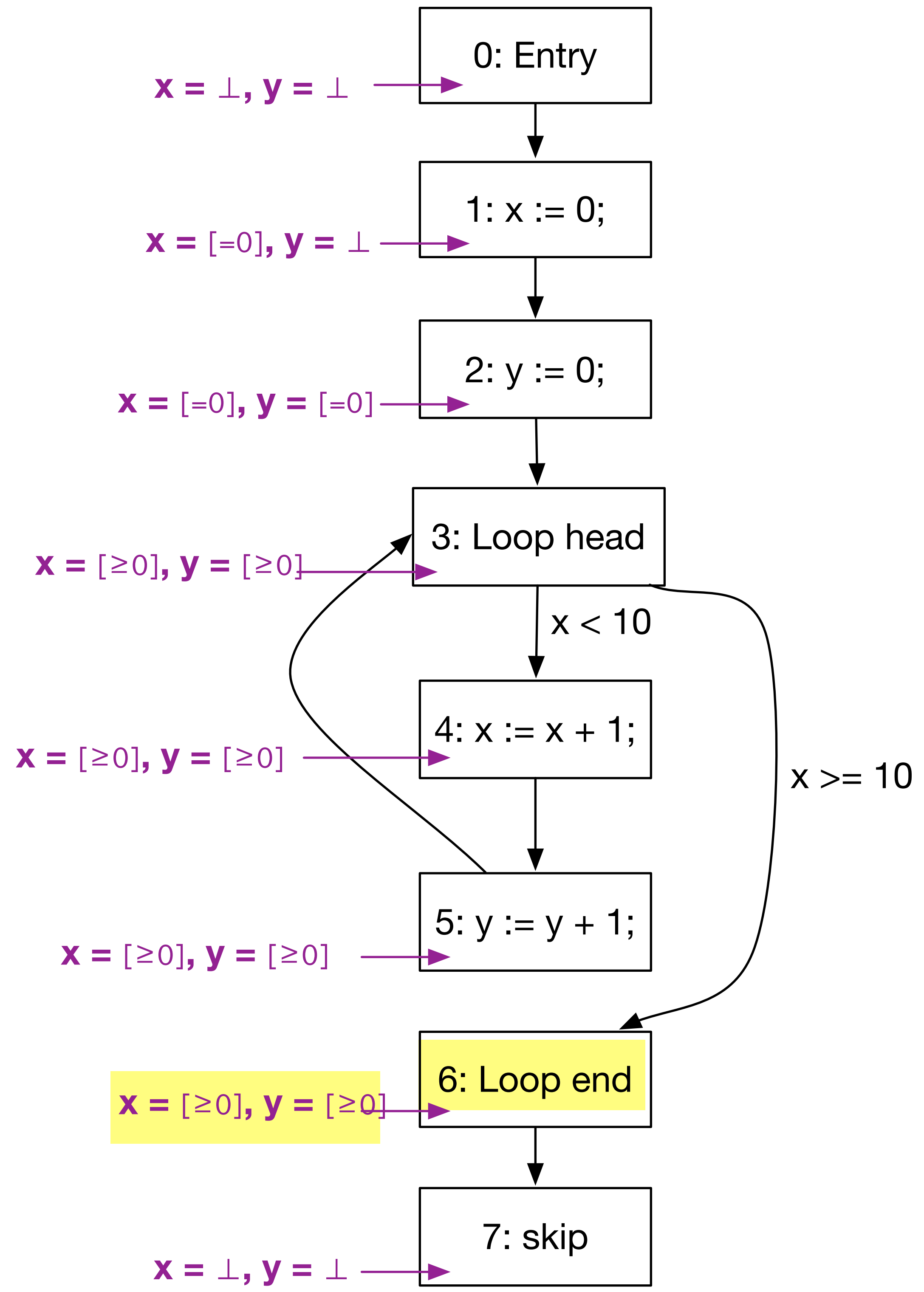

0: Entry
x = ⊥, y = ⊥
1: x := 0;
x = [=0], y = ⊥
2: y := 0;
x = [=0], y = [=0]
3: Loop head
x = [=0], y = [=0]
x < 10
4: x := x + 1;
x = [≥0], y = [=0]
x >= 10
5: y := y + 1;
x = [≥0], y = [≥0]
6: Loop end
x = ⊥, y = ⊥
7: skip
x = ⊥, y = ⊥


0: Entry
x = ⊥, y = ⊥
1: x := 0;
x = [=0], y = ⊥
2: y := 0;
x = [=0], y = [=0]
⊔
3: Loop head
x = [≥0], y = [≥0]
x < 10
4: x := x + 1;
x = [≥0], y = [=0]
x >= 10
5: y := y + 1;
x = [≥0], y = [≥0]
6: Loop end
x = ⊥, y = ⊥
7: skip
x = ⊥, y = ⊥

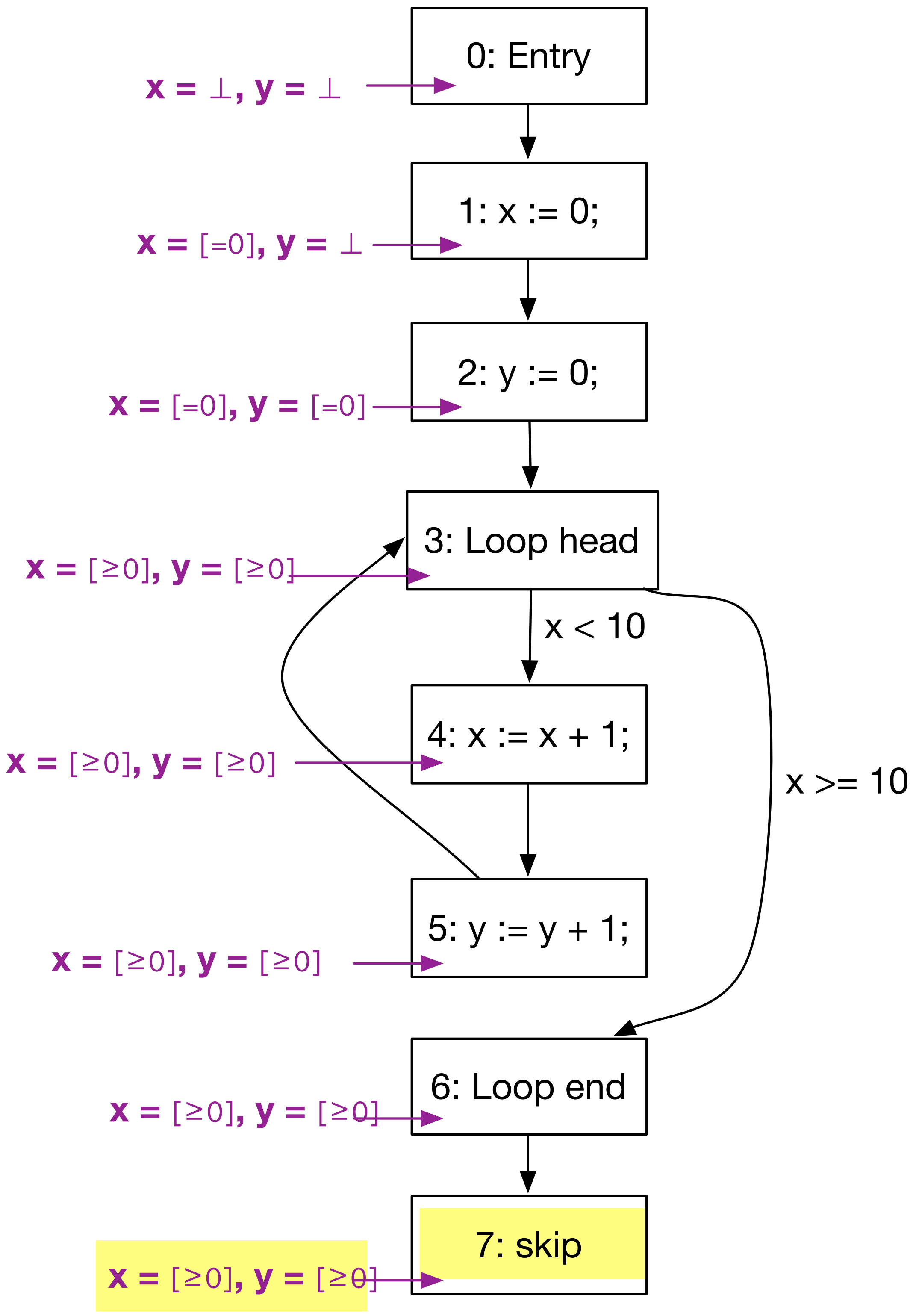


0: Entry
1: x := 0;
2: y := 0;
3: Loop head
4: x := x + 1;
5: y := y + 1;
6: Loop end
7: skip
x < 10
x >= 10
x = ⊥, y = ⊥
x = [=0], y = ⊥
x = [=0], y = [=0]
x = [≥0], y = [≥0]
x = [≥0], y = [≥0]
x = [≥0], y = [≥0]
x = ⊥, y = ⊥
x = ⊥, y = ⊥

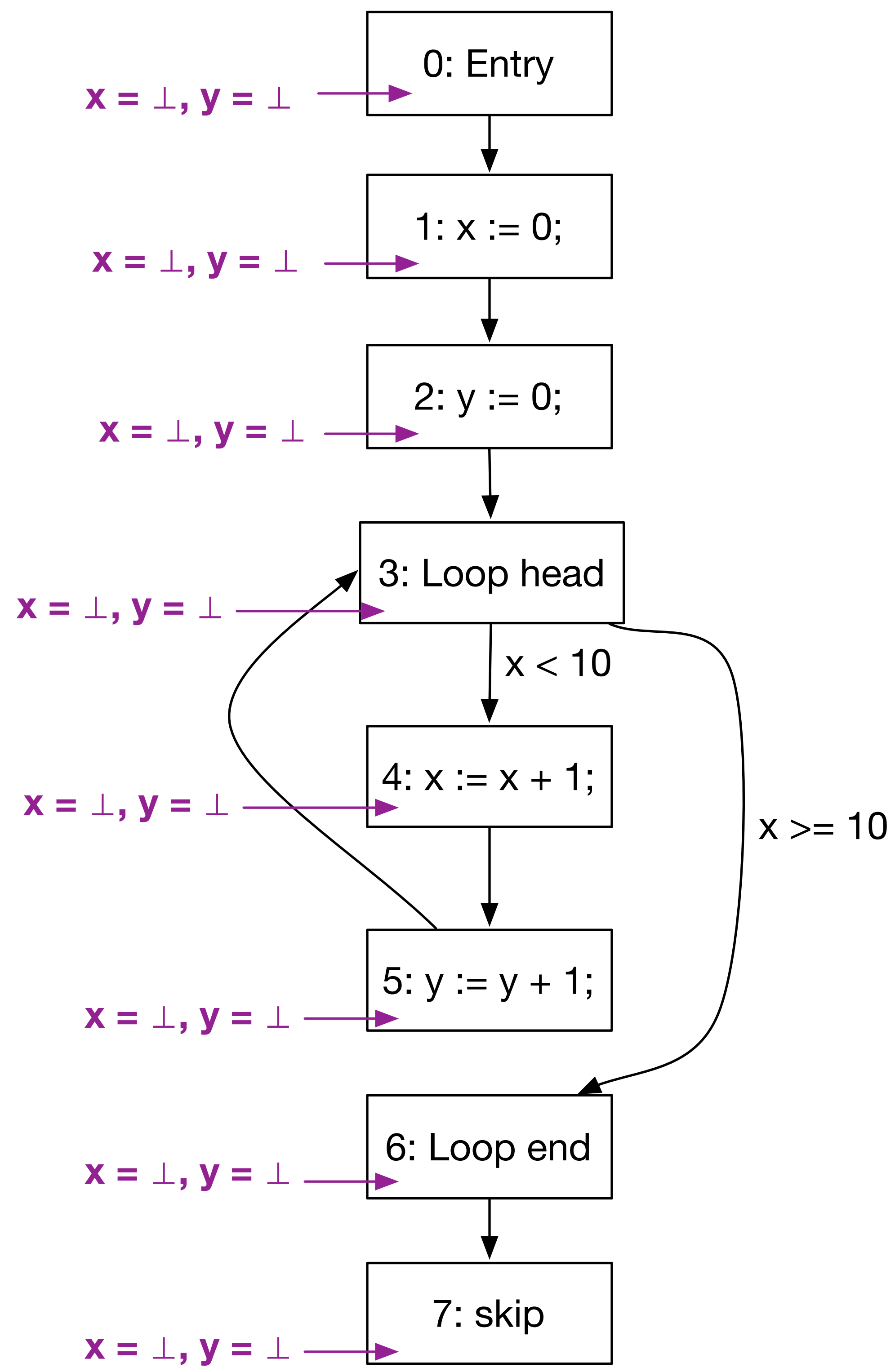

0: Entry
x = ⊥, y = ⊥
1: x := 0;
x = [=0], y = ⊥
2: y := 0;
x = [=0], y = [=0]
3: Loop head
x = [≥0], y = [≥0]
x < 10
4: x := x + 1;
x = [≥0], y = [≥0]
x >= 10
No update
5: y := y + 1;
x = [≥0], y = [≥0]
6: Loop end
x = ⊥, y = ⊥
7: skip
x = ⊥, y = ⊥

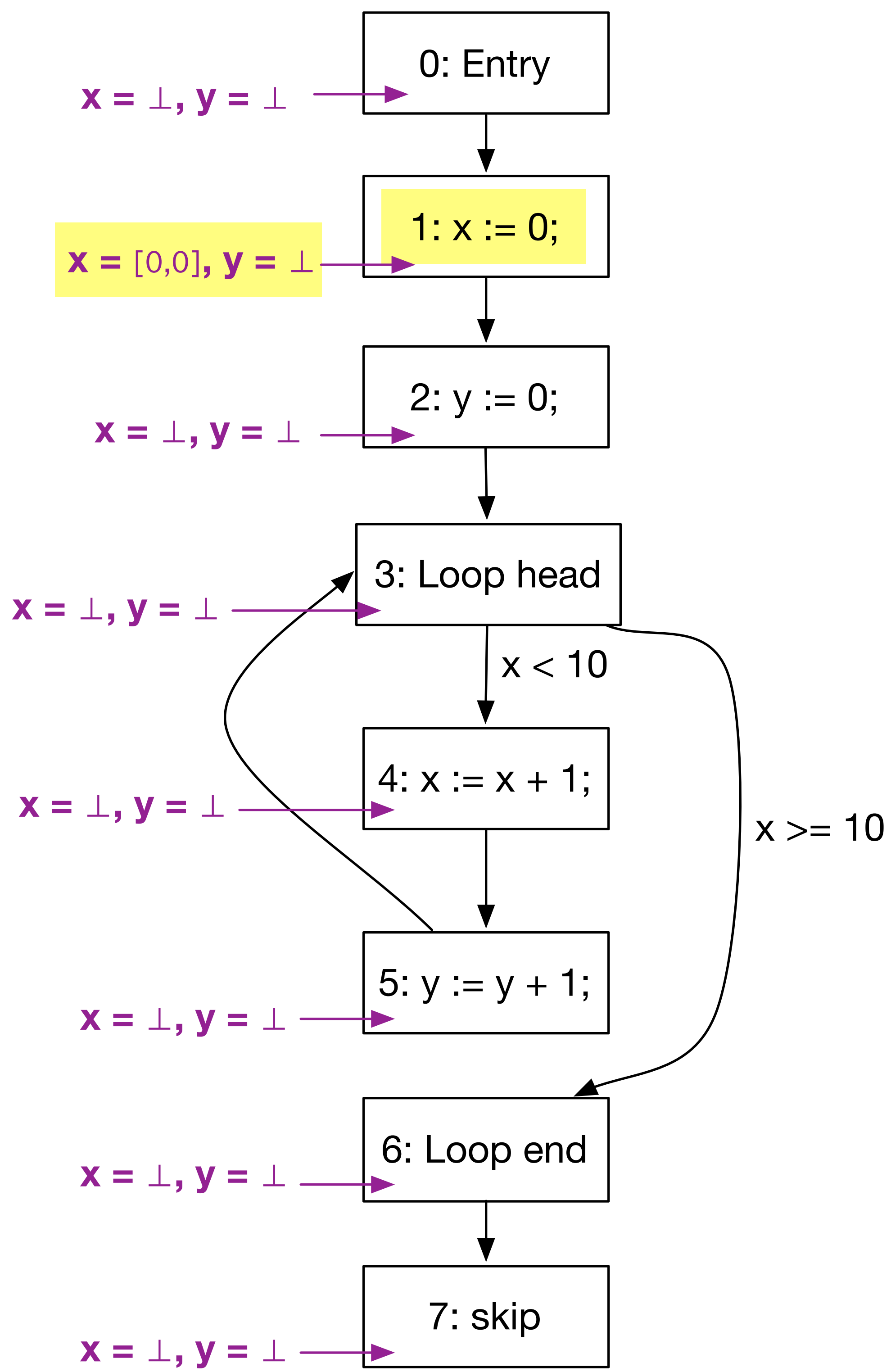


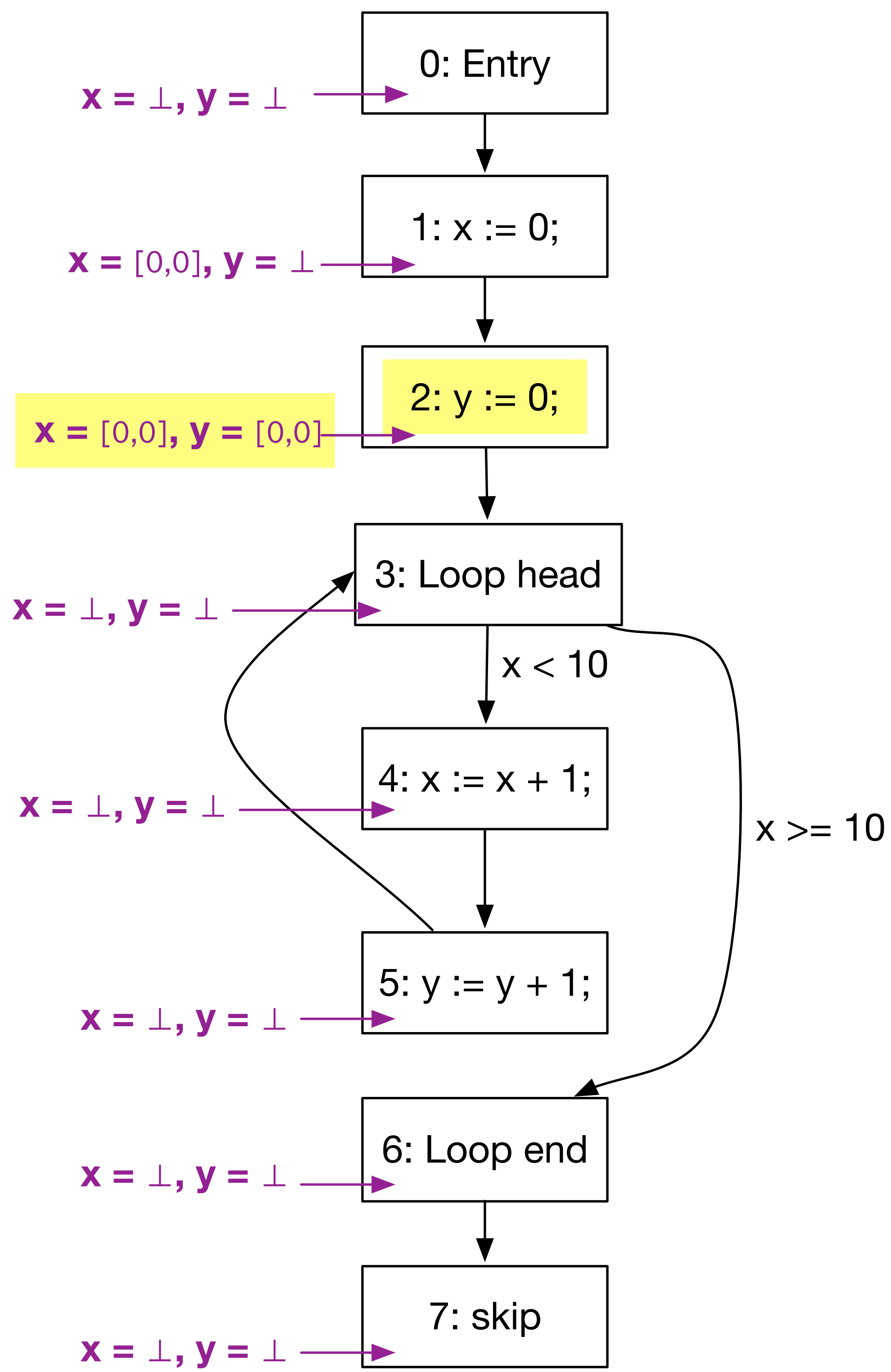
0: Entry
1: x := 0;
2: y := 0;
3: Loop head
4: x := x + 1;
5: y := y + 1;
6: Loop end
7: skip
x < 10
x >= 10
x = ⊥, y = ⊥
x = [=0], y = ⊥
x = [=0], y = [=0]
x = [≥0], y = [≥0]
x = [≥0], y = [≥0]
x = [≥0], y = [≥0]
x = [≥0], y = [≥0]
x = ⊥, y = ⊥

