Data-Flow Analysis Implementation

Sergey Mechtaev
mechtaev@pku.edu.cn

Peking University

Reaching Definitions Analysis (Recap)

Reaching Definitions = Most common Data Flow schema

An assignment (aka definition) of the form $[x := a]^l$ **may reach** a certain program point if there is an execution of the program where x was last assigned a value at l when the program point is reached.

GOAL: identify all definitions reaching the *Entry* and *Exit* of each elementary block.

Example Program (Factorial)

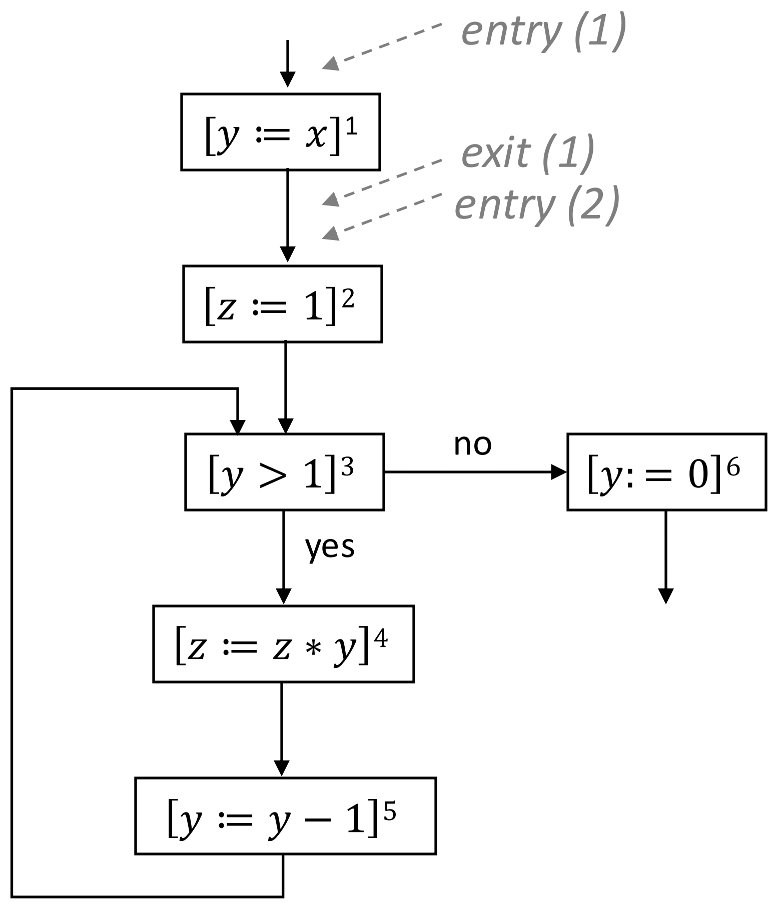
Computes factorial of number stored in variable x and stores result in variable z.

```
$[y := x]^1$;
$[z := 1]^2$;
$while\ [y > 1]^3\ do$
    $([z := z * y]^4$;
      $[y := y - 1]^5)$;
$[y := 0]^6$
```

The factorial of a non-negative integer n is the product of all positive integers less than or equal to n:

$$\begin{aligned} n! &= n * (n-1) * (n-2) * (n-3) * \cdots * 3 * 2 * 1 \\ &= n * (n-1)! \end{aligned}$$

Reaching Definition Analysis



NODES = elementary blocks

EDGES = describe how control might pass from one elementary block to another

A definition of a variable v at program point d **reaches** program point u (possibly using that variable) iff $\exists$ a control-flow path from d to u, where along that path there are no other assignments to that variable v.

Reaching Definition Analysis

```
$[y := x]^1;$
$[z := 1]^2;$
$while\ [y > 1]^3\ do$
    $([z := z * y]^4;$
      $[y := y - 1]^5);$
$[y := 0]^6$
```

$[y := x]^1$ is a **reaching definition** for $[z := 1]^2$

$\underbrace{[y := x]^1}_{(y,1)}$ *reaches the entry and exit of* $\underbrace{[z := 1]^2}_{2}$

$(x, \underset{*}{?})$ *reaches the entry* to 2

* "?" used to record uninitialised variables

Reaching Definition Analysis

$[y := x]^1;$
$[z := 1]^2;$
$while\ [y > 1]^3\ do$
$\quad ([z := z * y]^4;$
$\quad\ [y := y - 1]^5);$
$[y := 0]^6$

$[z := 1]^2$ is **NOT a reaching definition** for $[y := y - 1]^5$

$\underbrace{[z := 1]^2}_{(z,2)}$ *cannot reach the entry* to $\underbrace{[y := y - 1]^5}_{5}$

The value of z defined at 2 is no longer available when the program point (label 5) is reached

$(\underbrace{[z := z * y]^4}_{(z,4)}$ kills it (update it)

Expected Result

$[y := x]^1;$
$[z := 1]^2;$
$while\ [y > 1]^3\ do$
$\quad([z := z * y]^4;$
$\qquad[y := y - 1]^5);$
$[y := 0]^6$

l	$\boldsymbol{RD_{entry}(l)}$	$\boldsymbol{RD_{exit}(l)}$
1	$(x,?),(y,?),(z,?)$	$(x,?),(y,1),(z,?)$
2	$(x,?),(y,1),(z,?)$	$(x,?),(y,1),(z,2)$
3	$(x,?),(y,1),(y,5),(z,2),(z,4)$	$(x,?),(y,1),(y,5),(z,2),(z,4)$
4	$(x,?),(y,1),(y,5),(z,2),(z,4)$	$(x,?),(y,1),(y,5),(z,4)$
5	$(x,?),(y,1),(y,5),(z,4)$	$(x,?),(y,5),(z,4)$
6	$(x,?),(y,1),(y,5),(z,2),(z,4)$	$(x,?),(y,6),(z,2),(z,4)$

Expected Result

$[y := x]^1;$
$[z := 1]^2;$
$while\ [y > 1]^3\ do$
$\quad ([z := z * y]^4;$
$\qquad [y := y - 1]^5);$
$[y := 0]^6$

l	$\boldsymbol{RD_{entry}(l)}$	$\boldsymbol{RD_{exit}(l)}$
1	$(x,?),(y,?),(z,?)$	$(x,?),(y,1),(z,?)$
2	$(x,?),(y,1),(z,?)$	$(x,?),(y,1),(z,2)$
3	$(x,?),(y,1),(y,5),(z,2),(z,4)$	$(x,?),(y,1),(y,5),(z,2),(z,4)$
4	$(x,?),(y,1),(\mathbf{y},\mathbf{5})^*,(z,2),(\mathbf{z},\mathbf{4})^*$	$(x,?),(y,1),(y,5),(z,4)$
5	$(x,?),(y,1),(\mathbf{y},\mathbf{5})^*,(z,4)$	$(x,?),(y,5),(z,4)$
6	$(x,?),(y,1),(y,5),(z,2),(z,4)$	$(x,?),(y,6),(z,2),(z,4)$

* Assuming you are entering in the *While* loop for the second time.

Data Flow Analysis

$(z, 2)$ no longer available
(because updated)

$[y := x]^1;$
$[z := 1]^2;$
$while\ [y > 1]^3\ do$
$\quad ([z := z * y]^4;$
$\quad\ [y := y - 1]^5);$
$[y := 0]^6$

l	$\boldsymbol{RD_{entry}(l)}$	$\boldsymbol{RD_{exit}(l)}$
1	$(x, ?), (y, ?), (z, ?)$	$(x, ?), (y, 1), (z, ?)$
2	$(x, ?), (y, 1), (z, ?)$	$(x, ?), (y, 1), (z, 2)$
3	$(x, ?), (y, 1), (y, 5), (z, 2), (z, 4)$	$(x, ?), (y, 1), (y, 5), (z, 2), (z, 4)$
4	$(x, ?), (y, 1), (y, 5), (z, 2), (z, 4)$	$(x, ?), (y, 1), (y, 5), (z, 4)$
5	$(x, ?), (y, 1), (y, 5), (z, 4)$	$(x, ?), (y, 5), (z, 4)$
6	$(x, ?), (y, 1), (y, 5), (z, 2), (z, 4)$	$(x, ?), (y, 6), (z, 2), (z, 4)$

Data Flow Analysis Equations (Recap)

Equations that capture how information is updated by each node:

$$RD_{exit}(1) = (RD_{entry}(1) \setminus \{(y,l) \mid l \in Lab\} \cup \{(y,1)\}$$
$$RD_{exit}(2) = (RD_{entry}(2) \setminus \{(z,l) \mid l \in Lab\} \cup \{(z,2)\}$$
$$RD_{exit}(3) = RD_{entry}(3)$$
$$RD_{exit}(4) = (RD_{entry}(4) \setminus \{(z,l) \mid l \in Lab\} \cup \{(z,4)\}$$
$$RD_{exit}(5) = (RD_{entry}(5) \setminus \{(y,l) \mid l \in Lab\} \cup \{(y,5)\}$$
$$RD_{exit}(6) = (RD_{entry}(6) \setminus \{(y,l) \mid l \in Lab\} \cup \{(y,6)\}$$

Equations that capture how information is propagated through CFG:

$$RD_{entry}(1) = \{(x,?),(y,?),(z,?)\}$$
$$RD_{entry}(2) = RD_{exit}(1)$$
$$RD_{entry}(3) = RD_{exit}(2) \cup RD_{exit}(5)$$
$$RD_{entry}(4) = RD_{exit}(3)$$
$$RD_{entry}(5) = RD_{exit}(4)$$
$$RD_{entry}(6) = RD_{exit}(3)$$

Data Flow Analysis Equations

- The resulting **system of equations** defines the twelve sets:

$$RD_{entry}(1), \dots, RD_{exit}(6) = \overrightarrow{RD} = RD_1, \dots\dots, RD_{12}$$

- Therefore, $\overrightarrow{RD}$ is a **tuple of 12 sets** of pairs of variables (v) and labels (l):

$$\{\underbrace{((v,l),(v,l),\dots,(v,l))}_{RD_1},\underbrace{((v,l),(v,l),\dots,(v,l))}_{RD_2},\dots\dots,\underbrace{((v,l),(v,l),\dots,(v,l))}_{RD_{12}}\}$$

- Can be represented as defining a **function**:

$$\overrightarrow{RD} = \boldsymbol{F}(\overrightarrow{RD})$$

Data Flow Analysis Equations

- $\boldsymbol{F}\!: (\underbrace{P(\mathrm{Var}_* \times \mathrm{Lab}_*)}_{})^{12} \rightarrow (P(\mathrm{Var}_* \times \mathrm{Lab}_*))$

  Power set of all possible couples $(v, l)\ \forall v \in Var_* \wedge l \in Lab_*$

  $\{(\emptyset, (x,1), \ldots, (x,6), (y,1), .., (y,6), (z,1), \ldots, (z,6), ((x,1)(x,2)), ((x,1)(x,2)(x,3)), \ldots, \xi)\}$

- $(P(\mathrm{Var}_* \times \mathrm{Lab}_*))^{12}$ can be treated as **partially ordered set**

