Automatic Inference of Ranking Functions by Abstract Interpretation

Caterina Urban

MPRI 2-6: Abstract Interpretation,
Application to Verification and Static Analysis

15th October 2014

The cost of software failure

- Patriot MIM-104 failure, 25 February 1991
  (death of 28 soldiers[1])
- Ariane 5 failure, 4 June 1996
  (cost estimated at more than 370 000 000 US$[2])
- Toyota electronic throttle control system failure, 2005
  (at least 89 death[3])
- Heartbleed bug in OpenSSL, April 2014

- economic cost of software bugs is tremendous[4]

---

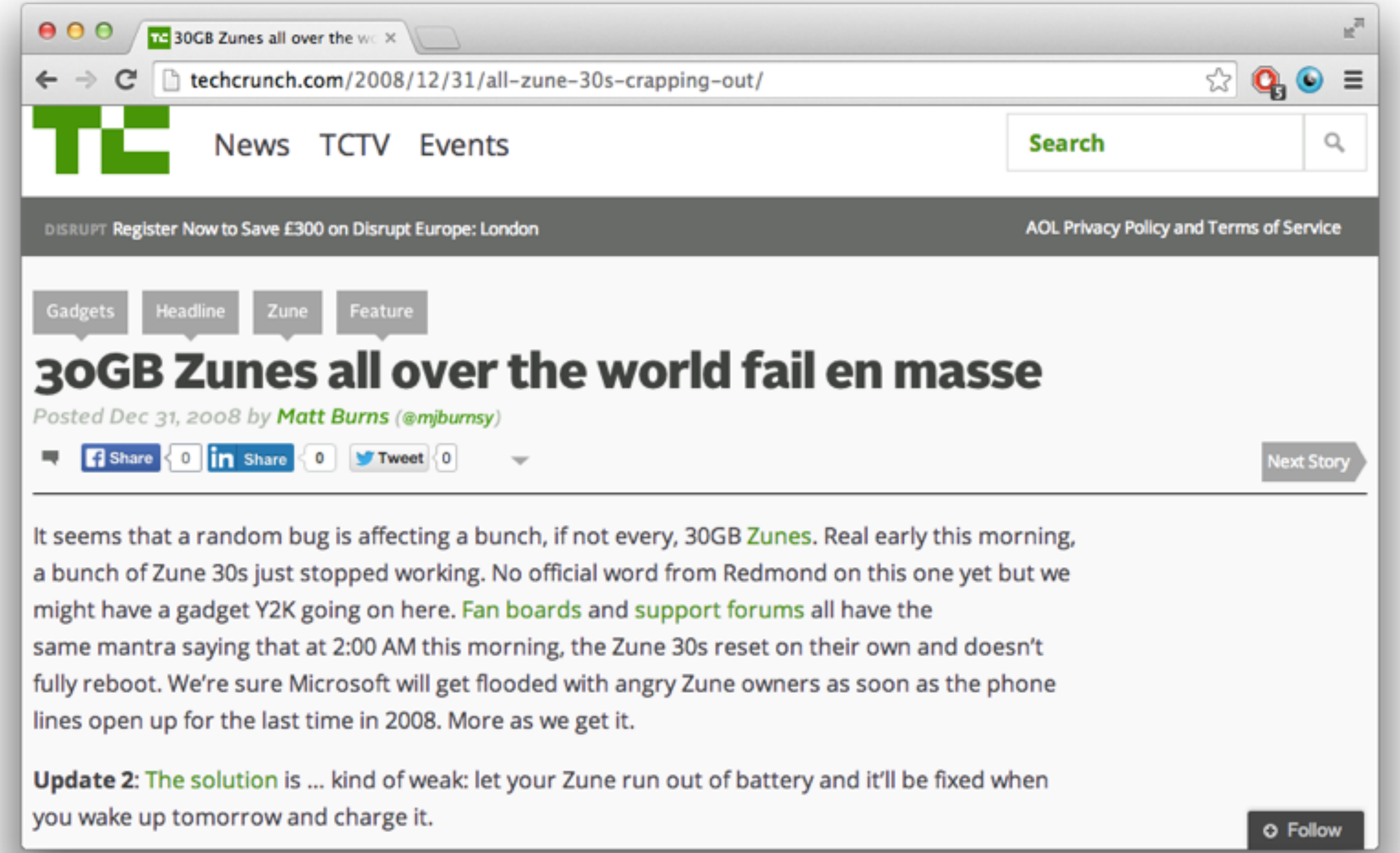
[1] R. Skeel. "Roundoff Error and the Patriot Missile". SIAM News, volume 25, nr 4.

[2] M. Dowson. "The Ariane 5 Software Failure". Software Engineering Notes 22 (2): 84, March 1997.

[3] CBSNews. Toyota "Unintended Acceleration" Has Killed 89. 20 March 2014.

[4] NIST. Software errors cost U.S. economy $59.5 billion annually. Tech. report, NIST Planning Report, 2002.

The Zune Bug December 31st, 2008



30GB Zunes all over the world fail en masse

Posted Dec 31, 2008 by Matt Burns (@mjburnsy)

It seems that a random bug is affecting a bunch, if not every, 30GB Zunes. Real early this morning, a bunch of Zune 30s just stopped working. No official word from Redmond on this one yet but we might have a gadget Y2K going on here. Fan boards and support forums all have the same mantra saying that at 2:00 AM this morning, the Zune 30s reset on their own and doesn't fully reboot. We're sure Microsoft will get flooded with angry Zune owners as soon as the phone lines open up for the last time in 2008. More as we get it.

Update 2: The solution is … kind of weak: let your Zune run out of battery and it'll be fixed when you wake up tomorrow and charge it.

- failure due to **non-termination**

`http://techcrunch.com/2008/12/31/zune-bug-explained-in-detail/`

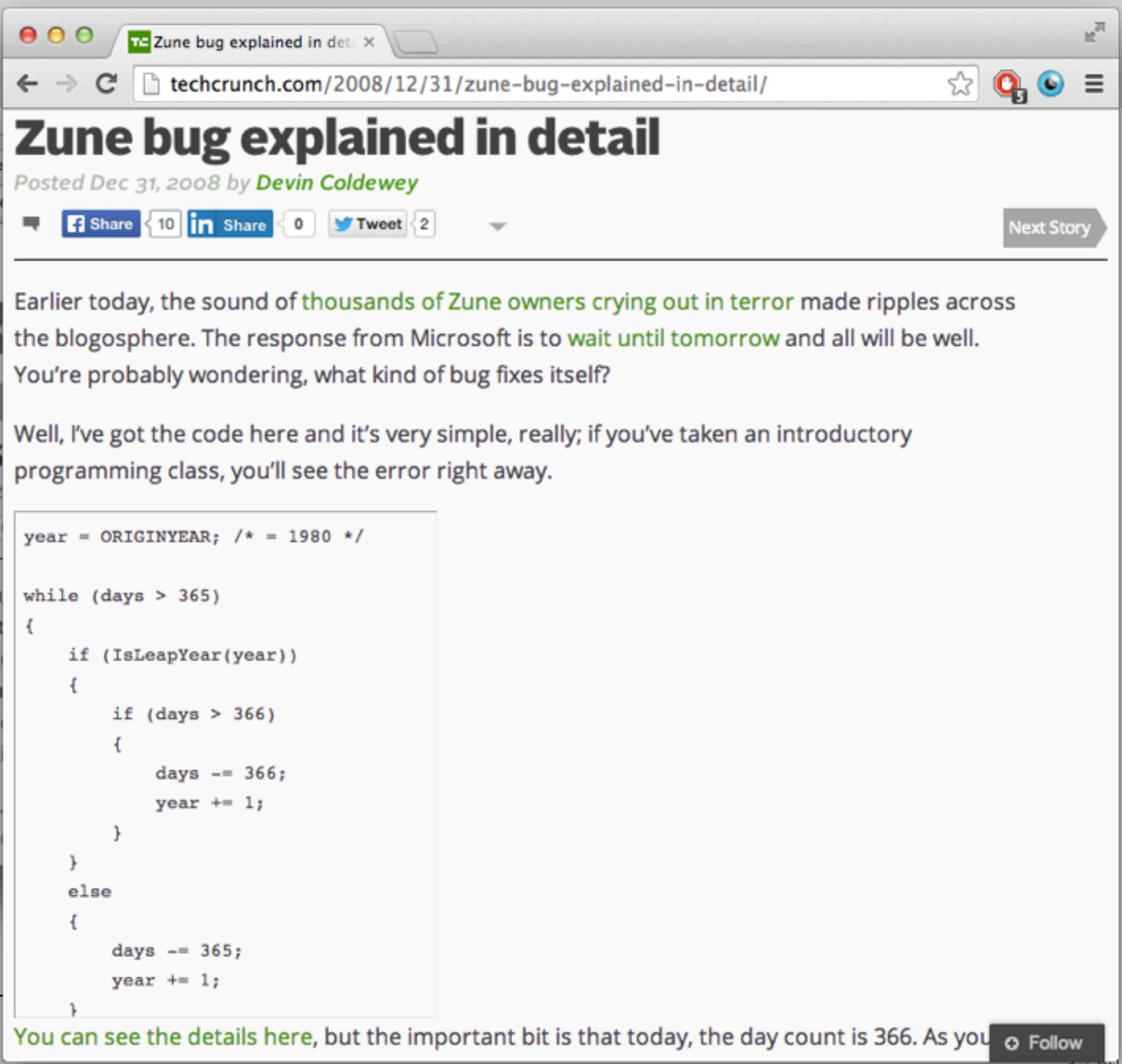
The Zune Bug

30GB Zunes

Posted Dec 31, 2008 by Mat

It seems that a random bu
a bunch of Zune 30s just st
might have a gadget Y2K g
same mantra saying that a
fully reboot. We're sure Mi
lines open up for the last t

Update 2: The solution is ..
you wake up tomorrow an



Zune bug explained in detail

Posted Dec 31, 2008 by Devin Coldewey

Earlier today, the sound of thousands of Zune owners crying out in terror made ripples across the blogosphere. The response from Microsoft is to wait until tomorrow and all will be well. You're probably wondering, what kind of bug fixes itself?

Well, I've got the code here and it's very simple, really; if you've taken an introductory programming class, you'll see the error right away.

```
year = ORIGINYEAR; /* = 1980 */

while (days > 365)
{
    if (IsLeapYear(year))
    {
        if (days > 366)
        {
            days -= 366;
            year += 1;
        }
    }
    else
    {
        days -= 365;
        year += 1;
    }
```

You can see the details here, but the important bit is that today, the day count is 366. As you

- failure due to

---

http://techcrunch

The Zune Bug

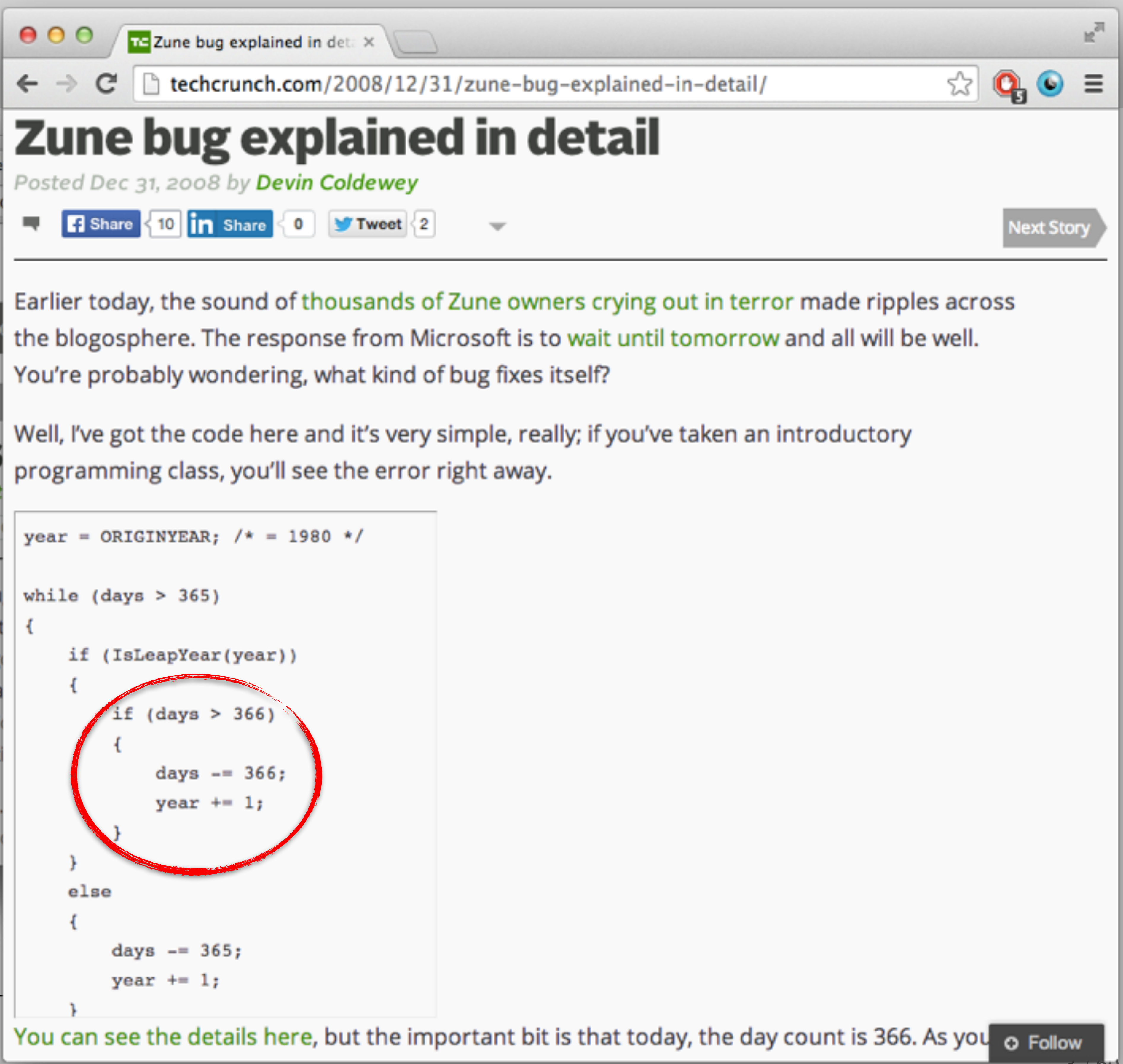
Register Now to Save £30

30GB Zunes

Posted Dec 31, 2008 by Mat

It seems that a random bu
a bunch of Zune 30s just st
might have a gadget Y2K g
same mantra saying that a
fully reboot. We're sure Mi
lines open up for the last t

Update 2: The solution is ...
you wake up tomorrow an



Zune bug explained in detail

Posted Dec 31, 2008 by Devin Coldewey

Earlier today, the sound of thousands of Zune owners crying out in terror made ripples across the blogosphere. The response from Microsoft is to wait until tomorrow and all will be well. You're probably wondering, what kind of bug fixes itself?

Well, I've got the code here and it's very simple, really; if you've taken an introductory programming class, you'll see the error right away.

```
year = ORIGINYEAR; /* = 1980 */

while (days > 365)
{
    if (IsLeapYear(year))
    {
        if (days > 366)
        {
            days -= 366;
            year += 1;
        }
    }
    else
    {
        days -= 365;
        year += 1;
    }
```

You can see the details here, but the important bit is that today, the day count is 366. As you

- failure due to

http://techcrunch

Liveness properties

Idea: liveness property $P \in \mathcal{P}(\Sigma^\infty)$

Liveness properties model that "something good eventually occurs"

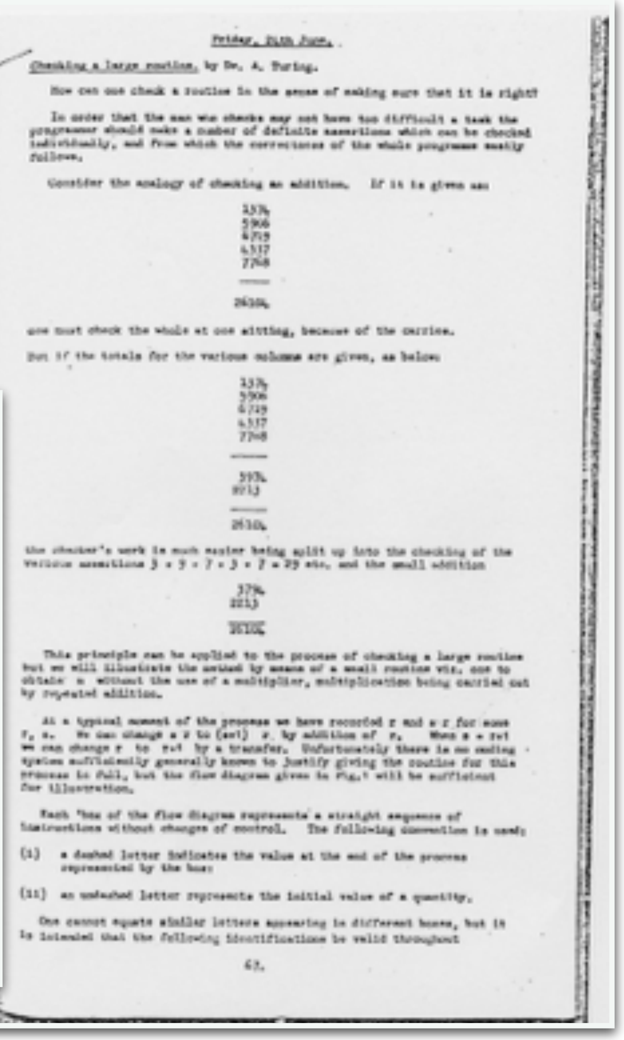
- P cannot be proved by testing
  (if nothing good happens in a prefix execution,
  it can still happen in the rest of the execution)
- disproving P requires exhibiting an infinite execution not in P

Examples:

- termination: $P \overset{\text{def}}{=} \Sigma^*$,
- inevitability: $P \overset{\text{def}}{=} \Sigma^* \cdot a \cdot \Sigma^\infty$,
  (a eventually occurs in all executions)
- state properties are not liveness properties.

Ranking Functions

- functions that strictly **decrease** at each program step...
- ...and that are **bounded** from below



Friday, 24th June.

Checking a large routine, by Dr. A. Turing.

How can one check a routine in the sense of making sure that it is right?

In order that the man who checks may not have too difficult a task the programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.

Robert W. Floyd

ASSIGNING MEANINGS TO PROGRAMS[1]

Introduction. This paper attempts to provide an adequate basis for formal definitions of the meanings of programs in appropriately defined programming languages, in such a way that a rigorous standard is established for proofs about computer programs, including proofs of correctness, equivalence, and termination. The basis of our approach is the notion of an interpretation of a program: that is, an association of a proposition with each connection in the flow of control through a program, where the proposition is asserted to hold whenever that connection is taken. To prevent an interpretation from being chosen arbitrarily, a condition is imposed on each command of the program. This condition guarantees that whenever a command is reached by way of a connection whose associated proposition is then true, it will be left (if at all) by a connection whose associated proposition will be true at that time. Then by induction on the number of commands executed, one sees that if a program is entered by a connection whose associated proposition is then true, it will be left (if at all) by a connection whose associated proposition will be true at that time. By this means, we may prove certain properties of programs, particularly properties of the form: "If the initial values of the program variables satisfy the relation R_1, the final values on completion will satisfy the relation R_2." Proofs of termination are dealt with by showing that each step of a program decreases some entity which cannot decrease indefinitely.

These modes of proof of correctness and termination are not original; they are based on ideas of Perlis and Gorn, and may have made their earliest appearance in an unpublished paper by Gorn. The establishment of formal standards for proofs about programs in languages which admit assignments, transfer of control, etc., and the proposal that the semantics of a programming language may be defined independently of all processors for that language, by establishing standards of rigor for proofs about

---

[1] This work was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (SD-146).

19

Turing - *Checking a Large Routine* (1949)
Floyd - *Assigning Meanings To Programs* (1967)

Proving liveness properties

Variance proof method: (informal definition)

Find a decreasing quantity until something good happens.

Example: termination proof

- find $f : \Sigma \to \mathcal{S}$ where $(\mathcal{S}, \sqsubseteq)$ is well-ordered;

  (f is called a "ranking function")

- $\sigma \in \mathcal{B} \implies f = \min \mathcal{S}$;
- $\sigma \to \sigma' \implies f(\sigma') \sqsubset f(\sigma)$.

(f counts the number of steps remaining before termination)

- **idea**: inference of ranking functions by **abstract interpretation**

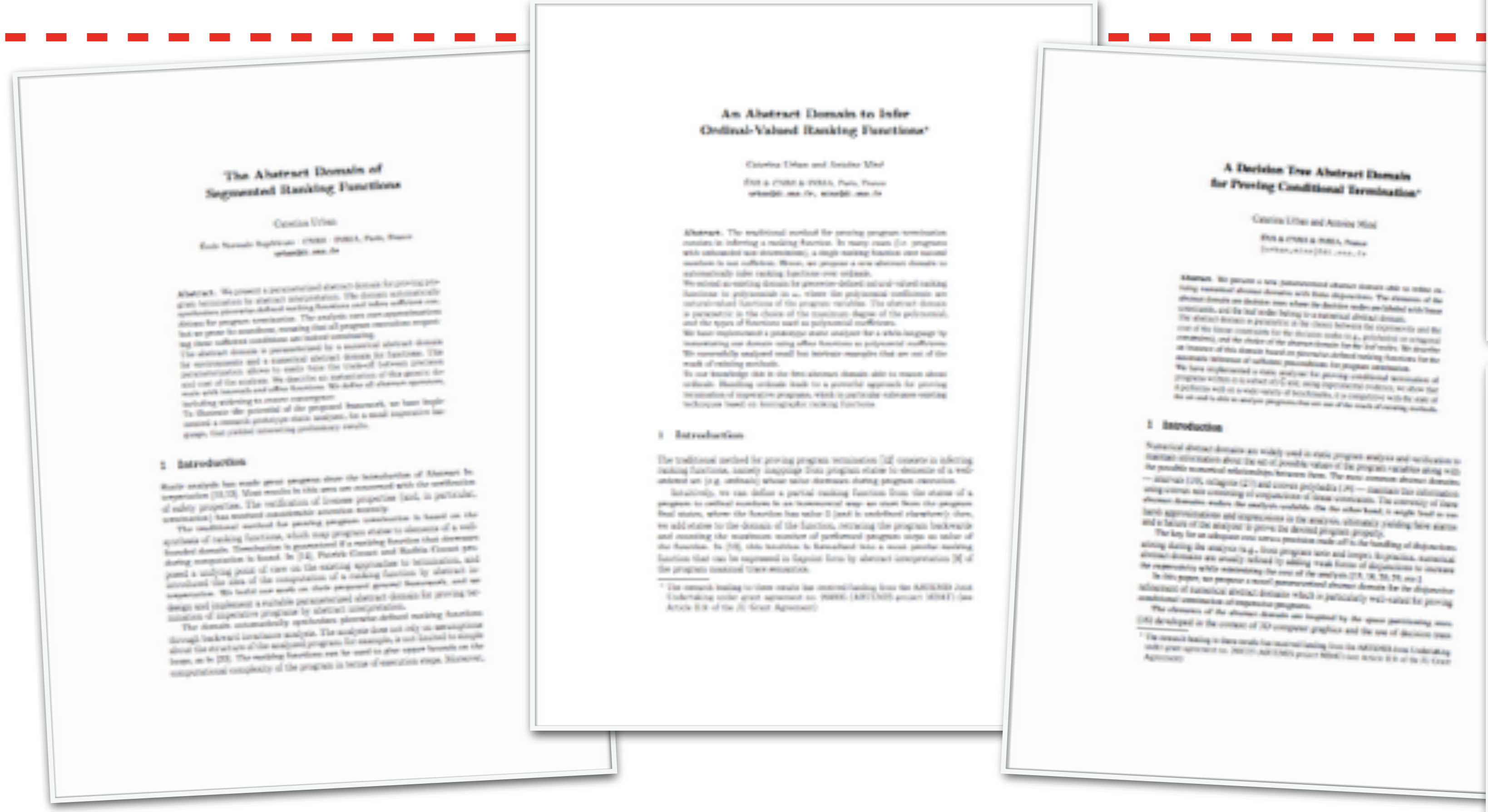
An Abstract Interpretation Framework for Termination

Abstract

Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)

- **idea**: inference of ranking functions by **abstract interpretation**

- family of **abstract domains** for program termination
  - **piecewise-defined ranking functions**
  - **backward analysis**
  - **sufficient preconditions** for termination



The Abstract Domain of Segmented Ranking Functions

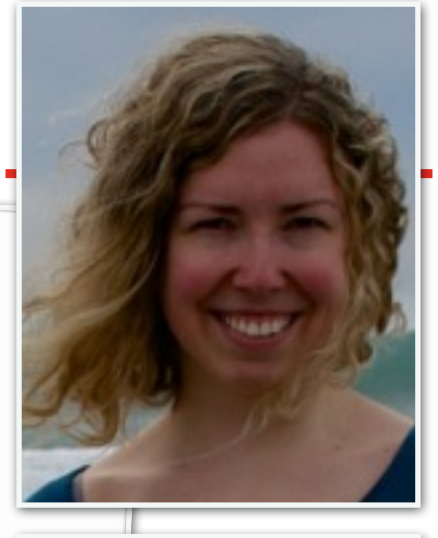
Caterina Urban

An Abstract Domain to Infer Ordinal-Valued Ranking Functions*

Caterina Urban and Antoine Miné

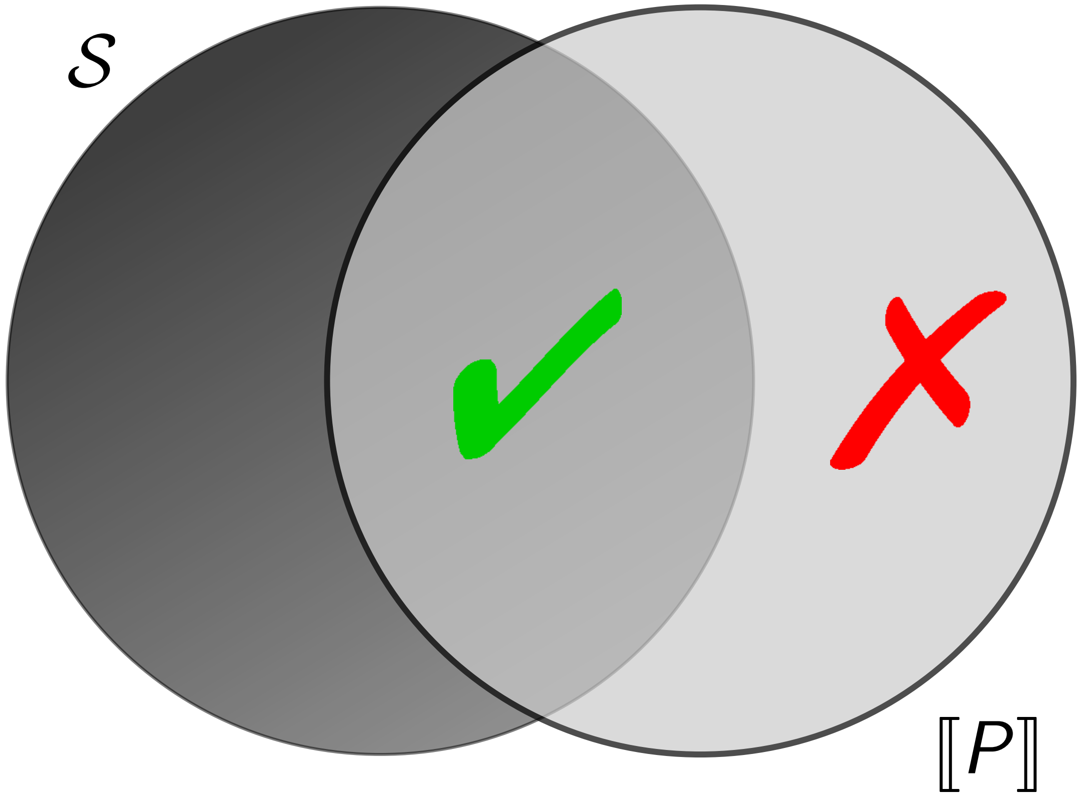
A Decision Tree Abstract Domain for Proving Conditional Termination*

Caterina Urban and Antoine Miné



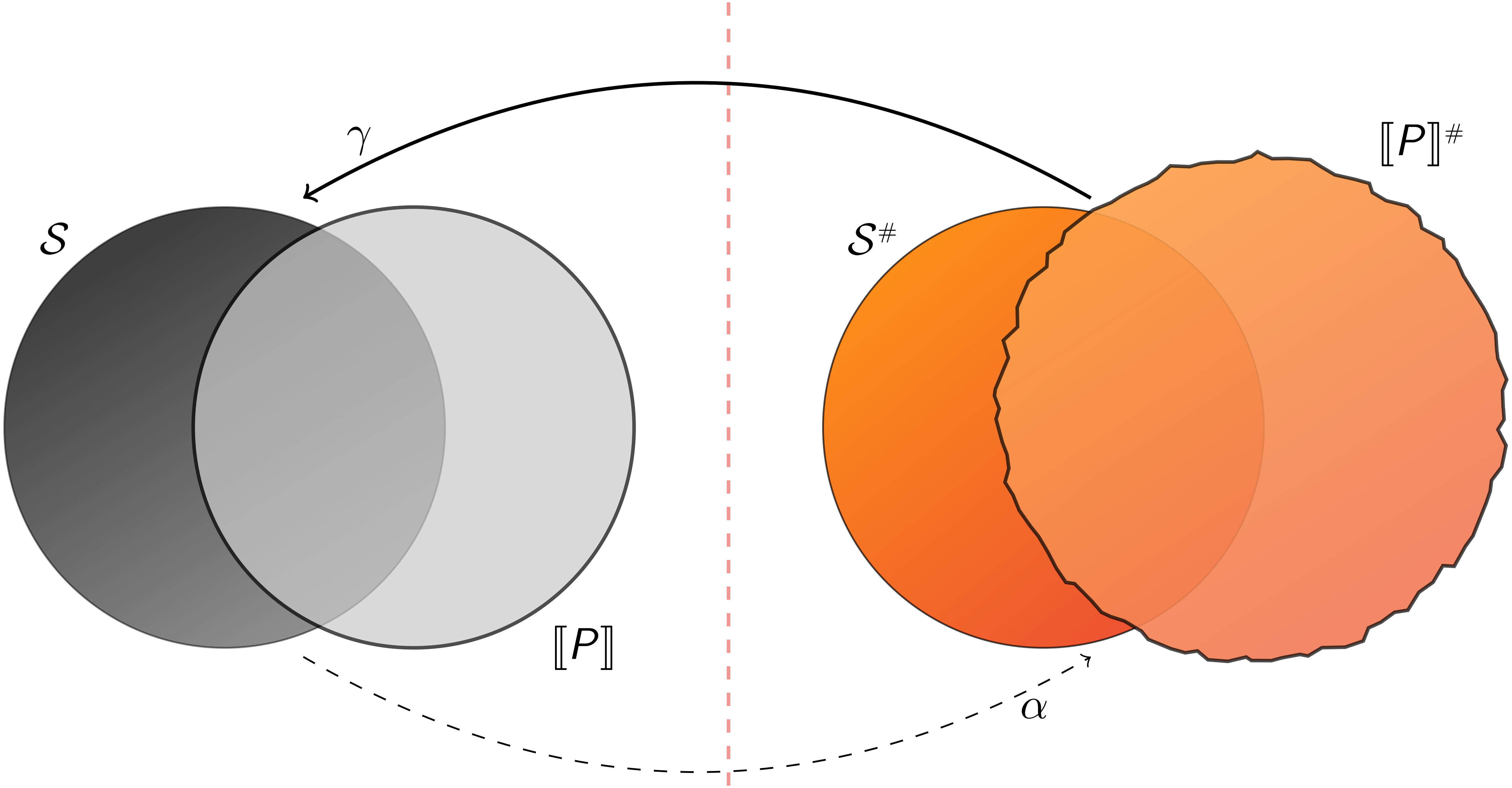
Urban - *The Abstract Domain of Segmented Ranking Functions* (SAS 2013)
Urban&Miné - *An Abstract Domain to Infer Ordinal-Valued Ranking Functions* (ESOP 2014)
Urban&Miné - *A Decision Tree Abstract Domain for Proving Conditional Termination* (SAS 2014)

Abstract Interpretation



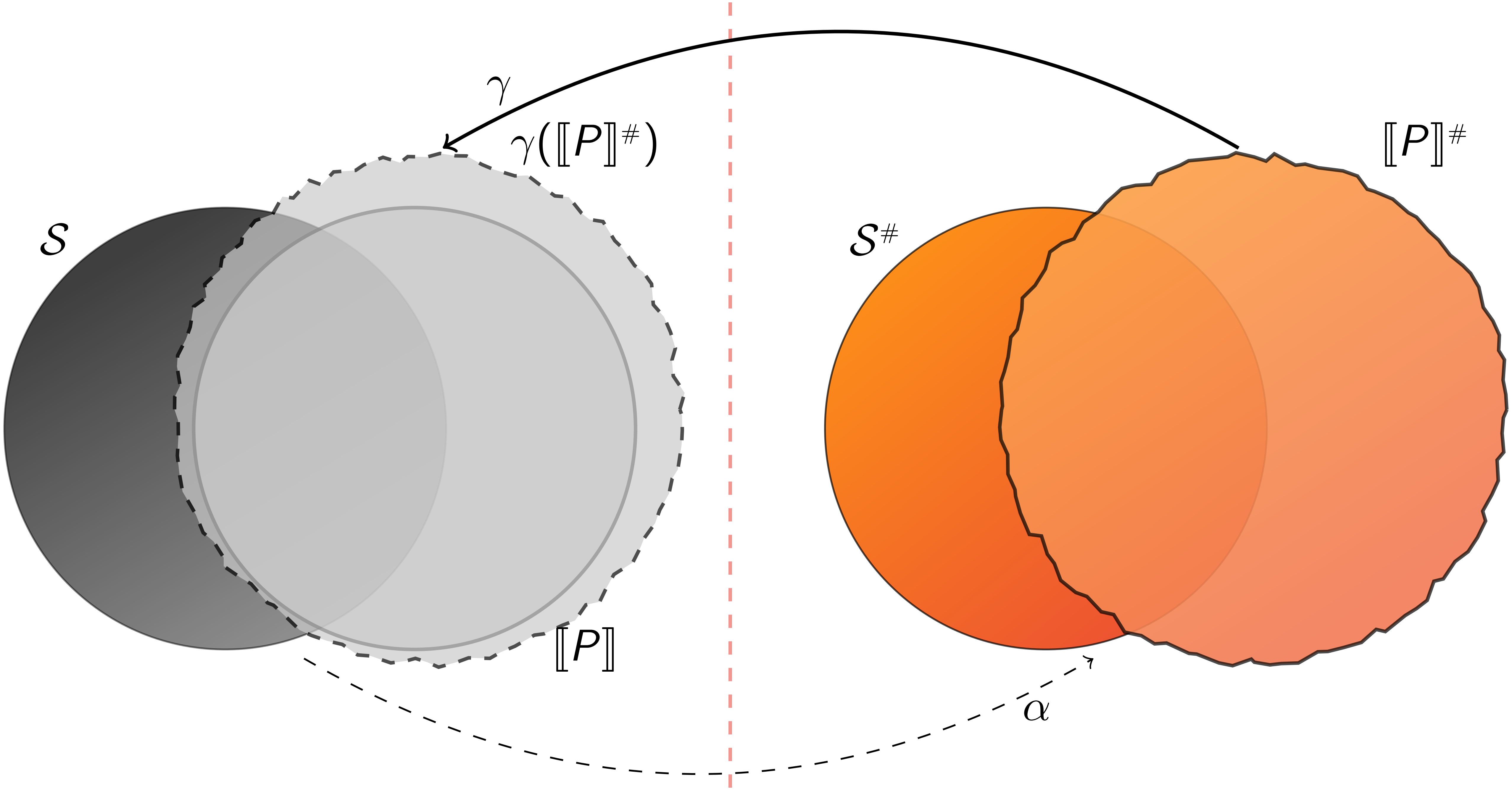
Cousot&Cousot - *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints* (POPL 1977)

Abstract Interpretation

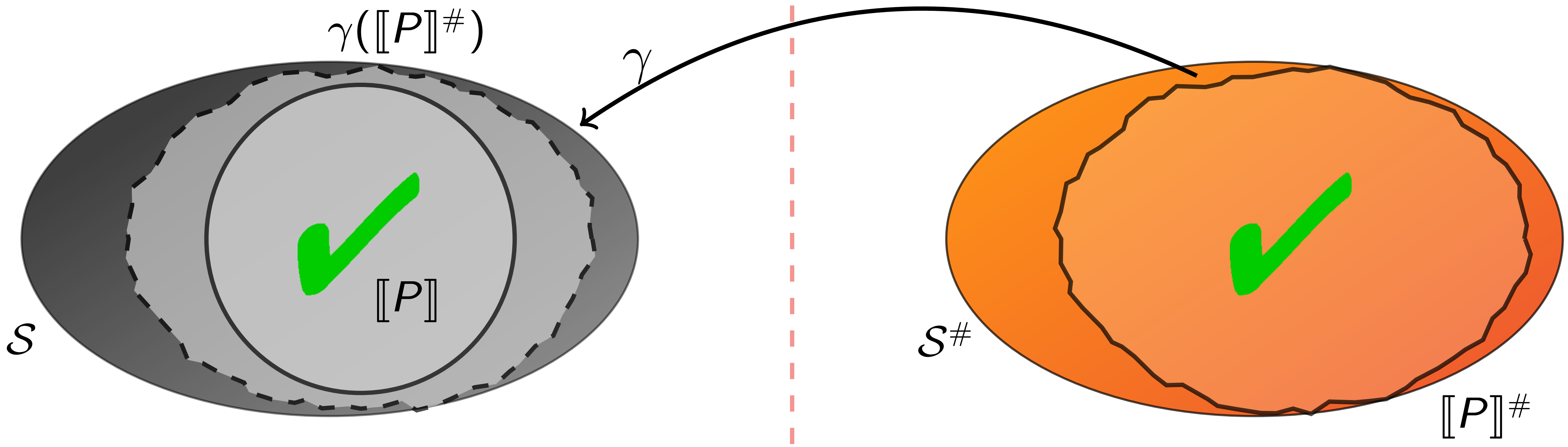


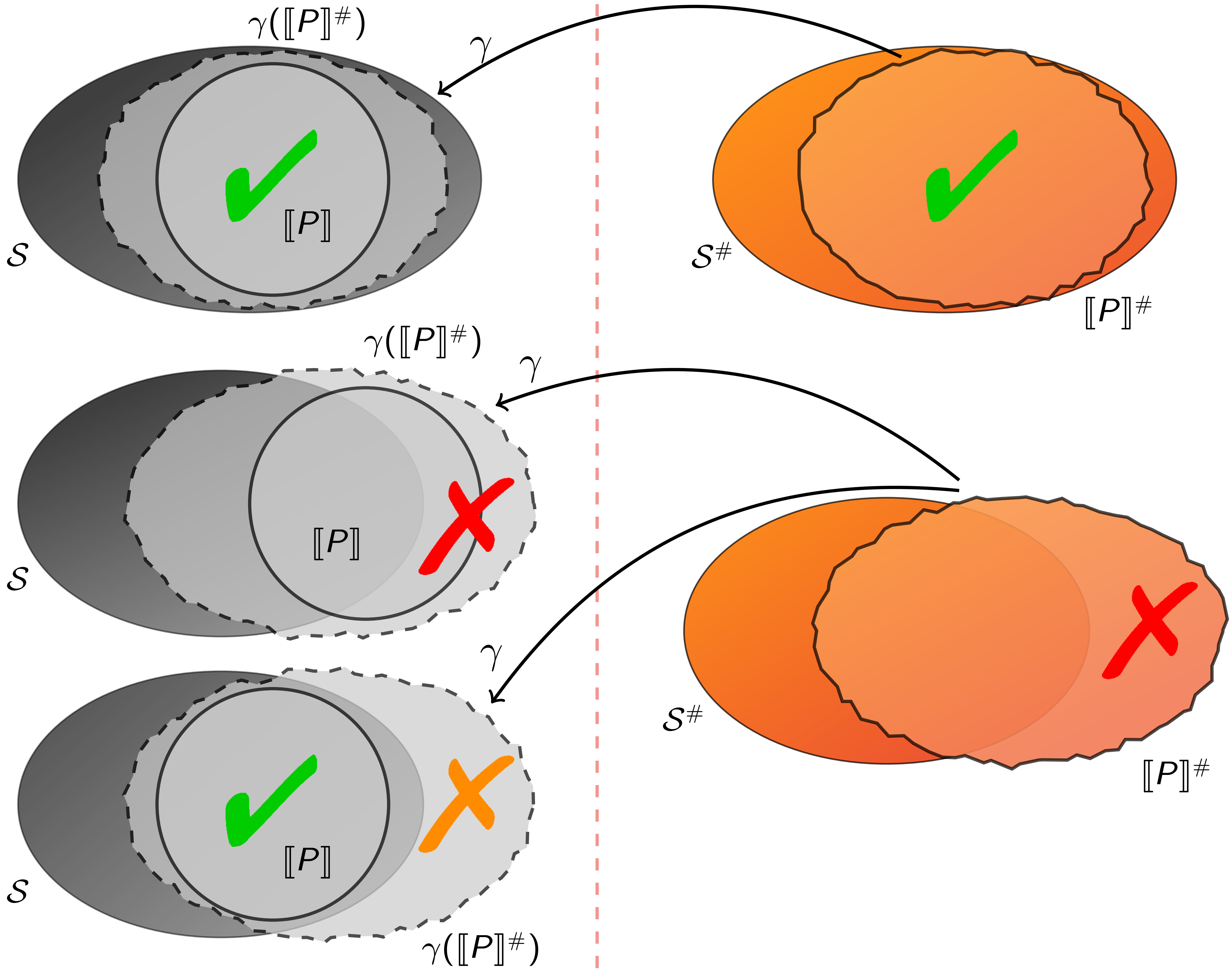
Cousot&Cousot - *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints* (POPL 1977)

Abstract Interpretation



Cousot&Cousot - *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints* (POPL 1977)


$\gamma(\llbracket P \rrbracket^{\#})$
γ
$\mathcal{S}$
$\llbracket P \rrbracket$
✓
$\mathcal{S}^{\#}$
✓
$\llbracket P \rrbracket^{\#}$


$\gamma(\llbracket P \rrbracket^\#)$
γ
$\llbracket P \rrbracket$
$\mathcal{S}$
$\mathcal{S}^\#$
$\llbracket P \rrbracket^\#$
$\gamma(\llbracket P \rrbracket^\#)$
γ
$\llbracket P \rrbracket$
$\mathcal{S}$
γ
$\mathcal{S}^\#$
$\llbracket P \rrbracket^\#$
$\llbracket P \rrbracket$
$\mathcal{S}$
$\gamma(\llbracket P \rrbracket^\#)$

Termination Semantics

An Abstract Interpretation Framework for Termination

Patrick Cousot

Radhia Cousot

Transition systems: definition

Language-neutral formalism to discuss about program semantics.

Transition system: (Σ, τ)

- set of states Σ,

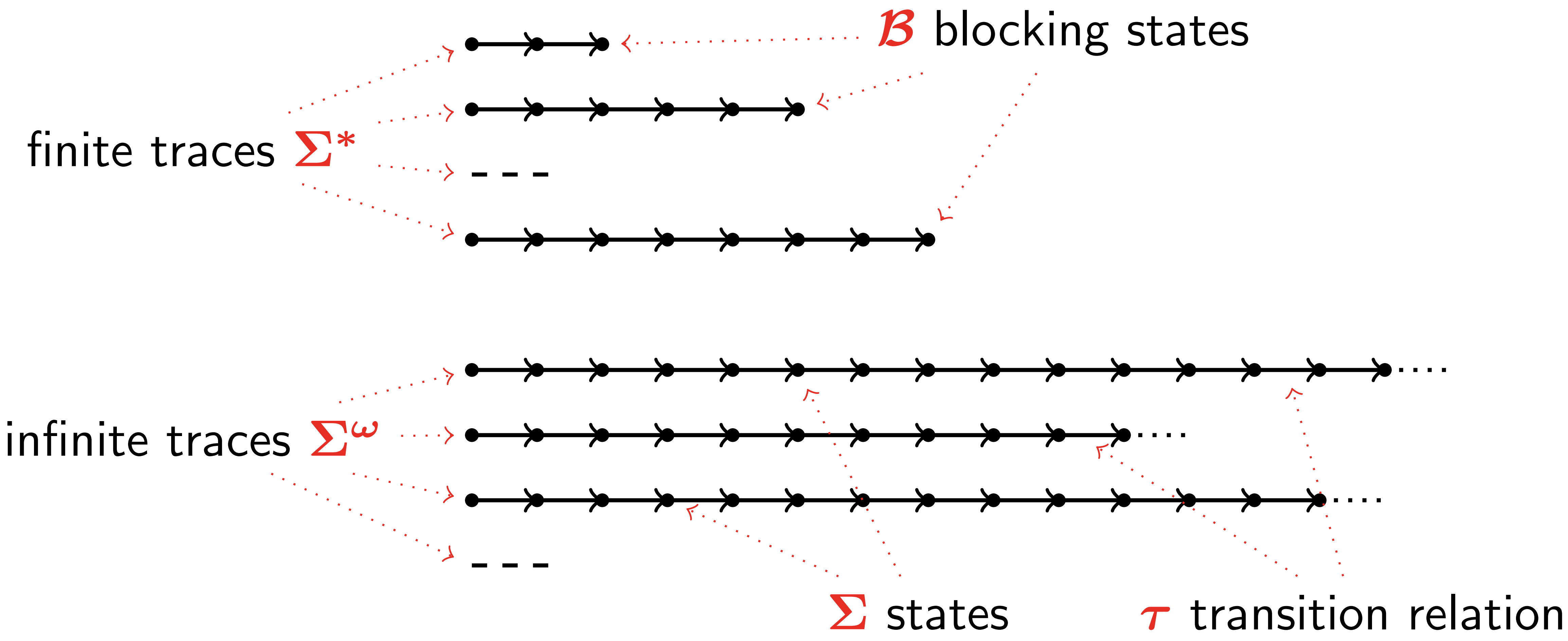
  (memory states, $\lambda-$terms, configurations, etc., generally infinite)
- transition relation $\tau \subseteq \Sigma \times \Sigma$.

(Σ, τ) is a general form of small-step operational semantics.

$(\sigma, \sigma') \in \tau$ is noted $\sigma \to \sigma'$:

starting in state σ, after an execution step, we can go to state σ'.

program ↦ **maximal trace semantics**



Least fixpoint formulation of maximal traces

Idea: To get a fixpoint formulation for whole $\mathcal{M}_\infty$,
merge finite and infinite maximal trace fixpoint forms.

Fixpoint fusion

$\mathcal{M}_\infty \cap \Sigma^*$ is best defined on $(\Sigma^*, \subseteq, \cup, \cap, \emptyset, \Sigma^*)$.
$\mathcal{M}_\infty \cap \Sigma^\omega$ is best defined on $(\Sigma^\omega, \supseteq, \cap, \cup, \Sigma^\omega, \emptyset)$.

We mix them into a new complete lattice $(\Sigma^\infty, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$:

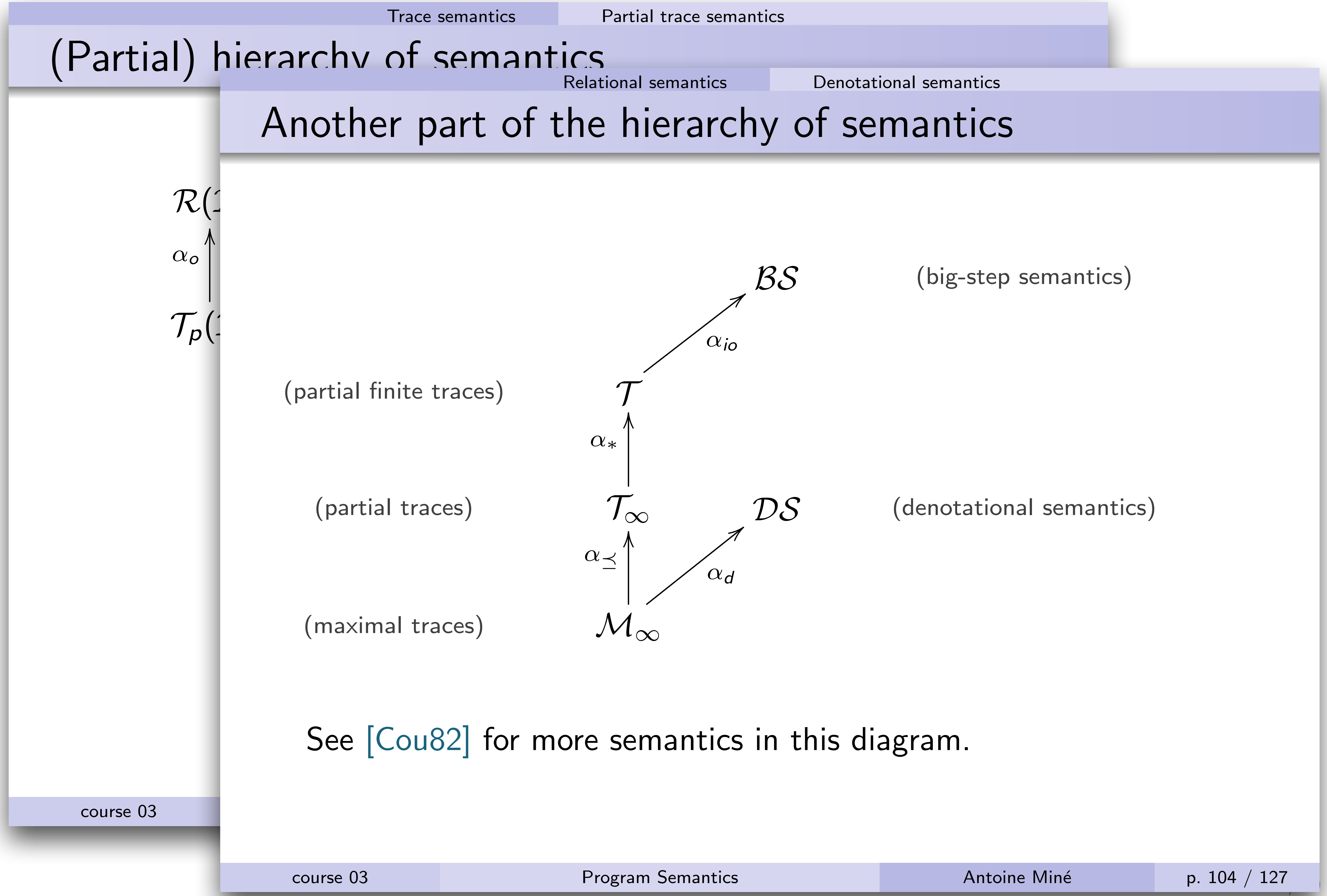
- $A \sqsubseteq B \overset{\text{def}}{\iff} (A \cap \Sigma^*) \subseteq (B \cap \Sigma^*) \wedge (A \cap \Sigma^\omega) \supseteq (B \cap \Sigma^\omega)$
- $A \sqcup B \overset{\text{def}}{=} ((A \cap \Sigma^*) \cup (B \cap \Sigma^*)) \cup ((A \cap \Sigma^\omega) \cap (B \cap \Sigma^\omega))$
- $A \sqcap B \overset{\text{def}}{=} ((A \cap \Sigma^*) \cap (B \cap \Sigma^*)) \cup ((A \cap \Sigma^\omega) \cup (B \cap \Sigma^\omega))$
- $\bot \overset{\text{def}}{=} \Sigma^\omega$
- $\top \overset{\text{def}}{=} \Sigma^*$

In this lattice, $\boxed{\mathcal{M}_\infty = \operatorname{lfp} F_s \text{ where } F_s(T) \overset{\text{def}}{=} \mathcal{B} \cup \tau \frown T.}$

(proof on next slides)

(Partial) hierarchy of semantics

$$\begin{tikzcd}
\mathcal{R}(\mathcal{I}) & & \mathcal{C}(\mathcal{F}) & \text{(states)} \\
\mathcal{T}_p(\mathcal{I}) \arrow[u, "\alpha_o"] & & \mathcal{T}_s(\mathcal{F}) \arrow[u, "\alpha_o"'] & \text{(anchored traces)} \\
& \mathcal{T} \arrow[ul, "\alpha_{\mathcal{I}}"] \arrow[ur, "\alpha_{\mathcal{F}}"'] & & \text{(partial finite traces)} \\
& \mathcal{T}_\infty \arrow[u, "\alpha_*"] & & \text{(partial traces)} \\
& \mathcal{M}_\infty \arrow[u, "\alpha_\preceq"] & & \text{(maximal traces)}
\end{tikzcd}$$



(Partial) hierarchy of semantics

$$\begin{tikzcd}
\mathcal{R}(\\
\mathcal{T}_p(\arrow[u, "\alpha_o"]
\end{tikzcd}$$

Another part of the hierarchy of semantics

$$\begin{tikzcd}
 & & \mathcal{BS} & \text{(big-step semantics)} \\
\text{(partial finite traces)} & \mathcal{T} \arrow[ur, "\alpha_{io}"] & & \\
\text{(partial traces)} & \mathcal{T}_\infty \arrow[u, "\alpha_*"] & \mathcal{DS} & \text{(denotational semantics)} \\
\text{(maximal traces)} & \mathcal{M}_\infty \arrow[u, "\alpha_\preceq"] \arrow[ur, "\alpha_d"'] & &
\end{tikzcd}$$

See [Cou82] for more semantics in this diagram.

program $\mapsto$ maximal trace semantics $\rightarrow$ **termination semantics**

$$\mathcal{T}_t \in \Sigma \rightharpoonup \mathbb{O}$$

$$\mathcal{T}_t \overset{\text{def}}{=} \text{lfp } F_t$$

idea = define a ranking function **counting the number of program steps** from the end of the program

$$F_t(v)s \overset{\text{def}}{=} \begin{cases} 0 & s \in \mathcal{B} \\ \sup\{\ v(s') + 1 \mid s \rightarrow s'\ \} & s \in \widetilde{\text{pre}}(\text{dom}(v)) \\ \text{undefined} & \text{otherwise} \end{cases}$$

Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)

program ↦ maximal trace semantics → **termination semantics**

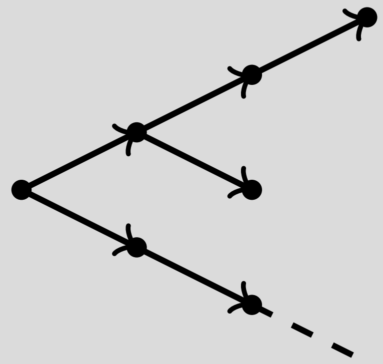
$$\mathcal{T}_t \in \Sigma \rightharpoonup \mathbb{O}$$

$$\mathcal{T}_t \overset{\text{def}}{=} \text{lfp } F_t$$

idea = define a ranking function **counting the number of program steps** from the end of the program

$$F_t(v)s \overset{\text{def}}{=} \begin{cases} 0 & s \in \mathcal{B} \\ \sup\{\, v(s') + 1 \mid s \to s' \,\} & s \in \widetilde{\text{pre}}(\text{dom}(v)) \\ \text{undefined} & \text{otherwise} \end{cases}$$

Example



Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)

program ↦ maximal trace semantics → **termination semantics**

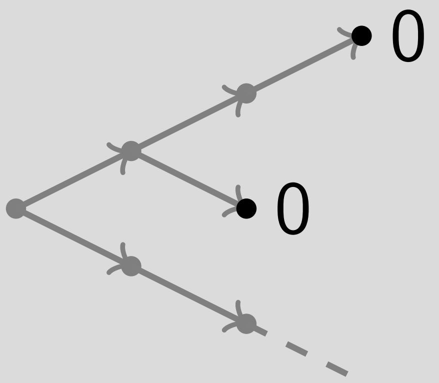
$$\mathcal{T}_{\mathbf{t}} \in \Sigma \rightharpoonup \mathbb{O}$$

$$\mathcal{T}_{\mathbf{t}} \overset{\text{def}}{=} \mathbf{lfp}\ \mathbf{F}_{\mathbf{t}}$$

idea = define a ranking function **counting the number of program steps** from the end of the program

$$\mathbf{F}_{\mathbf{t}}(\mathbf{v})\mathbf{s} \overset{\text{def}}{=} \begin{cases} \mathbf{0} & \mathbf{s} \in \mathcal{B} \longleftarrow \\ \mathbf{sup}\{\ \mathbf{v}(\mathbf{s}') + \mathbf{1} \mid \mathbf{s} \rightarrow \mathbf{s}'\ \} & \mathbf{s} \in \widetilde{\mathbf{pre}}(\mathbf{dom}(\mathbf{v})) \\ \mathbf{undefined} & \mathbf{otherwise} \end{cases}$$