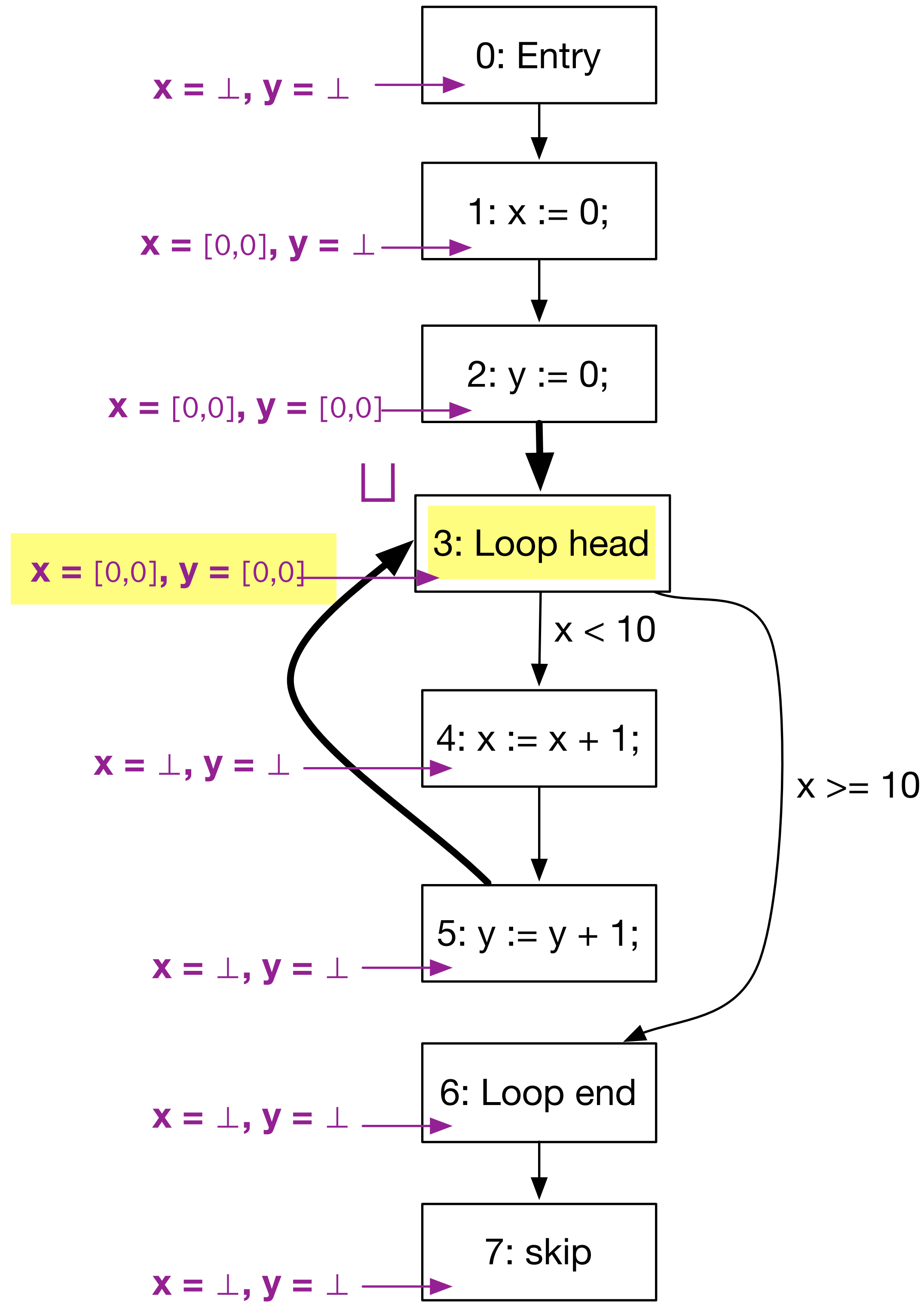

0: Entry
x = ⊥, y = ⊥
1: x := 0;
x = [=0], y = ⊥
2: y := 0;
x = [=0], y = [=0]
3: Loop head
x = [≥0], y = [≥0]
x < 10
4: x := x + 1;
x = [≥0], y = [≥0]
x >= 10
5: y := y + 1;
x = [≥0], y = [≥0]
6: Loop end
x = [≥0], y = [≥0]
7: skip
x = [≥0], y = [≥0]

Interval Analysis

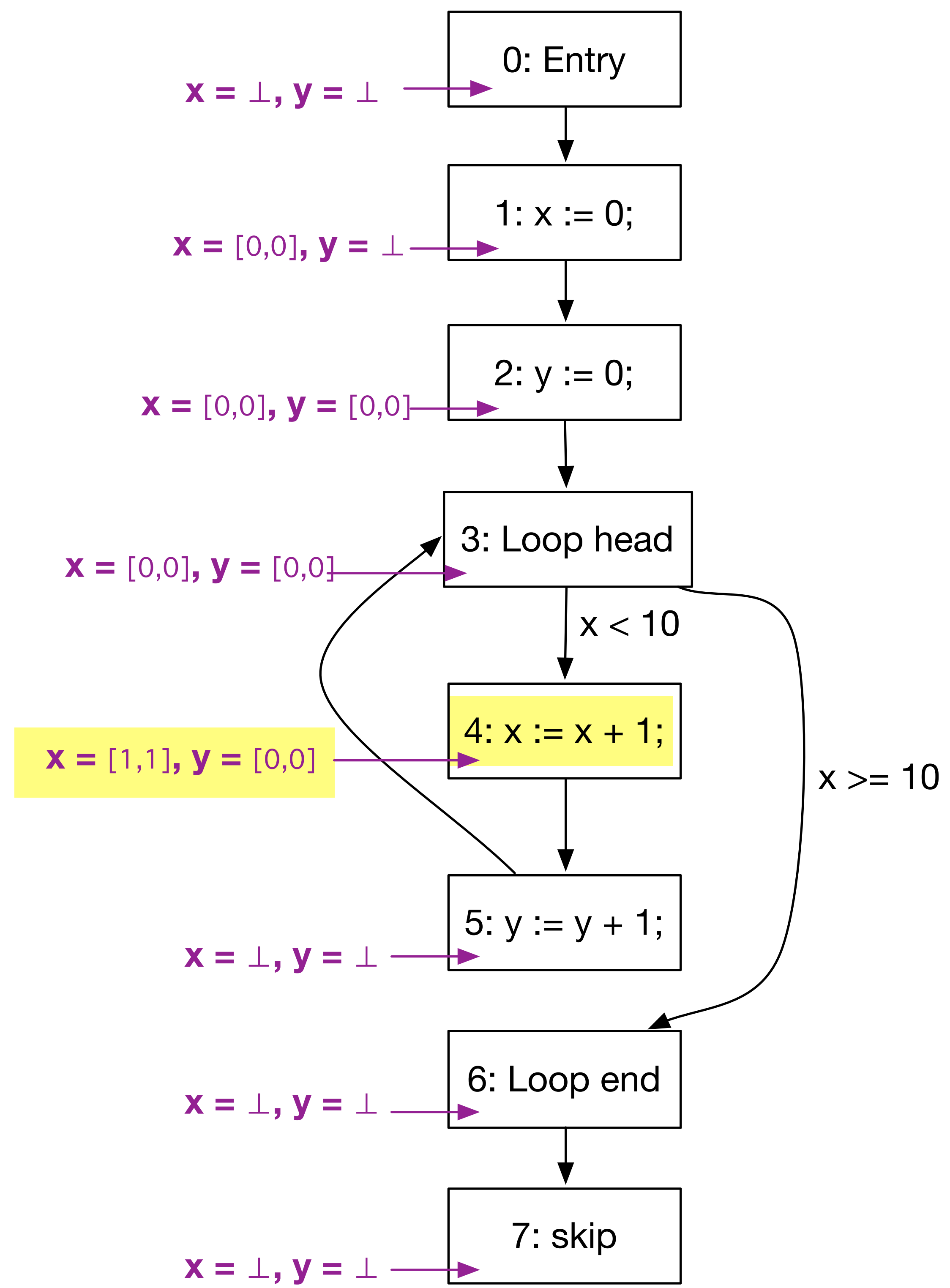


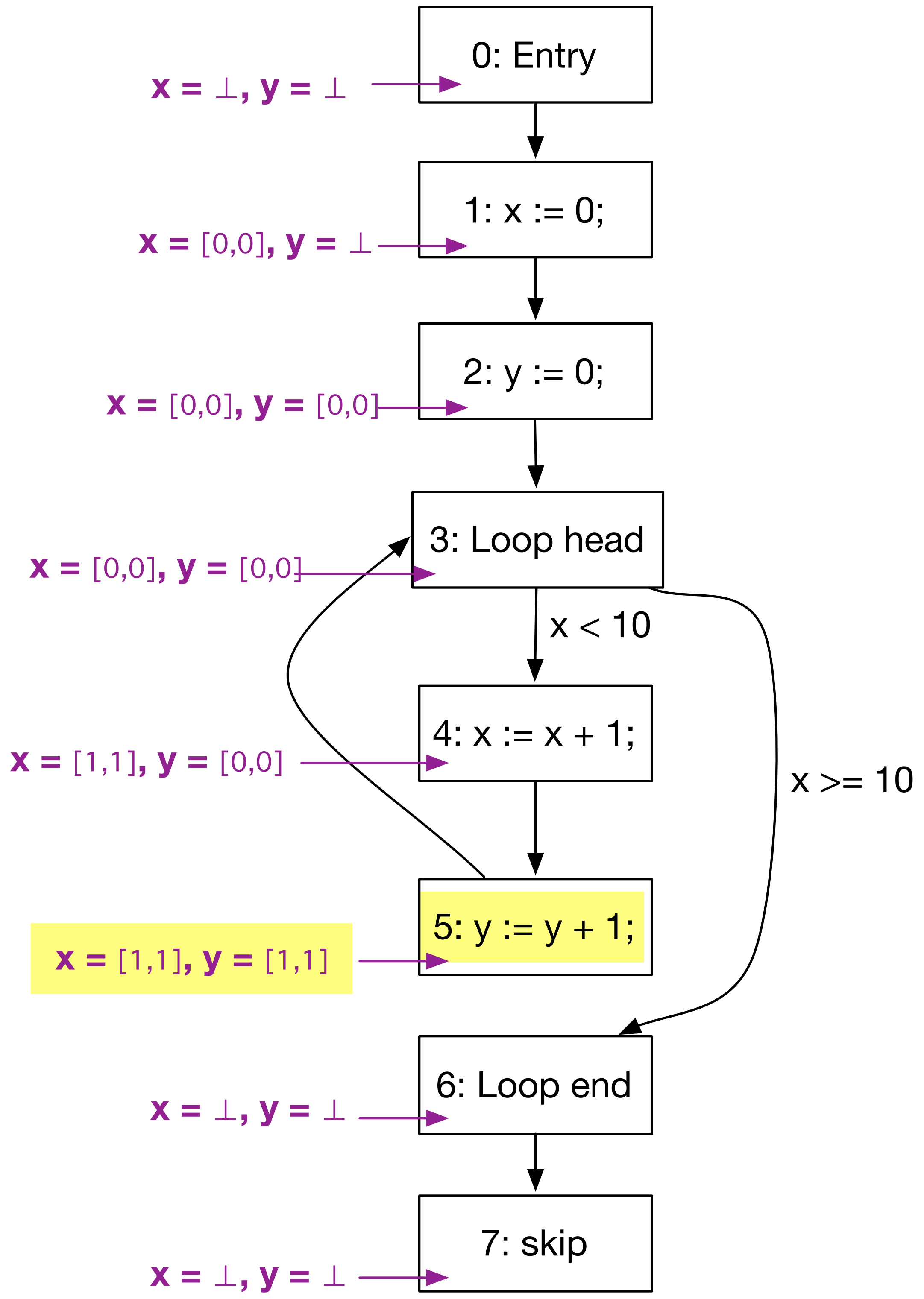

0: Entry
1: x := 0;
2: y := 0;
3: Loop head
4: x := x + 1;
5: y := y + 1;
6: Loop end
7: skip
x < 10
x >= 10
x = ⊥, y = ⊥
x = [0,0], y = ⊥
x = ⊥, y = ⊥
x = ⊥, y = ⊥
x = ⊥, y = ⊥
x = ⊥, y = ⊥
x = ⊥, y = ⊥
x = ⊥, y = ⊥


0: Entry
x = ⊥, y = ⊥
1: x := 0;
x = [0,0], y = ⊥
2: y := 0;
x = [0,0], y = [0,0]
3: Loop head
x = ⊥, y = ⊥
x < 10
4: x := x + 1;
x = ⊥, y = ⊥
x >= 10
5: y := y + 1;
x = ⊥, y = ⊥
6: Loop end
x = ⊥, y = ⊥
7: skip
x = ⊥, y = ⊥

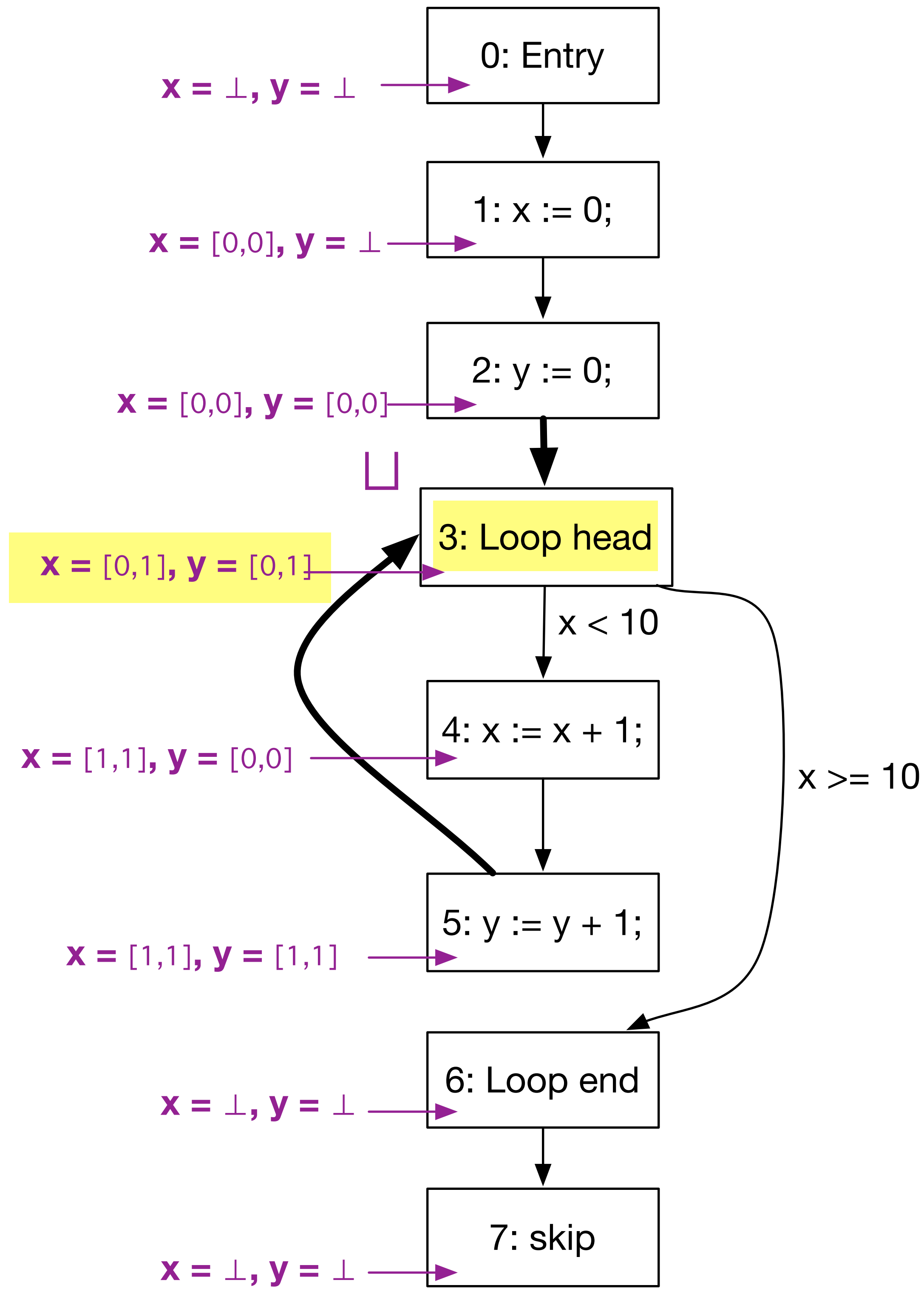


0: Entry
x = ⊥, y = ⊥
1: x := 0;
x = [0,0], y = ⊥
2: y := 0;
x = [0,0], y = [0,0]
⊔
3: Loop head
x = [0,0], y = [0,0]
x < 10
4: x := x + 1;
x = ⊥, y = ⊥
x >= 10
5: y := y + 1;
x = ⊥, y = ⊥
6: Loop end
x = ⊥, y = ⊥
7: skip
x = ⊥, y = ⊥