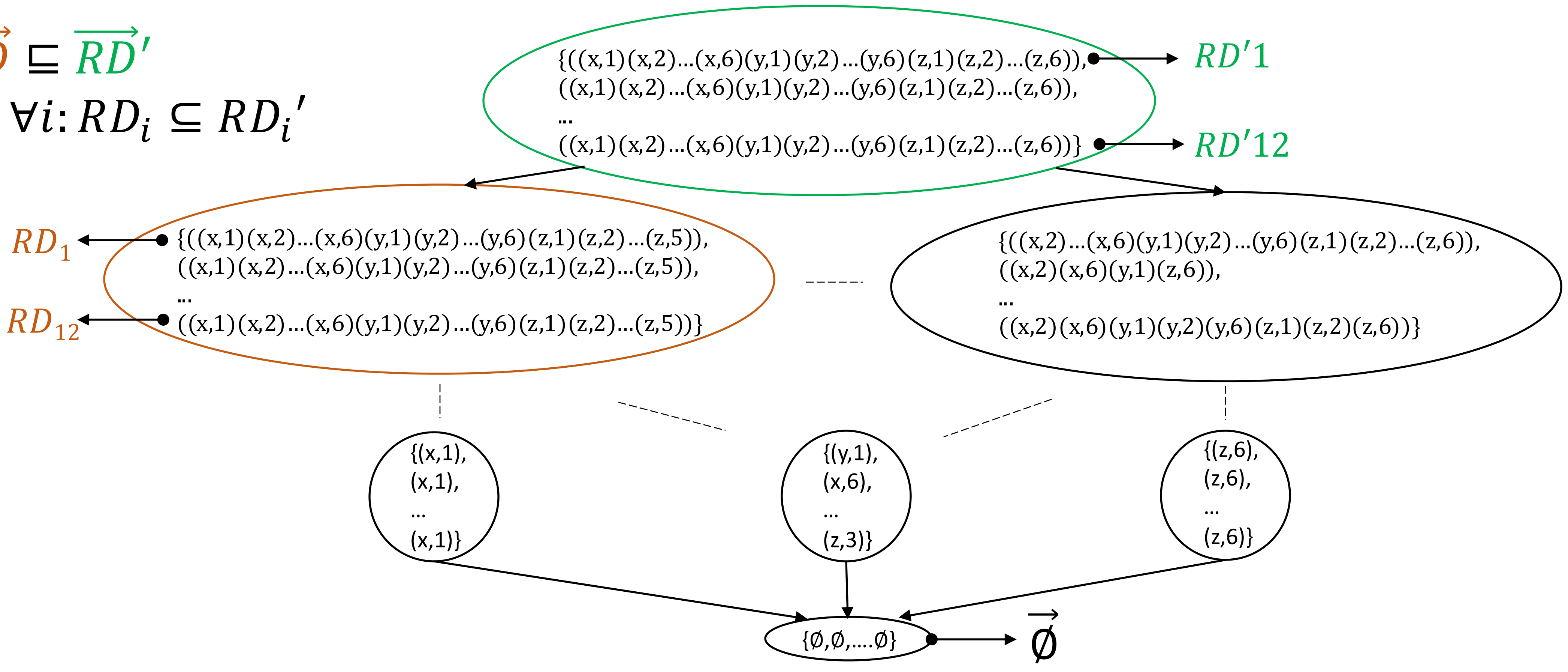
- Let define the ordering $\overrightarrow{RD} \sqsubseteq \overrightarrow{RD}'$ iff $\forall i: RD_i \subseteq RD_i{}'$

Data Flow Analysis: *The Least Solution*

- **Partially ordered set**

$\overrightarrow{RD} \sqsubseteq \overrightarrow{RD}'$

iff $\forall i: RD_i \subseteq RD_i'$


{((x,1)(x,2)…(x,6)(y,1)(y,2)…(y,6)(z,1)(z,2)…(z,6)),
((x,1)(x,2)…(x,6)(y,1)(y,2)…(y,6)(z,1)(z,2)…(z,6)),
…
((x,1)(x,2)…(x,6)(y,1)(y,2)…(y,6)(z,1)(z,2)…(z,6))}
RD'1
RD'12
RD1
RD12
{((x,1)(x,2)…(x,6)(y,1)(y,2)…(y,6)(z,1)(z,2)…(z,5)),
((x,1)(x,2)…(x,6)(y,1)(y,2)…(y,6)(z,1)(z,2)…(z,5)),
…
((x,1)(x,2)…(x,6)(y,1)(y,2)…(y,6)(z,1)(z,2)…(z,5))}
{((x,2)…(x,6)(y,1)(y,2)…(y,6)(z,1)(z,2)…(z,6)),
((x,2)(x,6)(y,1)(z,6)),
…
((x,2)(x,6)(y,1)(y,2)(y,6)(z,1)(z,2)(z,6))}
{(x,1), (x,1), … (x,1)}
{(y,1), (x,6), … (z,3)}
{(z,6), (z,6), … (z,6)}
{∅,∅,….∅}
$\vec{\emptyset}$

Theorem

A **fixed point** of a function F is a value x such that $x = F(x)$.

Theorem. Let S be a finite set $\{x_1, x_2, x_3, \ldots, x_n\}$, L be the powerset of S, that is $P(S)$. Let $F: L \rightarrow L$ be a monotone function, that is $l \sqsubseteq l'$ implies that $F(l) \sqsubseteq F(l')$. Then,

- there exists such n that $F^{n+1}(\emptyset) = F^n\ (\emptyset)$
- $F^n(\emptyset)$ is the least fixed point of F

This theorem enables a simple algorithm to solve data-flow equations: finding the least solution by iteratively applying F until it stabilises:

- $F^n(\vec{\emptyset})$ is the **least solution** to the equation system
- Other solutions are less precise

Partial Ordering

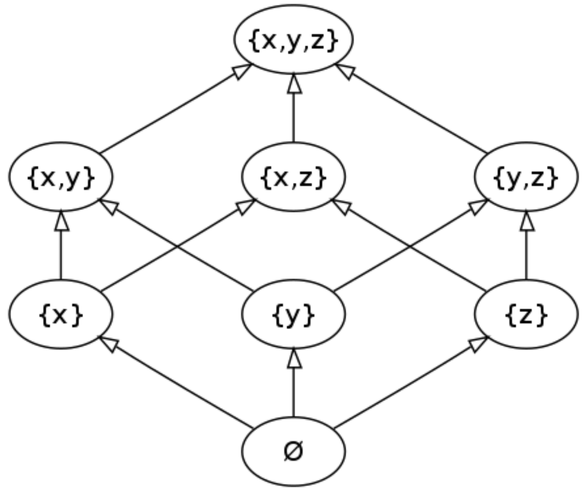
A **partial ordering** is a relation $\sqsubseteq: L \times L \to \{true, false\}$ that is

- reflexive ($\forall l: l \sqsubseteq l$)
- transitive ($\forall l_1, l_2, l_3: l_1 \sqsubseteq l_2 \wedge l_2 \sqsubseteq l_3 \Rightarrow l_1 \sqsubseteq l_3$)
- anti-symmetric ($\forall l_1, l_2: l_1 \sqsubseteq l_2 \wedge l_2 \sqsubseteq l_1 \Rightarrow l_1 = l_2$).

A **partially ordered set** $(L, \sqsubseteq)$ is a set L equipped with a partial ordering $\sqsubseteq$.

$$l_1 \sqsubset l_2 \coloneqq l_1 \sqsubseteq l_2 \wedge l_1 \neq l_2$$

Partial Order Example



- $L \coloneqq 2^{\mathbb{Z}}$ – subsets of whole numbers
- $\sqsubseteq \coloneqq \subseteq$ – subset relation

- Reflexivity, transitivity and anti-symmetry are implied by the definition of $\subseteq$
- This relation is a *partial order* and not a *total order*, because some elements are incomparable
  - For example, neither $\{1,2\} \subseteq \{2,3\}$ nor $\{2,3\} \subseteq \{1,2\}$

Upper Bounds

A subset Y of L has

- an **upper bound** $l \in L$ if $\forall l' \in Y: l' \sqsubseteq l$;
- a **lower bound** $l \in L$ if $\forall l' \in Y: l \sqsubseteq l'$.

The **least upper bound** l of Y (or $\bigsqcup Y$) satisfies $l \sqsubseteq l_0$ whenever l_0 is another upper bound of Y. Wh

The **greatest lower bound** l of Y (or $\bigsqcap Y$) satisfies $l_0 \sqsubseteq l$ whenever l_0 is another lower bound of Y.

$\bigsqcup$ is the **join operator**: $\mathrm{l}_1 \sqcup l_2 \coloneqq \bigsqcup\{l_1, l_2\}$;
$\bigsqcap$ is the **meet operator**: $l_1 \sqcap l_2 \coloneqq \bigsqcap\{l_1, l_2\}$.

Upper and Lower Bounds: *Example*

$Y = \{\{x,y\},\{x,z\},\{y,z\},\{y\}\}$

$\sqcup Y = \{x,y,z\}$

$Y = \{\{x,y\},\{y,z\}\}$

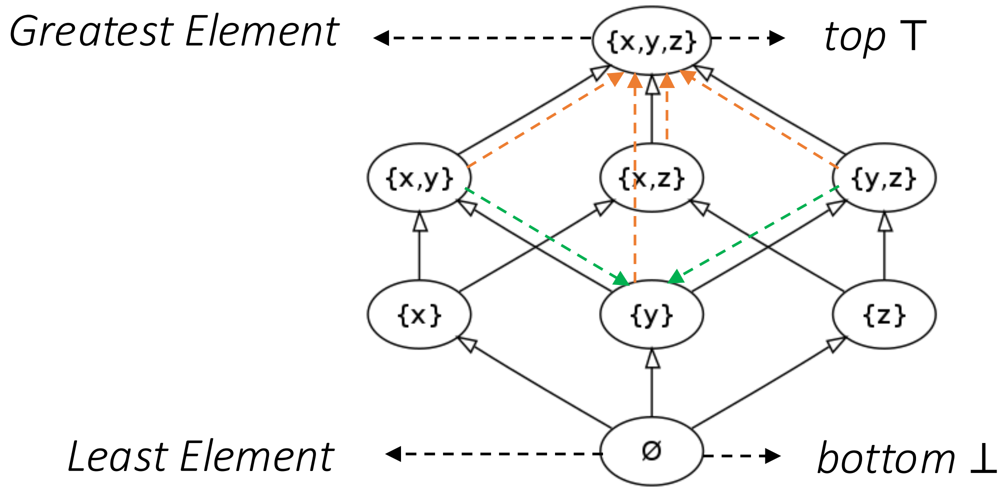
$\sqcap Y = \{y\}$

$\sqcup \{\,\} = \{\emptyset\}$

$\sqcap \{\,\} = \{x,y,z\}$

$\sqcup L = \{x,y,z\}$

$\sqcap L = \{\emptyset\}$



Complete Lattice

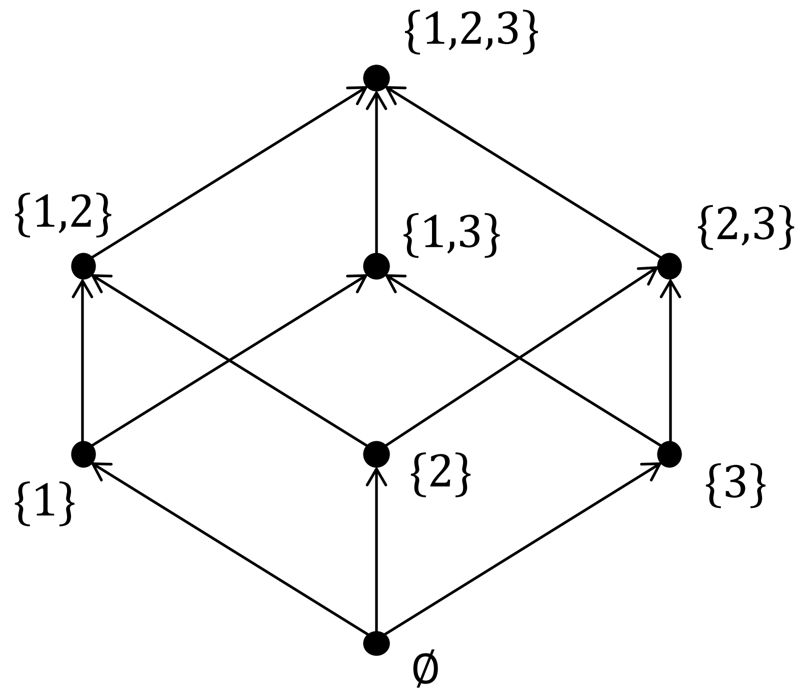
A **complete lattice** $L = (L, \sqsubseteq)$ is a partially ordered set such that all subsets have least upper bounds as well as greatest lower bounds. Furthermore, $\bot = \bigsqcup \emptyset = \bigsqcap L$ is the least element, $\top = \bigsqcap \emptyset = \bigsqcup L$ is the greatest element.

Complete Lattice Example

- $L := 2^{\{1,2,3\}}$ – subsets of the set of three numbers
- $\sqsubseteq \; := \; \subseteq$ – subset relation

Arrow between l_1 and l_2 means $l_1 \sqsubseteq l_2$

$\{1,2,3\}$ is the greatest element
$\emptyset$ is the least element



Chains

A subset $Y \subseteq L$ of a partially ordered set $(L, \sqsubseteq)$ is a **chain** if $\forall l_1, l_2 \in Y: (l_1 \sqsubseteq l_2) \vee (l_2 \sqsubseteq l_1)$. Thus, a chain is a totally ordered subset of L.

A sequence $(l_n)_{n\in\mathrm{N}}$ is an ascending chain if $n \leq m \Rightarrow l_n \sqsubseteq l_m$

A sequence $(l_n)_{n\in\mathrm{N}}$ **eventually stabilises** iff

$$\exists n_0 \in N: \forall n \in: n \geq n_0 \Rightarrow l_n = l_{n_0}$$

A partially ordered set $(L, \sqsubseteq)$ satisfies **ascending chain condition** iff all ascending chains eventually stabilise.