Example



Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)

program ↦ maximal trace semantics → **termination semantics**

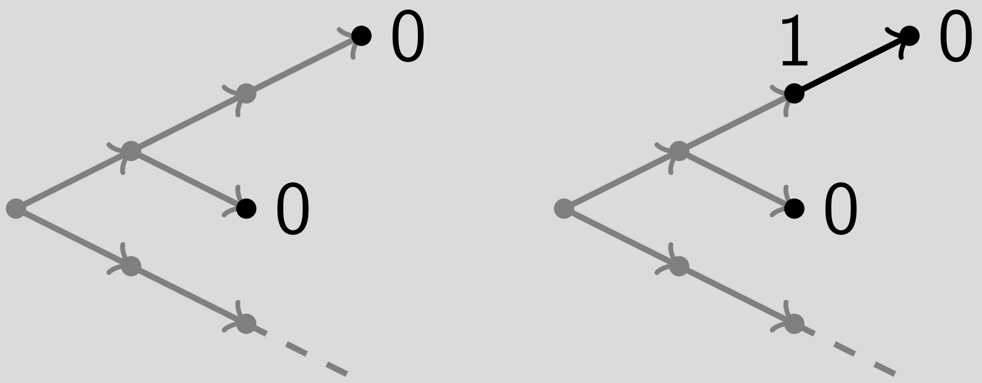
idea = define a ranking function **counting the number of program steps** from the end of the program

$$\mathcal{T}_t \in \Sigma \rightharpoonup \mathbb{O}$$

$$\mathcal{T}_t \overset{\text{def}}{=} \text{lfp } F_t$$

$$F_t(v)s \overset{\text{def}}{=} \begin{cases} 0 & s \in \mathcal{B} \\ \sup\{\, v(s') + 1 \mid s \to s' \,\} & s \in \widetilde{\text{pre}}(\text{dom}(v)) \longleftarrow \\ \text{undefined} & \text{otherwise} \end{cases}$$

Example



Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)

program ↦ maximal trace semantics → **termination semantics**

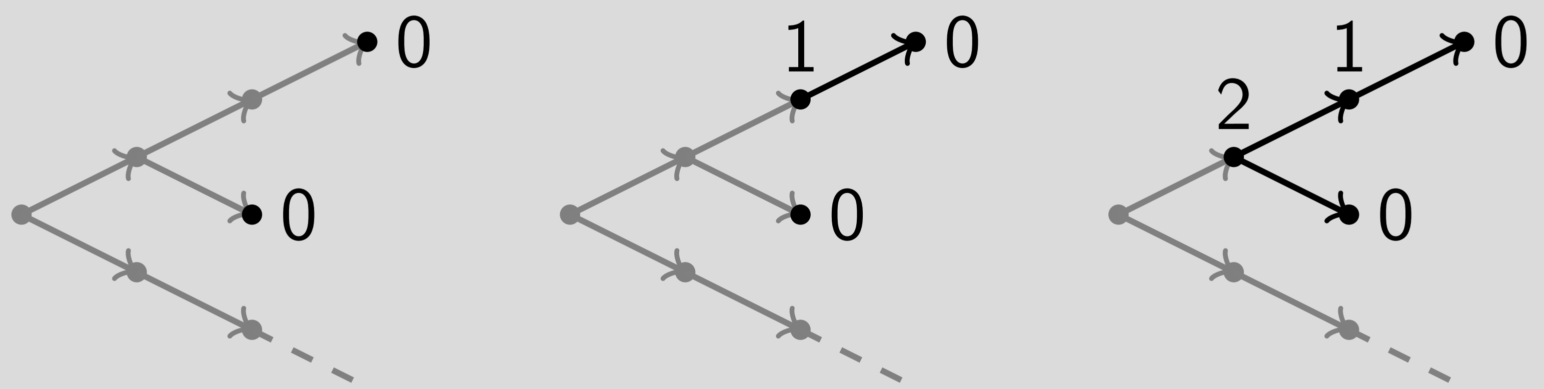
$$\mathcal{T}_t \in \Sigma \rightharpoonup \mathbb{O}$$

$$\mathcal{T}_t \overset{\text{def}}{=} \text{lfp}\ F_t$$

idea = define a ranking function **counting the number of program steps** from the end of the program

$$F_t(v)s \overset{\text{def}}{=} \begin{cases} 0 & s \in \mathcal{B} \\ \sup\{\ v(s') + 1 \mid s \to s'\ \} & s \in \widetilde{\text{pre}}(\text{dom}(v)) \\ \text{undefined} & \text{otherwise} \end{cases}$$

Example



Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)

program ↦ maximal trace semantics → **termination semantics**

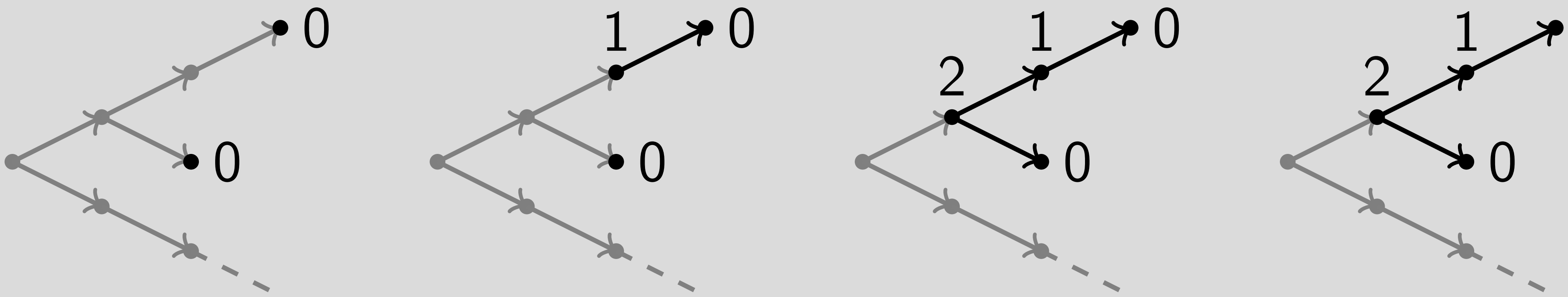
idea = define a ranking function **counting the number of program steps** from the end of the program

$$\mathcal{T}_t \in \Sigma \rightharpoonup \mathbb{O}$$

$$\mathcal{T}_t \overset{\text{def}}{=} \text{lfp } F_t$$

$$F_t(v)s \overset{\text{def}}{=} \begin{cases} 0 & s \in \mathcal{B} \\ \sup\{\, v(s') + 1 \mid s \to s' \,\} & s \in \widetilde{\text{pre}}(\text{dom}(v)) \\ \text{undefined} & \text{otherwise} \end{cases}$$

Example



Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)

program $\mapsto$ maximal trace semantics $\rightarrow$ **termination semantics**

$$\mathcal{T}_t \in \Sigma \rightharpoonup \mathbb{O}$$

$$\mathcal{T}_t \overset{\text{def}}{=} \mathrm{lfp}\; F_t$$

$$F_t(v)s \overset{\text{def}}{=} \begin{cases} 0 & s \in \mathcal{B} \\ \sup\{\, v(s') + 1 \mid s \rightarrow s' \,\} & s \in \widetilde{\mathrm{pre}}(\mathrm{dom}(v)) \\ \text{undefined} & \text{otherwise} \end{cases}$$

Definition (Computational Order)

$$v_1 \preccurlyeq v_2 \overset{\text{def}}{=} \boxed{\mathrm{dom}(v_1) \subseteq \mathrm{dom}(v_2)} \wedge \forall x \in \mathrm{dom}(v_1) : v_1(x) \leq v_2(x)$$

Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)

program $\mapsto$ maximal trace semantics $\rightarrow$ **termination semantics**

$$\mathcal{T}_t \in \Sigma \rightharpoonup \mathbb{O}$$

$$\mathcal{T}_t \overset{\text{def}}{=} \text{lfp } F_t$$

$$F_t(v)s \overset{\text{def}}{=} \begin{cases} 0 & s \in \mathcal{B} \\ \sup\{\, v(s') + 1 \mid s \rightarrow s' \,\} & s \in \widetilde{\text{pre}}(\text{dom}(v)) \\ \text{undefined} & \text{otherwise} \end{cases}$$

Theorem (Soundness and Completeness)

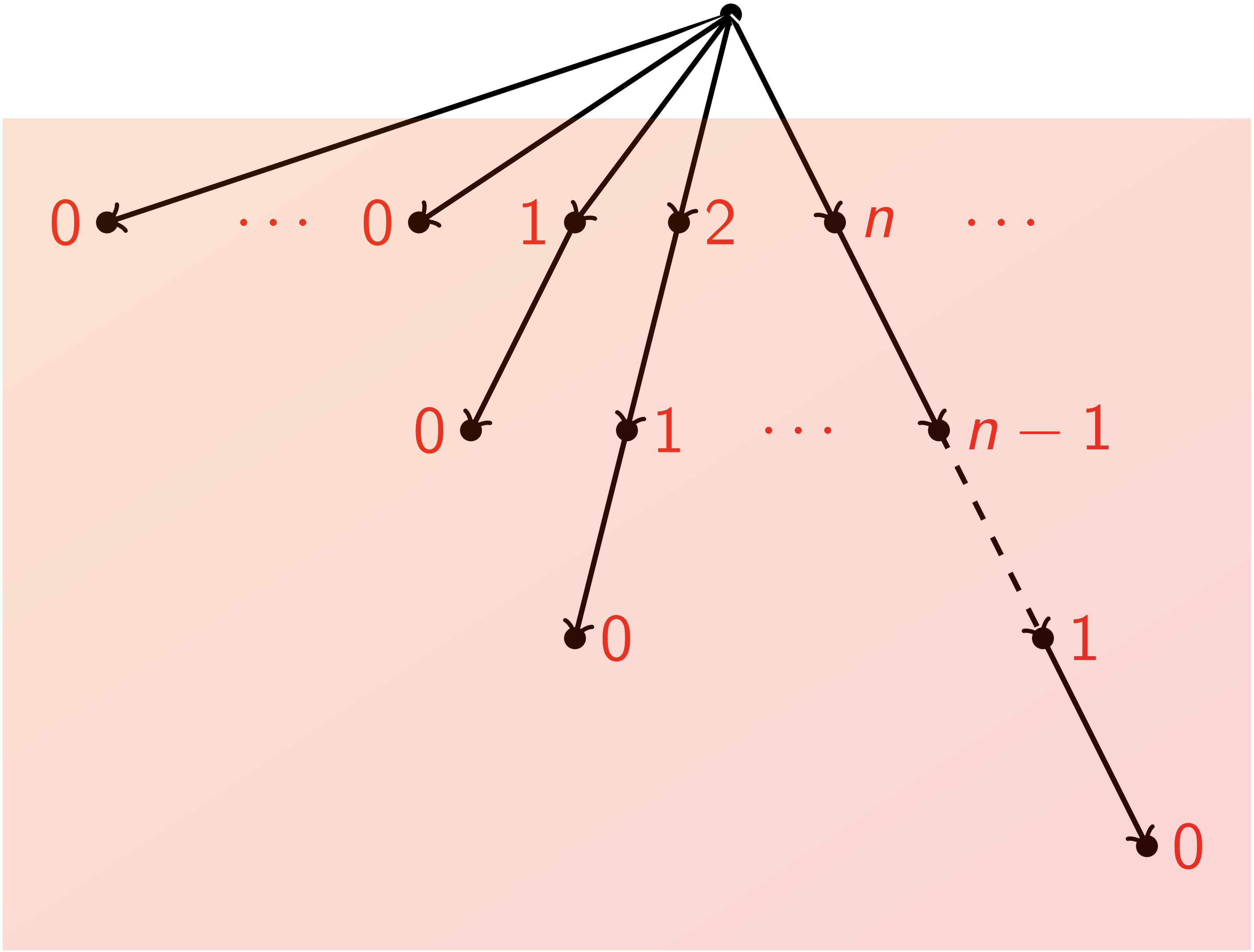
the termination semantics is **sound** *and* **complete**
to prove the termination of programs

Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)

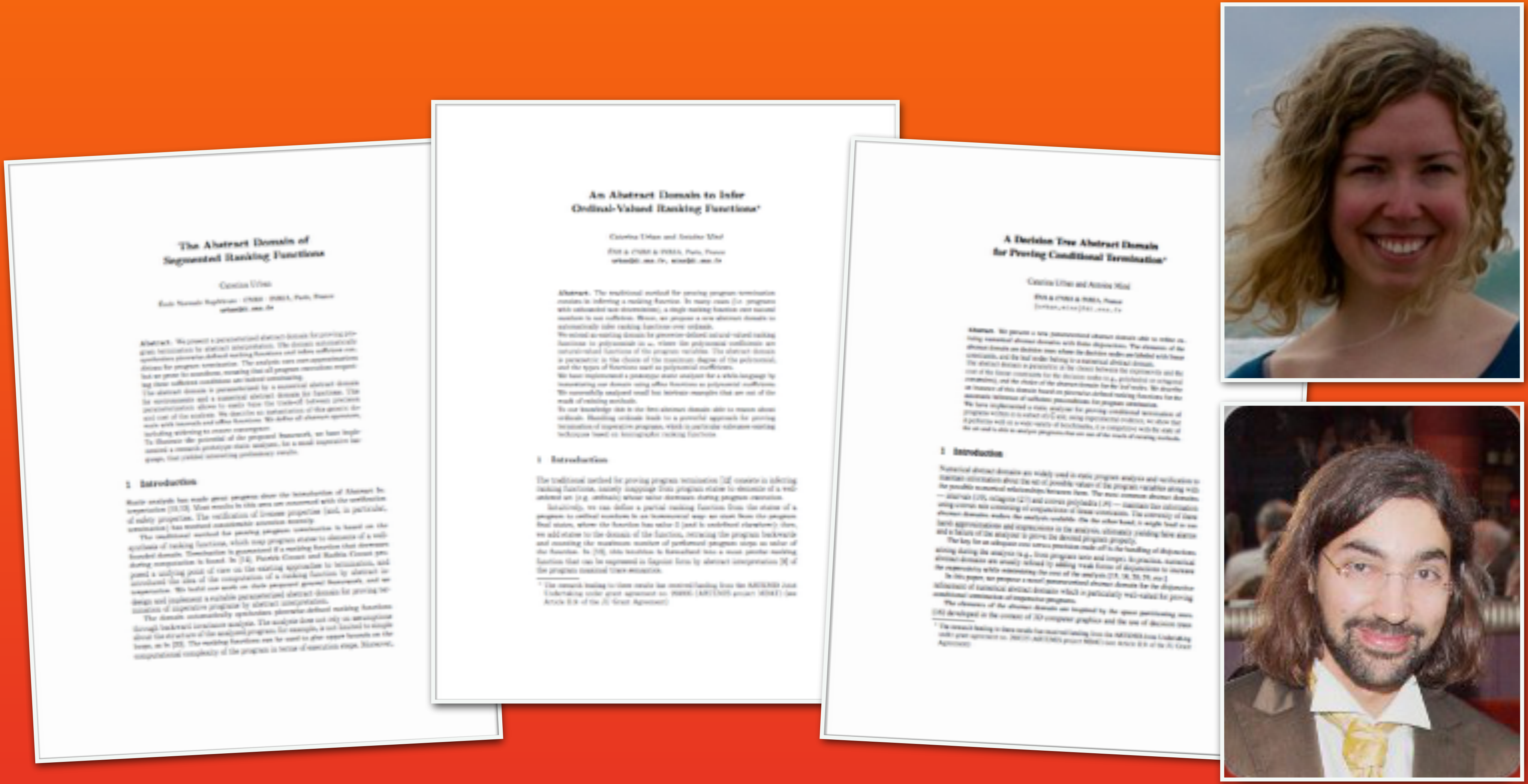
- **remark**: the termination semantics is **not computable**!

Example

```
int : x
x := ?
while (x > 0) do
    x := x − 1
od
```



Piecewise-Defined Ranking Functions



The Abstract Domain of Segmented Ranking Functions

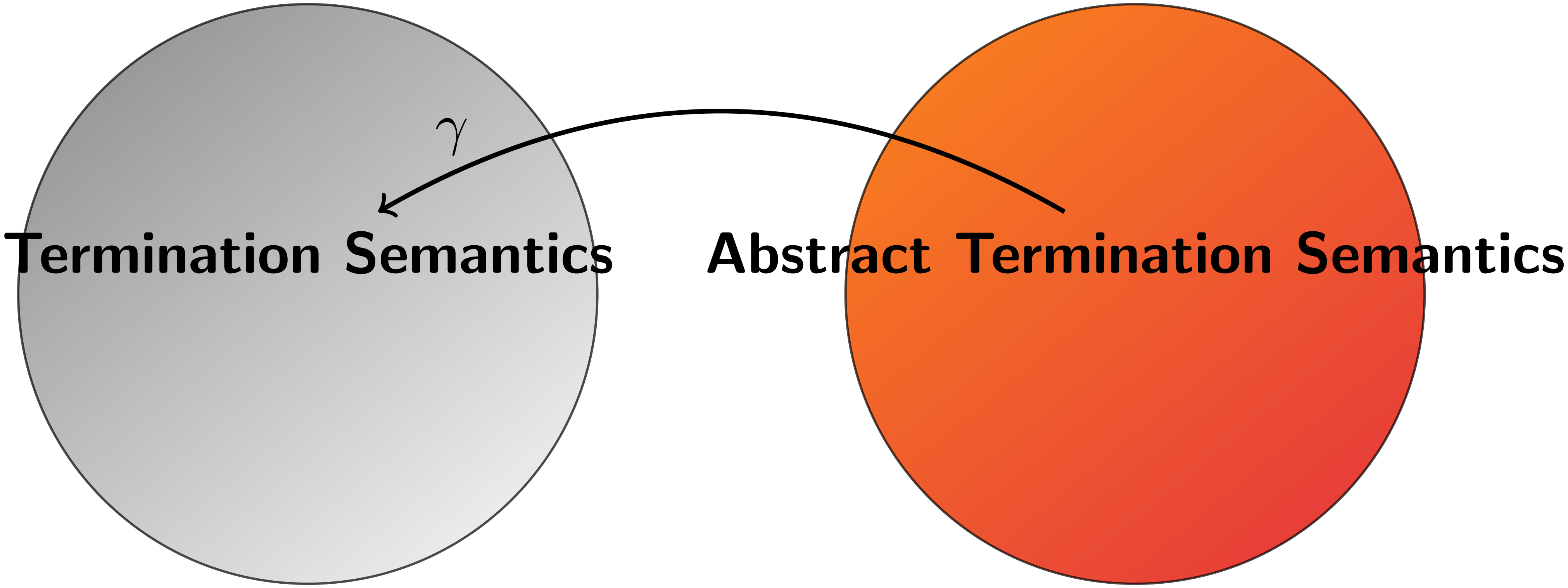
Caterina Urban

An Abstract Domain to Infer Ordinal-Valued Ranking Functions*

Caterina Urban and Antoine Miné

A Decision Tree Abstract Domain for Proving Conditional Termination*

Caterina Urban and Antoine Miné



Termination Semantics

Abstract Termination Semantics

Urban - *The Abstract Domain of Segmented Ranking Functions* (SAS 2013)

Abstractions in the concretization framework

Given a concrete $(C, \leq)$ and an abstract $(A, \sqsubseteq)$ posets and a monotonic concretization $\gamma : A \rightarrow C$

($\gamma(a)$ is the "meaning" of a in C; we use intervals in our examples)

- $a \in A$ is a sound abstraction of $c \in C$ if $c \leq \gamma(a)$.

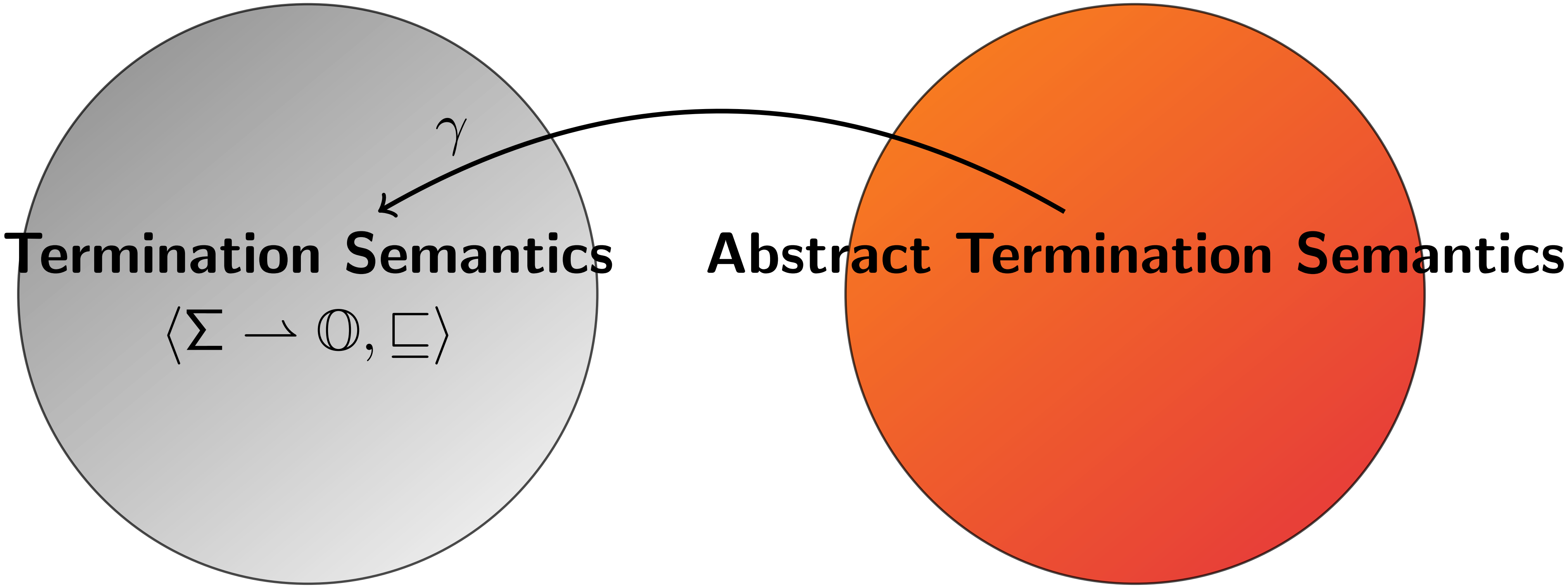
  (e.g.: $[0, 10]$ is a sound abstraction of $\{0, 1, 2, 5\}$ in the integer interval domain)

- $g : A \rightarrow A$ is a sound abstraction of $f : C \rightarrow C$ if $\forall a \in A$: $(f \circ \gamma)(a) \leq (\gamma \circ g)(a)$.

  (e.g.: $\lambda([a, b]).[-\infty, +\infty]$ is a sound abstraction of $\lambda X.\{\, x+1 \,|\, x \in X \,\}$ in the interval domain)

- $g : A \rightarrow A$ is an exact abstraction of $f : C \rightarrow C$ if $f \circ \gamma = \gamma \circ g$.

  (e.g.: $\lambda([a, b]).[a+1, b+1]$ is an exact abstraction of $\lambda X.\{\, x+1 \,|\, x \in X \,\}$ in the interval domain)



Termination Semantics

$\langle \Sigma \rightharpoonup \mathbb{O}, \sqsubseteq \rangle$

Abstract Termination Semantics

Definition (**Approximation Order**)

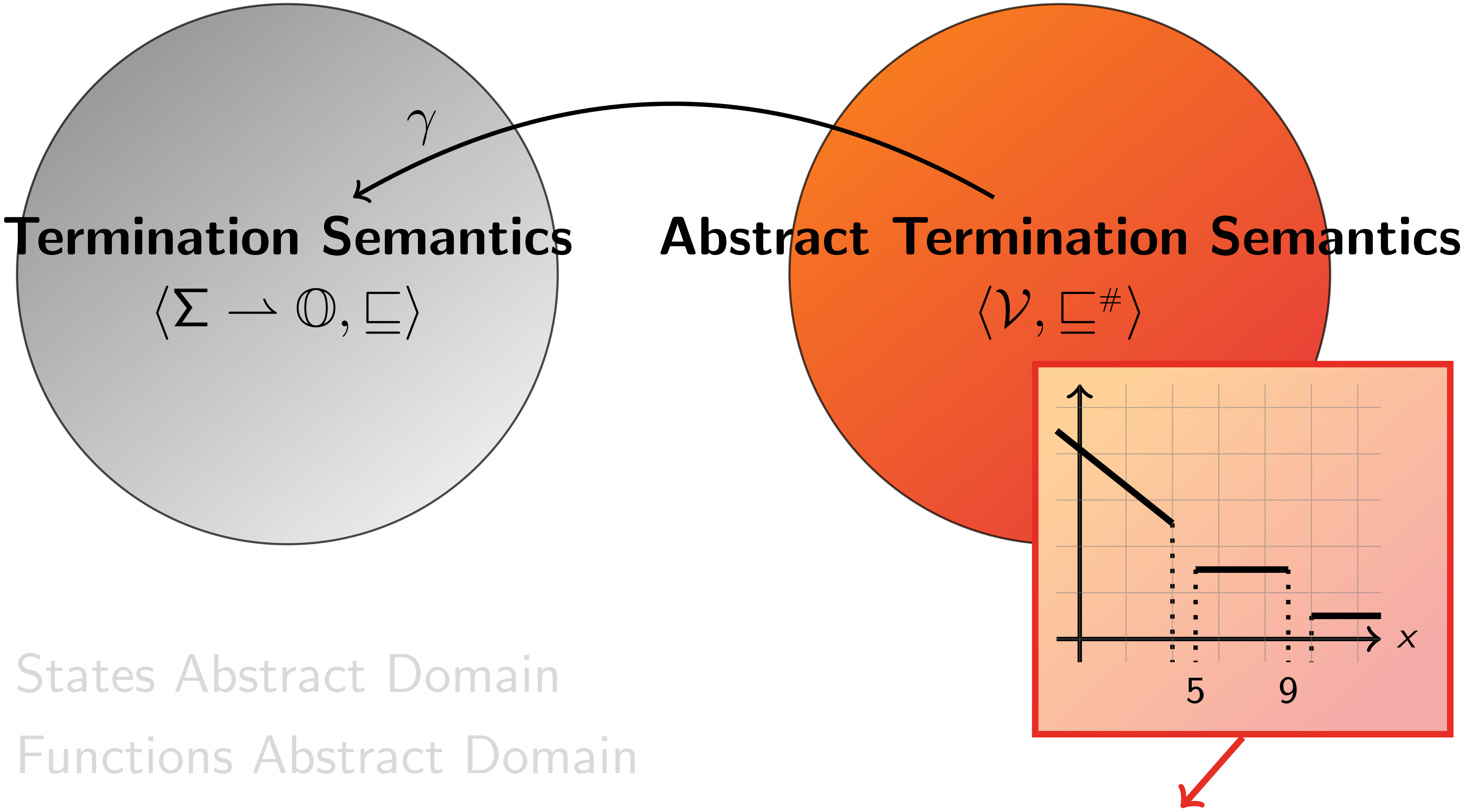
$$v_1 \sqsubseteq v_2 \overset{\text{def}}{=} \boxed{\text{dom}(v_1) \supseteq \text{dom}(v_2)} \wedge \ \forall x \in \text{dom}(v_1) : v_1(x) \leq v_2(x)$$

Urban - *The Abstract Domain of Segmented Ranking Functions* (SAS 2013)

Use of posets (informally)

Posets are a very useful notion to discuss about:

- logic: ordered by implication $\Longrightarrow$
- approximations: $\sqsubseteq$ is an information order
  (“$a \sqsubseteq b$” means: “a caries more information than b”)
- program verification: program semantics $\sqsubseteq$ specification
  (e.g.: behaviors of program $\subseteq$ accepted behaviors)
- iteration: fixpoint computation
  (e.g., a computation is directed, with a limit: $X_1 \sqsubseteq X_2 \sqsubseteq \cdots \sqsubseteq X_n$)



Termination Semantics

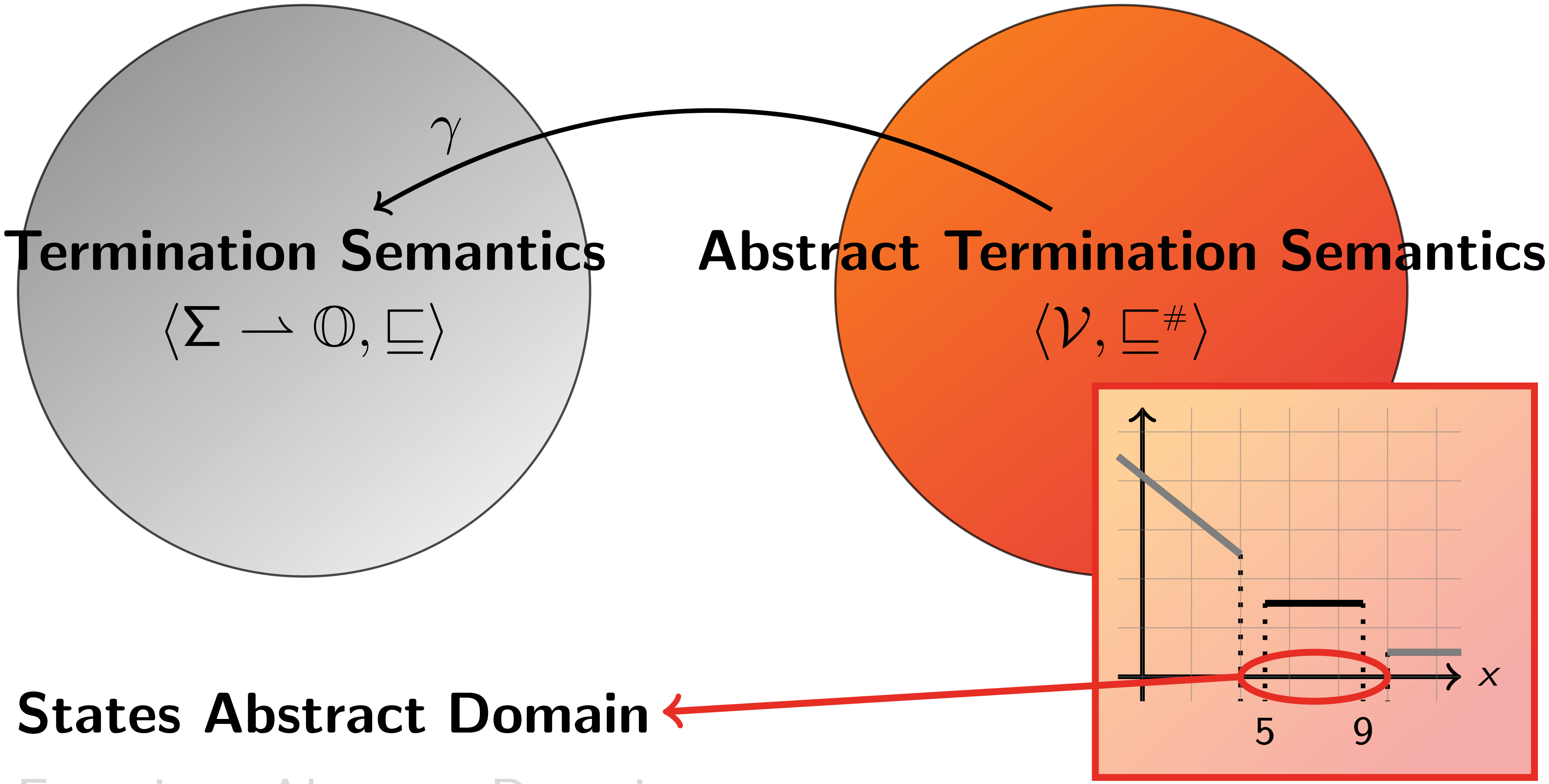
$\langle \Sigma \rightharpoonup \mathbb{O}, \sqsubseteq \rangle$

Abstract Termination Semantics

$\langle \mathcal{V}, \sqsubseteq^{\#} \rangle$

- States Abstract Domain S
- Functions Abstract Domain F
- **Piecewise-Defined Ranking Functions Abstract Domain** V(S, F)

Urban - *The Abstract Domain of Segmented Ranking Functions* (SAS 2013)

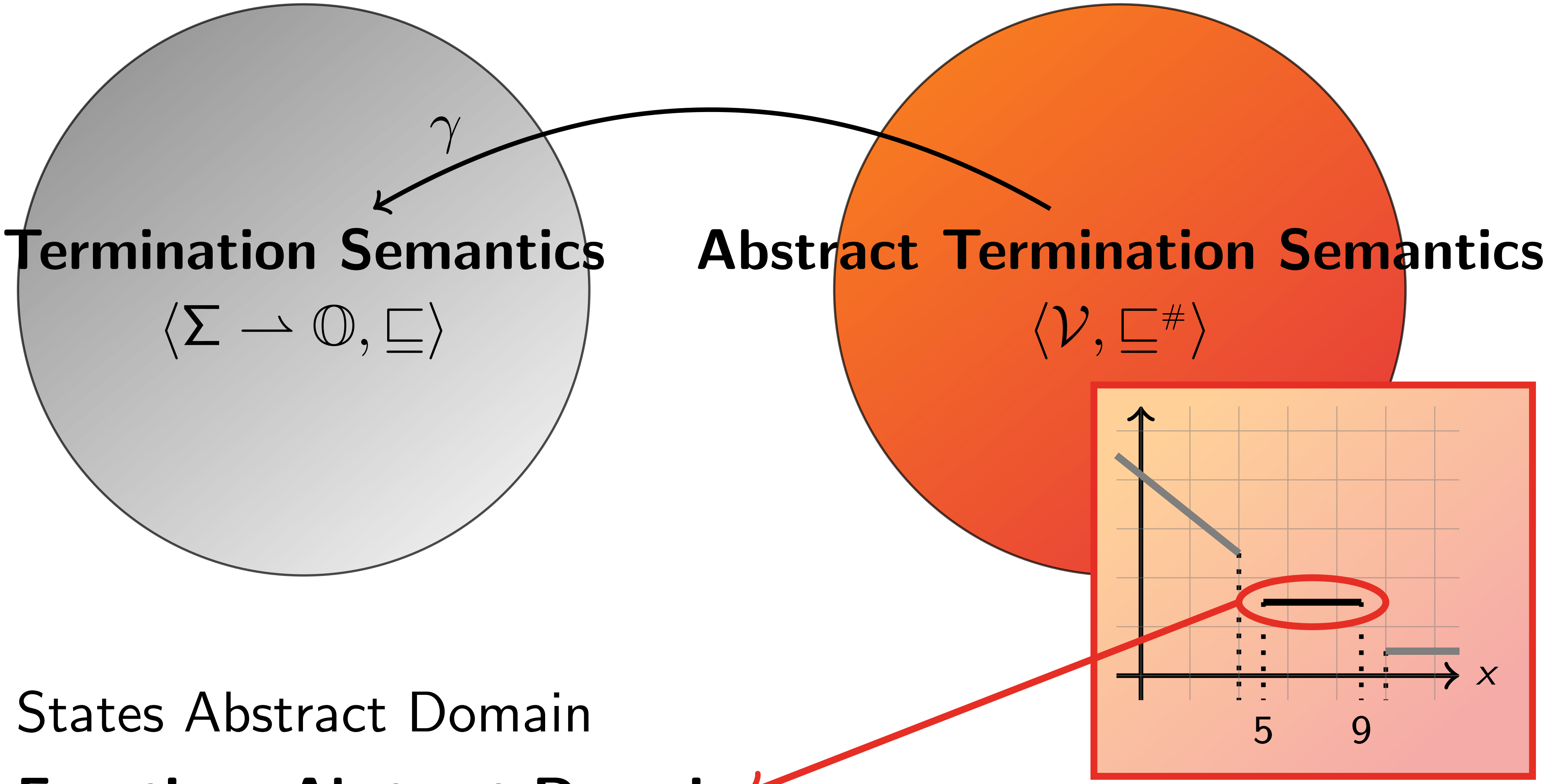


Termination Semantics
$\langle \Sigma \rightharpoonup \mathbb{O}, \sqsubseteq \rangle$

Abstract Termination Semantics
$\langle \mathcal{V}, \sqsubseteq^{\#} \rangle$

States Abstract Domain	S
Functions Abstract Domain	F
Piecewise-Defined Ranking Functions Abstract Domain	V(S, F)

Urban - *The Abstract Domain of Segmented Ranking Functions* (SAS 2013)



γ

Termination Semantics

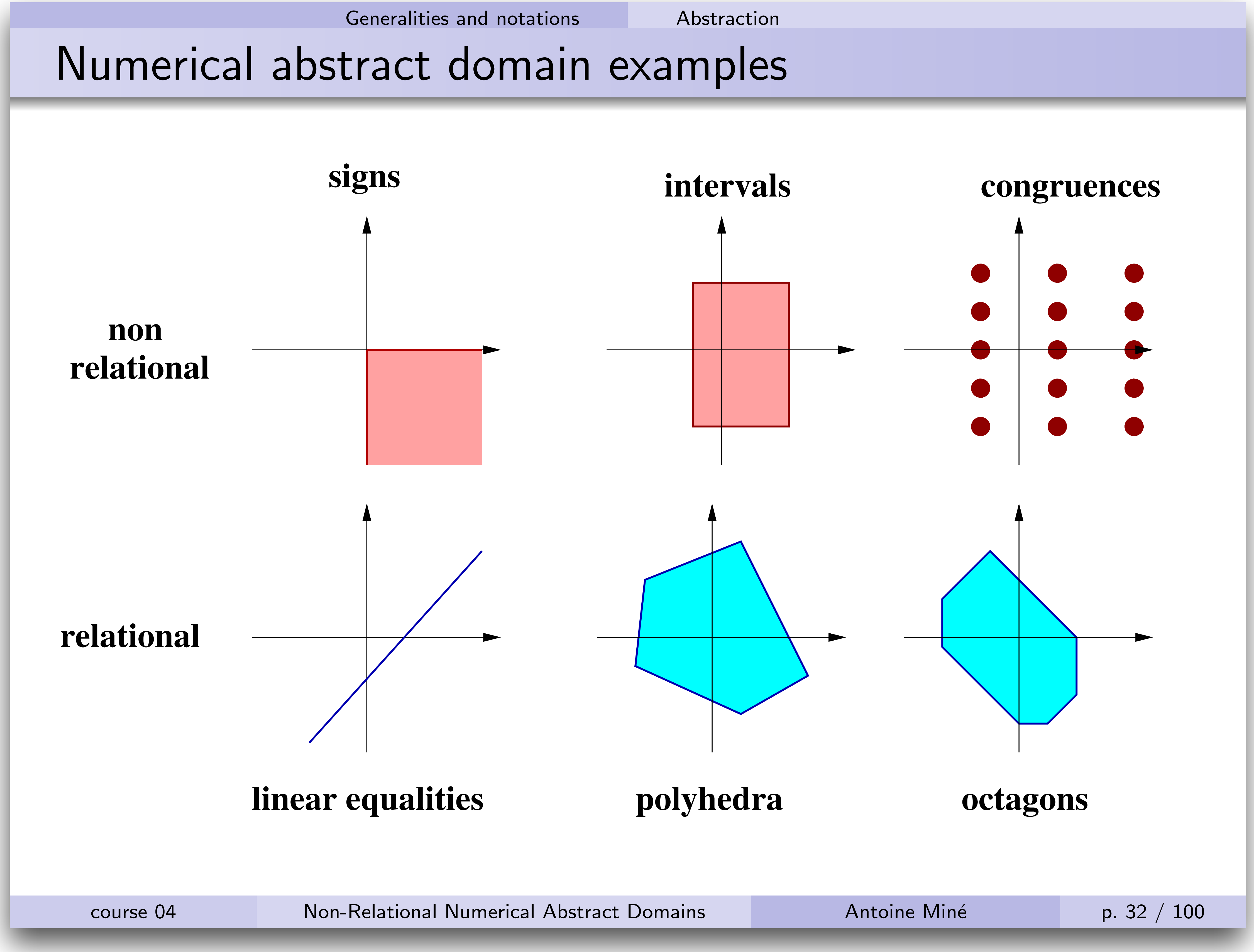
$\langle \Sigma \rightharpoonup \mathbb{O}, \sqsubseteq \rangle$

Abstract Termination Semantics

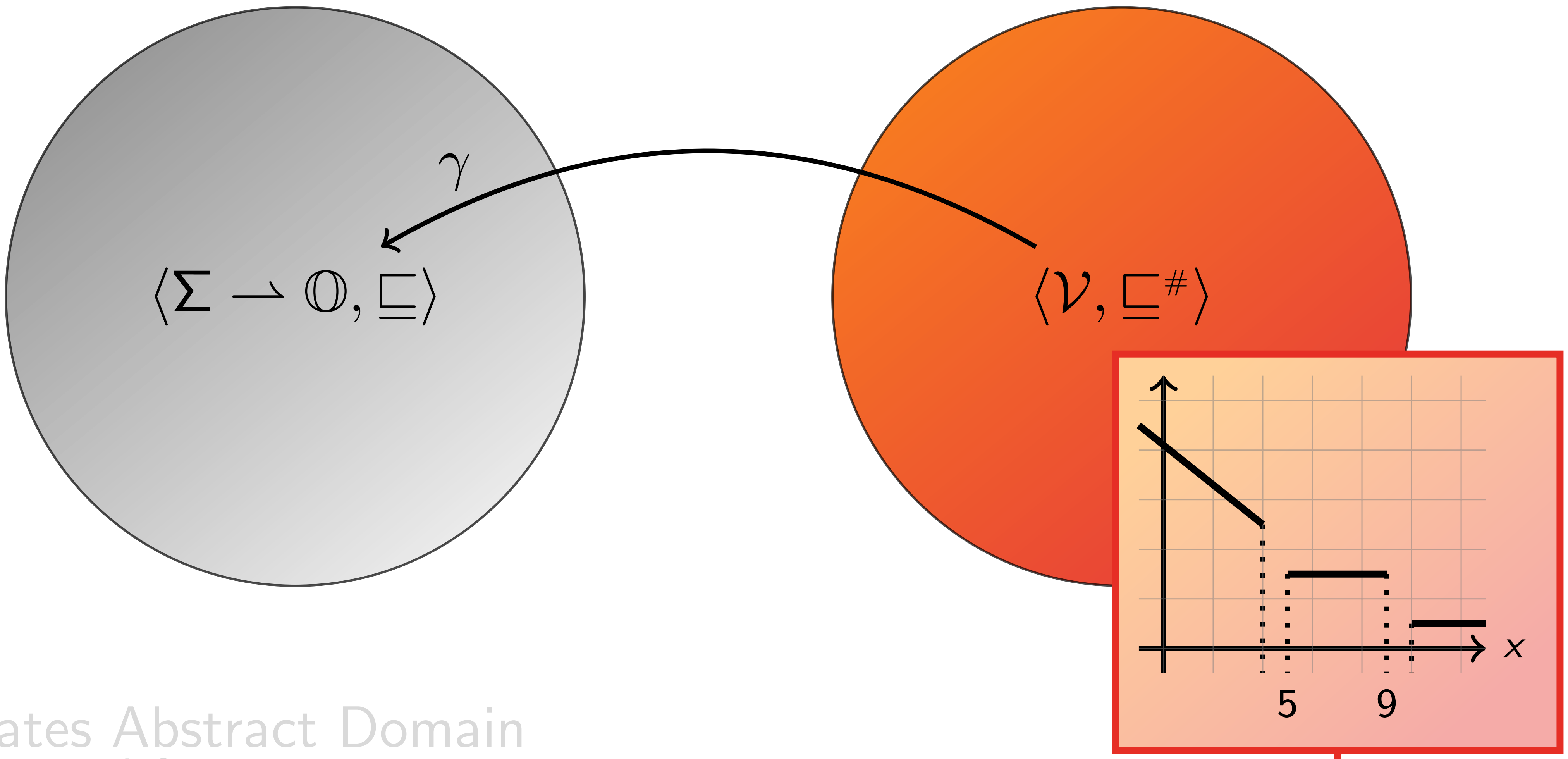
$\langle \mathcal{V}, \sqsubseteq^{\#} \rangle$

States Abstract Domain	S
Functions Abstract Domain	F
Piecewise-Defined Ranking Functions Abstract Domain	V(S, F)

Urban - *The Abstract Domain of Segmented Ranking Functions* (SAS 2013)

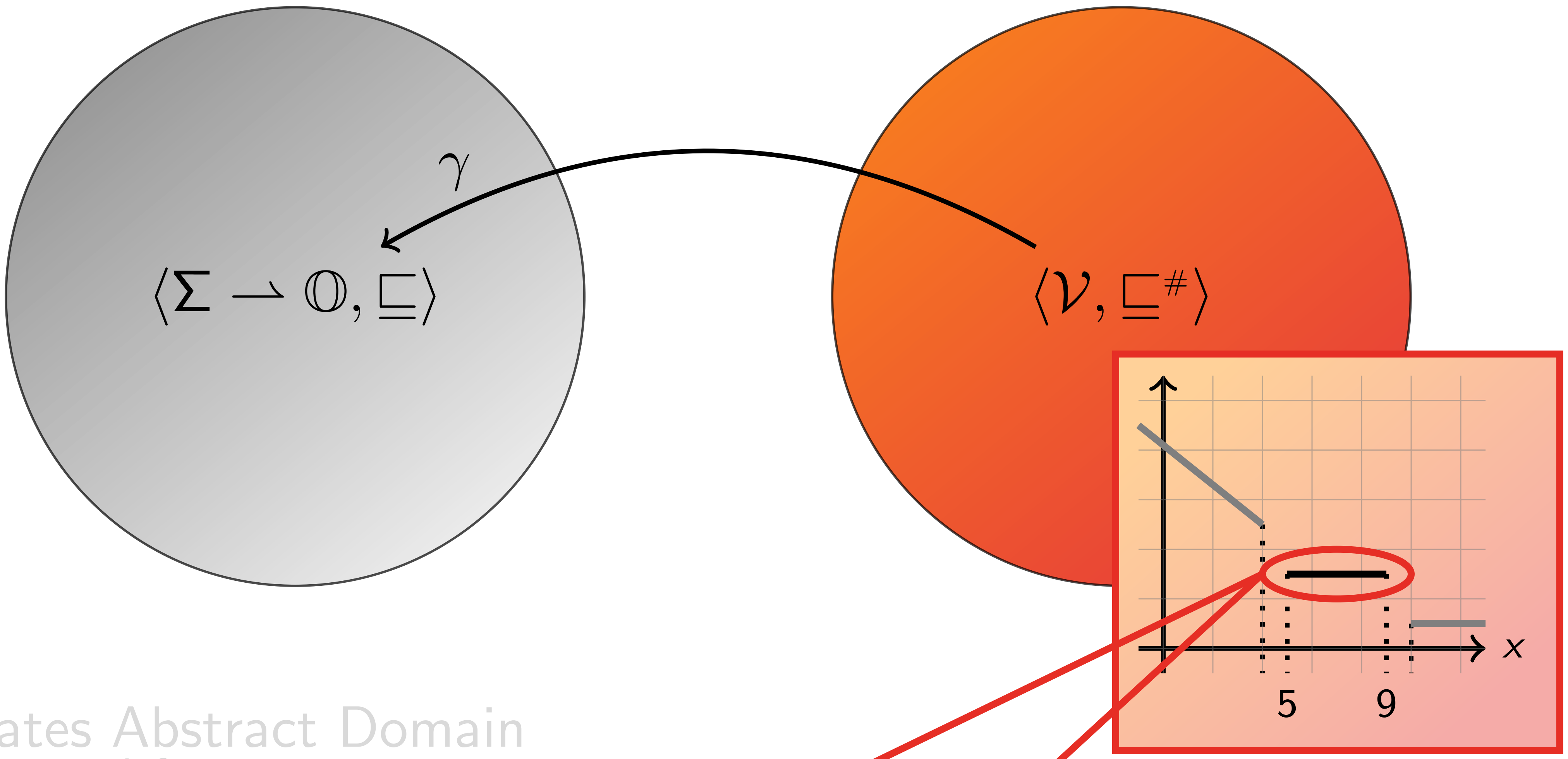


Numerical abstract domain examples



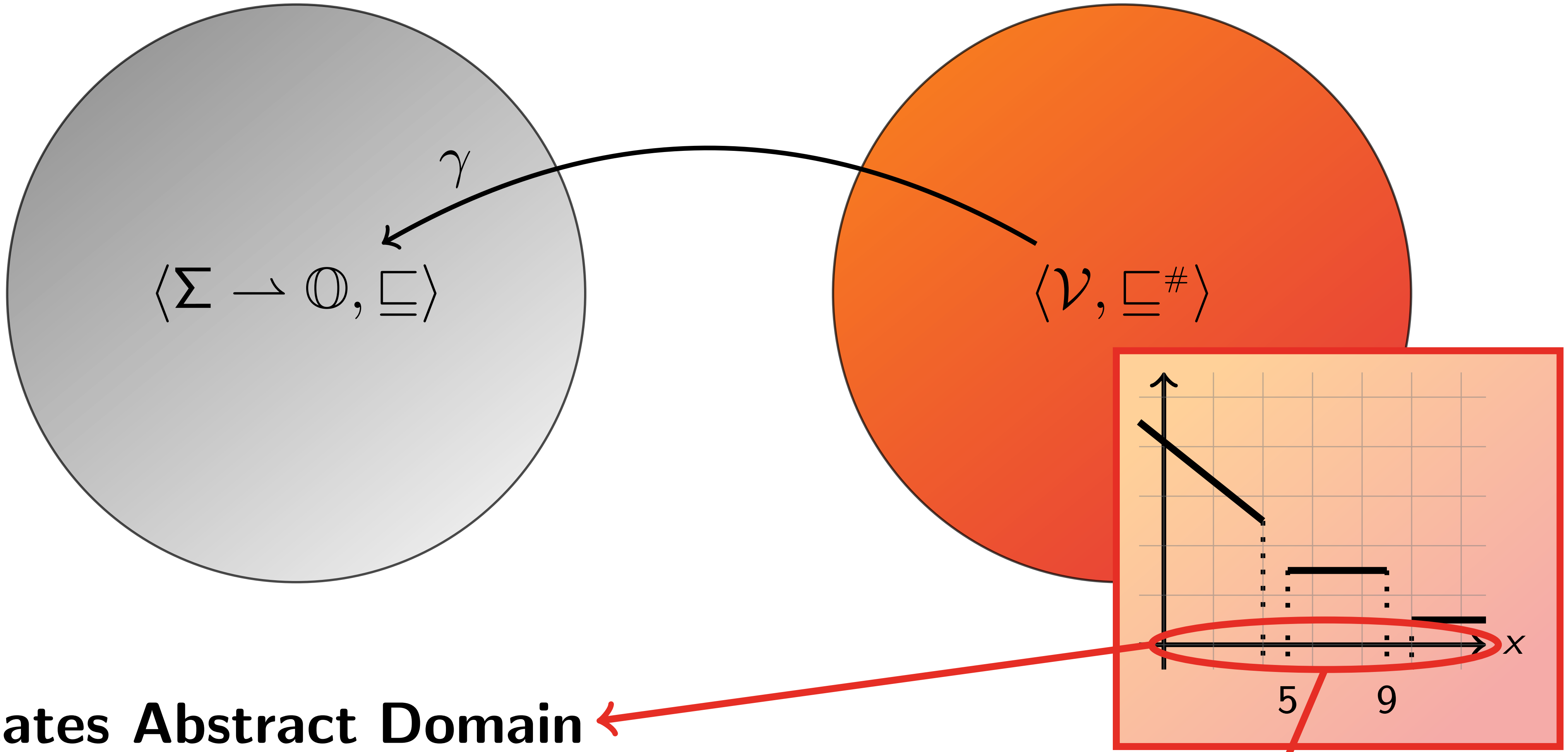
- States Abstract Domain
  - $\mathcal{L} \overset{\text{def}}{=}$ Interval/Octagonal/Polyhedral Linear Constraints
- Functions Abstract Domain
  - $\mathcal{F} \overset{\text{def}}{=} \{\bot\} \cup \{f \mid f \in \mathbb{Z}^n \to \mathbb{N}\} \cup \{\top\}$
    where $f \equiv f(x_1, \dots, x_n) = m_1 x_1 + \cdots + m_n x_n + q$
- **Piecewise-Defined Ranking Functions Abstract Domain**
  - $\mathcal{V} \overset{\text{def}}{=} \{\text{LEAF} : f \mid f \in \mathcal{F}\} \cup \{\text{NODE}\{c\} : t_1, t_2 \mid c \in \mathcal{L} \wedge t_1, t_2 \in \mathcal{V}\}$

Urban&Miné - *A Decision Tree Abstract Domain for Proving Conditional Termination* (SAS 2014)



- States Abstract Domain
  - $\mathcal{L} \overset{\text{def}}{=}$ Interval/Octagonal/Polyhedral Linear Constraints
- **Functions Abstract Domain**
  - $\mathcal{F} \overset{\text{def}}{=} \{\bot\} \cup \{f \mid f \in \mathbb{Z}^n \to \mathbb{N}\} \cup \{\top\}$
    where $f \equiv f(x_1, \ldots, x_n) = m_1 x_1 + \cdots + m_n x_n + q$
- Piecewise-Defined Ranking Functions Abstract Domain
  - $\mathcal{V} \overset{\text{def}}{=} \{\mathbf{LEAF : f \mid f \in \mathcal{F}}\} \cup \{\mathrm{NODE}\{c\} : t_1, t_2 \mid c \in \mathcal{L} \wedge t_1, t_2 \in \mathcal{V}\}$

Urban&Miné - *A Decision Tree Abstract Domain for Proving Conditional Termination* (SAS 2014)



- **States Abstract Domain**
  - $\mathcal{L} \overset{\text{def}}{=}$ Interval/Octagonal/Polyhedral Linear Constraints
- Functions Abstract Domain
  - $\mathcal{F} \overset{\text{def}}{=} \{\bot\} \cup \{f \mid f \in \mathbb{Z}^n \to \mathbb{N}\} \cup \{\top\}$
    where $f \equiv f(x_1, \ldots, x_n) = m_1 x_1 + \cdots + m_n x_n + q$
- Piecewise-Defined Ranking Functions Abstract Domain
  - $\mathcal{V} \overset{\text{def}}{=} \{\text{LEAF} : f \mid f \in \mathcal{F}\} \cup \{\textbf{NODE}\{\mathbf{c}\} : \mathbf{t_1}, \mathbf{t_2} \mid \mathbf{c} \in \boldsymbol{\mathcal{L}} \wedge \mathbf{t_1}, \mathbf{t_2} \in \boldsymbol{\mathcal{V}}\}$

Urban&Miné - *A Decision Tree Abstract Domain for Proving Conditional Termination* (SAS 2014)

