Principles of Abstract Interpretation

Abstract Interpretation Framework

Abstract interpretation Framework

real execution $[\![P]\!] = fix\ F \in D$ A domain of concrete states (e.g.sets of integers)

abstract execution $[\![\widehat{P}]\!] = fix\ \widehat{F} \in D$ A domain of abstract states (e.g.sets of intervals)

correctness $[\![P]\!] \approx [\![\widehat{P}]\!]$

implementation computation of $[\![\widehat{P}]\!]$

Abstract interpretation Framework

real execution $[\![P]\!] = fix\ F \in D$

abstract execution $[\![\widehat{P}]\!] = fix\ \widehat{F} \in D$

correctness $[\![P]\!] \approx [\![\widehat{P}]\!]$

The framework requires:

- a relation between D and $\widehat{D}$
- A relation between $F : D \to D$ and $\widehat{F} : \widehat{D} \to \widehat{D}$

A function corresponding to one-step abstract execution

A function corresponding to one-step concrete execution

The framework guarantees:

- correctness and implementation
- freedom: any such $\widehat{D}$ and $\widehat{F}$ are fine

Recipe for the construction of an abstract interpreter

Step 1 : Define the language and a concrete semantics

Step 2 : Select an abstraction describing the set of properties

Step 3 : Derive the abstract semantics

The language

Assume a syntax for arithmetic expressions E and Boolean expression B, the syntax for the command is the following

C	$::=$		commands
	$\mid$	**skip**	command that "does nothing"
	$\mid$	$C; C$	sequence of commands
	$\mid$	$\mathtt{x} := E$	assignment command
	$\mid$	**input**(x)	command reading of a value
	$\mid$	**if**$(B)\{C\}$**else**$\{C\}$	conditional command
	$\mid$	**while**$(B)\{C\}$	loop command
P	$::=$	C	program

Step 1: Define concrete Semantics

Formalization of a single program execution

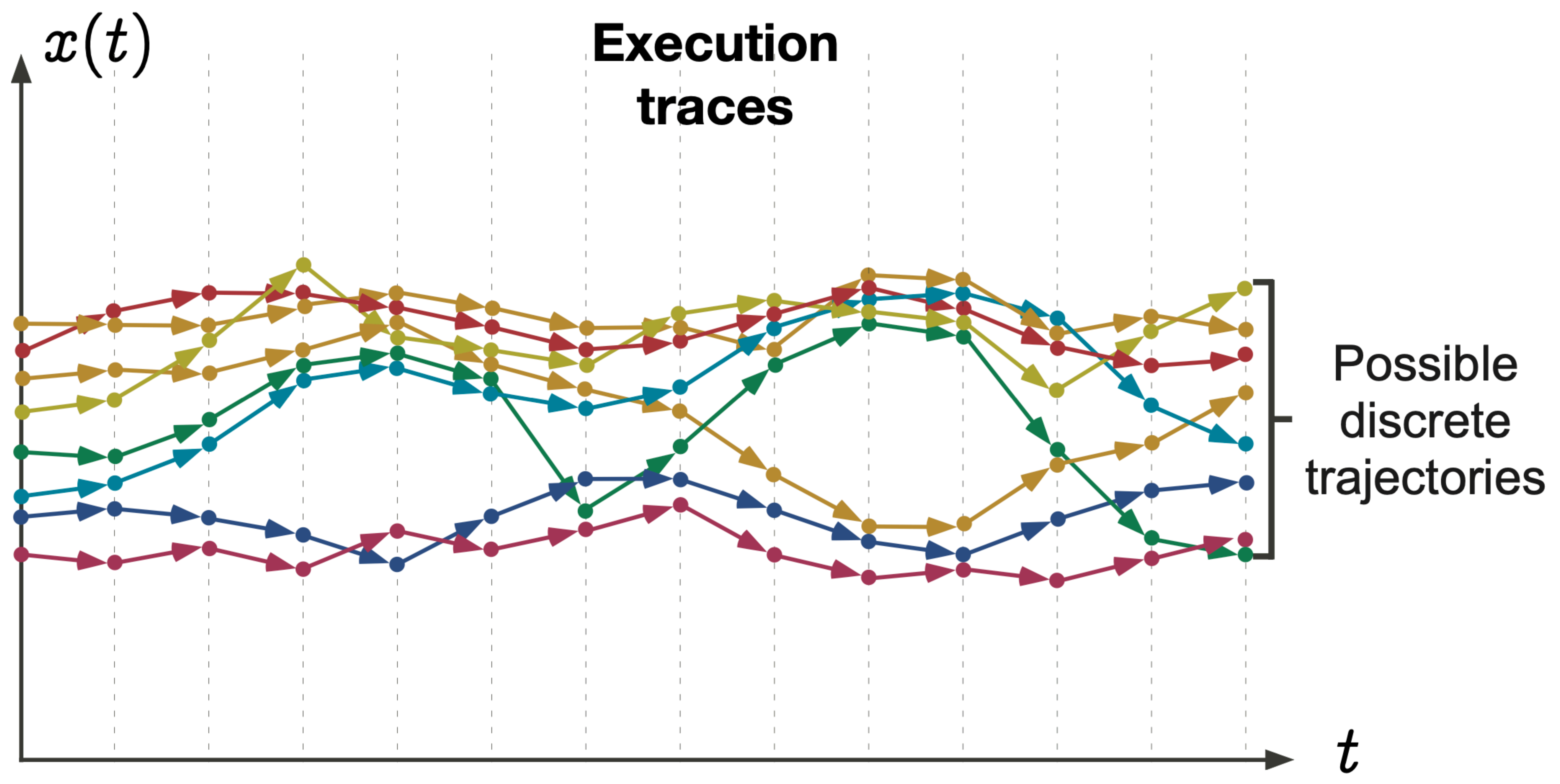
Operational semantics (transitional style)

- Big-step / small-step

Denotational semantics (compositional style)

- State $\rightarrow$ State

Step 1: Define concrete Semantics



Semantics Style: Compositional vs.Transitional

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

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

For some realistic languages, even defining their compositional ("denotational") semantics is a hurdle

- goto, exceptions, function calls

Transitional-style ("operational") semantics avoids the hurdle

$$[\![AB]\!] = \{s_1 \rightarrow s_2 \rightarrow ...\}$$

Step 1: Define concrete Semantics

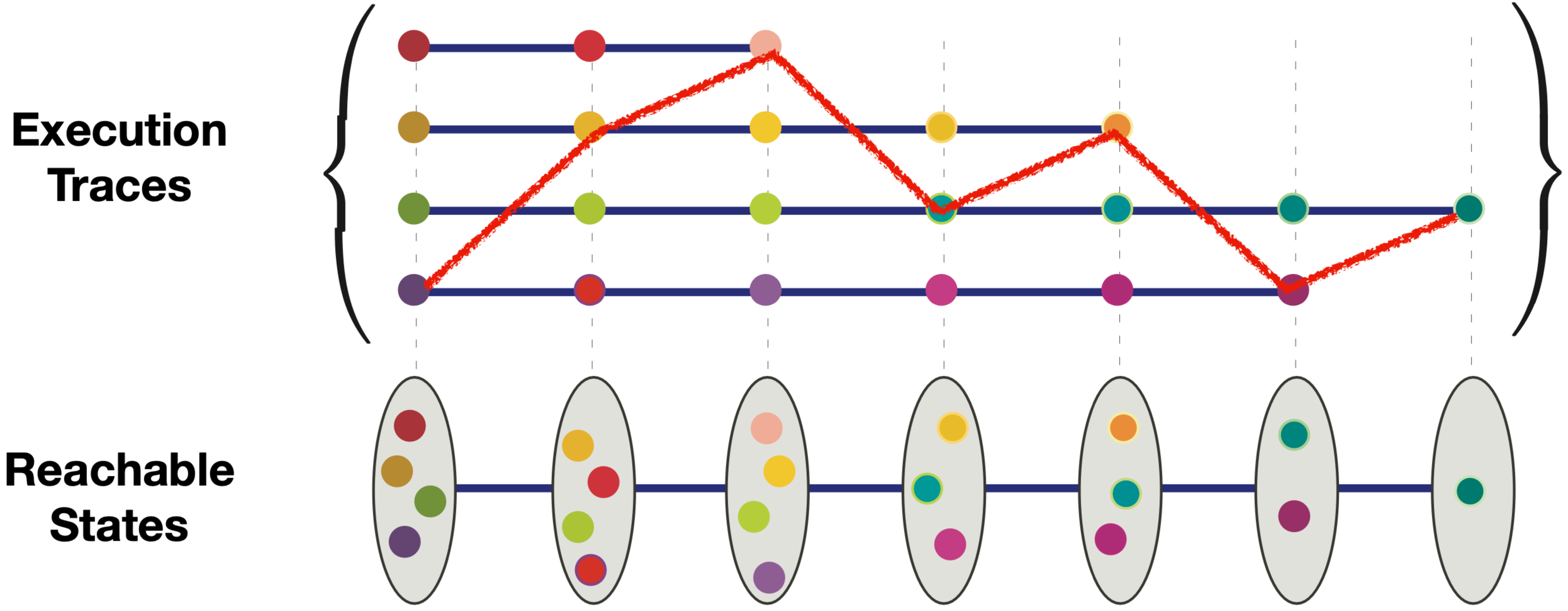
Formalization of all possible program executions

Also called collecting semantics

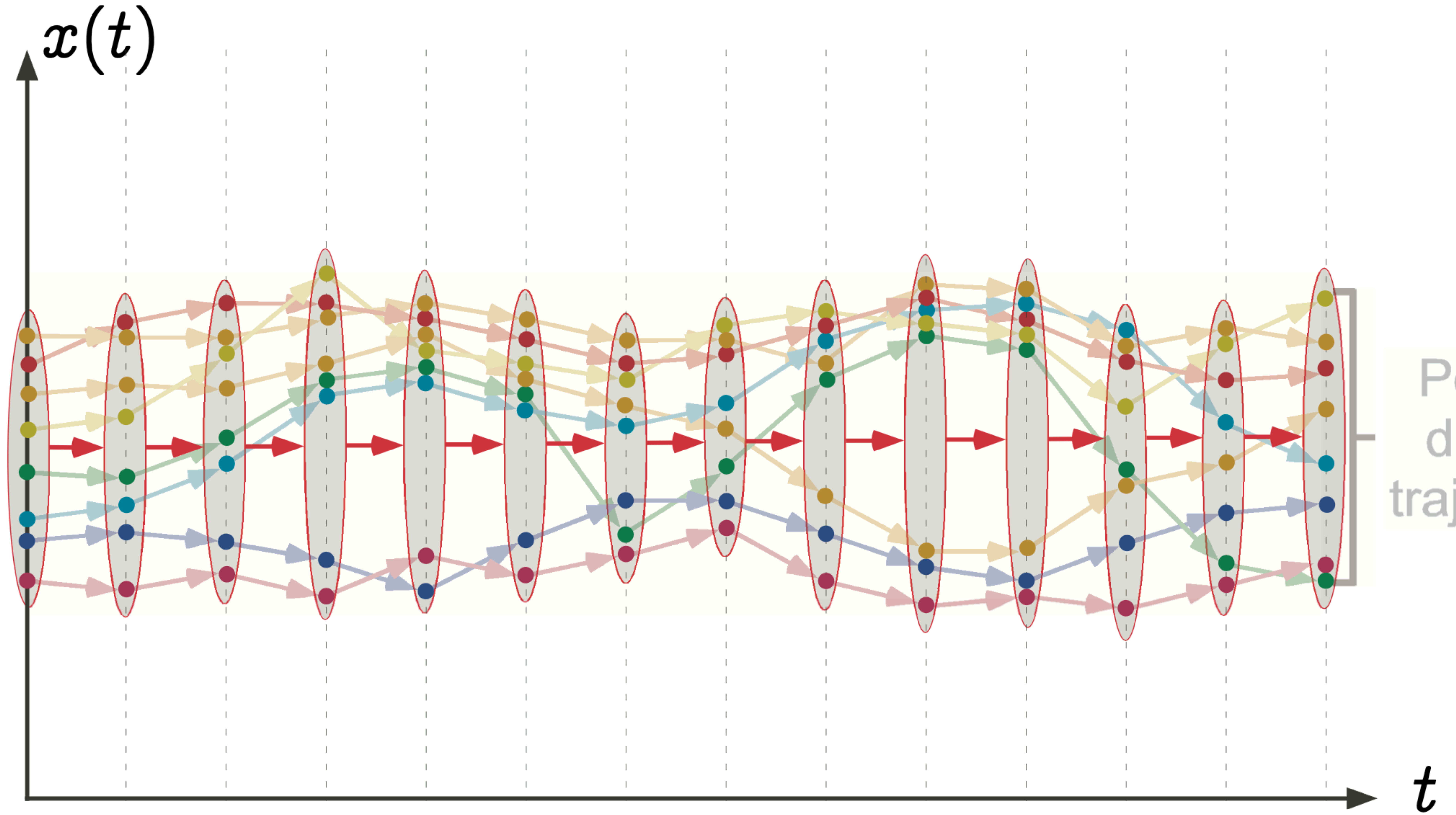
Simple extension of the standard semantics in general