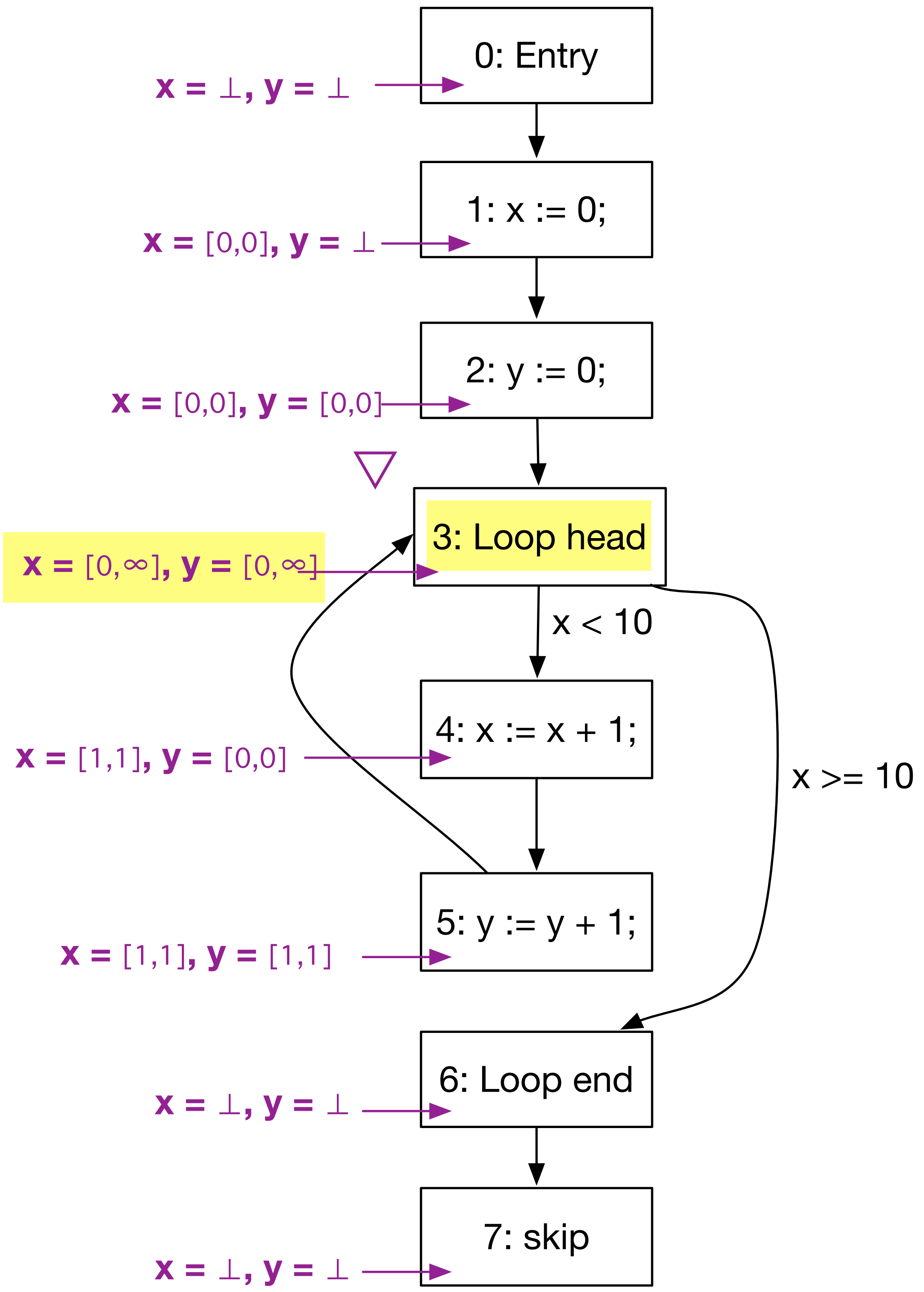

0: Entry
x = ⊥, y = ⊥
1: x := 0;
x = [0,0], y = ⊥
2: y := 0;
x = [0,0], y = [0,0]
3: Loop head
x = [0,0], y = [0,0]
x < 10
4: x := x + 1;
x = [1,1], y = [0,0]
x >= 10
5: y := y + 1;
x = ⊥, y = ⊥
6: Loop end
x = ⊥, y = ⊥
7: skip
x = ⊥, y = ⊥



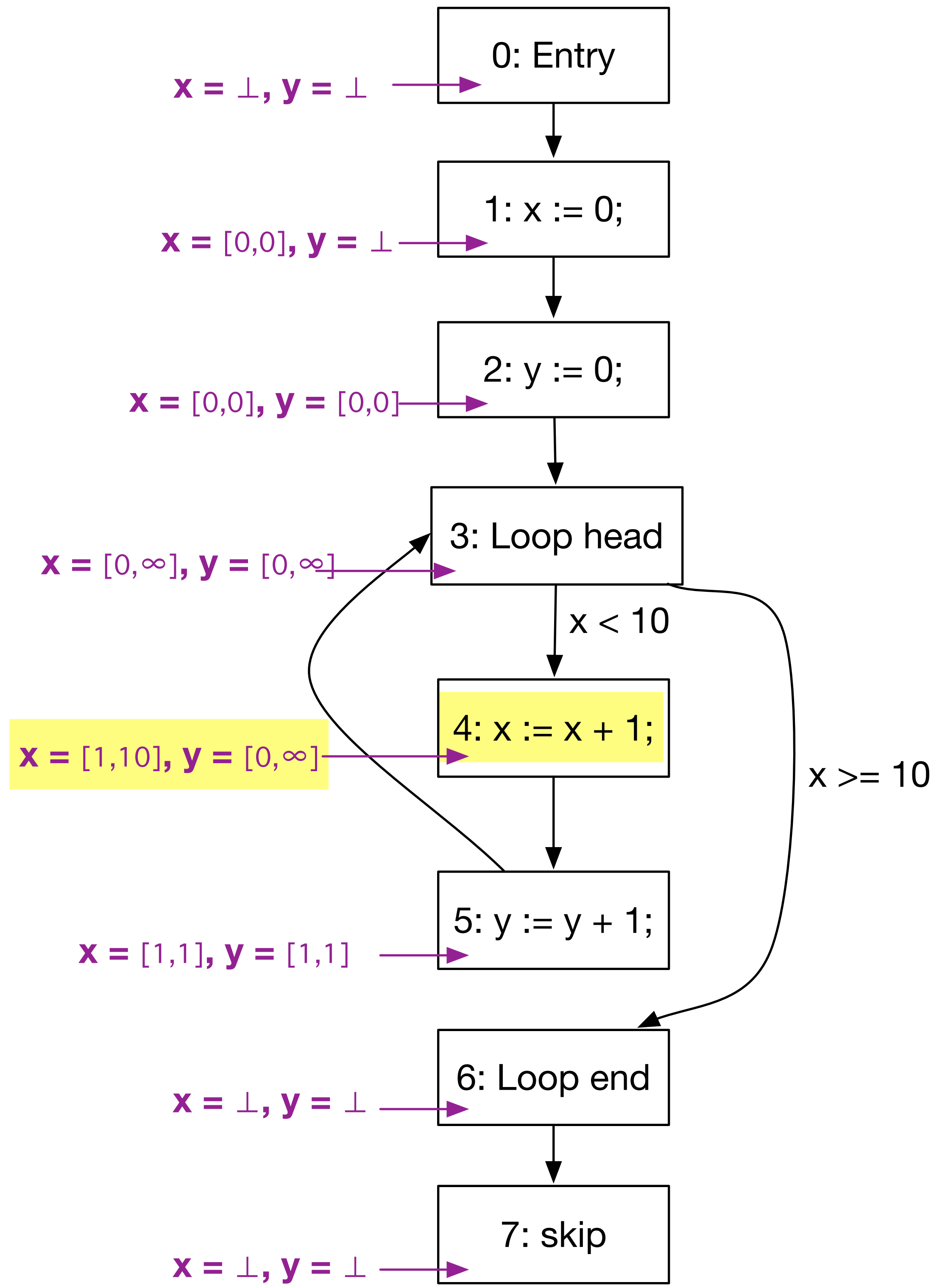
0: Entry
1: x := 0;
2: y := 0;
3: Loop head
4: x := x + 1;
5: y := y + 1;
6: Loop end
7: skip
x < 10
x >= 10
x = ⊥, y = ⊥
x = [0,0], y = ⊥
x = [0,0], y = [0,0]
x = [0,0], y = [0,0]
x = [1,1], y = [0,0]
x = [1,1], y = [1,1]
x = ⊥, y = ⊥
x = ⊥, y = ⊥

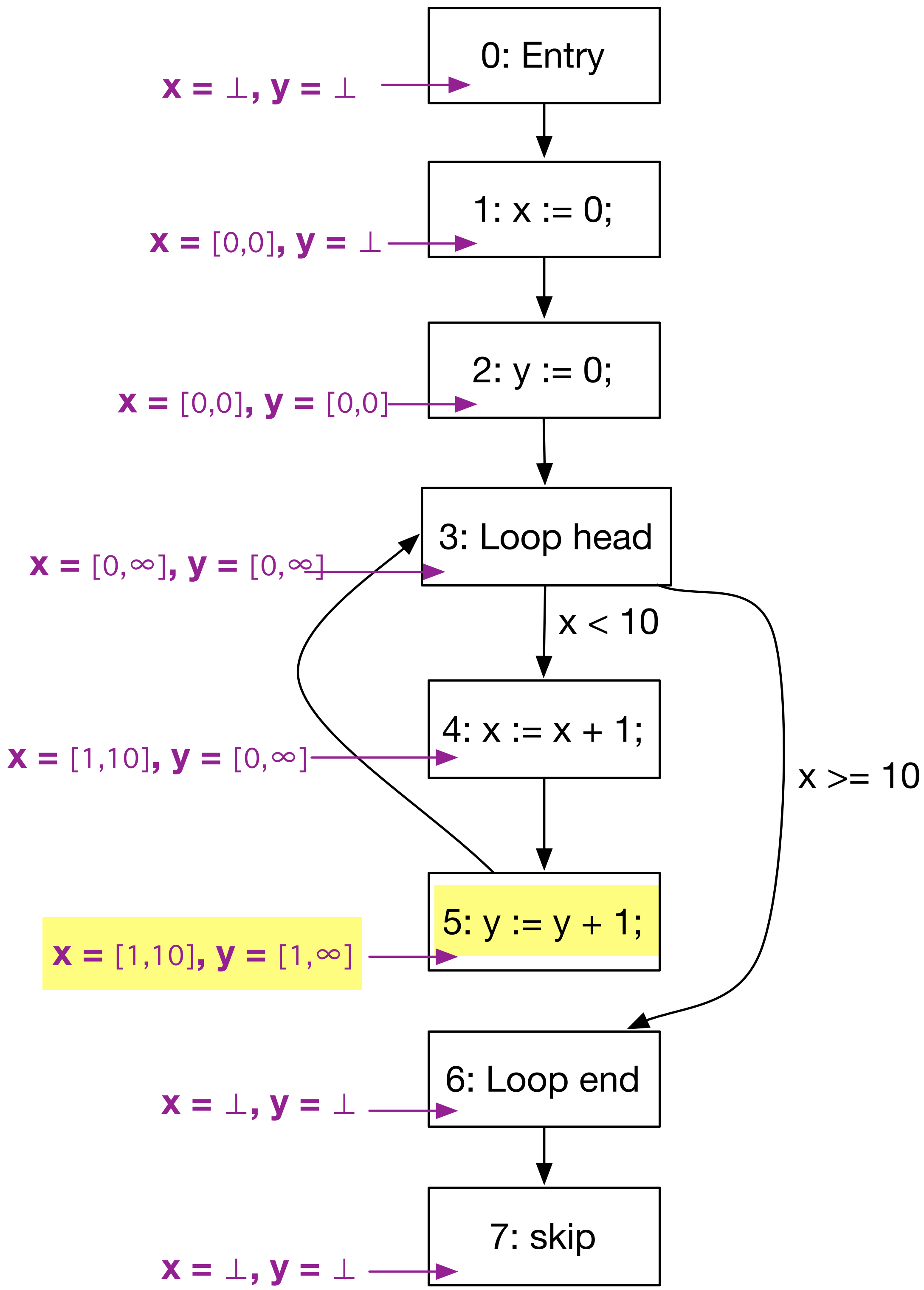


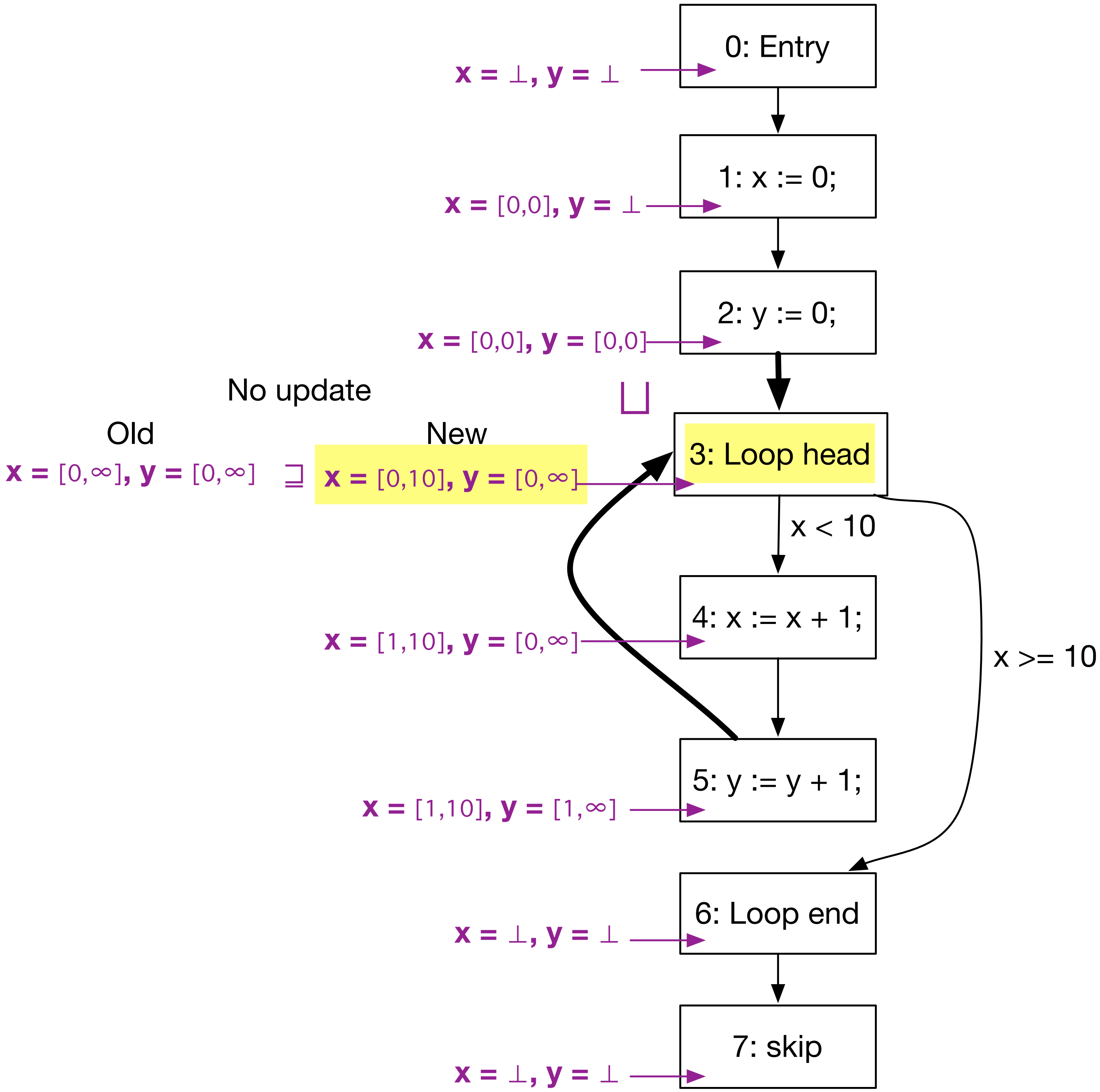
0: Entry
1: x := 0;
2: y := 0;
3: Loop head
4: x := x + 1;
5: y := y + 1;
6: Loop end
7: skip
x < 10
x >= 10
x = ⊥, y = ⊥
x = [0,0], y = ⊥
x = [0,0], y = [0,0]
⊔
x = [0,1], y = [0,1]
x = [1,1], y = [0,0]
x = [1,1], y = [1,1]
x = ⊥, y = ⊥
x = ⊥, y = ⊥

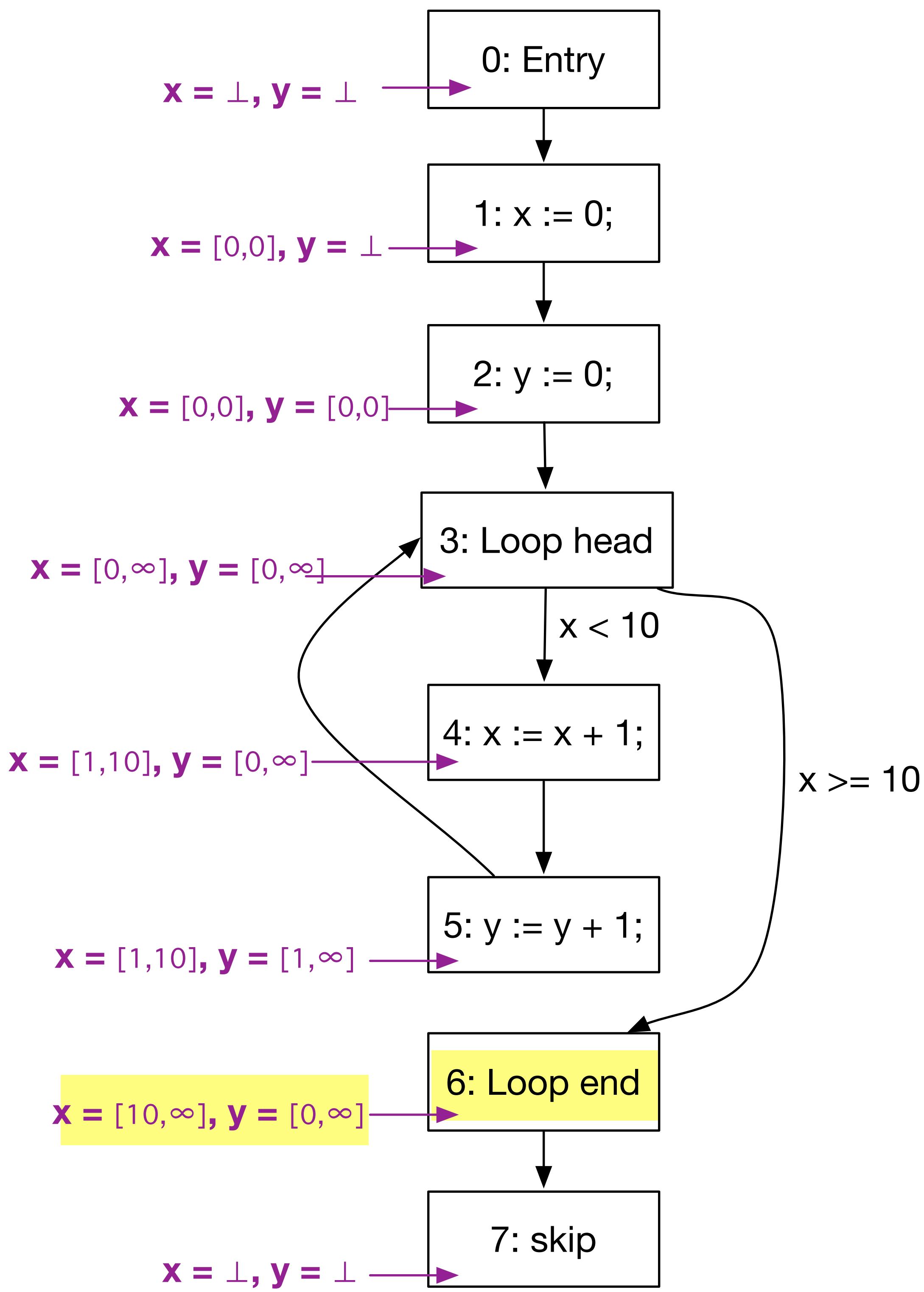


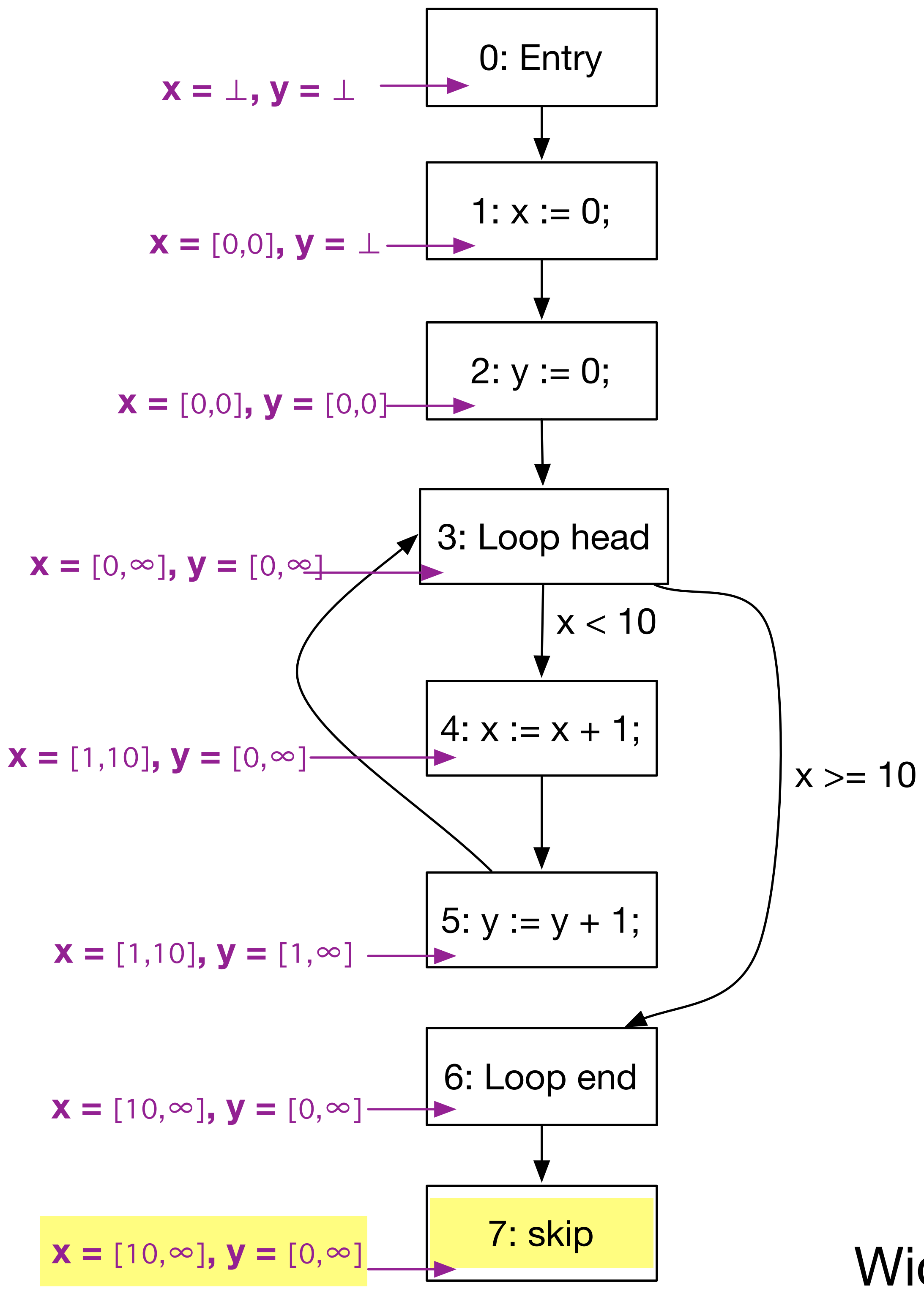
Apply widening


0: Entry
x = ⊥, y = ⊥
1: x := 0;
x = [0,0], y = ⊥
2: y := 0;
x = [0,0], y = [0,0]
3: Loop head
x = [0,∞], y = [0,∞]
x < 10
4: x := x + 1;
x = [1,10], y = [0,∞]
x >= 10
5: y := y + 1;
x = [1,1], y = [1,1]
6: Loop end
x = ⊥, y = ⊥
7: skip
x = ⊥, y = ⊥