Example

```
int : $x$,  $y$
while [1]($x > 0$) do
    [2]$x := x - y$
od[3]
```

Example

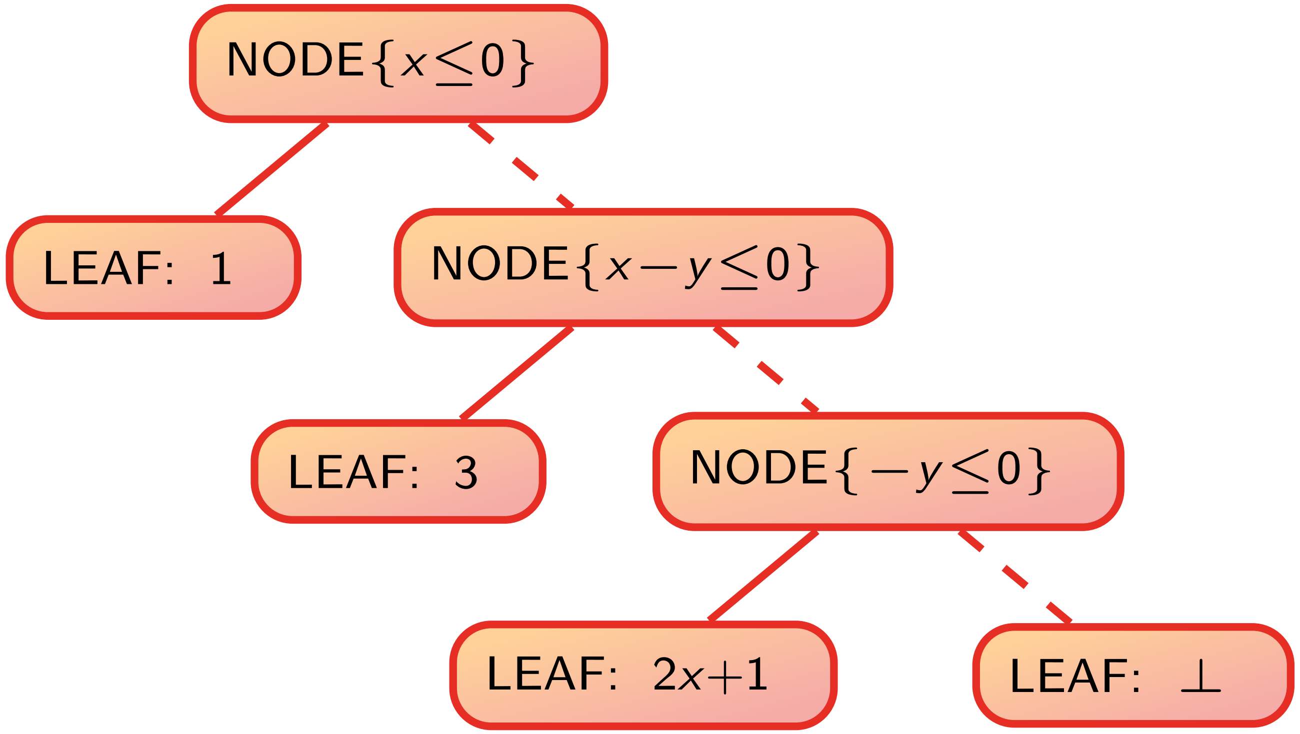
```
int : $x$,  $y$
while $^1$($x > 0$) do
    $^2 x := x - y$
od$^3$
```

the program terminates if and only if $\mathbf{x \leq 0 \vee y > 0}$

Example

```
int : $x, y$
while $^1(x > 0)$ do
    $^2x := x - y$
od$^3$
```

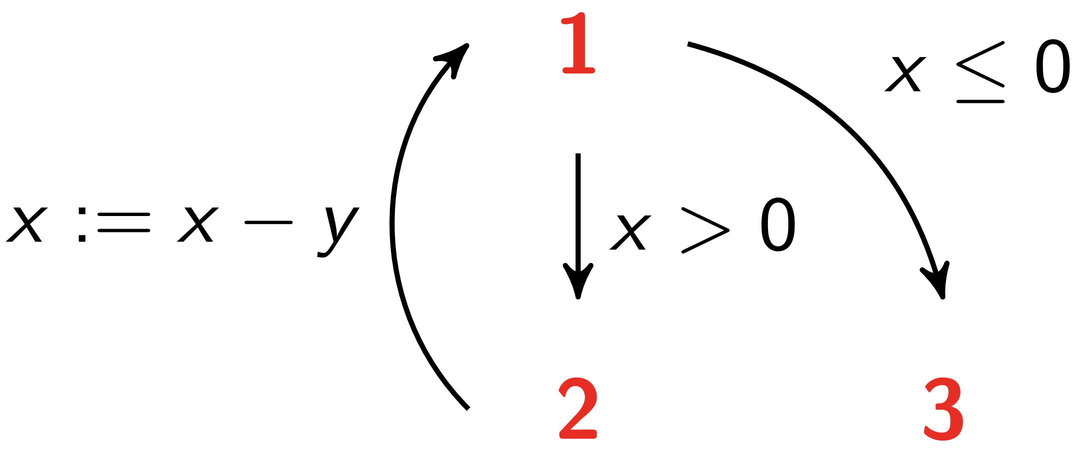
the program terminates if and only if $\mathbf{x \leq 0 \vee y > 0}$


NODE$\{x \leq 0\}$
LEAF: 1
NODE$\{x - y \leq 0\}$
LEAF: 3
NODE$\{-y \leq 0\}$
LEAF: $2x+1$
LEAF: $\bot$

Example

```
int : $x$, $y$
while $^1(x > 0)$ do
    $^2x := x - y$
od$^3$
```

we will map each point to a function of x and y giving an **upper bound** on the steps before termination



1

$x \leq 0$

$x := x - y$

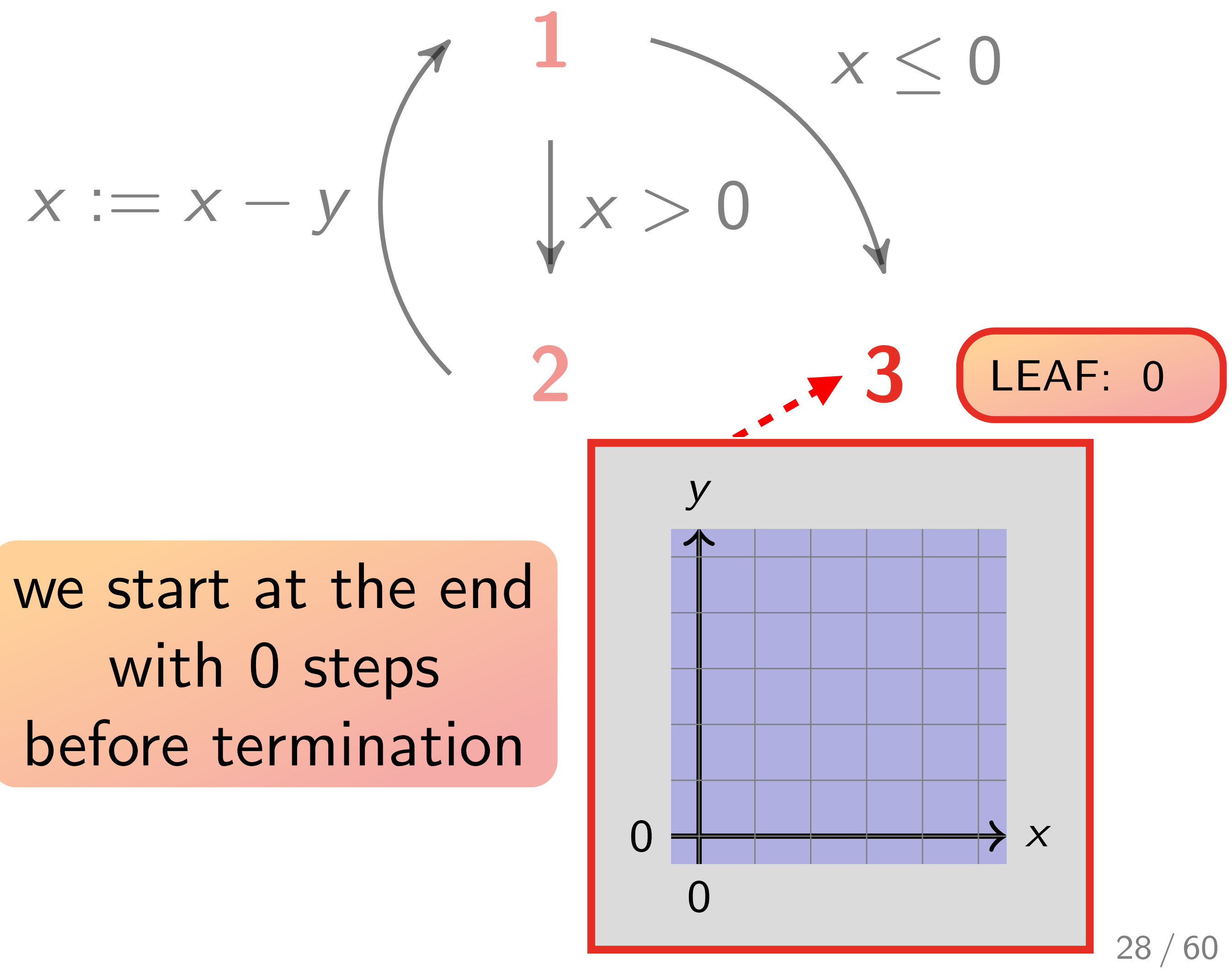
$x > 0$

2

3

Example

```
int : $x$, $y$
while $^1(x > 0)$ do
    $^2x := x - y$
od$^3$
```



we start at the end with 0 steps before termination

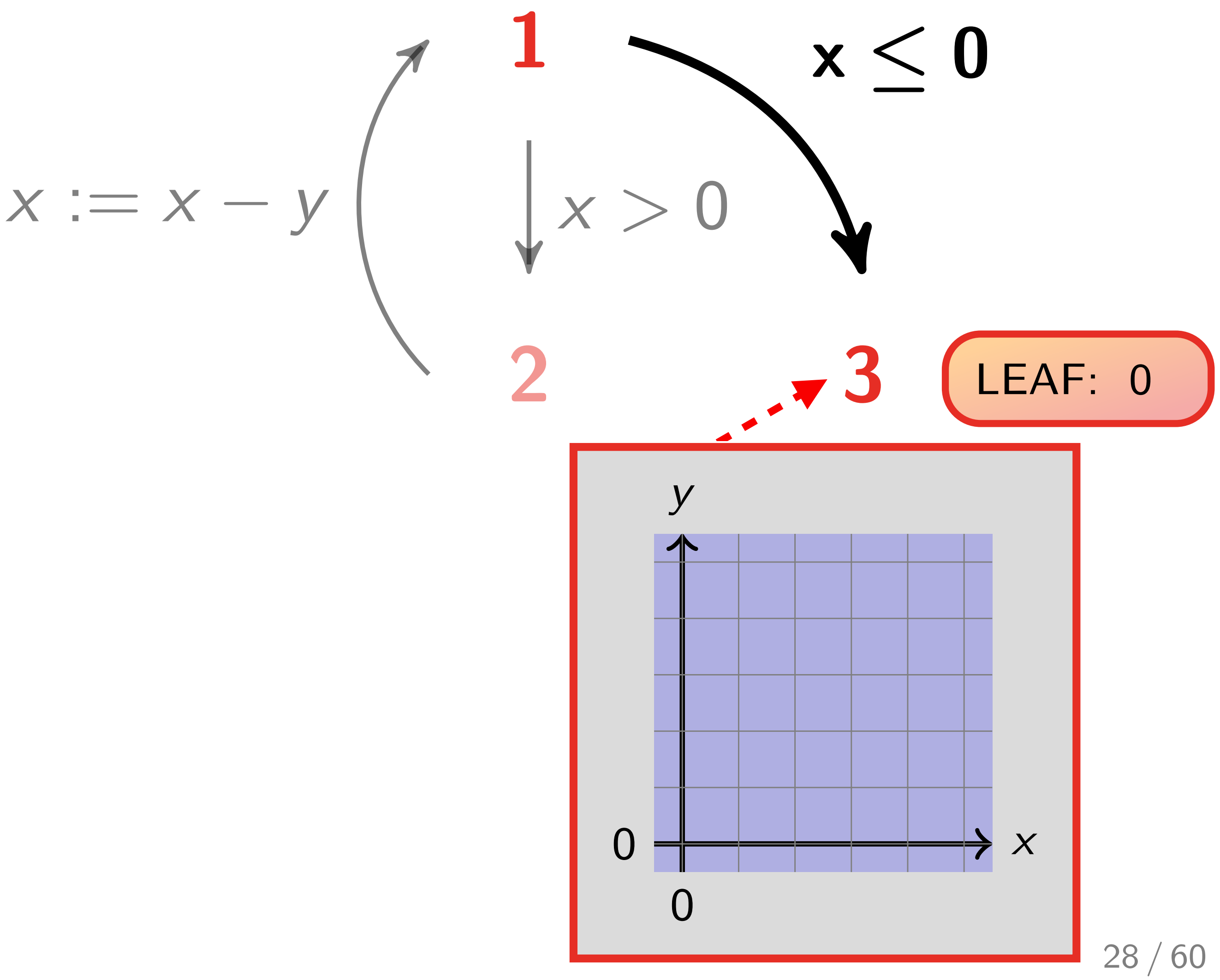
Example

int : x, y

while ${}^{1}(x > 0)$ do

  ${}^{2}x := x - y$

od^{3}



$x := x - y$

1

$x > 0$

$\mathbf{x \leq 0}$

2

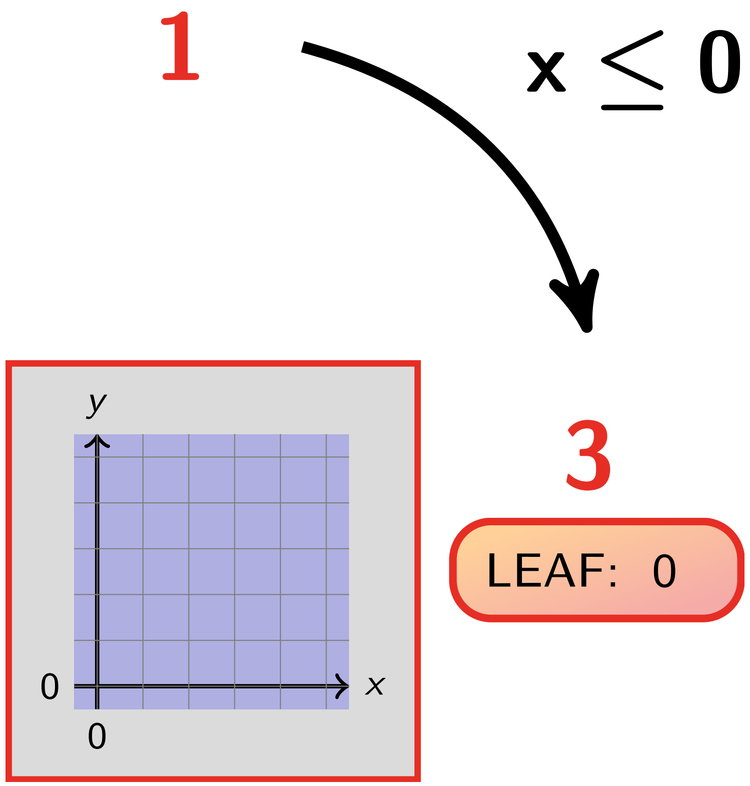
3

LEAF: 0

Tests

Algorithm 4 : Tree Filter

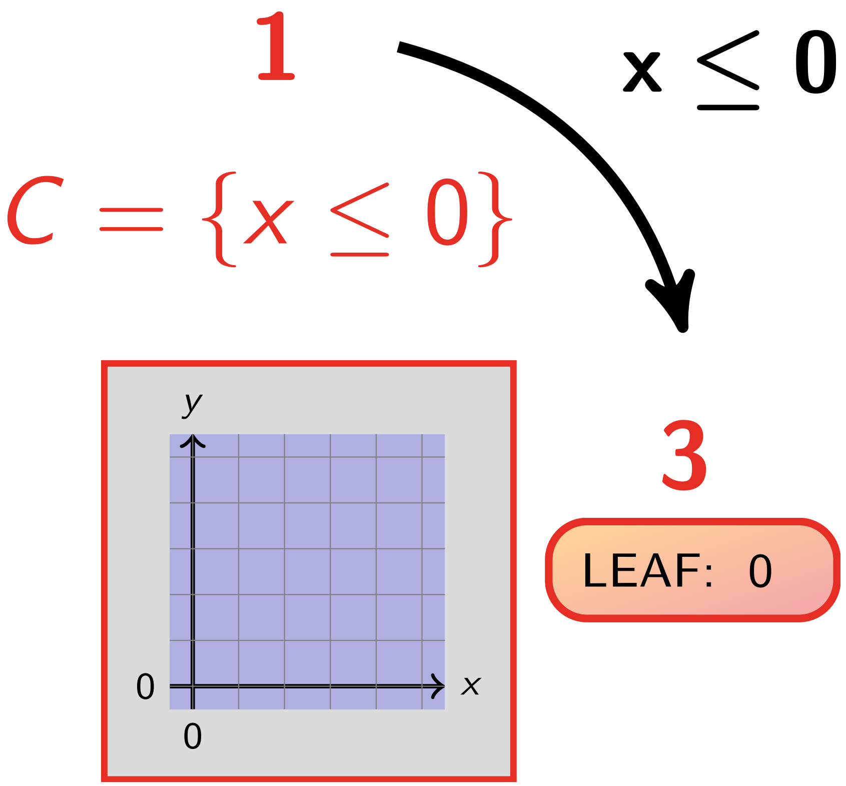
```
function FILTER-AUX($t$,$c$)
    if ISLEAF($t$) then return LEAF : FILTER$_F$($f, c$)          /* $t \triangleq$ LEAF : $f$ */
    else return NODE{$t.c$} : FILTER-AUX($t.l, c$); FILTER-AUX($t.r, c$)

function FILTER($t$,$c$)
    $C \leftarrow$ FILTER$_L$($c$)
    return AUGMENT(FILTER-AUX($t, c$), $C$)
```



Tests

Algorithm 4 : Tree Filter

```
function FILTER-AUX($t$,$c$)
    if ISLEAF($t$) then return LEAF : FILTER$_F$($f, c$)        /* $t \triangleq$ LEAF : $f$ */
    else return NODE{$t.c$} : FILTER-AUX($t.l, c$); FILTER-AUX($t.r, c$)

function FILTER($t$,$c$)
    $C \leftarrow$ FILTER$_L$($c$)
    return AUGMENT(FILTER-AUX($t, c$), $C$)
```



1

$C = \{x \leq 0\}$

$\mathbf{x \leq 0}$

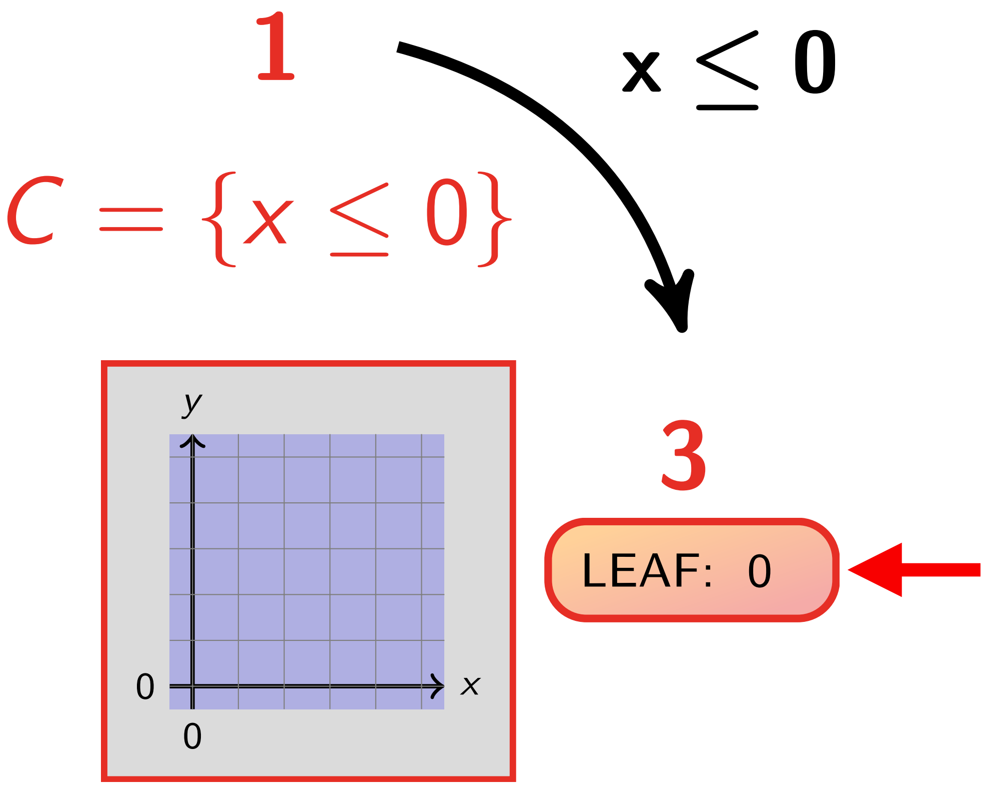
3

LEAF: 0

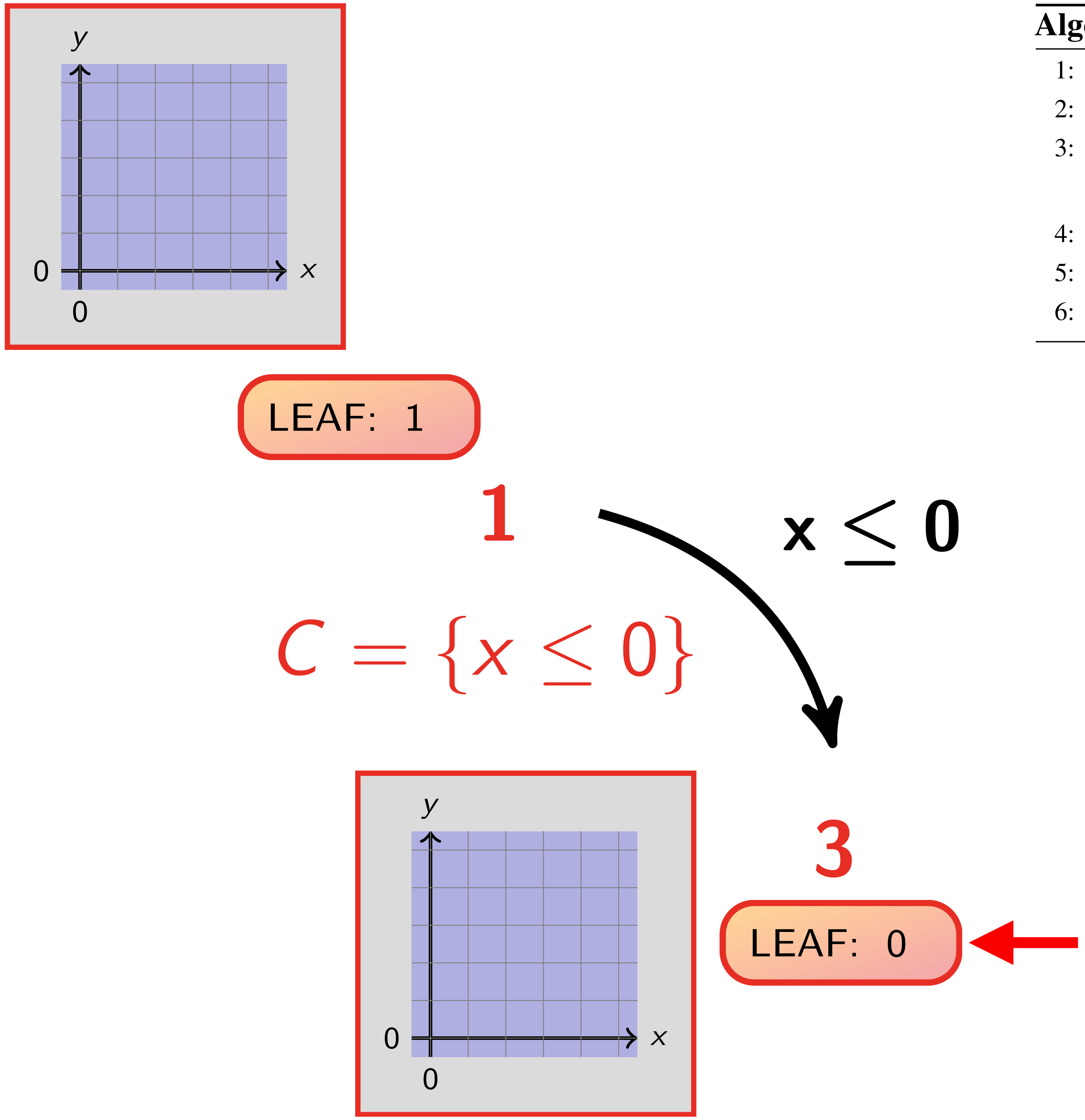
Tests

Algorithm 4 : Tree Filter

```
function FILTER-AUX($t$,$c$)
    if ISLEAF($t$) then return LEAF : FILTER$_F$($f, c$)        /* $t \triangleq$ LEAF : $f$ */
    else return NODE$\{t.c\}$ : FILTER-AUX($t.l, c$); FILTER-AUX($t.r, c$)

function FILTER($t$,$c$)
    $C \leftarrow$ FILTER$_L$($c$)
    return AUGMENT(FILTER-AUX($t, c$), $C$)
```



Tests



LEAF: 1

1

$C = \{x \leq 0\}$

$\mathbf{x \leq 0}$

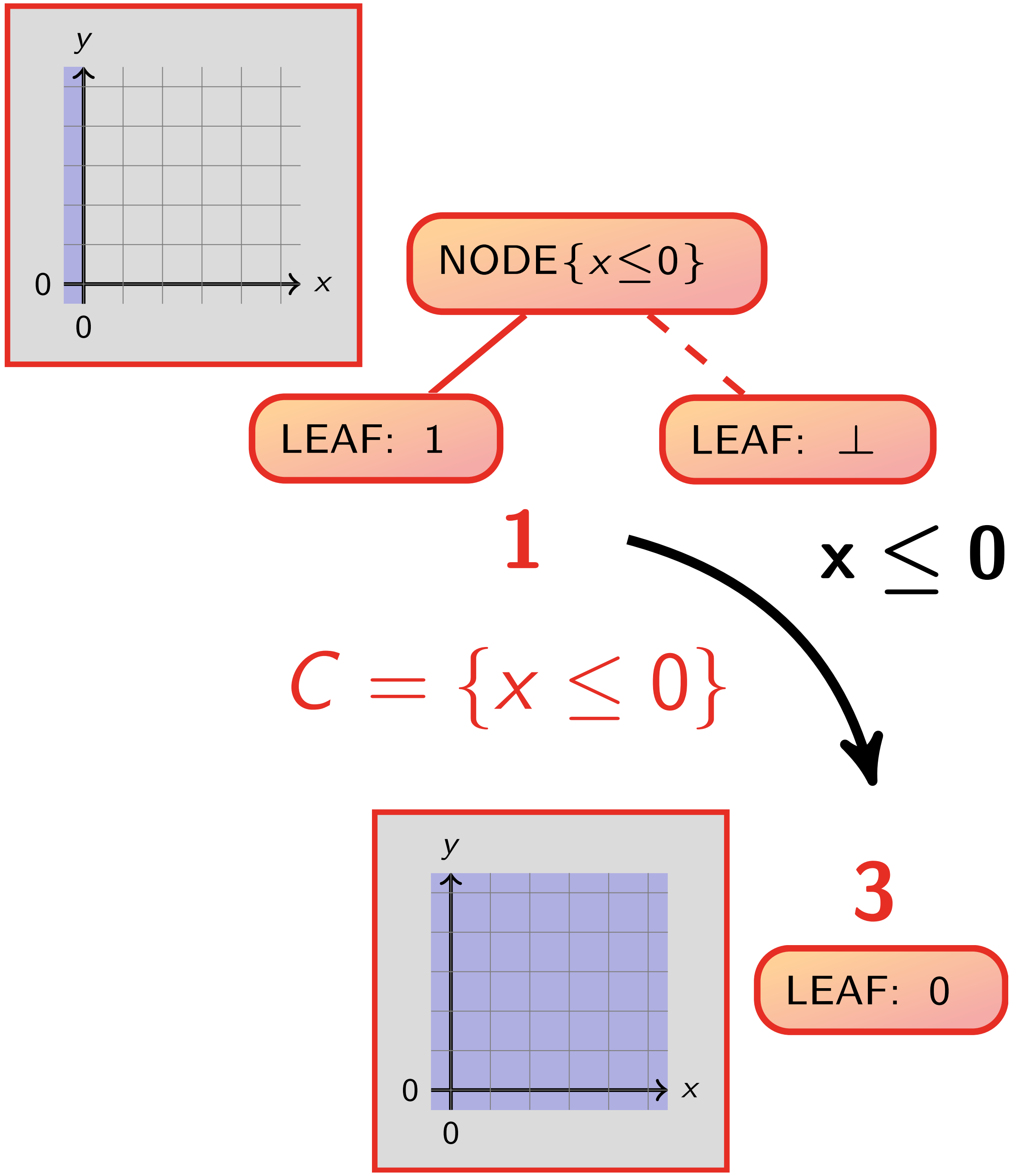
3

LEAF: 0

Algorithm 4 : Tree Filter

```
function FILTER-AUX($t$,$c$)
    if ISLEAF($t$) then return LEAF : FILTER_F($f, c$)        /* $t \triangleq$ LEAF : $f$ */
    else return NODE{$t.c$} : FILTER-AUX($t.l, c$); FILTER-AUX($t.r, c$)

function FILTER($t$,$c$)
    $C \leftarrow$ FILTER_L($c$)
    return AUGMENT(FILTER-AUX($t, c$), $C$)
```

Tests

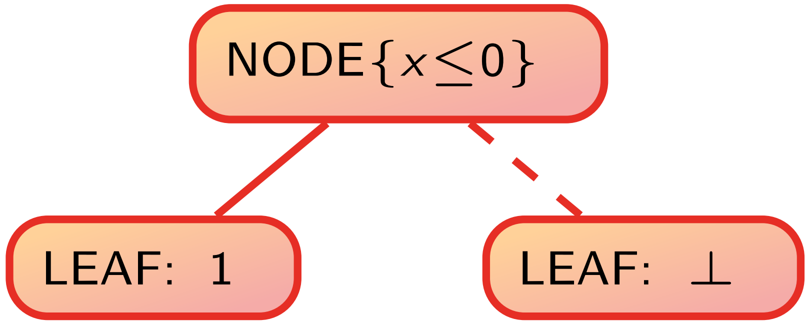


Algorithm 4 : Tree Filter

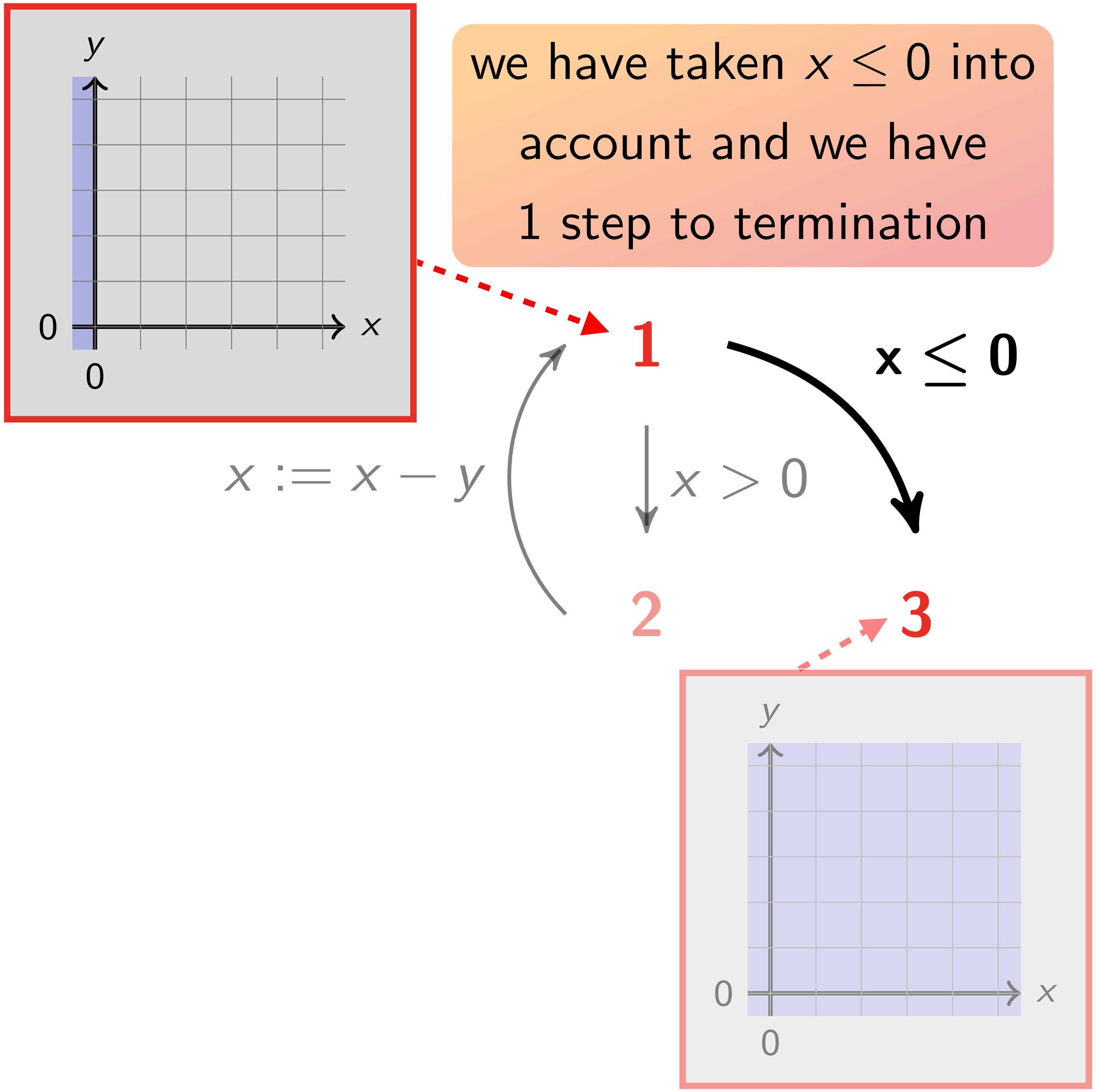
```
function FILTER-AUX($t$,$c$)
    if ISLEAF($t$) then return LEAF : FILTER_F($f$, $c$)        /* $t \triangleq$ LEAF : $f$ */
    else return NODE{$t.c$} : FILTER-AUX($t.l$, $c$); FILTER-AUX($t.r$, $c$)

function FILTER($t$,$c$)
    $C \leftarrow$ FILTER_L($c$)
    return AUGMENT(FILTER-AUX($t$, $c$), $C$)
```

Example

```
int : $x$, $y$
while $^{1}(x > 0)$ do
    $^{2}x := x - y$
od$^{3}$
```

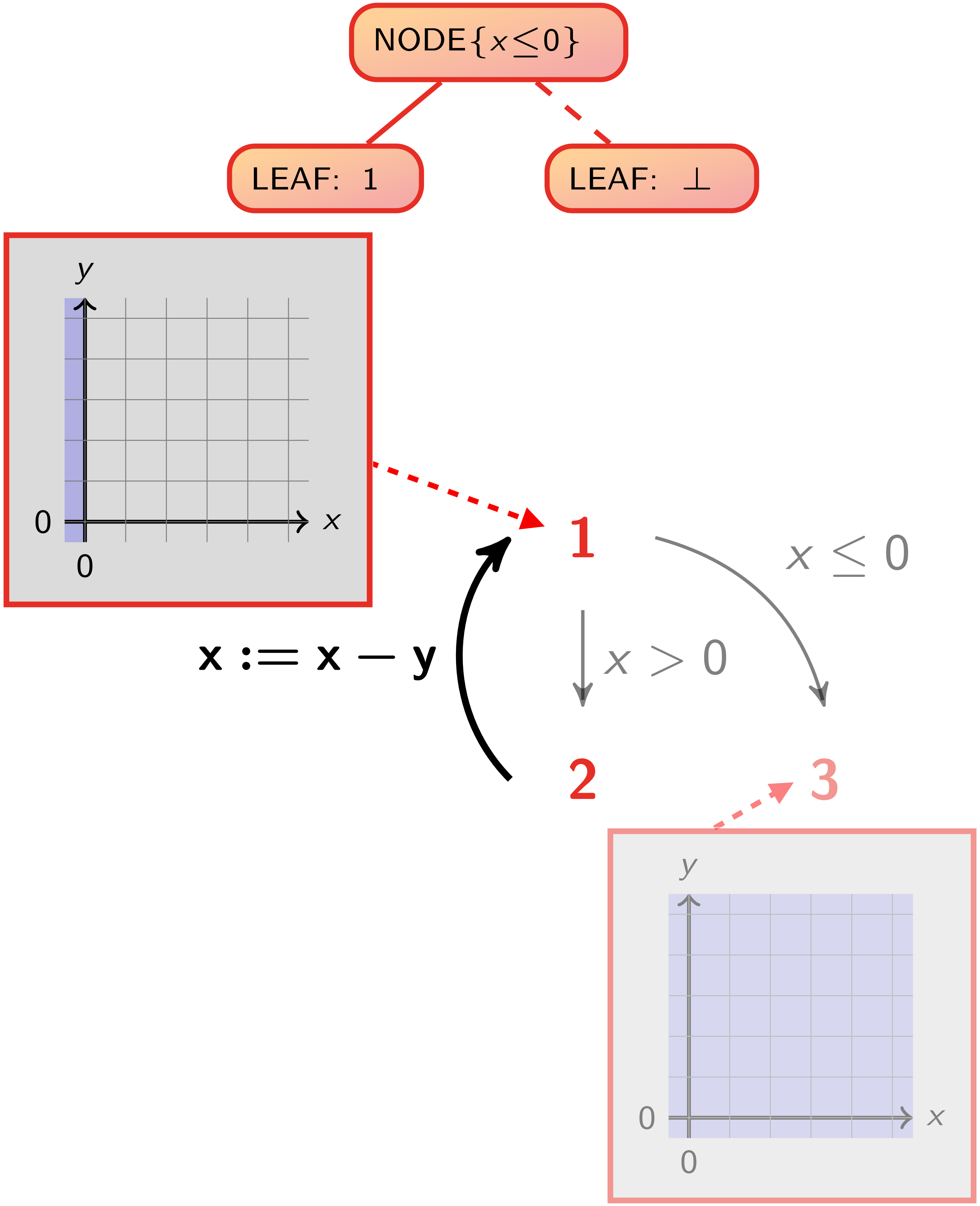


we have taken $x \leq 0$ into account and we have 1 step to termination

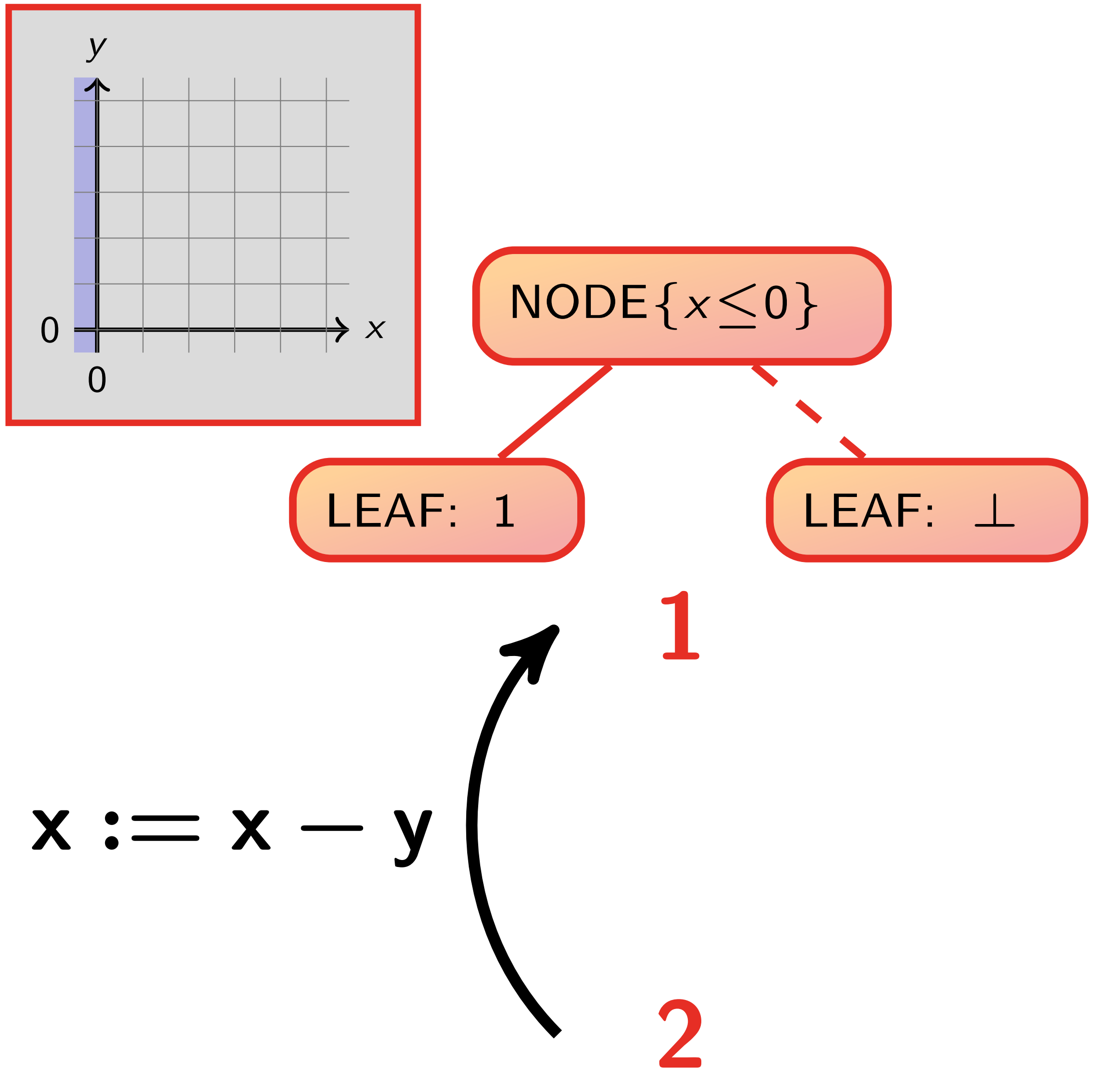


Example

```
int : $x$, $y$
while $^{1}(x > 0)$ do
    $^{2}x := x - y$
od$^{3}$
```



Assignments



NODE$\{x \leq 0\}$

LEAF: 1

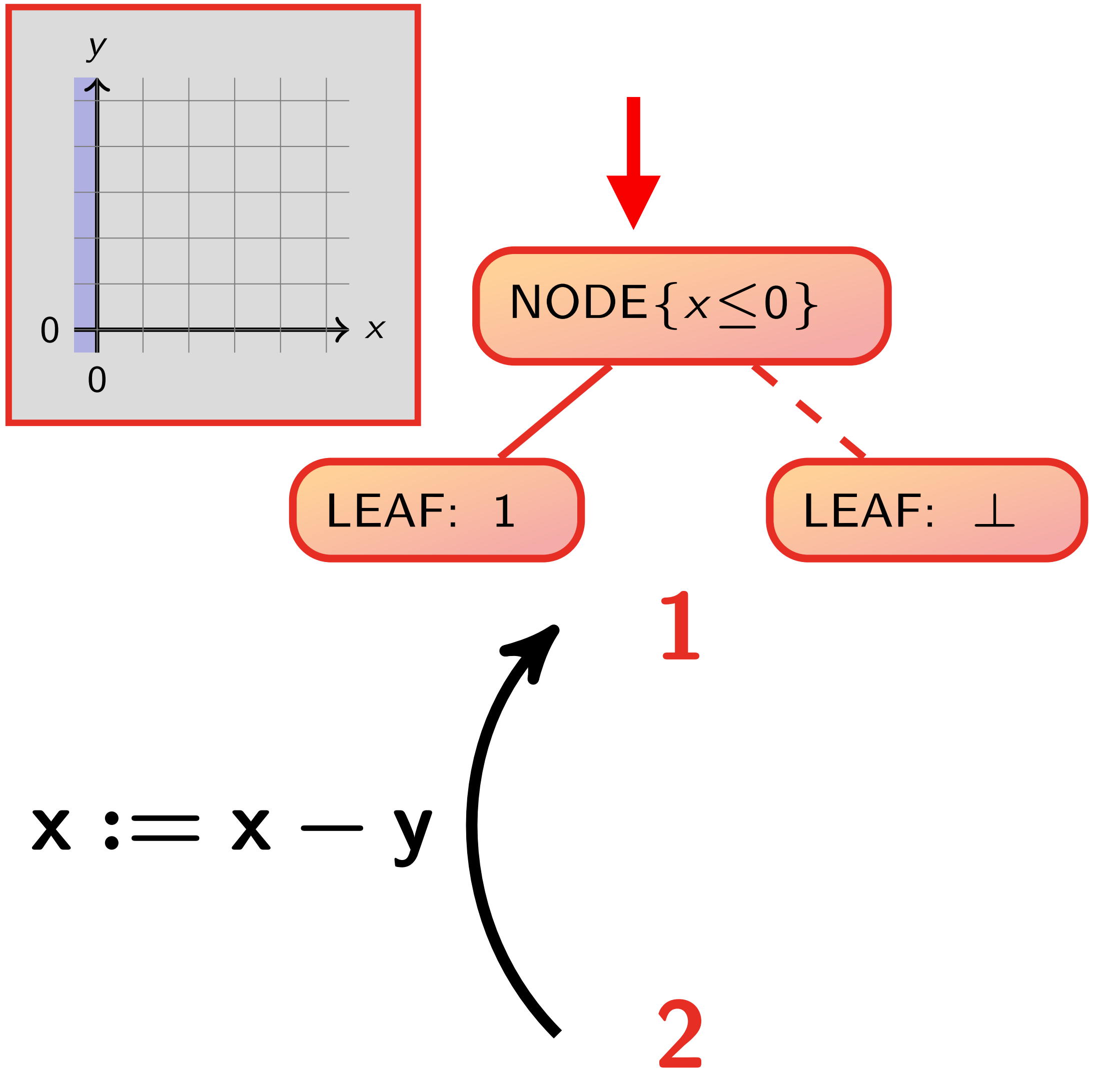
LEAF: $\perp$

x := x − y

Algorithm 3 : Tree Assign

```
function ASSIGN($t$,$x := a$)
    if ISLEAF($t$) then return LEAF : ASSIGN_F($f, x := a$)          /* $t \triangleq$ LEAF : $f$ */
    else
        $C \leftarrow$ ASSIGN_L($t.c, x := a$)
        if ISEMPTY($C$) then return ASSIGN($t.l, x := a$) $\sqcup_T$ ASSIGN($t.r, x := a$)
        else if ISUNSAT($C$) then return ASSIGN($t.r, x := a$)
        else
            $l \leftarrow$ AUGMENT(ASSIGN($t.l, x := a$), $C$)
            $r \leftarrow$ AUGMENT(ASSIGN($t.r, x := a$), $C$)
            return NODE$\{l.c\}$ : $l$; $r$
```

Assignments

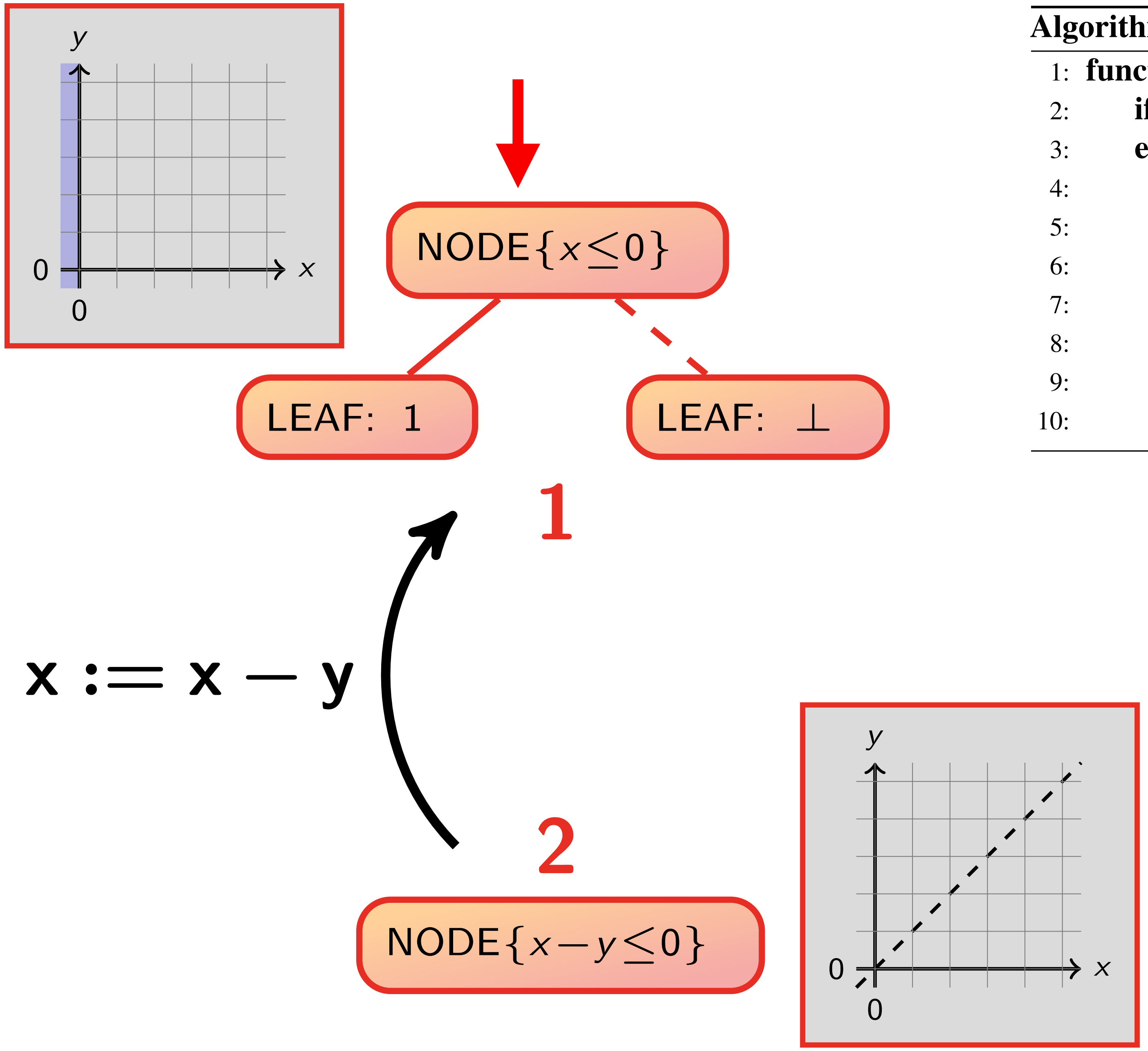


x := x − y

Algorithm 3 : Tree Assign

```
function ASSIGN($t$,$x := a$)
    if ISLEAF($t$) then return LEAF : ASSIGN_F($f, x := a$)        /* $t \triangleq$ LEAF : $f$ */
    else
        $C \leftarrow$ ASSIGN_L($t.c, x := a$)
        if ISEMPTY($C$) then return ASSIGN($t.l, x := a$) $\sqcup_T$ ASSIGN($t.r, x := a$)
        else if ISUNSAT($C$) then return ASSIGN($t.r, x := a$)
        else
            $l \leftarrow$ AUGMENT(ASSIGN($t.l, x := a$), $C$)
            $r \leftarrow$ AUGMENT(ASSIGN($t.r, x := a$), $C$)
            return NODE{$l.c$} : $l; r$
```

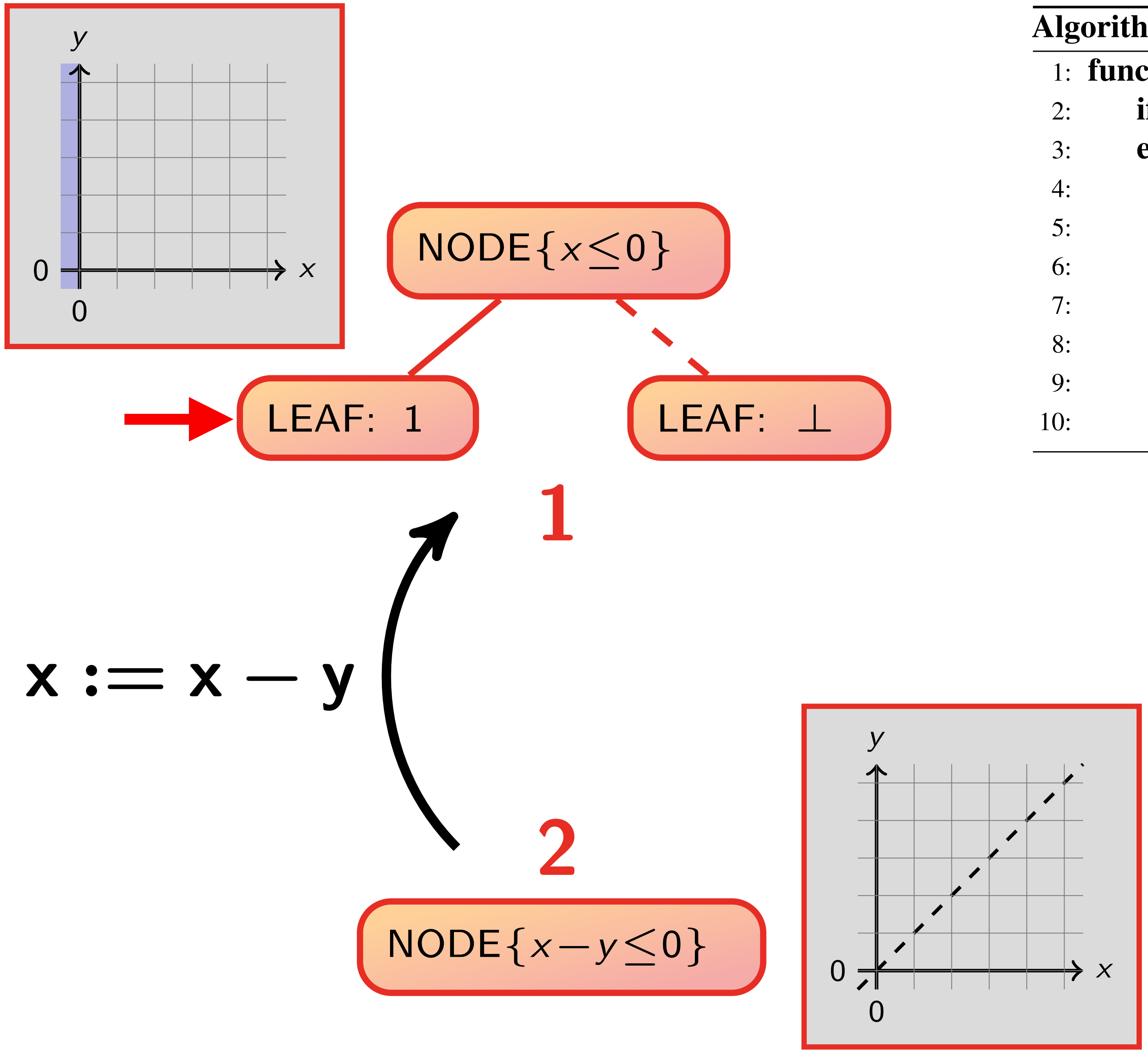
Assignments



Algorithm 3 : Tree Assign

```
function ASSIGN($t$, $x := a$)
    if ISLEAF($t$) then return LEAF : ASSIGN$_F$($f, x := a$)        /* $t \triangleq$ LEAF : $f$ */
    else
        $C \leftarrow$ ASSIGN$_L$($t.c, x := a$)
        if ISEMPTY($C$) then return ASSIGN($t.l, x := a$) $\sqcup_T$ ASSIGN($t.r, x := a$)
        else if ISUNSAT($C$) then return ASSIGN($t.r, x := a$)
        else
            $l \leftarrow$ AUGMENT(ASSIGN($t.l, x := a$), $C$)
            $r \leftarrow$ AUGMENT(ASSIGN($t.r, x := a$), $C$)
            return NODE$\{l.c\}$ : $l; r$
```