Monotone Framework Definition

- L is the property space
- $\bigsqcup : 2^L \rightarrow L$ is the combination operator
- $f_\ell : L \rightarrow L$ is the transfer function
  - A function f_ℓ is monotone iff $l \sqsubseteq l'$ implies $f_\ell(l) \sqsubseteq f_\ell(l')$
- A **monotone framework** is
  - L is a complete lattice that satisfies the ascending chain condition
  - A set $\mathcal{F}$ of monotone functions from L to L that contains the identity function and that is closed under function composition

Instances Of Monotone Frameworks

An **instance** of a monotone framework

- The complete lattice, L
- The space of functions, $\mathcal{F}$
- A finite flow, F, that typically is $flow(program)$ or $flow^R(program)$
- A finite set of extremal labels, E, that it typically $\{init(program)\}$ or $final(program)$
- An extremal value, $\iota \in L$, for the extremal labels
- A mapping f. From labels to transfer functions $\mathcal{F}$

Analysis Equations

$$Analysis_{\Diamond}(\ell) = \bigsqcup\{Analysis_{\blacklozenge}(\ell') \mid (\ell', \ell) \in F\} \sqcup \iota_E^{\ell}$$

where $\iota_E^{\ell} = \begin{cases} \iota & if\ \ell \in E \\ \bot & if\ \ell \notin E \end{cases}$

$$Analysis_{\blacklozenge}(\ell) = f_{\ell}(Analysis_{\Diamond}(\ell))$$

◆ The output of each basic block is computed by applying the transfer function to the analysis (for example at the beginning of the block).

◇ The analysis results for each edge of the control flow are then combined with some initial information in case of extremal elements.

Forward/Backward, Must/May

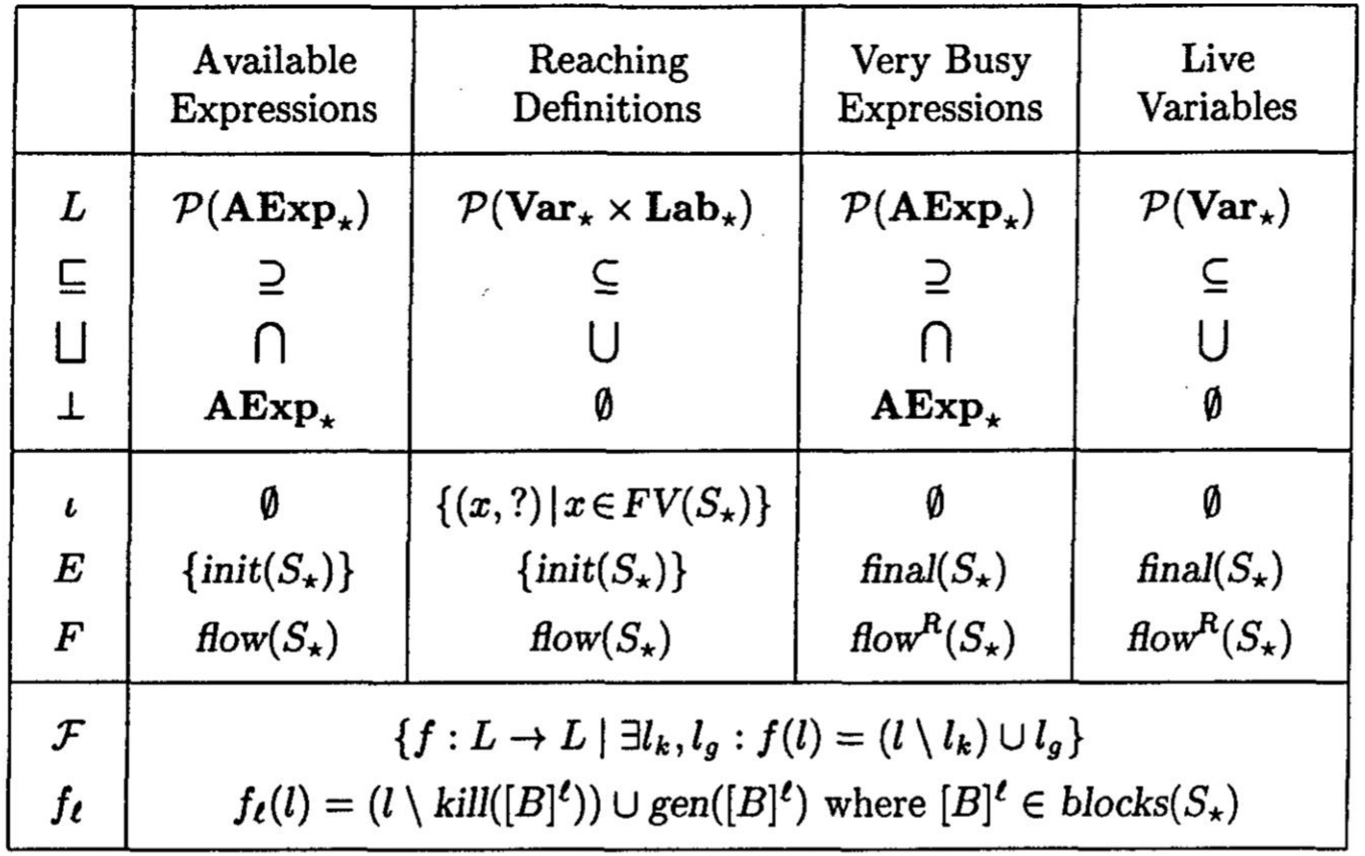
- **Forward**: F is $flow(program)$, $Analysis_{\diamond}$ concerns entry conditions, $Analysis_{\blacklozenge}$ concerns exit conditions
- **Backward**: F is $flow^{R}(program)$, $Analysis_{\diamond}$ concerns exit conditions, $Analysis_{\blacklozenge}$ concerns entry conditions
- **Must**: $\sqcup$ is $\cap$, search for the greatest solution
- **May**: $\sqcup$ is $\cup$, search for the least solution

Transfer Functions in Classical Analyses

- $\mathcal{F} \coloneqq \{\mathrm{f}: \mathrm{L} \rightarrow \mathrm{L} \mid \exists \mathrm{l_k}, \mathrm{l_g}: \mathrm{f}(\mathrm{l}) = (\mathrm{l} \backslash \mathrm{l_k}) \cup l_g\}$
    - Includes the identity function ($\mathrm{l_k}, = \mathrm{l_g} = \emptyset$)
    - Monotone
- $f_\ell(l) \coloneqq \left(\mathrm{l} \backslash \mathrm{kill}\left([\mathrm{B}]^\ell\right)\right) \cup gen([B]^\ell)$,
  where $[B]^\ell \in blocks(program)$

Instances of Classical Analyses

Each of the four Classical Analyses satisfies the monotone framework:



	Available Expressions	Reaching Definitions	Very Busy Expressions	Live Variables
L	$\mathcal{P}(\mathbf{AExp}_\star)$	$\mathcal{P}(\mathbf{Var}_\star \times \mathbf{Lab}_\star)$	$\mathcal{P}(\mathbf{AExp}_\star)$	$\mathcal{P}(\mathbf{Var}_\star)$
$\sqsubseteq$	$\supseteq$	$\subseteq$	$\supseteq$	$\subseteq$
$\bigsqcup$	$\bigcap$	$\bigcup$	$\bigcap$	$\bigcup$
$\perp$	$\mathbf{AExp}_\star$	$\emptyset$	$\mathbf{AExp}_\star$	$\emptyset$
ι	$\emptyset$	$\{(x,?) \mid x \in FV(S_\star)\}$	$\emptyset$	$\emptyset$
E	$\{init(S_\star)\}$	$\{init(S_\star)\}$	$final(S_\star)$	$final(S_\star)$
F	$flow(S_\star)$	$flow(S_\star)$	$flow^R(S_\star)$	$flow^R(S_\star)$
$\mathcal{F}$	$\{f : L \to L \mid \exists l_k, l_g : f(l) = (l \setminus l_k) \cup l_g\}$			
f_ℓ	$f_\ell(l) = (l \setminus kill([B]^\ell)) \cup gen([B]^\ell)$ where $[B]^\ell \in blocks(S_\star)$			

Available Expressions

- $L \coloneqq 2^{AExp(program)}$ – subsets of program expressions
- $\sqsubseteq \coloneqq \supseteq$
- $\sqcup \coloneqq \cap$
- $\bot \coloneqq AExp(program)$
- $i \coloneqq \emptyset$
- $E \coloneqq \{init(program)\}$
- $F \coloneqq flow(program)$

Reaching Definitions

- $L \coloneqq 2^{Vars(program) \times Labels(program)}$ – sets of pairs $(variable, program)$
- $\sqsubseteq \coloneqq \subseteq$
- $\sqcup \coloneqq \cup$
- $\bot \coloneqq \emptyset$
- $i \coloneqq \{(x, ?)|x \in Vars(program)\}$
- $E \coloneqq \{init(program)\}$
- $F \coloneqq flow(program)$

Very Busy Expressions

- $L \coloneqq 2^{AExp(program)}$ – subsets of program expressions
- $\sqsubseteq \coloneqq \supseteq$
- $\sqcup \coloneqq \cap$
- $\perp \coloneqq AExp(program)$
- $i \coloneqq \emptyset$
- $E \coloneqq final(program)$
- $F \ \coloneqq flow^R(program)$

Live Variables

- $L \coloneqq 2^{Vars(program)}$ – subsets of program variables
- $\sqsubseteq \coloneqq \subseteq$
- $\sqcup \coloneqq \cup$
- $\bot \coloneqq \emptyset$
- $i \coloneqq \emptyset$
- $E \coloneqq init(program)$
- $F \coloneqq flow^R(program)$

Property of Classical Analyses

Theorem. The four classical analyses are monotone frameworks.

Proof (sketch):

1. The functions in $\mathcal{F}$ are monotone, because
   - Easy to show that $(l \setminus l_k) \cup l\_g$ is monotone for both $\subseteq$ and $\supseteq$.
2. $\mathcal{F}$ has the identify function
3. Easy to show that $\mathcal{F}$ is closed under composition

Instance of Monotone Framework: *Example*

Available Expressions Analysis:

```
$[x := a + b]^1;$
$[y := a * b]^2;$
$while\ [y > a + b]^3\ do$
    $([a := a + 1]4;$
     $[x := a + b]^5;)$
```

Instance of Monotone Framework: *Example*

Available Expressions Analysis:

```
$[x := a + b]^1;$
$[y := a * b]^2;$
$while\ [y > a + b]^3\ do$
      $([a: = a + 1]4;$
        $[x: = a + b]^5; )$
```

$\longrightarrow$

Complete lattice:

$(P(\{a + b, a * b, a + 1\}), \supseteq)$

Least Element:

$\{a + b, a * b, a + 1\}$

Instance of Monotone Framework: *Example*

Available Expressions Analysis:

$[x := a + b]^1;$
$[y := a * b]^2;$
$while\ [y > a + b]^3\ do$
$\quad([a := a + 1]4;$
$\quad\ [x := a + b]^5;)$

$\longrightarrow$

Forward information flow:

$\{(1,2),(2,3),(3,4),(4,5),(5,3)\}$

Instance of Monotone Framework: *Example*

Available Expressions Analysis:

$[x := a + b]^1;$
$[y := a * b]^2;$
$while\ [y > a + b]^3\ do$
$\quad ([a := a + 1]4;$
$\quad\ [x := a + b]^5;)$

$\longrightarrow$

Extremal label:
$\{1\}$

Extremal value:
0

Instance of Monotone Framework: *Example*

Available Expressions Analysis:

$[x := a + b]^1;$
$[y := a * b]^2;$
$while\ [y > a + b]^3\ do$
$\quad ([a := a + 1]4;$
$\quad\ [x := a + b]^5;)$

$\longrightarrow$

Transfer functions:

$$f^{AE}{}_1(Y) = Y \cup \{a + b\}$$
$$f^{AE}{}_2(Y) = Y \cup \{a * b\}$$
$$f^{AE}{}_3(Y) = Y \cup \{a + b\}$$
$$f^{AE}{}_4(Y) = Y \setminus \{a + b, a * b, a + 1\}$$
$$f^{AE}{}_5(Y) = Y \cup \{a + b\}$$

for $Y \subseteq \{a + b, a * b, a + 1\}$

Maximal Fixed Point (MFP)

Computes the *least solution* to the data flow equation: *fixed point* of some function.