0: Entry
x = ⊥, y = ⊥
1: x := 0;
x = [0,0], y = ⊥
2: y := 0;
x = [0,0], y = [0,0]
3: Loop head
x = [0,∞], y = [0,∞]
x < 10
4: x := x + 1;
x = [1,10], y = [0,∞]
x >= 10
5: y := y + 1;
x = [1,10], y = [1,∞]
6: Loop end
x = ⊥, y = ⊥
7: skip
x = ⊥, y = ⊥

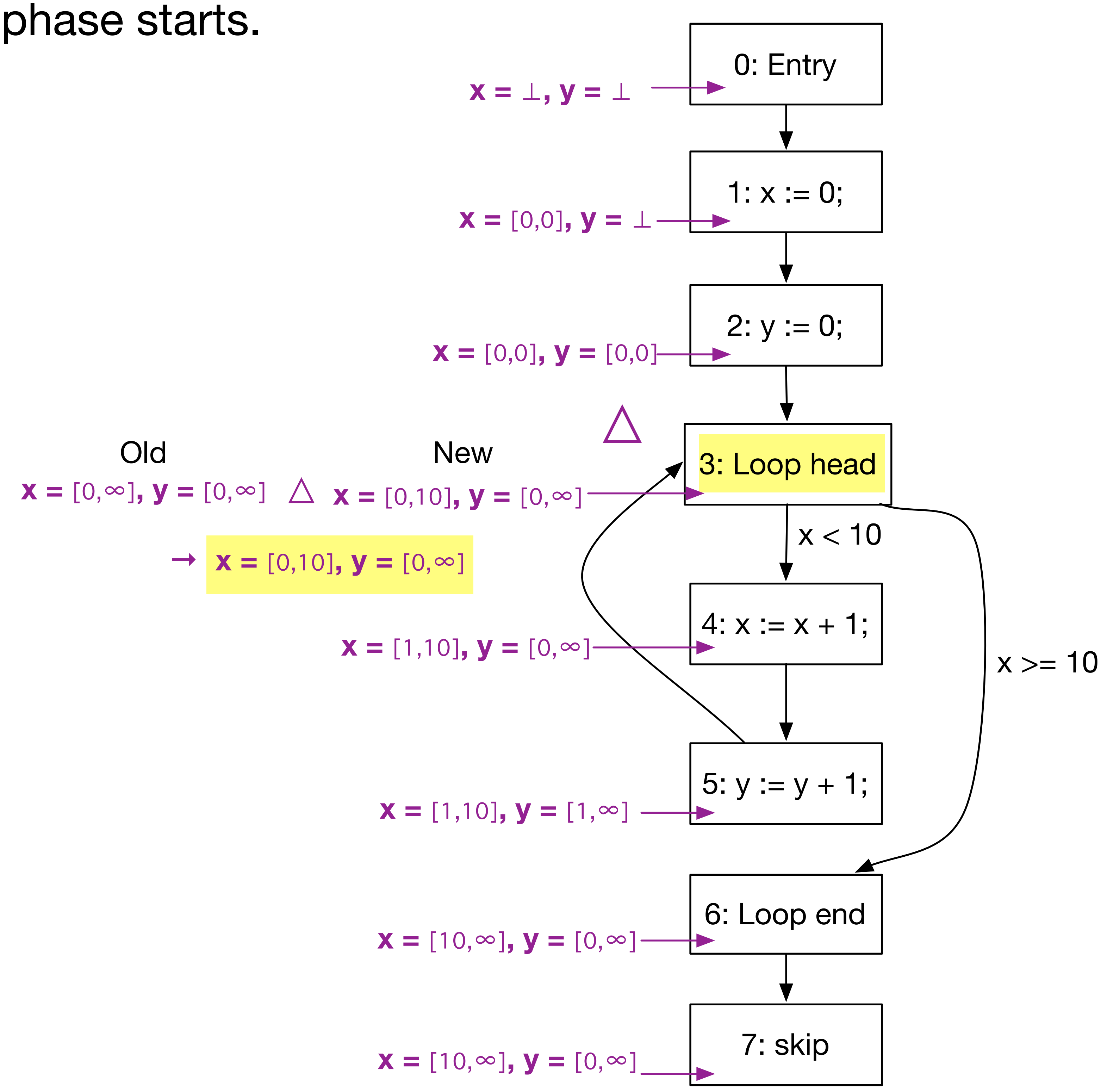

0: Entry
x = ⊥, y = ⊥
1: x := 0;
x = [0,0], y = ⊥
2: y := 0;
x = [0,0], y = [0,0]
No update
⊔
Old
New
x = [0,∞], y = [0,∞]
⊒
x = [0,10], y = [0,∞]
3: Loop head
x < 10
4: x := x + 1;
x = [1,10], y = [0,∞]
x >= 10
5: y := y + 1;
x = [1,10], y = [1,∞]
6: Loop end
x = ⊥, y = ⊥
7: skip
x = ⊥, y = ⊥

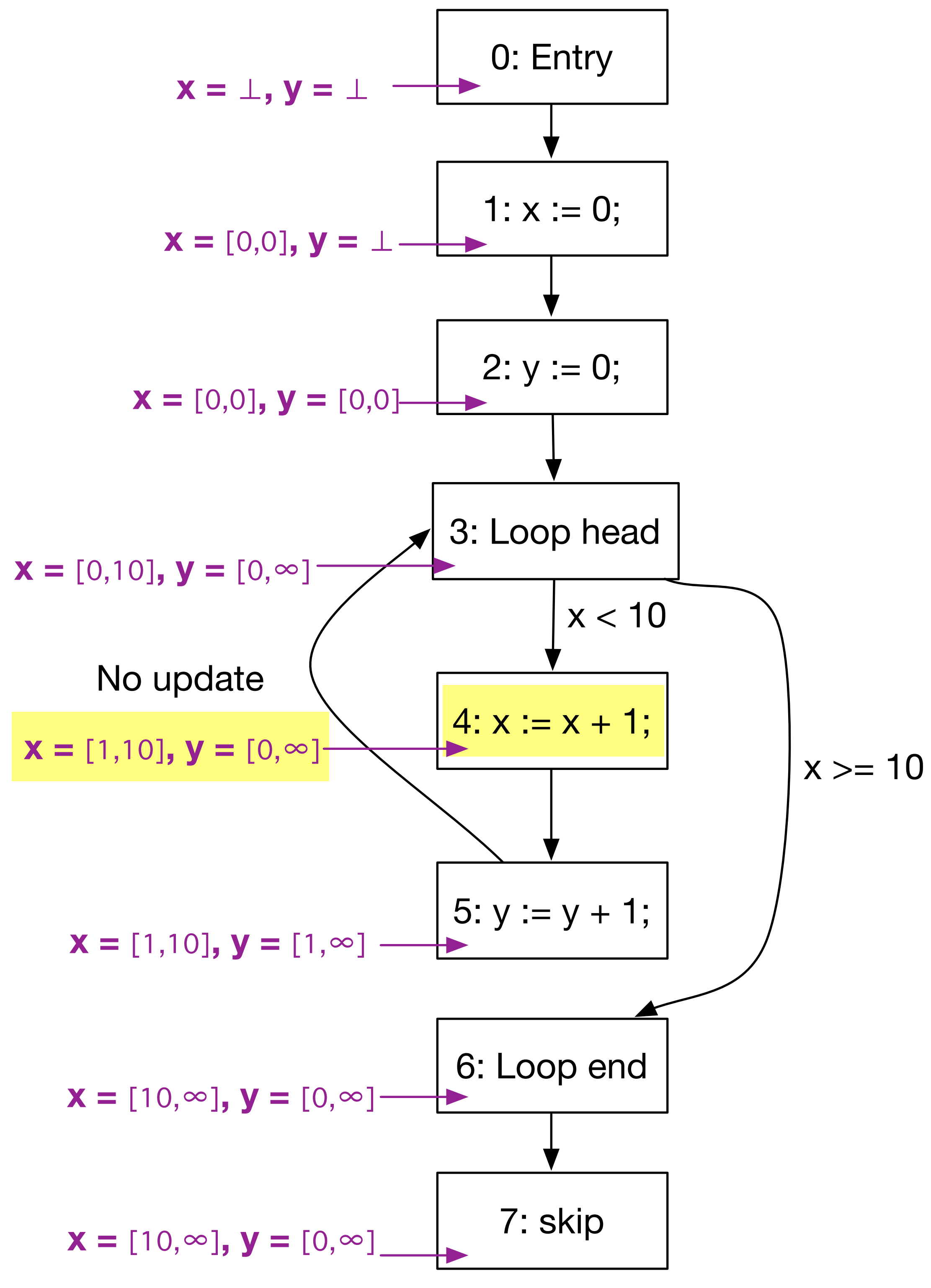

0: Entry
1: x := 0;
2: y := 0;
3: Loop head
4: x := x + 1;
5: y := y + 1;
6: Loop end
7: skip
x < 10
x >= 10
x = ⊥, y = ⊥
x = [0,0], y = ⊥
x = [0,0], y = [0,0]
x = [0,∞], y = [0,∞]
x = [1,10], y = [0,∞]
x = [1,10], y = [1,∞]
x = [10,∞], y = [0,∞]
x = ⊥, y = ⊥

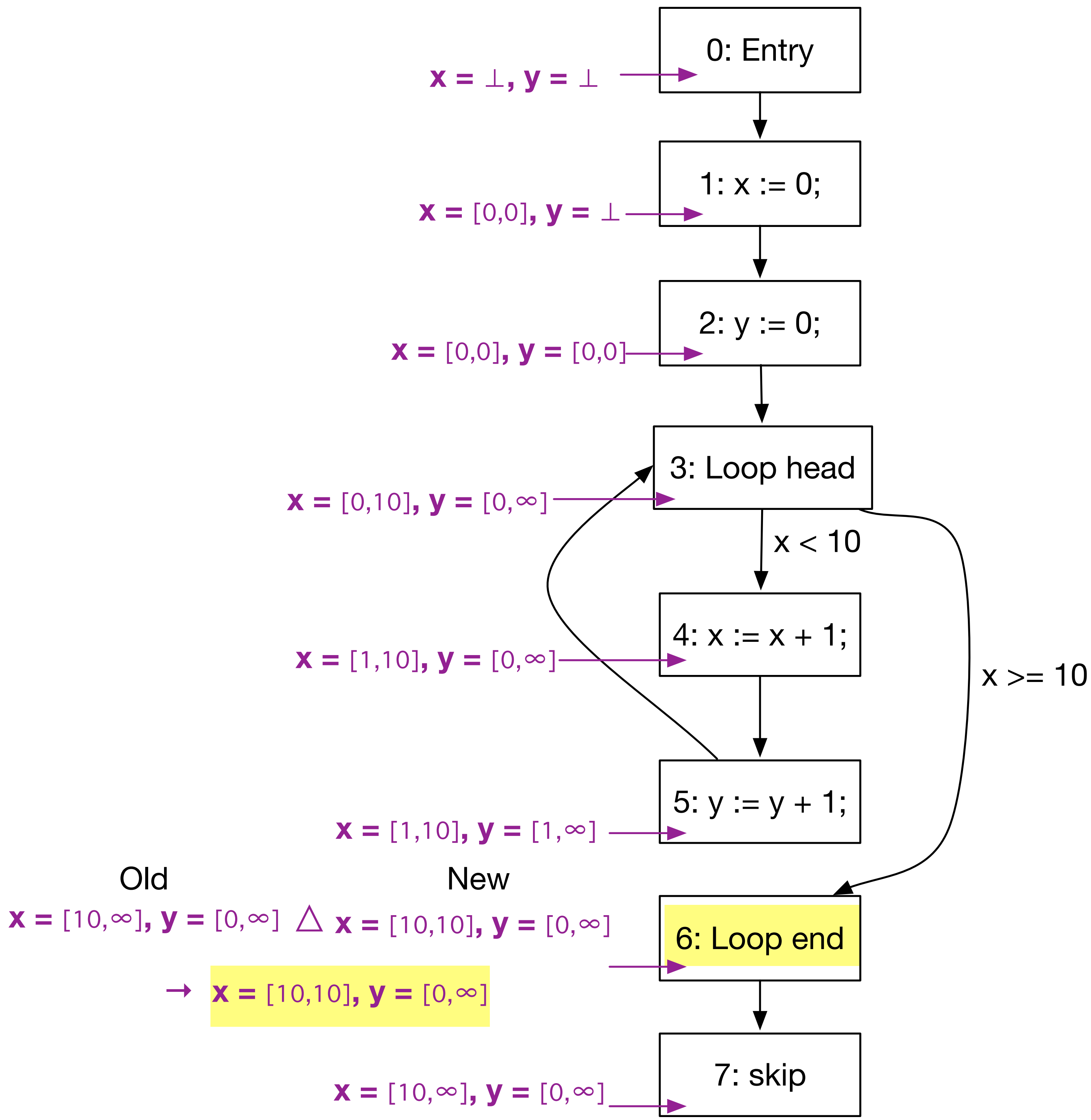


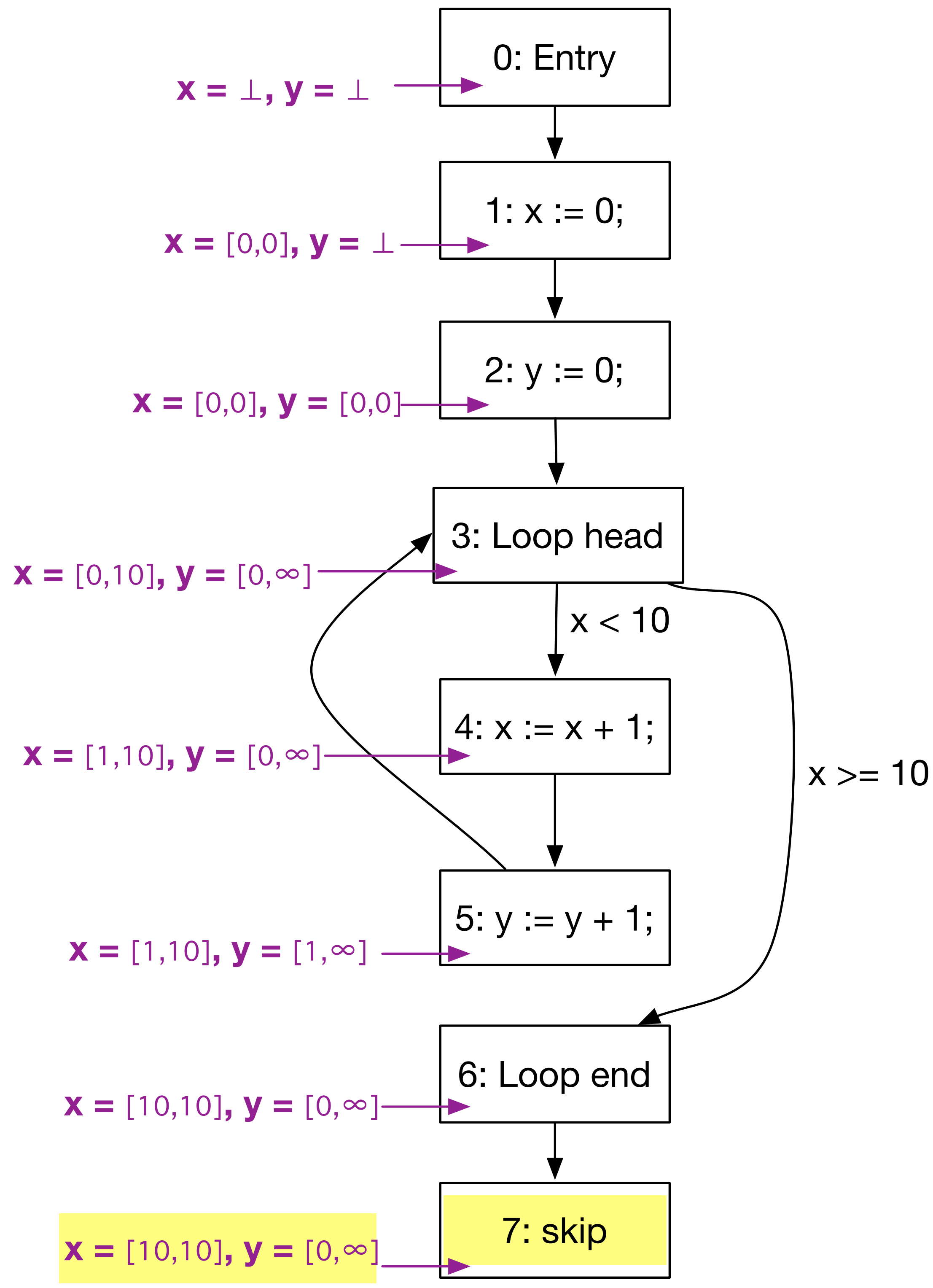
Widening phase done.

Narrowing phase starts.