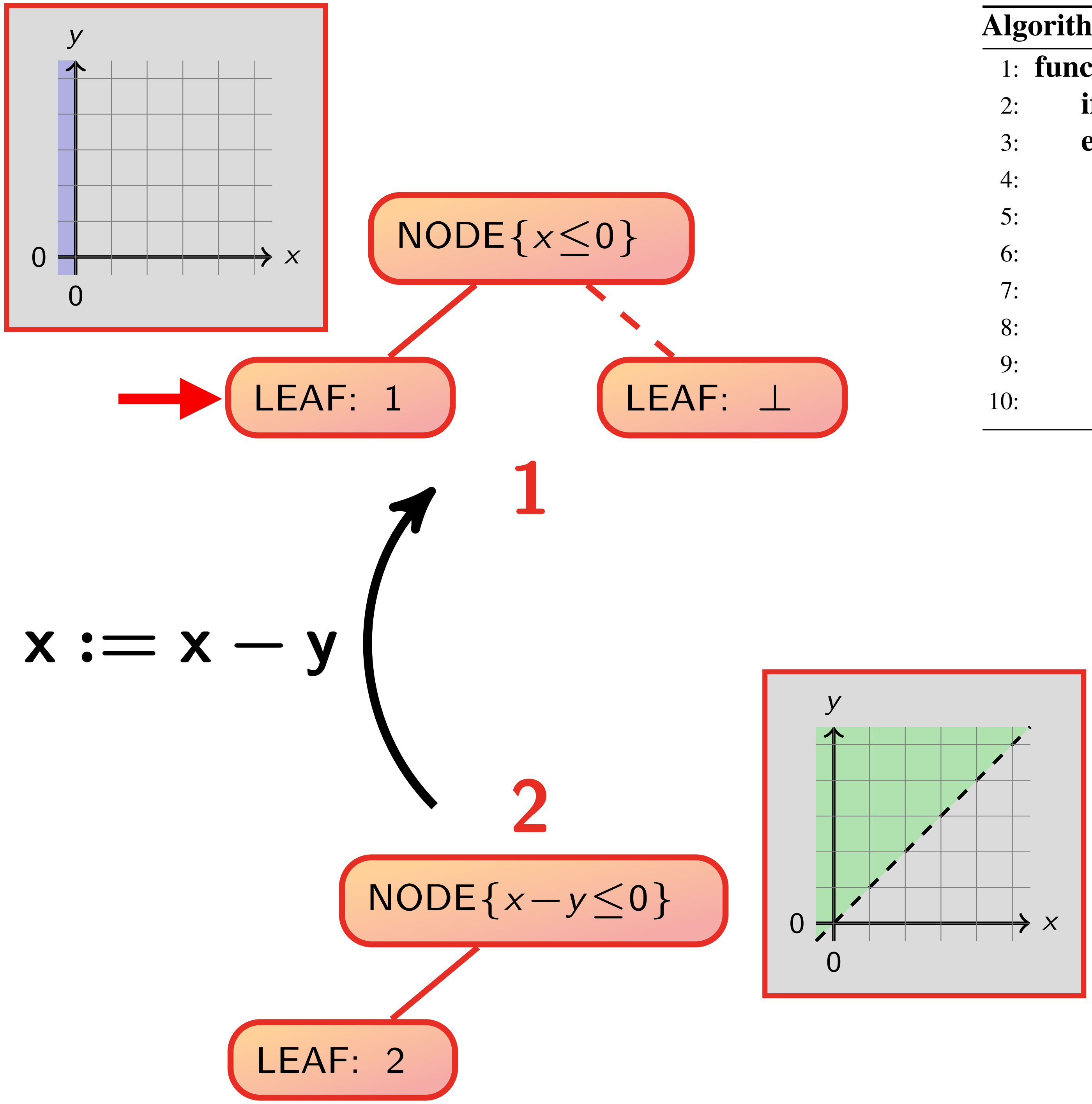
Assignments



Algorithm 3 : Tree Assign

```
function ASSIGN($t$, $x := a$)
    if ISLEAF($t$) then return LEAF : ASSIGN$_F$($f, x := a$)        /* $t \triangleq$ LEAF : $f$ */
    else
        $C \leftarrow$ ASSIGN$_L$($t.c, x := a$)
        if ISEMPTY($C$) then return ASSIGN($t.l, x := a$) $\sqcup_T$ ASSIGN($t.r, x := a$)
        else if ISUNSAT($C$) then return ASSIGN($t.r, x := a$)
        else
            $l \leftarrow$ AUGMENT(ASSIGN($t.l, x := a$), $C$)
            $r \leftarrow$ AUGMENT(ASSIGN($t.r, x := a$), $C$)
            return NODE{$l.c$} : $l$; $r$
```

Assignments



NODE$\{x \leq 0\}$

LEAF: 1

LEAF: $\perp$

1

$\mathbf{x := x - y}$

2

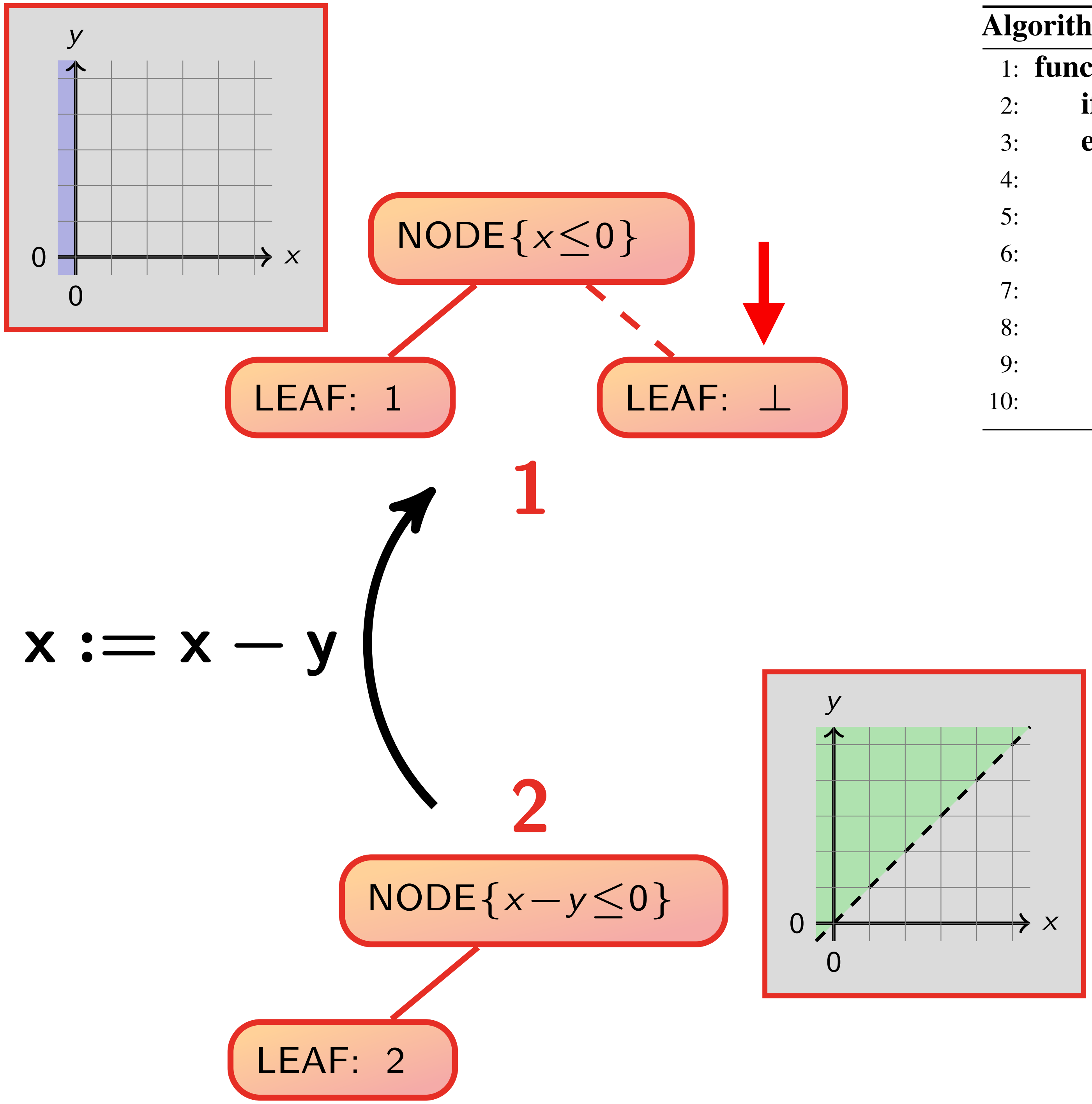
NODE$\{x - y \leq 0\}$

LEAF: 2

Algorithm 3 : Tree Assign

```
function ASSIGN($t$,$x := a$)
    if ISLEAF($t$) then return LEAF : ASSIGN_F($f, x := a$)        /* $t \triangleq$ LEAF : $f$ */
    else
        $C \leftarrow$ ASSIGN_L($t.c, x := a$)
        if ISEMPTY($C$) then return ASSIGN($t.l, x := a$) $\sqcup_T$ ASSIGN($t.r, x := a$)
        else if ISUNSAT($C$) then return ASSIGN($t.r, x := a$)
        else
            $l \leftarrow$ AUGMENT(ASSIGN($t.l, x := a$), $C$)
            $r \leftarrow$ AUGMENT(ASSIGN($t.r, x := a$), $C$)
            return NODE$\{l.c\}$ : $l$; $r$
```

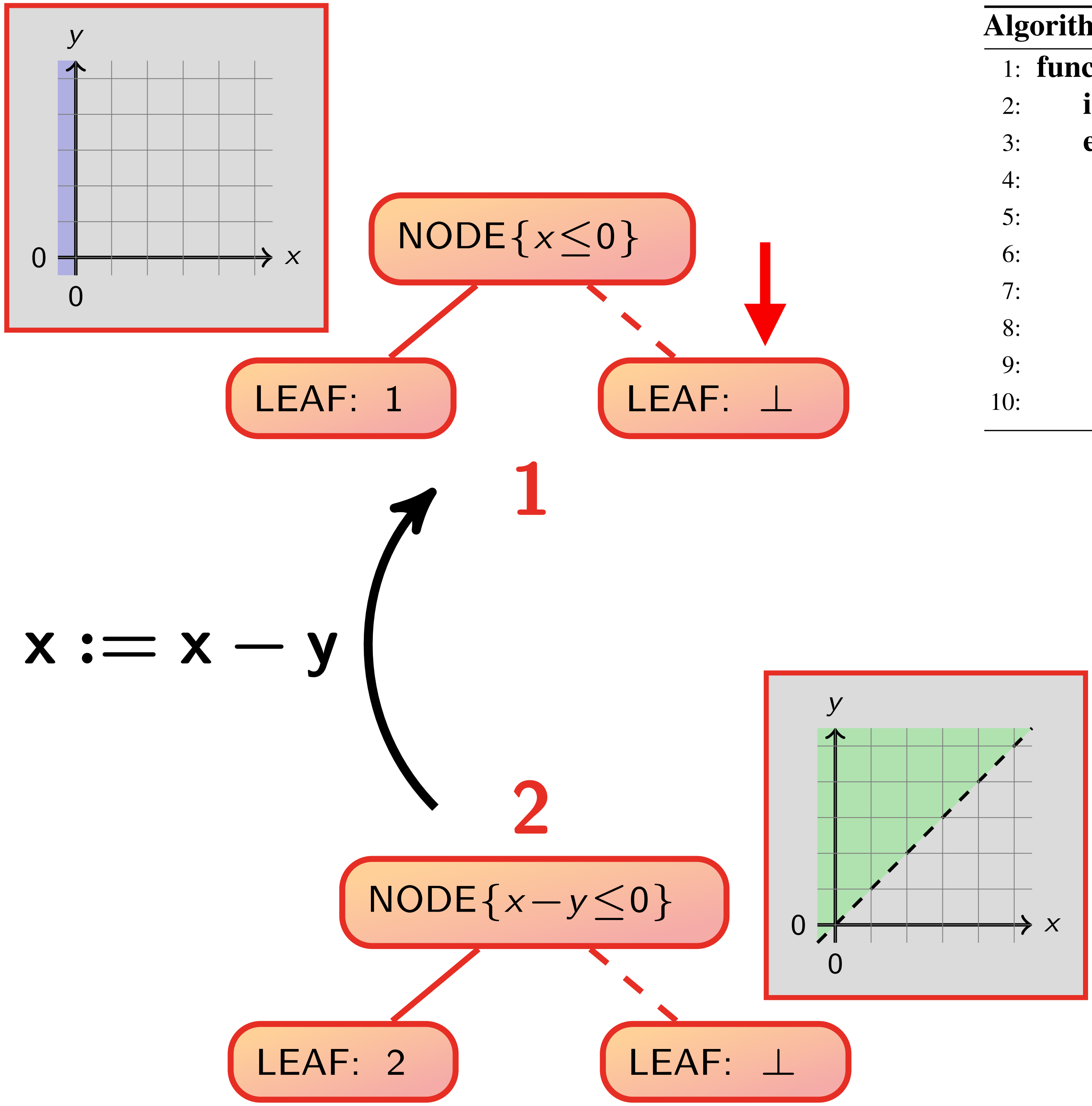
Assignments



Algorithm 3 : Tree Assign

```
function ASSIGN($t$,$x := a$)
    if ISLEAF($t$) then return LEAF : ASSIGN_F($f, x := a$)        /* $t \triangleq$ LEAF : $f$ */
    else
        $C \leftarrow$ ASSIGN_L($t.c, x := a$)
        if ISEMPTY($C$) then return ASSIGN($t.l, x := a$) $\sqcup_T$ ASSIGN($t.r, x := a$)
        else if ISUNSAT($C$) then return ASSIGN($t.r, x := a$)
        else
            $l \leftarrow$ AUGMENT(ASSIGN($t.l, x := a$), $C$)
            $r \leftarrow$ AUGMENT(ASSIGN($t.r, x := a$), $C$)
            return NODE{$l.c$} : $l; r$
```

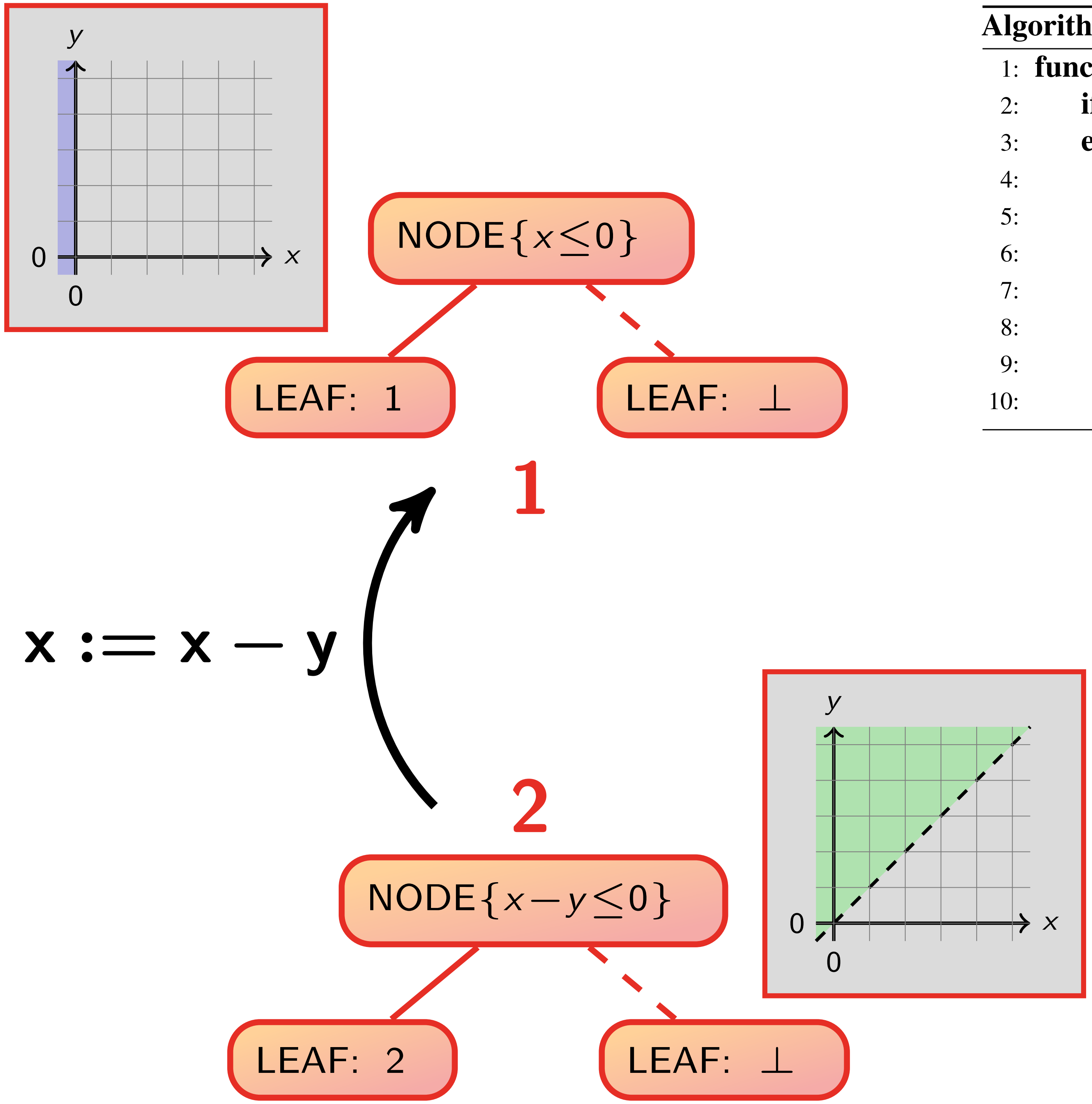
Assignments



Algorithm 3 : Tree Assign

```
function ASSIGN($t$,$x := a$)
    if ISLEAF($t$) then return LEAF : ASSIGN$_F$($f, x := a$)    /* $t \triangleq$ LEAF : $f$ */
    else
        $C \leftarrow$ ASSIGN$_L$($t.c, x := a$)
        if ISEMPTY($C$) then return ASSIGN($t.l, x := a$) $\sqcup_T$ ASSIGN($t.r, x := a$)
        else if ISUNSAT($C$) then return ASSIGN($t.r, x := a$)
        else
            $l \leftarrow$ AUGMENT(ASSIGN($t.l, x := a$), $C$)
            $r \leftarrow$ AUGMENT(ASSIGN($t.r, x := a$), $C$)
            return NODE$\{l.c\}$ : $l$; $r$
```

Assignments



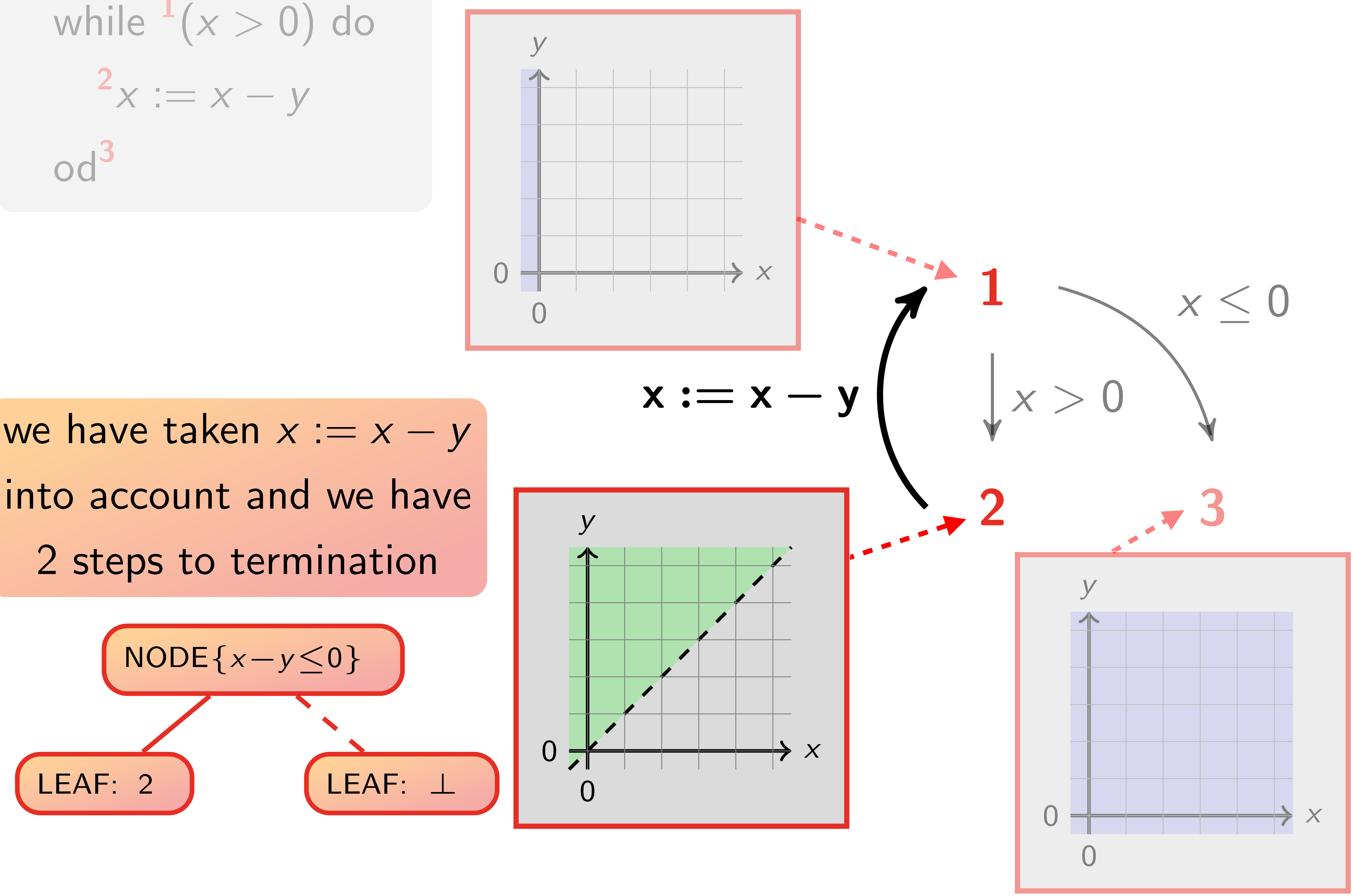
Algorithm 3 : Tree Assign

```
function ASSIGN($t$,$x := a$)
    if ISLEAF($t$) then return LEAF : ASSIGN_F($f, x := a$)        /* $t \triangleq$ LEAF : $f$ */
    else
        $C \leftarrow$ ASSIGN_L($t.c, x := a$)
        if ISEMPTY($C$) then return ASSIGN($t.l, x := a$) $\sqcup_T$ ASSIGN($t.r, x := a$)
        else if ISUNSAT($C$) then return ASSIGN($t.r, x := a$)
        else
            $l \leftarrow$ AUGMENT(ASSIGN($t.l, x := a$), $C$)
            $r \leftarrow$ AUGMENT(ASSIGN($t.r, x := a$), $C$)
            return NODE{$l.c$} : $l; r$
```

Example

```
int : $x$,  $y$
while [1]($x > 0$) do
    [2]$x := x - y$
od[3]
```

we have taken $x := x - y$ into account and we have 2 steps to termination



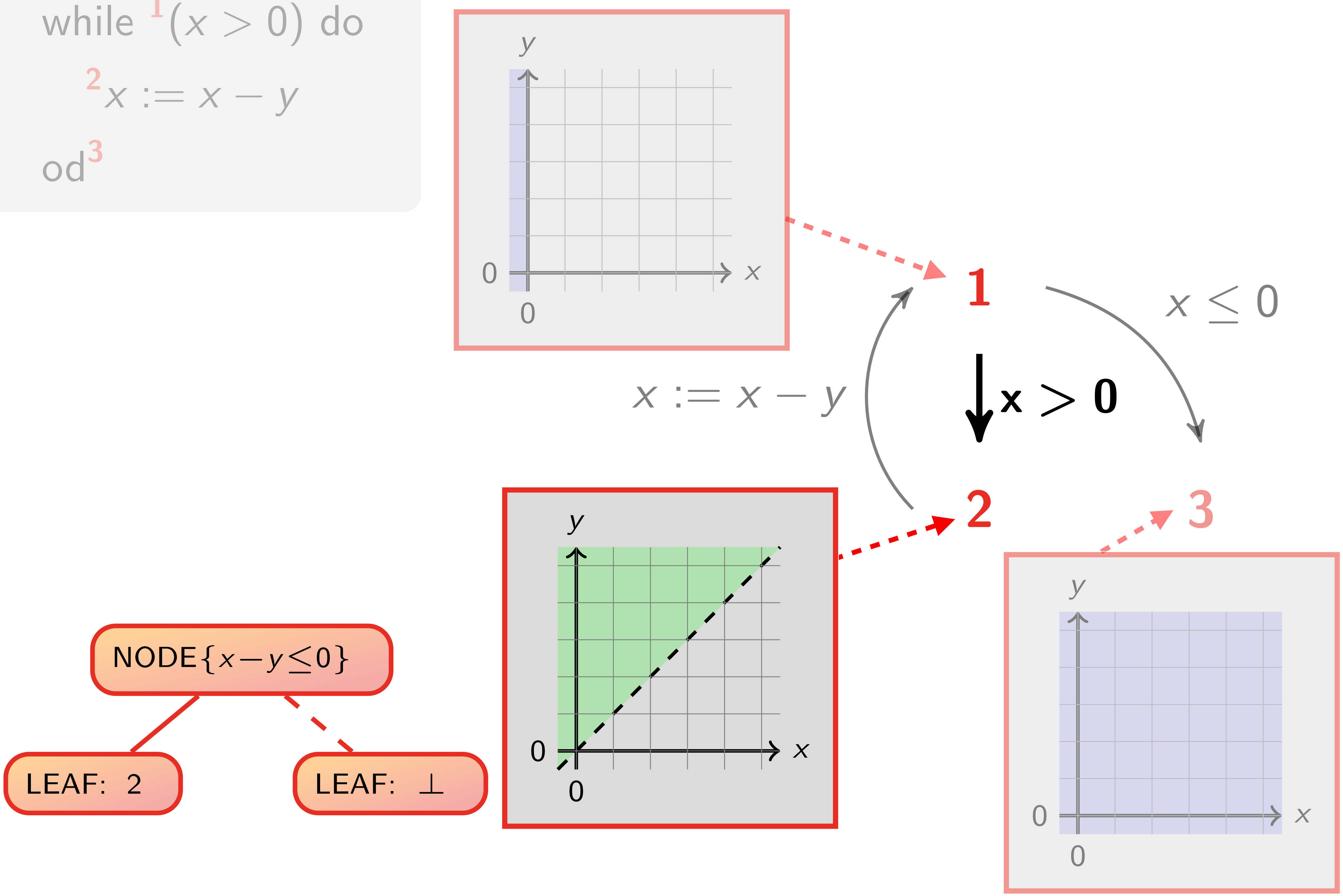
Example

int : x, y

while ${}^{1}(x > 0)$ do

${}^{2}x := x - y$

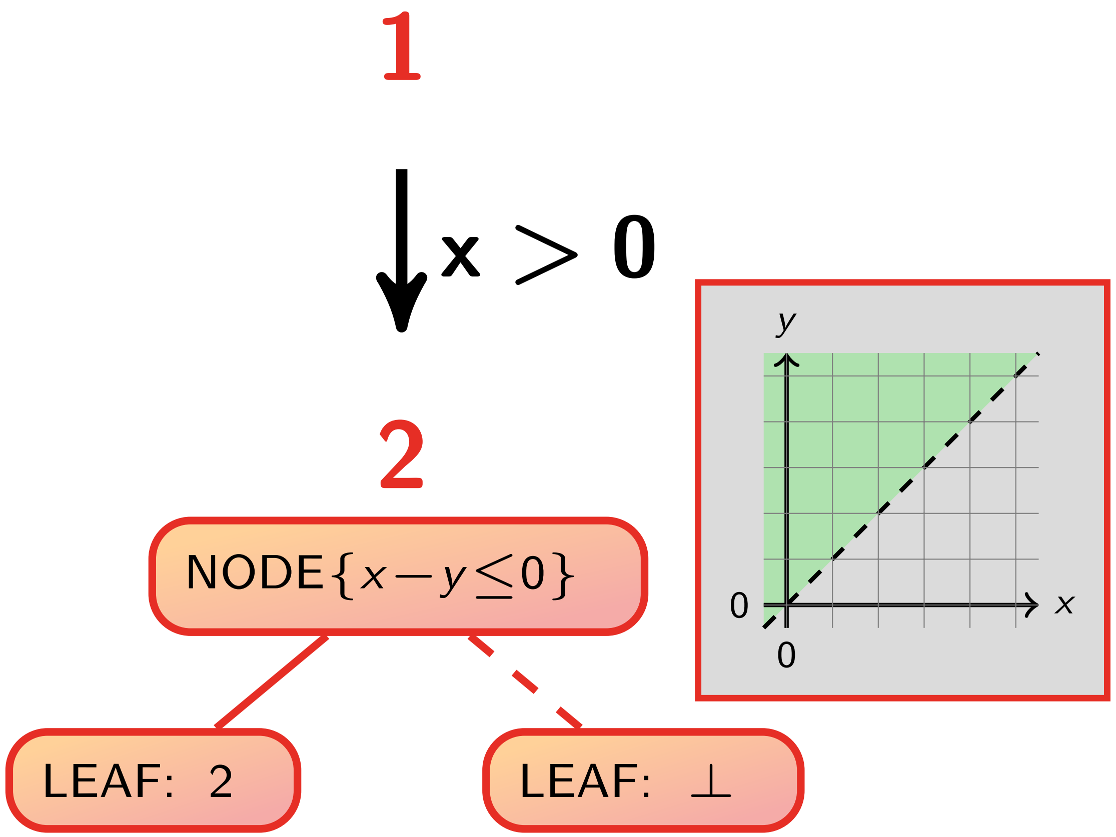
od^{3}



Tests

Algorithm 4 : Tree Filter

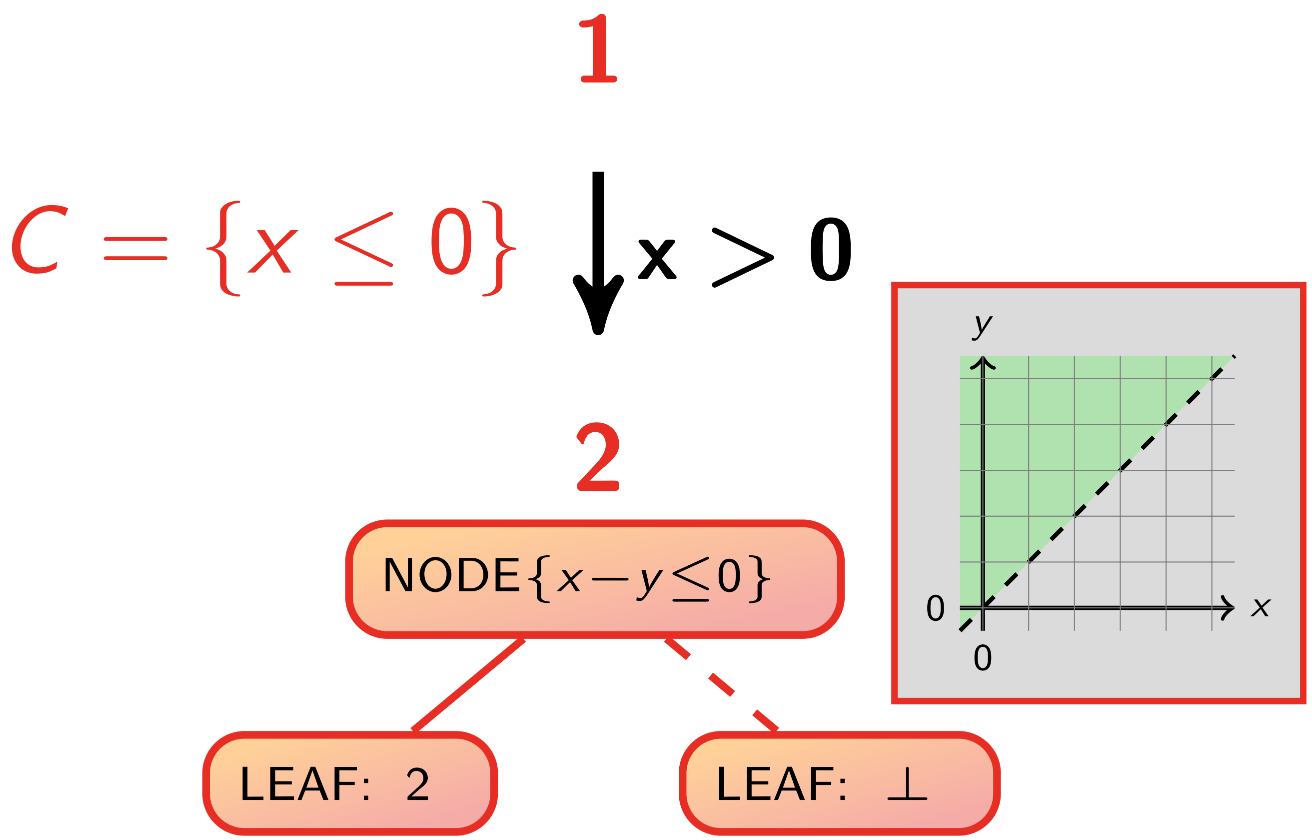
```
function FILTER-AUX($t$,$c$)
    if ISLEAF($t$) then return LEAF : FILTER$_F$($f$, $c$)        /* $t \triangleq$ LEAF : $f$ */
    else return NODE$\{t.c\}$ : FILTER-AUX($t.l$, $c$); FILTER-AUX($t.r$, $c$)

function FILTER($t$,$c$)
    $C \leftarrow$ FILTER$_L$($c$)
    return AUGMENT(FILTER-AUX($t$, $c$), $C$)
```



Tests

Algorithm 4 : Tree Filter

```
function FILTER-AUX(t,c)
    if ISLEAF($t$) then return LEAF : FILTER_F($f, c$)          /* $t \triangleq$ LEAF : $f$ */
    else return NODE{$t.c$} : FILTER-AUX($t.l, c$); FILTER-AUX($t.r, c$)

function FILTER(t,c)
    $C \leftarrow$ FILTER_L($c$)
    return AUGMENT(FILTER-AUX($t, c$), $C$)
```



1

$C = \{x \leq 0\}$

$\mathbf{x > 0}$

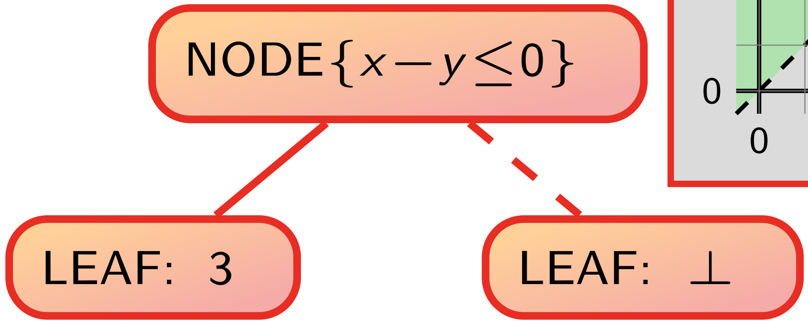
2

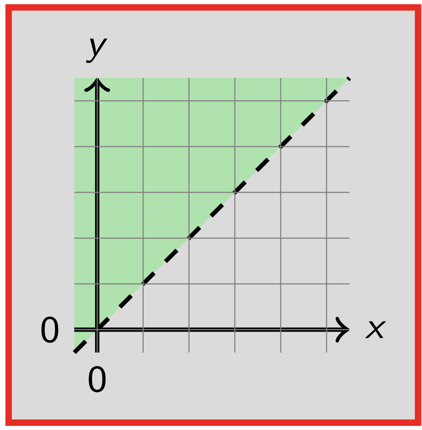
NODE$\{x-y\leq 0\}$

LEAF: 2

LEAF: $\perp$

Tests



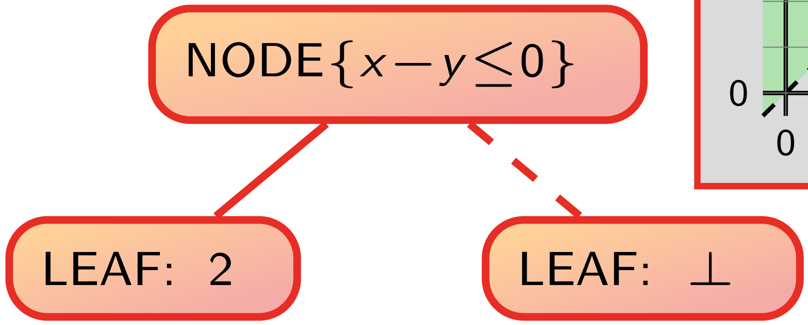


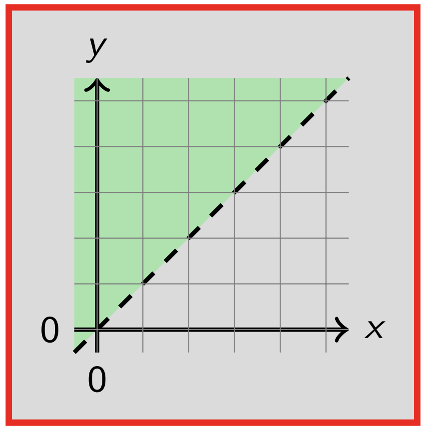
```
thm 4 : Tree Filter
nction FILTER-AUX($t$,$c$)
    if ISLEAF($t$) then return LEAF : FILTER$_F$($f, c$)        /* $t \triangleq$ LEAF : $f$ */
    else return NODE$\{t.c\}$ : FILTER-AUX($t.l, c$); FILTER-AUX($t.r, c$)

nction FILTER($t$,$c$)
    $C \leftarrow$ FILTER$_L$($c$)
    return AUGMENT(FILTER-AUX($t, c$), $C$)
```

1

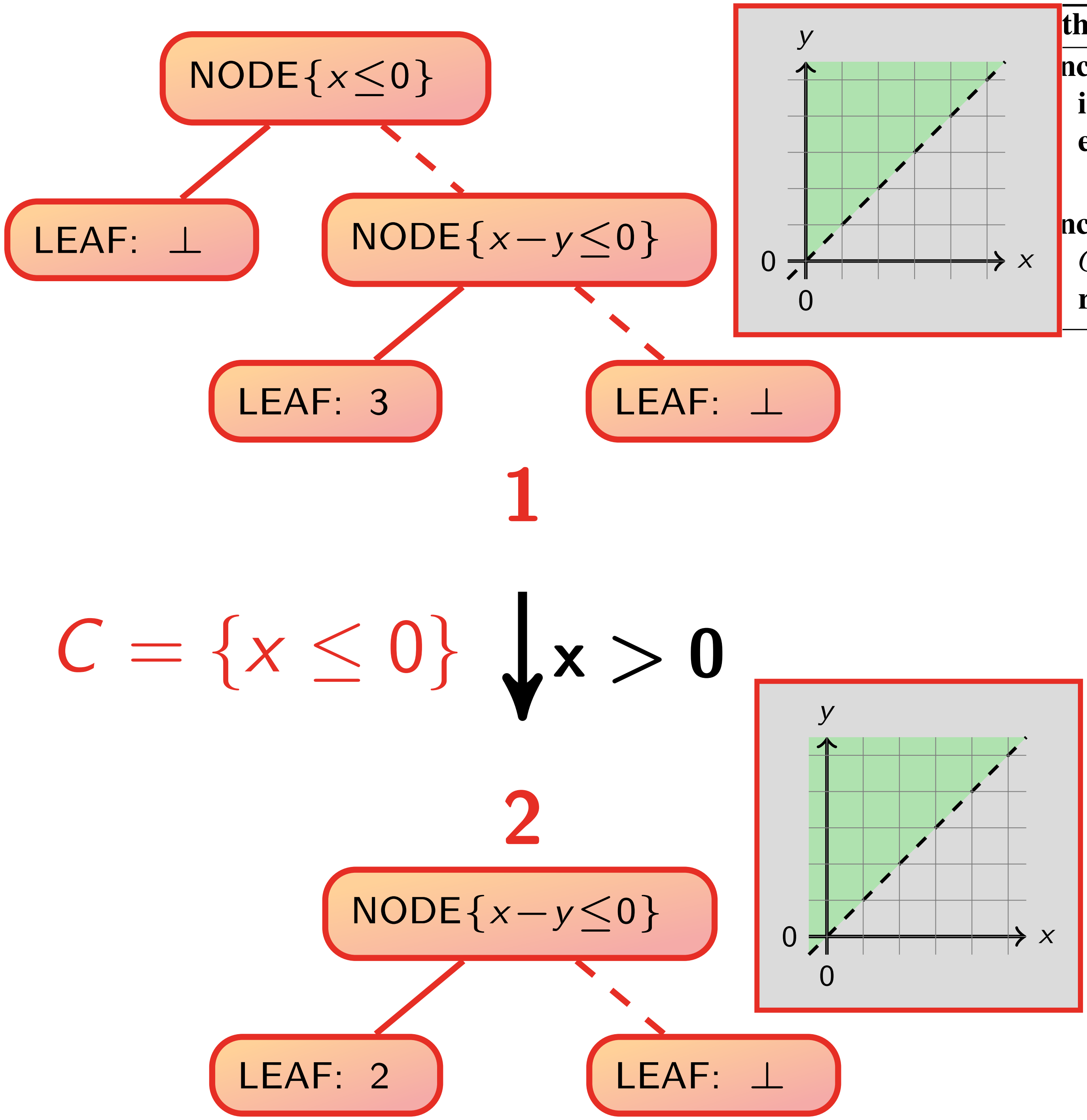
$C = \{x \leq 0\}$ $\downarrow$ **$x > 0$**

2





Tests



1

$C = \{x \leq 0\}$ $\big\downarrow$ $\mathbf{x > 0}$

2

```
thm 4 : Tree Filter
nction FILTER-AUX($t$,$c$)
    if ISLEAF($t$) then return LEAF : FILTER$_F$($f, c$)        /* $t \triangleq$ LEAF : $f$ */
    else return NODE{$t.c$} : FILTER-AUX($t.l, c$); FILTER-AUX($t.r, c$)

nction FILTER($t$,$c$)
    $C \leftarrow$ FILTER$_L$($c$)
    return AUGMENT(FILTER-AUX($t, c$), $C$)
```

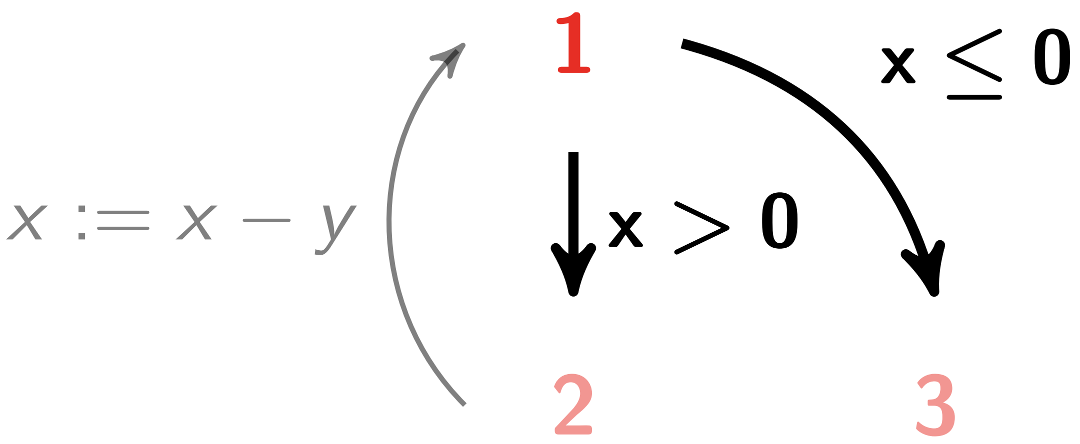
Example

int : x, y

while $^{1}(x > 0)$ do

$^{2}x := x - y$

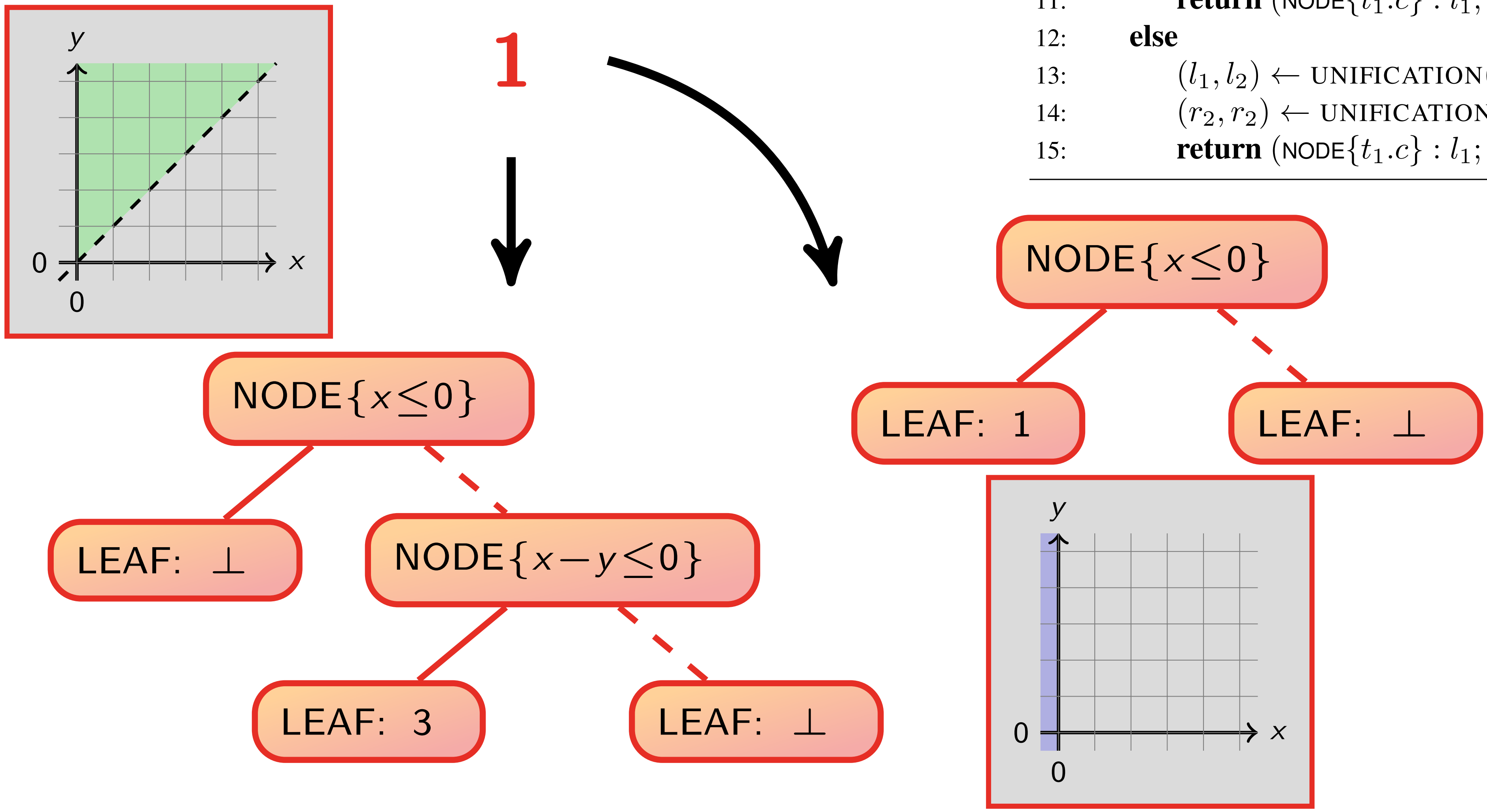
od^{3}



Join

Algorithm 1 : Tree Unification

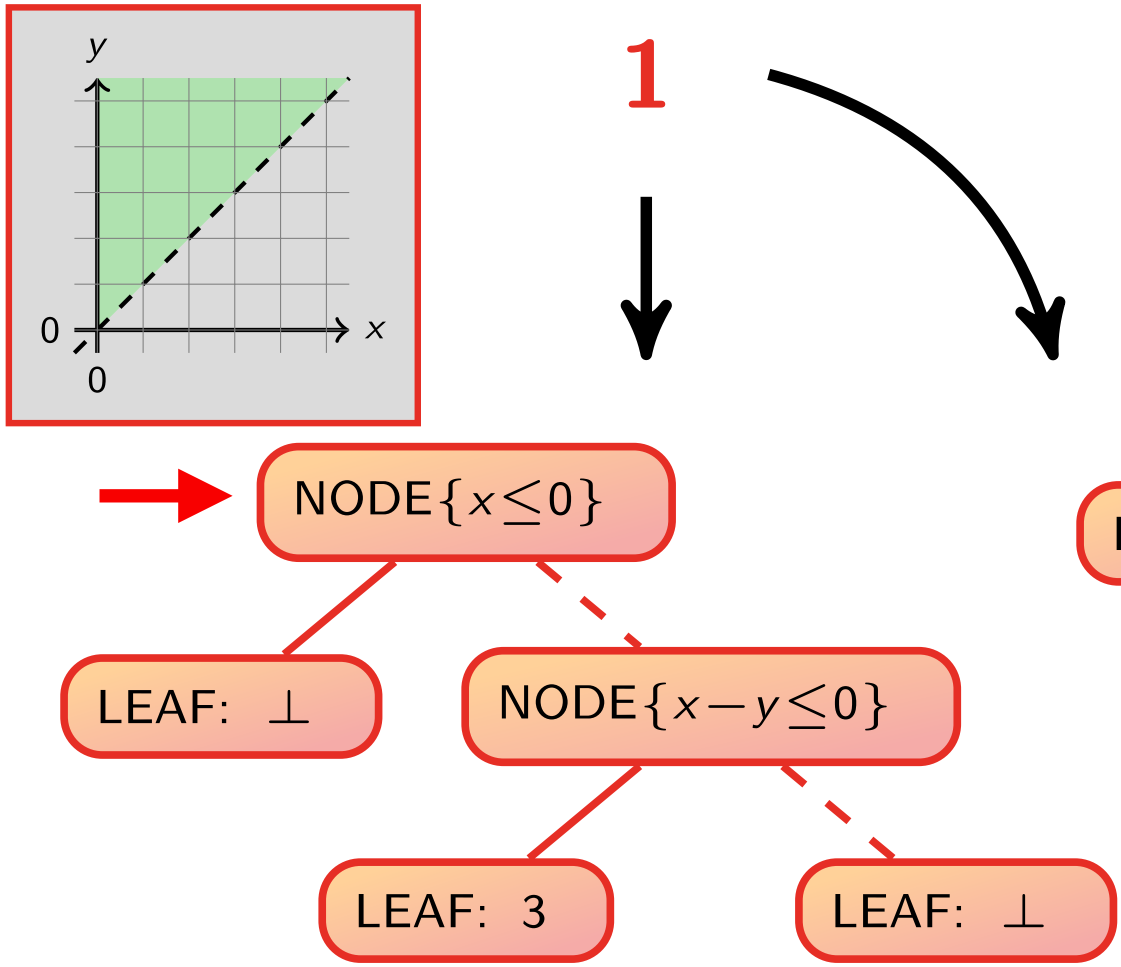
```
function UNIFICATION($t_1$,$t_2$)
    if ISLEAF($t_1$) $\wedge$ ISLEAF($t_2$) then
        return ($t_1$, $t_2$)
    else if ISLEAF($t_1$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_2.c <_{\mathsf{L}} t_1.c$) then
        $(l_1, l_2) \leftarrow$ UNIFICATION($t_1, t_2.l$)
        $(r_1, r_2) \leftarrow$ UNIFICATION($t_1, t_2.r$)
        return (NODE$\{t_2.c\} : l_1; r_1$, NODE$\{t_2.c\} : l_2; r_2$)
    else if ISLEAF($t_2$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_1.c <_{\mathsf{L}} t_2.c$) then
        $(l_1, l_2) \leftarrow$ UNIFICATION($t_1.l, t_2$)
        $(r_1, r_2) \leftarrow$ UNIFICATION($t_1.r, t_2$)
        return (NODE$\{t_1.c\} : l_1; r_1$, NODE$\{t_1.c\} : l_2; r_2$)
    else
        $(l_1, l_2) \leftarrow$ UNIFICATION($t_1.l, t_2.l$)
        $(r_2, r_2) \leftarrow$ UNIFICATION($t_1.r, t_2.r$)
        return (NODE$\{t_1.c\} : l_1; r_1$, NODE$\{t_2.c\} : l_2; r_2$)
```

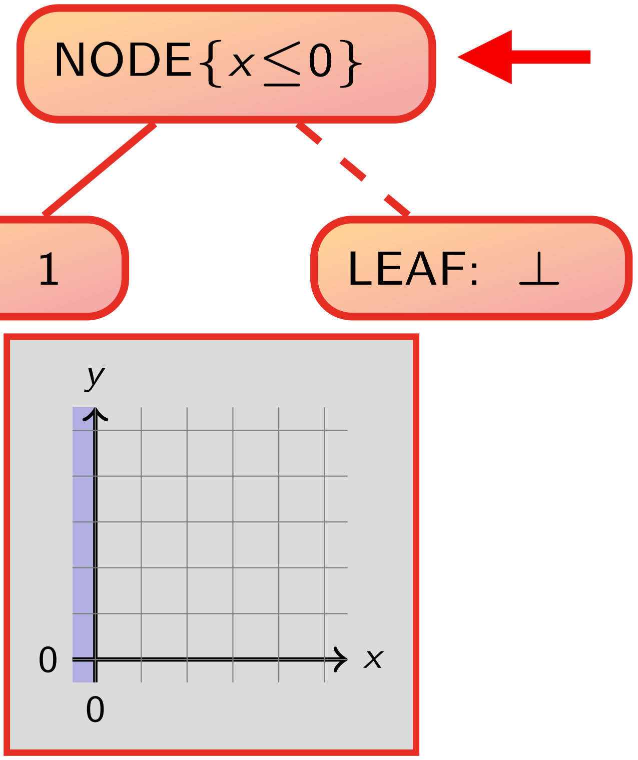


Join

Algorithm 1 : Tree Unification

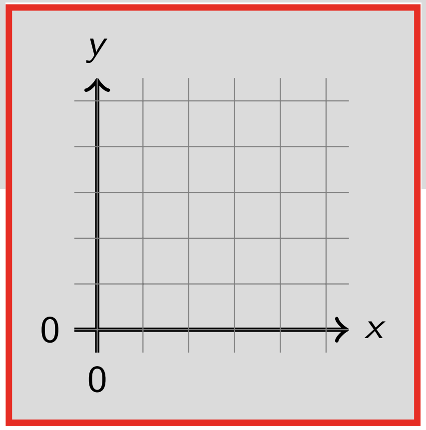
```
function UNIFICATION($t_1$,$t_2$)
    if ISLEAF($t_1$) $\wedge$ ISLEAF($t_2$) then
        return ($t_1$, $t_2$)
    else if ISLEAF($t_1$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_2.c <_{\mathsf{L}} t_1.c$) then
        $(l_1, l_2) \leftarrow$ UNIFICATION($t_1, t_2.l$)
        $(r_1, r_2) \leftarrow$ UNIFICATION($t_1, t_2.r$)
        return (NODE$\{t_2.c\} : l_1; r_1$, NODE$\{t_2.c\} : l_2; r_2$)
    else if ISLEAF($t_2$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_1.c <_{\mathsf{L}} t_2.c$) then
        $(l_1, l_2) \leftarrow$ UNIFICATION($t_1.l, t_2$)
        $(r_1, r_2) \leftarrow$ UNIFICATION($t_1.r, t_2$)
        return (NODE$\{t_1.c\} : l_1; r_1$, NODE$\{t_1.c\} : l_2; r_2$)
    else
        $(l_1, l_2) \leftarrow$ UNIFICATION($t_1.l, t_2.l$)
        $(r_2, r_2) \leftarrow$ UNIFICATION($t_1.r, t_2.r$)
        return (NODE$\{t_1.c\} : l_1; r_1$, NODE$\{t_2.c\} : l_2; r_2$)
```