$$2^{States} \rightarrow 2^{States}$$

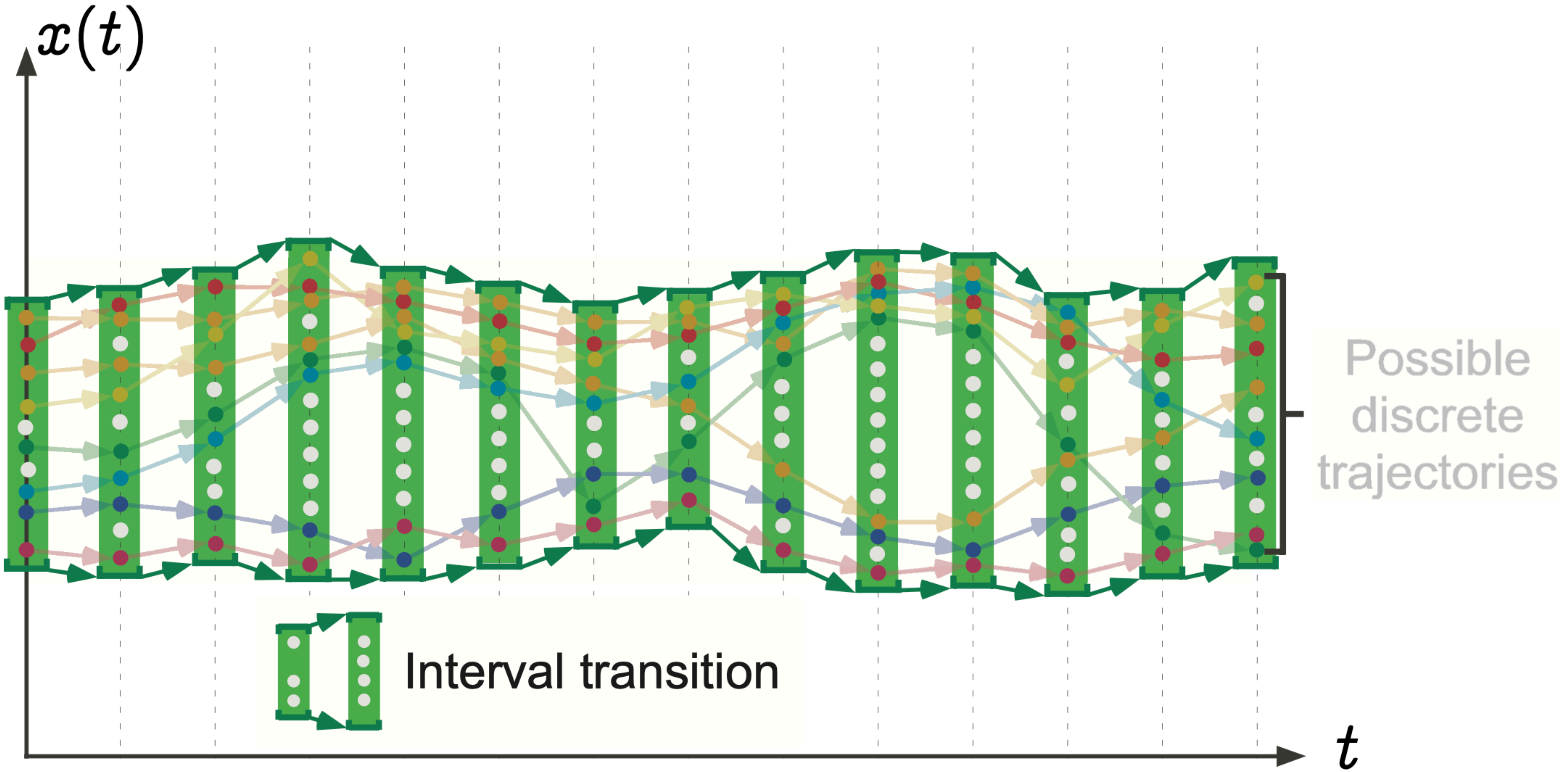
Traces vs. Reachable States



Transitions of sets of States



Transitions of Abstract States



Collecting Semantics

$$x \in \mathbb{X} = ProgramVariables$$

$$\mathbb{V} = \mathbb{Z}$$

$$m \in \mathbb{M} = \mathbb{X} \to \mathbb{V}$$

Memories

Assume $[\![E]\!] : \mathbb{M} \to \mathbb{V}$ and $[\![B]\!] : \mathbb{M} \to \mathbb{B}$

$$[\![C]\!] : \wp(\mathbb{M}) \to \wp(\mathbb{M})$$

$M \in \wp(\mathbb{M})$

$$[\![\ \texttt{skip}\ ]\!](M) = M$$

$$[\![\ \texttt{x} := E]\!](M) = \{\ m[\texttt{x} \mapsto [\![\ \texttt{E}]\!](m)] \mid m \in M\}$$

$$[\![\ \texttt{C}_0; \texttt{C}_1\ ]\!](M) = [\![\ \texttt{C}_1\ ]\!]([\![\ \texttt{C}_0\ ]\!](M))$$

$$[\![\ \texttt{input}]\!](M) = \{\ m[\texttt{x} \mapsto n] \mid n \in \mathbb{V}, m \in M\}$$

Filtering function for the conditional

Since M is a set of states, the conditional filters the memories for which the condition is true and for them evaluate the first branch, do the same for the memories for which the condition is false and take the union

For each Boolean expression B, the filtering function

$$\mathcal{F}_B(M) = \{\, m \in M \mid [\![B]\!](m) = \texttt{true}\}$$

Collecting semantics for the conditional

$$\mathcal{F}_B(M) = \{\, m \in M \mid [\![B]\!](m) = \texttt{true}\}$$

Syntactic negation
E.g. $\neg(x > 3) = x \leq 3$

$$\mathcal{F}_{\neg B}(M) = \{\, m \in M \mid [\![\neg B]\!](m) = \texttt{true}\} = \{\, m \in M \mid [\![B]\!](m) = \texttt{false}\}$$

$$[\![\,\texttt{if}\ \ (B)\{\texttt{C}_0\}\ \texttt{else}\ \{\texttt{C}_1\}]\!](M) = [\![\texttt{C}_0]\!]\ \mathcal{F}_B(M) \cup [\![\texttt{C}_1]\!]\ \mathcal{F}_{\neg B}(M)$$

Collecting semantics $[\![\text{while}(B)\{C\}]\!](M)$

We can partition executions based on the number of iterations they spend inside the loop before exit

M_i denotes the memories that are produced by program executions that wer thought the loop body exactly i times starting from M

$$M_1 = \mathcal{F}_{\neg B}(\ [\![C]\!]\ \mathcal{F}_B(M))$$

$$M_2 = \mathcal{F}_{\neg B}(\ [\![C]\!]\ \mathcal{F}_B\ [\![C]\!]\ \mathcal{F}_B(M)) = \mathcal{F}_{\neg B}((\ [\![C]\!]\ \mathcal{F}_B)^2(M))$$

$$M_i = \mathcal{F}_{\neg B}((\ [\![C]\!]\ \mathcal{F}_B)^i(M))$$

Collecting semantics $[\![\text{while}(B)\{C\}]\!](M)$

Thus, the set of output states of the loop is

$$\bigcup_{i\geq 0} M_i = \bigcup_{i\geq 0} \mathcal{F}_{\neg B}((\,[\![C]\!]\,\mathcal{F}_B)^i(M))$$

Since $\mathscr{F}_B$ commutes with the union

$$\bigcup_{i\geq 0} M_i = \mathcal{F}_{\neg B}(\bigcup_{i\geq 0}(\,[\![C]\!]\,\mathcal{F}_B)^i(M))$$

Definition as fix-point

$$[\![\text{while}(B)\{C\}]\!](M) = \mathcal{F}_{\neg B}(\bigcup_{i\geq 0}(\,[\![C]\!]\,\mathcal{F}_B)^i(M))$$

This can be rewritten as

$$[\![\text{while}(B)\{C\}]\!](M) = \mathcal{F}_{\neg B}(fix\ F_M)$$

where $$F_M = \lambda M'.M \cup [\![C]\!]\,\mathcal{F}_B(M')$$

Definition as fix-point

$$[\![\text{while}(B)\{C\}]\!](M) = \mathcal{F}_{\neg B}(fix\ F_M)$$

$$F_M = \lambda M'.M \cup [\![C]\!]\ \mathcal{F}_B(M')$$

F_M is continuous then we can apply the Kleene's theorem to compute the invariant

$$F_M^0 = F_M(\emptyset) = M$$
$$F_M^1 = F_M(F_M^0) = M \cup [\![C]\!]\ \mathcal{F}_B(M)$$
$$F_M^2 = F_M(F_M^1) = M \cup ([\![C]\!]\ \mathcal{F}_B)^2(M)$$
$$\vdots$$
$$F_M^i = M \cup ([\![C]\!]\ \mathcal{F}_B)^i(M)$$

$$[\![\text{while}(B)\{C\}]\!](M) = \mathcal{F}_{\neg B}(\cup_{i<0} F_M^i)$$

Toward abstraction

Our concrete domain $(\wp(\mathbb{M}), \subseteq)$

We abstract each concrete element with an abstract element

$c \vDash a$ when the abstract element a describes c

$$M_0 = \{m \in \mathbb{M} \mid 0 \leq m(x) \leq m(y) \leq 8\} \models M^{\#} = \{x \mapsto [0, 10], y \mapsto [0, 80]\}$$

$$M_1 = \{m \in \mathbb{M} \mid 1 \leq m(x)\} \not\models M^{\#} = \{x \mapsto [0, 10], y \mapsto [0, 80]\}$$

Abstract relation

Given a concrete domain $(C, \subseteq)$ an abstraction is defined by an abstract domain $(A, \sqsubseteq)$ and an abstract relation $\models \subseteq C \times A$ such that

- if $a_0 \sqsubseteq a_1$ and $c \models a_0$ then also $c \models a_1$

$$a_0 = \{x \mapsto [0,10], y \mapsto [0,80]\} \sqsubseteq a_1 = \{y \mapsto [0,100]\}$$

$$c_1 = \{m \in \mathbb{M} \mid 0 \leq m(x) \leq m(y) \leq 8\} \models a_0 \implies c_1 \models a_1$$

- if $c_0 \subseteq c_1$ and $c_1 \models a$ then also $c_0 \models a$

$$c_0 = \{m \in \mathbb{M} \mid 0 \leq m(x) \leq 4, m(y) = 6\} \subseteq c_1$$

$$c_1 \models a_0 \implies c_0 \models a_0$$

Concretization function

A common way to describe the abstract relation ⊨ is by defining a function that maps each abstract element to the largest concrete element it describes

Definition

Concretization function $\gamma : A \rightarrow C$ is a monotone function that maps abstract a into the greatest concrete c that satisfies a ($c \vDash a$).

$$c \models a \;\Leftrightarrow\; c \subseteq \gamma(a)$$

$$\gamma(a_0) = \gamma(\{x \mapsto [0,10], y \mapsto [0,80]\}) = \{m \in \mathbb{M} \mid 0 \leq m(x) \leq 10, 0 \leq m(y) \leq 80\}$$

$$c_1 = \{m \in \mathbb{M} \mid 0 \leq m(x) \leq m(y) \leq 8\} \;\models\; a_0 \text{ since } c_1 \subseteq \gamma(a_0)$$

Abstraction function

Another way to describe the abstract relation ⊨ is by defining a function that maps each concrete element to the smallest abstract element that describes it

Definition

Abstraction function $\alpha : C \to A$ (if it exists) is a monotone function

that maps concrete c into the most precise abstract a that describes c ($c \models a$).

$$c \models a \;\Leftrightarrow\; \alpha(c) \sqsubseteq a$$

$$\alpha(c_1) = \alpha(\{m \in \mathbb{M} |\ 0 \leq m(x) \leq m(y) \leq 8\}) = \{x \mapsto [0,8], y \mapsto [0,8]\}$$

$$c_1 \models a_1 = \{y \mapsto [0,100]\} \text{ since } \alpha(c1) \sqsubseteq a_1$$

Galois connection

α and γ should agree on a same abstraction relation $\models$

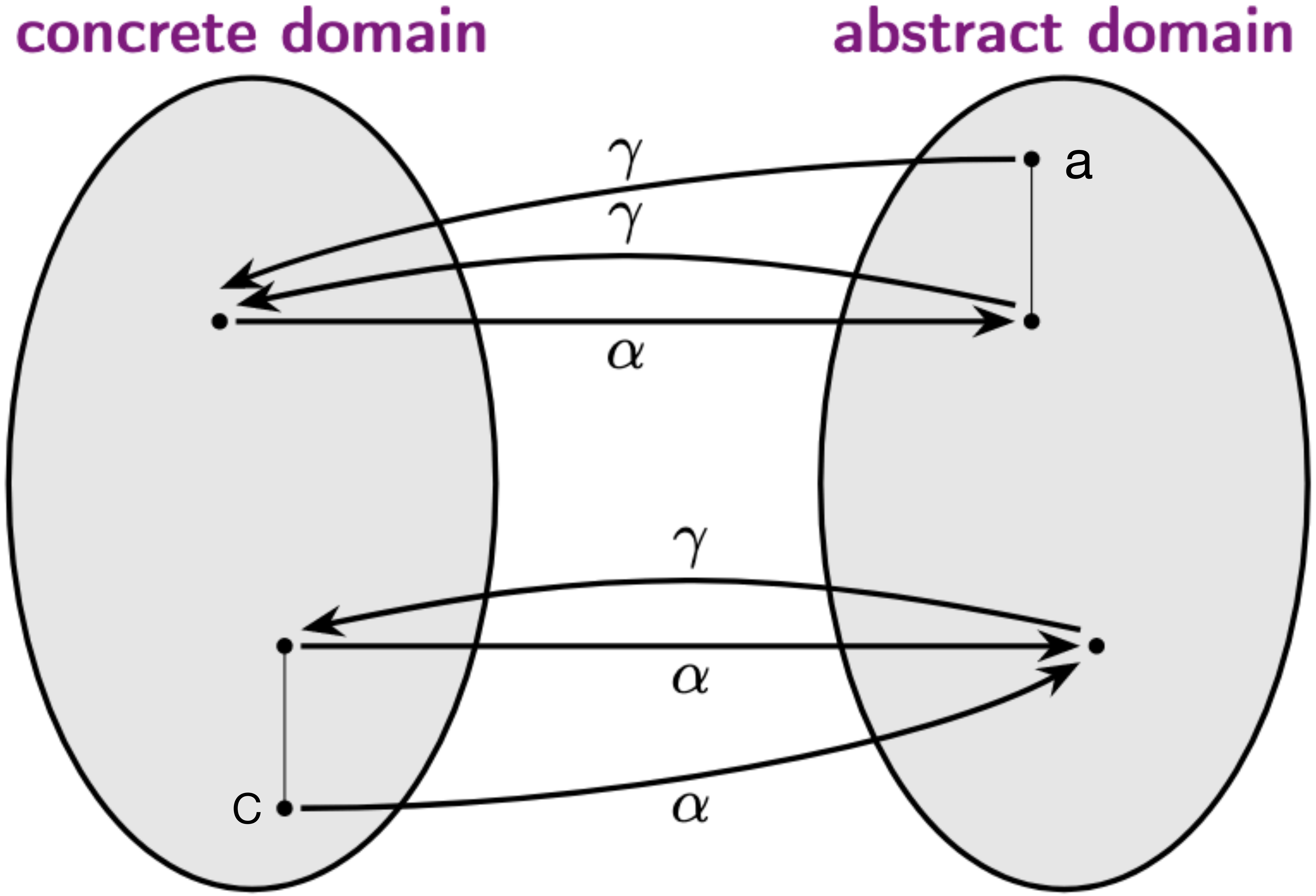
$$c \models a \Leftrightarrow c \subseteq \gamma(a) \Leftrightarrow \alpha(c) \sqsubseteq a$$

Definition

Galois connection: a pair of concretization function $\gamma : A \rightarrow C$ and an abstraction function $\alpha : C \rightarrow A$ such that

$$c \subseteq \gamma(a) \Leftrightarrow \alpha(c) \sqsubseteq a$$

Properties of Galois connections



- α and γ are monotone
- $c \subseteq \gamma(\alpha(c))$
- $\alpha(\gamma(a)) \sqsubseteq a$