*A **fixed point** of a function is an element that is mapped to itself by the function. That is, c is a fixed point of a function f if c belongs to both the domain and the codomain of f, $\boldsymbol{f(c) = c}$.* [Wikipedia]

Maximal Fixed Point (MFP)

- W is the worklist of edges
- $w \in W$ means that the result of analysis has changed at the exit of w
- $Analysis$ is an array containing the analysis results (labels are indexes)

MFP Algorithm

- Starts from initial results (*e.g.* an initial value) and iteratively applies transfer functions until a fixed point is reached.
- Computes 2 functions: $MFP_\diamond$ and $MFP_\blacklozenge$ (results of the analysis).

$$\downarrow$$

- **INPUT**: An instance $(L, \mathcal{F}, F, E, \iota, f)$ of a Monotone Framework
- **OUTPUT**: $MFP_\diamond$ and $MFP_\blacklozenge$

Example

```
$[x \coloneqq a + b]^1;$
$[y \coloneqq a * b]^2;$
$while\ [y > a + b]^3\ do$
    $[a \coloneqq a + 1]^4;$
    $[x \coloneqq a + b]^5;$
```

Available Expressions Analysis

Lattice: $(2^{\{a+b, a*b, a+1\}}, \supseteq)$

Transfer functions:

$$f_\ell(l) \coloneqq \left(l \setminus kill_{AE}\left([B]^\ell\right)\right) \cup gen_{AE}\left([B]^\ell\right)$$

MFP Algorithm

STEP 1 - INITIALISATION (of W and $Analysis$)

```
$W := nil,$
$for\ (\ell, \ell') \in F\colon\ W := cons\big((\ell, \ell'), W\big),$
$for\ \ell \in F \cup E\colon\ \ if\ l \in E\ then\ Analysis[\ell] :=\ \iota$
                                   $else\ Analysis[\ell] = \bot$
```

STEP 2 - ITERATION (updating W and $Analysis$)

```
$while\ W \neq nil$
    $(\ell, \ell') := head(W),$
    $W := tail(W),$
    $if\ f_\ell(Analysis[\ell]) \not\sqsubseteq Analysis[\ell']\ then$
        $Analysis[\ell'] := Analysis[\ell'] \sqcup f_\ell(Analysis[\ell]),$
        $for\ \ell''such\ that\ (\ell', \ell'') \in F\colon\ W := cons((\ell', \ell''), W)$
```

STEP 3 - PRESENTING THE RESULT ($MFP_\diamond$ and $MFP_\blacklozenge$)

```
$for\ \ell \in F \cup \mathrm{E}\colon\ \ MFP_\diamond(\ell) := Analysis[\ell],$
                   $MFP_\blacklozenge(\ell) := f_\ell(Analysis[\ell])$
```

MFP Algorithm

STEP 1 - INITIALISATION (of W and $Analysis$) $\longrightarrow$

```
$W := nil,$
$for\ (\ell, \ell') \in F:\ W := cons((\ell, \ell'), W),$
$for\ \ell \in F \cup E:\ \ if\ l \in E\ then\ Analysis[\ell] := \iota$
                                  $else\ Analysis[\ell] = \bot$
```

STEP 2 - ITERATION (updating W and $Analysis$)

```
$while\ W \neq nil$
    $(\ell, \ell') := head(W),$
    $W := tail(W),$
    $if\ f_\ell(Analysis[\ell]) \not\sqsubseteq Analysis[\ell']\ then$
        $Analysis[\ell'] := Analysis[\ell'] \sqcup f_\ell(Analysis[\ell]),$
        $for\ \ell''\, such\ that\ (\ell', \ell'') \in F:\ W := cons((\ell', \ell''), W)$
```

STEP 3 - PRESENTING THE RESULT ($MFP_\diamond$ and $MFP_\blacklozenge$)

```
$for\ \ell \in F \cup$ E:   $MFP_\diamond(\ell) := Analysis[\ell],$
                   $MFP_\blacklozenge(\ell) := f_\ell(Analysis[\ell])$
```

- The *worklist is initialised* by adding all the edges (from the CFG) to be analysed.
- For each label (node), the analysis array is initialised. It will store the result of the *e.g.* Available Expressions for that label, that will be computed from the initial results using the result for the extreme labels $\iota \in L$ (*e.g.* empty).

MFP Algorithm

STEP 1 - INITIALISATION (of W and $Analysis$)

```
$W := nil,$
$for\ (\ell,\ell') \in F:\ W := cons((\ell,\ell'), W),$
$for\ \ell \in F \cup E:\ \ if\ l \in E\ then\ Analysis[\ell] := \iota$
                          $else\ Analysis[\ell] = \bot$
```

STEP 2 - ITERATION (updating W and $Analysis$) $\longrightarrow$

```
$while\ W \neq nil$
    $(\ell,\ell') := head(W),$
    $W := tail(W),$
    $if\ f_\ell(Analysis[\ell]) \not\sqsubseteq Analysis[\ell']\ then$
        $Analysis[\ell'] := Analysis[\ell'] \sqcup f_\ell(Analysis[\ell]),$
        $for\ \ell''such\ that\ (\ell',\ell'') \in F:\ W := cons((\ell',\ell''), W)$
```

STEP 3 - PRESENTING THE RESULT ($MFP_\diamond$ and $MFP_\blacklozenge$)

```
$for\ \ell \in F \cup E:\ \ MFP_\diamond(\ell) := Analysis[\ell],$
                   $MFP_\blacklozenge(\ell) := f_\ell(Analysis[\ell])$
```

- The *while loop is executed* until the worklist of edges has at least one edge (*i.e.* until a fixed point is reached).
- For each iteration, one of the edges is analysed:
  - select the one at the top of the worklist
  - take the analysis results' array in the beginning of the edge
  - try to propagate information along this edge (applying recursively the transfer functions)
  - check whether there has been any update (something new learnt, *i.e.* the information in the analysis result for the outgoing edge is different because it includes the new result)

. . . . .

MFP Algorithm

STEP 1 - INITIALISATION (of W and $Analysis$)

```
$W := nil,$
$for\ (\ell,\ell') \in F:\ W := cons((\ell,\ell'), W),$
$for\ \ell \in F \cup E:\ \ if\ l \in E\ then\ Analysis[\ell] := \iota$
                                   $else\ Analysis[\ell] = \bot$
```

STEP 2 - ITERATION (updating W and $Analysis$) $\longrightarrow$

```
$while\ W \neq nil$
    $(\ell,\ell') := head(W),$
    $W := tail(W),$
    $if\ f_\ell(Analysis[\ell]) \not\sqsubseteq Analysis[\ell']\ then$
        $Analysis[\ell'] := Analysis[\ell'] \sqcup f_\ell(Analysis[\ell]),$
        $for\ \ell''\ such\ that\ (\ell',\ell'') \in F:\ W := cons((\ell',\ell''), W)$
```

STEP 3 - PRESENTING THE RESULT ($MFP_\diamond$ and $MFP_\blacklozenge$)

```
$for\ \ell \in F \cup E:\ \ MFP_\diamond(\ell) := Analysis[\ell],$
                   $MFP_\blacklozenge(\ell) := f_\ell(Analysis[\ell])$
```

. . . . .

- If TRUE:

  the information at the end of the edge needs to be updated with the new propagated results.

  Make sure that, at the next step, this new information is going to be propagated:

  - take all the edges starting from the node where the information was updated and add them to the worklist (so that all subsequent edges are also updated).

- If FALSE:

  No propagated information.
  No need to consider this edge again.
  The sequence stabilises:

  That's why *Ascending Chain Condition* is needed

MFP Algorithm

STEP 1 - INITIALISATION (of W and $Analysis$)

```
$W := nil,$
$for\ (\ell,\ell') \in F:\ W := cons((\ell,\ell'),W),$
$for\ \ell \in F \cup E:\ \ if\ l \in E\ then\ Analysis[\ell] := \iota$
                              $else\ Analysis[\ell] = \bot$
```

STEP 2 - ITERATION (updating W and $Analysis$) $\rightarrow$

```
$while\ W \neq nil$
    $(\ell,\ell') := head(W),$
    $W := tail(W),$
    $if\ f_\ell(Analysis[\ell]) \not\sqsubseteq Analysis[\ell']\ then$
        $Analysis[\ell'] := Analysis[\ell'] \sqcup f_\ell(Analysis[\ell]),$
        $for\ \ell''\ such\ that\ (\ell',\ell'') \in F:\ W := cons((\ell',\ell''),W)$
```

STEP 3 - PRESENTING THE RESULT ($MFP_\circ$ and $MFP_\bullet$)

```
$for\ \ell \in F \cup E:\ \ MFP_\circ(\ell) := Analysis[\ell],$
                     $MFP_\bullet(\ell) := f_\ell(Analysis[\ell])$
```

The algorithm produces a chain that, given the *AC* property of the framework, "eventually" stabilises. Therefore the algorithm is going to terminate.

$\uparrow$

That's why *Ascending Chain Condition* is needed

MFP Algorithm

STEP 1 - INITIALISATION (of W and $Analysis$)

```
$W := nil,$
$for\ (\ell,\ell') \in F:\ W := cons((\ell,\ell'), W),$
$for\ \ell \in F \cup E:\ \ if\ l \in E\ then\ Analysis[\ell] := \iota$
$                          else\ Analysis[\ell] = \bot$
```

STEP 2 - ITERATION (updating W and $Analysis$)

```
$while\ W \neq nil$
    $(\ell,\ell') := head(W),$
    $W := tail(W),$
    $if\ f_\ell(Analysis[\ell]) \not\sqsubseteq Analysis[\ell']\ then$
        $Analysis[\ell'] := Analysis[\ell'] \sqcup f_\ell(Analysis[\ell]),$
        $for\ \ell''such\ that\ (\ell',\ell'') \in F:\ W := cons((\ell',\ell''), W)$
```

STEP 3 - PRESENTING THE RESULT ($MFP_\diamond$ and $MFP_\blacklozenge$) $\rightarrow$

```
$for\ \ell \in F \cup \mathrm{E}:\ \ MFP_\diamond(\ell) := Analysis[\ell],$
                 $MFP_\blacklozenge(\ell) := f_\ell(Analysis[\ell])$
```

- After the *while loop terminates*:
  - return the analysis results
    (*functions* $MFP_\diamond$ *and* $MFP_\blacklozenge$)
    stored at the exit of each label, to which the transfer functions were applied.

MFP: *Example*

$[x := a + b]^1;$
$[y := a * b]^2;$
$while\ [y > a + b]^3\ do$
$\quad [a := a + 1]^4;$
$\quad [x := a + b]^5;$

INPUT: **Available Expressions Analysis**

instance of the monotone framework, as it creates:

Lattice $\boldsymbol{L}$: $(2^{\{a+b, a*b, a+1\}}, \supseteq)$

→ **finite:** it satisfies the *Ascending Chain Condition*

Transfer functions:

$$f_\ell(l) := \left(l \setminus kill_{AE}([B]^\ell)\right) \cup gen_{AE}([B]^\ell)$$

→ **monotone:** constant sets combined preserving all the conditions of the monotone framework

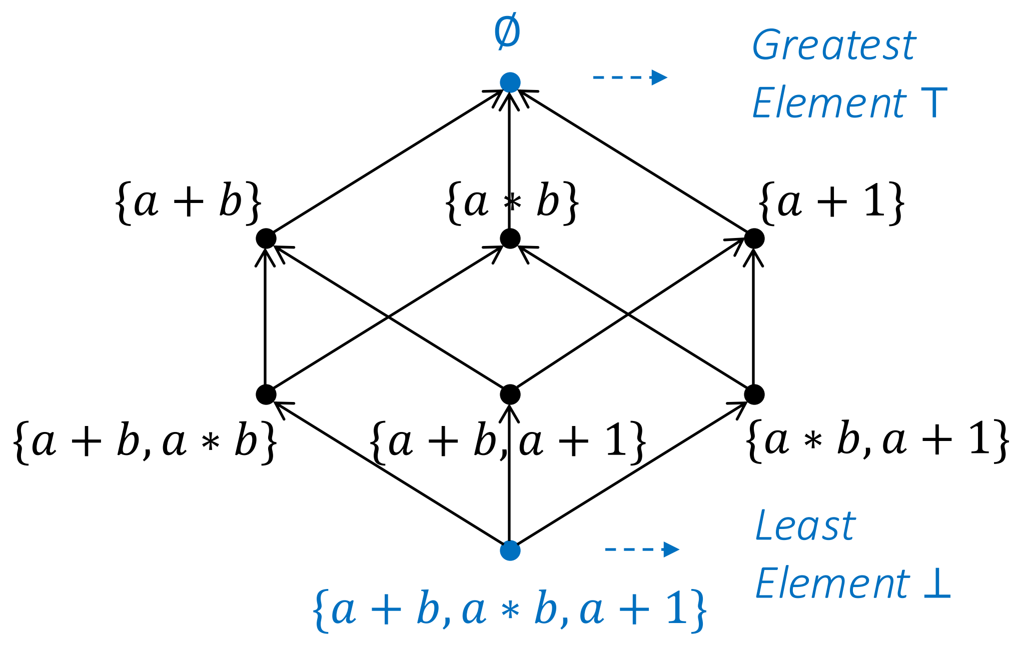
MFP: *Example*

```
[$x := a + b$]$^1$;
[$y := a * b$]$^2$;
while [$y > a + b$]$^3$ do
    [$a := a + 1$]$^4$;
    [$x := a + b$]$^5$;
```