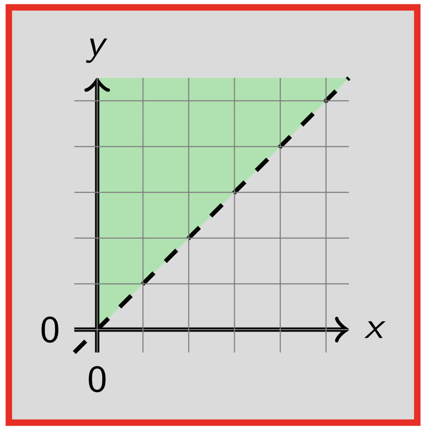
Join

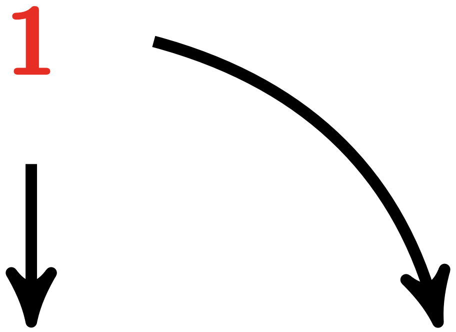
NODE{$x \leq 0$}

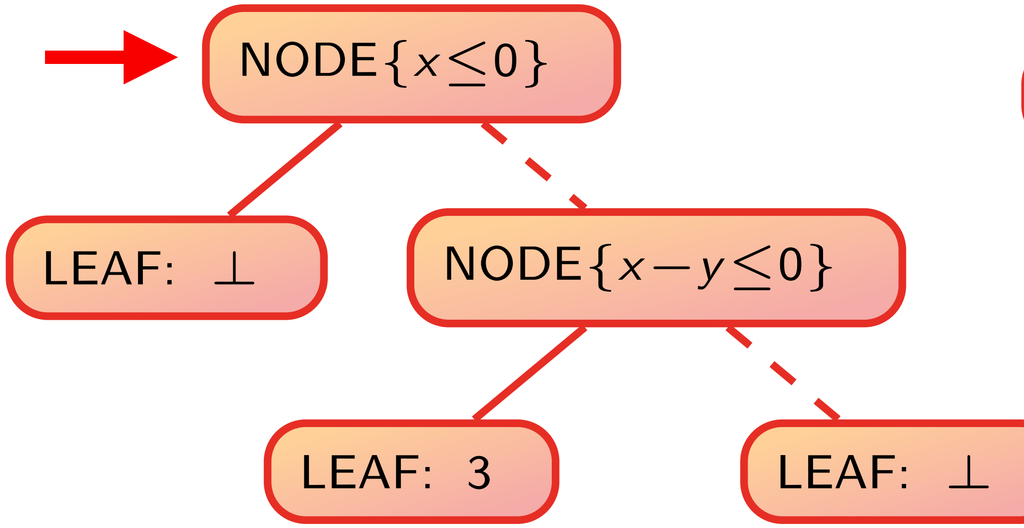


Algorithm 1 : Tree Unification

```
function UNIFICATION($t_1$,$t_2$)
    if ISLEAF($t_1$) $\wedge$ ISLEAF($t_2$) then
        return ($t_1$, $t_2$)
    else if ISLEAF($t_1$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_2.c <_L t_1.c$) then
        $(l_1, l_2) \leftarrow$ UNIFICATION($t_1, t_2.l$)
        $(r_1, r_2) \leftarrow$ UNIFICATION($t_1, t_2.r$)
        return (NODE$\{t_2.c\} : l_1; r_1$, NODE$\{t_2.c\} : l_2; r_2$)
    else if ISLEAF($t_2$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_1.c <_L t_2.c$) then
        $(l_1, l_2) \leftarrow$ UNIFICATION($t_1.l, t_2$)
        $(r_1, r_2) \leftarrow$ UNIFICATION($t_1.r, t_2$)
        return (NODE$\{t_1.c\} : l_1; r_1$, NODE$\{t_1.c\} : l_2; r_2$)
    else
        $(l_1, l_2) \leftarrow$ UNIFICATION($t_1.l, t_2.l$)
        $(r_2, r_2) \leftarrow$ UNIFICATION($t_1.r, t_2.r$)
        return (NODE$\{t_1.c\} : l_1; r_1$, NODE$\{t_2.c\} : l_2; r_2$)
```







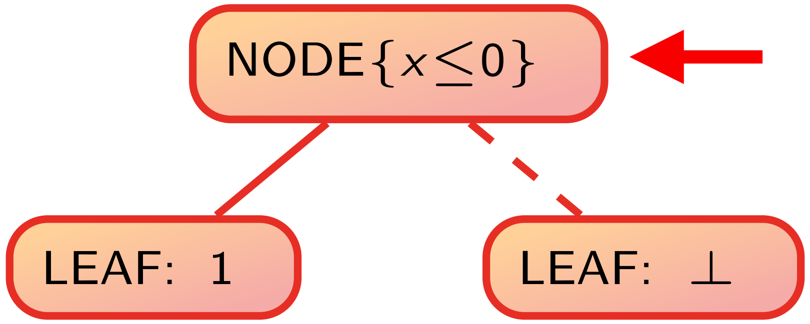
NODE{$x \leq 0$}

LEAF: $\bot$

NODE{$x - y \leq 0$}

LEAF: 3

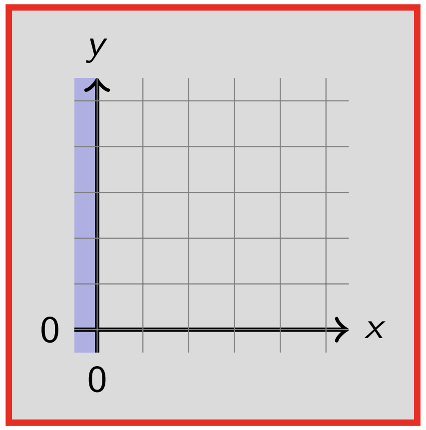
LEAF: $\bot$



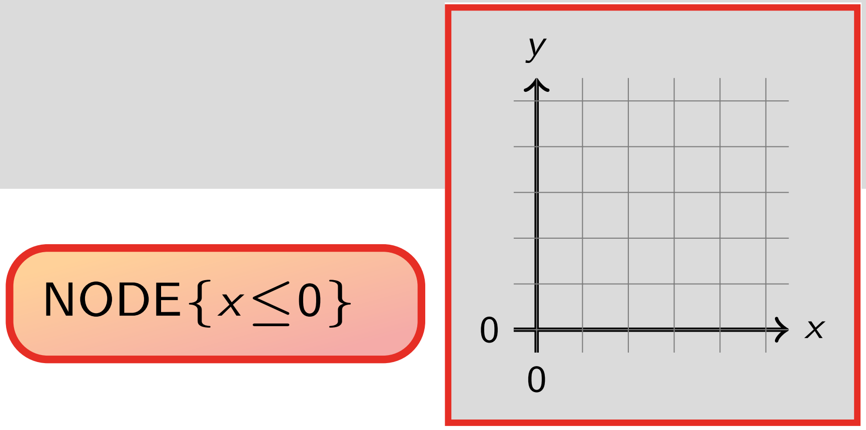
NODE{$x \leq 0$}

LEAF: 1

LEAF: $\bot$

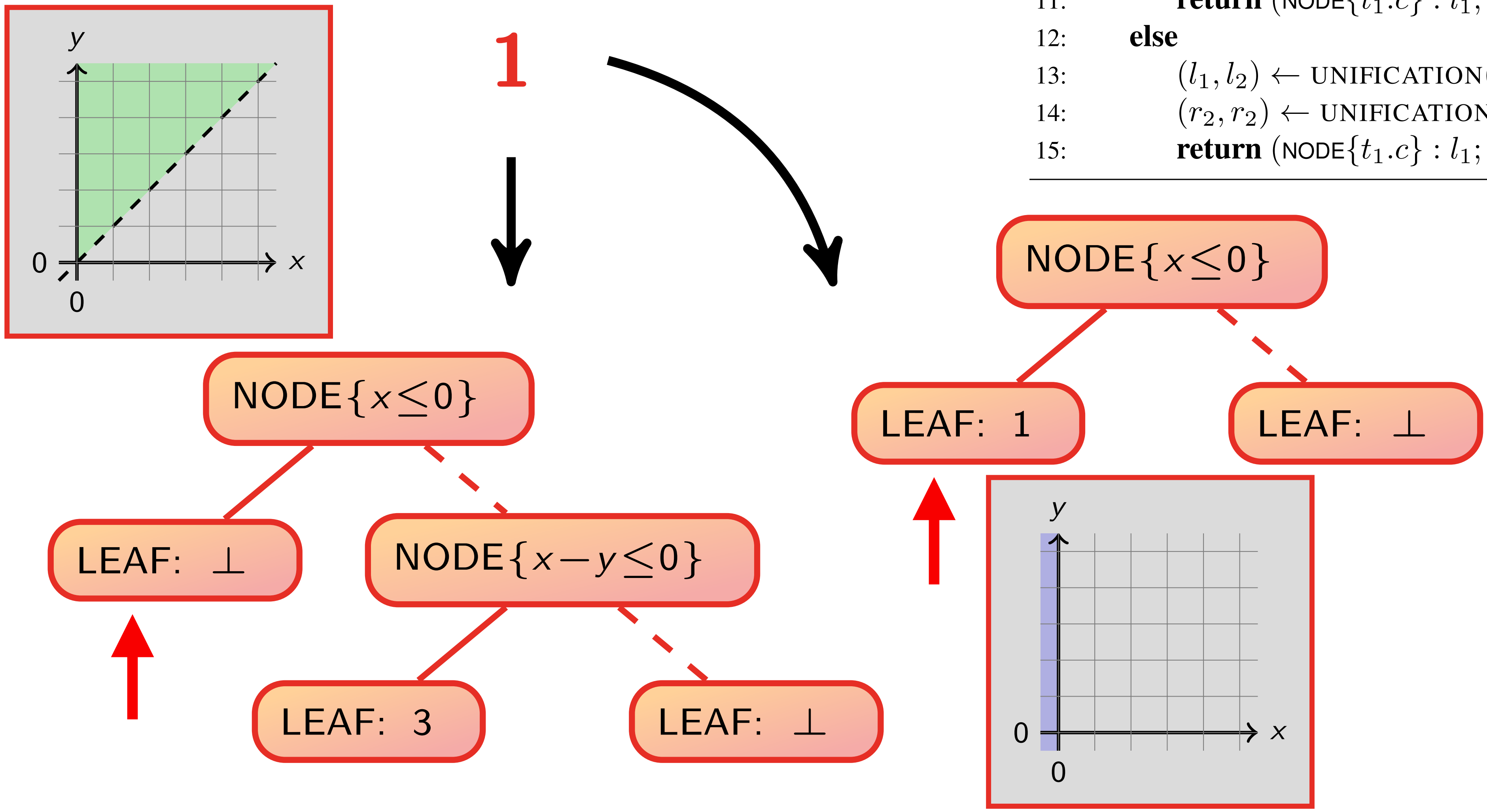


Join

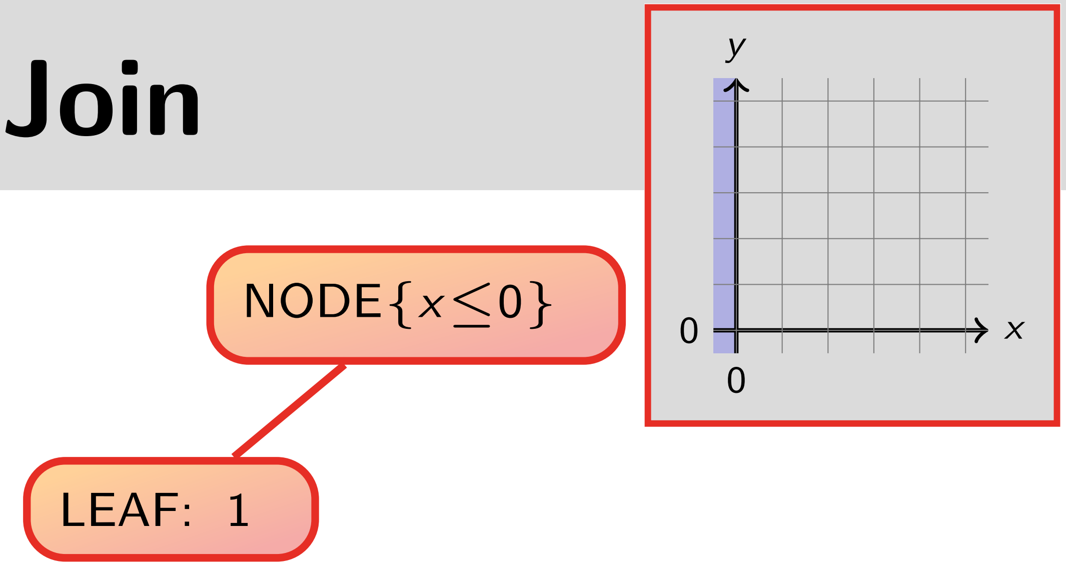


Algorithm 1 : Tree Unification

```
function UNIFICATION($t_1$,$t_2$)
    if ISLEAF($t_1$) $\wedge$ ISLEAF($t_2$) then
        return ($t_1$, $t_2$)
    else if ISLEAF($t_1$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_2.c <_L t_1.c$) then
        $(l_1, l_2) \leftarrow$ UNIFICATION($t_1, t_2.l$)
        $(r_1, r_2) \leftarrow$ UNIFICATION($t_1, t_2.r$)
        return (NODE$\{t_2.c\}$ : $l_1; r_1$, NODE$\{t_2.c\}$ : $l_2; r_2$)
    else if ISLEAF($t_2$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_1.c <_L t_2.c$) then
        $(l_1, l_2) \leftarrow$ UNIFICATION($t_1.l, t_2$)
        $(r_1, r_2) \leftarrow$ UNIFICATION($t_1.r, t_2$)
        return (NODE$\{t_1.c\}$ : $l_1; r_1$, NODE$\{t_1.c\}$ : $l_2; r_2$)
    else
        $(l_1, l_2) \leftarrow$ UNIFICATION($t_1.l, t_2.l$)
        $(r_2, r_2) \leftarrow$ UNIFICATION($t_1.r, t_2.r$)
        return (NODE$\{t_1.c\}$ : $l_1; r_1$, NODE$\{t_2.c\}$ : $l_2; r_2$)
```

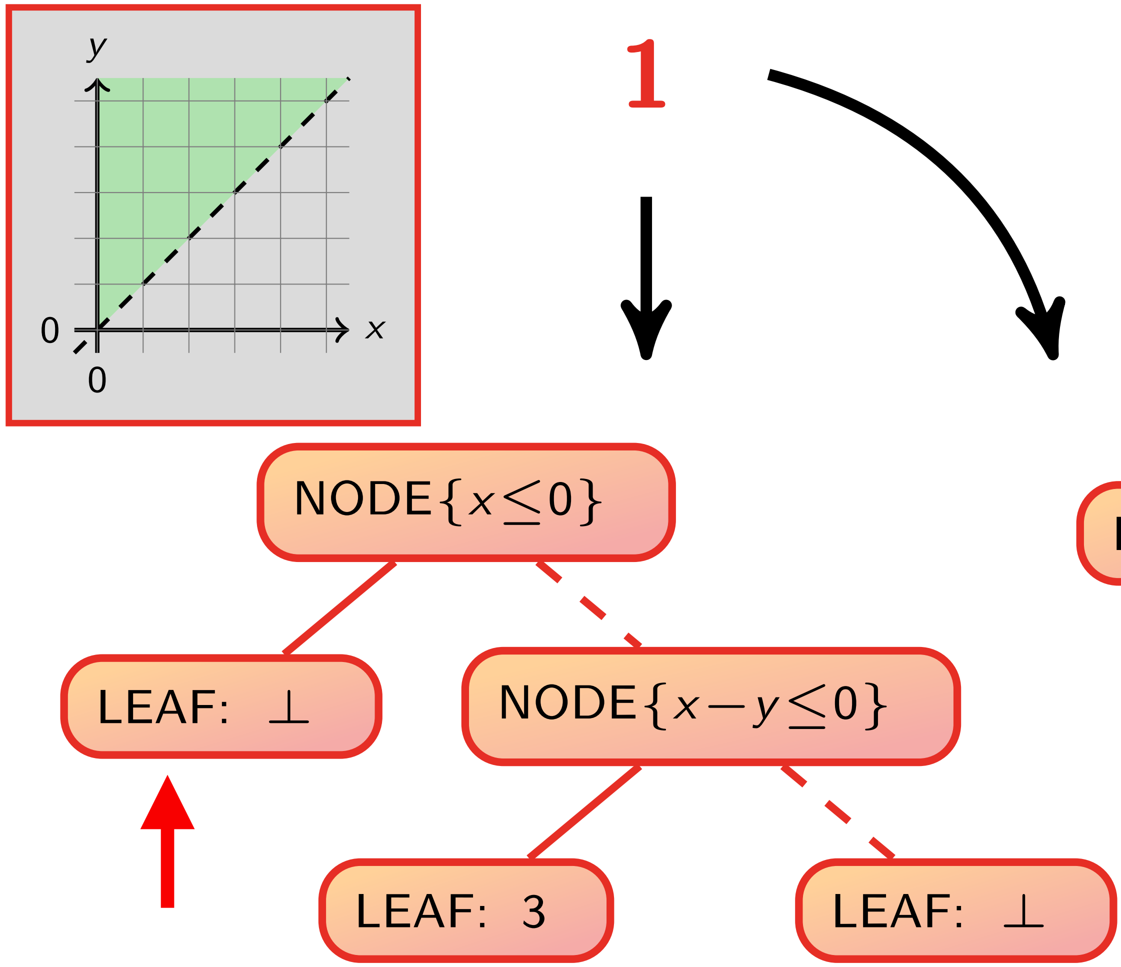


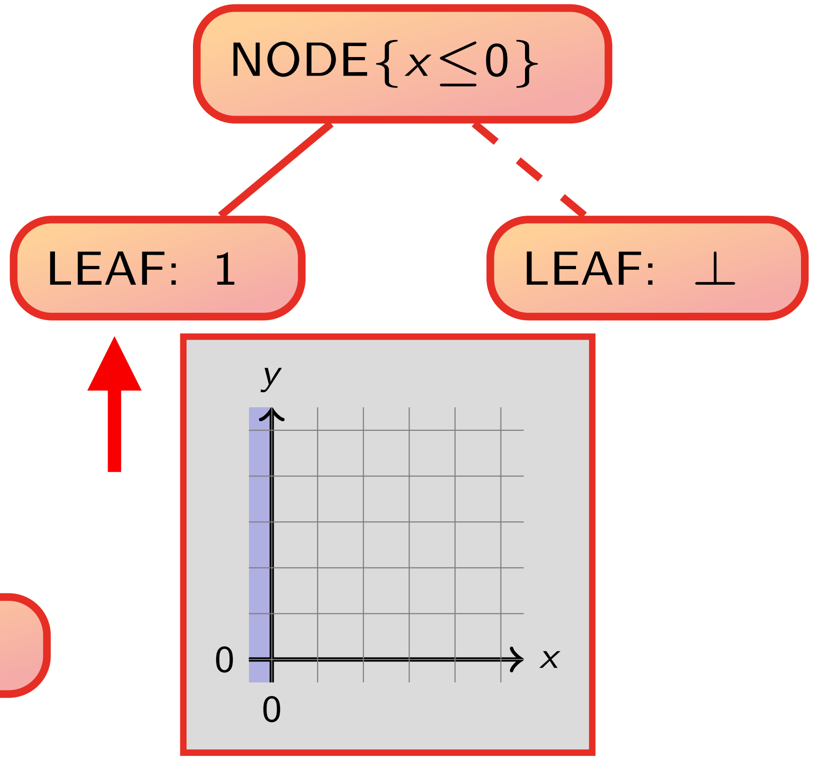
Join


NODE{$x \leq 0$}
LEAF: 1

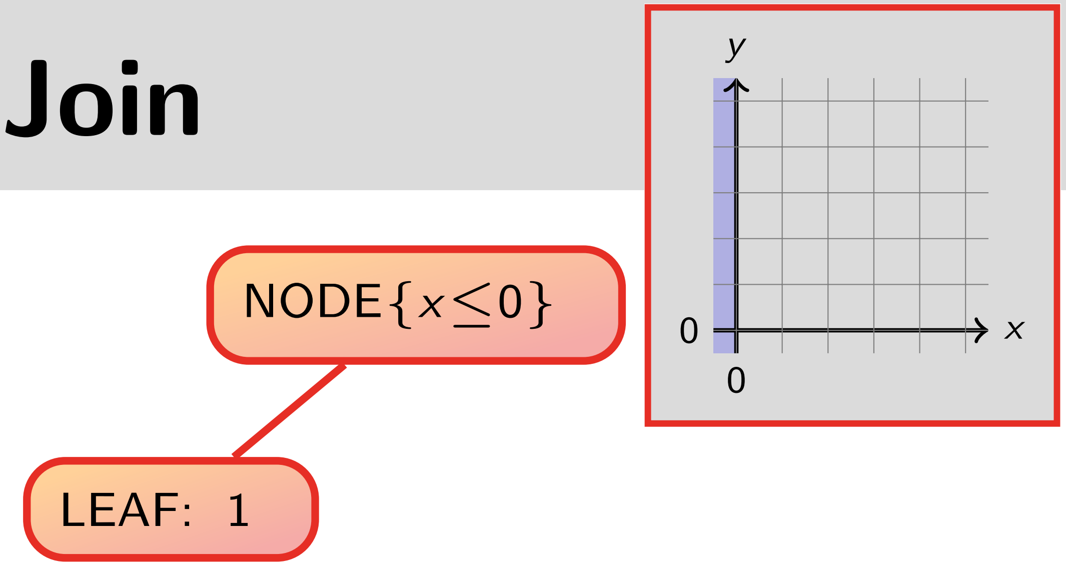
Algorithm 1 : Tree Unification

```
function UNIFICATION($t_1$,$t_2$)
    if ISLEAF($t_1$) $\wedge$ ISLEAF($t_2$) then
        return ($t_1$, $t_2$)
    else if ISLEAF($t_1$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_2.c <_{\mathsf{L}} t_1.c$) then
        $(l_1, l_2) \leftarrow$ UNIFICATION($t_1, t_2.l$)
        $(r_1, r_2) \leftarrow$ UNIFICATION($t_1, t_2.r$)
        return (NODE$\{t_2.c\} : l_1; r_1$, NODE$\{t_2.c\} : l_2; r_2$)
    else if ISLEAF($t_2$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_1.c <_{\mathsf{L}} t_2.c$) then
        $(l_1, l_2) \leftarrow$ UNIFICATION($t_1.l, t_2$)
        $(r_1, r_2) \leftarrow$ UNIFICATION($t_1.r, t_2$)
        return (NODE$\{t_1.c\} : l_1; r_1$, NODE$\{t_1.c\} : l_2; r_2$)
    else
        $(l_1, l_2) \leftarrow$ UNIFICATION($t_1.l, t_2.l$)
        $(r_2, r_2) \leftarrow$ UNIFICATION($t_1.r, t_2.r$)
        return (NODE$\{t_1.c\} : l_1; r_1$, NODE$\{t_2.c\} : l_2; r_2$)
```


1
NODE{$x \leq 0$}
LEAF: ⊥
NODE{$x-y \leq 0$}
LEAF: 3
LEAF: ⊥

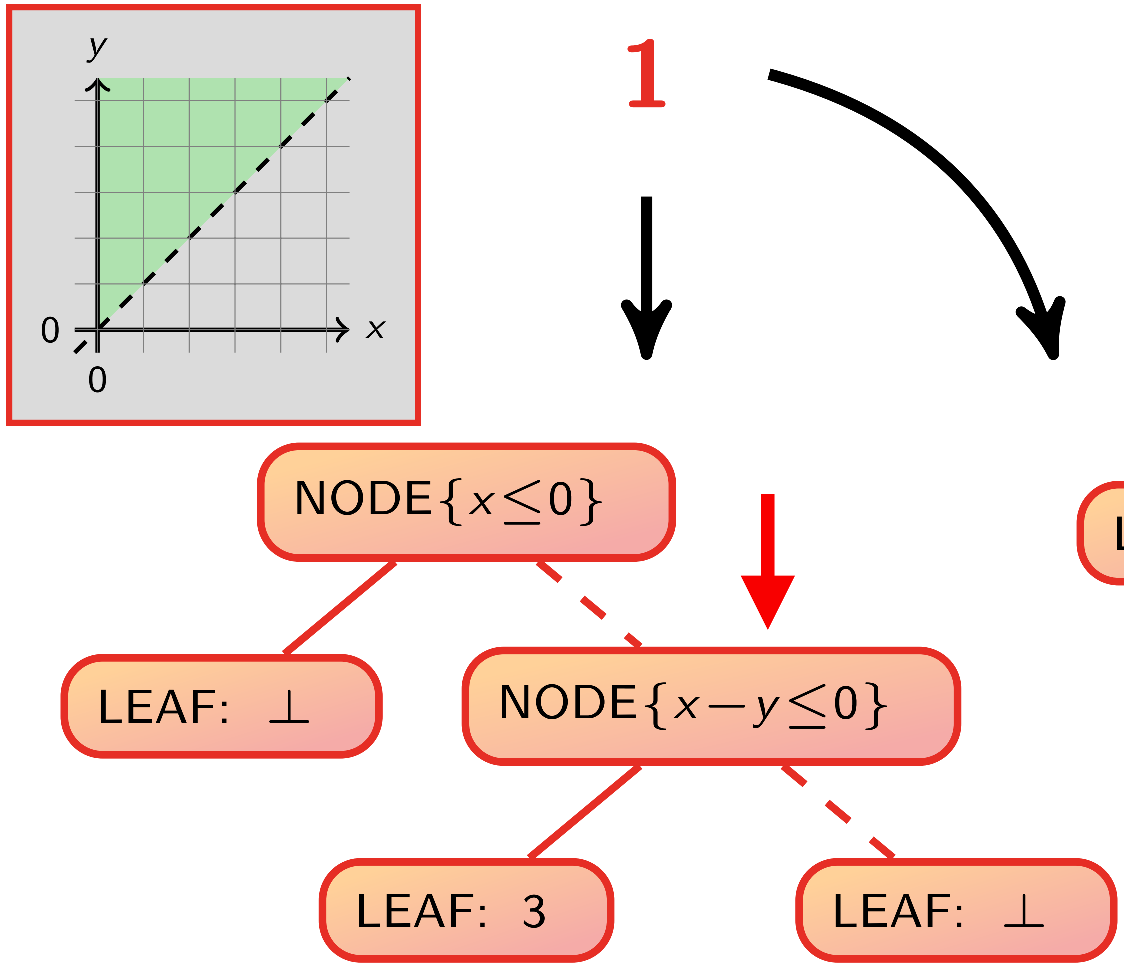

NODE{$x \leq 0$}
LEAF: 1
LEAF: ⊥

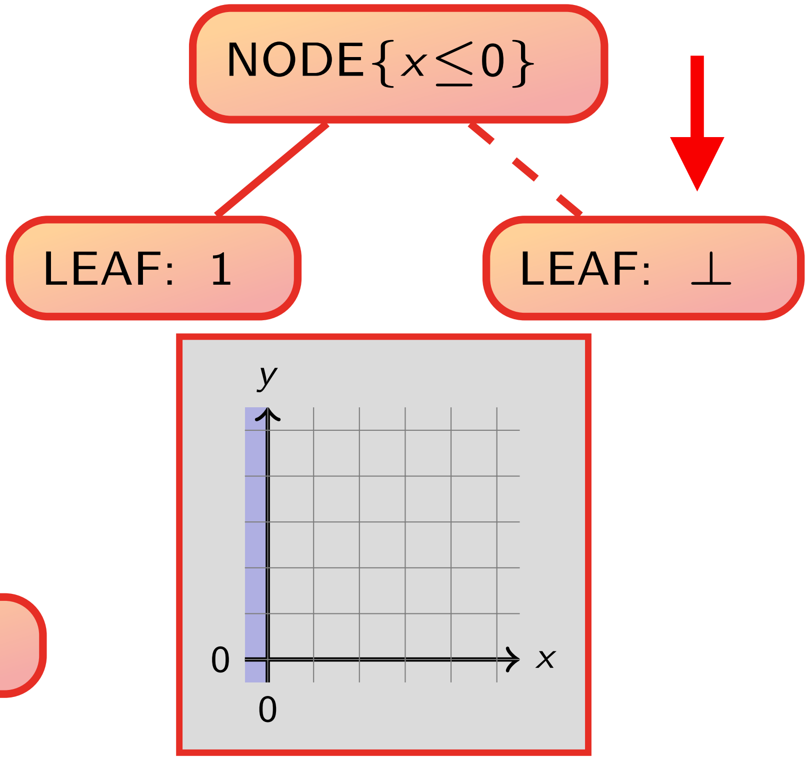
Join



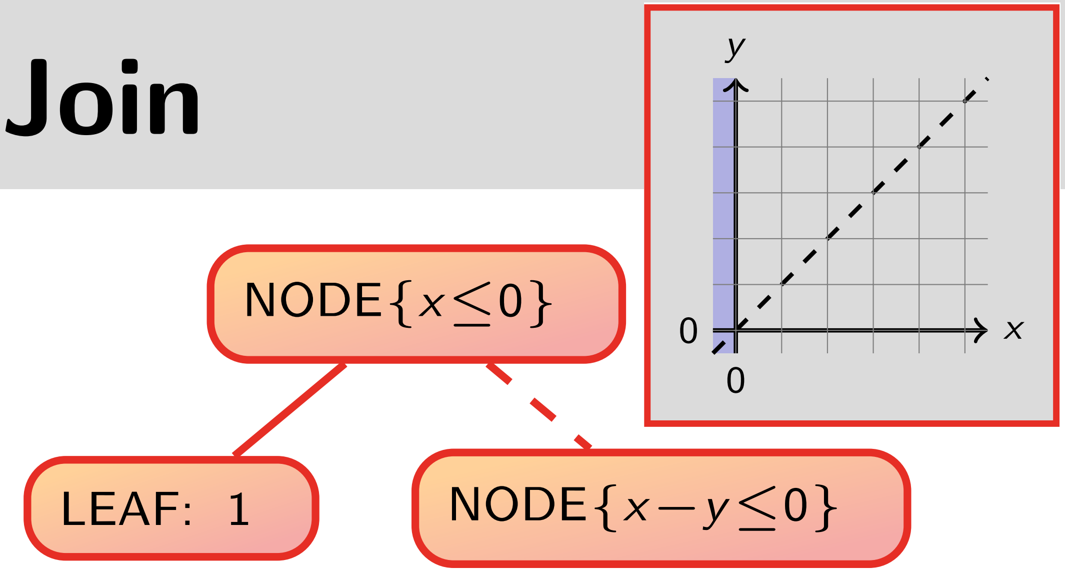
Algorithm 1 : Tree Unification

```
function UNIFICATION($t_1$,$t_2$)
    if ISLEAF($t_1$) $\wedge$ ISLEAF($t_2$) then
        return ($t_1$, $t_2$)
    else if ISLEAF($t_1$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_2.c <_L t_1.c$) then
        $(l_1, l_2) \leftarrow$ UNIFICATION($t_1, t_2.l$)
        $(r_1, r_2) \leftarrow$ UNIFICATION($t_1, t_2.r$)
        return (NODE$\{t_2.c\} : l_1; r_1$, NODE$\{t_2.c\} : l_2; r_2$)
    else if ISLEAF($t_2$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_1.c <_L t_2.c$) then
        $(l_1, l_2) \leftarrow$ UNIFICATION($t_1.l, t_2$)
        $(r_1, r_2) \leftarrow$ UNIFICATION($t_1.r, t_2$)
        return (NODE$\{t_1.c\} : l_1; r_1$, NODE$\{t_1.c\} : l_2; r_2$)
    else
        $(l_1, l_2) \leftarrow$ UNIFICATION($t_1.l, t_2.l$)
        $(r_2, r_2) \leftarrow$ UNIFICATION($t_1.r, t_2.r$)
        return (NODE$\{t_1.c\} : l_1; r_1$, NODE$\{t_2.c\} : l_2; r_2$)
```





Join



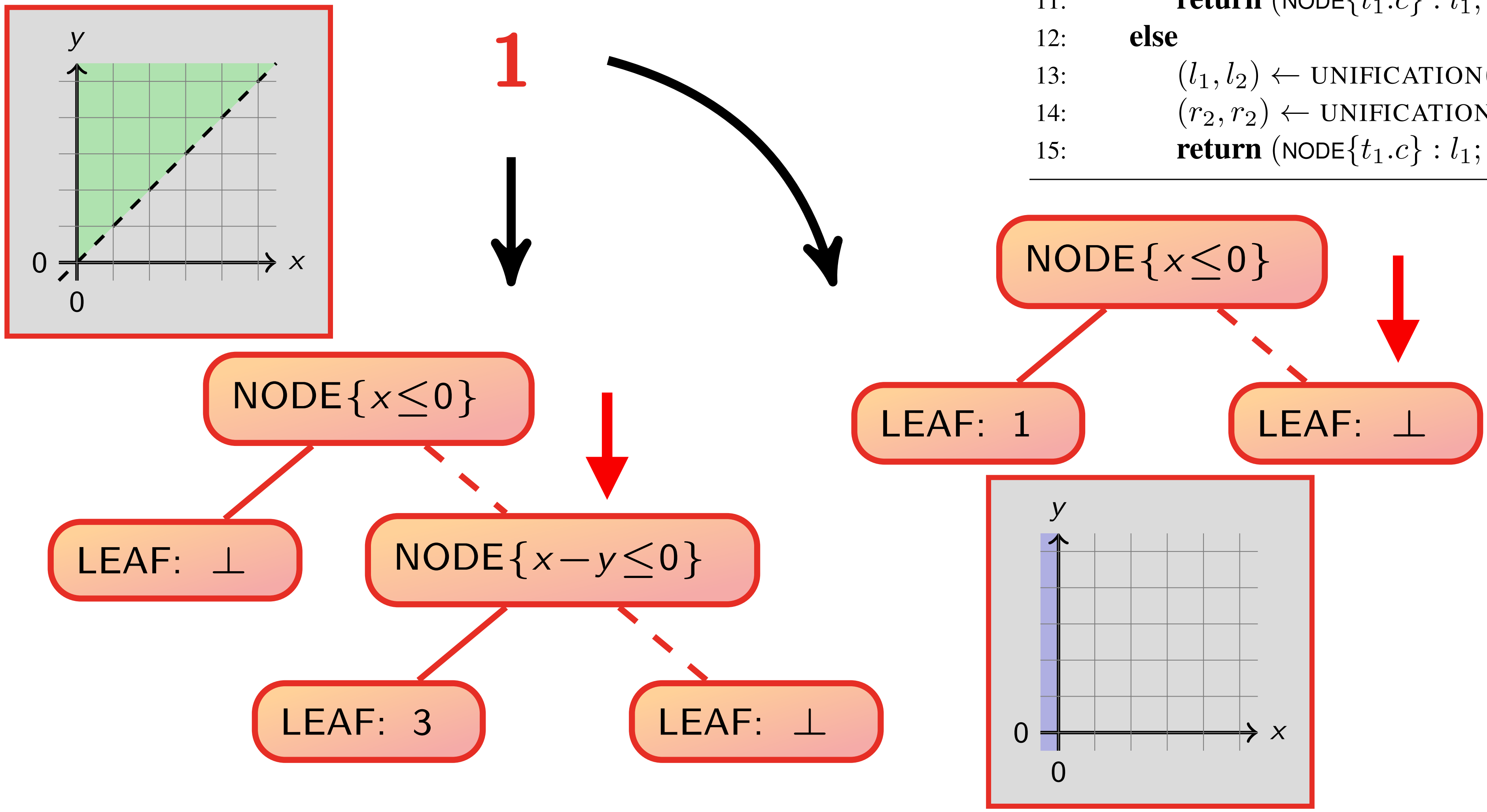
NODE$\{x \leq 0\}$

LEAF: 1

NODE$\{x - y \leq 0\}$

Algorithm 1 : Tree Unification

```
function UNIFICATION($t_1$,$t_2$)
    if ISLEAF($t_1$) $\wedge$ ISLEAF($t_2$) then
        return ($t_1$, $t_2$)
    else if ISLEAF($t_1$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_2.c <_L t_1.c$) then
        ($l_1, l_2$) $\leftarrow$ UNIFICATION($t_1, t_2.l$)
        ($r_1, r_2$) $\leftarrow$ UNIFICATION($t_1, t_2.r$)
        return (NODE$\{t_2.c\} : l_1; r_1$, NODE$\{t_2.c\} : l_2; r_2$)
    else if ISLEAF($t_2$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_1.c <_L t_2.c$) then
        ($l_1, l_2$) $\leftarrow$ UNIFICATION($t_1.l, t_2$)
        ($r_1, r_2$) $\leftarrow$ UNIFICATION($t_1.r, t_2$)
        return (NODE$\{t_1.c\} : l_1; r_1$, NODE$\{t_1.c\} : l_2; r_2$)
    else
        ($l_1, l_2$) $\leftarrow$ UNIFICATION($t_1.l, t_2.l$)
        ($r_2, r_2$) $\leftarrow$ UNIFICATION($t_1.r, t_2.r$)
        return (NODE$\{t_1.c\} : l_1; r_1$, NODE$\{t_2.c\} : l_2; r_2$)
```



NODE$\{x \leq 0\}$

LEAF: $\bot$

NODE$\{x - y \leq 0\}$

LEAF: 3

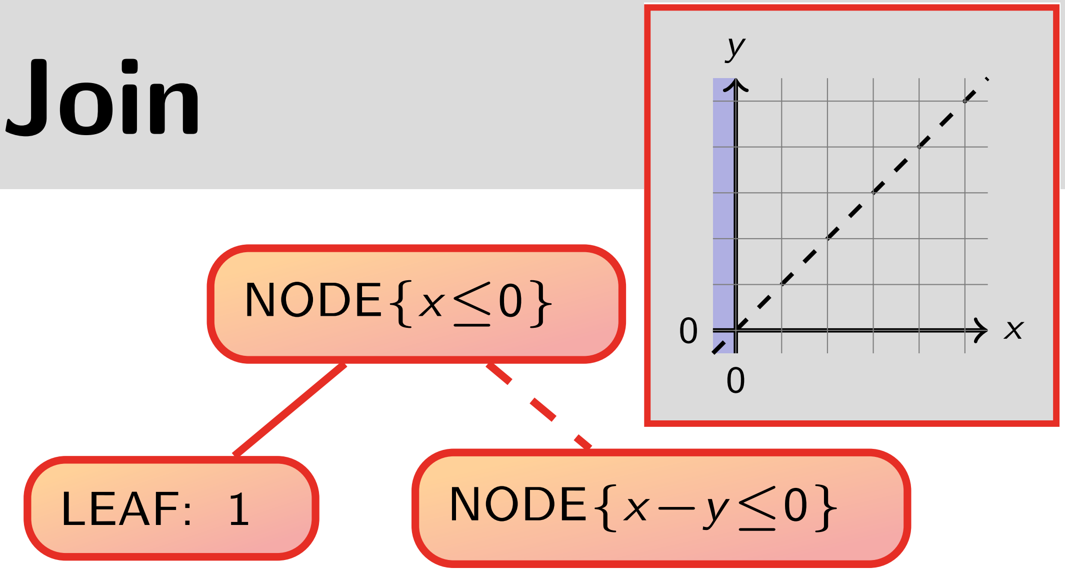
LEAF: $\bot$

NODE$\{x \leq 0\}$

LEAF: 1

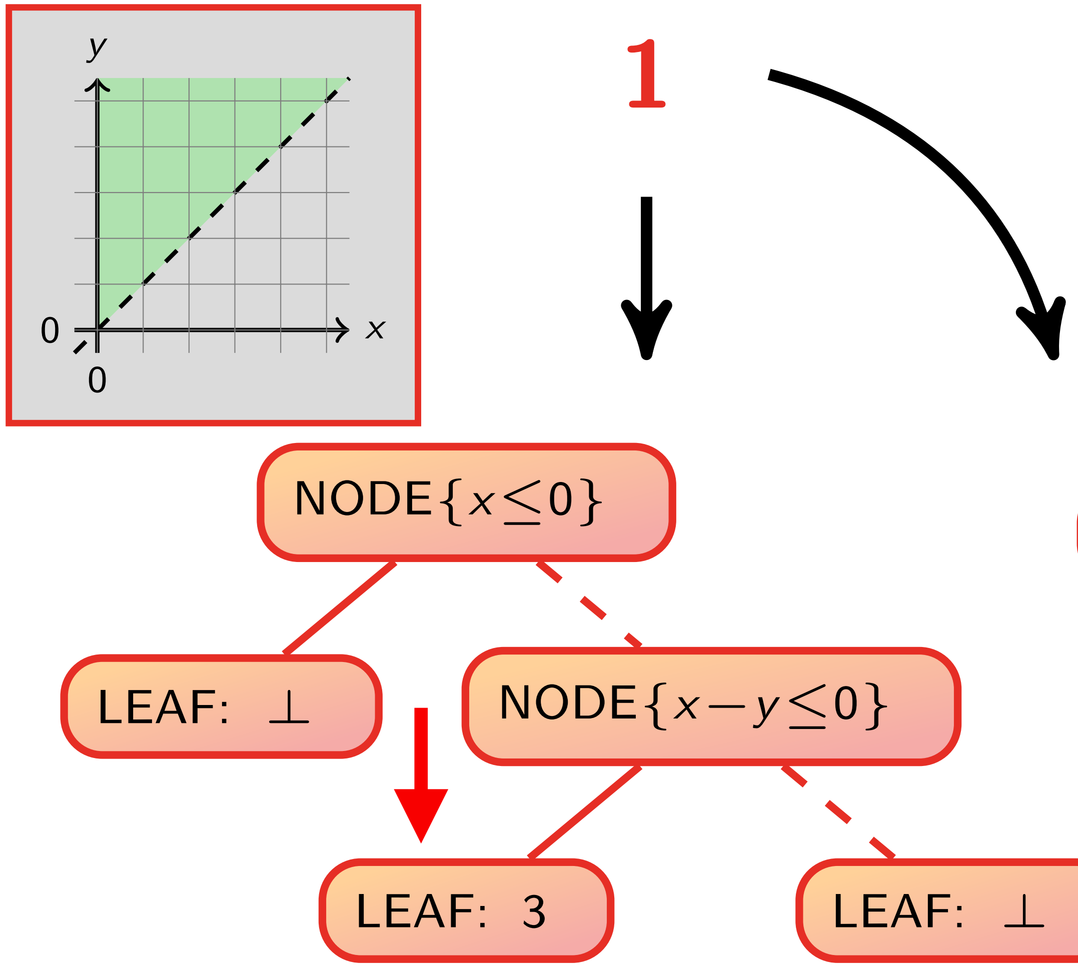
LEAF: $\bot$

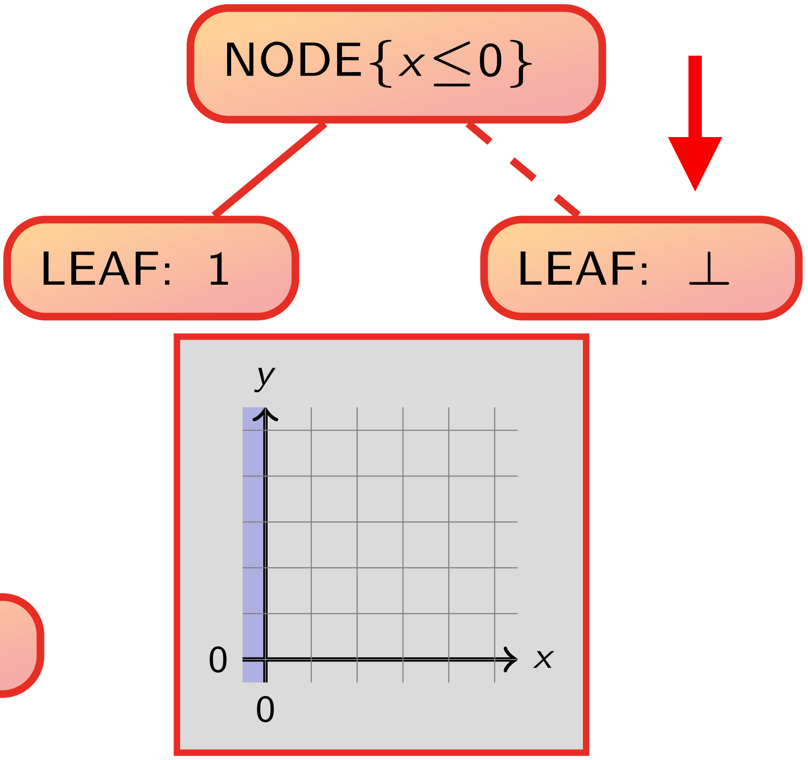
Join



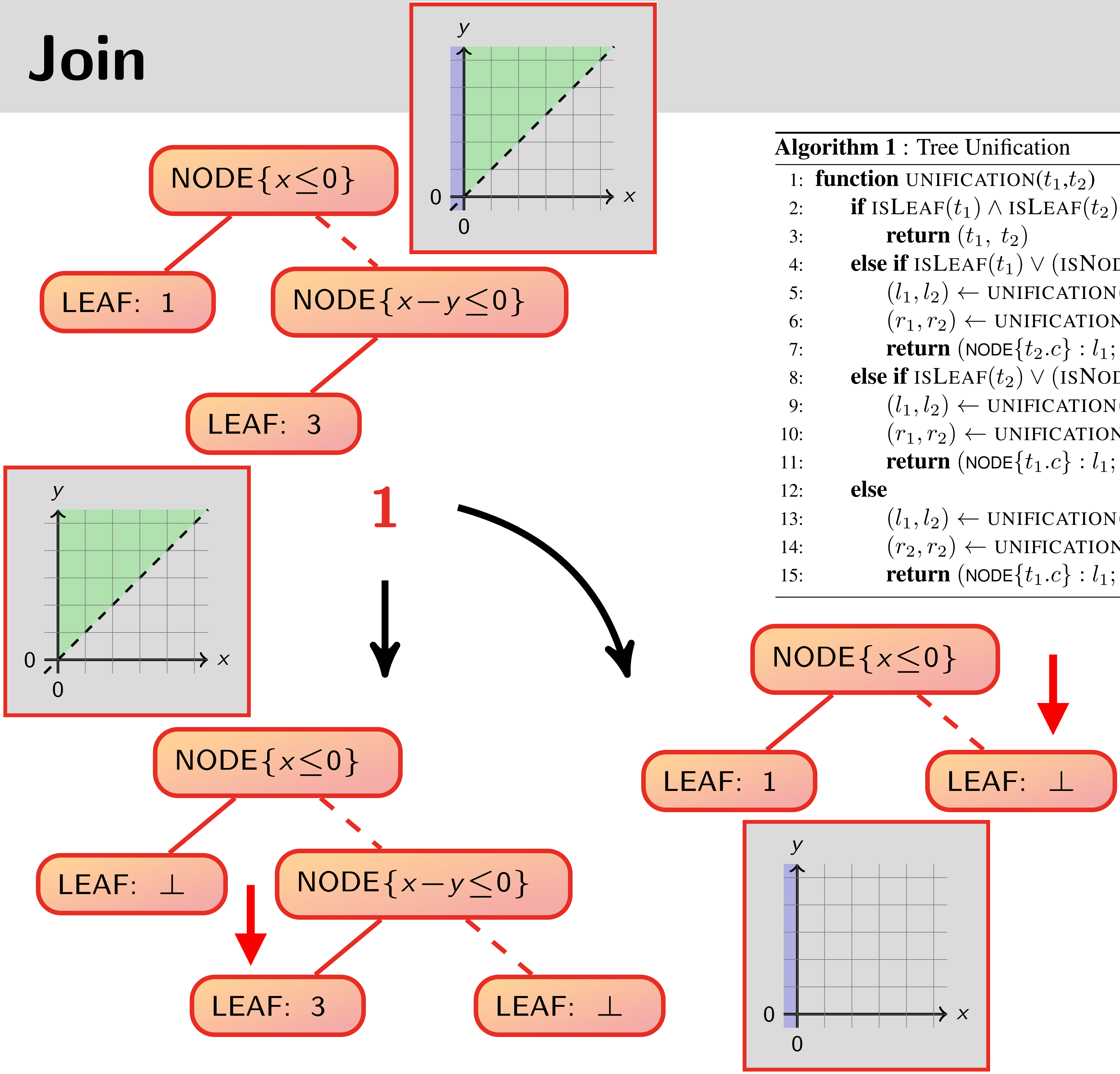
Algorithm 1 : Tree Unification

```
function UNIFICATION($t_1$,$t_2$)
    if ISLEAF($t_1$) $\wedge$ ISLEAF($t_2$) then
        return ($t_1$, $t_2$)
    else if ISLEAF($t_1$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_2.c <_L t_1.c$) then
        $(l_1, l_2) \leftarrow$ UNIFICATION($t_1, t_2.l$)
        $(r_1, r_2) \leftarrow$ UNIFICATION($t_1, t_2.r$)
        return (NODE$\{t_2.c\} : l_1; r_1$, NODE$\{t_2.c\} : l_2; r_2$)
    else if ISLEAF($t_2$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_1.c <_L t_2.c$) then
        $(l_1, l_2) \leftarrow$ UNIFICATION($t_1.l, t_2$)
        $(r_1, r_2) \leftarrow$ UNIFICATION($t_1.r, t_2$)
        return (NODE$\{t_1.c\} : l_1; r_1$, NODE$\{t_1.c\} : l_2; r_2$)
    else
        $(l_1, l_2) \leftarrow$ UNIFICATION($t_1.l, t_2.l$)
        $(r_2, r_2) \leftarrow$ UNIFICATION($t_1.r, t_2.r$)
        return (NODE$\{t_1.c\} : l_1; r_1$, NODE$\{t_2.c\} : l_2; r_2$)
```





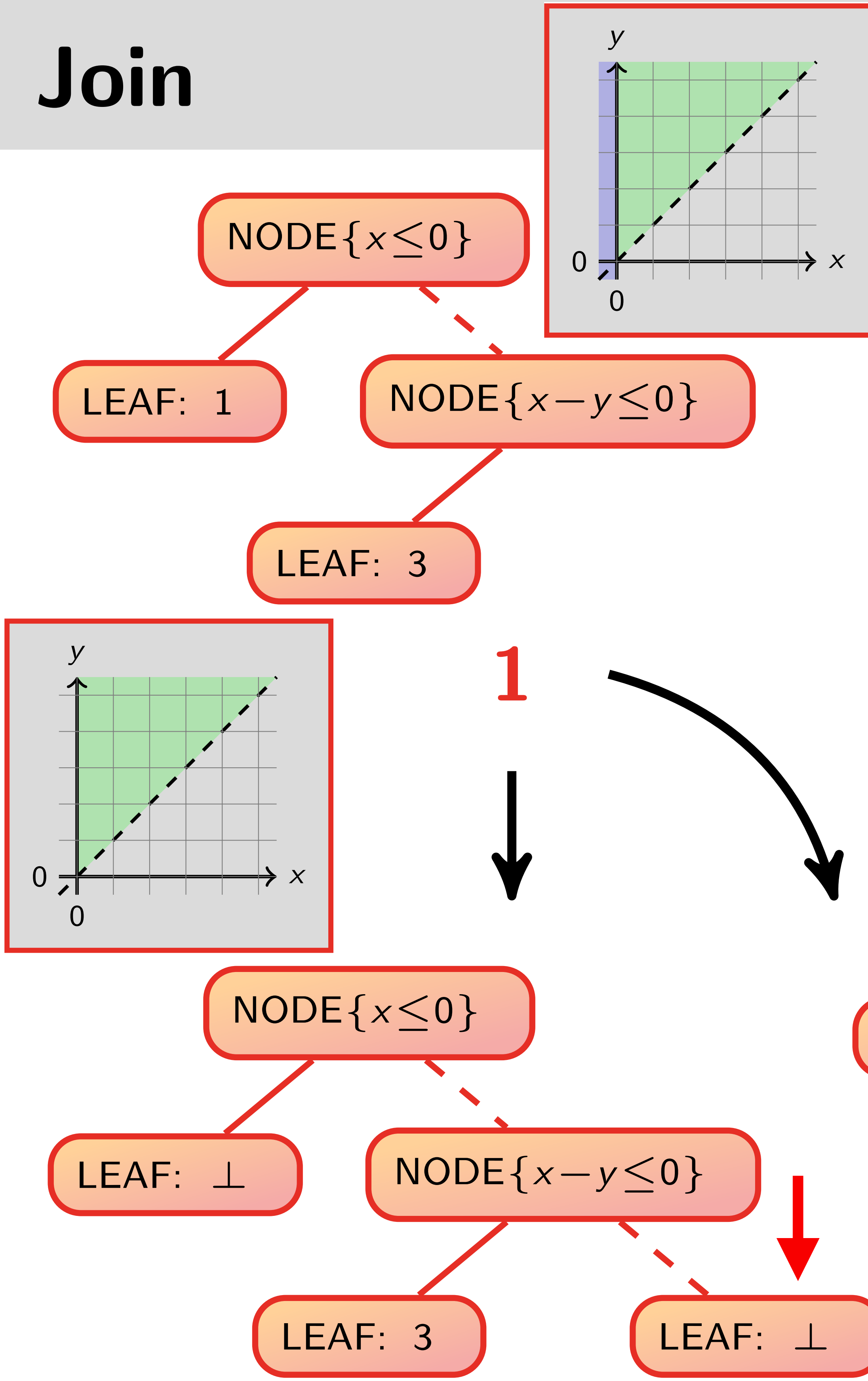
Join

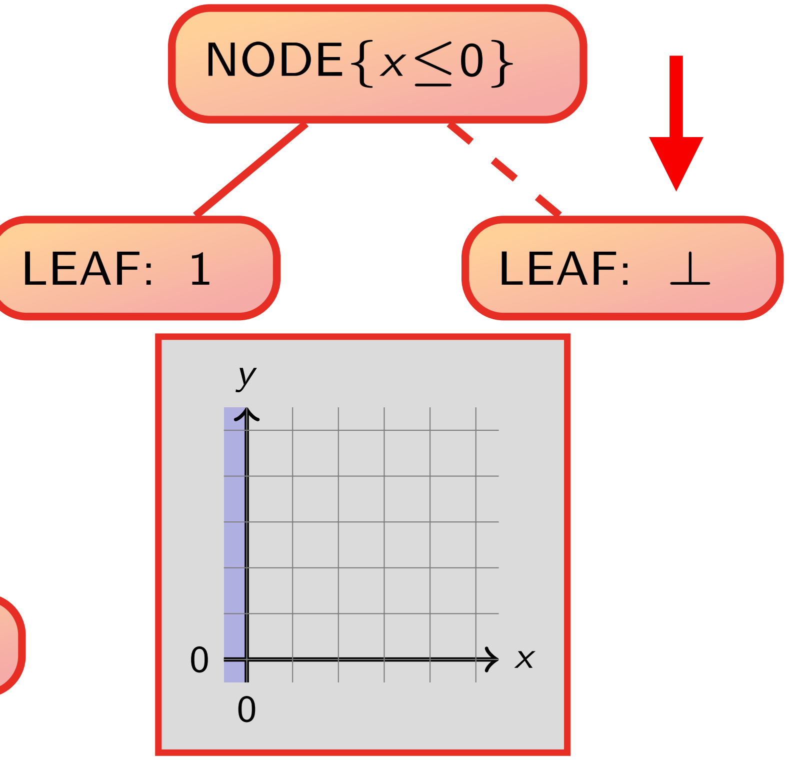


Algorithm 1 : Tree Unification

```
function UNIFICATION($t_1$,$t_2$)
    if ISLEAF($t_1$) $\wedge$ ISLEAF($t_2$) then
        return ($t_1$, $t_2$)
    else if ISLEAF($t_1$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_2.c <_L t_1.c$) then
        ($l_1, l_2$) $\leftarrow$ UNIFICATION($t_1, t_2.l$)
        ($r_1, r_2$) $\leftarrow$ UNIFICATION($t_1, t_2.r$)
        return (NODE{$t_2.c$} : $l_1; r_1$, NODE{$t_2.c$} : $l_2; r_2$)
    else if ISLEAF($t_2$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_1.c <_L t_2.c$) then
        ($l_1, l_2$) $\leftarrow$ UNIFICATION($t_1.l, t_2$)
        ($r_1, r_2$) $\leftarrow$ UNIFICATION($t_1.r, t_2$)
        return (NODE{$t_1.c$} : $l_1; r_1$, NODE{$t_1.c$} : $l_2; r_2$)
    else
        ($l_1, l_2$) $\leftarrow$ UNIFICATION($t_1.l, t_2.l$)
        ($r_2, r_2$) $\leftarrow$ UNIFICATION($t_1.r, t_2.r$)
        return (NODE{$t_1.c$} : $l_1; r_1$, NODE{$t_2.c$} : $l_2; r_2$)
```