Step 2: Non-relational abstractions

Non-relational abstractions: they forget relations among program variables

All the values for variables are abstracted indipendently

They proceed in two steps:

1. Collect the values a variables may take across a set of states

2. Over-approximate the set of values for each variable with an abstract element of a domain of value abstraction

Abstract states

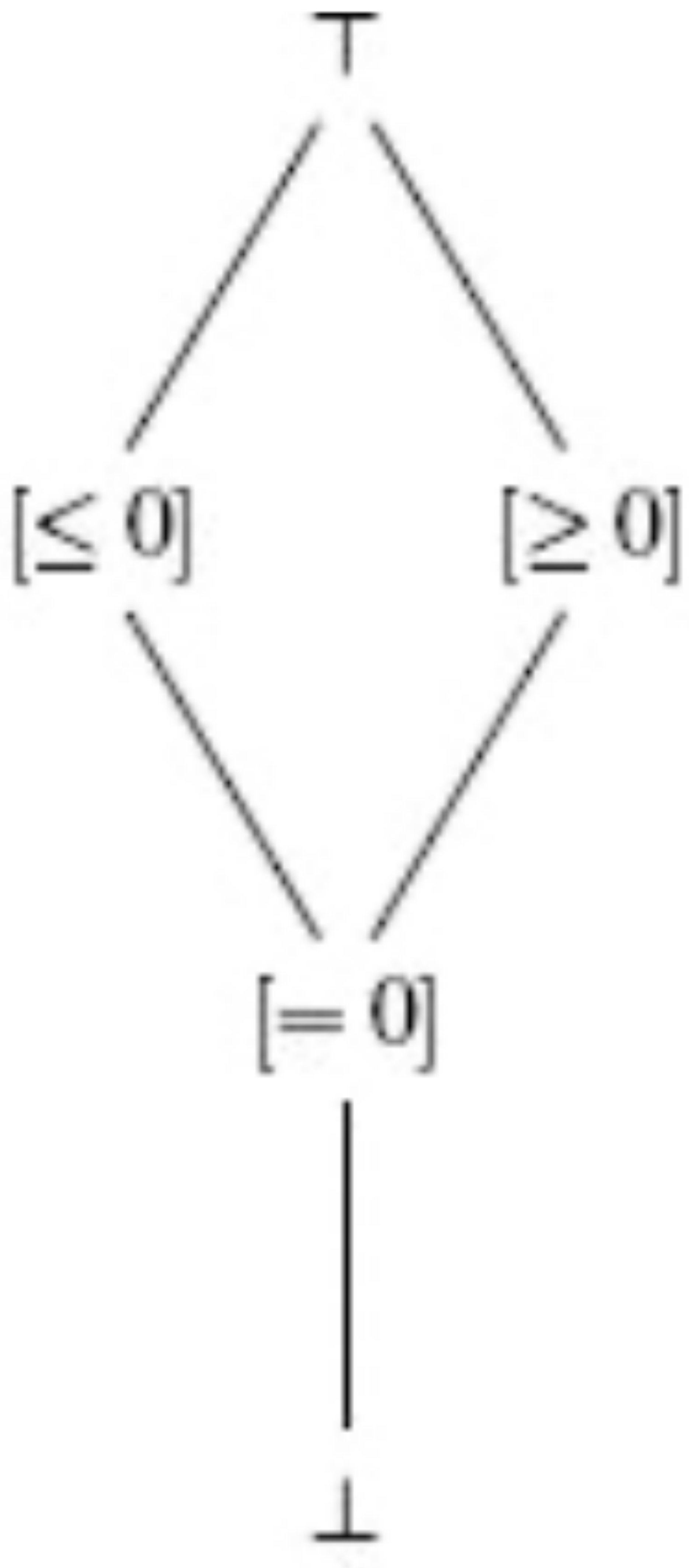
$$(\wp(\mathbb{M}), \subseteq) \xleftarrow[\alpha_M]{\gamma_M} \!\!\!\!\xrightarrow{} (\mathbb{M}^\sharp, \sqsubseteq_M)$$

$$M^\# \in \mathbb{M}^\# = \mathbb{X} \to \mathbb{V}^\#$$

$$\alpha_M(M)(x) = \alpha_V(\{m(x) \mid m \in M\})$$

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

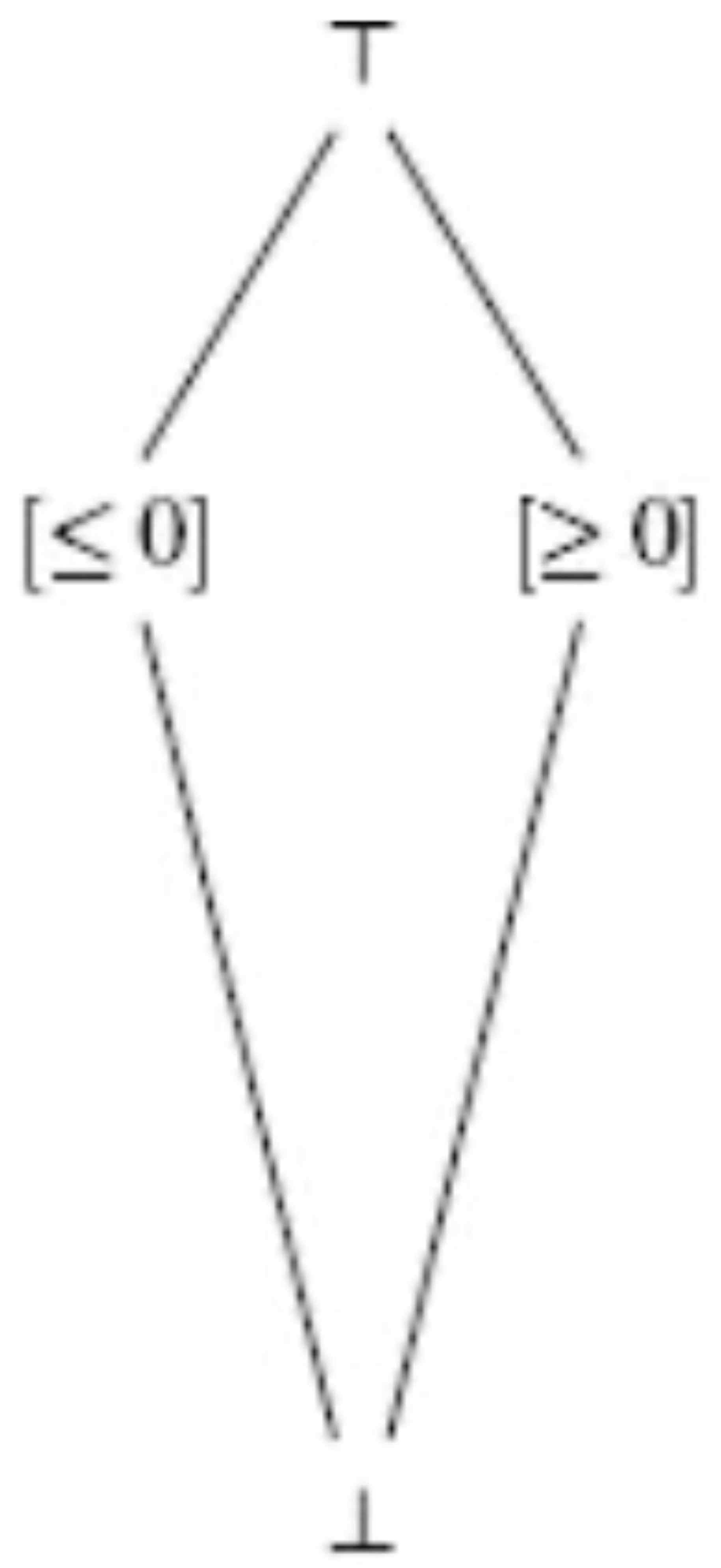
Signs



$$\gamma([\geq 0]) = \{n \in \mathbb{V} \mid n \geq 0\}$$
$$\gamma([\leq 0]) = \{n \in \mathbb{V} \mid n \leq 0\}$$
$$\gamma([= 0]) = \{0\}$$
$$\gamma(\top) = \mathbb{V}$$
$$\gamma(\bot) = \{\}$$

$$\alpha(\{2, 4, 8, 16, ..\}) = [\geq 0]$$
$$\alpha(\{0\}) = [0]$$
$$\alpha(\{-1, 1\}) = \top$$

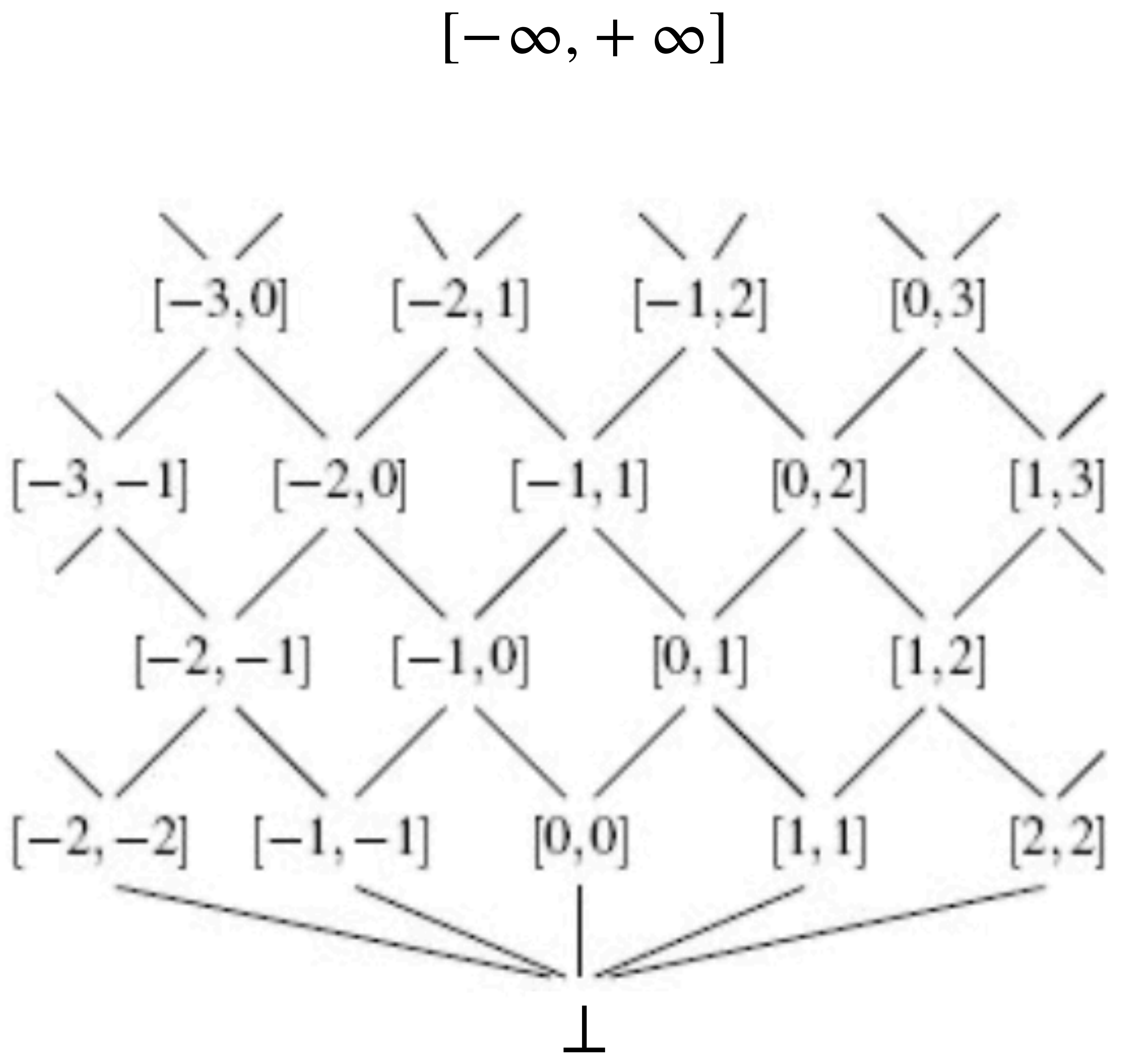
Variation of Signs



$$\gamma([\geq 0]) = \{n \in \mathbb{V} \mid n \geq 0\}$$
$$\gamma([\leq 0]) = \{n \in \mathbb{V} \mid n \leq 0\}$$
$$\gamma(\top) = \mathbb{V}$$
$$\gamma(\bot) = \{\}$$

There is no α since $\{0\} \models [\ \geq 0]$, $\{0\} \models [\ \leq 0]$ and $\{0\} \models \top$ but the smallest element does not exists

Intervals



Elements of *A*:

- $\perp$ the empty set of values
- $(n_0, n_1),\ n_0 \in (\mathbb{Z} \cup \{-\infty\}),\ \ n_1 \in (\mathbb{Z} \cup \{+\infty\}), n_0 \leq n_1$

$\sqsubseteq$ is the interval inclusion

$$\gamma(\perp) = \{\}$$
$$\gamma([n_0, n_1]) = \{\ n \in \mathbb{V}\ \mid n_0 \leq n \leq n_1\}$$
$$\gamma([-\infty, n_1]) = \{\ n \in \mathbb{V} \mid n \leq n_1\}$$
$$\gamma([n_0, +\infty]) = \{\ n \in \mathbb{V} \mid n_0 \leq n\}$$
$$\gamma([-\infty, +\infty]) = \mathbb{V}$$

$$\alpha(c) = \perp \text{ if } c = \emptyset,$$
$$\alpha(c) = [min(c), max(c)] \text{ if } c \neq \emptyset, min(c) \text{ and } max(c) \text{ exists}$$
$$\alpha(c) = [min(c), +\infty] \text{ if } c \neq \emptyset, min(c) \text{ exists}$$
$$\alpha(c) = [-\infty, max(c)] \text{ if } c \neq \emptyset, max(c) \text{ exists}$$
$$\alpha(c) = [-\infty, +\infty] \text{ otherwise}$$

Congruences

Elements of A:

- $\bot$ the empty set of values
- $(p\mathbb{Z}, n)$ with $p \in \mathbb{N}, n \in \mathbb{Z}$