INPUT: **Available Expressions Analysis**

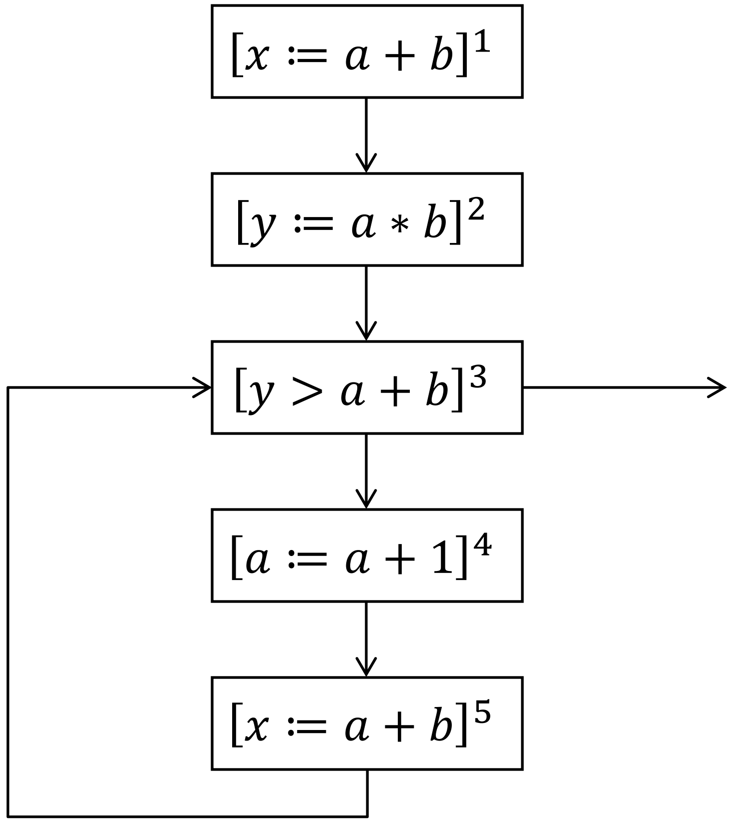
instance of the monotone framework, as it creates:

Lattice L: $(2^{\{a+b,a*b,a+1\}}, \supseteq)$



Step 1 (Initialisation)

$W: \{(1,2),(2,3),(3,4),(4,5),(5,3)\}$



∀ label, the Analysis array is initialised

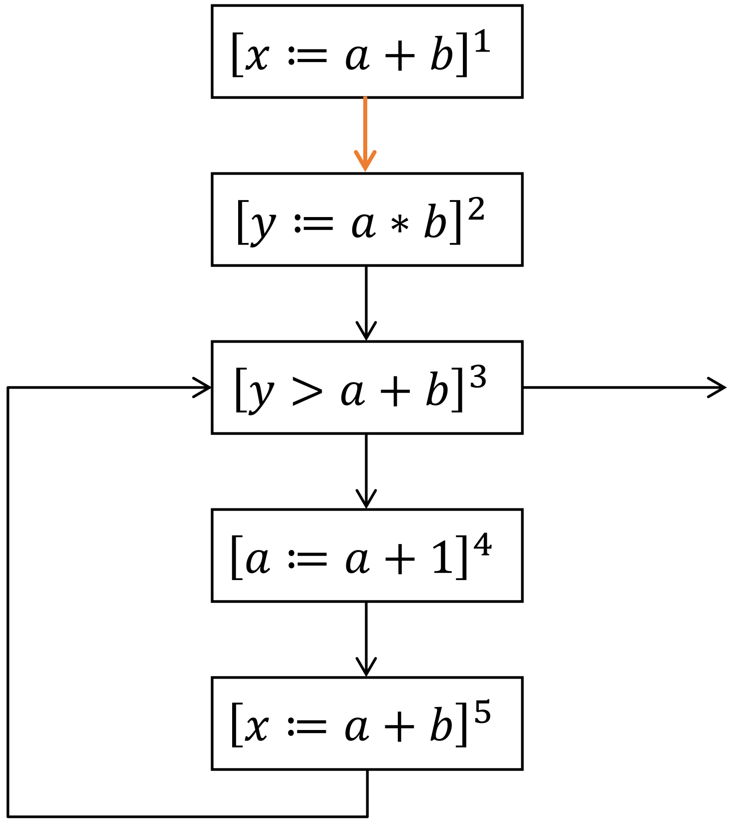
Results of Analysis

$Analysis[1]: \emptyset$
$Analysis[2]: \{a+b, a*b, a+1\}$
$Analysis[3]: \{a+b, a*b, a+1\}$
$Analysis[4]: \{a+b, a*b, a+1\}$
$Analysis[5]: \{a+b, a*b, a+1\}$

The analysis result for initial label (node 1) is empty.

Step 2

$W: \{(\mathbf{1},\mathbf{2}), (2,3), (3,4), (4,5), (5,3)\}$



Inspect 1st edge in the worklist: $(\mathbf{1},\mathbf{2})$
+
Propagate information, from node 1 to 2

Results of Analysis

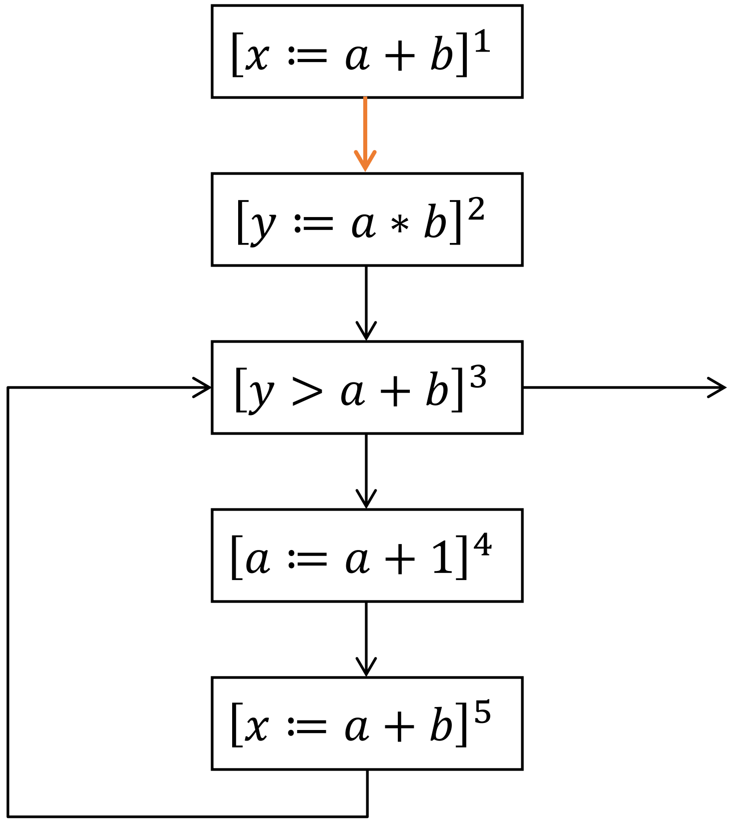
$Analysis[1]: \emptyset$
$Analysis[2]: \{a + b\}$
$Analysis[3]: \{a + b, a * b, a + 1\}$
$Analysis[4]: \{a + b, a * b, a + 1\}$
$Analysis[5]: \{a + b, a * b, a + 1\}$

The analysis result for node 2 (outgoing edge) has changed: only $(a + b)$ has propagated from the previous node.

Step 2

$W: \{(\mathbf{2}, \mathbf{3}), (2{,}3), (3{,}4), (4{,}5), (5{,}3)\}$

Edge $(\mathbf{2}, \mathbf{3})$ added on top of the worklist



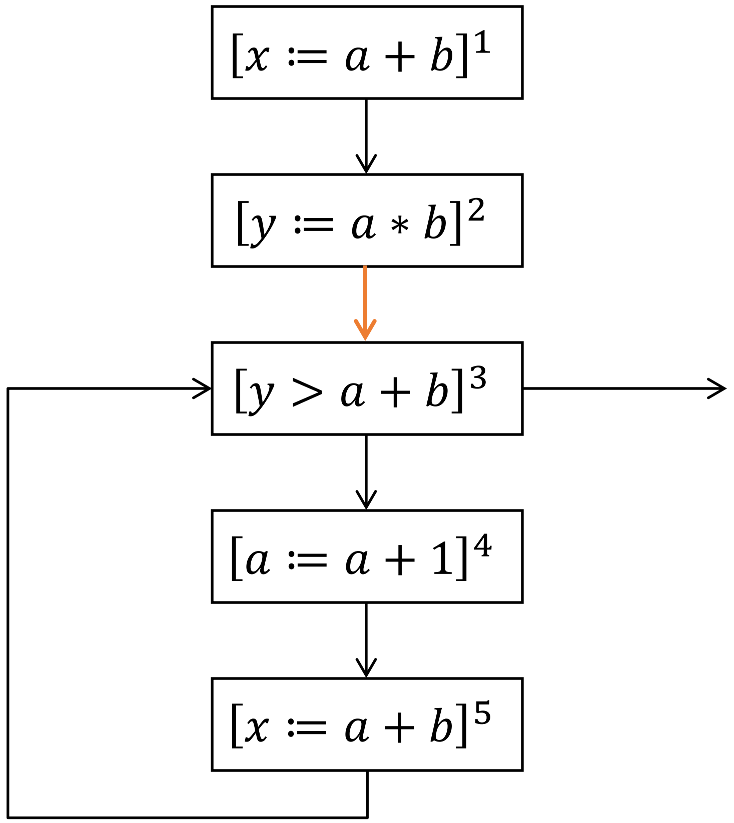
Results of Analysis

$Analysis[1]: \emptyset$
$Analysis[2]: \{a + b\}$
$Analysis[3]: \{a + b, a * b, a + 1\}$
$Analysis[4]: \{a + b, a * b, a + 1\}$
$Analysis[5]: \{a + b, a * b, a + 1\}$

- PROPAGATION: YES
- UPDATED NODE: 2
- **EDGES TO UPDATE: (2,3)**

Step 3

$W: \{(\mathbf{2}, \mathbf{3}), (2,3), (3,4), (4,5), (5,3)\}$



Inspect 1st edge in the worklist: $(\mathbf{2}, \mathbf{3})$

+

Propagate information, from node 2 to 3

Results of Analysis

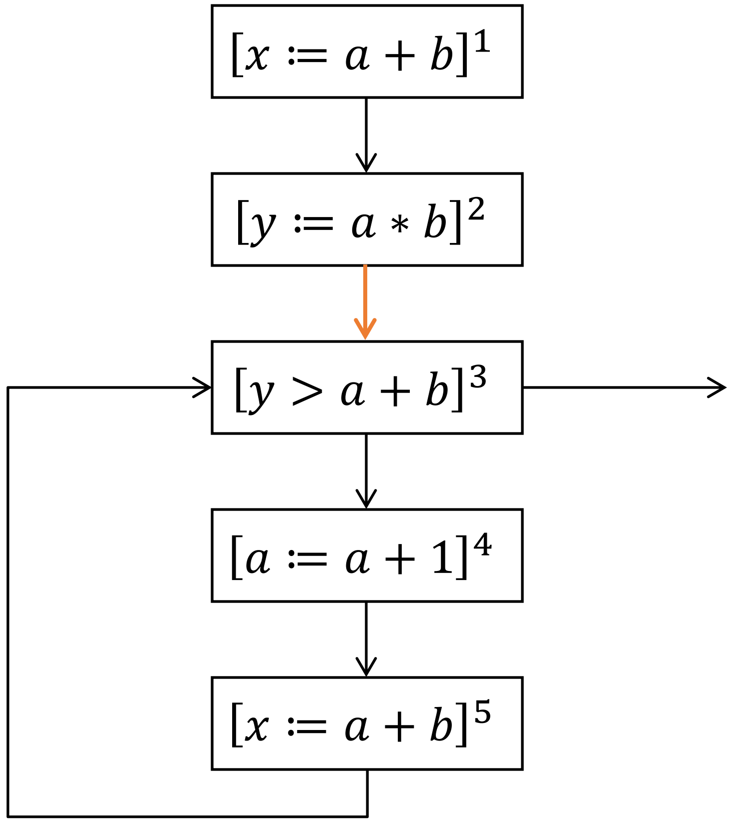
$Analysis[1]: \emptyset$
$Analysis[2]: \{a + b\}$
$Analysis[3]: \{a + b, a * b\}$
$Analysis[4]: \{a + b, a * b, a + 1\}$
$Analysis[5]: \{a + b, a * b, a + 1\}$

The propagated information now contains the expressions $(a + b)$ which is the analysis result for the previous node 2, and $(a * b)$ computed at node 2.