0: Entry
x = ⊥, y = ⊥
1: x := 0;
x = [0,0], y = ⊥
2: y := 0;
x = [0,0], y = [0,0]
3: Loop head
x = [0,10], y = [0,∞]
x < 10
No update
4: x := x + 1;
x = [1,10], y = [0,∞]
x >= 10
5: y := y + 1;
x = [1,10], y = [1,∞]
6: Loop end
x = [10,∞], y = [0,∞]
7: skip
x = [10,∞], y = [0,∞]


0: Entry
x = ⊥, y = ⊥
1: x := 0;
x = [0,0], y = ⊥
2: y := 0;
x = [0,0], y = [0,0]
3: Loop head
x = [0,10], y = [0,∞]
x < 10
4: x := x + 1;
x = [1,10], y = [0,∞]
x >= 10
5: y := y + 1;
x = [1,10], y = [1,∞]
Old
New
x = [10,∞], y = [0,∞] △ x = [10,10], y = [0,∞]
6: Loop end
→ x = [10,10], y = [0,∞]
7: skip
x = [10,∞], y = [0,∞]


0: Entry
1: x := 0;
2: y := 0;
3: Loop head
4: x := x + 1;
5: y := y + 1;
6: Loop end
7: skip
x < 10
x >= 10
x = ⊥, y = ⊥
x = [0,0], y = ⊥
x = [0,0], y = [0,0]
x = [0,10], y = [0,∞]
x = [1,10], y = [0,∞]
x = [1,10], y = [1,∞]
x = [10,10], y = [0,∞]
x = [10,10], y = [0,∞]

Language

x	$\in$	$\mathbb{X}$	program variables
C	::=		statements
	|	**skip**	nop statement
	|	C ; C	sequence of statements
	|	x := E	assignment
	|	**input** x	read an integer input
	|	**if** B C C	condition statement
	|	**while** B C	loop statement
	|	**goto** E	goto with dynamic label
E	::=		expression
	|	n	integer
	|	x	variable
	|	$E + E$	addition
B	::=		boolean expression
	|	**true** | **false**	
	|	$E < E$	comparison
	|	$E = E$	equality
P	::=	C	program

We assume each statement of the program is uniquely *labeled*.

Transitional Semantics

State transition sequence

$$s_0 \hookrightarrow s_1 \hookrightarrow s_2 \hookrightarrow \cdots$$

where $\hookrightarrow$ is a transition relation between states $\mathbb{S}$

$$\hookrightarrow \subseteq \mathbb{S} \times \mathbb{S}$$

A state $s \in \mathbb{S}$ of the program is a pair (l, m) of a program label l and the machine state m at that program label during execution.

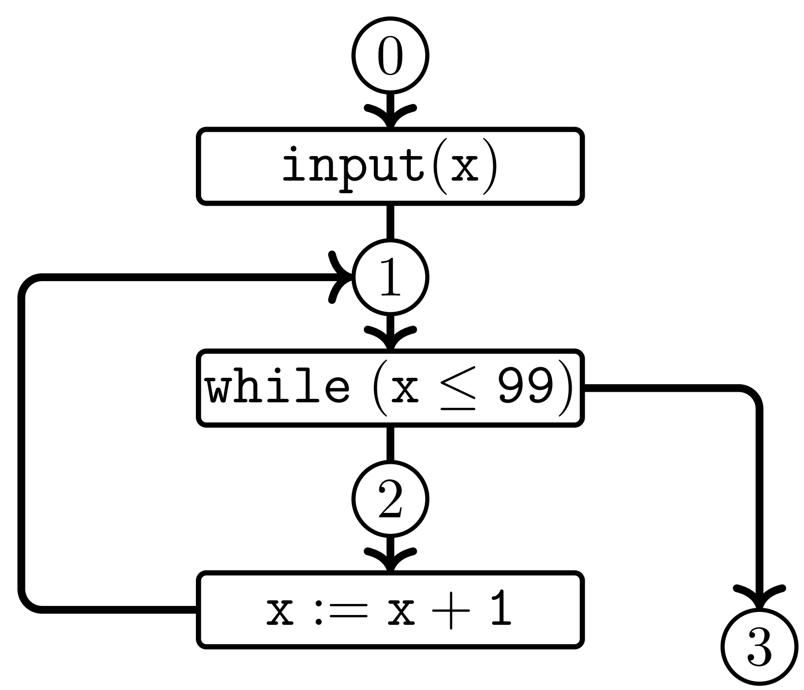
Concrete Transition Sequence

Example

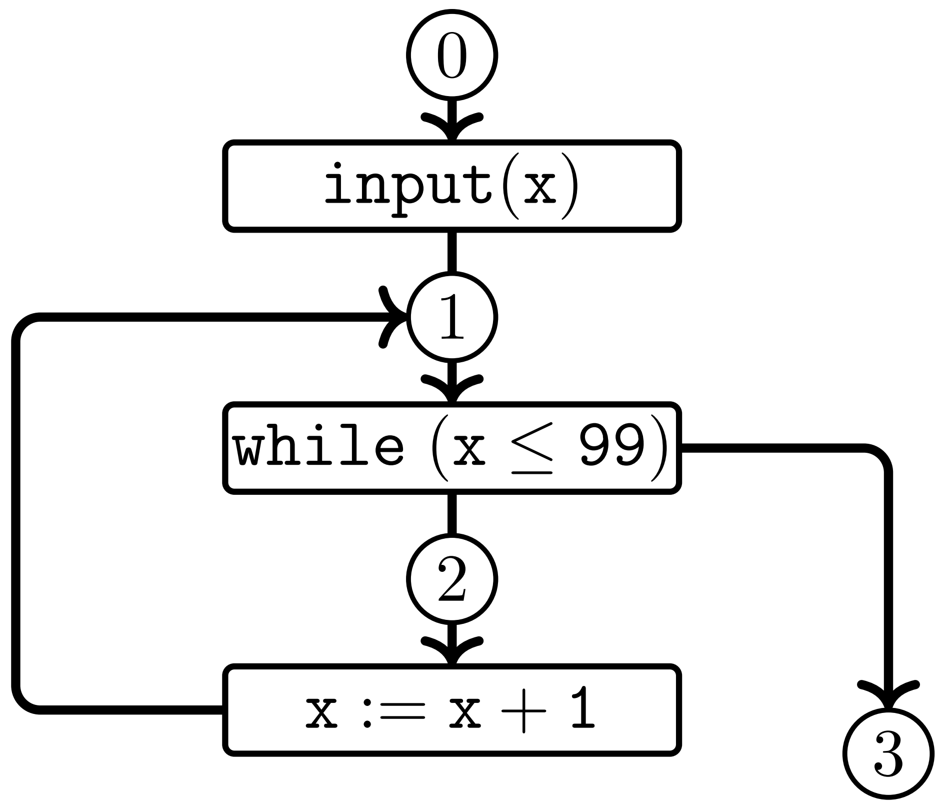
Consider the following program

$$\begin{array}{ll} \textbf{0:} & \texttt{input}(\mathrm{x}); \\ \textbf{1:} & \texttt{while}\ (\mathrm{x} \leq 99) \\ \textbf{2:} & \qquad \{\mathrm{x} := \mathrm{x} + 1\} \end{array}$$

Let labels be "program points". Such labeled representations of this program in graph is



Concrete Transition Sequence



Let the initial state be the empty memory $\emptyset$. Some transition sequences are:

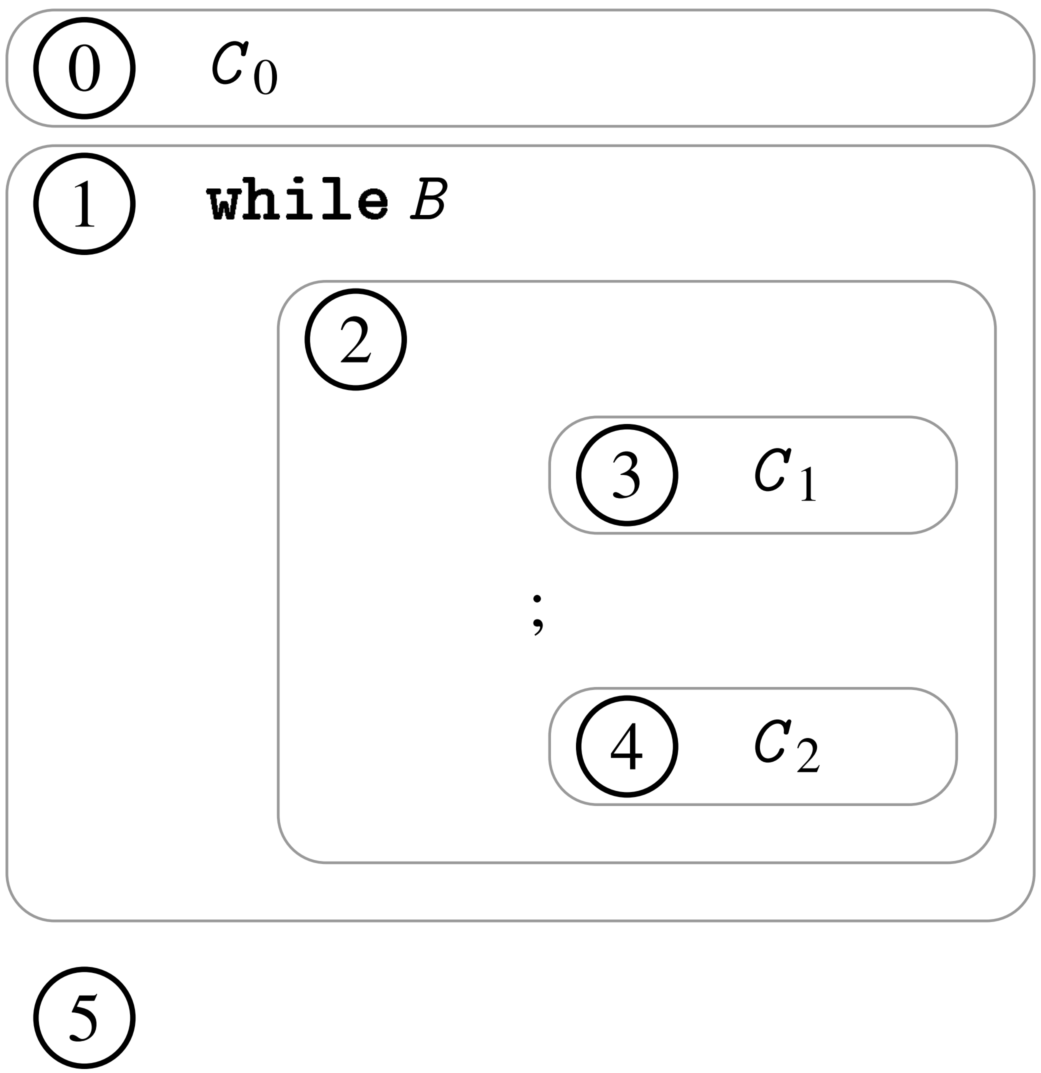
For input 100: $(0, \emptyset) \hookrightarrow (1, x \mapsto 100) \hookrightarrow (3, x \mapsto 100)$.
For input 99: $(0, \emptyset) \hookrightarrow (1, x \mapsto 99) \hookrightarrow (2, x \mapsto 99) \hookrightarrow (1, x \mapsto 100) \hookrightarrow (3, x \mapsto 100)$.
For input 0: $(0, \emptyset) \hookrightarrow (1, x \mapsto 0) \hookrightarrow (2, x \mapsto 0) \hookrightarrow (1, x \mapsto 1) \hookrightarrow \cdots \hookrightarrow (3, x \mapsto 100)$.

Semantic Domains

$$\mathbb{S}\text{tate} \stackrel{\text{def}}{=} \mathbb{L}\text{abel} \times \mathbb{M}\text{emory}$$

$$\begin{aligned} \mathbb{M}\text{emory} &\stackrel{\text{def}}{=} \mathit{Vars} \to \mathbb{V}\text{alue} \\ \mathbb{V}\text{alue} &\stackrel{\text{def}}{=} \mathbb{Z} \cup \mathbb{L}\text{abel}. \end{aligned}$$

Program Labels and Execution Order



$\mathtt{next}(0) = 1$

$\mathtt{nextTrue}(1) = 2 \quad \mathtt{next}(2) = 3$

$\mathtt{nextFalse}(1) = 5 \quad \mathtt{next}(3) = 4$

$\mathtt{next}(4) = 1$

State Transition

- The state transition relation $(l,m) \hookrightarrow (l',m')$ is defined by case analysis on statement labeled by l

$$
\begin{array}{rcl}
\texttt{skip} & : & (l,m) \hookrightarrow (\texttt{next}(l), m) \\
\texttt{input}\ \texttt{x} & : & (l,m) \hookrightarrow (\texttt{next}(l), \mathit{update}_{\texttt{x}}(m,z)) \qquad \text{for an input integer } z \\
\texttt{x} := E & : & (l,m) \hookrightarrow (\texttt{next}(l), \mathit{update}_{\texttt{x}}(m, \mathit{eval}_E(m))) \\
C_1; C_2 & : & (l,m) \hookrightarrow (\texttt{next}(l), m) \\
\texttt{if}\ B\ C_1\ C_2 & : & (l,m) \hookrightarrow (\texttt{nextTrue}(l), \mathit{filter}_B(m)) \\
 & : & (l,m) \hookrightarrow (\texttt{nextFalse}(l), \mathit{filter}_{\neg B}(m)) \\
\texttt{while}\ B\ C & : & (l,m) \hookrightarrow (\texttt{nextTrue}(l), \mathit{filter}_B(m)) \\
 & : & (l,m) \hookrightarrow (\texttt{nextFalse}(l), \mathit{filter}_{\neg B}(m)) \\
\texttt{goto}\ E & : & (l,m) \hookrightarrow (\mathit{eval}_E(m), m)
\end{array}
$$

Semantic Operators

- The memory update operation

$$update_x : \mathbb{M} \times \mathbb{V} \to \mathbb{M}$$
$$update_x(m, n) = m\{x \mapsto n\}$$

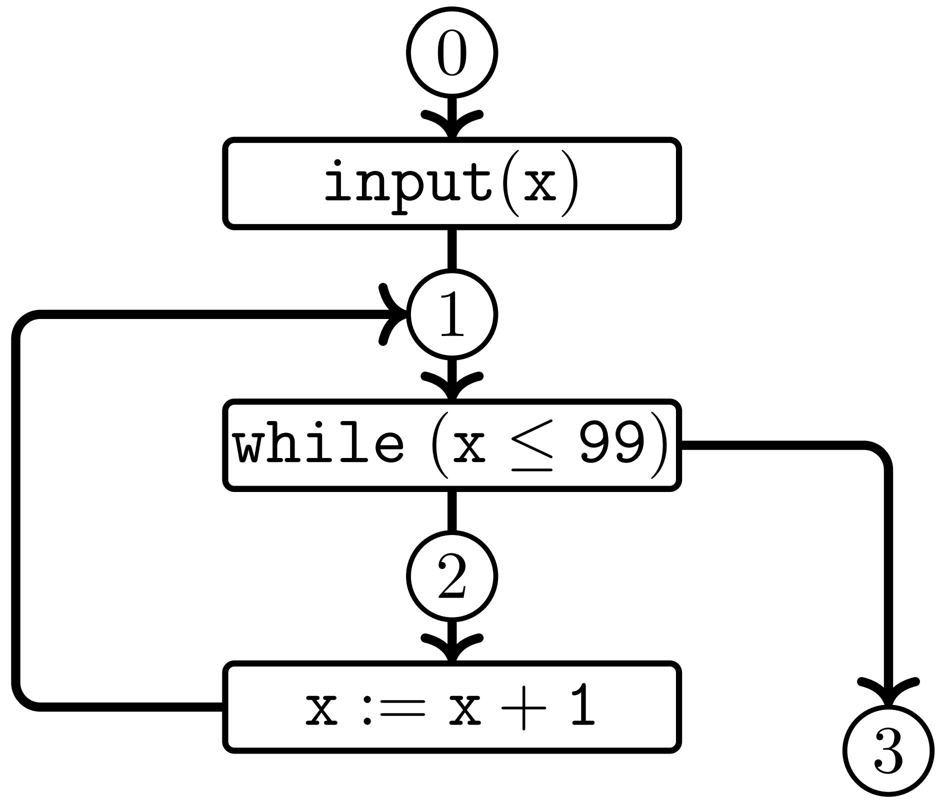
- The expression-evaluation operation

$$eval_E : \mathbb{M} \to \mathbb{V}$$
$$eval_n(m) = n$$
$$eval_x(m) = m(x)$$
$$eval_{E_1 \oplus E_2}(m) = eval_{E_1}(m) \oplus eval_{E_2}(m)$$

- The memory filter operation

$$filter_E : \mathbb{M} \to \mathbb{M}$$
$$filter_E(m) = m \quad \text{if } eval_E(m) = \mathsf{true}$$

Reachable States



Assume that the possible inputs are 0, 99, and 100. Then, the set of all reachable states are the set of states occurring in the three transition sequences:

$$
\begin{array}{ll}
& \{(0,\emptyset),(1,x\mapsto 100),(3,x\mapsto 100)\} \\
\cup & \{(0,\emptyset),(1,x\mapsto 99),(2,x\mapsto 99),(1,x\mapsto 100),(3,x\mapsto 100)\} \\
\cup & \{(0,\emptyset),(1,x\mapsto 0),(2,x\mapsto 0),(1,x\mapsto 1),\cdots,(2,x\mapsto 99),(1,x\mapsto 100),(3,x\mapsto 100)\} \\
= & \{(0,\emptyset),(1,x\mapsto 0),\cdots,(1,x\mapsto 100),(2,x\mapsto 0),\cdots,(2,x\mapsto 99),(3,x\mapsto 100)\}
\end{array}
$$

Concrete Semantics: the Set of Reachable States

Given a program, let I be the set of its initial states and *Step* be the powerset-lifted version of $\hookrightarrow$:

$$\begin{array}{l} \mathit{Step} : \wp(\mathbb{S}) \to \wp(\mathbb{S}) \\ \mathit{Step}(X) = \{s' \mid s \hookrightarrow s', s \in X\} \end{array}$$

The set of reachable states is

$$I \cup \mathit{Step}^1(I) \cup \mathit{Step}^2(I) \cup \cdots .$$

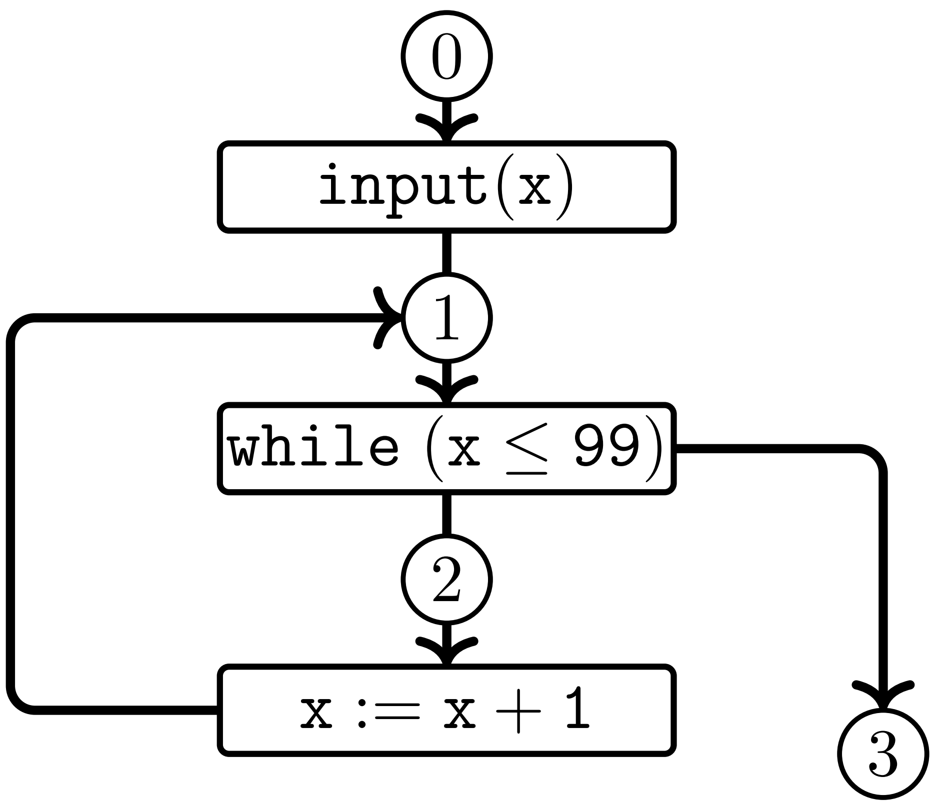
which is, equivalently, the limit of C_is

$$\begin{array}{rcl} C_0 & = & I \\ C_{i+1} & = & I \cup \mathit{Step}(C_i) \end{array}$$

which is, the least solution of

$$X = I \cup \mathit{Step}(X).$$

Example



From the set $I = \{(0, \emptyset)\}$ of initial states, assuming the possible inputs are 0, 99, and 100

$$
\begin{aligned}
\texttt{Step}^0(I) &= I \\
\texttt{Step}^1(I) &= \{(1, x \mapsto 100), (1, x \mapsto 99), (1, x \mapsto 0)\} \\
\texttt{Step}^2(I) &= \{(3, x \mapsto 100), (2, x \mapsto 99), (2, x \mapsto 0)\} \\
\texttt{Step}^3(I) &= \{(1, x \mapsto 100), (1, x \mapsto 1)\} \\
\texttt{Step}^4(I) &= \{(3, x \mapsto 100), (2, x \mapsto 1)\} \\
\texttt{Step}^5(I) &= \{(1, x \mapsto 2)\} \\
\texttt{Step}^6(I) &= \{(2, x \mapsto 2)\} \\
\texttt{Step}^7(I) &= \{(1, x \mapsto 3)\} \\
&\vdots
\end{aligned}
$$

All reachable states: $I \cup \texttt{Step}^1(I) \cup \texttt{Step}^2(I) \cup \cdots .$

Concrete Semantics: the Set of Reachable States

The least solution of

$$X = I \cup \textit{Step}(X)$$

is also called *the least fixpoint* of F

$$F : \wp(\mathbb{S}) \to \wp(\mathbb{S})$$
$$F(X) = I \cup \textit{Step}(X)$$

written as

$$\mathbf{lfp} F.$$

Concrete Semantics: the Set of Reachable States

Definition (Concrete semantics, the set of reachable states)

Given a program, let $\mathbb{S}$ be the set of states and $\hookrightarrow$ be the one-step transition relation $\subseteq \mathbb{S} \times \mathbb{S}$. Let I be the set of its initial states and *Step* be the powerset-lifted version of $\hookrightarrow$:

$$\begin{array}{l} \textit{Step} : \wp(\mathbb{S}) \rightarrow \wp(\mathbb{S}) \\ \textit{Step}(X) = \{s' \mid s \hookrightarrow s', s \in X\}. \end{array}$$

Then the concrete semantics of the program, the set of all reachable states from I, is defined as the least fixpoint $\mathbf{lfp}F$ of F

$$F(X) = I \cup \textit{Step}(X).$$

Analysis Goal

Program-label-wise reachability

For each program label we want to know the set of memories that can occur at that label during executions of the input program.