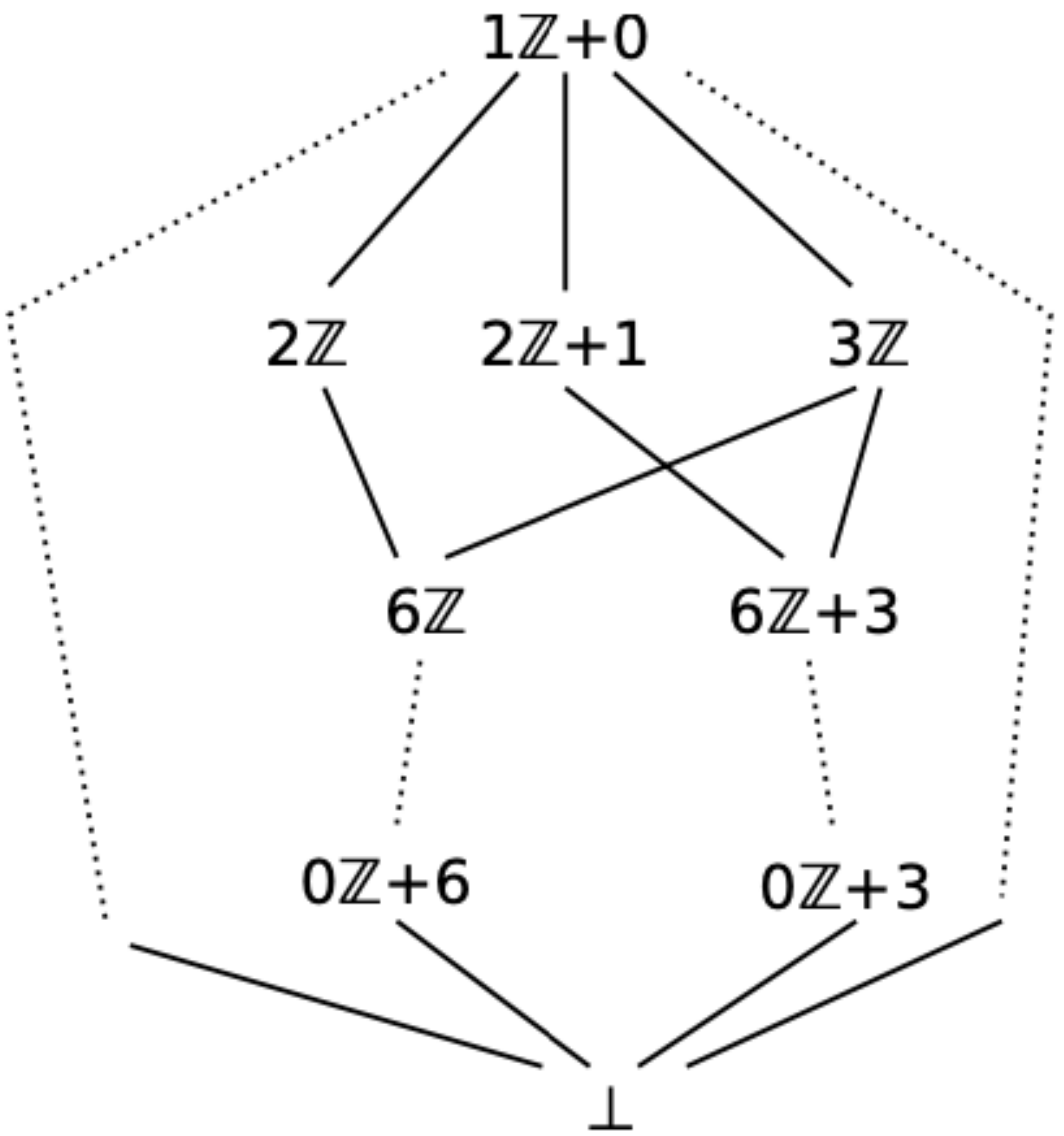
If $p \neq 0$ then $0 \leq n < p$

$$\gamma(\bot) = \{\}$$

$$\gamma((p\mathbb{Z}, n)) = \{\, pk + n \mid k \in \mathbb{Z}\}$$

the greatest element is $1\mathbb{Z} + 0$

singletons $\{c\}$ are represented as $0\mathbb{Z} + c$



Example

Consider the following set of memories M

m_0 :	$x \mapsto 25$	$y \mapsto 7$	$z \mapsto -12$
m_1 :	$x \mapsto 28$	$y \mapsto -7$	$z \mapsto -11$
m_2 :	$x \mapsto 20$	$y \mapsto 0$	$z \mapsto -10$
m_3 :	$x \mapsto 35$	$y \mapsto 8$	$z \mapsto -9$

With the Sign abstraction

$M^{\#}: \quad x \mapsto [\geq 0] \quad y \mapsto \top \quad z \mapsto [\leq 0]$

With the interval abstraction

$M^{\#}: \quad x \mapsto [25,35] \quad y \mapsto [-7,8] \quad z \mapsto [-12,-9]$

Example

Consider the following set of memories M

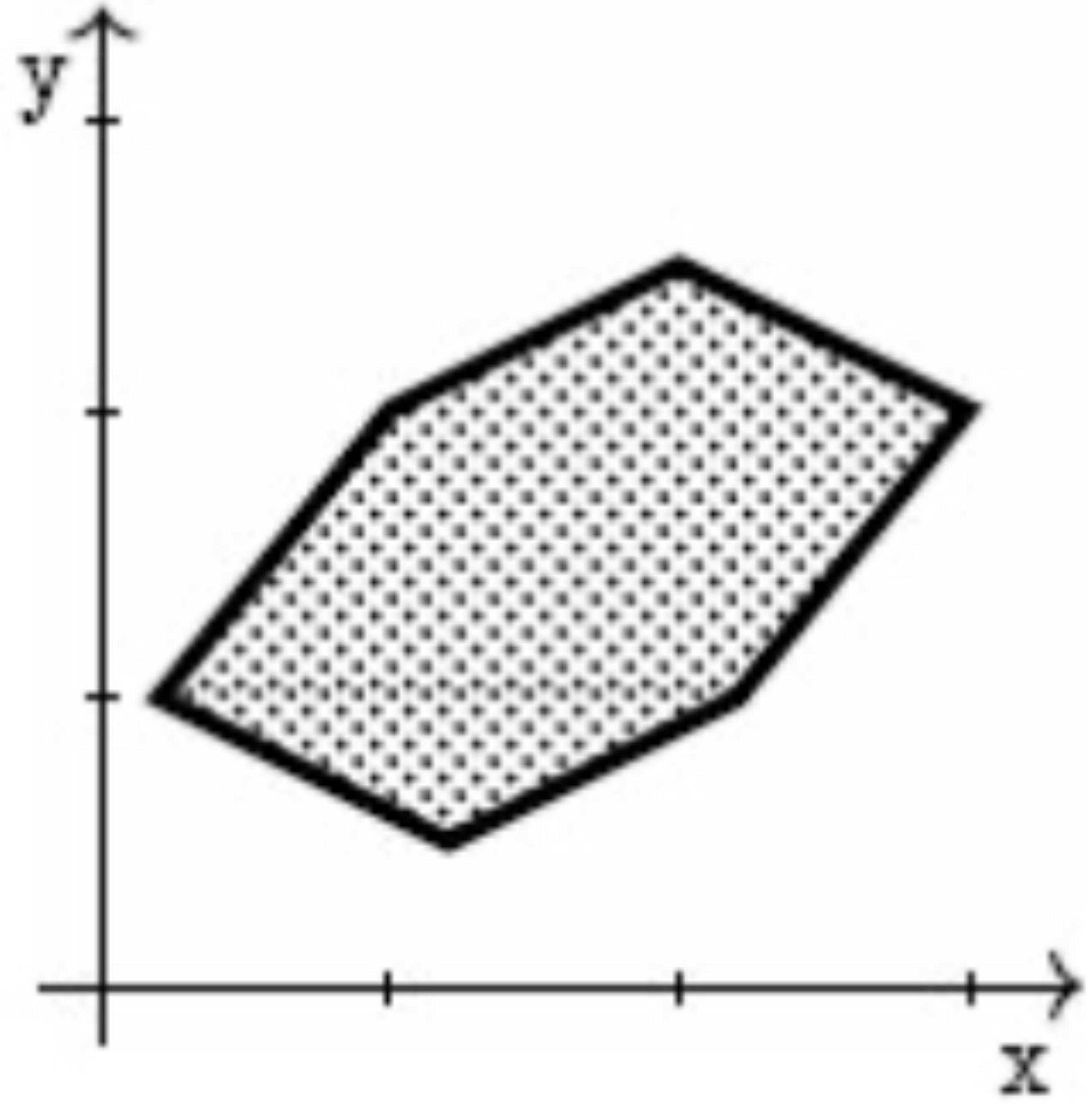
$$m_0 : \{x \mapsto 100, y \mapsto 201\}$$
$$m_1 : \{x \mapsto 1, y \mapsto 2\}$$
$$m_2 : \{x \mapsto 27, y \mapsto 55\}$$
$$m_3 : \{x \mapsto 30, y \mapsto 61\}$$
$$m_4 : \{x \mapsto 45, y \mapsto 91\}$$

A non relational domain is not able to model the relation between variables $y = 2x + 1$

With the interval abstraction

$$M^{\#} : \{x \mapsto [1, 100], y \mapsto [2, 201]\}$$

Relational domain
Convex Polyhedra domain



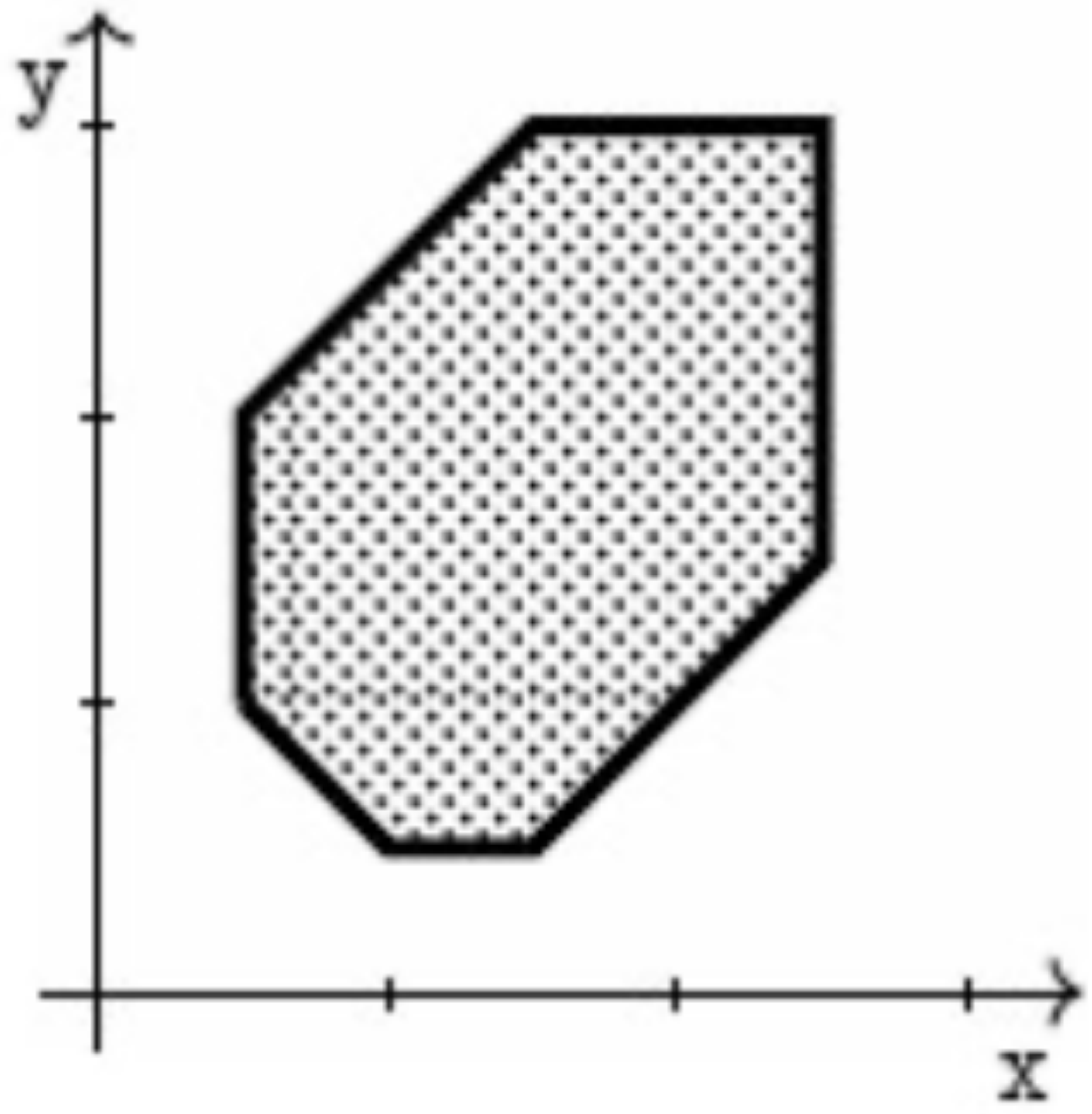
sets of numerical constraints of the form

$$c_1 x + c_2 y \leq c$$

(at most two variables per constraint,
with unit coefficients)

does not admit a best abstraction

Relational domain
Octagon domain



sets of numerical constraints of the form

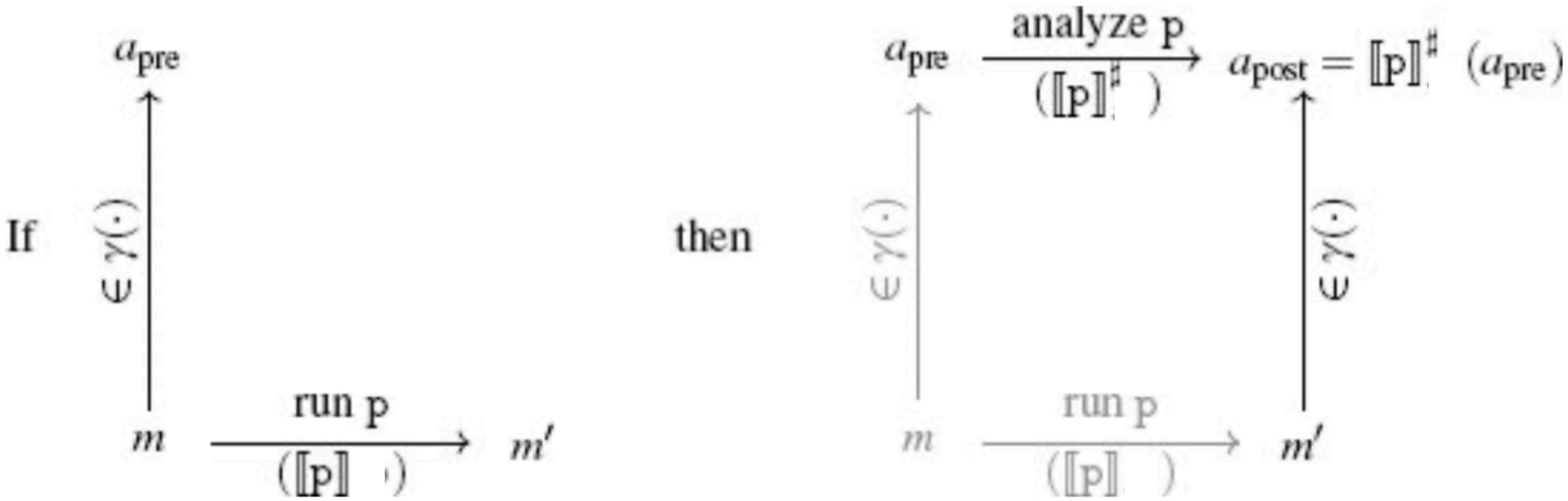
$$\pm x \pm y \leq c$$

(at most two variables per constraint, with unit coefficients)