Join

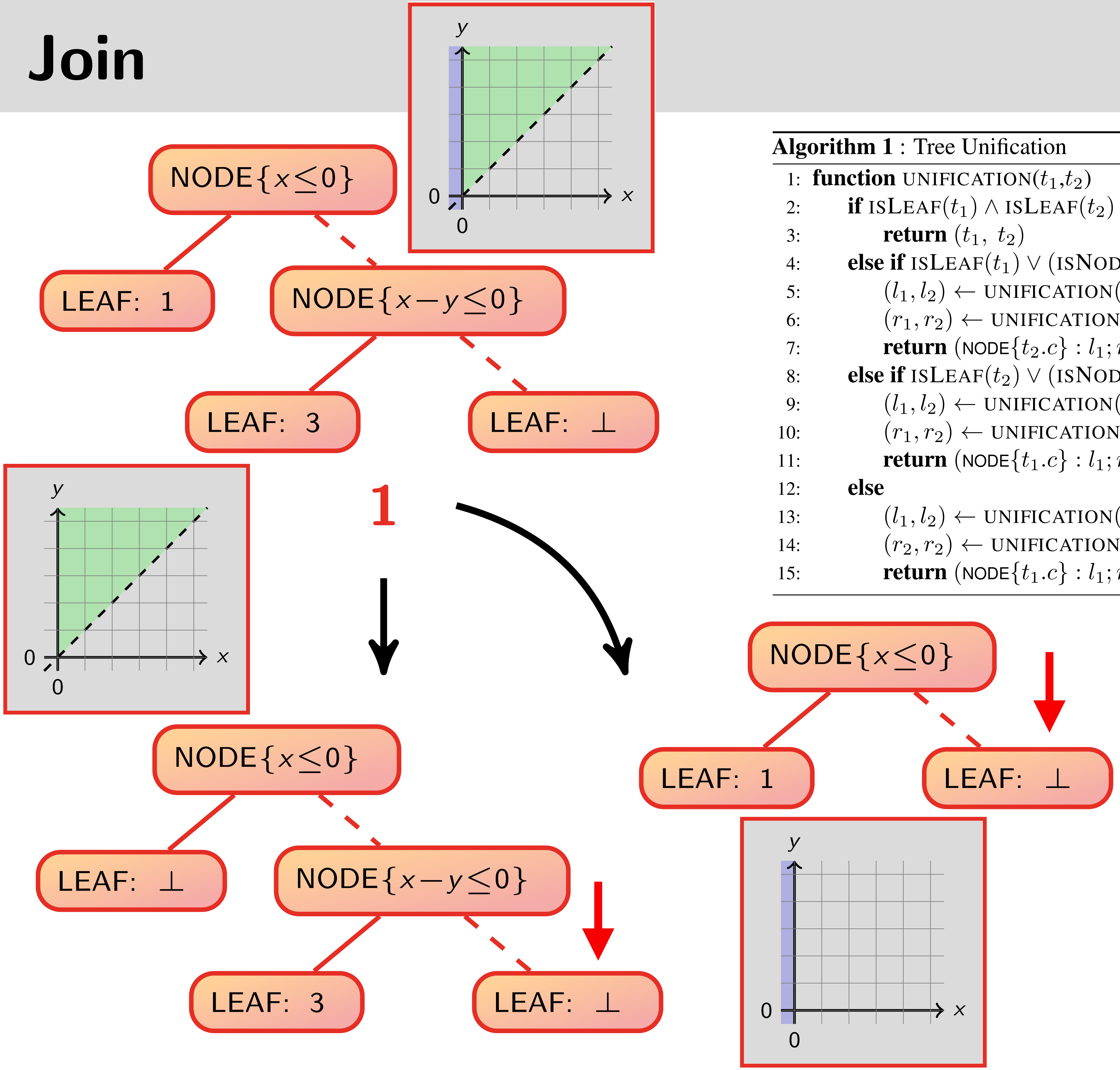




Algorithm 1 : Tree Unification

```
function UNIFICATION($t_1$,$t_2$)
    if ISLEAF($t_1$) $\wedge$ ISLEAF($t_2$) then
        return ($t_1$, $t_2$)
    else if ISLEAF($t_1$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_2.c <_{\mathsf{L}} t_1.c$) then
        $(l_1, l_2) \leftarrow$ UNIFICATION($t_1, t_2.l$)
        $(r_1, r_2) \leftarrow$ UNIFICATION($t_1, t_2.r$)
        return (NODE$\{t_2.c\} : l_1; r_1$, NODE$\{t_2.c\} : l_2; r_2$)
    else if ISLEAF($t_2$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_1.c <_{\mathsf{L}} t_2.c$) then
        $(l_1, l_2) \leftarrow$ UNIFICATION($t_1.l, t_2$)
        $(r_1, r_2) \leftarrow$ UNIFICATION($t_1.r, t_2$)
        return (NODE$\{t_1.c\} : l_1; r_1$, NODE$\{t_1.c\} : l_2; r_2$)
    else
        $(l_1, l_2) \leftarrow$ UNIFICATION($t_1.l, t_2.l$)
        $(r_2, r_2) \leftarrow$ UNIFICATION($t_1.r, t_2.r$)
        return (NODE$\{t_1.c\} : l_1; r_1$, NODE$\{t_2.c\} : l_2; r_2$)
```

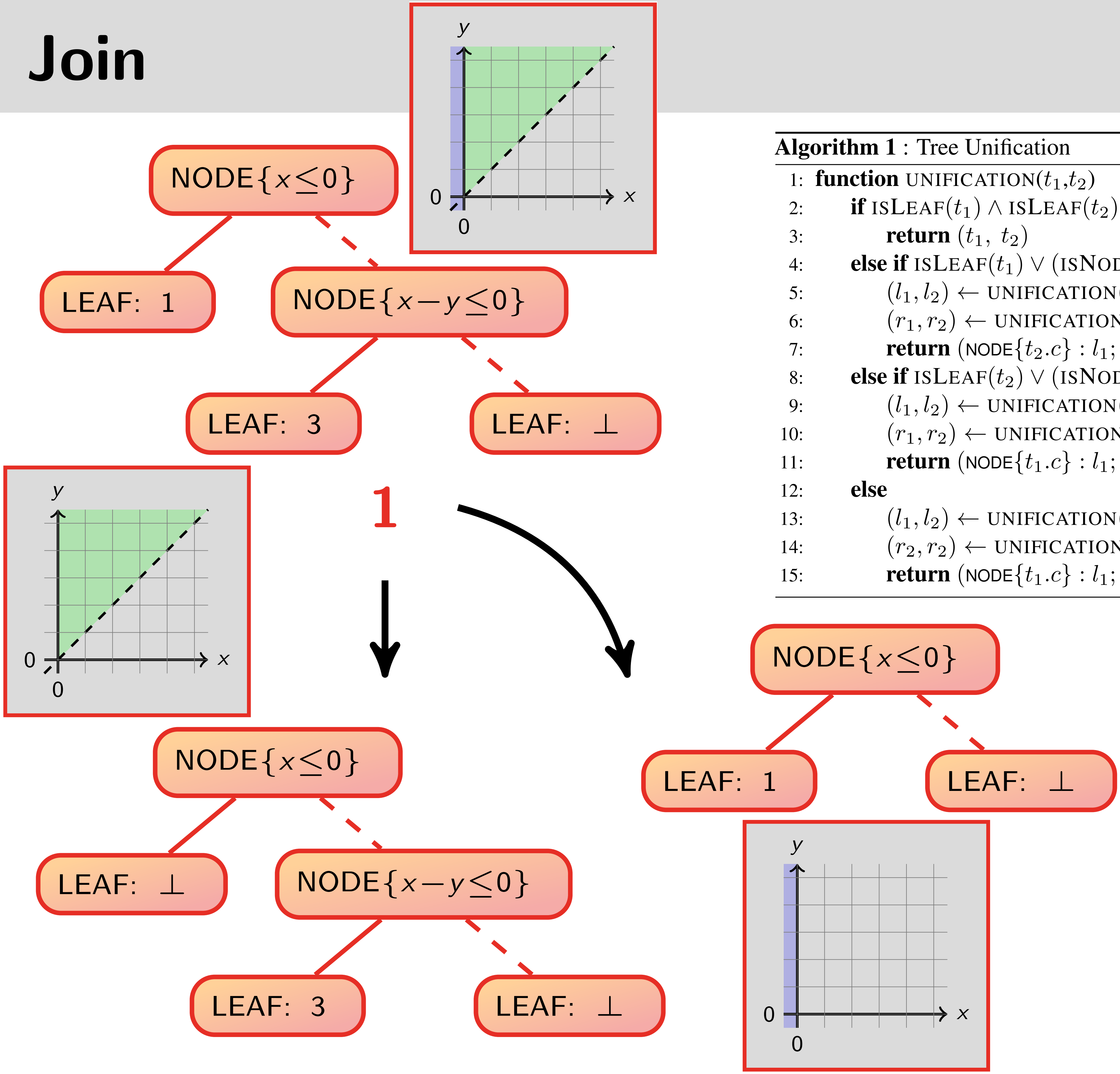
Join



Algorithm 1 : Tree Unification

```
function UNIFICATION($t_1$,$t_2$)
    if ISLEAF($t_1$) $\wedge$ ISLEAF($t_2$) then
        return ($t_1$, $t_2$)
    else if ISLEAF($t_1$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_2.c <_L t_1.c$) then
        ($l_1, l_2$) $\leftarrow$ UNIFICATION($t_1, t_2.l$)
        ($r_1, r_2$) $\leftarrow$ UNIFICATION($t_1, t_2.r$)
        return (NODE{$t_2.c$} : $l_1; r_1$, NODE{$t_2.c$} : $l_2; r_2$)
    else if ISLEAF($t_2$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_1.c <_L t_2.c$) then
        ($l_1, l_2$) $\leftarrow$ UNIFICATION($t_1.l, t_2$)
        ($r_1, r_2$) $\leftarrow$ UNIFICATION($t_1.r, t_2$)
        return (NODE{$t_1.c$} : $l_1; r_1$, NODE{$t_1.c$} : $l_2; r_2$)
    else
        ($l_1, l_2$) $\leftarrow$ UNIFICATION($t_1.l, t_2.l$)
        ($r_2, r_2$) $\leftarrow$ UNIFICATION($t_1.r, t_2.r$)
        return (NODE{$t_1.c$} : $l_1; r_1$, NODE{$t_2.c$} : $l_2; r_2$)
```

Join



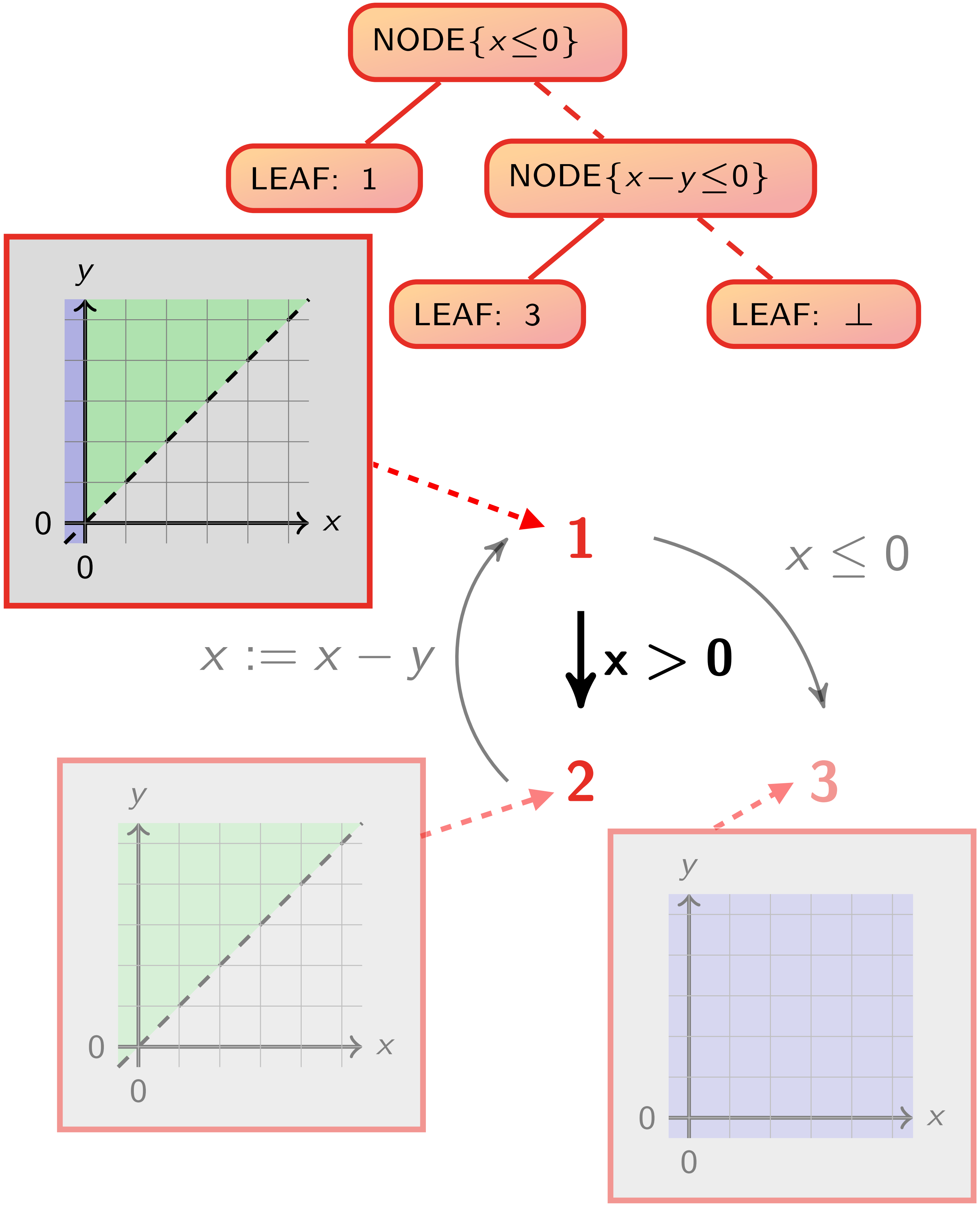
Algorithm 1 : Tree Unification

```
function UNIFICATION($t_1$,$t_2$)
    if ISLEAF($t_1$) $\wedge$ ISLEAF($t_2$) then
        return ($t_1$, $t_2$)
    else if ISLEAF($t_1$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_2.c <_L t_1.c$) then
        $(l_1, l_2) \leftarrow$ UNIFICATION($t_1, t_2.l$)
        $(r_1, r_2) \leftarrow$ UNIFICATION($t_1, t_2.r$)
        return (NODE$\{t_2.c\} : l_1; r_1$, NODE$\{t_2.c\} : l_2; r_2$)
    else if ISLEAF($t_2$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_1.c <_L t_2.c$) then
        $(l_1, l_2) \leftarrow$ UNIFICATION($t_1.l, t_2$)
        $(r_1, r_2) \leftarrow$ UNIFICATION($t_1.r, t_2$)
        return (NODE$\{t_1.c\} : l_1; r_1$, NODE$\{t_1.c\} : l_2; r_2$)
    else
        $(l_1, l_2) \leftarrow$ UNIFICATION($t_1.l, t_2.l$)
        $(r_2, r_2) \leftarrow$ UNIFICATION($t_1.r, t_2.r$)
        return (NODE$\{t_1.c\} : l_1; r_1$, NODE$\{t_2.c\} : l_2; r_2$)
```

Example

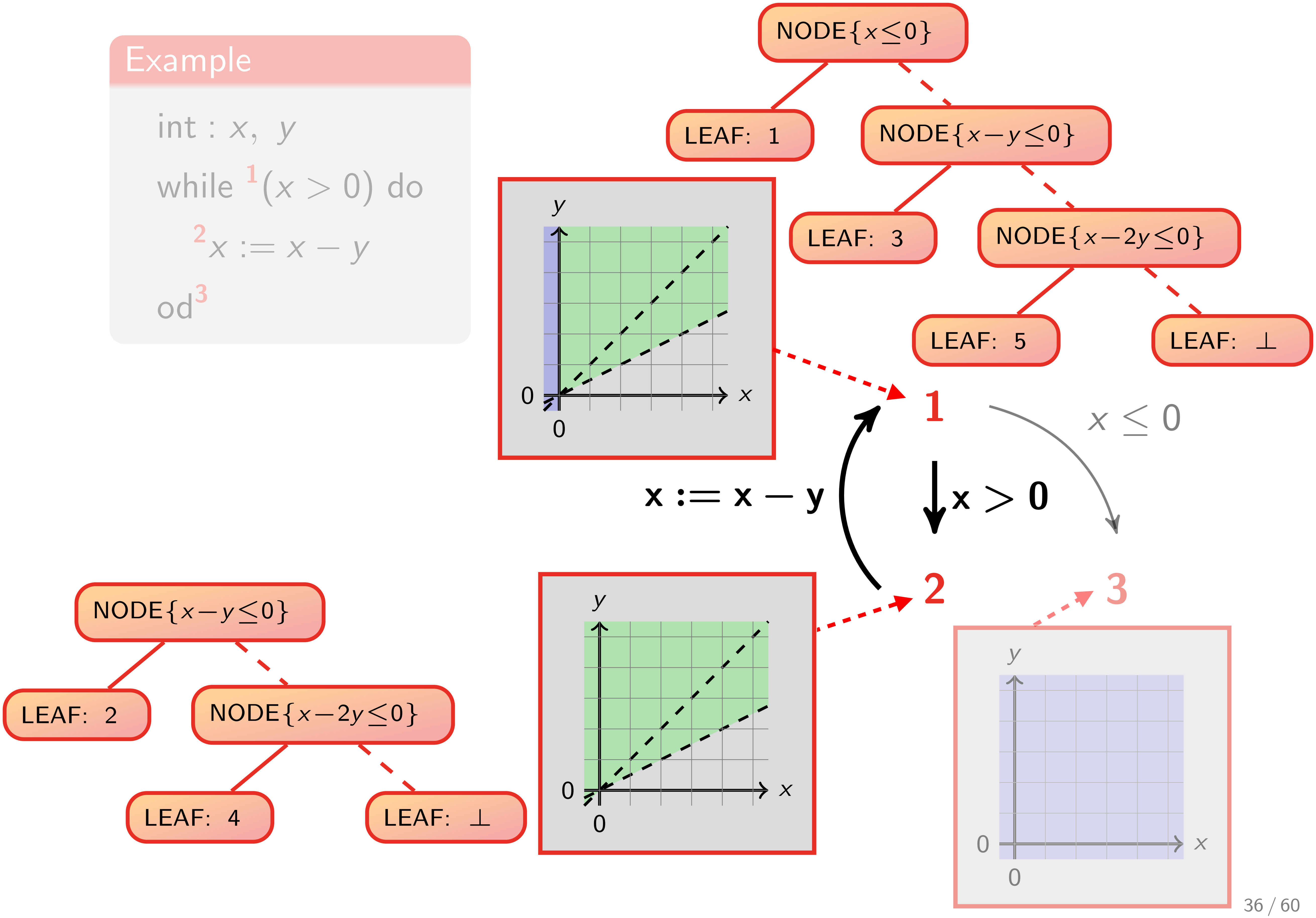
```
int : x,  y
while ¹(x > 0) do
    ²x := x − y
od³
```

we have taken $x > 0$ into account and we have done the join



Example

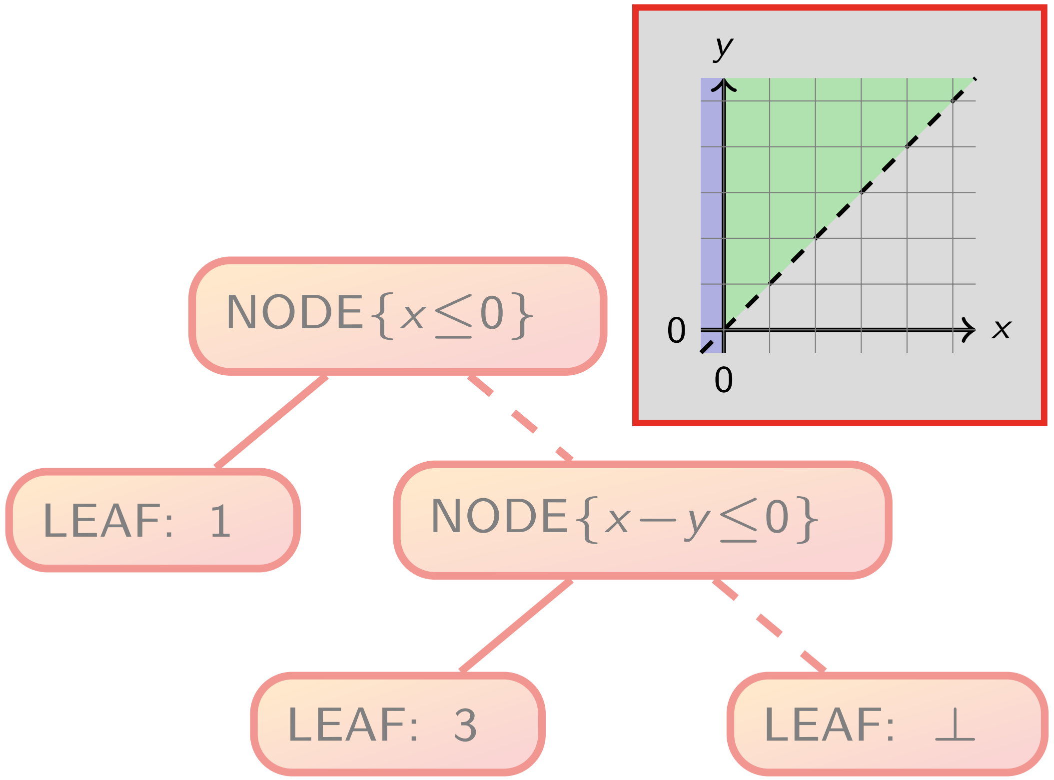
```
int : x, y
while ¹(x > 0) do
    ²x := x − y
od³
```

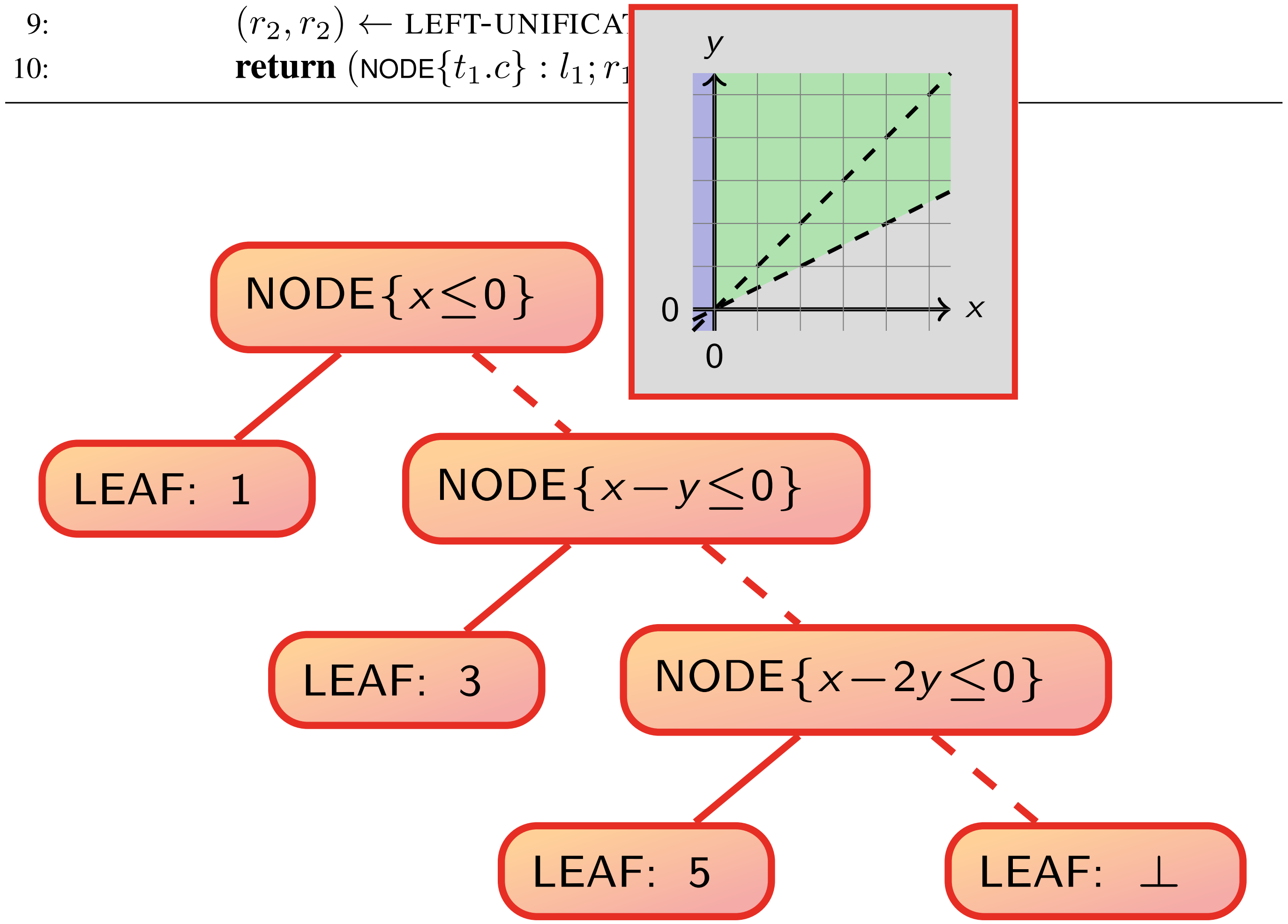


Widening: Left Unification

Algorithm 5 : Tree Left Unification

```
function LEFT-UNIFICATION($t_1$,$t_2$)
    if ISLEAF($t_1$) $\wedge$ ISLEAF($t_2$) then
        return ($t_1$, $t_2$)
    else
        if ISLEAF($t_1$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_2.c <_L t_1.c$) then
            return LEFT-UNIFICATION($t_1, t_2.l \sqcup_T t_2.r$)
        else
            $(l_1, l_2) \leftarrow$ LEFT-UNIFICATION($t_1.l, t_2.l$)
            $(r_2, r_2) \leftarrow$ LEFT-UNIFICAT
            return (NODE$\{t_1.c\}$ : $l_1; r_1$
```

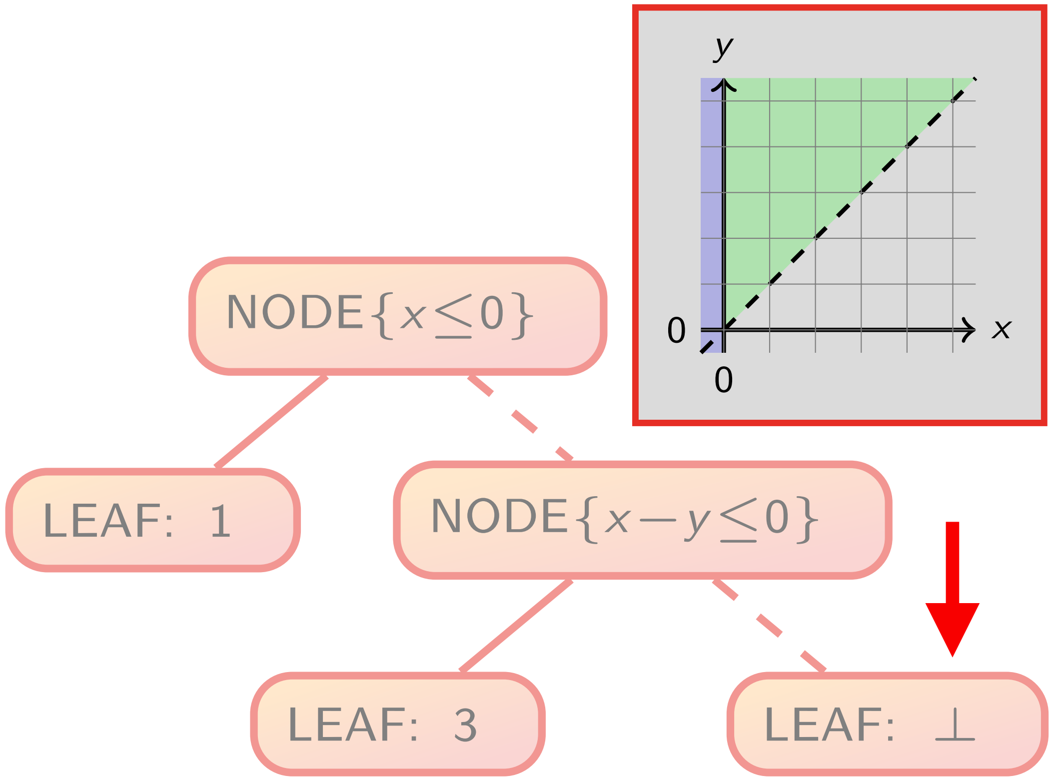


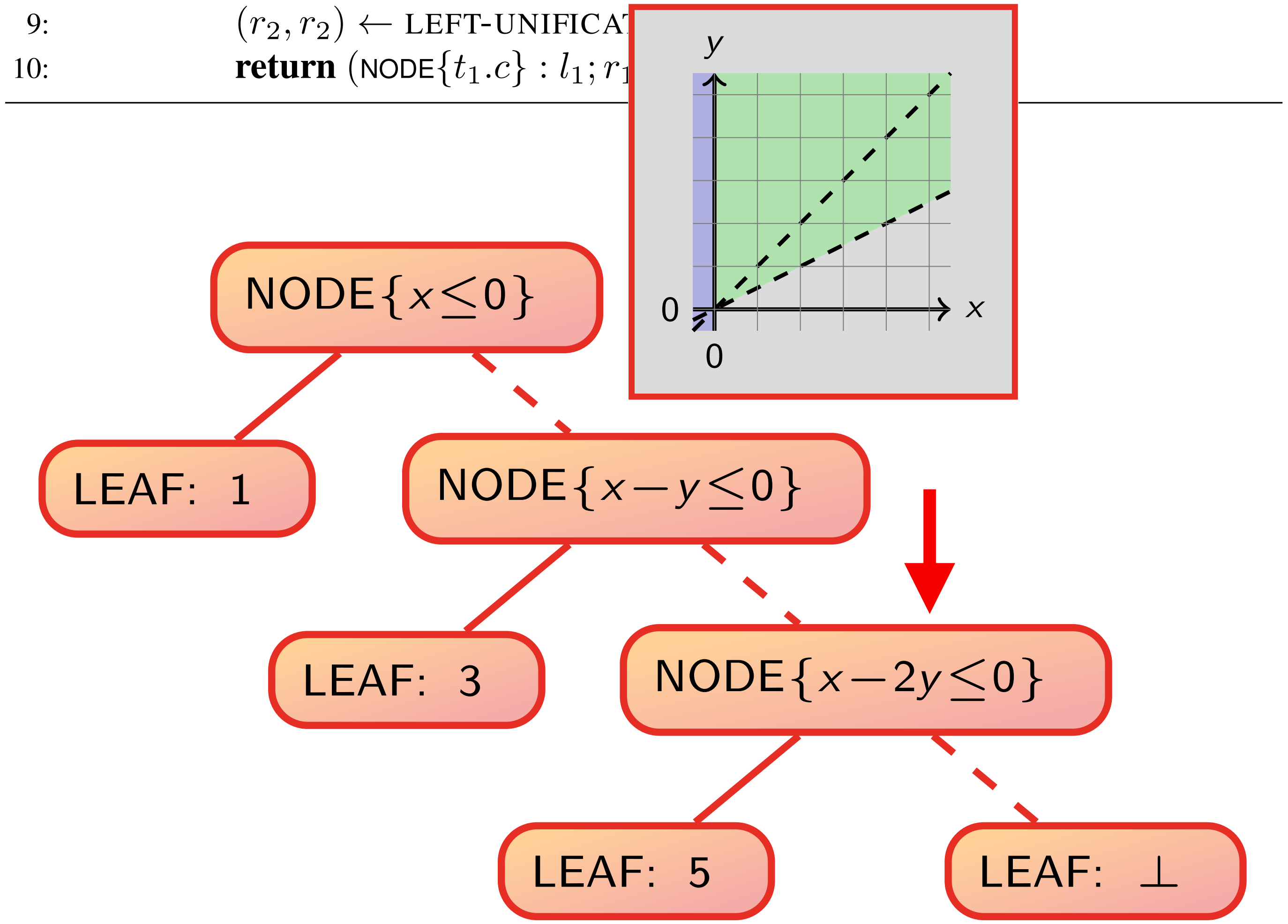


Widening: Left Unification

Algorithm 5 : Tree Left Unification

```
function LEFT-UNIFICATION($t_1$,$t_2$)
    if ISLEAF($t_1$) $\wedge$ ISLEAF($t_2$) then
        return ($t_1$, $t_2$)
    else
        if ISLEAF($t_1$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_2.c <_{\mathsf{L}} t_1.c$) then
            return LEFT-UNIFICATION($t_1, t_2.l \sqcup_{\mathsf{T}} t_2.r$)
        else
            $(l_1, l_2) \leftarrow$ LEFT-UNIFICATION($t_1.l, t_2.l$)
            $(r_2, r_2) \leftarrow$ LEFT-UNIFICAT
            return (NODE$\{t_1.c\}$ : $l_1; r_1$
```

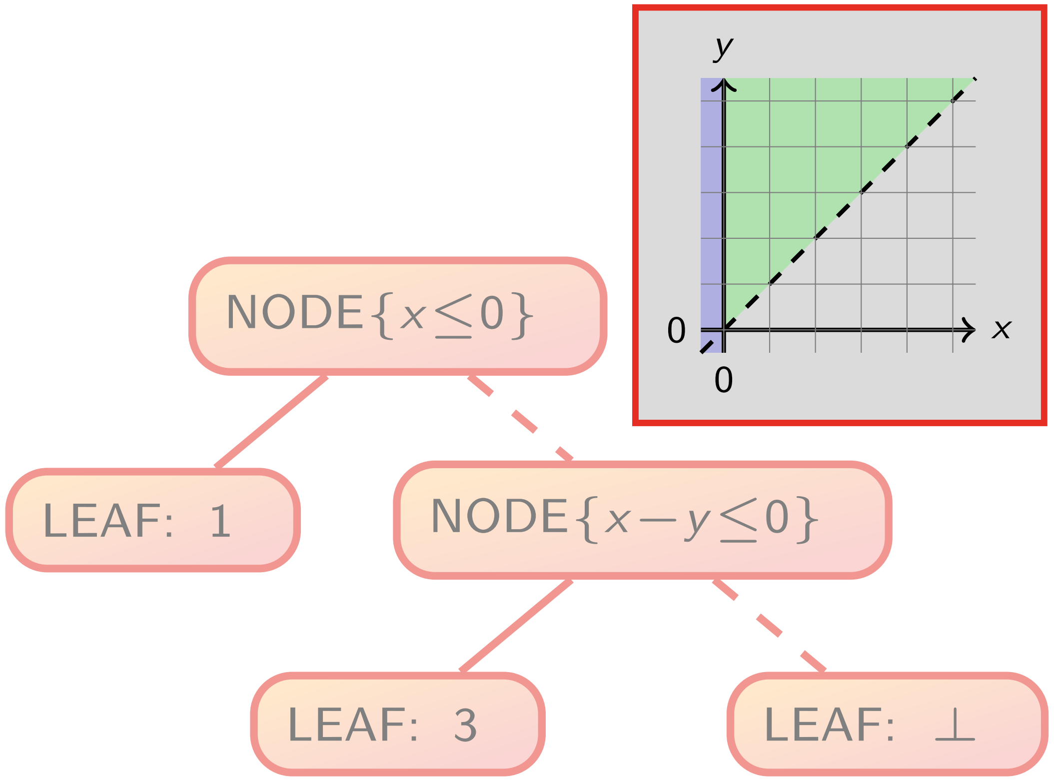


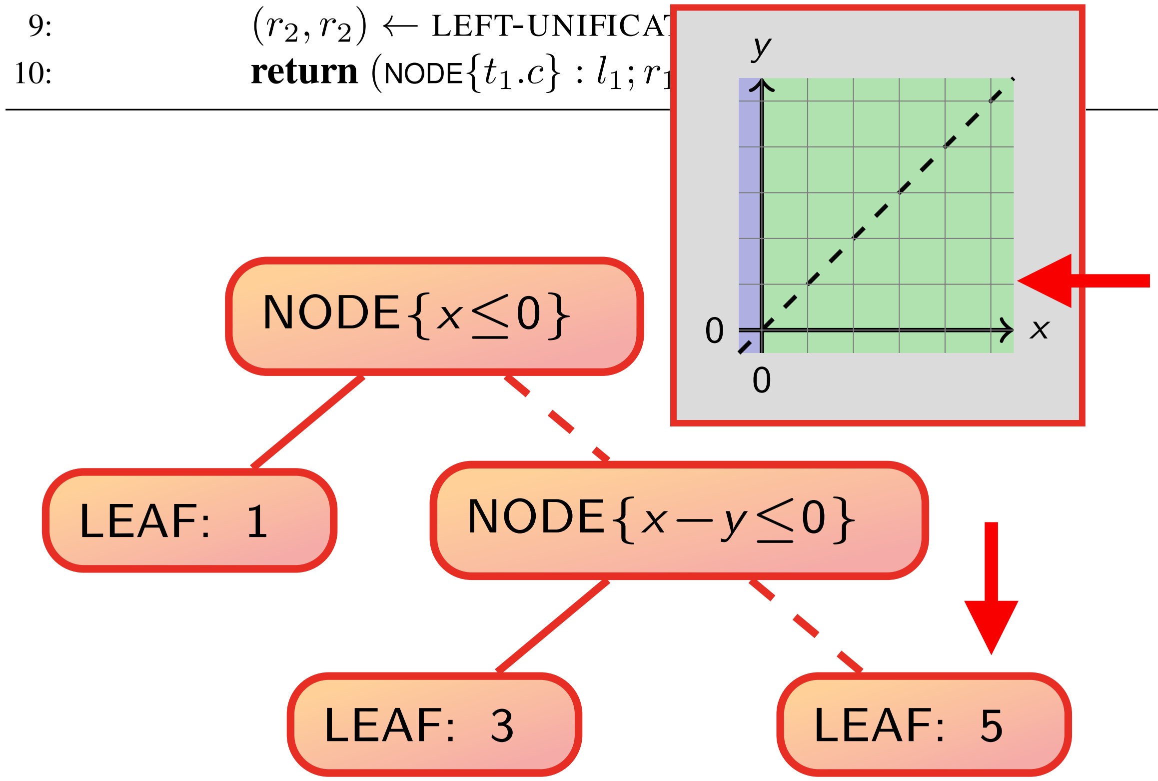


Widening: Left Unification

Algorithm 5 : Tree Left Unification

```
function LEFT-UNIFICATION($t_1$,$t_2$)
    if ISLEAF($t_1$) $\wedge$ ISLEAF($t_2$) then
        return ($t_1$, $t_2$)
    else
        if ISLEAF($t_1$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_2.c <_L t_1.c$) then
            return LEFT-UNIFICATION($t_1, t_2.l \sqcup_T t_2.r$)
        else
            $(l_1, l_2) \leftarrow$ LEFT-UNIFICATION($t_1.l, t_2.l$)
            $(r_2, r_2) \leftarrow$ LEFT-UNIFICAT
            return (NODE$\{t_1.c\} : l_1; r_1$
```





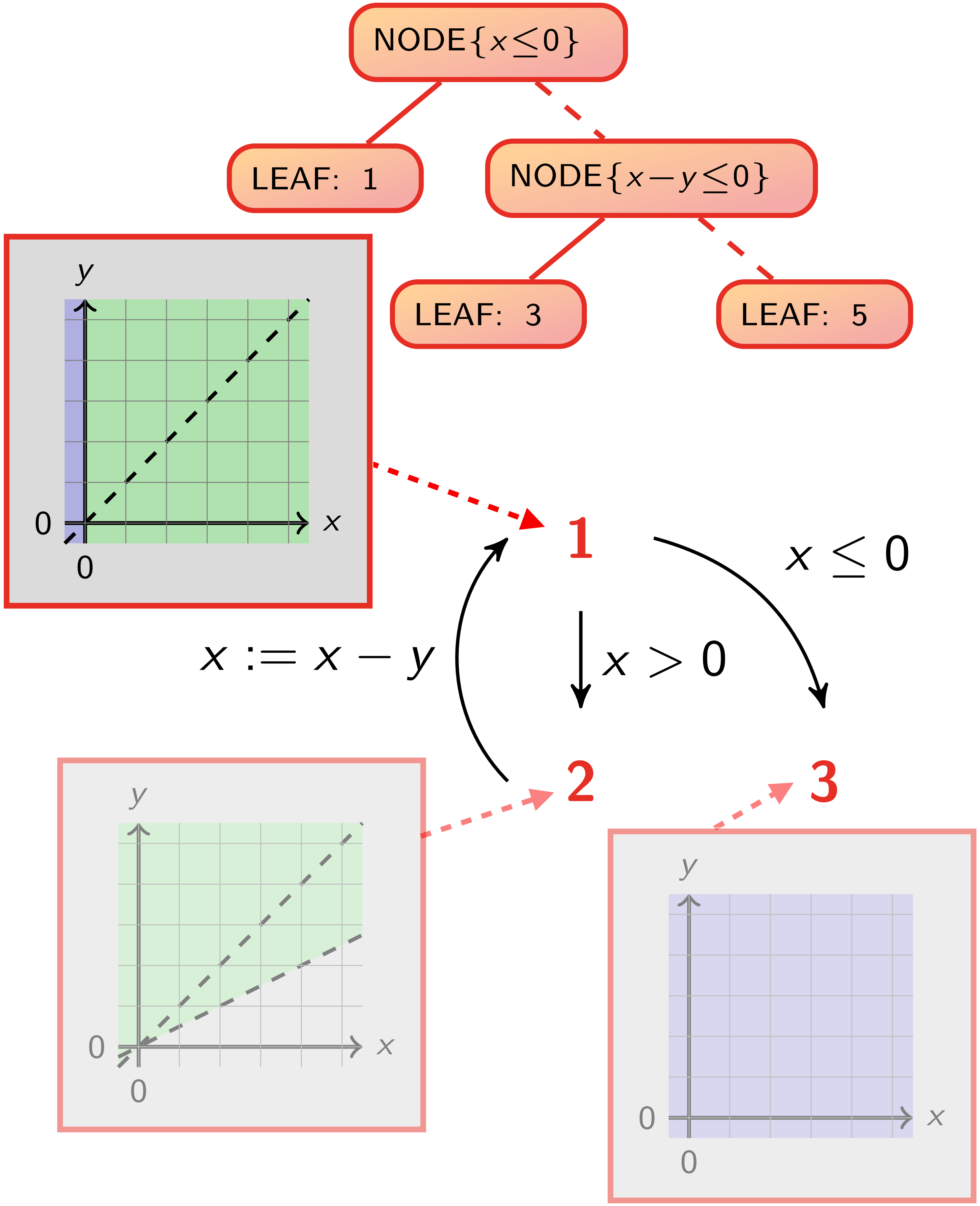
Example

int : x, y

while [1]$(x > 0)$ do

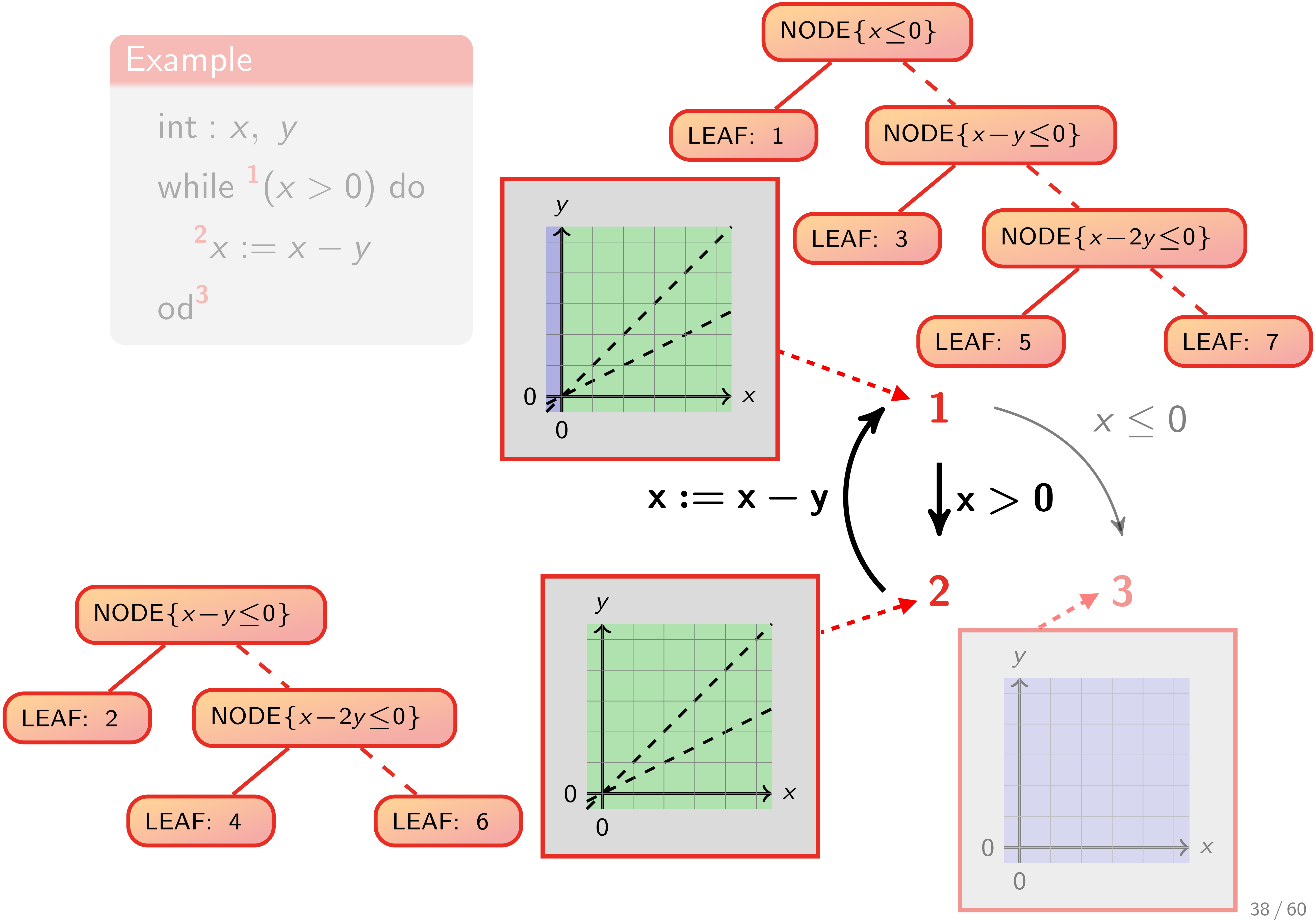
[2]$x := x - y$

od[3]



Example

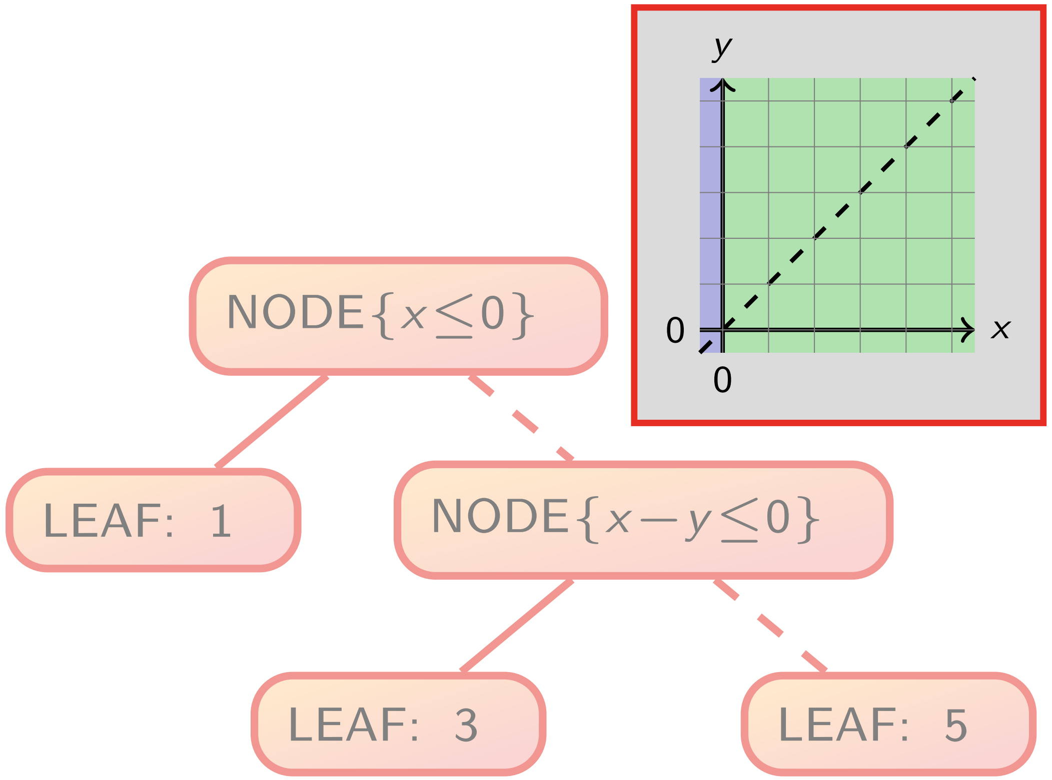
```
int : $x$, $y$
while $^1(x > 0)$ do
    $^2x := x - y$
od$^3$
```

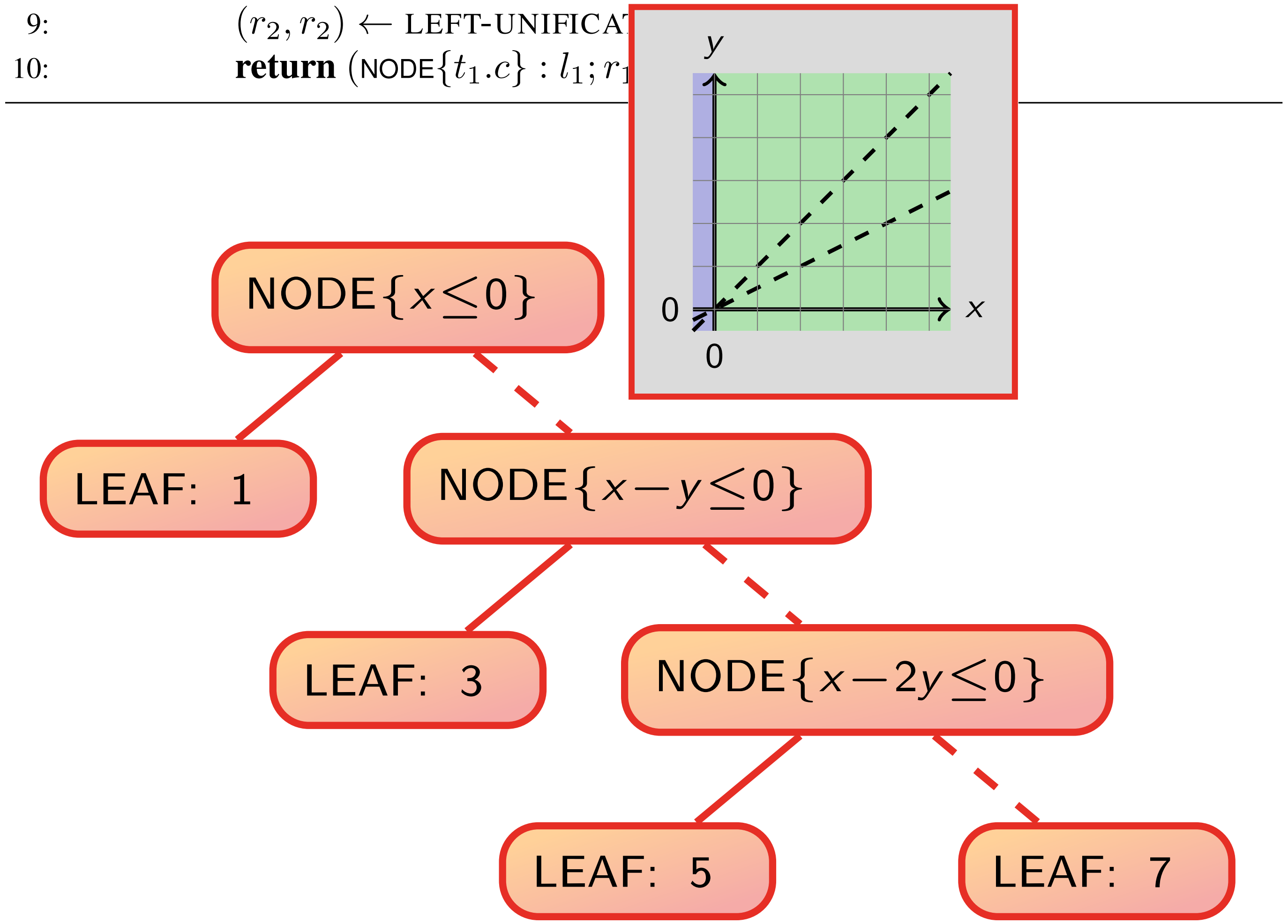


Widening: Domain Over-Approximation

Algorithm 5 : Tree Left Unification

```
function LEFT-UNIFICATION($t_1$,$t_2$)
    if ISLEAF($t_1$) $\wedge$ ISLEAF($t_2$) then
        return $(t_1,\ t_2)$
    else
        if ISLEAF($t_1$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_2.c <_{\mathsf{L}} t_1.c$) then
            return LEFT-UNIFICATION($t_1, t_2.l \sqcup_{\mathsf{T}} t_2.r$)
        else
            $(l_1, l_2) \leftarrow$ LEFT-UNIFICATION($t_1.l, t_2.l$)
            $(r_2, r_2) \leftarrow$ LEFT-UNIFICAT
            return (NODE$\{t_1.c\} : l_1; r_1$
```