Notations

- An element of $A \to B$ is interchangeably an element in $\wp(A \times B)$
- A relation $f \subseteq A \times B$ is interchangeably a function $f \in A \to \wp(B)$:

$$f(a) = \{b \mid (a, b) \in f\}$$

  For example, $(\hookrightarrow) \subseteq \mathbb{S} \times \mathbb{S}$ is interchangeably a function $(\hookrightarrow) \in \mathbb{S} \to \wp(\mathbb{S})$

- For function $f : A \to B$, we write $\wp(f)$ is its powers version:

$$\wp(f) : \wp(A) \to \wp(B), \qquad \wp(f)(X) = \{f(x) \mid x \in X\}$$

Notations

- For function $f : A \to \wp(B)$, we write $\breve{\wp}(f)$ as a shorthand for $\cup \circ \wp(f)$:

  $$\breve{\wp}(f) : \wp(A) \to \wp(B), \qquad \breve{\wp}(f)(X) = \bigcup\{f(x) \mid x \in X\}$$

  For example, power-set-lifted function $\mathit{Step} : \wp(\mathbb{S}) \to \wp(\mathbb{S})$ of relation $\hookrightarrow$

  $$\mathit{Step}(X) = \{s' \mid s \hookrightarrow s', s \in X\}$$

  is equivalently, by regarding $\hookrightarrow$ as a function of $\mathbb{S} \to \wp(\mathbb{S})$:

  $$\mathit{Step}(X) = \bigcup\{(\hookrightarrow)(s) \mid s \in X\} = \cup \circ \wp(\hookrightarrow)(X) = \breve{\wp}(\hookrightarrow)(X)$$

- For function $f : A \to B$ and $g : A' \to B'$, we write (f, g) for

  $$(f, g) : A \times A' \to B \times B'$$
  $$(f, g)(a, a') = (f(a), g(a'))$$

Abstract Semantics

Define the abstract semantics similarly to the concrete semantics

$$F : \wp(\mathbb{S}) \rightarrow \wp(\mathbb{S})$$
$$F(X) = I \cup \mathit{Step}(X)$$

$$F^\sharp : \mathbb{S}^\sharp \rightarrow \mathbb{S}^\sharp$$
$$F^\sharp(X^\sharp) = I^\sharp \cup^\sharp \mathit{Step}^\sharp(X^\sharp)$$

Abstraction of the Semantic Domain $\wp(\mathbb{S})$

$$\wp(\mathbb{S}) \quad \text{where} \quad \mathbb{S} = \mathbb{L} \times \mathbb{M}$$

Design an abstract domain as a CPO that is Galois-connected with the concrete domain:

$$(\wp(\mathbb{L} \times \mathbb{M}), \subseteq) \xrightleftharpoons[\alpha]{\gamma} (\mathbb{L} \to \mathbb{M}^\sharp, \sqsubseteq).$$

- Abstraction α defines how each concrete elmt (set of concrete states) is abstracted into an abstract elmt.
- Concretization γ defines the set of concrete states implied by each abstract state.
- Partial order $\sqsubseteq$ is the label-wise order:

$$a^\sharp \sqsubseteq b^\sharp \quad \text{iff} \quad \forall l \in \mathbb{L} : a^\sharp(l) \sqsubseteq_M b^\sharp(l)$$

where $\sqsubseteq_M$ is the partial order of $\mathbb{M}^\sharp$.

Abstraction of the Semantic Domain $\wp(\mathbb{S})$

Label-wise (two-step) abstraction of states:

set of states	to	label-wise collect	to	label-wise abstraction
$\wp(\mathbb{L} \times \mathbb{M})$	$\xrightarrow{\text{abstraction}}$	$\mathbb{L} \to \wp(\mathbb{M})$	$\xrightarrow{\text{abstraction}}$	$\mathbb{L} \to \mathbb{M}^\sharp$.

$$
\begin{array}{l}
\langle 0, m_0 \rangle, \langle 0, m_0' \rangle, \cdots, \\
\langle 1, m_1 \rangle, \langle 1, m_1' \rangle, \cdots, \\
\vdots \\
\langle n, m_n \rangle, \langle n, m_n' \rangle, \cdots,
\end{array}
$$

collection of all states

partitioning abstraction →

$$
\begin{array}{l}
\langle 0, \{m_0, m_0', \cdots\} \rangle \\
\langle 1, \{m_1, m_1', \cdots\} \rangle \\
\vdots \\
\langle n, \{m_n, m_n', \cdots\} \rangle
\end{array}
$$

label-wise collection of memories

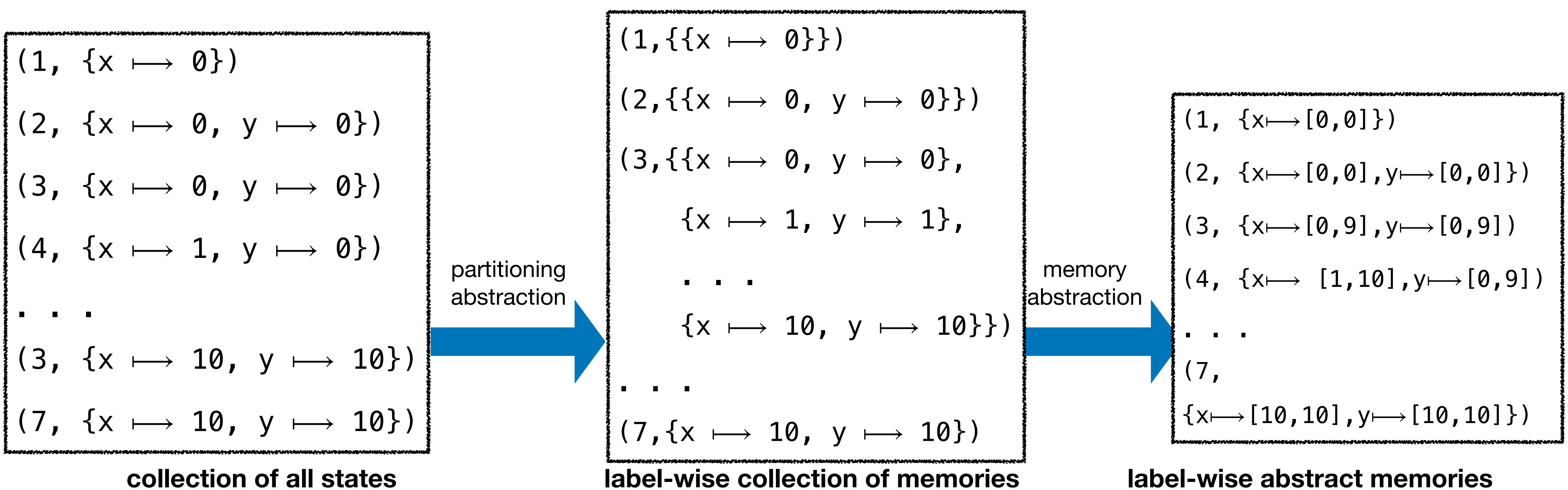
memory abstraction →

$$
\begin{array}{c}
\langle 0, m_0^\sharp \rangle \\
\langle 1, m_1^\sharp \rangle \\
\vdots \\
\langle n, m_n^\sharp \rangle
\end{array}
$$

label-wise abstract memories

Abstraction of the Semantic Domain $\wp(\mathbb{S})$ (Example)

```
1:  x := 0;
2:  y := 0;
3:  while (x < 10) {
4:    x := x + 1;
5:    y := y + 1;
6:  }
7:  skip
```



Abstraction of the Semantic Domain $\wp(\mathbb{S})$

The above Galois connection (abstraction)

$$(\wp(\mathbb{L} \times \mathbb{M}), \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathbb{L} \to \mathbb{M}^\sharp, \sqsubseteq).$$

composes two Galois connections:

$$\begin{array}{l} (\wp(\mathbb{L} \times \mathbb{M}), \subseteq) \\ \underset{\alpha_0}{\overset{\gamma_0}{\leftrightarrows}} \quad (\mathbb{L} \to \wp(\mathbb{M}), \sqsubseteq) \quad (\sqsubseteq \text{ is the label-wise } \subseteq) \\ \qquad\qquad \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} (\mathbb{L} \to \mathbb{M}^\sharp, \sqsubseteq) \quad (\sqsubseteq \text{ is the label-wise } \sqsubseteq_M) \end{array}$$

Partitioning Abstraction

$$(\wp(\mathbb{L} \times \mathbb{M}), \subseteq) \underset{\alpha_0}{\overset{\gamma_0}{\longleftrightarrow}} (\mathbb{L} \to \wp(\mathbb{M}), \sqsubseteq)$$ ($\sqsubseteq$ is the label-wise $\subseteq$)

$$\alpha_0 \left\{ \begin{array}{l} (0, m_0), (0, m_0'), \cdots, \\ \quad \vdots \\ (n, m_n), (n, m_n'), \cdots \end{array} \right\} = \left\{ \begin{array}{l} (0, \{m_0, m_0', \cdots\}), \\ \quad \vdots \\ (n, \{m_n, m_n', \cdots\}) \end{array} \right\}$$

$$\alpha_0(S) = \lambda l.\ \{m \in \mathbb{M} \mid (l, m) \in S\}$$

$$\gamma_0(\Pi) = \{(l, m) \mid m \in \Pi(l)\}$$

Memory Abstraction

$(\mathbb{L} \to \wp(\mathbb{M}), \sqsubseteq)$ ($\sqsubseteq$ is the label-wise $\subseteq$)

$\xleftrightarrow[\alpha_1]{\gamma_1} (\mathbb{L} \to \mathbb{M}^\sharp, \sqsubseteq)$ ($\sqsubseteq$ is the label-wise $\sqsubseteq_M$)

$$\alpha_1 \left\{ \begin{array}{c} (0, \{m_0, m_0', \cdots\}), \\ \vdots \\ (n, \{m_n, m_n', \cdots\}) \end{array} \right\} = \left\{ \begin{array}{c} (0, M_0^\sharp), \\ \vdots \\ (n, M_n^\sharp) \end{array} \right\}$$

$$\alpha_1(X) = \lambda l.\ \alpha_{\mathbb{M}}(X(l))$$
$$\gamma_1(X^\#) = \lambda l.\ \gamma_{\mathbb{M}}(X^\#(l))$$

where

$$(\wp(\mathbb{M}), \subseteq) \xleftrightarrow[\alpha_M]{\gamma_M} (\mathbb{M}^\sharp, \sqsubseteq_M).$$

Abstract Domains

- Galois connection for abstract memories

$$(\wp(\mathbb{M}\text{emory}), \subseteq) \underset{\alpha_M}{\overset{\gamma_M}{\leftrightarrows}} (\mathbb{M}\text{emory}^\sharp, \sqsubseteq_M).$$

$$m^\sharp \in \mathbb{M}\text{emory}^\sharp \stackrel{\text{def}}{=} \mathit{Vars} \to \mathit{Value}^\sharp$$

$$\alpha_{\mathbb{M}}(M) = \lambda x.\ \alpha_V(\{m(x) \mid m \in M\})$$

$$\gamma_{\mathbb{M}}(m^\#) = \{m \mid \forall x.\ m(x) \in \gamma_V(m^\#(x))\}$$

- Ordered variable-wise

$$m_1^\sharp \sqsubseteq_{\mathbb{M}^\sharp} m_2^\sharp \iff \forall x \in \mathbb{X}.\ m_1^\sharp(x) \sqsubseteq_{\mathbb{V}^\sharp} m_2^\sharp(x)$$

$$m_1^\sharp \sqcup_{\mathbb{M}^\sharp} m_2^\sharp = \lambda x.\big(m_1^\sharp(x) \sqcup_{\mathbb{V}^\sharp} m_2^\sharp(x)\big)$$

Abstract Domains

- Abstract values

$$(\wp(\mathbb{V}\text{alue}), \subseteq) \xleftarrow[\alpha_V]{\gamma_V} (\mathbb{V}\text{alue}^\sharp, \sqsubseteq_V).$$

$$\mathbb{V}\text{alue}^\sharp \stackrel{\text{def}}{=} \mathbb{Z}^\sharp \times \mathbb{L}\text{abel}^\sharp$$

where $\mathbb{Z}^\sharp$ is an interval domain (a CPO) and $\mathbb{L}\text{abel}^\sharp$ is just a powerset $\wp(\mathbb{L}\text{abel})$ of labels (a CPO).

Abstract State Transition

- The abstract semantics is defined using a transition system $(\mathbb{S}^\sharp, \hookrightarrow^\sharp)$
    - $\mathbb{S}^\sharp = \mathbb{L} \times \mathbb{M}^\sharp$: the set of states $\langle l, m^\sharp \rangle$
    - $(\hookrightarrow^\sharp) \subseteq \mathbb{S}^\sharp \times \mathbb{S}^\sharp$: the transition relation that describes computation steps

Abstract State Transition

The abstract state transition relation $(l, m^\sharp) \hookrightarrow^\sharp (l', m^{\sharp\prime})$

Case the l-labeled statement of

$$
\begin{array}{rcl}
\texttt{skip} & : & (l, m^\sharp) \hookrightarrow^\sharp (\texttt{next}(l), m^\sharp) \\
\texttt{input x} & : & (l, m^\sharp) \hookrightarrow^\sharp (\texttt{next}(l), \mathit{update}^\sharp_{\texttt{x}}(m^\sharp, \alpha(\mathbb{Z}))) \\
\texttt{x} := E & : & (l, m^\sharp) \hookrightarrow^\sharp (\texttt{next}(l), \mathit{update}^\sharp_{\texttt{x}}(m^\sharp, \mathit{eval}^\sharp_E(m^\sharp))) \\
C_1; C_2 & : & (l, m^\sharp) \hookrightarrow^\sharp (\texttt{next}(l), m^\sharp) \\
\texttt{if}\ B\ C_1\ C_2 & : & (l, m^\sharp) \hookrightarrow^\sharp (\texttt{nextTrue}(l), \mathit{filter}^\sharp{}_B(m^\sharp)) \\
 & : & (l, m^\sharp) \hookrightarrow^\sharp (\texttt{nextFalse}(l), \mathit{filter}^\sharp{}_{\neg B}(m^\sharp)) \\
\texttt{while}\ B\ C & : & (l, m^\sharp) \hookrightarrow^\sharp (\texttt{nextTrue}(l), \mathit{filter}^\sharp{}_B(m^\sharp)) \\
 & : & (l, m^\sharp) \hookrightarrow^\sharp (\texttt{nextFalse}(l), \mathit{filter}^\sharp{}_{\neg B}(m^\sharp)) \\
\texttt{goto}\ E & : & (l, m^\sharp) \hookrightarrow^\sharp (l', m^\sharp) \quad \text{for } l' \in L \text{ of } (z^\sharp, L) \stackrel{\text{def}}{=} \mathit{eval}^\sharp_E(m^\sharp)
\end{array}
$$

Abstract Semantic Operators

- The abstract memory update operation:

$$update^\sharp_x : \mathbb{V}^\sharp \times \mathbb{M}^\sharp \to \mathbb{M}^\sharp$$
$$update^\sharp_x(n^\sharp, m^\sharp) = m^\sharp\{x \mapsto n^\sharp\}$$

- The abstract expression-evaluation operation:

$$eval^\sharp_E : \mathbb{M}^\sharp \to \mathbb{V}^\sharp$$
$$eval^\sharp_n(m) = \alpha_{\mathbb{Z}}(\{n\})$$
$$eval^\sharp_x(m) = m^\sharp(x)$$
$$eval^\sharp_{E_1 \oplus E_2}(m) = eval^\sharp_{E_1}(m^\sharp) \oplus^\sharp eval^\sharp_{E_2}(m^\sharp)$$

- The abstract memory filter operation:

$$filter^\sharp_E : \mathbb{M}^\sharp \to \mathbb{M}^\sharp$$
$$filter^\sharp_E(m^\sharp) = \alpha_{\mathbb{M}}(\{m \in \gamma_{\mathbb{M}}(m^\sharp) \mid eval_E(m) = \mathsf{true}\})$$

Abstract Semantics

- The abstract semantic functions:

$$F^\sharp : \mathbb{S}\text{tate}^\sharp \to State^\sharp$$

$$F^\sharp(S^\sharp) \stackrel{\text{def}}{=} \alpha(I) \cup^\sharp \texttt{Step}^\sharp(S^\sharp)$$

$$\texttt{Step}^\sharp \stackrel{\text{def}}{=} \wp(id, \sqcup_M) \circ \pi \circ \breve{\wp}(\hookrightarrow^\sharp).$$

where

$$\pi : \wp(\mathbb{S}^\sharp) \to (\mathbb{L} \to \wp(\mathbb{M}^\sharp))$$

$$\pi(X) = \lambda l.\{m^\sharp \in \mathbb{M}^\sharp \mid \langle l, m^\sharp \rangle \in X\}$$

- Soundness: $\mathbf{lfp}F \subseteq \gamma_0 \circ \gamma_1 (\bigsqcup_{i \geq 0} F^{\#^i}(\bot))$