A dmits the best abstraction

Step 3: Abstract semantics

We want to define a sound abstract semantics



Abstract semantics of command

It will defined by induction on the syntax

$$[\![C]\!]^{\#} \bot = \bot$$

$$[\![\texttt{skip}]\!]^{\#} M^{\#} = M^{\#}$$

$$[\![\texttt{C}_0 ; \texttt{C}_1]\!]^{\#} M^{\#} = [\![\texttt{C}_1]\!]^{\#} ([\![\texttt{C}_0]\!]^{\#} (M^{\#}))$$

This and all inductive construction relay on the following result:
Let
$F_0, F_1 : \wp(\mathbb{M}) \to \wp(\mathbb{M})$
$F_0^{\#}, F_1^{\#} : \mathbb{A} \to \mathbb{A}$.
If $F_i \gamma \subseteq \gamma F_i^{\#}$,
then
$F_0 F_1 \gamma \subseteq \gamma F_0^{\#} F_1^{\#}$

Abstract interpretation of expressions

$$[\![\mathtt{E}]\!]^\# : \mathbb{M}^\# \to \mathbb{V}^\#$$

$$[\![\mathtt{n}]\!]^\# M^\# = \alpha(\{n\})$$

$$[\![\mathtt{x}]\!]^\# M^\# = M^\#(x)$$

$$[\![\mathtt{E}_0 + \mathtt{E}_1]\!]^\# M^\# = [\![\mathtt{E}_0]\!]^\# +^\# [\![\mathtt{E}_1]\!]^\#$$

Sign domain

$$[\geq 0] +^\# [\leq 0] = \top$$

$$[\geq 0] +^\# [\geq 0] = [\geq 0]$$

Interval domain

$$[0, 6] +^\# [-2, 3] = [-2, 9]$$

$$[-\infty, -2] +^\# [4, 18] = [-\infty, 16]$$

Analysis of assignment

$$[\![\texttt{x} := \texttt{E}]\!]^{\#} M^{\#} = \ M^{\#}[x \mapsto ([\![\texttt{E}]\!]^{\#}(M^{\#}))]$$

$$[\![\texttt{input}(x)]\!]^{\#} M^{\#} = \ M^{\#}[x \mapsto \top]$$

Sign domain

$$]\!]x := x + 6 + y]\!]\{x \mapsto [\geq 0], y \mapsto \top\}$$
$$= \{x \mapsto \top, y \mapsto \top\}$$

Interval domain

$$]\!]x := x + 6 + y]\!]\{x \mapsto [3, 8], y \mapsto [-3, 5]\}$$
$$= \{x \mapsto [6, 19], y \mapsto [-3, 5]\}$$

Abstract interpretation of the conditional branching

$$[\![\, \texttt{if}\ \ (B)\{\texttt{C}_0\}\ \ \texttt{else}\ \ \{\texttt{C}_1\}]\!](M) = [\![\texttt{C}_0]\!]\, \mathcal{F}_B(M) \cup [\![\texttt{C}_1]\!]\, \mathcal{F}_{\neg B}(M)$$

We use the compositional principle and we need to define over approximations of

- $\mathscr{F}_B$ and of $\mathscr{F}_{\neg B}$
- the join operator ∪

Analysis of conditions

For all $M^{\#}$, $\mathscr{F}_B(\gamma(M^{\#})) \subseteq \gamma(\mathscr{F}^{\#}_B(M^{\#}))$

Sign domain

$$\mathscr{F}^{\sharp}_{\mathrm{x}<0}(M^{\sharp}) = \begin{cases} (\mathrm{y} \in \mathbb{X}) \longmapsto \bot & \text{if } M^{\sharp}(\mathrm{x}) = [\geq 0] \text{ or } [=0] \text{ or } \bot \\ M^{\sharp}[\mathrm{x} \mapsto [\leq 0]] & \text{if } M^{\sharp}(\mathrm{x}) = [\leq 0] \text{ or } \top \end{cases}$$

Interval domain

$$\mathscr{F}^{\sharp}_{\mathrm{x}<n}(M^{\sharp}) = \begin{cases} (\mathrm{y} \in \mathbb{X}) \longmapsto \bot & \text{if } a > n \\ M^{\sharp}[\mathrm{x} \mapsto [a, n]] & \text{if } a \leq n \leq b \\ M^{\sharp} & \text{if } b \leq n \end{cases}$$

Analysis of conditions

```
if(x > 7){
    y := x - 7
}else{
    y := 7 - x
}
```

Interval domain

$$\mathcal{F}^{\#}_{x>7}(\{x \mapsto \top, y \mapsto \top\}) = \{x \mapsto [8, +\infty], y \mapsto \top\}$$
$$\mathcal{F}^{\#}_{x\leq 7}(\{x \mapsto \top, y \mapsto \top\}) = \{x \mapsto [-\infty, 7], y \mapsto \top\}$$

Analysis of flow joins

We need to define a correct over approximation of the join ∪, that is, an abstract join $\cup^{\#}$ s.t.

$$\gamma(M_0^\sharp) \cup \gamma(M_1^\sharp) \subseteq \gamma(M_0^\sharp \sqcup^\sharp M_1^\sharp)$$

$$\begin{array}{rcl} M_0^\sharp & = & \{\mathrm{x} \mapsto [0,3], \mathrm{y} \mapsto [6,7], \mathrm{z} \mapsto [4,8]\} \\ M_1^\sharp & = & \{\mathrm{x} \mapsto [5,6], \mathrm{y} \mapsto [0,2], \mathrm{z} \mapsto [6,9]\} \end{array}$$

For the interval domain is defined in terms of min and max of intervals

$$M_0^\sharp \sqcup^\sharp M_1^\sharp \quad = \quad \{\mathrm{x} \mapsto [0,6], \mathrm{y} \mapsto [0,7], \mathrm{z} \mapsto [4,9]\}$$

Analysis of Conditional Command

$$[\![\texttt{if } (B)\{\texttt{C}_0\} \texttt{ else } \{\texttt{C}_1\}]\!]^\#(M^\#) = [\![\texttt{C}_0]\!]^\# \mathcal{F}^\#_B(M) \cup^\# [\![\texttt{C}_1]\!]^\# \mathcal{F}^\#_{\neg B}(M^\#)$$

```
if(x > 7){
    y:=x - 7
}else{
    y:=7 - x
}
```

Starting with $\{x \mapsto \top, y \mapsto \top\}$

on the true branch we filter for condition $x > 7$

$$\mathcal{F}^\#_{x>7}(\{x \mapsto \top, y \mapsto \top\}) = \{x \mapsto [8, +\infty], y \mapsto \top\}$$

$$[\![\texttt{y:=x-7}]\!]^\#(\{x \mapsto [8, +\infty], y \mapsto \top\}) = \{x \mapsto [8, +\infty], y \mapsto [1, +\infty]\}$$

on the false branch we filter for condition $x \leq 7$

$$\mathcal{F}^\#_{x\leq 7}(\{x \mapsto \top, y \mapsto \top\}) = \{x \mapsto [-\infty, 7], y \mapsto \top\}$$

$$[\![\texttt{y:=7-x}]\!]^\#(\{x \mapsto [-\infty, 7], y \mapsto \top\}) = \{x \mapsto [-\infty, 7], y \mapsto [0, +\infty]\}$$

Applying the abstract join we obtain $\{x \mapsto \top, y \mapsto [0, +\infty]\}$

Abstract interpretation of the loop

Recall the concrete semantics of the loop

For F= $[\![C]\!]\mathcal{F}_B$

$$[\![\text{while}(B)\{C\}]\!](M) = \mathcal{F}_{\neg B}(\bigcup_{i\geq 0}(\,[\![C]\!]\,\mathcal{F}_B)^i(M)) = \mathcal{F}_{\neg B}(\bigcup_{i\geq 0} F^i(M))$$

We can approximate $\mathscr{F}_B$ and F so the problem we need to solve is how to compute an approximation of an infinite union $\bigcup_{i>0} F^i(M)$

Concrete iterations

$$M_n = \bigcup_{i=0}^{n} F^i(M)$$

$$M_0 = M$$

$$M_{k+1} = M_k \cup F(M_k)$$

Abstract iterations

$$M_0^\# = M^\#$$

$$M_{k+1}^\# = M_k^\# \cup F^\#(M_k^\#)$$

Abstract iterations

```
x := 0;
while(x ≥ 0){
    x := x + 1
}
```

After the first assignment we have $M^{\#} = \{x \mapsto [0,0]\}$

$$
\begin{aligned}
M_0^\sharp &= \{x \mapsto [0,0]\} \\
M_1^\sharp &= \{x \mapsto [0,1]\} \\
M_2^\sharp &= \{x \mapsto [0,2]\} \\
\vdots &= \vdots \\
M_n^\sharp &= \{x \mapsto [0,n]\} \\
\vdots &= \vdots
\end{aligned}
$$

```
x := 0;
while(x ≤ 100){
    if(x ≥ 50){
        x := 10
    }else{
        x := x + 1
    }
}
```

$$
\begin{aligned}
M_0^\sharp &= \{x \mapsto [0,0]\} \\
M_1^\sharp &= \{x \mapsto [0,1]\} \\
M_2^\sharp &= \{x \mapsto [0,2]\} \\
\vdots &= \vdots \\
M_{49}^\sharp &= \{x \mapsto [0,49]\} \\
M_{50}^\sharp &= \{x \mapsto [0,50]\} \\
M_{51}^\sharp &= \{x \mapsto [0,50]\} \\
M_{52}^\sharp &= \{x \mapsto [0,50]\} \\
\vdots &= \vdots
\end{aligned}
$$

Convergence of iterates

The computation of abstract iterations may not converge or it can converge too slowly

We can choose to use finite Height Domain

We can design widening operators

Finite height lattices

If the abstract domain has finite height the abstract iterations are finite

$$
\begin{array}{l}
\texttt{abs\_iter}(F^\sharp, M^\sharp)\\
\quad R \leftarrow M^\sharp;\\
\quad \text{repeat}\\
\qquad T \leftarrow R;\\
\qquad R \leftarrow R \sqcup^\sharp F^\sharp(R);\\
\quad \text{until } R = T\\
\quad \text{return } M^\sharp_{\lim} = T;
\end{array}
$$

$$
\begin{array}{l}
x := 0;\\
\textbf{while}(x \geq 0)\{\\
\qquad x := x + 1\\
\}
\end{array}
$$

$$
\begin{array}{rcl}
M^\sharp_0 & = & \{x \mapsto [=0]\}\\
M^\sharp_1 & = & \{x \mapsto [\geq 0]\}\\
M^\sharp_2 & = & \{x \mapsto [\geq 0]\}
\end{array}
$$

$$
\begin{array}{l}
x := 0;\\
\textbf{while}(x \leq 100)\{\\
\quad \textbf{if}(x \geq 50)\{\\
\qquad x := 10\\
\quad \}\textbf{else}\{\\
\qquad x := x + 1\\
\quad \}\\
\}
\end{array}
$$

Widening operator

Definition *A* widening operator over an abstract domain is a binary operator s.t.

- it holds $\gamma(a_0) \cup \gamma(a_1) \subseteq \gamma(a_0 \nabla a_1)$

- for any sequence $(a_n)_{n\in\mathbb{N}}$, the sequence $(a'_n)_{n\in\mathbb{N}}$ defined as follows is ultimately stationary:

$$a'_0 = a_0$$
$$a'_{n+1} = a'_n \nabla a_n$$

Widening operator for intervals

$$[n,p] \mathbin{\nabla_{\mathcal{Y}}} [n,q] = \begin{cases} [n,p] & \text{if } p \geq q \\ [n,+\infty) & \text{if } p < q \end{cases}$$