Step 3

$W: \{(\mathbf{3},\mathbf{4}),(2{,}3),(3{,}4),(4{,}5),(5{,}3)\}$

$\boxed{\textbf{Edge } (\mathbf{3},\mathbf{4}) \text{ added on top of the worklist}}$



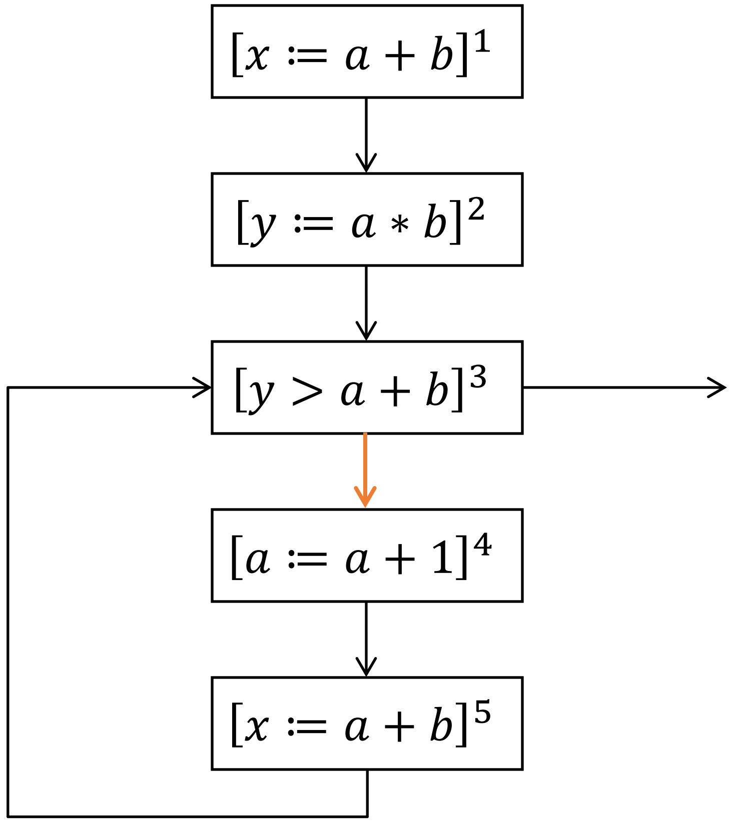
Results of Analysis

$Analysis[1]: \emptyset$
$Analysis[2]: \{a+b\}$
$Analysis[3]: \{a+b, a*b\}$
$Analysis[4]: \{a+b, a*b, a+1\}$
$Analysis[5]: \{a+b, a*b, a+1\}$

- PROPAGATION: YES
- UPDATED NODE: 3
- **EDGES TO UPDATE: (3,4)**

Step 4

$W: \{(\mathbf{3},\mathbf{4}), (2,3), (3,4), (4,5), (5,3)\}$



Inspect 1st edge in the worklist: $(\mathbf{3},\mathbf{4})$

+

Propagate information, from node 3 to 4

Results of Analysis

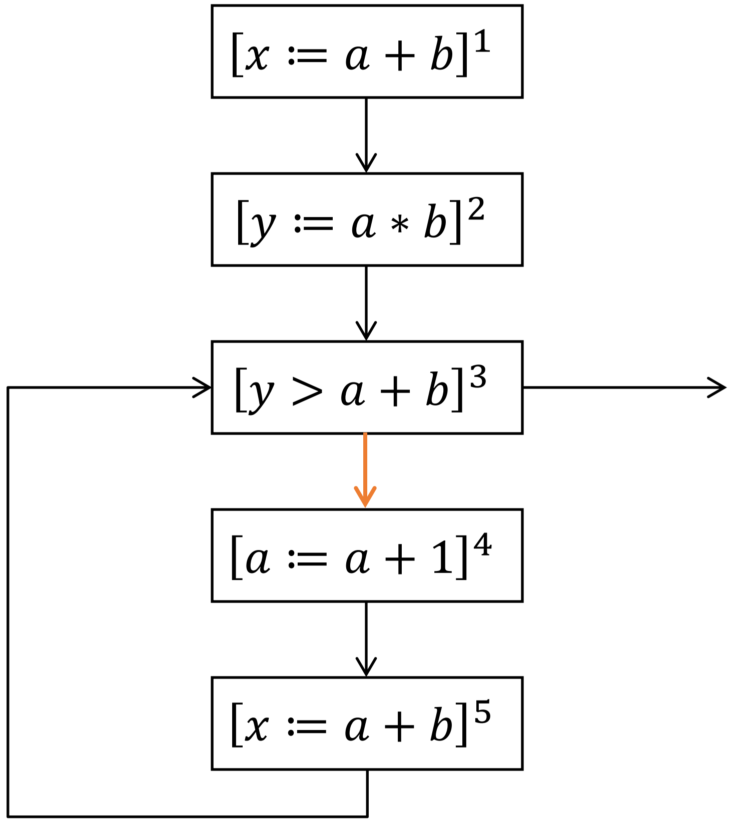
$Analysis[1]: \emptyset$
$Analysis[2]: \{a + b\}$
$Analysis[3]: \{a + b, a * b\}$
$Analysis[4]: \{a + b, a * b\}$
$Analysis[5]: \{a + b, a * b, a + 1\}$

Node 4 analysis results updated.

Step 4

$W: \{(\mathbf{4}, \mathbf{5}), (2,3), (3,4), (4,5), (5,3)\}$

Edge $(\mathbf{4}, \mathbf{5})$ added on top of the worklist



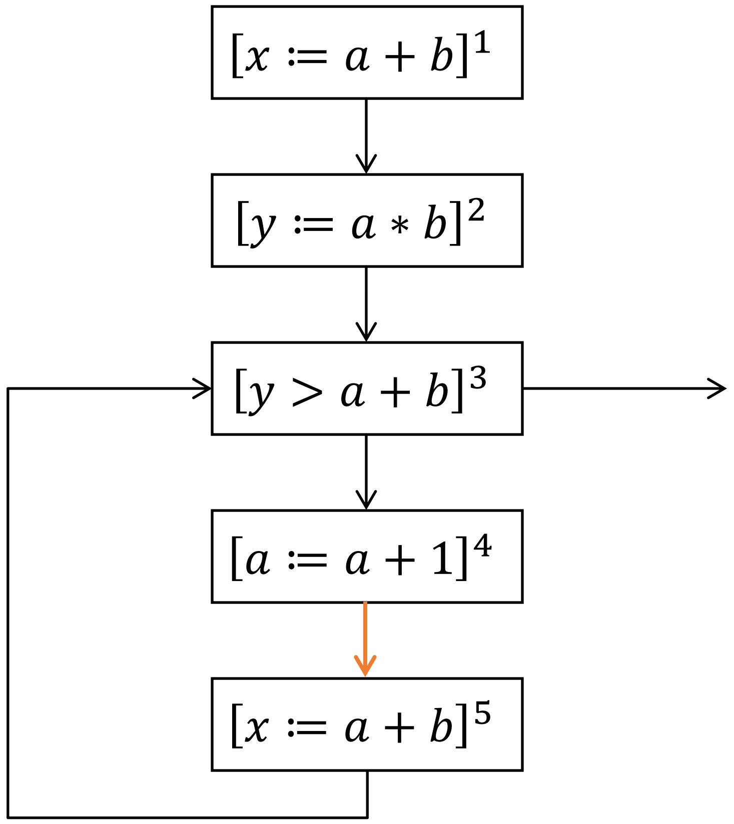
Results of Analysis

$Analysis[1]: \emptyset$
$Analysis[2]: \{a + b\}$
$Analysis[3]: \{a + b, a * b\}$
$Analysis[4]: \{a + b, a * b\}$
$Analysis[5]: \{a + b, a * b, a + 1\}$

- PROPAGATION: YES
- UPDATED NODE: 4
- **EDGES TO UPDATE: (4,5)**

Step 5

$W: \{(\mathbf{4}, \mathbf{5}), (2,3), (3,4), (4,5), (5,3)\}$



Inspect 1st edge in the worklist: $(\mathbf{4}, \mathbf{5})$

+

Propagate information, from node 4 to 5

Results of Analysis

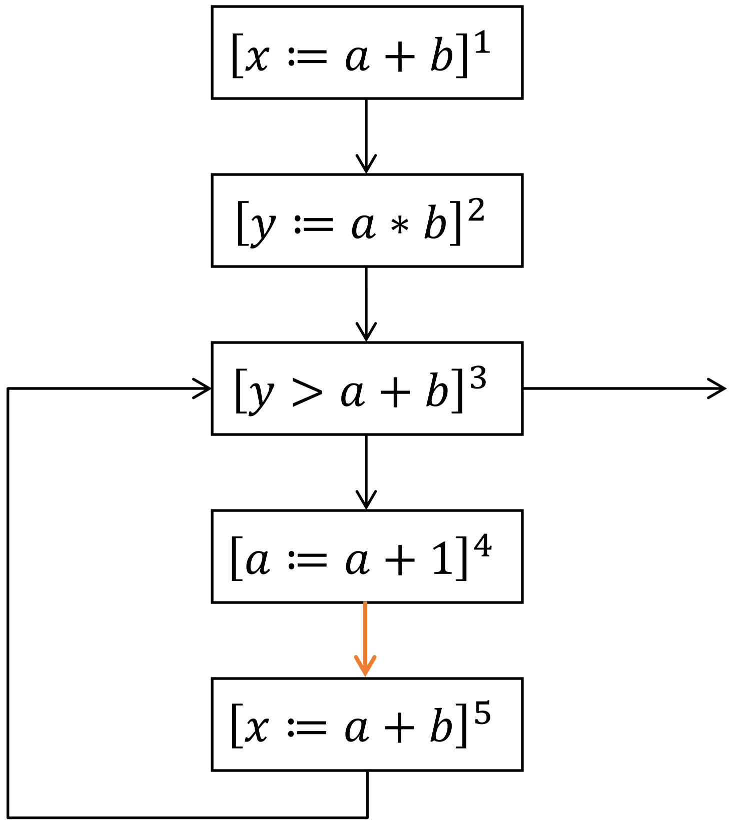
$Analysis[1]: \emptyset$
$Analysis[2]: \{a + b\}$
$Analysis[3]: \{a + b, a * b\}$
$Analysis[4]: \{a + b, a * b\}$
$Analysis[5]: \emptyset$

The analysis result for node 5 is emptied because all the available expressions are killed by the assignment at node 4.