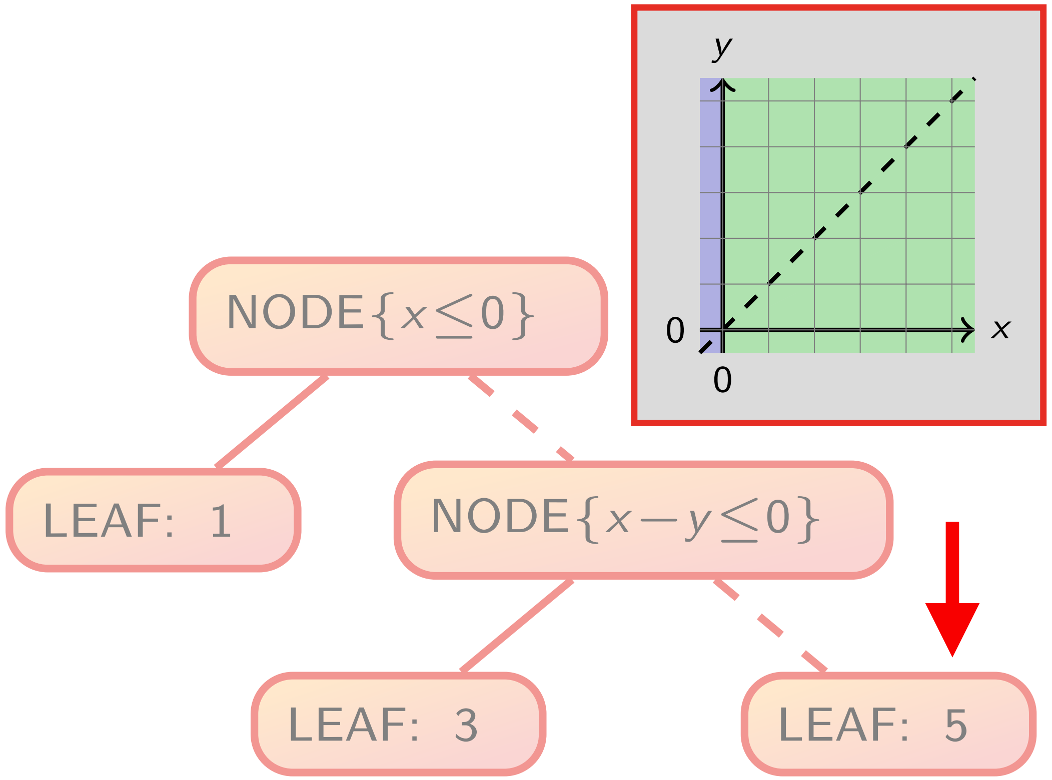


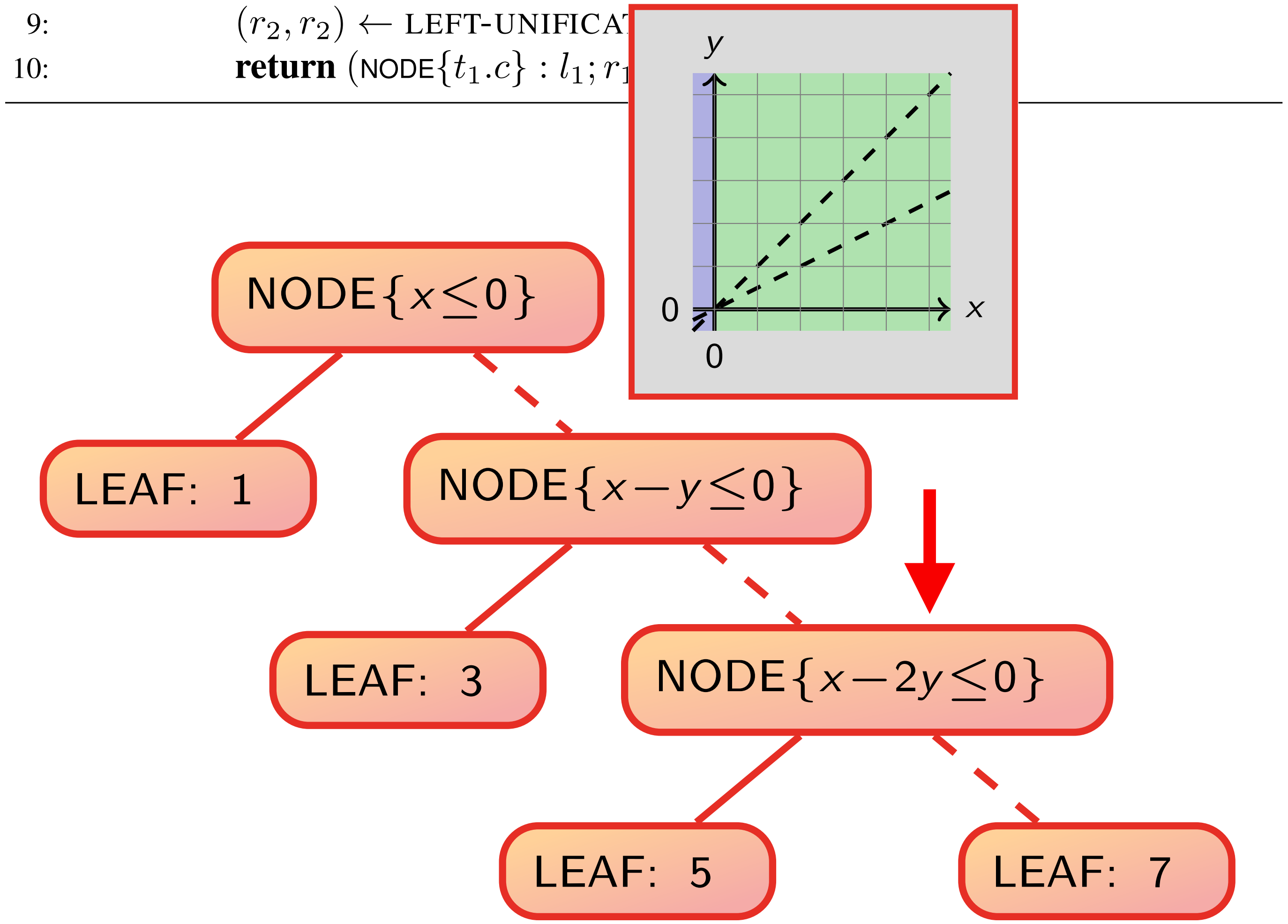


Widening: Domain Over-Approximation

Algorithm 5 : Tree Left Unification

```
function LEFT-UNIFICATION($t_1$,$t_2$)
    if ISLEAF($t_1$) $\wedge$ ISLEAF($t_2$) then
        return ($t_1$, $t_2$)
    else
        if ISLEAF($t_1$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_2.c <_L t_1.c$) then
            return LEFT-UNIFICATION($t_1, t_2.l \sqcup_T t_2.r$)
        else
            $(l_1, l_2) \leftarrow$ LEFT-UNIFICATION($t_1.l, t_2.l$)
            $(r_2, r_2) \leftarrow$ LEFT-UNIFICAT
            return (NODE$\{t_1.c\}$ : $l_1; r_1$
```

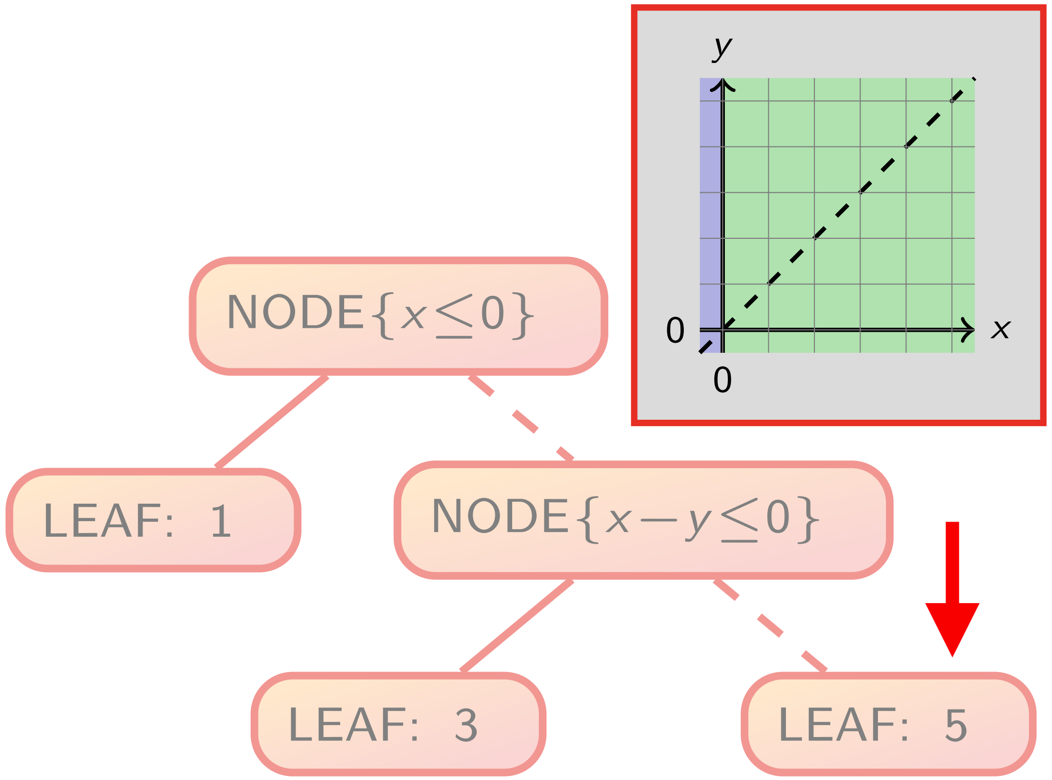


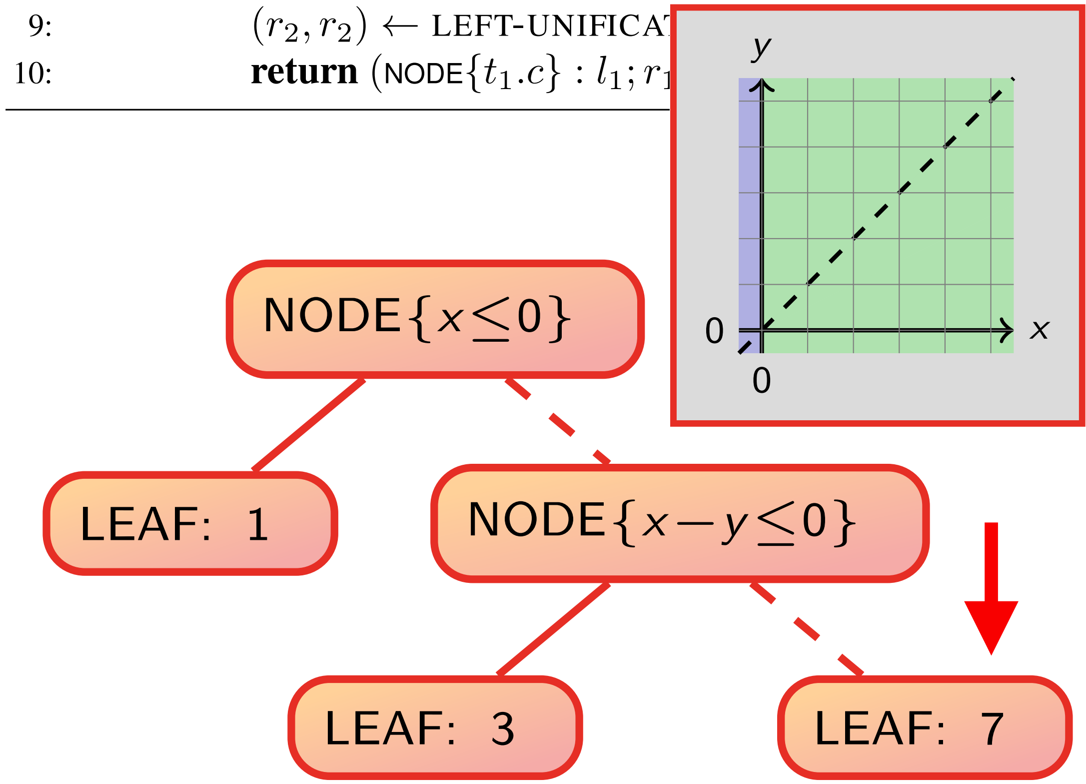


Widening: Domain Over-Approximation

Algorithm 5 : Tree Left Unification

```
function LEFT-UNIFICATION($t_1$,$t_2$)
    if ISLEAF($t_1$) $\wedge$ ISLEAF($t_2$) then
        return ($t_1$, $t_2$)
    else
        if ISLEAF($t_1$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_2.c <_L t_1.c$) then
            return LEFT-UNIFICATION($t_1, t_2.l \sqcup_T t_2.r$)
        else
            $(l_1, l_2) \leftarrow$ LEFT-UNIFICATION($t_1.l, t_2.l$)
            $(r_2, r_2) \leftarrow$ LEFT-UNIFICA
            return (NODE$\{t_1.c\}$ : $l_1; r_1$
```

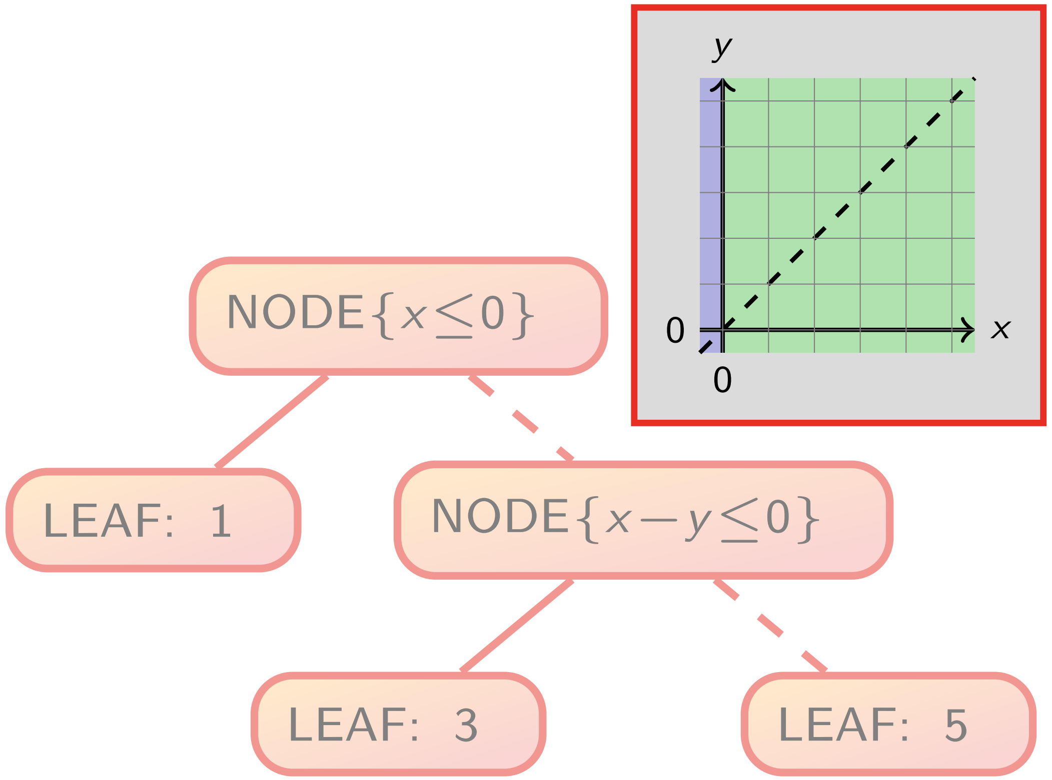


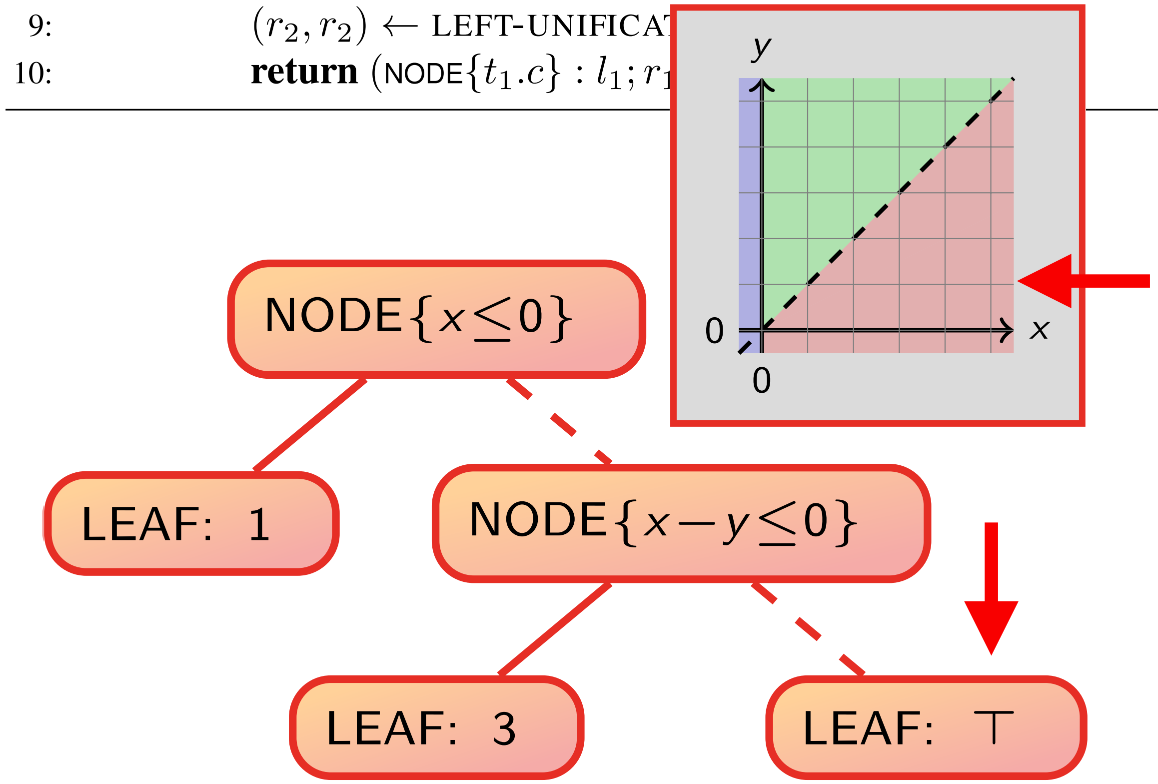


Widening: Domain Over-Approximation

Algorithm 5 : Tree Left Unification

```
function LEFT-UNIFICATION($t_1$,$t_2$)
    if ISLEAF($t_1$) $\wedge$ ISLEAF($t_2$) then
        return ($t_1$, $t_2$)
    else
        if ISLEAF($t_1$) $\vee$ (ISNODE($t_1$) $\wedge$ ISNODE($t_2$) $\wedge$ $t_2.c <_L t_1.c$) then
            return LEFT-UNIFICATION($t_1$, $t_2.l \sqcup_T t_2.r$)
        else
            $(l_1, l_2) \leftarrow$ LEFT-UNIFICATION($t_1.l$, $t_2.l$)
            $(r_2, r_2) \leftarrow$ LEFT-UNIFICAT
            return (NODE$\{t_1.c\}$ : $l_1; r_1$
```

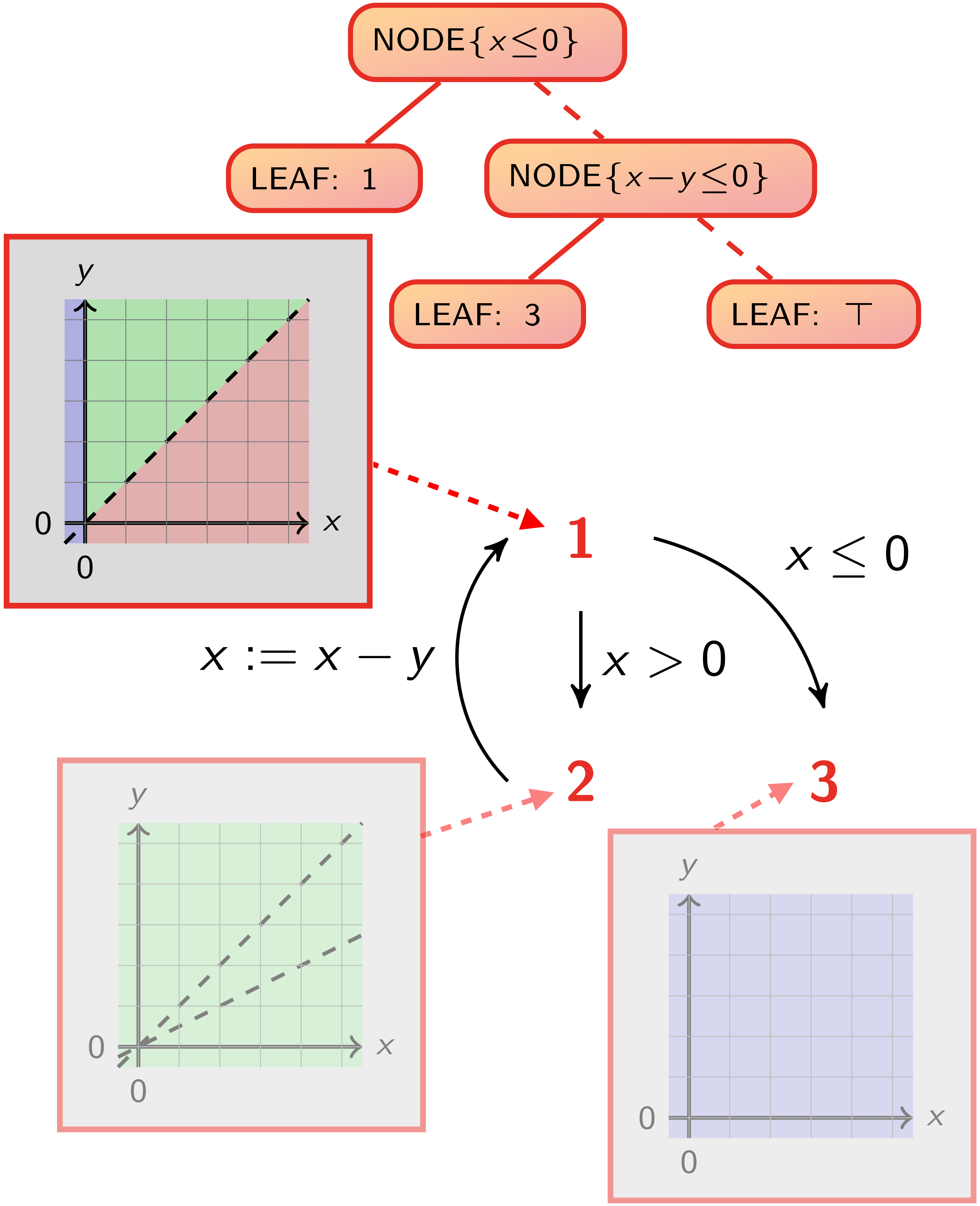




Example

```
int : $x$, $y$
while $^1(x > 0)$ do
    $^2x := x - y$
od$^3$
```

the widening is **sound**!



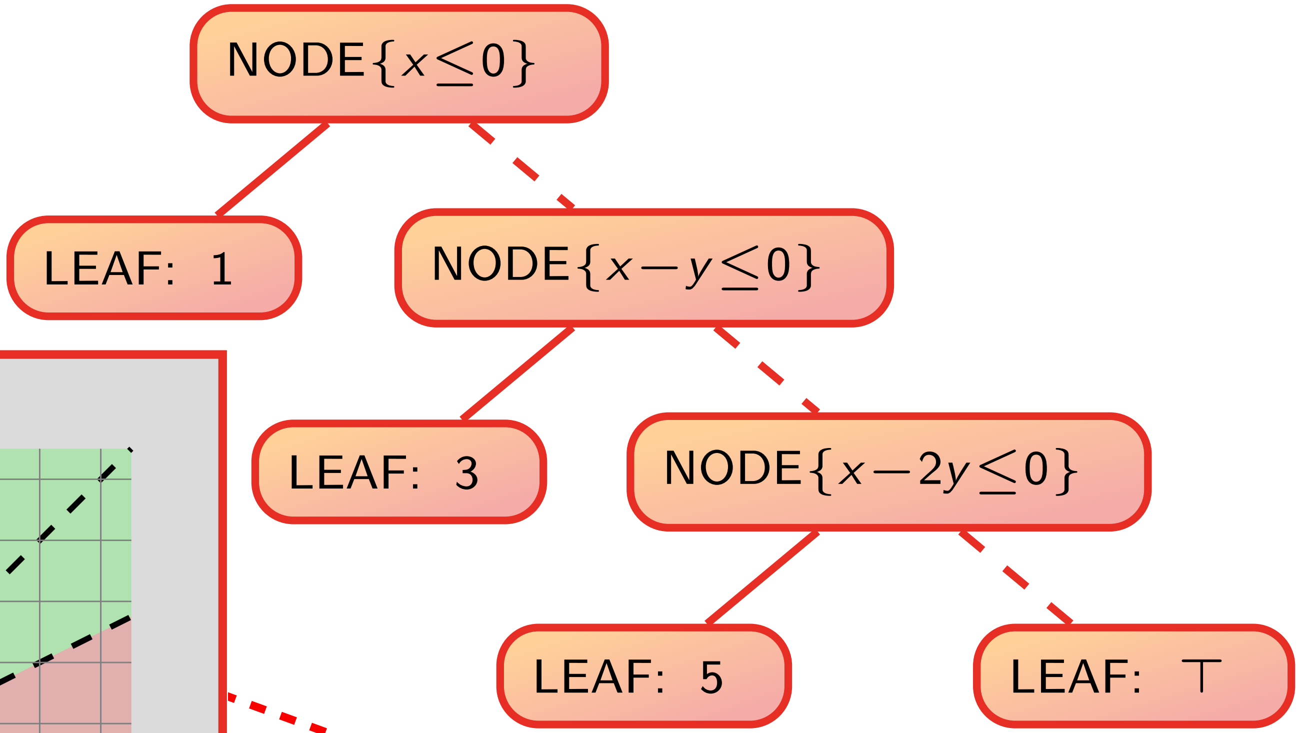
Example

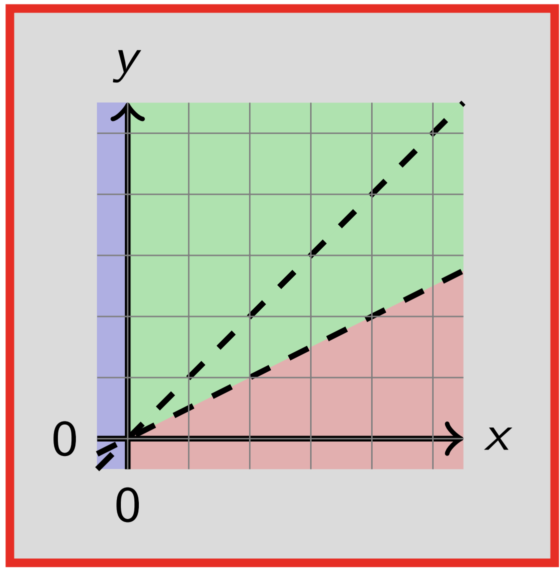
int : x, y

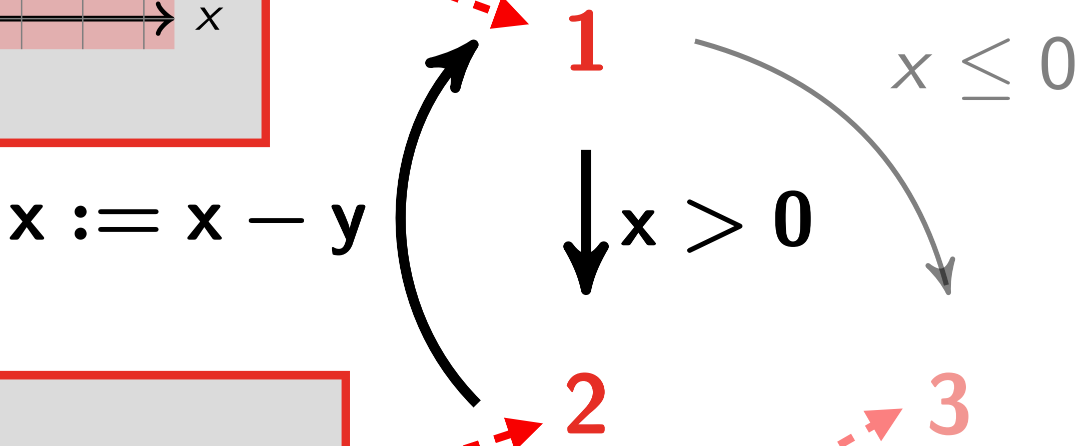
while ${}^{1}(x > 0)$ do

${}^{2}x := x - y$

od^{3}


NODE$\{x \leq 0\}$
LEAF: 1
NODE$\{x - y \leq 0\}$
LEAF: 3
NODE$\{x - 2y \leq 0\}$
LEAF: 5
LEAF: $\top$




1

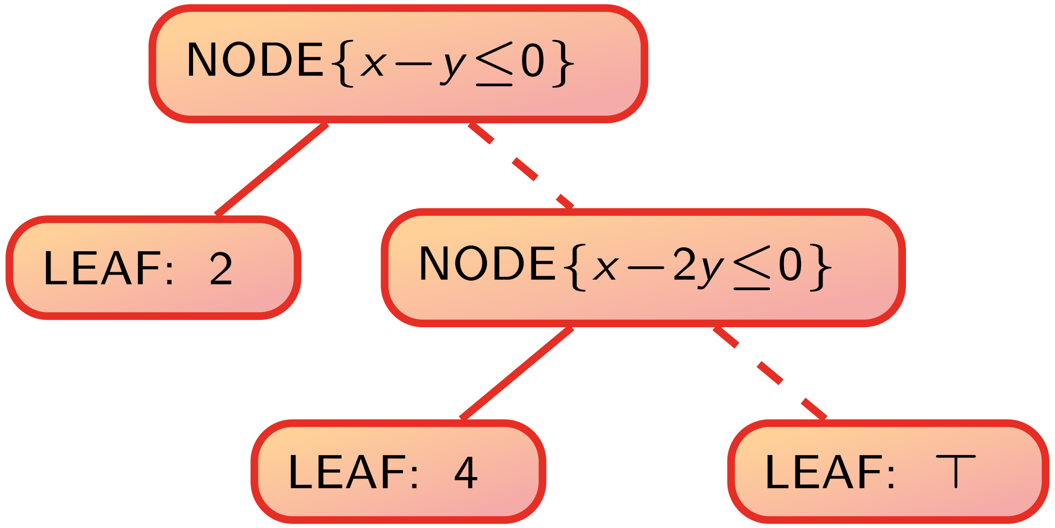
$x \leq 0$

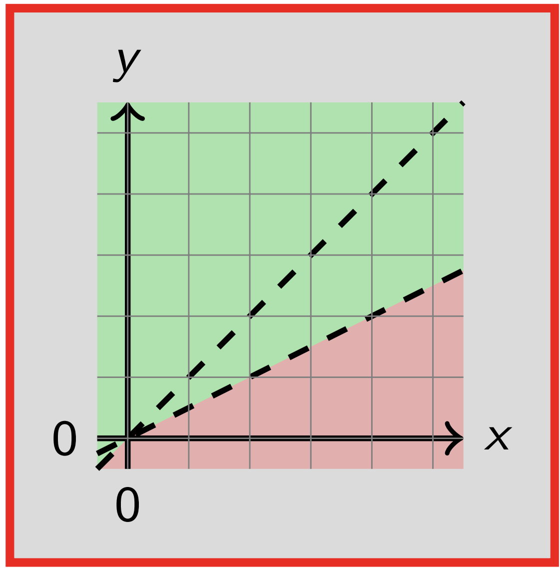
$\mathbf{x > 0}$

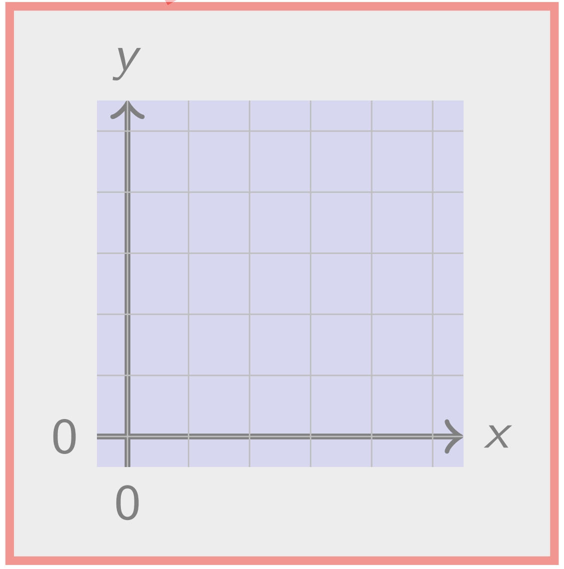
$\mathbf{x := x - y}$

2

3


NODE$\{x - y \leq 0\}$
LEAF: 2
NODE$\{x - 2y \leq 0\}$
LEAF: 4
LEAF: $\top$





Example

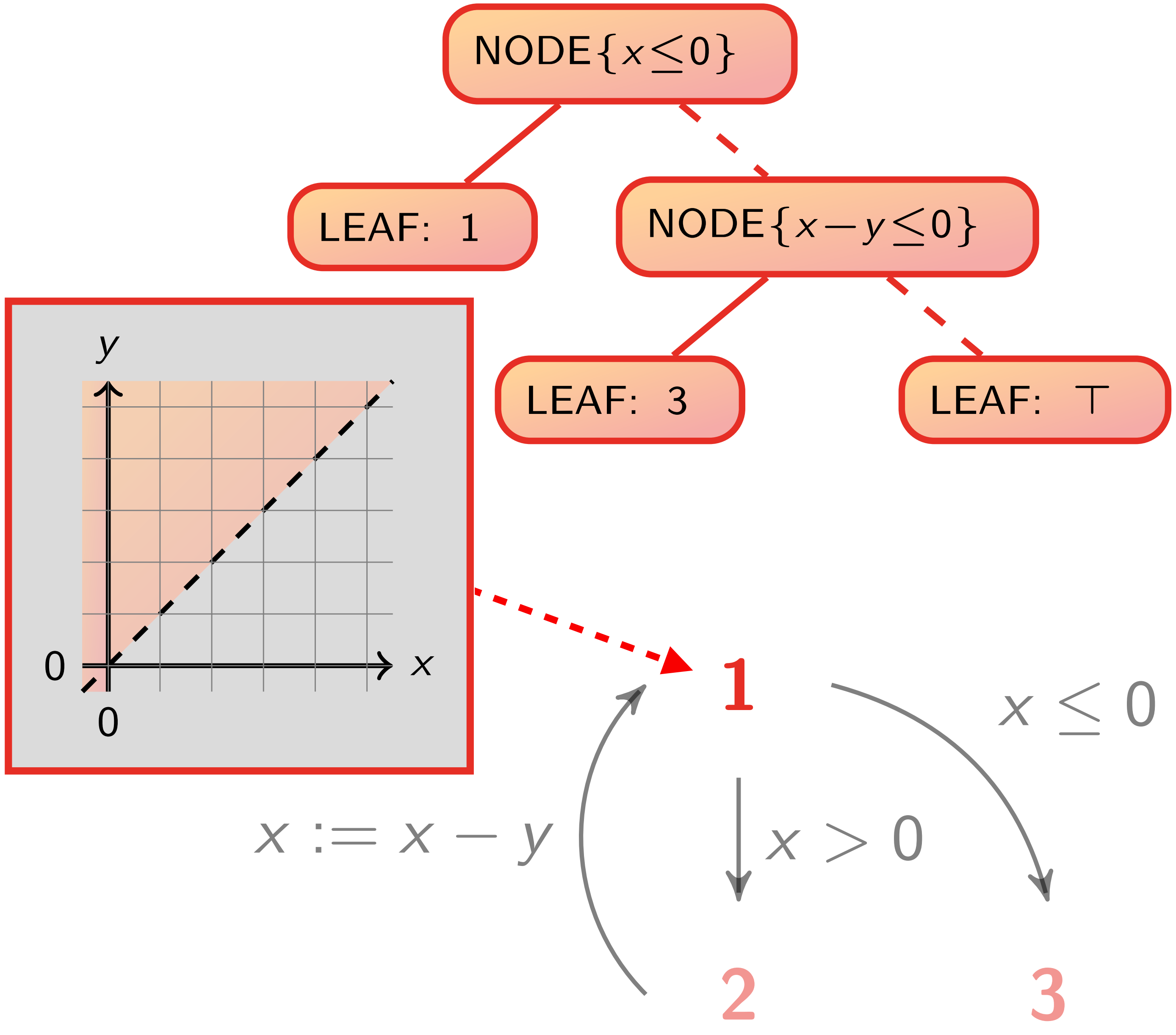
int : x, y

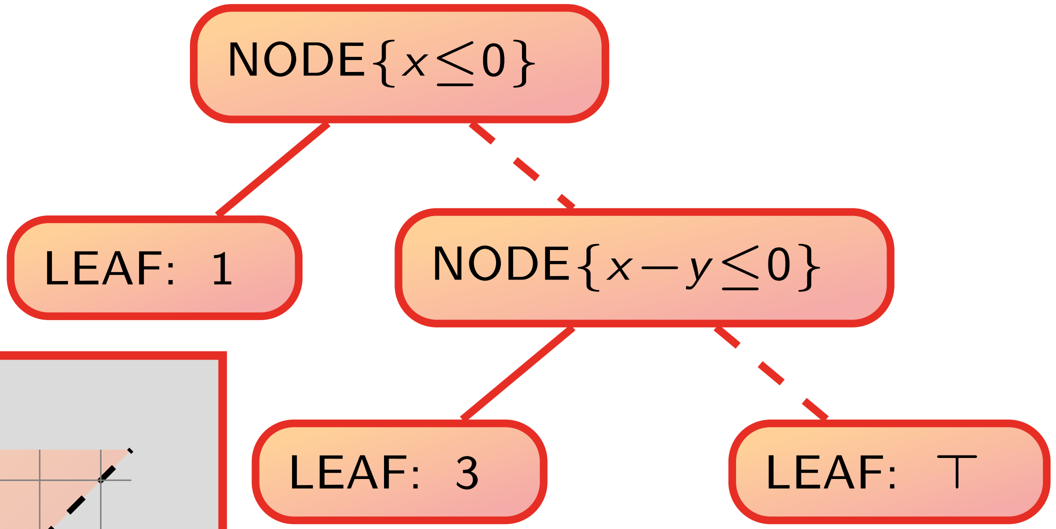
while $^{1}(x > 0)$ do

$^{2}x := x - y$

od^{3}

the analysis gives $x \leq 0 \vee x \leq y$ as **sufficient precondition**



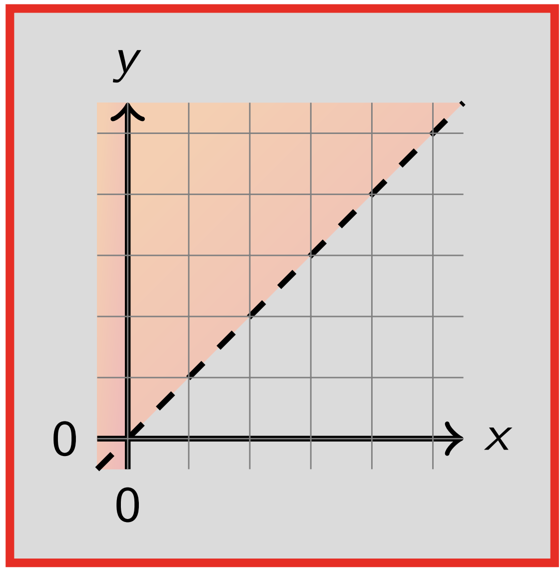


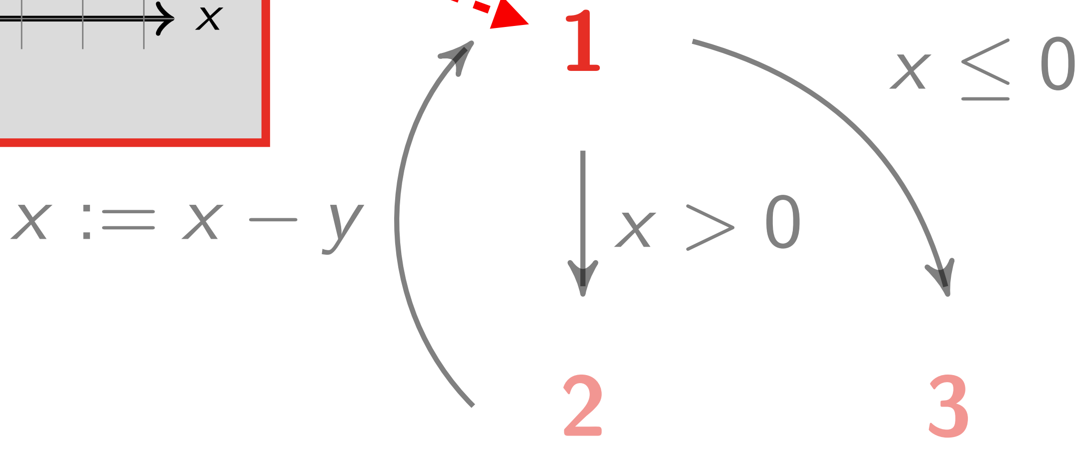
Example

```
int : $x, y$
while $^1(x > 0)$ do
    $^2x := x - y$
od$^3$
```

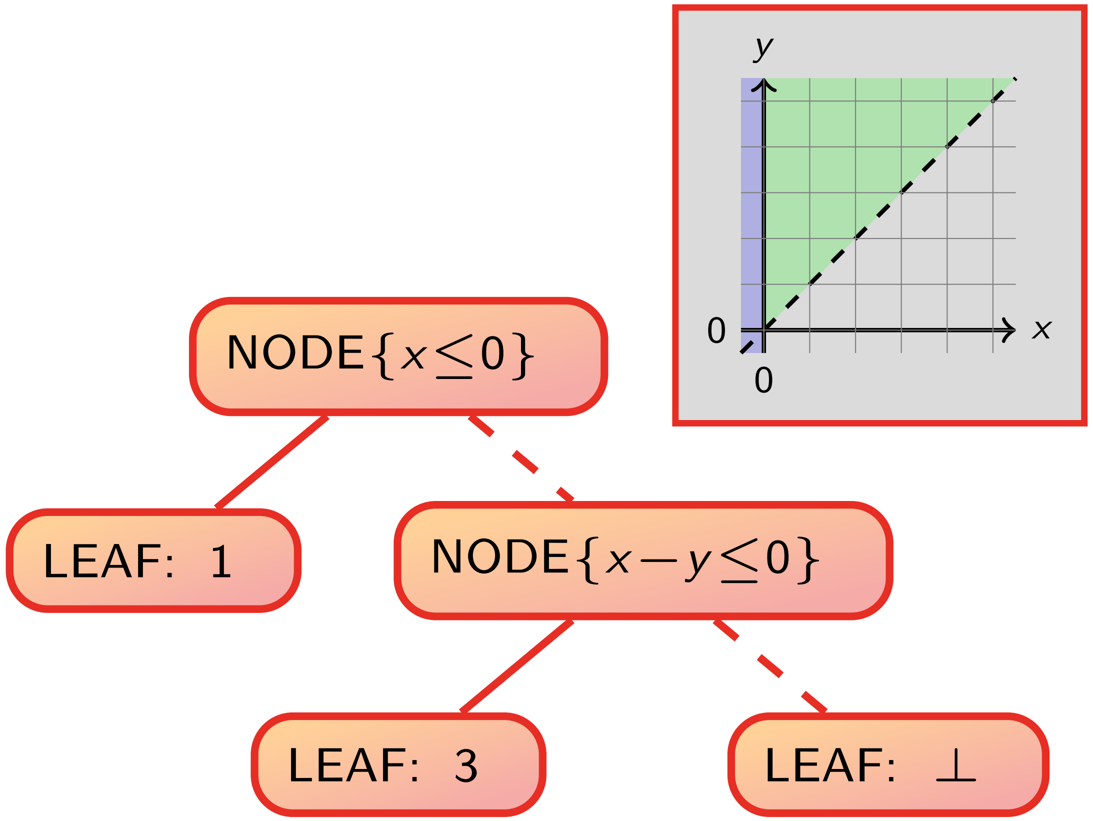
the analysis gives $x \leq 0 \vee x \leq y$ as **sufficient precondition**

the **weakest precondition** is $\mathbf{x \leq 0 \vee y > 0}$





Widening



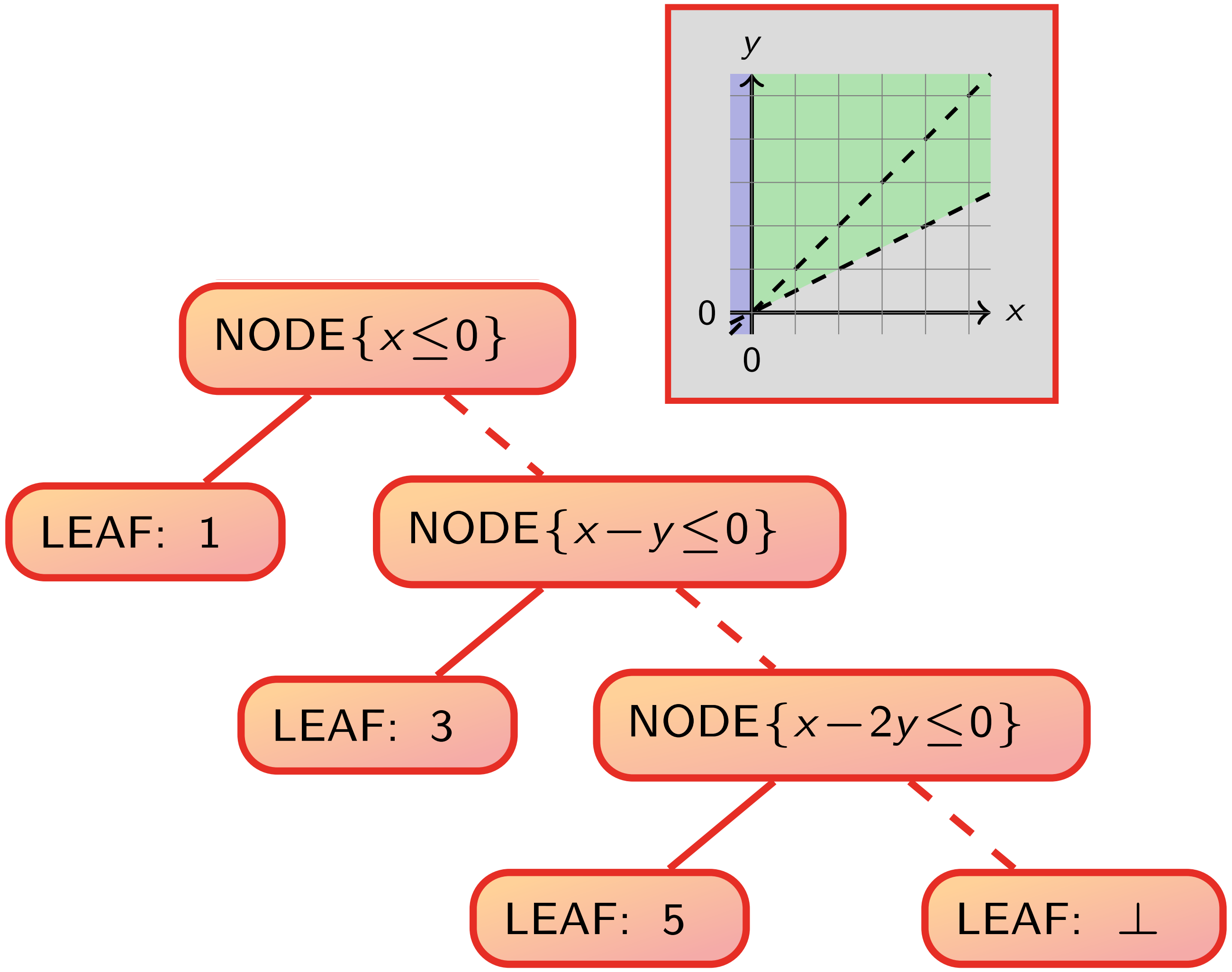
NODE$\{x \leq 0\}$

LEAF: 1

NODE$\{x - y \leq 0\}$

LEAF: 3

LEAF: $\perp$



NODE$\{x \leq 0\}$

LEAF: 1

NODE$\{x - y \leq 0\}$

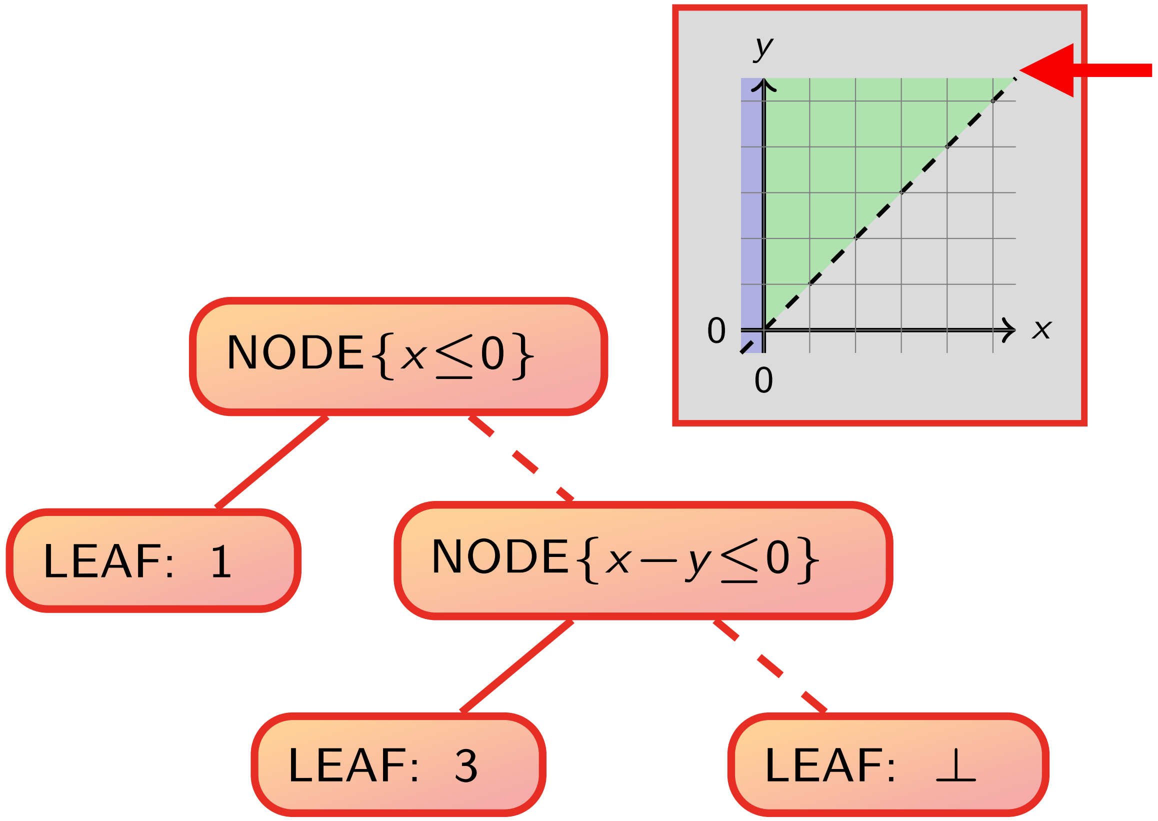
LEAF: 3

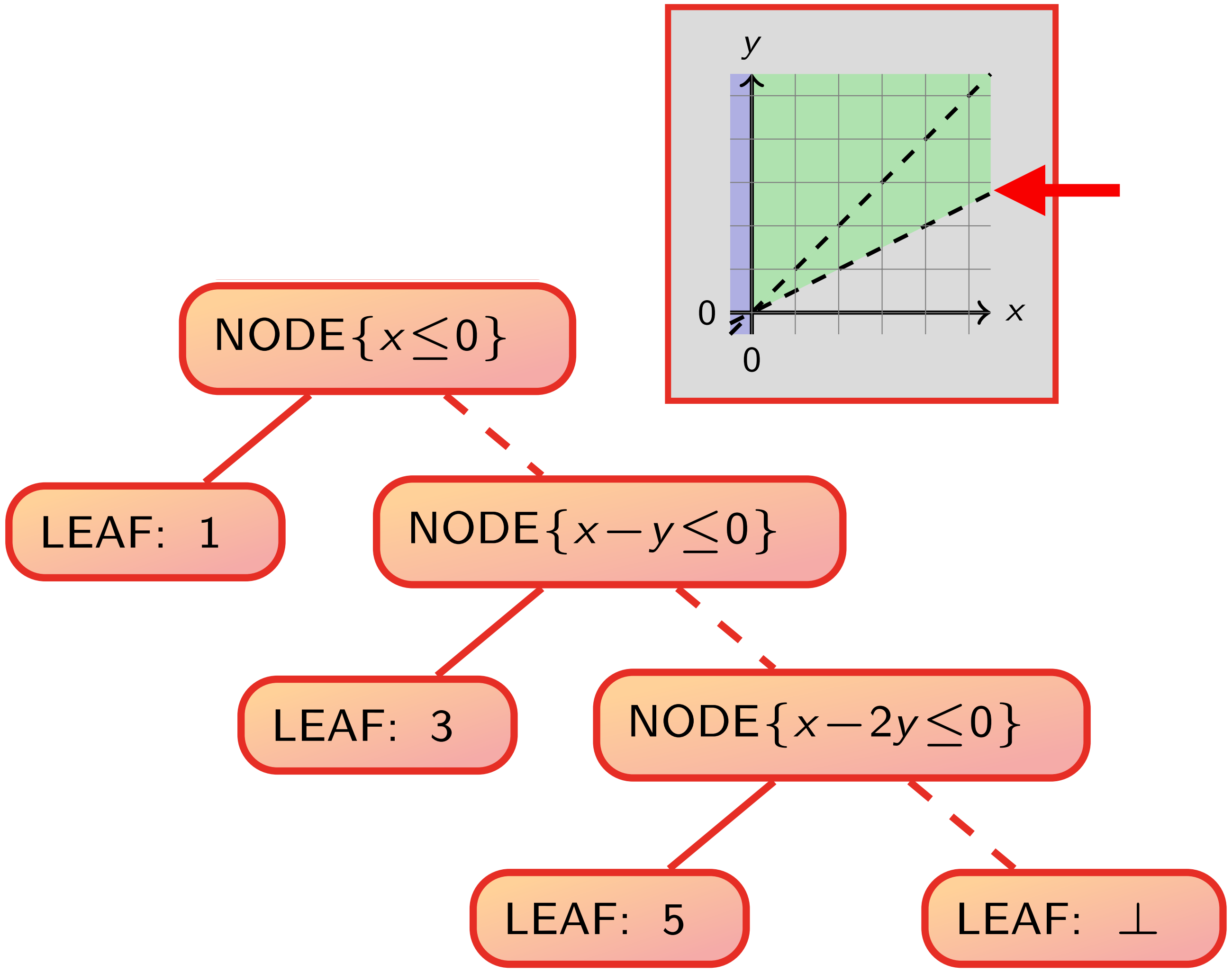
NODE$\{x - 2y \leq 0\}$

LEAF: 5

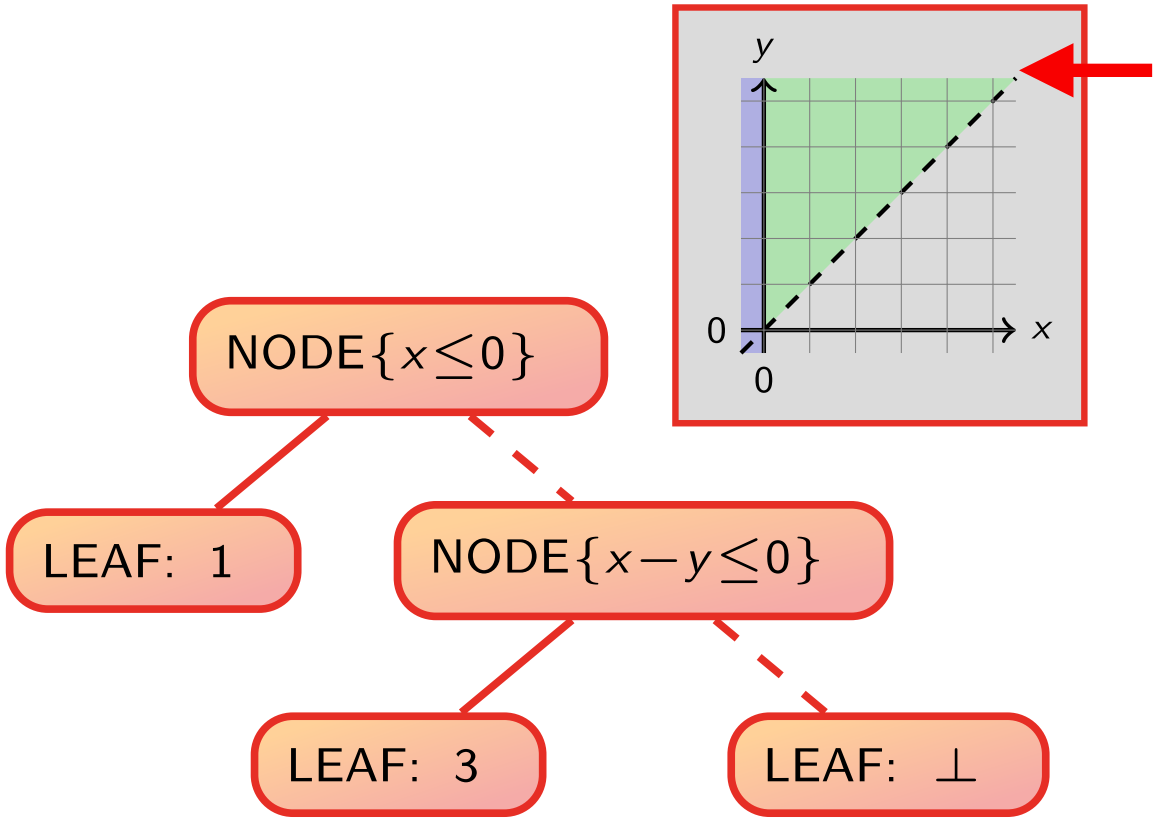
LEAF: $\perp$

Widening


y
0
0
x
NODE{$x \leq 0$}
LEAF: 1
NODE{$x - y \leq 0$}
LEAF: 3
LEAF: $\perp$


y
0
0
x
NODE{$x \leq 0$}
LEAF: 1
NODE{$x - y \leq 0$}
LEAF: 3
NODE{$x - 2y \leq 0$}
LEAF: 5
LEAF: $\perp$