The same for the other bound

The abstract iterations become

$\texttt{abs\_iter}(F^\sharp, M^\sharp)$

$\mathrm{R} \leftarrow M^\sharp;$

repeat

$\mathrm{T} \leftarrow \mathrm{R};$

$\mathrm{R} \leftarrow \mathrm{R} \mathbin{\nabla} F^\sharp(\mathrm{R});$

until $\mathrm{R} = \mathrm{T}$

return $M^\sharp_{\mathrm{lim}} = \mathrm{T};$

Example

```
x := 0;
while(x ≥ 0){
    x := x + 1
}
```

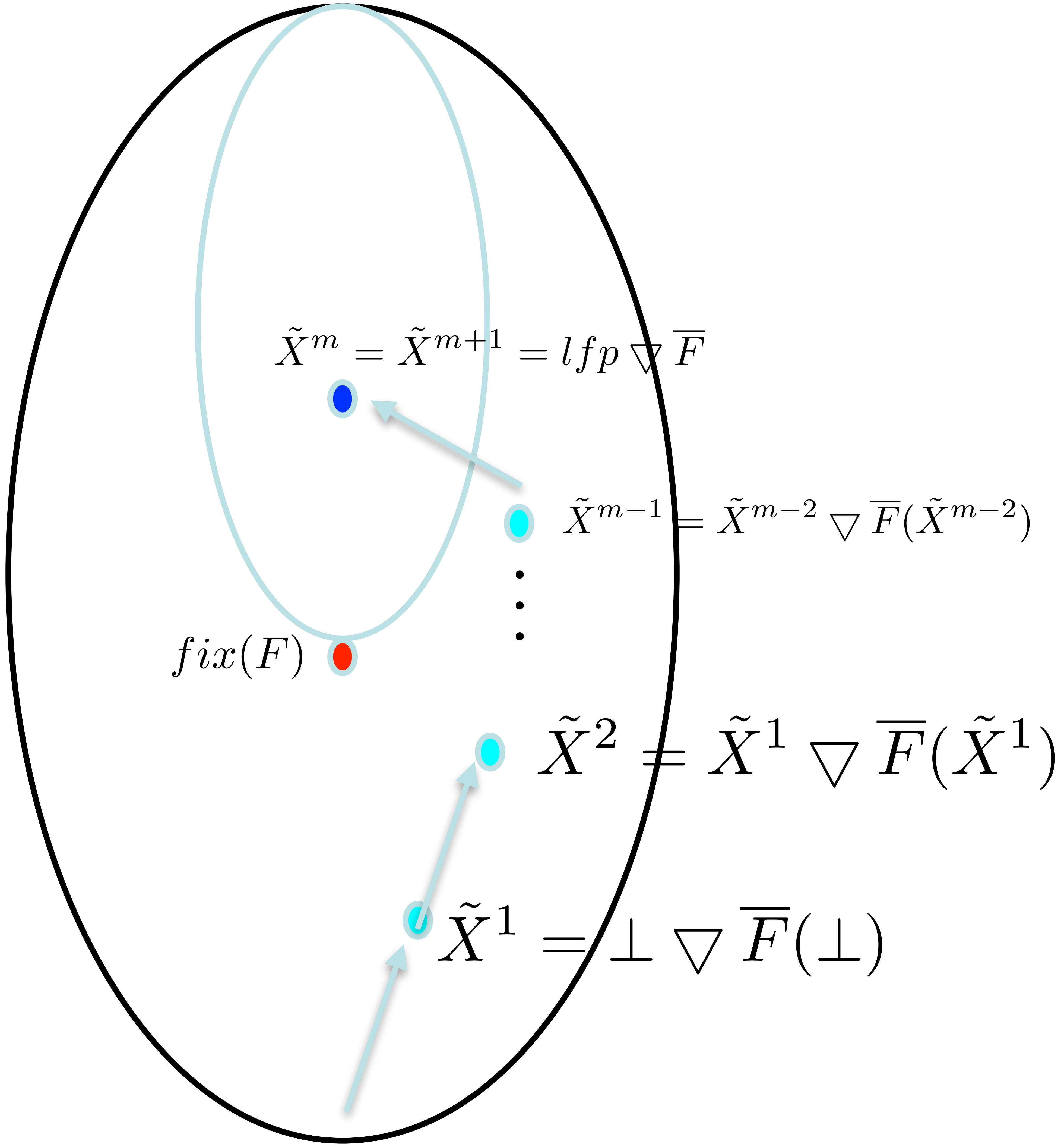
$$M_0^\# = \{x \mapsto [0,0]\}$$
$$M_1^\# = \{x \mapsto [0,+\infty]\}$$
$$M_2^\# = \{x \mapsto [0,+\infty]\}$$

Stable! Not very precise

```
x := 0;
while(x ≤ 100){
    if(x ≥ 50){
        x := 10
    }else{
        x := x + 1
    }
}
```

Widening

$$\tilde{X}^m \sqsupseteq \overline{F}(\tilde{X}^m)$$



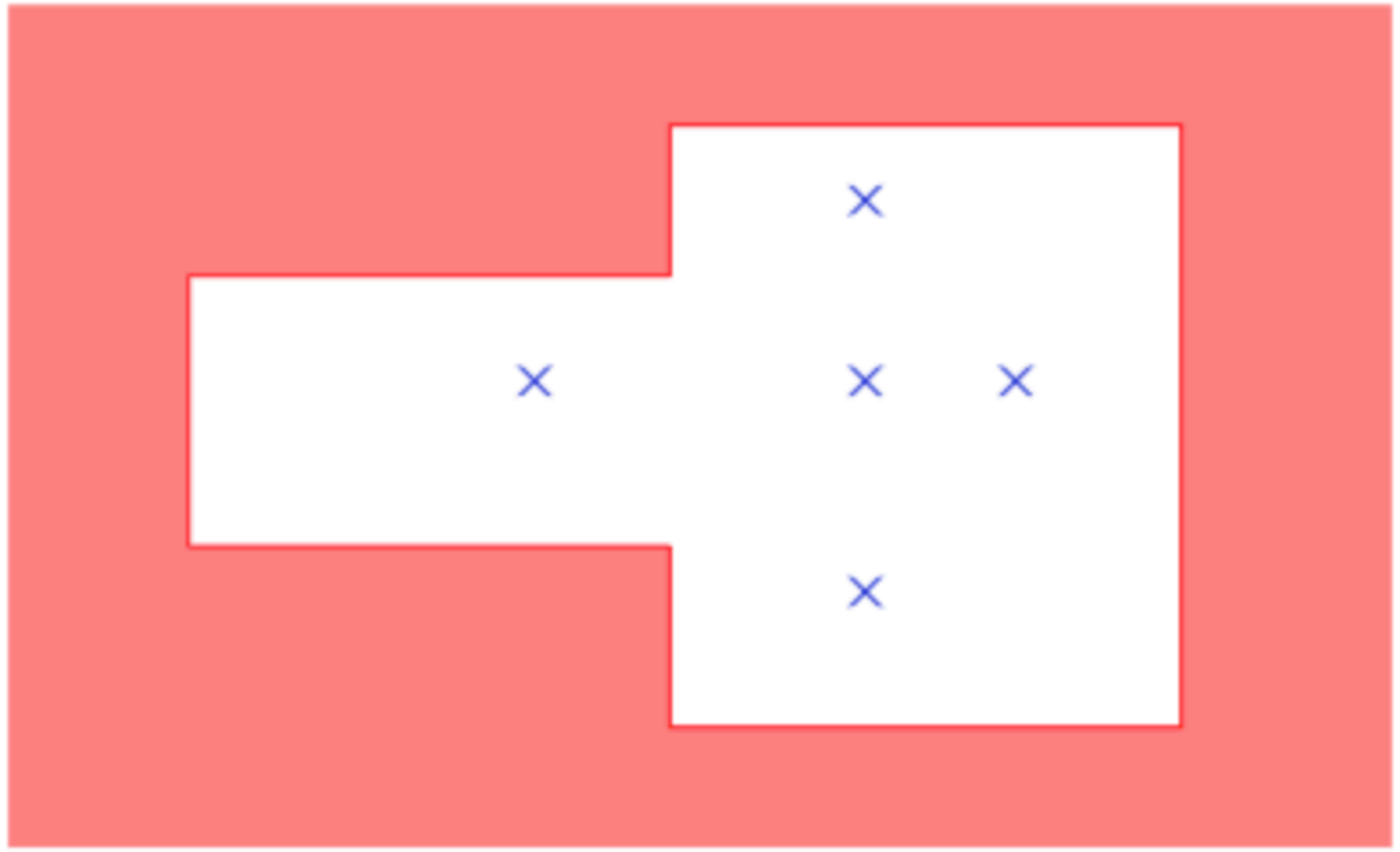
The analysis

$$[\![\mathtt{n}]\!]^{\#} M^{\#} = \alpha(\{n\})$$
$$[\![\mathtt{x}]\!]^{\#} M^{\#} = M^{\#}(x)$$
$$[\![\mathtt{E}_0 + \mathtt{E}_1]\!]^{\#} M^{\#} = [\![\mathtt{E}_0]\!]^{\#} +^{\#} [\![\mathtt{E}_1]\!]^{\#}$$
$$[\![\mathtt{x} := \mathtt{E}]\!]^{\#} M^{\#} = M^{\#}[x \mapsto ([\![\mathtt{E}]\!]^{\#}(M^{\#}))]$$
$$[\![\mathtt{input}(x)]\!]^{\#} M^{\#} = M^{\#}[x \mapsto \top]$$
$$[\![\mathtt{if}\ (B)\{\mathtt{C}_0\}\ \mathtt{else}\ \{\mathtt{C}_1\}]\!]^{\#}(M^{\#}) = [\![\mathtt{C}_0]\!]^{\#}\ \mathcal{F}^{\#}_{B}(M) \cup^{\#} [\![\mathtt{C}_1]\!]^{\#}\ \mathcal{F}^{\#}_{\neg B}(M^{\#})$$
$$[\![\mathrm{while}(B)\{C\}]\!]^{\#}(M^{\#}) = \mathcal{F}^{\#}_{\neg B}(\mathtt{abs\_iter}(\ [\![C]\!]^{\#}\ \mathcal{F}^{\#}_{B}, M^{\#}))$$

Design the widening ∇

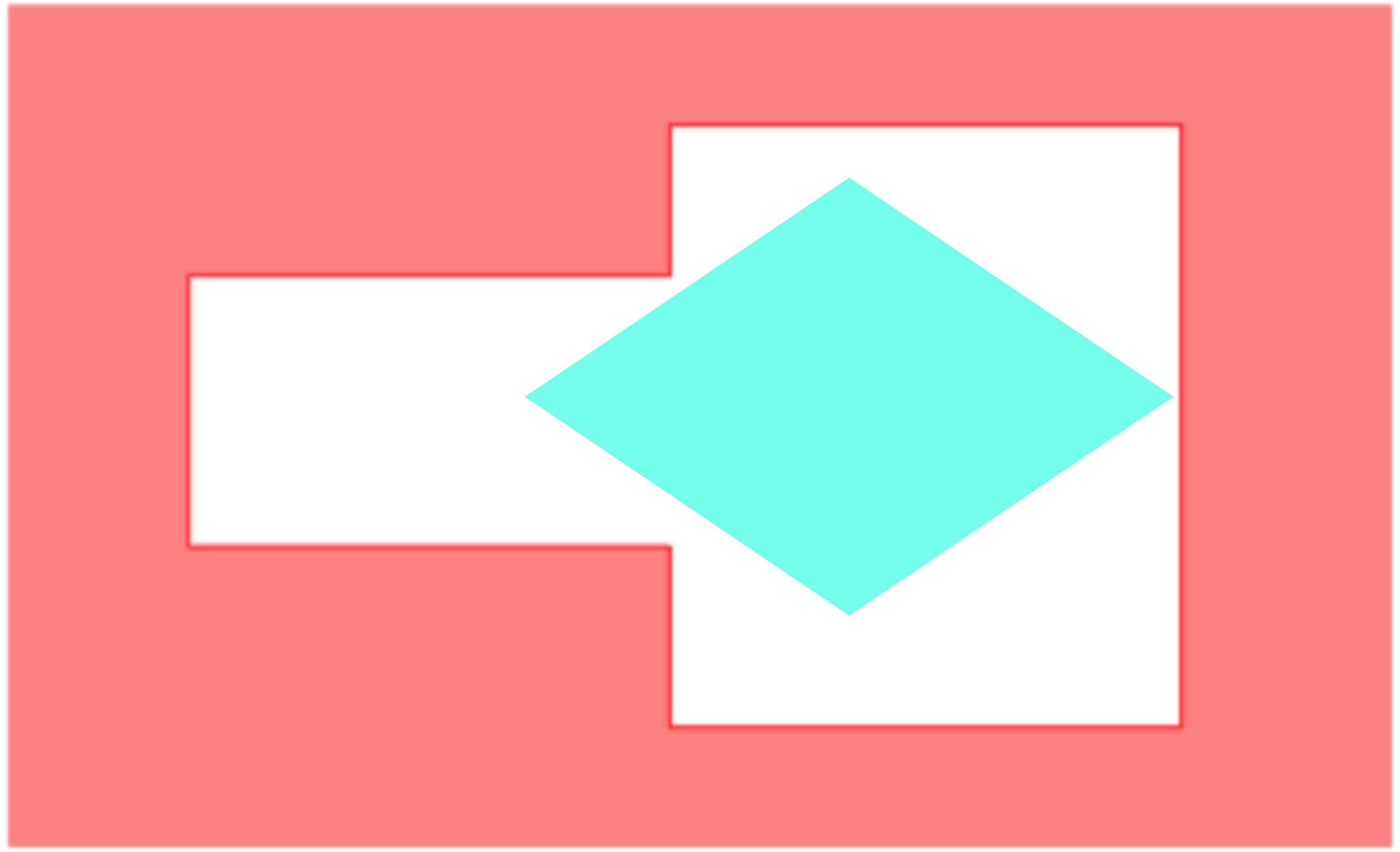
Theorem The computation of $[\![\mathtt{C}]\!]^{\#} M^{\#}$ terminates and $[\![\mathtt{C}]\!]\gamma(M^{\#}) \subseteq \gamma([\![\mathtt{C}]\!]^{\#}(M^{\#}))$