Step 5

$W: \{(\mathbf{5}, \mathbf{3}), (2,3), (3,4), (4,5), (5,3)\}$

Edge $(\mathbf{5}, \mathbf{3})$ added on top of the worklist



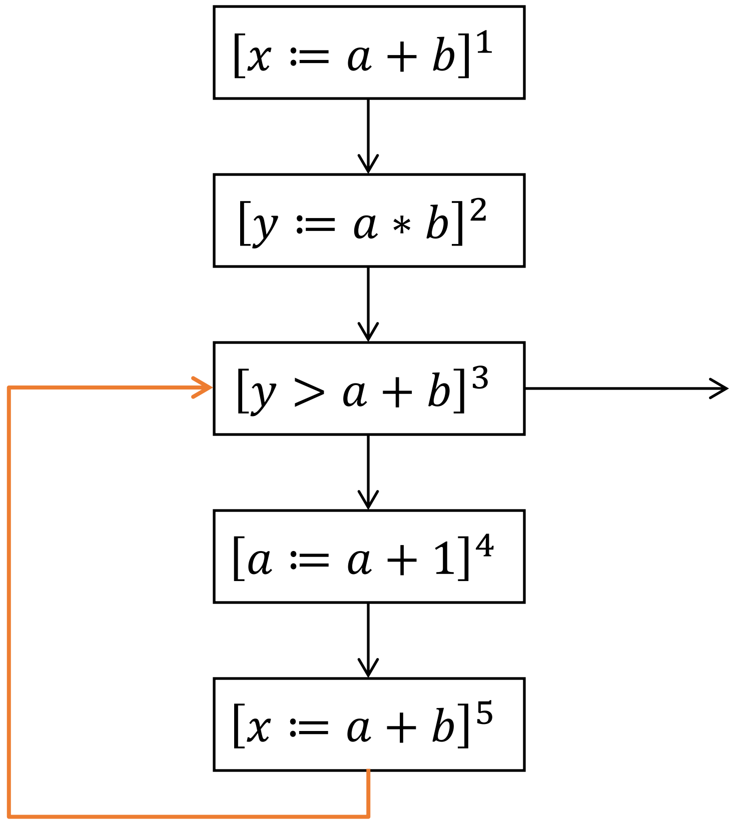
Results of Analysis

$Analysis[1]: \emptyset$
$Analysis[2]: \{a + b\}$
$Analysis[3]: \{a + b, a * b\}$
$Analysis[4]: \{a + b, a * b\}$
$Analysis[5]: \emptyset$

- PROPAGATION: YES
- UPDATED NODE: 5
- **EDGES TO UPDATE: (5,3)**

Step 6

$W: \{(\mathbf{5}, \mathbf{3}), (2,3), (3,4), (4,5), (5,3)\}$



Inspect 1st edge in the worklist: $(\mathbf{5}, \mathbf{3})$

+

Propagate information, from node 5 to 3

Results of Analysis

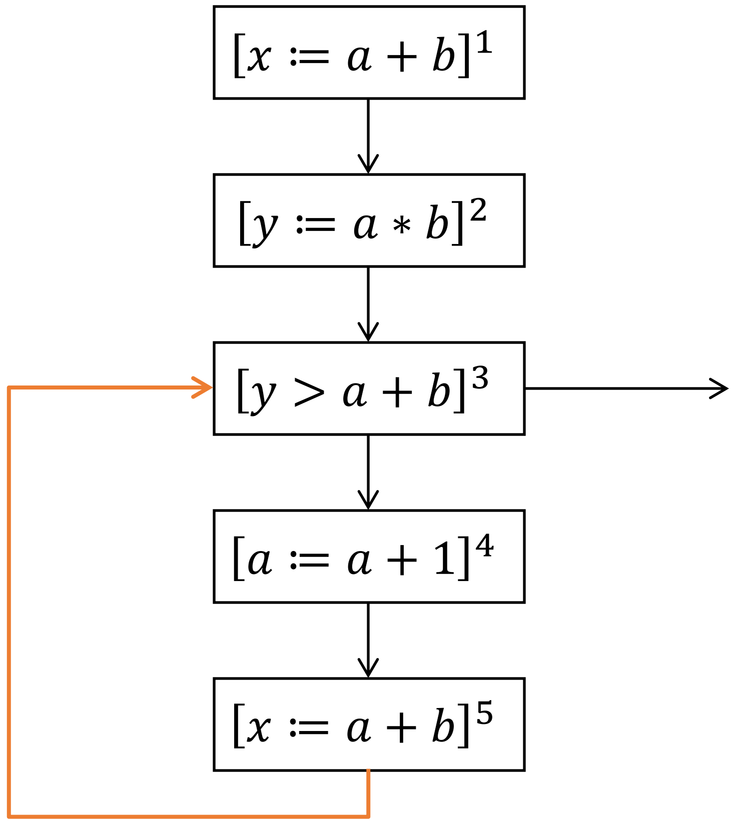
$Analysis[1]: \emptyset$
$Analysis[2]: \{a + b\}$
$Analysis[3]: \{a + b\}$
$Analysis[4]: \{a + b, a * b\}$
$Analysis[5]: \emptyset$

Expression $(a * b)$ removed from node 3, because previously killed by statement 4.

Step 6

$W: \{(\mathbf{3},\mathbf{4}), (2,3), (3,4), (4,5), (5,3)\}$

Edge $(\mathbf{3},\mathbf{4})$ added on top of the worklist



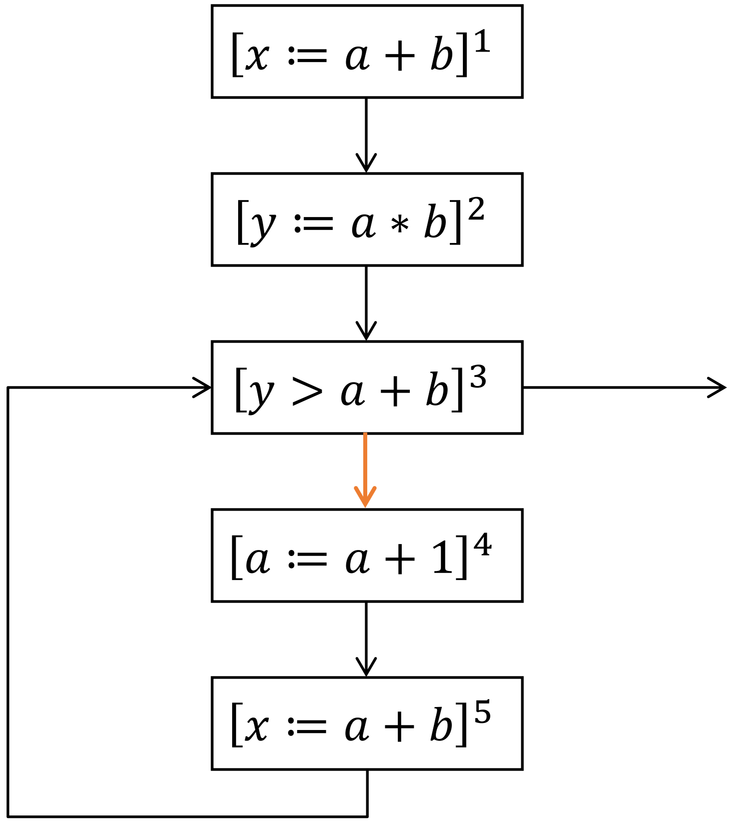
Results of Analysis

$Analysis[1]: \emptyset$
$Analysis[2]: \{a + b\}$
$Analysis[3]: \{a + b\}$
$Analysis[4]: \{a + b, a * b\}$
$Analysis[5]: \emptyset$

- PROPAGATION: YES
- UPDATED NODE: 3
- **EDGES TO UPDATE: (3,4)**

Step 7

$W: \{(\mathbf{3},\mathbf{4}), (2,3), (3,4), (4,5), (5,3)\}$



Inspect 1st edge in the worklist: $(\mathbf{3},\mathbf{4})$
+
Propagate information, from node 3 to 4

Results of Analysis

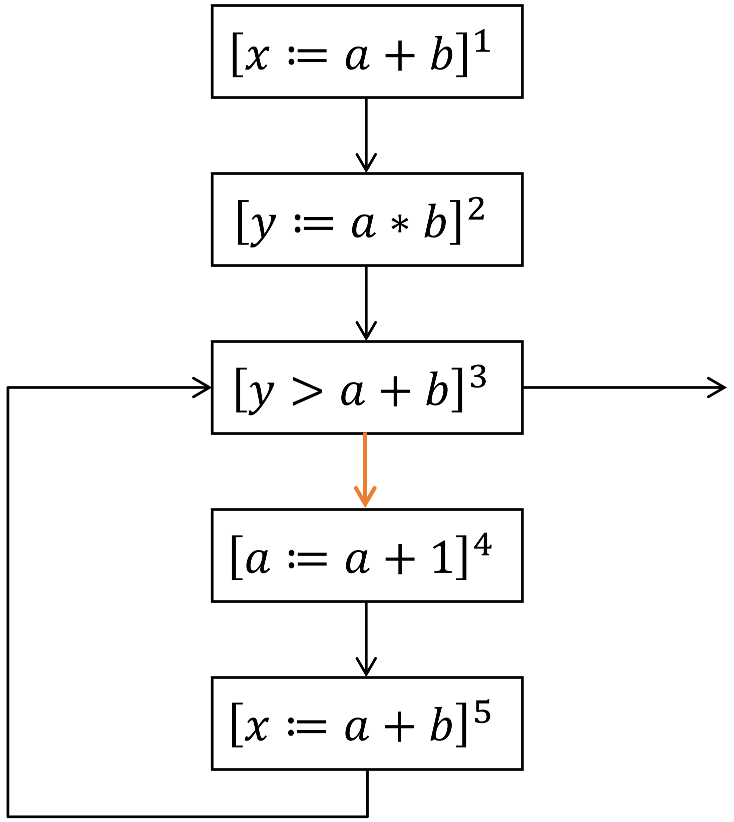
$Analysis[1]: \emptyset$
$Analysis[2]: \{a+b\}$
$Analysis[3]: \{a+b\}$
$Analysis[4]: \{a+b\}$
$Analysis[5]: \emptyset$

Node 4 analysis results updated.

Step 7

$W: \{(\mathbf{4}, \mathbf{5}), (2,3), (3,4), (4,5), (5,3)\}$

Edge $(\mathbf{4}, \mathbf{5})$ added on top of the worklist



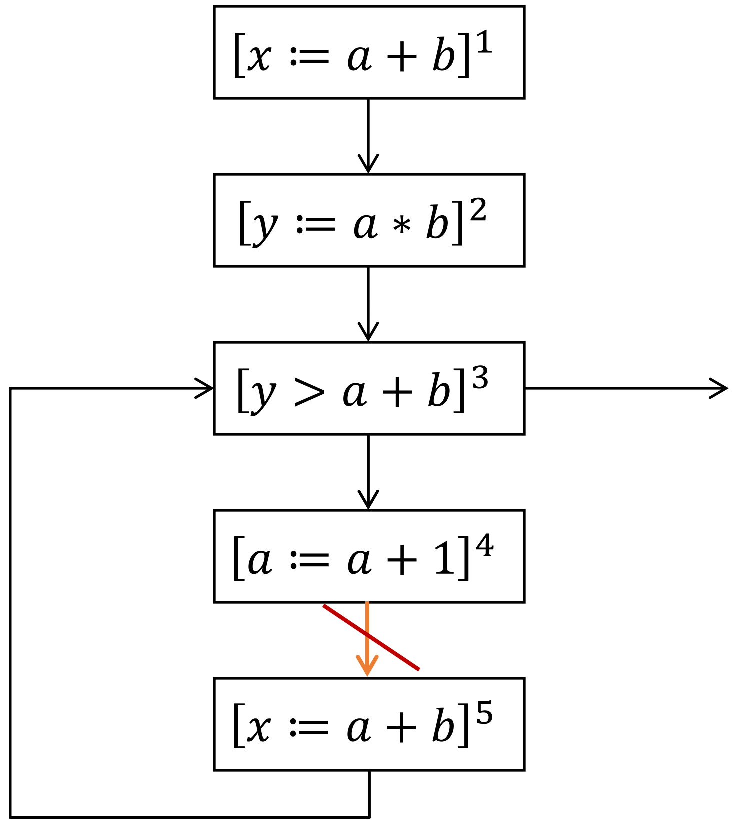
Results of Analysis

$Analysis[1]: \emptyset$
$Analysis[2]: \{a + b\}$
$Analysis[3]: \{a + b\}$
$Analysis[4]: \{a + b\}$
$Analysis[5]: \emptyset$

- PROPAGATION: YES
- UPDATED NODE: 4
- **EDGES TO UPDATE: (4,5)**

Step 8

$W: \{(\mathbf{4},\mathbf{5}), (2,3), (3,4), (4,5), (5,3)\}$



Inspect 1st edge in the worklist: **(4,5)**
+
Propagate information, from node 4 to 5

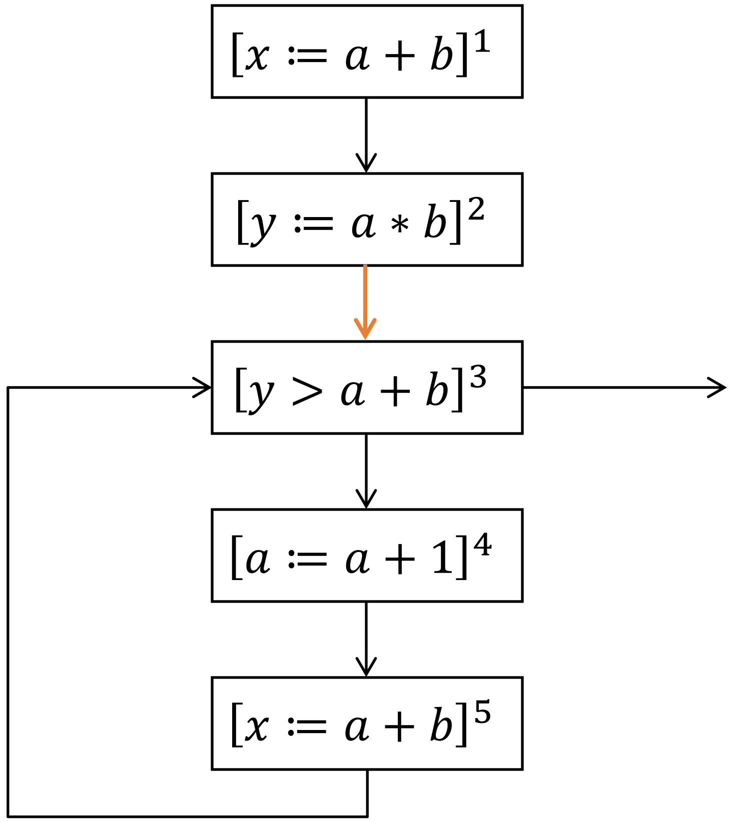
Results of Analysis

$Analysis[1]: \emptyset$
$Analysis[2]: \{a + b\}$
$Analysis[3]: \{a + b\}$
$Analysis[4]: \{a + b\}$
$Analysis[5]: \emptyset$

No change in the outcoming node 5 of current edge.