Abstract Step Function

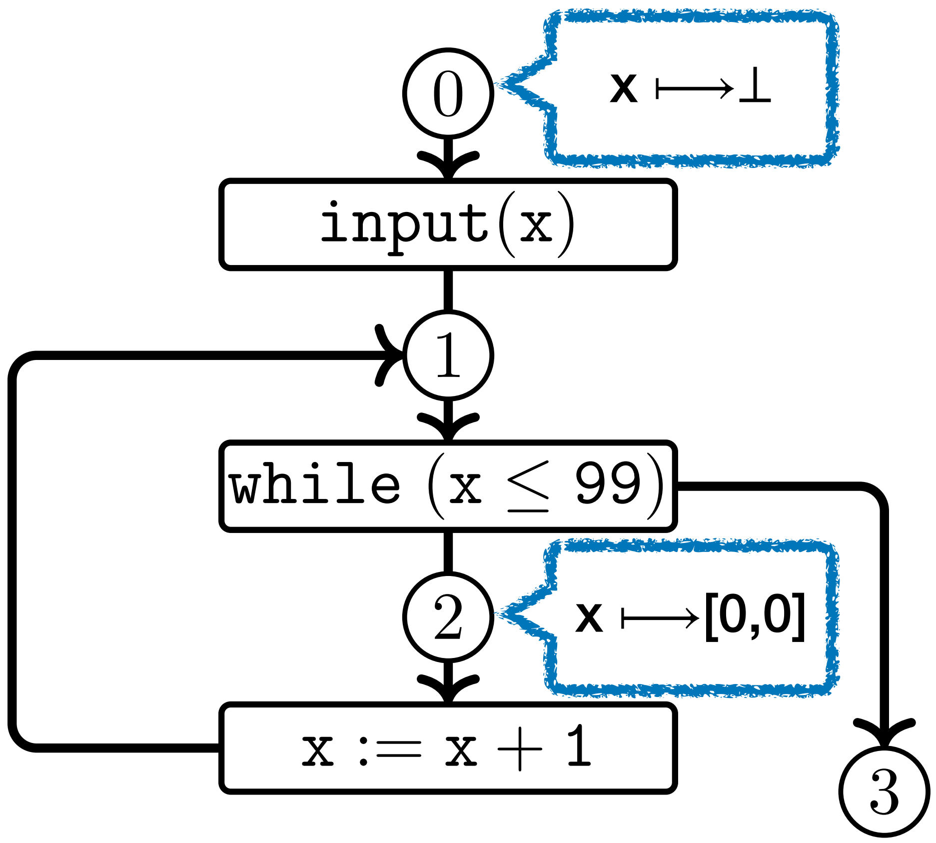
$\textit{Step}^\sharp : (\mathbb{L} \to \mathbb{M}^\sharp) \to (\mathbb{L} \to \mathbb{M}^\sharp)$

- Abstract transition $\breve{\wp}(\hookrightarrow^\sharp)$:
  - a set $\subseteq \mathbb{L} \times \mathbb{M}^\sharp \quad \mapsto \quad$ a set $\subseteq \mathbb{L} \times \mathbb{M}^\sharp$
- Paritioning π:
  - a set $\subseteq \mathbb{L} \times \mathbb{M}^\sharp \quad \mapsto \quad$ a set $\subseteq \mathbb{L} \times \wp(\mathbb{M}^\sharp)$
- Joining $\wp(\mathrm{id}, \sqcup_M)$:
  - a set $\subseteq \mathbb{L} \times \wp(\mathbb{M}^\sharp) \quad \mapsto \quad$ an abstract state $\in \mathbb{L} \to \mathbb{M}^\sharp$

$$\wp(id, \sqcup)(X) = \{(id(l), \bigsqcup M^{\#}) \mid (l, M^{\#}) \in X\}$$

Abstract Step Function

Let $S^{\#} = \{(0, x \mapsto \bot), (2, x \mapsto [0,0])\}$

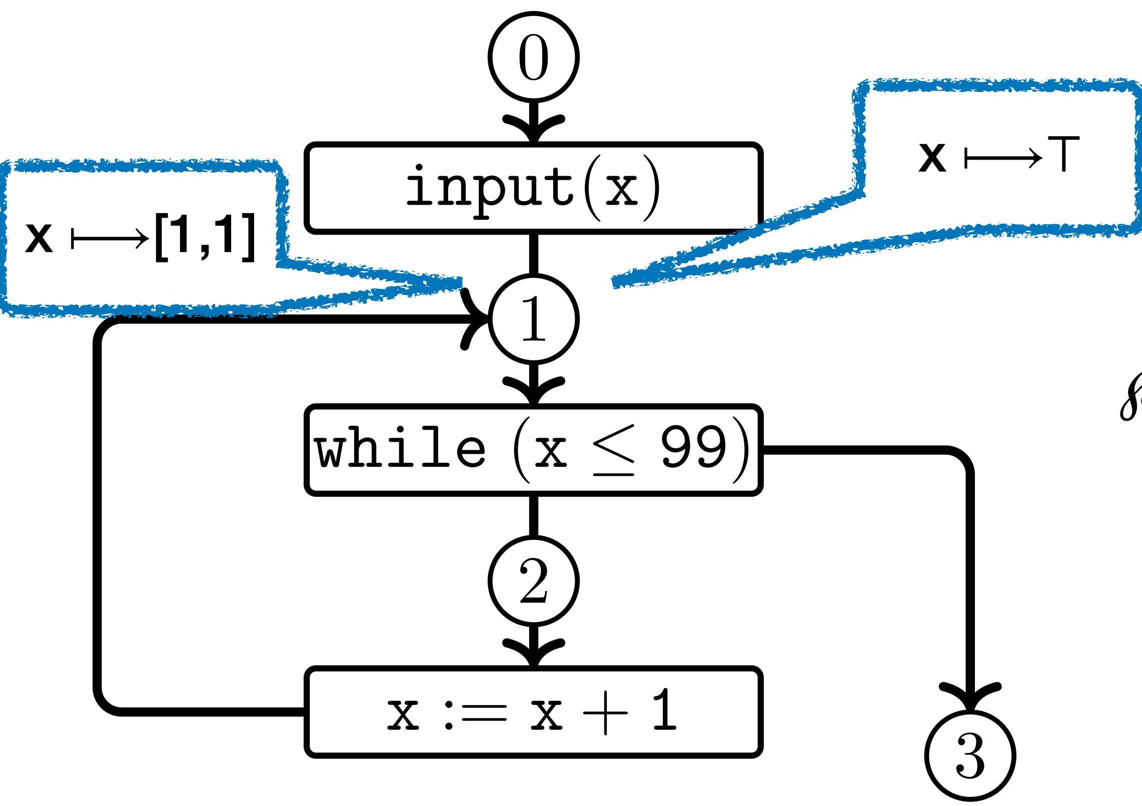


$$Step^{\sharp} = \wp(\mathrm{id}, \sqcup_M) \circ \pi \circ \breve{\wp}(\hookrightarrow^{\sharp})$$

$\mathrm{Step}^{\#}(S^{\#})$:

Abstract Step Function

Let $S^{\#} = \{(0, x \mapsto \bot), (2, x \mapsto [0,0])\}$



$$Step^{\sharp} = \wp(\mathrm{id}, \sqcup_M) \circ \pi \circ \breve{\wp}(\hookrightarrow^{\sharp})$$

$\mathrm{Step}^{\#}(S^{\#})$:

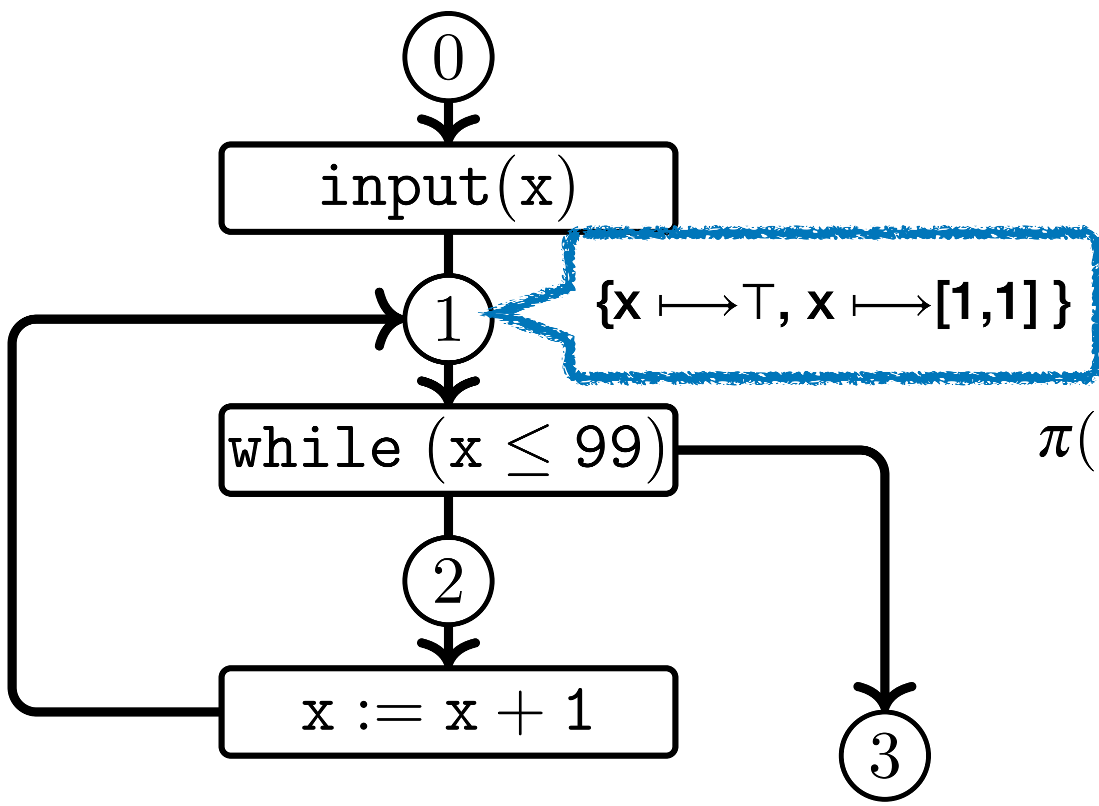
$$\begin{aligned}
\breve{\wp}(\hookrightarrow^{\#})(S^{\#}) &= \hookrightarrow^{\#}(0, x \mapsto \bot) \cup \hookrightarrow^{\#}(2, x \mapsto [0,0]) \\
&= \{(1, x \mapsto \top)\} \cup \{(1, x \mapsto [1,1])\} \\
&= \{(1, x \mapsto \top), (1, x \mapsto [1,1])\}
\end{aligned}$$

$$\pi(\quad) = \cdots$$

Abstract Step Function

Let $S^{\#} = \{(0, x \mapsto \bot), (2, x \mapsto [0,0])\}$

$$\mathit{Step}^{\sharp} = \wp(\mathrm{id}, \sqcup_M) \circ \pi \circ \breve{\wp}(\hookrightarrow^{\sharp})$$



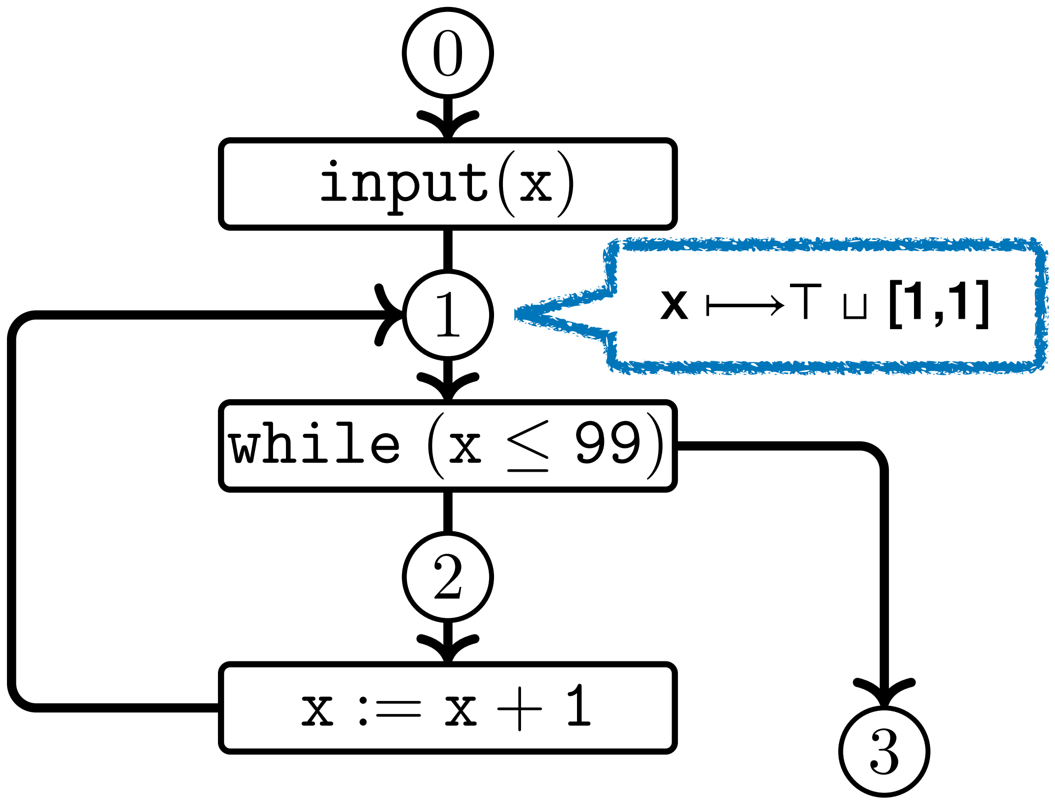
$\mathrm{Step}^{\#}(S^{\#})$:

$$\pi(\{(1, x \mapsto \top), (1, x \mapsto [1,1])\}) \quad = \quad \{(1, \{x \mapsto \top, x \mapsto [1,1]\})\}$$

$$\wp(id, \sqcup)(\quad) = \cdots$$

Abstract Step Function

Let $S^{\#} = \{(0, x \mapsto \bot), (2, x \mapsto [0,0])\}$



$$\mathit{Step}^{\sharp} = \wp(\mathrm{id}, \sqcup_M) \circ \pi \circ \breve{\wp}(\hookrightarrow^{\sharp})$$

$\mathrm{Step}^{\#}(S^{\#})$:

$$\begin{aligned} \wp(id, \sqcup)(\{(1, \{x \mapsto \top, x \mapsto [1,1]\})\}) &= \{(id(1), \bigsqcup_M \{x \mapsto \top, x \mapsto [1,1]\})\} \\ &= \{1, x \mapsto \top\} \end{aligned}$$

Basic Fixpoint Computation Algorithm

- If the abstract domain $\mathbb{S}\text{tate}^\sharp$ is of finite-height, and $F^\sharp$ is monotone or extensive, the increasing chain

  $$\bot \sqsubseteq (F^\sharp)^1(\bot) \sqsubseteq (F^\sharp)^2(\bot) \sqsubseteq \cdots$$

  is finite and its biggest element is

  $$\bigsqcup_{i \geq 0} {F^\sharp}^i(\bot).$$

  and over-approximates $\mathbf{lfp}F$

Basic Fixpoint Computation Algorithm

- Otherwise, find a widening operator ∇, then the following chain $X_0 \sqsubseteq X_1 \sqsubseteq \cdots$

$$X_0 = \bot \qquad X_{i+1} = X_i \bigtriangledown F^\sharp(X_i)$$

is finite and its last element over-approximates the concrete semantics **lfp**F.

Basic Fixpoint Computation Algorithm

- Hence, if the abstract domain is finite, the algorithm is

$$
\begin{cases}
\texttt{C} \leftarrow \bot \\
\text{repeat} \\
\qquad \texttt{R} \leftarrow \texttt{C} \\
\qquad \texttt{C} \leftarrow F^\sharp(\texttt{C}) \\
\text{until } \texttt{C} \sqsubseteq \texttt{R} \\
\text{return } \texttt{R}
\end{cases}
$$

Basic Fixpoint Computation Algorithm

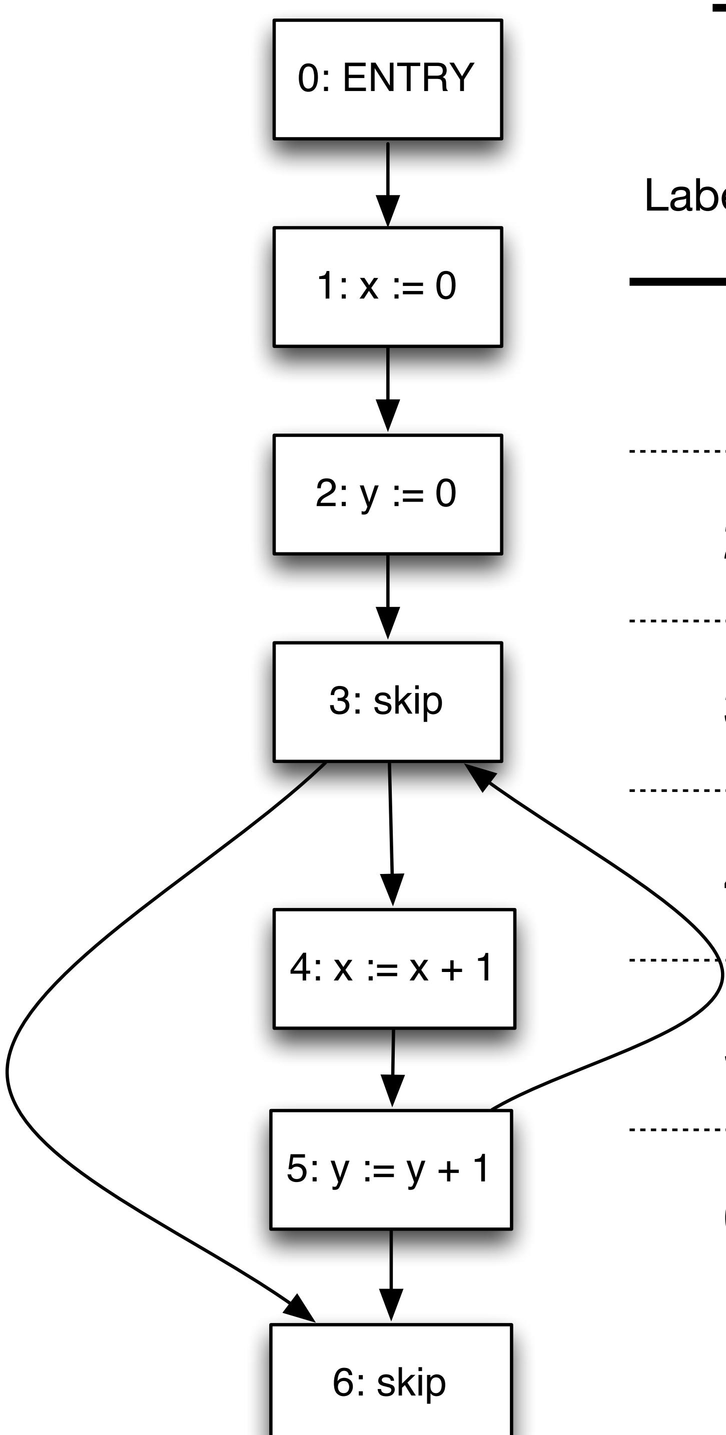
- Hence, if the abstract domain is finite, the algorithm is

$$
\left\{
\begin{array}{l}
\mathtt{C} \leftarrow \bot \\
\text{repeat} \\
\qquad \mathtt{R} \leftarrow \mathtt{C} \\
\qquad \mathtt{C} \leftarrow \underbrace{(\wp(id, \sqcup) \circ \pi \circ \breve{\wp}(\hookrightarrow^{\sharp}))}_{F^{\sharp}}(\mathtt{C}) \\
\text{until } \mathtt{C} \sqsubseteq \mathtt{R} \\
\text{return } \mathtt{R}
\end{array}
\right.
$$

Example: Sign Analysis

Fixpoint reached!