Using analysis'results



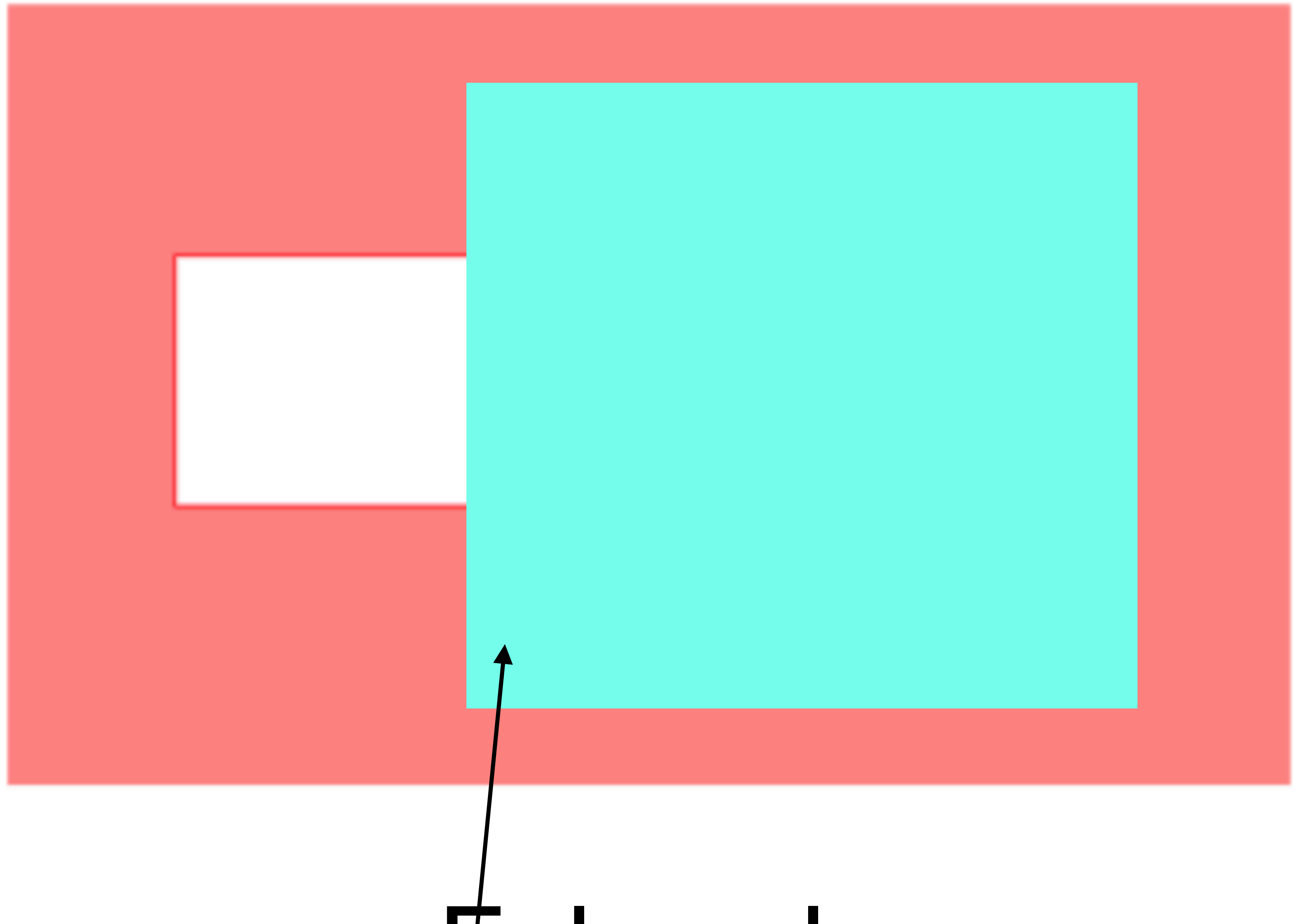
The program is correct

Using analysis’results



The program is correct and
our approximation can prove it

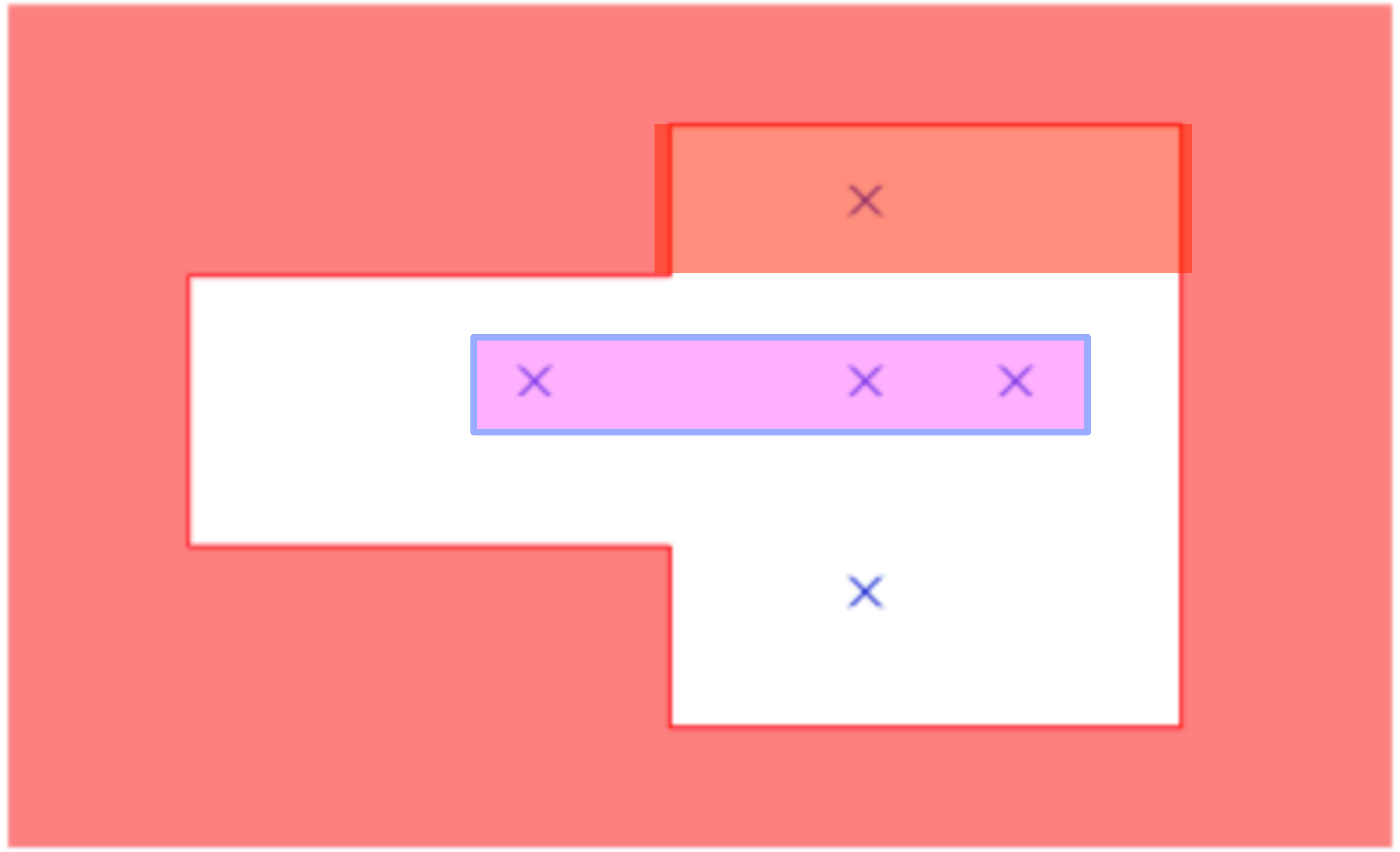
Using analysis'results



False alarm

The program is correct and
our approximation can't prove it

Unsound analysis



The program is not correct and
our approximation says it is correct

Trace-based operational semantics

```
p0 : while isEven(x) {
       p1 : x = x div 2;
     }
p2 : x = 4 * x;
p3 : exit
```

The operational semantics updates a program-point, storage-cell pair, pp, x, using these four transition rules:

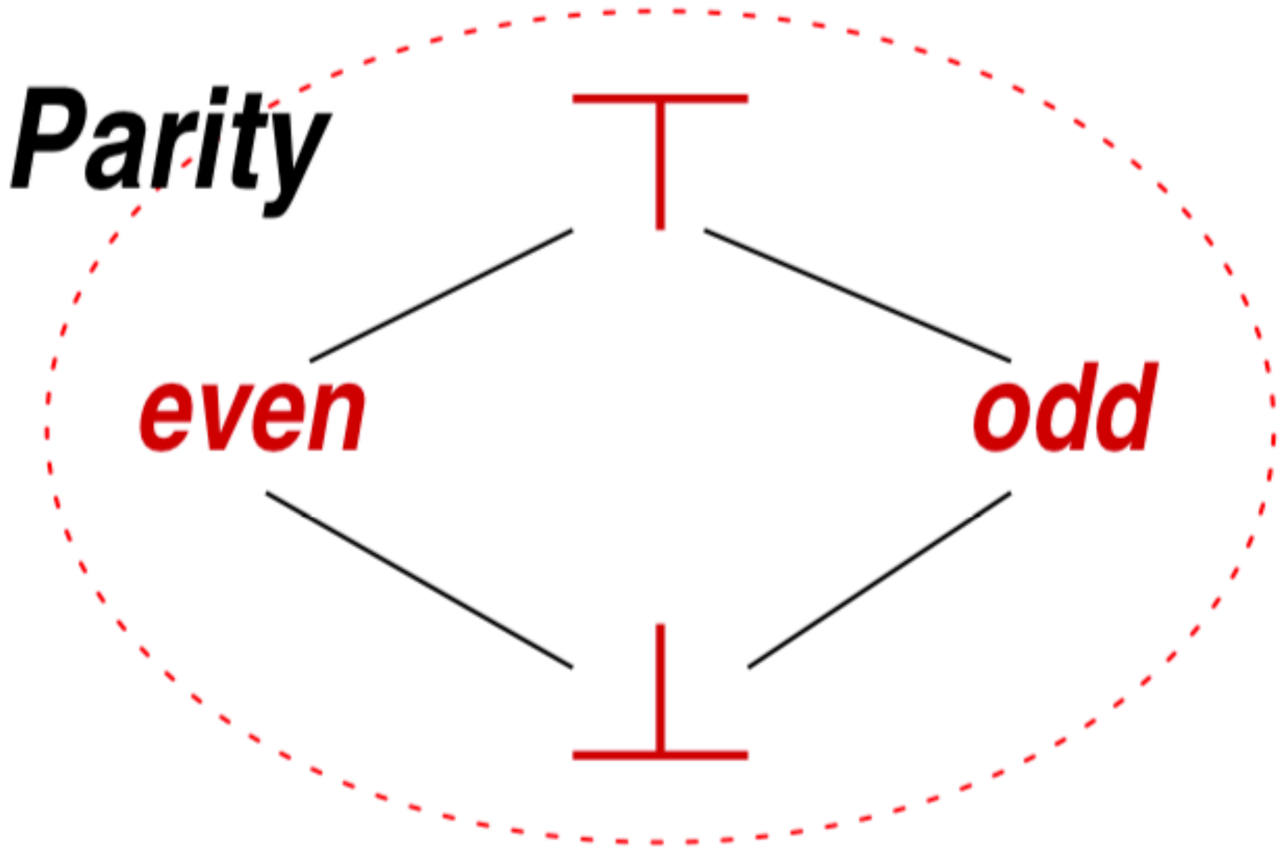
$$p_0, 2n \longrightarrow p_1, 2n \qquad p_1, n \longrightarrow p_0, n/2$$

$$p_0, 2n+1 \longrightarrow p_2, 2n+1 \qquad p_2, n \longrightarrow p_3, 4n$$

A program's operational semantics is written as a trace:

$$p_0, 12 \longrightarrow p_1, 12 \longrightarrow p_0, 6 \longrightarrow p_1, 6 \longrightarrow p_0, 3 \longrightarrow p_2, 3 \longrightarrow p_3, 12$$

The parity domain



$\gamma : \text{Parity} \to \mathcal{P}(\text{Int})$

$\gamma(\text{even}) = \{..., -2, 0, 2, ...\}$

$\gamma(\text{odd}) = \{..., -1, 1, 3, ...\}$

$\gamma(\top) = \text{Int}, \quad \gamma(\bot) = \{\,\}$

$\alpha : \mathcal{P}(\text{Int}) \to \text{Parity}$

$\alpha(S) = \sqcup\{\beta(v) | v \in S\}$, where $\beta(2n) = \text{even}$ and $\beta(2n+1) = \text{odd}$

The abstract transition rules are synthesized from the orginals:

$$p_i, a \longrightarrow p_j, \alpha(v'), \text{ if } v \in \gamma(a) \text{ and } p_i, v \longrightarrow p_j, v'$$

This recipe ensures that every transition in the original, “concrete” semantics is simulated by one in the abstract semantics.

The abstraction rules

$p_0, 2n \longrightarrow p_1, 2n$

$p_0, 2n+1 \longrightarrow p_2, 2n+1$

$p_1, n \longrightarrow p_0, n/2$

$p_2, n \longrightarrow p_3, 4n$

```
p0 : while isEven(x) {
        p1 : x = x div 2;
     }
p2 : x = 4 * x;
p3 : exit
```