Step 8

$W: \{(2,3), (3,4), (4,5), (5,3)\}$



The worklist starts to get smaller until it becomes empty

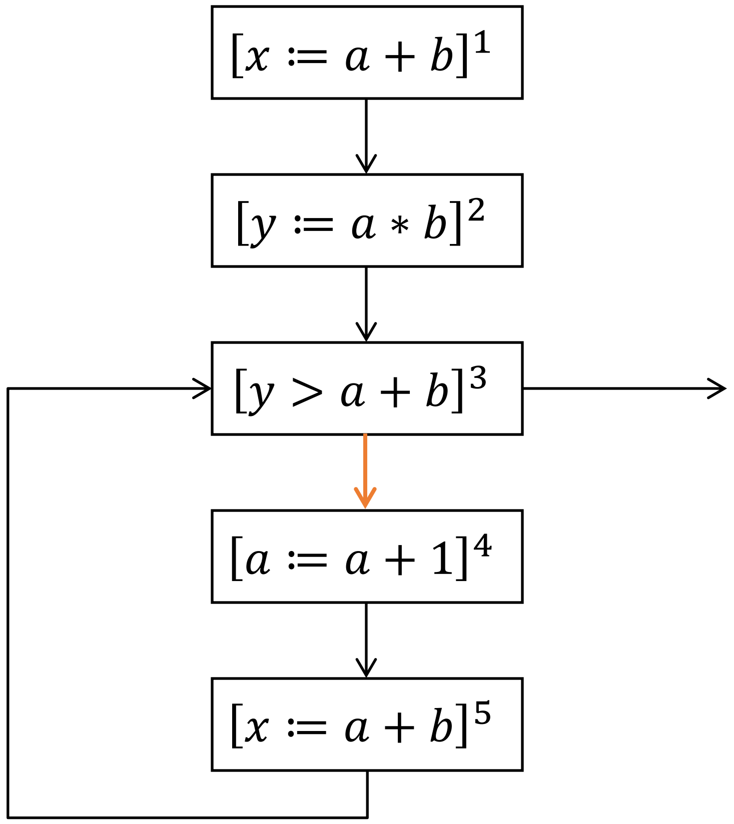
Results of Analysis

$Analysis[1]: \emptyset$
$Analysis[2]: \{a + b\}$
$Analysis[3]: \{a + b\}$
$Analysis[4]: \{a + b\}$
$Analysis[5]: \emptyset$

- PROPAGATION: NO
  At each iteration, no new information is propagated (the analysis results array no longer changes) .

Step 9

$W: \{(3,4), (4,5), (5,3)\}$

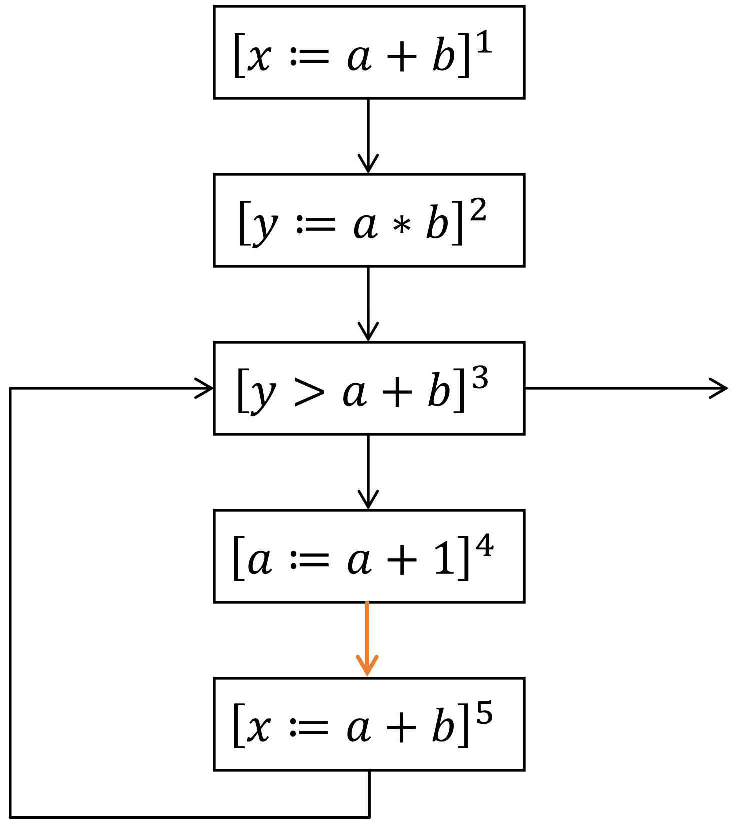


Results of Analysis

$Analysis[1]: \emptyset$
$Analysis[2]: \{a + b\}$
$Analysis[3]: \{a + b\}$
$Analysis[4]: \{a + b\}$
$Analysis[5]: \emptyset$

Step 10

$W: \{(4,5), (5,3)\}$

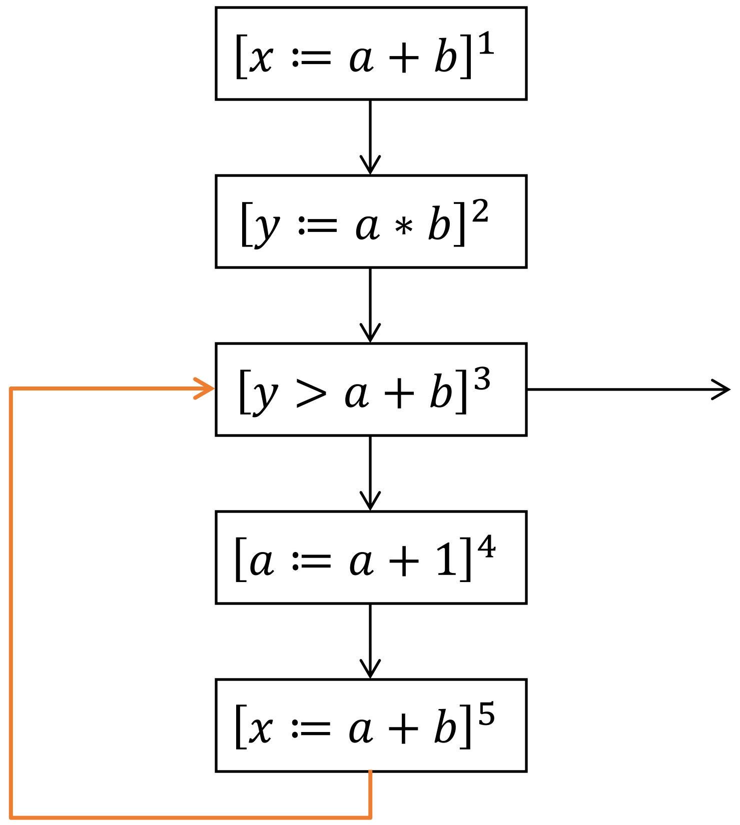


Results of Analysis

$Analysis[1]: \emptyset$
$Analysis[2]: \{a + b\}$
$Analysis[3]: \{a + b\}$
$Analysis[4]: \{a + b\}$
$Analysis[5]: \emptyset$

Step 11

$W: \{(5,3)\}$

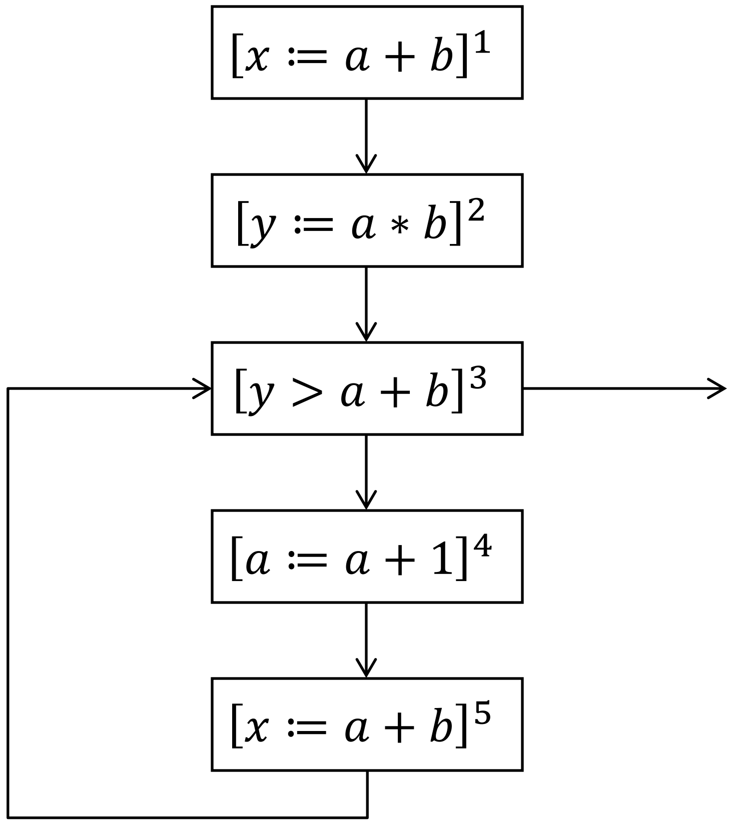


Results of Analysis

$Analysis[1]: \emptyset$
$Analysis[2]: \{a + b\}$
$Analysis[3]: \{a + b\}$
$Analysis[4]: \{a + b\}$
$Analysis[5]: \emptyset$

Step 12

$W: \{\}$



Worklist empty, the algorithm terminates.

It computes the least (or MFP) solution to the instance of the framework in input: final *AEs* ∀ node (*fixed point* reached).

Results of Analysis: OUTPUT

$Analysis[1]: \emptyset$
$Analysis[2]: \{a + b\}$
$Analysis[3]: \{a + b\}$
$Analysis[4]: \{a + b\}$
$Analysis[5]: \emptyset$

In node 3 the expression $(a + b)$ can be reused because already computed in other nodes (except for node 1 and 5, where it is computed for the first time).

MOP (Meet Over Paths) Algorithm

- OUTPUT:
  - $MOP_{\diamond}(\ell) \coloneqq \bigsqcup\{f_{\vec{\ell}}(\iota) \mid \vec{l} \in path_{\diamond}(\ell)\}$
  - $MOP_{\blacklozenge}(\ell) \coloneqq \bigsqcup\{f_{\vec{\ell}}(\iota) \mid \vec{l} \in path_{\blacklozenge}(\ell)\}$

$\rightarrow$ The analysis result is a combination (join) of the results (information) across all paths.

BUT

Generally, not computable in practice (undecidable) because there can be infinite n° of paths in a program!

MFP vs MOP

Theorem. MFP algorithm always terminates and computes the least solution to the instance of the monotone framework given as input.

Theorem. MOP is undecidable (for certain kind of monotone frameworks).

Theorem. For monotone frameworks,

$$MFP_{\diamond} \underbrace{\sqsupseteq MOP_{\diamond} \text{ and } MFP_{\blacklozenge} \sqsupseteq}_{\text{MFP safe approximates MOP:}} MOP_{\blacklozenge}$$

$$\downarrow$$

Except for quite simple formulations, for which they produce an identical result:
MFP results are **less accurate** (always **greater** than MOP results) but **more practical**.
e.g. If MOP returns the least (greatest) solution, MFP returns a slightly bigger (smaller) set.