+ : [≥0] **0 :** [=0]



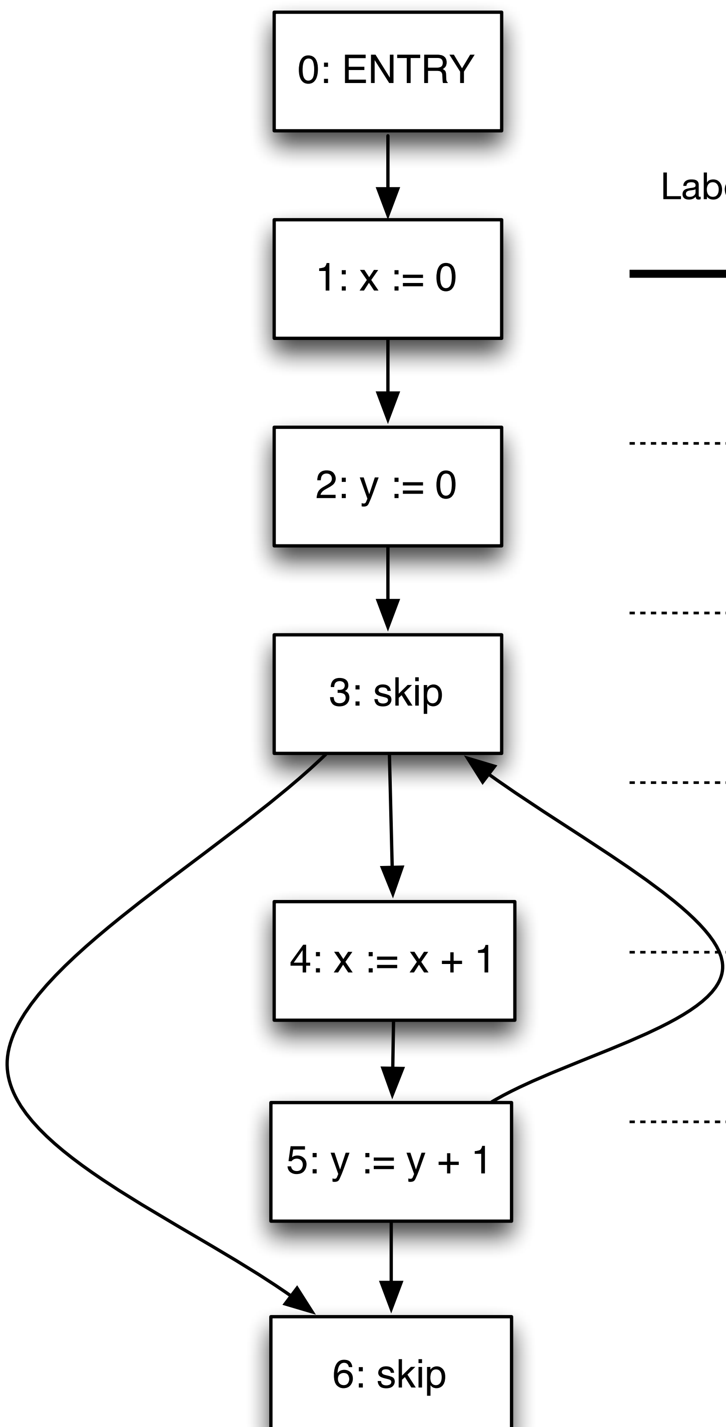
Label\Iter	1	2	3	4	5	6	7	8
1	{x ↦ **0**}	{x ↦ **0**}	{x ↦ **0**}	{x ↦ **0**}	{x ↦ **0**}	{x ↦ **0**}	{x ↦ **0**}	{x ↦ **0**}
2	{y ↦ **0**}	{x ↦ **0**, y ↦ **0**}	{x ↦ **0**, y ↦ **0**}	{x ↦ **0**, y ↦ **0**}	{x ↦ **0**, y ↦ **0**}	{x ↦ **0**, y ↦ **0**}	{x ↦ **0**, y ↦ **0**}	{x ↦ **0**, y ↦ **0**}
3	{}	{y ↦ **0**}	{x ↦ **0**, y ↦ **0**}	{x ↦ **0**, y ↦ **0**}	{x ↦ **0**, y ↦ **0**}	{x ↦ **+**, y ↦ **+**}	{x ↦ **+**, y ↦ **+**}	{x ↦ **+**, y ↦ **+**}
4	{}	{}	{}	{x ↦ **+**, y ↦ **0**}	{x ↦ **+**, y ↦ **0**}	{x ↦ **+**, y ↦ **0**}	{x ↦ **+**, y ↦ **+**}	{x ↦ **+**, y ↦ **+**}
5	{}	{}	{}	{}	{x ↦ **+**, y ↦ **+**}	{x ↦ **+**, y ↦ **+**}	{x ↦ **+**, y ↦ **+**}	{x ↦ **+**, y ↦ **+**}
6	{}	{}	{y ↦ **0**}	{x ↦ **0**, y ↦ **0**}	{x ↦ **0**, y ↦ **0**}	{x ↦ **+**, y ↦ **+**}	{x ↦ **+**, y ↦ **+**}	{x ↦ **+**, y ↦ **+**}

Basic Fixpoint Computation Algorithm

- If the abstract domain is of infinite-height, the algorithm is

$$\begin{cases} \mathtt{C} \leftarrow \bot \\ \text{repeat} \\ \qquad \mathtt{R} \leftarrow \mathtt{C} \\ \qquad \mathtt{C} \leftarrow \mathtt{C} \bigtriangledown F^{\sharp}(\mathtt{C}) \\ \text{until } \mathtt{C} \sqsubseteq \mathtt{R} \\ \text{return } \mathtt{R} \end{cases}$$

Example: Interval Analysis



[0,0] ▽ [0,1] = [0,+∞]

Fixpoint reached!

Label\Iter	1	2	3	4	5	6	7	8
1	{x ⟼[0,0]}	{x ⟼[0,0]}	{x ⟼[0,0]}	{x ⟼[0,0]}	{x ⟼[0,0]}	{x ⟼[0,0]}	{x ⟼[0,0]}	{x ⟼[0,0]}
2	{y ⟼[0,0]}	{x⟼[0,0], y⟼[0,0]}	{x⟼[0,0], y⟼[0,0]}	{x⟼[0,0], y⟼[0,0]}	{x⟼[0,0], y⟼[0,0]}	{x⟼[0,0], y⟼[0,0]}	{x⟼[0,0], y⟼[0,0]}	{x⟼[0,0], y⟼[0,0]}
3	{}	{y ⟼[0,0]}	{x⟼[0,0], y⟼[0,0]}	{x⟼[0,0], y⟼[0,0]}	{x⟼[0,0], y⟼[0,0]}	{x⟼[0,∞], y⟼[0,∞]}	{x⟼[0,∞], y⟼[0,∞]}	{x⟼[0,∞], y⟼[0,∞]}
4	{}	{}	{}	{x⟼[1,1], y⟼[0,0]}	{x⟼[1,1], y⟼[0,0]}	{x⟼[1,1], y⟼[0,0]}	{x⟼[1,∞], y⟼[0,∞]}	{x⟼[1,∞], y⟼[0,∞]}
5	{}	{}	{}	{}	{x⟼[1,1], y⟼[1,1]}	{x⟼[1,1], y⟼[1,1]}	{x⟼[1,1], y⟼[1,1]}	{x⟼[1,∞], y⟼[1,∞]}
6	{}	{}	{y ⟼[0,0]}	{x⟼[0,0], y⟼[0,0]}	{x⟼[0,0], y⟼[0,0]}	{x⟼[0,∞], y⟼[0,∞]}	{x⟼[0,∞], y⟼[0,∞]}	{x⟼[0,∞], y⟼[0,∞]}

Inefficiency of the Basic Algorithm

Recall the algirthm with $F^\sharp(\texttt{C})$ being inlined:

$$
\begin{array}{|l}
\texttt{C} \leftarrow \bot \\
\text{repeat} \\
\qquad \texttt{R} \leftarrow \texttt{C} \\
\qquad \texttt{C} \leftarrow \texttt{C} \bigtriangledown \underbrace{(\wp(\mathrm{id}, \sqcup) \circ \pi \circ \breve{\wp}(\hookrightarrow^\sharp))}_{F^\sharp}(\texttt{C}) \\
\text{until } \texttt{C} \sqsubseteq \texttt{R} \\
\text{return } \texttt{R}
\end{array}
$$

- $|\texttt{C}| \sim$ the number of labels in the input program!
- Better apply

$$\breve{\wp}(\hookrightarrow^\sharp)(\texttt{C})$$

  only to necessary labels

Worklist Algorithm

- worklist: the set of labels whose input memories are changed in the previous iteration

$$\mathtt{C} : \mathbb{L} \to \mathbb{M}^\sharp$$
$$F^\sharp : (\mathbb{L} \to \mathbb{M}^\sharp) \to (\mathbb{L} \to \mathbb{M}^\sharp)$$
$$\mathtt{WorkList} : \wp(\mathbb{L})$$

$\mathtt{WorkList} \leftarrow \mathbb{L}$

$\mathtt{C} \leftarrow \bot$

repeat

  $\mathtt{R} \leftarrow \mathtt{C}$

  $\mathtt{C} \leftarrow \mathtt{C} \bigtriangledown F^\sharp(\mathtt{C}|_{\mathtt{WorkList}})$

  $\mathtt{WorkList} \leftarrow \{l \mid \mathtt{C}(l) \not\sqsubseteq \mathtt{R}(l), l \in \mathbb{L}\}$

until $\mathtt{WorkList} = \emptyset$

return $\mathtt{R}$

Improvement of the Worklist Algorithm

- Inefficient: $\texttt{WorkList} \leftarrow \{l \mid \texttt{C}(l) \not\sqsubseteq \texttt{R}(l), l \in \mathbb{L}\}$ re-scans all the labels.
  - Better: At application $\hookrightarrow^\sharp$ to $(l, \texttt{C}(l))$, if its result $(l', M^\sharp)$ is changed $(M^\sharp \not\sqsubseteq \texttt{C}(l'))$, add l' to the worklist.
- Inefficient: $\texttt{C} \bigtriangledown F^\sharp(C|_{\texttt{WorkList}})$ widens at all the labels.
  - Better: Apply $\bigtriangledown$ only at the target of a loop. Use $\cup^\sharp$ at other labels.

Worklist Algorithm with Widening

$X : \mathbb{L} \to \mathbb{M}^\sharp$

$F^\sharp : (\mathbb{L} \to \mathbb{M}^\sharp) \to (\mathbb{L} \to \mathbb{M}^\sharp)$

$Worklist : \wp(\mathbb{L})$

begin

  $Worklist \leftarrow \mathbb{L}$

  $X \leftarrow \bot$

  repeat

    $(w, Worklist) \leftarrow \mathsf{pop}(Worklist)$

    $m^\sharp_{old} \leftarrow X(w)$

    $m^\sharp_{new} \leftarrow \bigsqcup\{m^\sharp_{out} \mid \langle l, X(l)\rangle \hookrightarrow^\sharp \langle w, m^\sharp_{out}\rangle\}$

    if $m^\sharp_{new} \not\sqsubseteq m^\sharp_{old}$ then

      $m^\sharp_{new} \leftarrow m^\sharp_{old} \triangledown m^\sharp_{new}$ if w is a loop head else $m^\#_{old} \sqcup m^\#_{new}$

      $X(w) \leftarrow m^\sharp_{new}$

      $Worklist \leftarrow Worklist \cup \{l \mid \langle w, m^\sharp_{new}\rangle \hookrightarrow^\sharp \langle l, \_\rangle\}$

    endif

  until $Worklist = \emptyset$

  return X

end

Worklist Algorithm with Narrowing

$X : \mathbb{L} \to \mathbb{M}^\sharp$

$F^\sharp : (\mathbb{L} \to \mathbb{M}^\sharp) \to (\mathbb{L} \to \mathbb{M}^\sharp)$

$\mathit{Worklist} : \wp(\mathbb{L})$

begin

  $\mathit{Worklist} \leftarrow \mathbb{L}$

  $X \leftarrow \bot$

  repeat

    $(w, \mathit{Worklist}) \leftarrow$ pop($\mathit{Worklist}$)

    $m^\sharp_{old} \leftarrow X(w)$

    $m^\sharp_{new} \leftarrow \bigsqcup\{m^\sharp_{out} \mid \langle l, X(l)\rangle \hookrightarrow^\sharp \langle w, m^\sharp_{out}\rangle\}$

    if $m^\sharp_{new} \not\sqsupseteq m^\sharp_{old}$ then

      $m^\sharp_{new} \leftarrow m^\sharp_{old} \bigtriangleup m^\sharp_{new}$

      $X(w) \leftarrow m^\sharp_{new}$

      $\mathit{Worklist} \leftarrow \mathit{Worklist} \cup \{l \mid \langle w, m^\sharp_{new}\rangle \hookrightarrow^\sharp \langle l, \_\rangle\}$

    endif

  until $\mathit{Worklist} = \emptyset$

  return X

end

Soundness

Theorem (Sound static analysis by $F^\sharp$)

Given a program, let F and $F^\sharp$ be defined as in the framework. If $\mathbb{S}^\sharp$ is of finite-height (every chain $\mathbb{S}^\sharp$ is finite) and $F^\sharp$ is monotone or extensive, then

$$\bigsqcup_{i \geq 0} {F^\sharp}^i(\bot)$$

is finitely computable and over-approximates $\mathsf{lfp} F$:

$$\mathsf{lfp} F \subseteq \gamma(\bigsqcup_{i \geq 0} {F^\sharp}^i(\bot)) \quad \textit{or equivalently} \quad \alpha(\mathsf{lfp} F) \sqsubseteq \bigsqcup_{i \geq 0} {F^\sharp}^i(\bot).$$

Soundness

- We need to show $F \circ \gamma \sqsubseteq \gamma \circ F^{\#}$ (or, equivalently $\alpha \circ F \sqsubseteq F^{\#} \circ \alpha$)
  - Then, the fixpoint transfer theorem would do.
- To show $F \circ \gamma \sqsubseteq \gamma \circ F^{\#}$ we need
  - sound condition for $\hookrightarrow^{\sharp}$:

$$\breve{\wp}(\hookrightarrow) \circ \gamma \ \subseteq \ \gamma \circ \breve{\wp}(\hookrightarrow^{\sharp})$$

  - sound condition for $\cup^{\sharp}$:

$$\cup \circ (\gamma, \gamma) \ \subseteq \ \gamma \circ \cup^{\sharp}$$

Soundness

Theorem (Soundness of $\hookrightarrow^\sharp$)

If the semantic operators satisfy the following soundness properties:

$$
\begin{array}{rcl}
\wp(eval_E) \circ \gamma_M & \subseteq & \gamma_V \circ eval^\sharp_E \\
\wp(update_x) \circ \times \circ (\gamma_M, \gamma_V) & \subseteq & \gamma_M \circ update^\sharp_x \\
\wp(filter_B) \circ \gamma_M & \subseteq & \gamma_M \circ filter^\sharp_B \\
\wp(filter_{\neg B}) \circ \gamma_M & \subseteq & \gamma_M \circ filter^\sharp_{\neg B}
\end{array}
$$

then $\breve{\wp}(\hookrightarrow) \circ \gamma \ \sqsubseteq \ \gamma \circ \breve{\wp}(\hookrightarrow^\sharp)$. *(The* $\times$ *is the Cartesian product operator of two sets.)*

Soundness (with Narrowing)

Theorem (Sound static analysis by $F^\sharp$ and widening operator $\bigtriangledown$)

Given a program, let F and $F^\sharp$ be defined as in the framework. Let $\bigtriangledown$ be a widening operator. Then the following chain $Y_0 \sqsubseteq Y_1 \sqsubseteq \cdots$

$$Y_0 = \bot \qquad Y_{i+1} = Y_i \bigtriangledown F^\sharp(Y_i)$$

is finite and its last element $Y_{\lim}$ over-approximates **lfp**F*:*

$$\textbf{lfp}F \subseteq \gamma(Y_{\lim}) \quad \textit{or equivalently} \quad \alpha(\textbf{lfp}F) \sqsubseteq Y_{\lim}.$$

Summary: Recipe for Designing Sound Static Analysis

1. Define $\mathbb{M}$ to be the set of memory states that can occur during program executions. Let $\mathbb{L}$ be the finite and fixed set of labels of a given program.
2. Define a concrete semantics as the **lfp**F where

$$
\begin{array}{rrcl}
\text{concrete domain} & \wp(\mathbb{S}) & = & \wp(\mathbb{L} \times \mathbb{M}) \\
\text{concrete semantic function} & F : \wp(\mathbb{S}) \to \wp(\mathbb{S}) & & \\
 & F(X) & = & I \cup \textit{Step}(X) \\
 & \textit{Step} & = & \breve{\wp}(\hookrightarrow) \\
 & \hookrightarrow & \subseteq & (\mathbb{L} \times \mathbb{M}) \times (\mathbb{L} \times \mathbb{M})
\end{array}
$$

The $\hookrightarrow$ is the one-step transition relation over $\mathbb{L} \times \mathbb{M}$.

Summary: Recipe for Designing Sound Static Analysis

4. Define its abstract domain and abstract semantic function as

$$
\begin{array}{rlcl}
\text{abstract domain} & \mathbb{S}^\sharp & = & \mathbb{L} \to \mathbb{M}^\sharp \\
\text{abstract semantic function} & F^\sharp : \mathbb{S}^\sharp \to \mathbb{S}^\sharp & & \\
& F^\sharp(X^\sharp) & = & \alpha(I) \cup^\sharp \mathit{Step}^\sharp(X^\sharp) \\
& \mathit{Step}^\sharp & = & \wp(\mathrm{id}, \sqcup_M) \circ \pi \circ \breve{\wp}(\hookrightarrow^\sharp) \\
& \hookrightarrow^\sharp & \subseteq & (\mathbb{L} \times \mathbb{M}^\sharp) \times (\mathbb{L} \times \mathbb{M}^\sharp)
\end{array}
$$

The $\hookrightarrow^\sharp$ is the one-step abstract transition relation over $\mathbb{L} \times \mathbb{M}^\sharp$. Function π partitions a set $\subseteq \mathbb{L} \times \mathbb{M}^\sharp$ by the labels in $\mathbb{L}$ returning an element in $\mathbb{L} \to \wp(\mathbb{M}^\sharp)$ represented as a set $\subseteq \mathbb{L} \times \wp(\mathbb{M}^\sharp)$.

Summary: Recipe for Designing Sound Static Analysis

5. Check the abstract domains $\mathbb{S}^\sharp$ and $\mathbb{M}^\sharp$ are CPOs, and forms a Galois-connection respectively with $\wp(\mathbb{S})$ and $\wp(\mathbb{M})$:

$$(\wp(\mathbb{S}), \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathbb{S}^\sharp, \sqsubseteq) \quad \text{and} \quad (\wp(\mathbb{M}), \subseteq) \underset{\alpha_M}{\overset{\gamma_M}{\leftrightarrows}} (\mathbb{M}^\sharp, \sqsubseteq_M)$$

   where the partial order $\sqsubseteq$ of $\mathbb{S}^\sharp$ is label-wise $\sqsubseteq_M$:

$$a^\sharp \sqsubseteq b^\sharp \quad \text{iff} \quad \forall l \in \mathbb{L} : a^\sharp(l) \sqsubseteq_M b^\sharp(l).$$

6. Check the abstract one-step transition $\hookrightarrow^\sharp$ and abstract union $\cup^\sharp$ satisfy:

$$\begin{aligned} \breve{\wp}(\hookrightarrow) \circ \gamma &\subseteq \gamma \circ \breve{\wp}(\hookrightarrow^\sharp) \\ \cup \circ (\gamma, \gamma) &\subseteq \gamma \circ \cup^\sharp \end{aligned}$$

Summary: Recipe for Designing Sound Static Analysis

7. Then, sound static analysis is defined as follows:
   - In case $\mathbb{S}^\sharp$ is of finite-height (every its chain is finite) and $F^\sharp$ is monotone or extensive, then

     $$\bigsqcup_{i \geq 0} {F^\sharp}^i(\bot)$$

     is finitely computable and over-approximates the concrete semantics **lfp**F.
   - Otherwise, find a widening operator $\bigtriangledown$, then the following chain $X_0 \sqsubseteq X_1 \sqsubseteq \cdots$

     $$X_0 = \bot \qquad X_{i+1} = X_i \bigtriangledown F^\sharp(X_i)$$

     is finite and its last element over-approximates the concrete semantics **lfp**F.