$p_0, even \longrightarrow p_1, even$

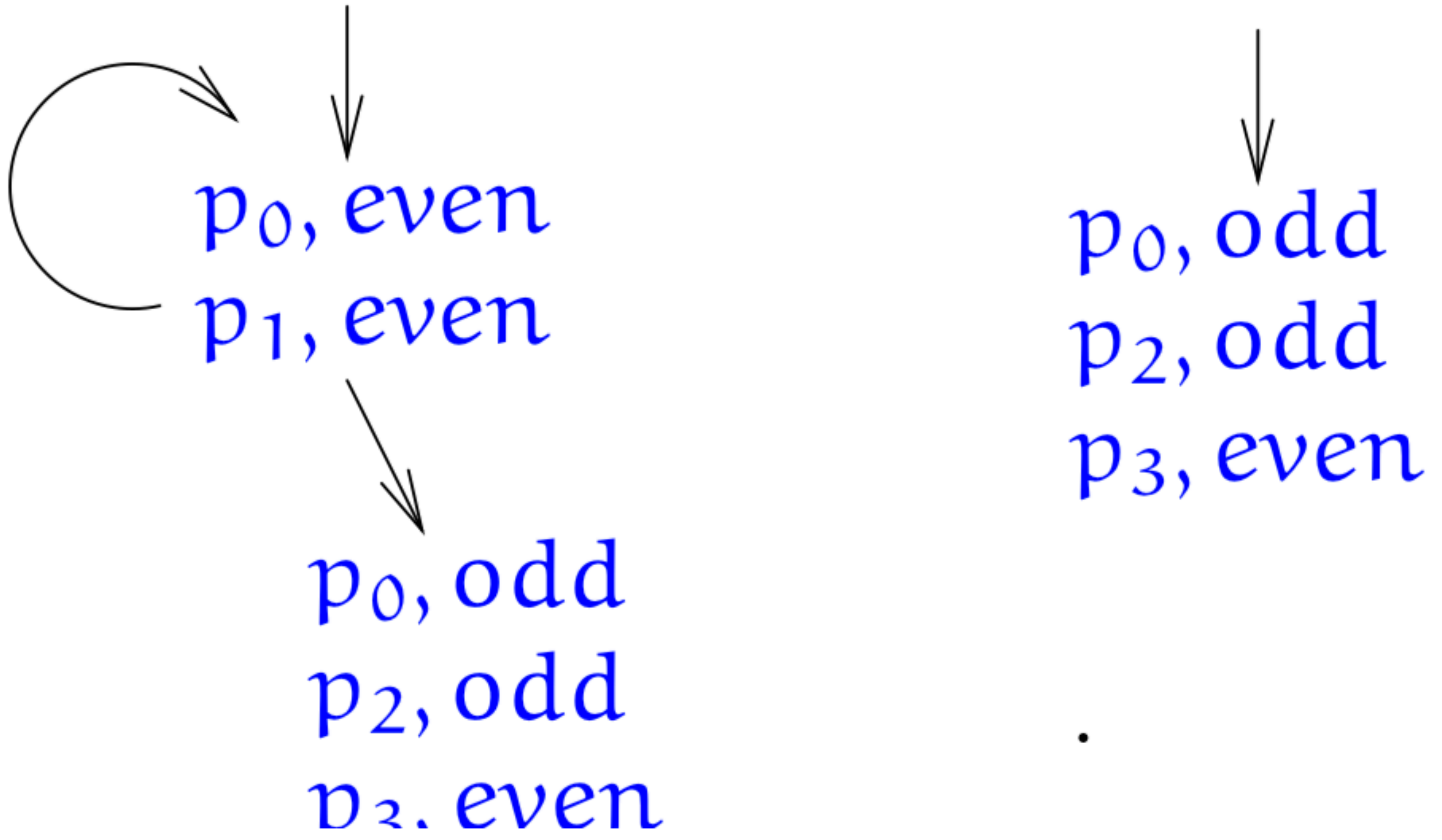
$p_0, odd \longrightarrow p_2, odd$

$p_1, even \longrightarrow p_0, even$

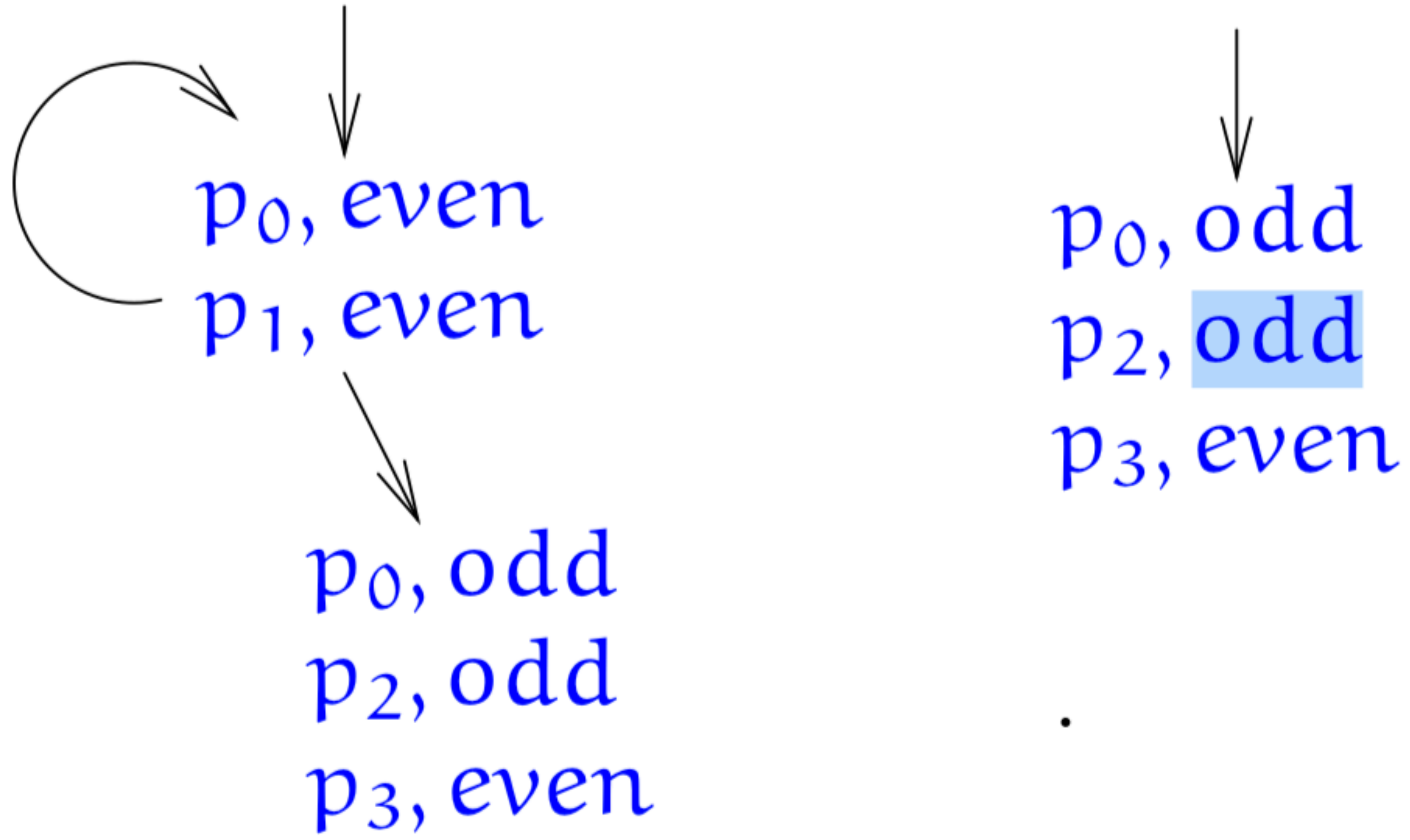
$p_1, even \longrightarrow p_0, odd$

$p_2, a \longrightarrow p_3, even$

Two trace trees cover the full range of inputs:



The interpretation of the program's semantics with the abstract values is an *abstract interpretation*:



We conclude that

- if the program terminates, `x` is even-valued
- if the input is odd-valued, the loop body, p_1, will not be entered

Due to the loss of precision, we can not decide termination for almost all the even-valued inputs. (Indeed, only 0 causes nontermination.)

Another example: array bounds using intervals

Integer variables receive values from the *interval domain*,

$$I = \{[i, j] \mid i, j \in \text{Int} \cup \{-\infty, +\infty\}\}.$$

We define $[a, b] \sqcup [a', b'] = [\min(a, a'), \max(b, b')]$.

```
int a = new int[10];
i = 0;
while (i < 10)  {
      ... a[i] ...
      i = i + 1;
}
```

$i = [0,0]$

p_1

$i = [0,0] \sqcap [-\infty,9] = [0,0]$

$i = [0,0] \sqcup [1,1] \sqcap [-\infty,9] = [0,1]$

...

p_2

$i = [1,1]$

$i = [1,1] \sqcup [2,2] = [1,2]$

...

At convergence, `i`'s ranges are

at p_1 : $[0..9]$

at p_2 : $[1..10]$

at loop exit : $[1..10] \sqcap [10, +\infty] = [10, 10]$

Constant Propagation analysis

```
p0 :  x = 1; y = 2;
p1 :  while (x < y + z)
        p2 : x = x + 1;
      }
p3 :  exit
```

where $m + n$ is interpreted

$$k_1 + k_2 \longrightarrow sum(k_1, k_2),$$

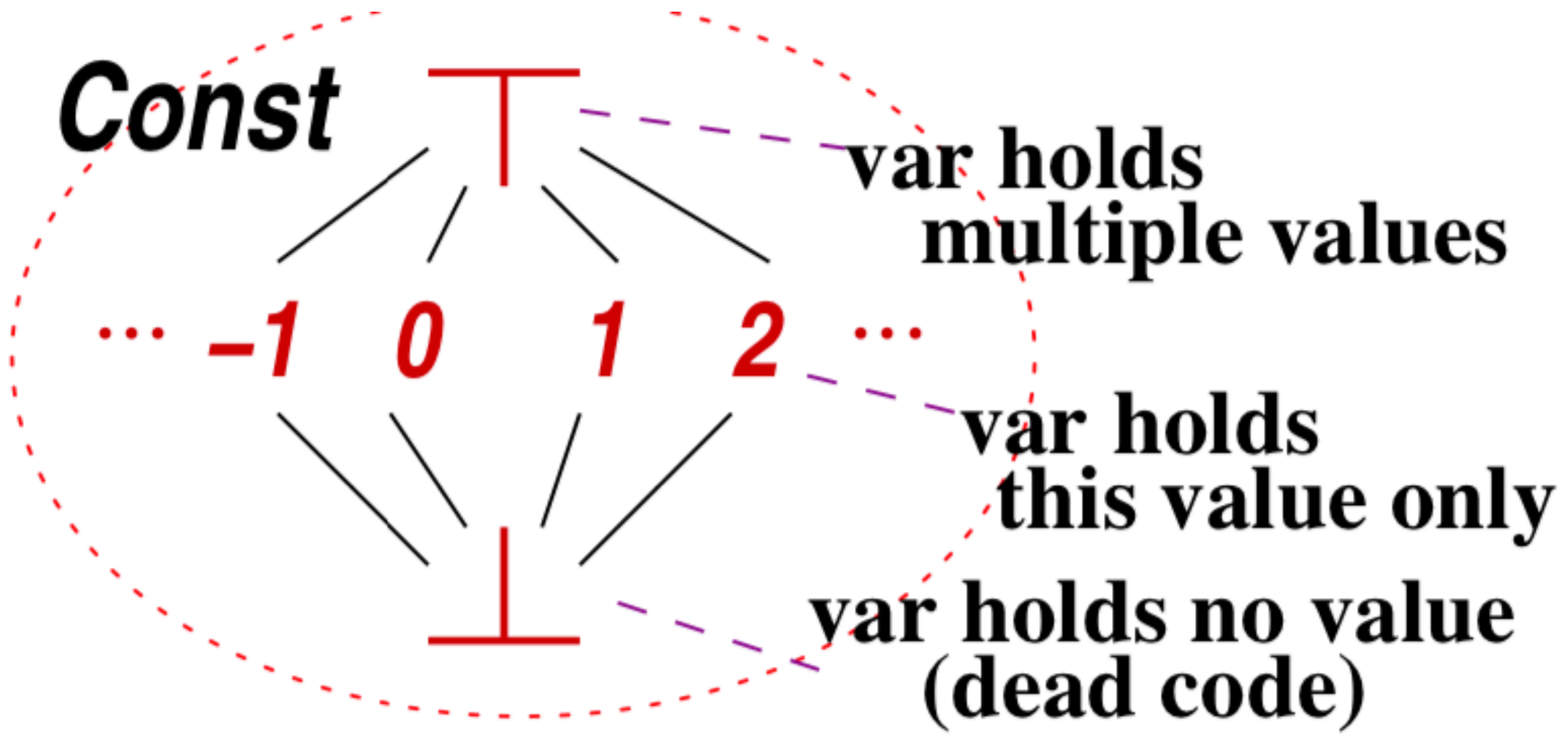
$$\top \neq k_i \neq \bot, i \in 1..2$$

$$\top + k \longrightarrow \top$$

$$k + \top \longrightarrow \top$$

Let $\langle u, v, w \rangle$ abbreviate

$\langle x : u, y : v, z : w \rangle$



Abstract trace:

$p_0, \langle \top, \top, \top \rangle$

$p_1, \langle 1, 2, \top \rangle$ → $p_3, \langle 1, 2, \top \rangle$

↓

$p_2, \langle 1, 2, \top \rangle$

$p_1, \langle 2, 2, \top \rangle$ → $p_3, \langle 2, 2, \top \rangle$

↓

$p_2, \langle 2, 2, \top \rangle$

$p_1, \langle 3, 2, \top \rangle$ → ...

↓

...

An acceleration is needed for finite convergence

$p_0, \langle \top, \top, \top \rangle$

$p_1, \langle 1, 2, \top \rangle$

$\downarrow \qquad \searrow p_3, \langle 1, 2, \top \rangle$

$p_2, \langle 1, 2, \top \rangle$

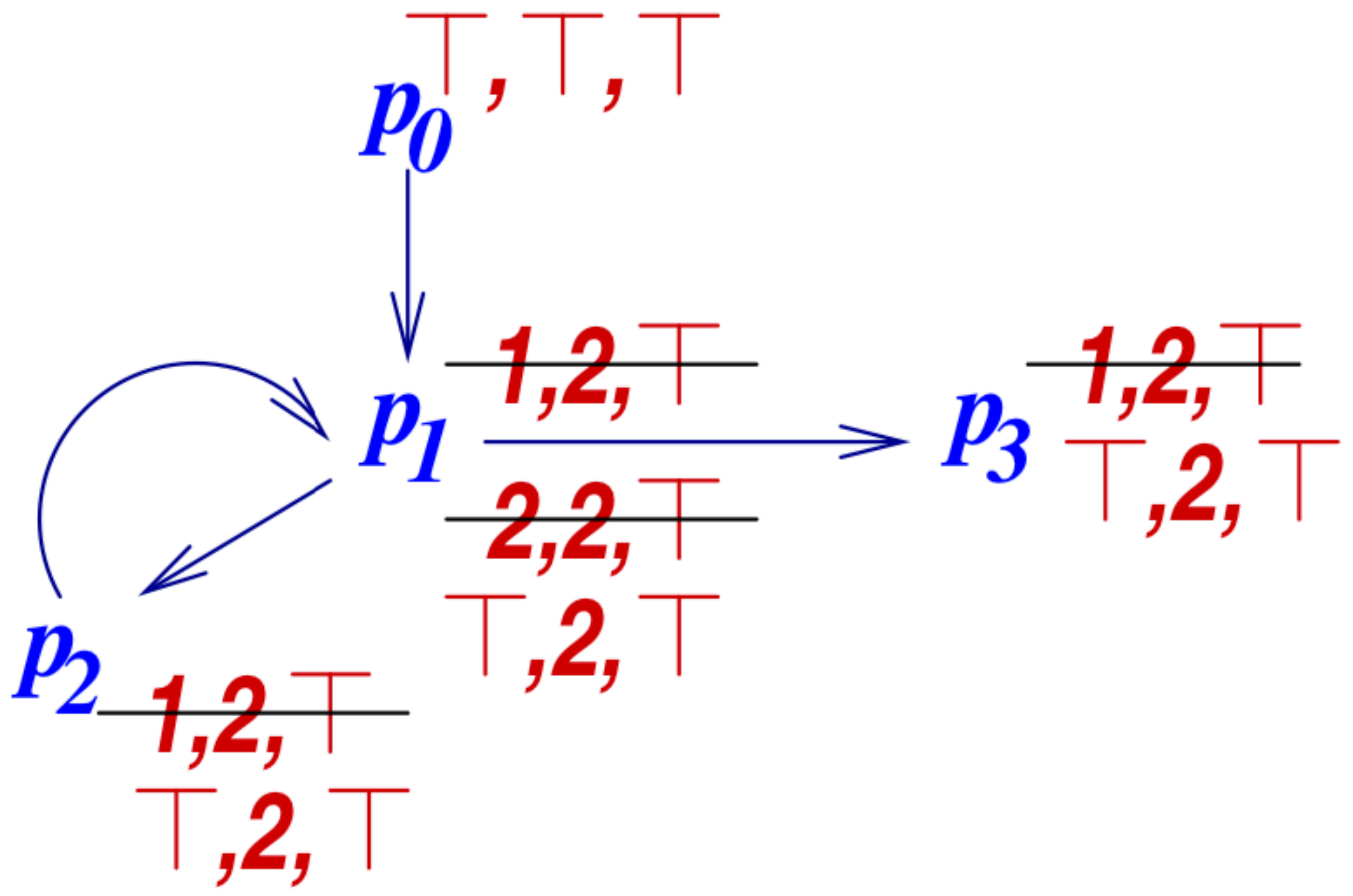
$p_1, \langle 2, 2, \top \rangle \sqcup \langle 1, 2, \top \rangle$

$= p_1, \langle \top, 2, \top \rangle$

$\downarrow \qquad \searrow p_3, \langle \top, 2, \top \rangle$

$p_2, \langle \top, 2, \top \rangle$

Drawn as a data–flow analysis:



The analysis tells us to replace y at p_1 by 2:

```
p0 :  x = 1; y = 2;
p1 :  while (x < y + z)  {
        p2 :  x = x + 1;
      }
p3 :  exit
```

2