Widening



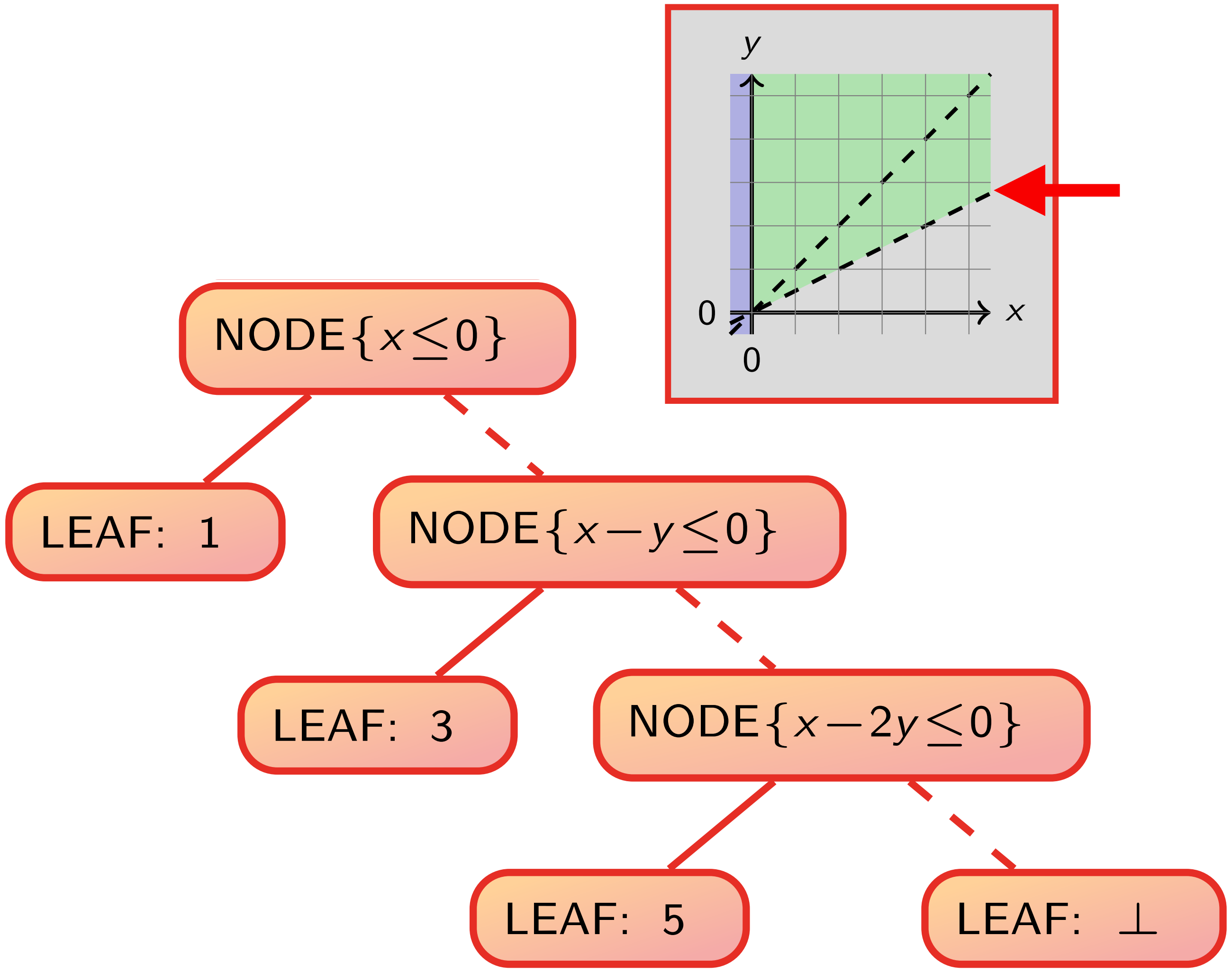
NODE$\{x \leq 0\}$

LEAF: 1

NODE$\{x - y \leq 0\}$

LEAF: 3

LEAF: $\bot$



NODE$\{x \leq 0\}$

LEAF: 1

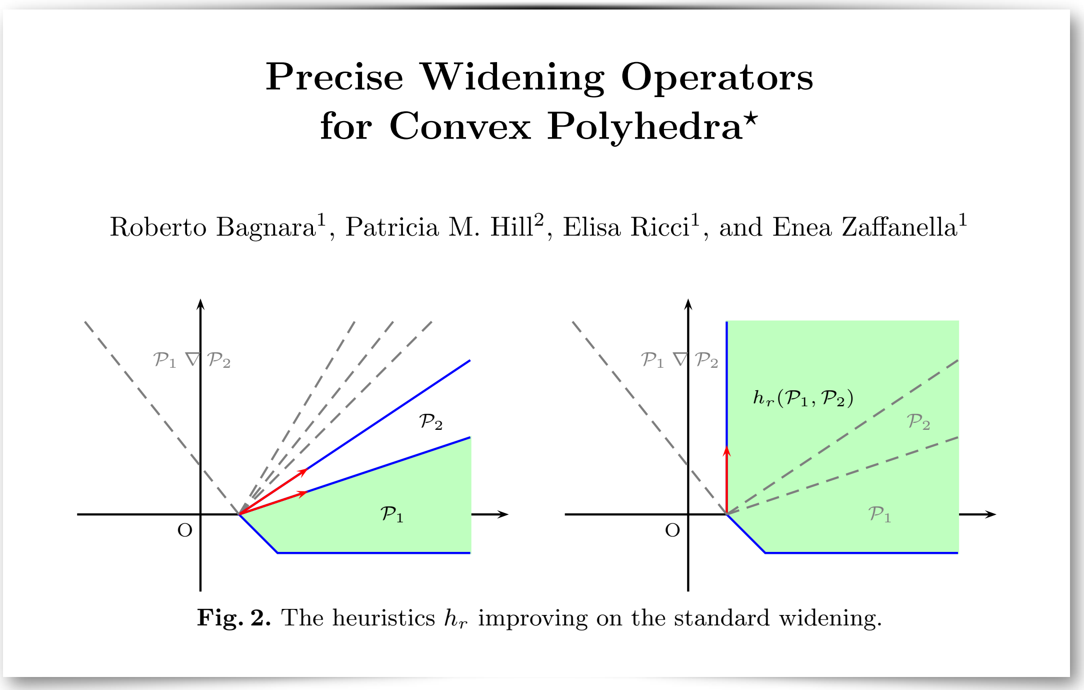
NODE$\{x - y \leq 0\}$

LEAF: 3

NODE$\{x - 2y \leq 0\}$

LEAF: 5

LEAF: $\bot$

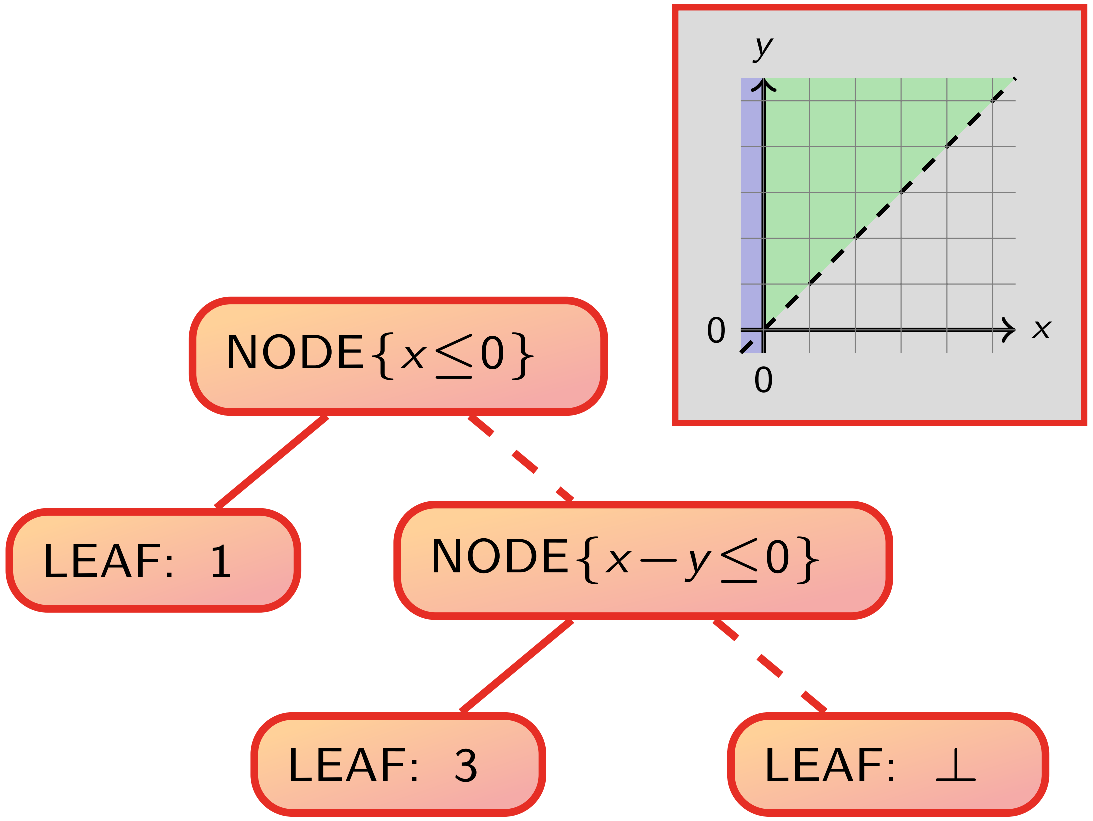


Precise Widening Operators
for Convex Polyhedra*

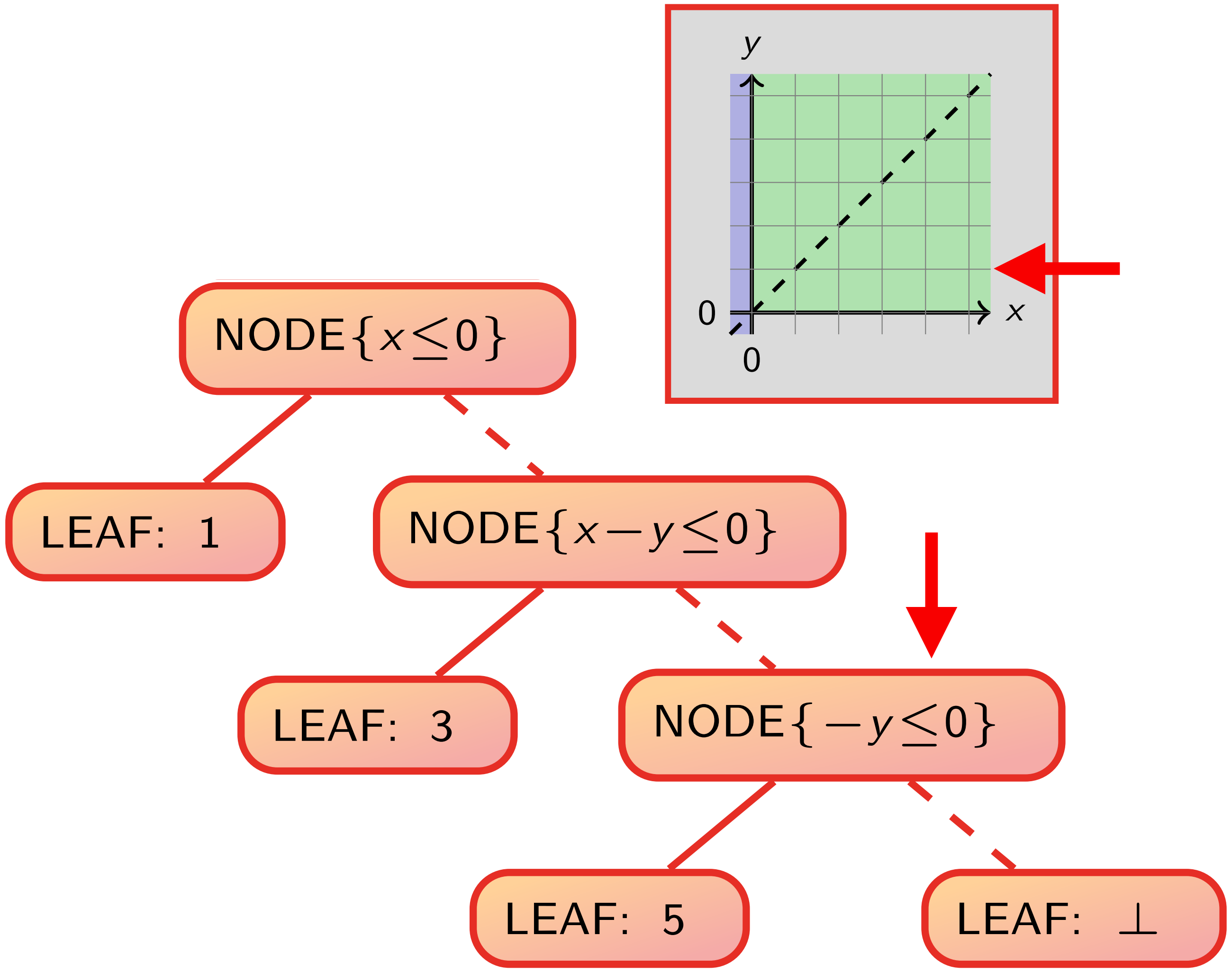
Roberto Bagnara[1], Patricia M. Hill[2], Elisa Ricci[1], and Enea Zaffanella[1]

Fig. 2. The heuristics h_r improving on the standard widening.

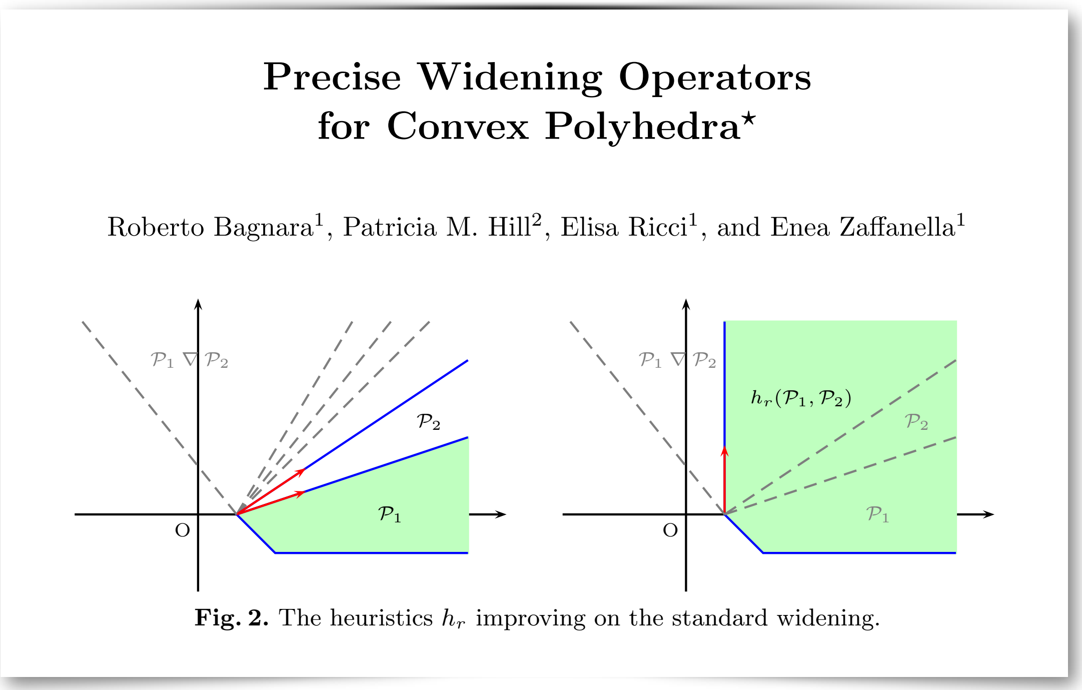
Widening



NODE$\{x \leq 0\}$
LEAF: 1
NODE$\{x - y \leq 0\}$
LEAF: 3
LEAF: ⊥



NODE$\{x \leq 0\}$
LEAF: 1
NODE$\{x - y \leq 0\}$
LEAF: 3
NODE$\{-y \leq 0\}$
LEAF: 5
LEAF: ⊥

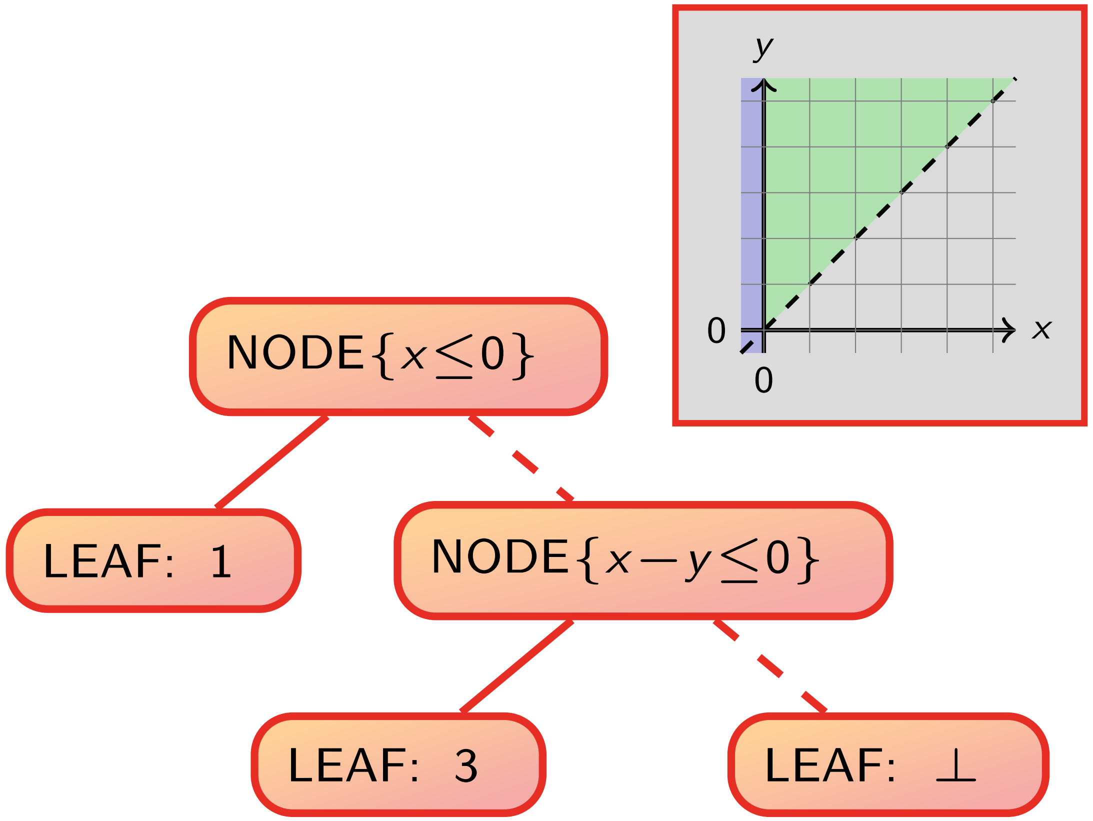


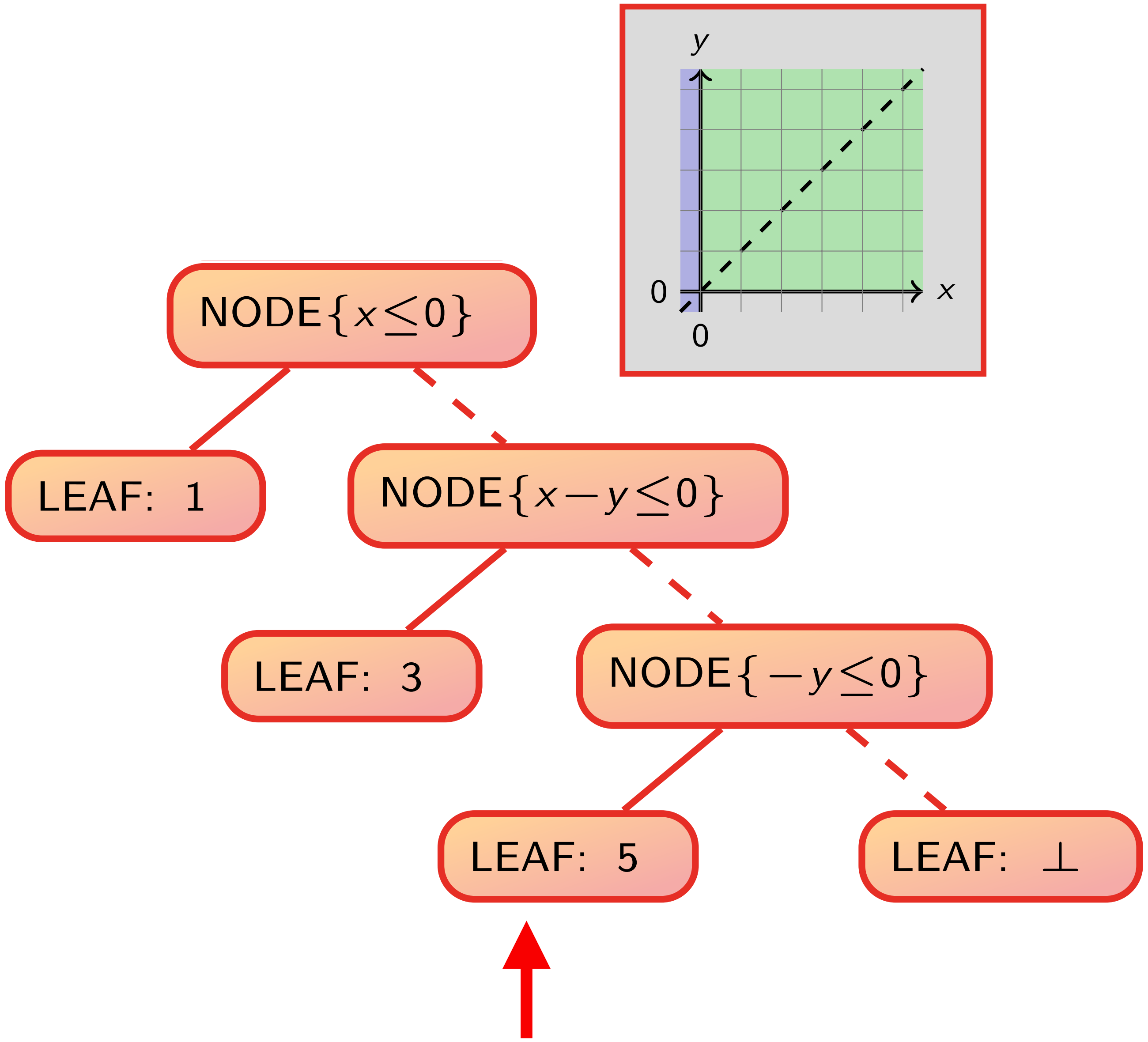
Precise Widening Operators for Convex Polyhedra*

Roberto Bagnara[1], Patricia M. Hill[2], Elisa Ricci[1], and Enea Zaffanella[1]

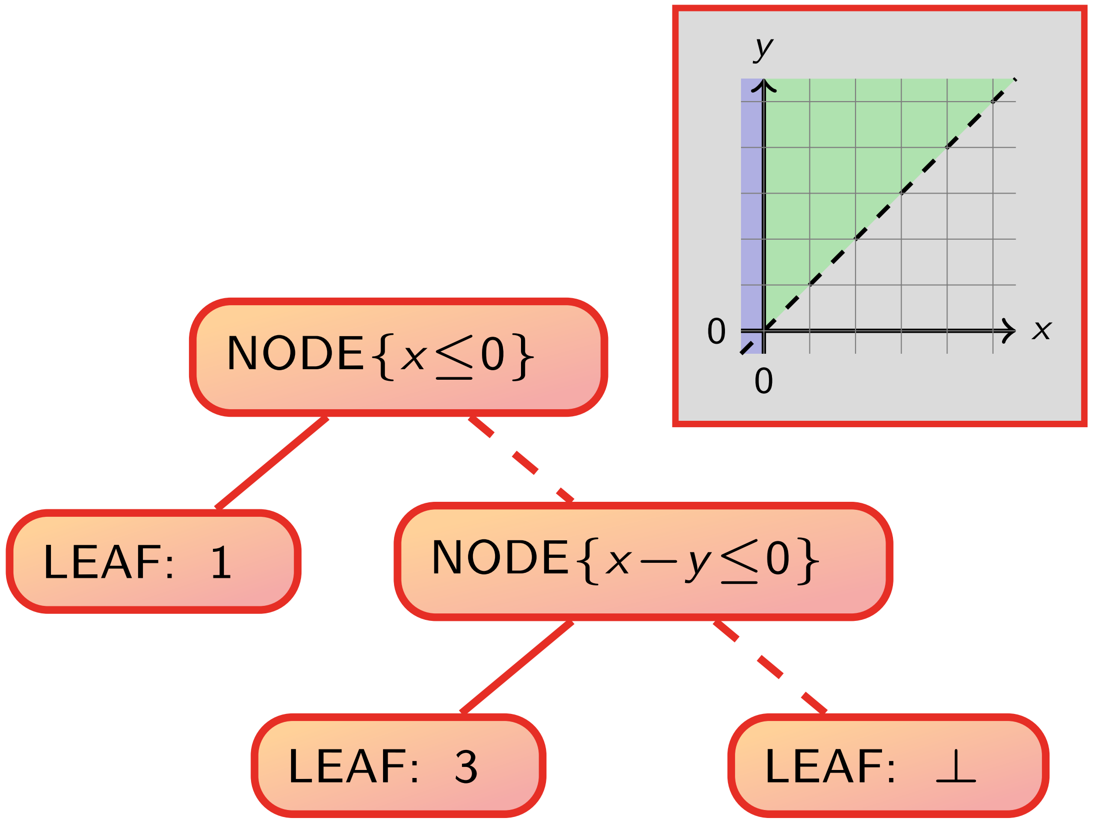
Fig. 2. The heuristics h_r improving on the standard widening.

Widening


NODE{$x \leq 0$}
LEAF: 1
NODE{$x - y \leq 0$}
LEAF: 3
LEAF: $\bot$
y
x
0
0


NODE{$x \leq 0$}
LEAF: 1
NODE{$x - y \leq 0$}
LEAF: 3
NODE{$-y \leq 0$}
LEAF: 5
LEAF: $\bot$
y
x
0
0

Widening



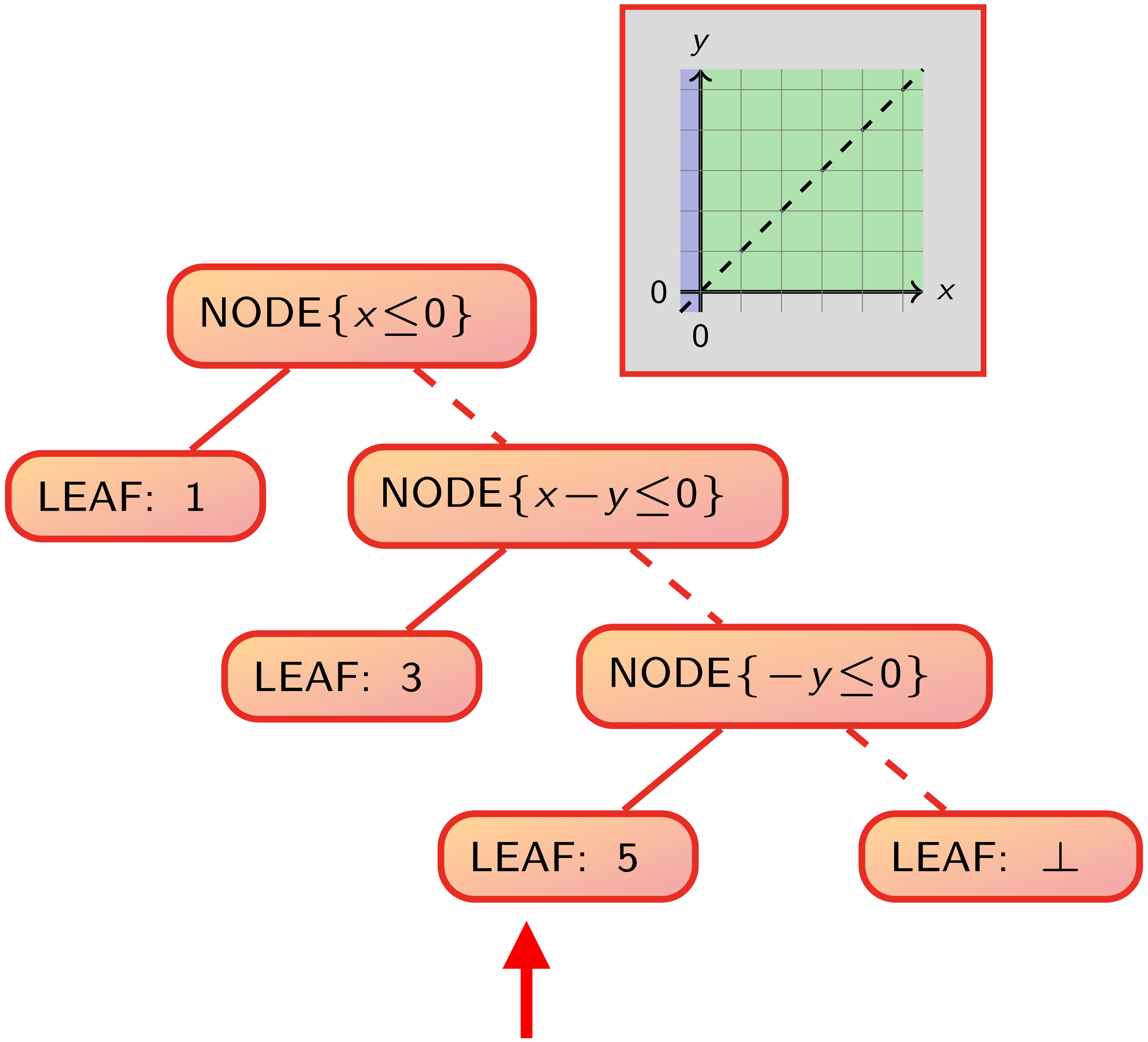
NODE$\{x \leq 0\}$

LEAF: 1

NODE$\{x - y \leq 0\}$

LEAF: 3

LEAF: $\bot$



NODE$\{x \leq 0\}$

LEAF: 1

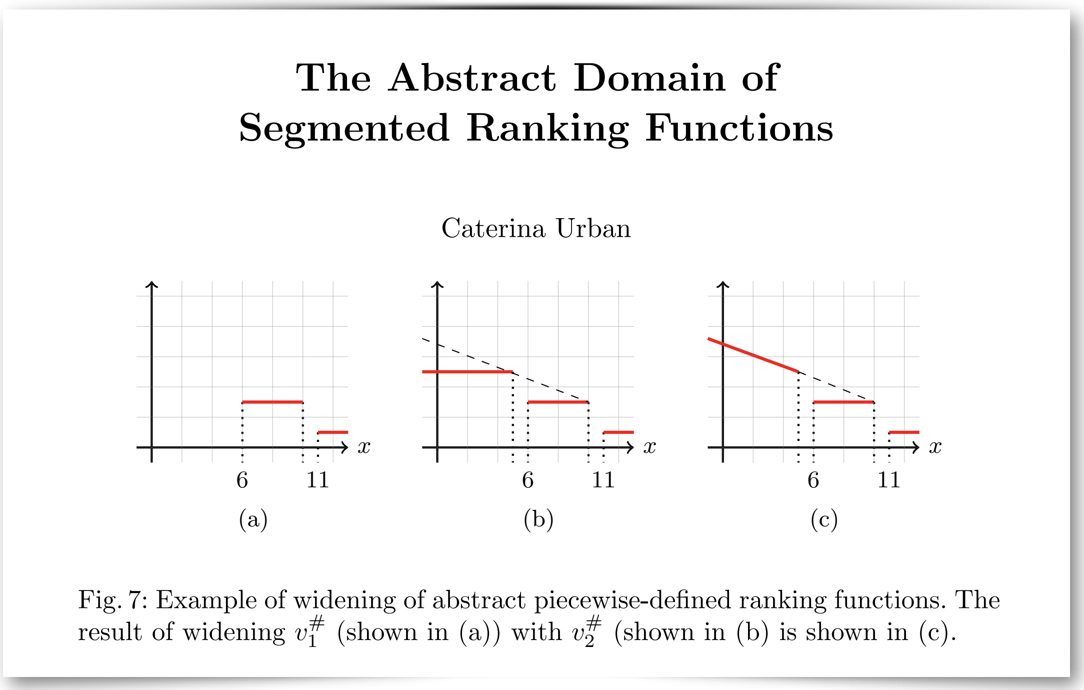
NODE$\{x - y \leq 0\}$

LEAF: 3

NODE$\{-y \leq 0\}$

LEAF: 5

LEAF: $\bot$



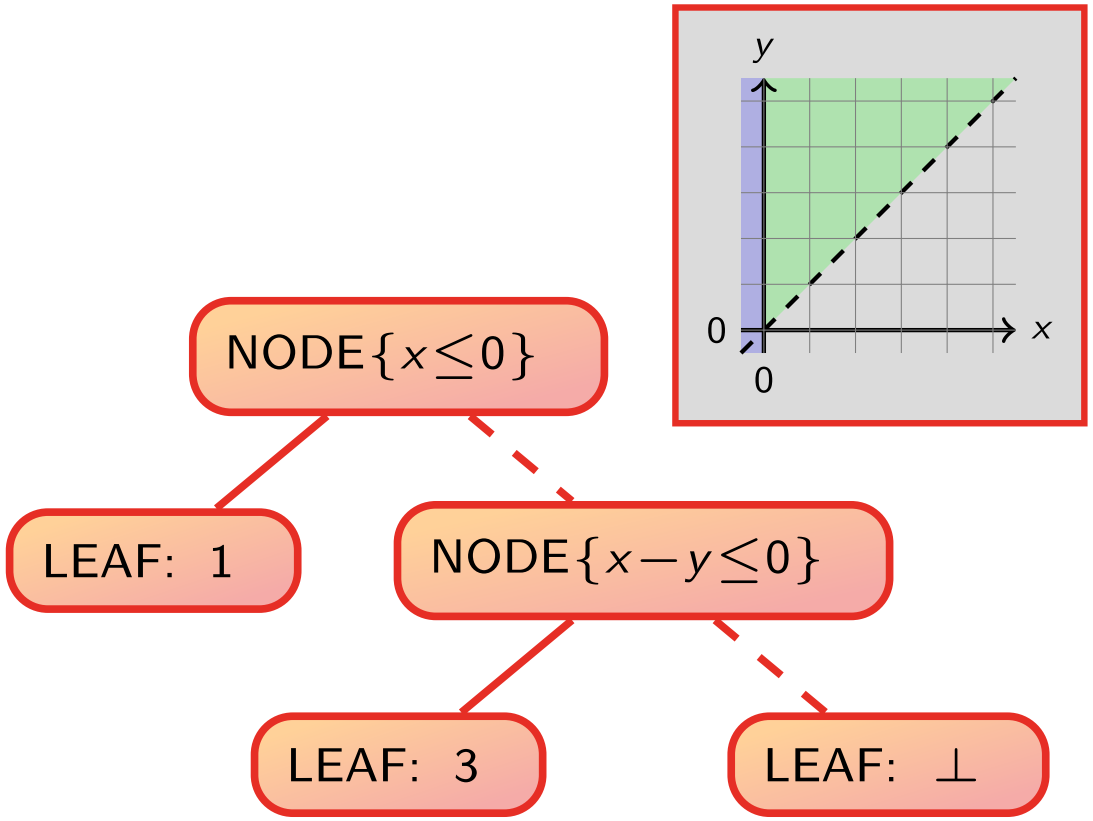
The Abstract Domain of Segmented Ranking Functions

Caterina Urban

(a) (b) (c)

Fig. 7: Example of widening of abstract piecewise-defined ranking functions. The result of widening $v_1^\#$ (shown in (a)) with $v_2^\#$ (shown in (b) is shown in (c).

Widening



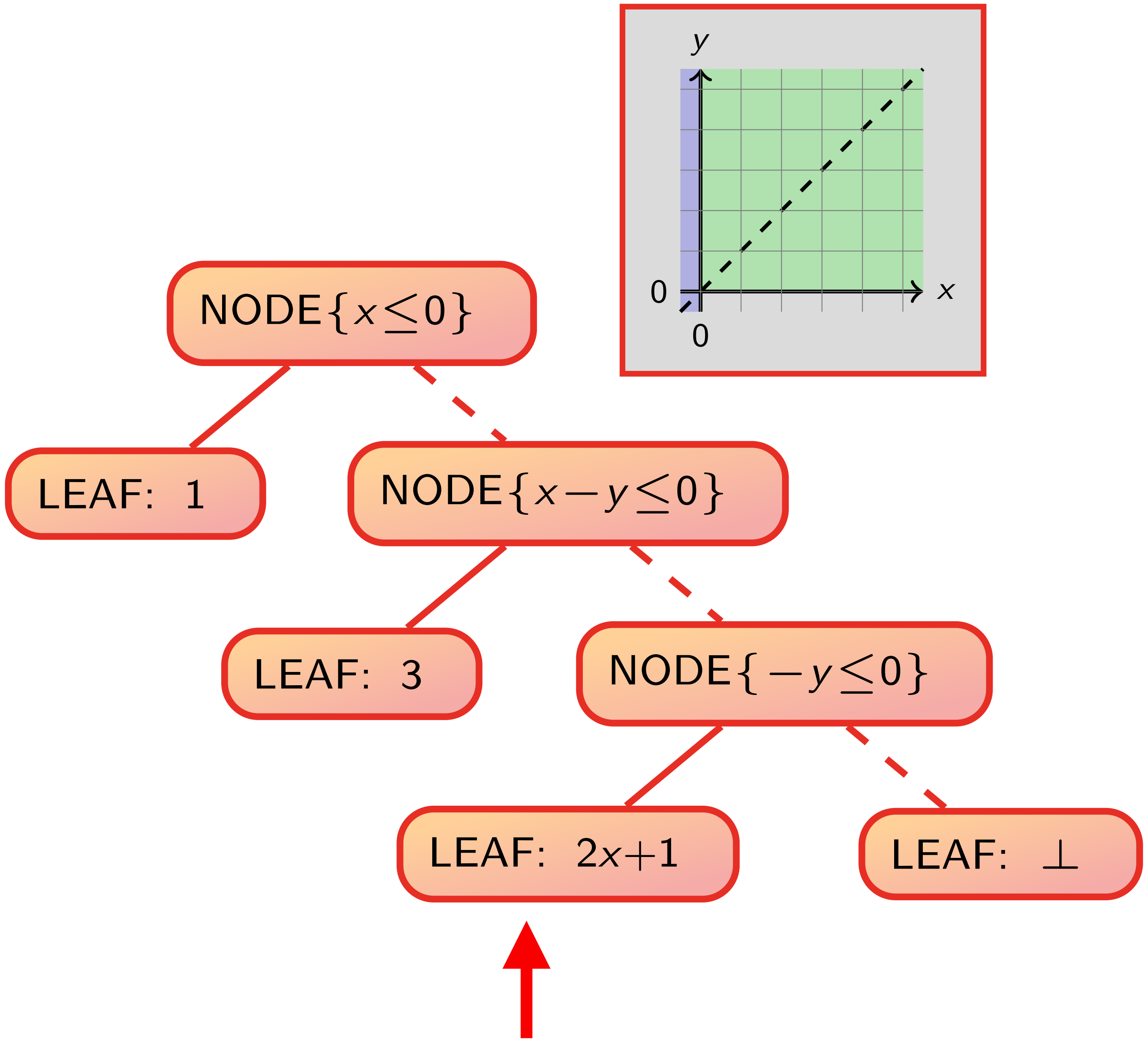
NODE$\{x \leq 0\}$

LEAF: 1

NODE$\{x - y \leq 0\}$

LEAF: 3

LEAF: $\bot$



NODE$\{x \leq 0\}$

LEAF: 1

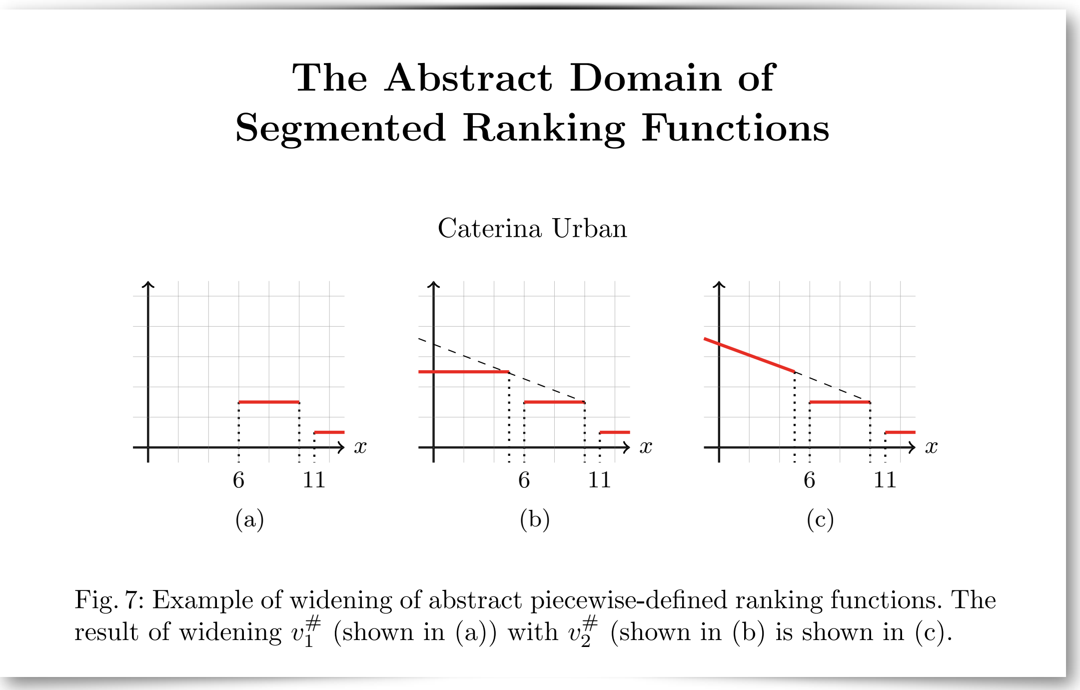
NODE$\{x - y \leq 0\}$

LEAF: 3

NODE$\{-y \leq 0\}$

LEAF: $2x+1$

LEAF: $\bot$



The Abstract Domain of
Segmented Ranking Functions

Caterina Urban

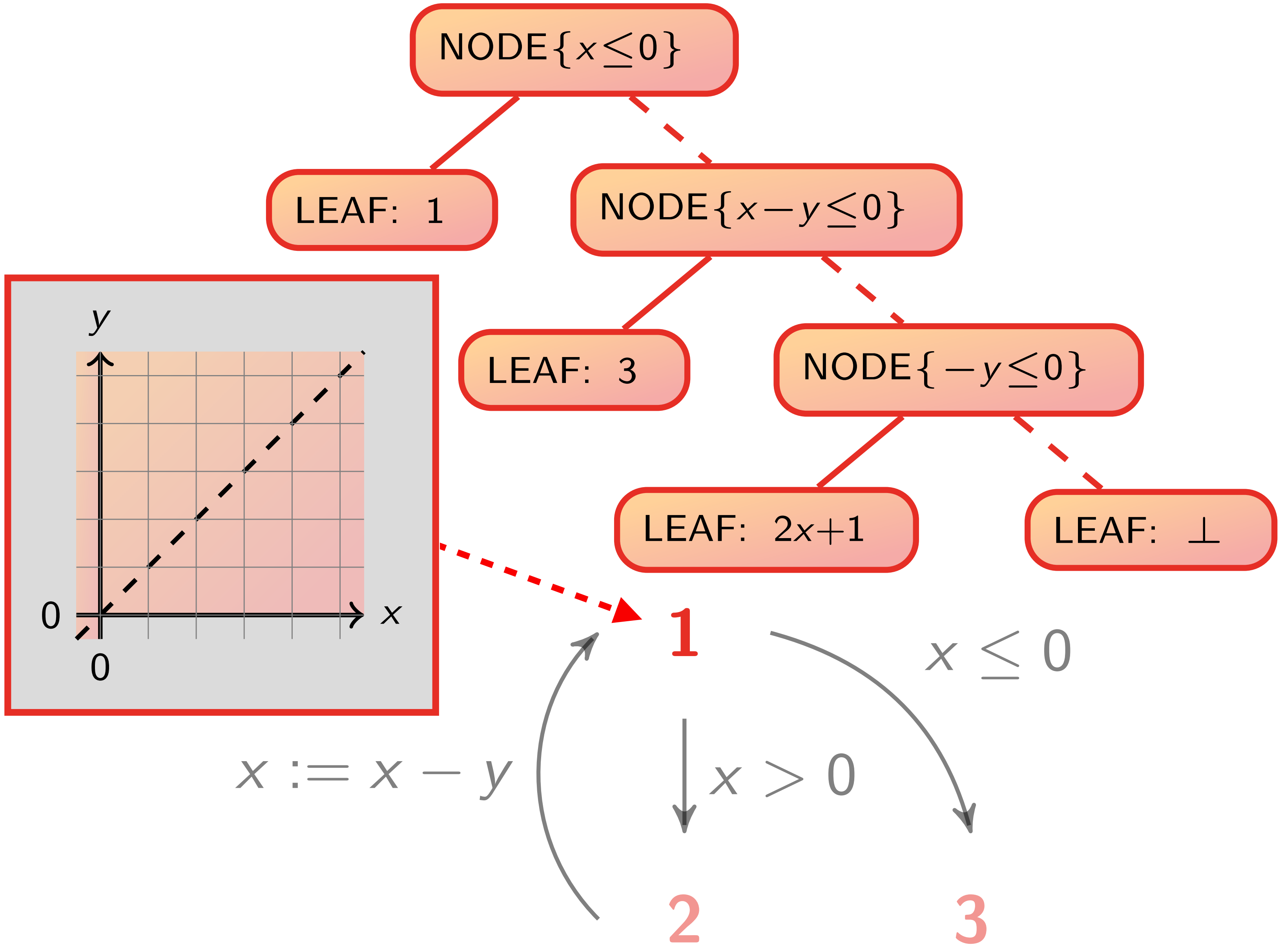
(a) (b) (c)

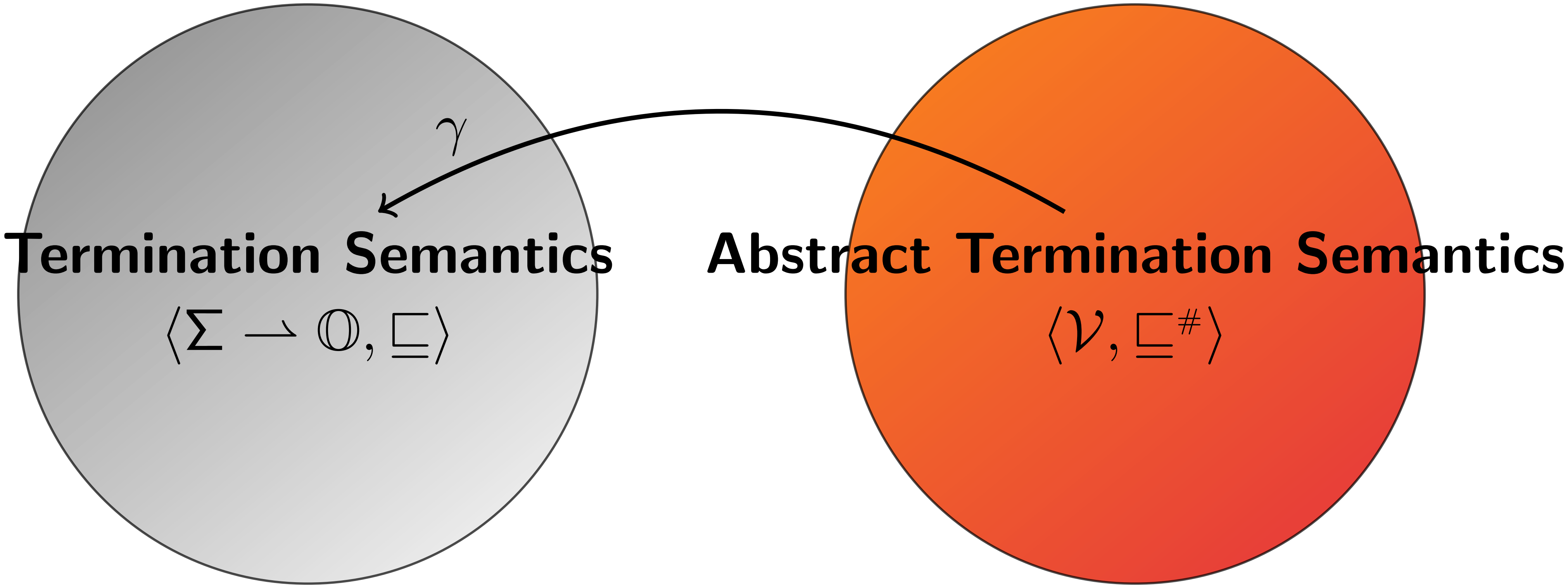
Fig. 7: Example of widening of abstract piecewise-defined ranking functions. The result of widening $v_1^\#$ (shown in (a)) with $v_2^\#$ (shown in (b) is shown in (c).

Example

```
int : $x, y$
while $^1(x > 0)$ do
    $^2x := x - y$
od$^3$
```

the analysis gives the **weakest precondition** $x \leq 0 \vee y > 0$





Termination Semantics

$\langle \Sigma \rightharpoonup \mathbb{O}, \sqsubseteq \rangle$

Abstract Termination Semantics

$\langle \mathcal{V}, \sqsubseteq^{\#} \rangle$

Theorem (**Soundness**)

the abstract termination semantics is **sound**
to prove the termination of programs

- **remark**: natural-valued ranking functions are **not sufficient**!

Example

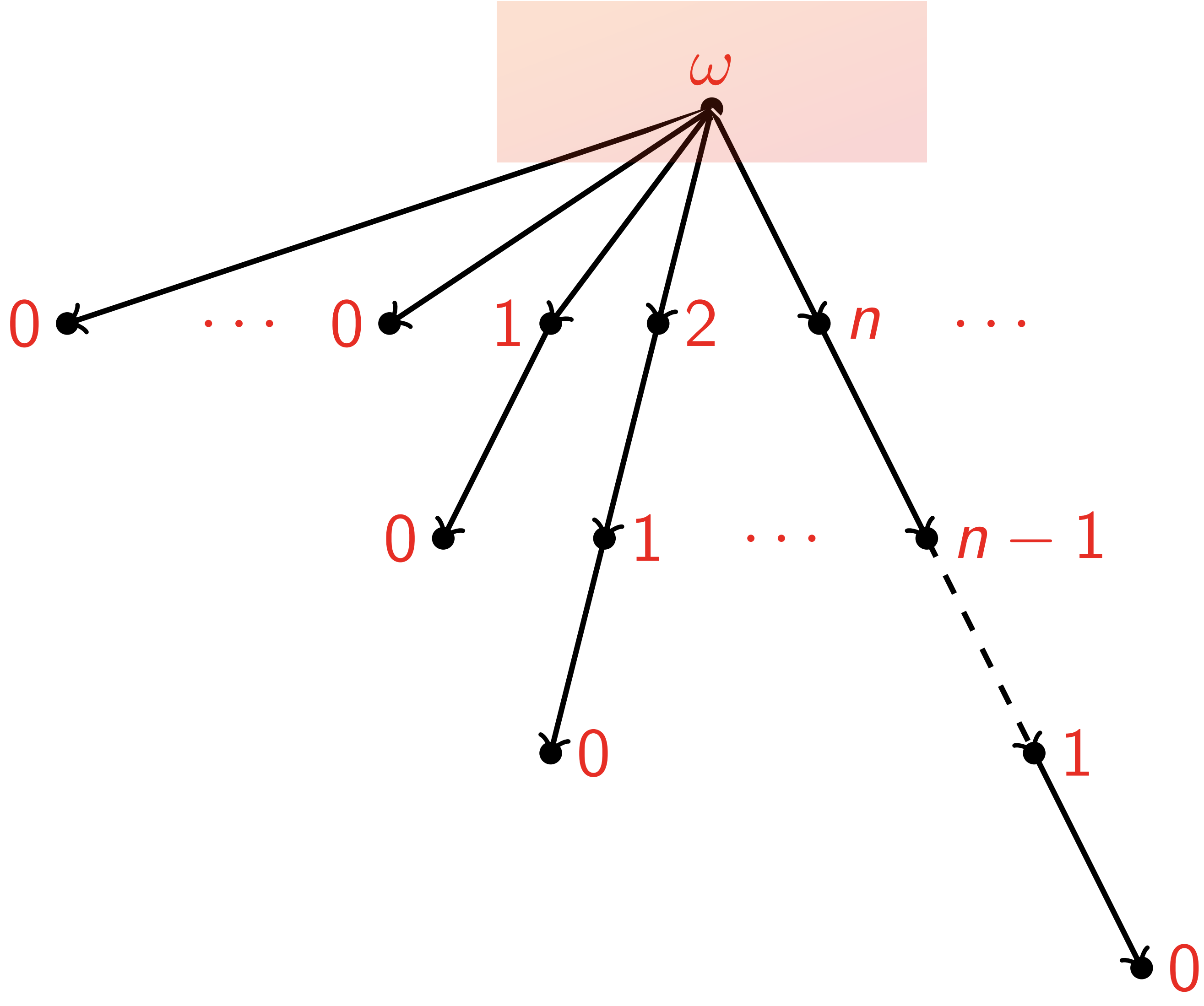
int : x

$x :=\ ?$

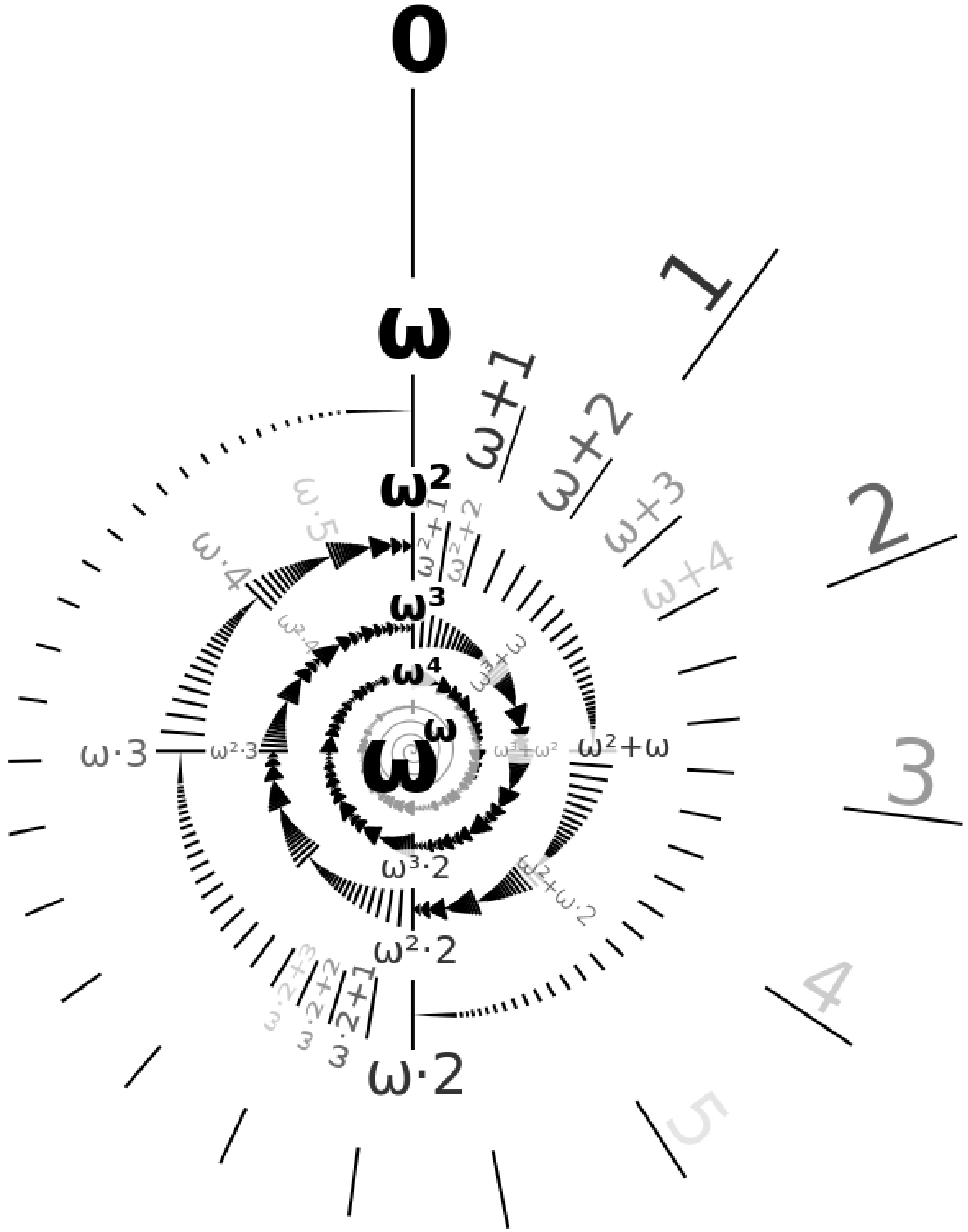
while $(x > 0)$ do

  $x := x - 1$

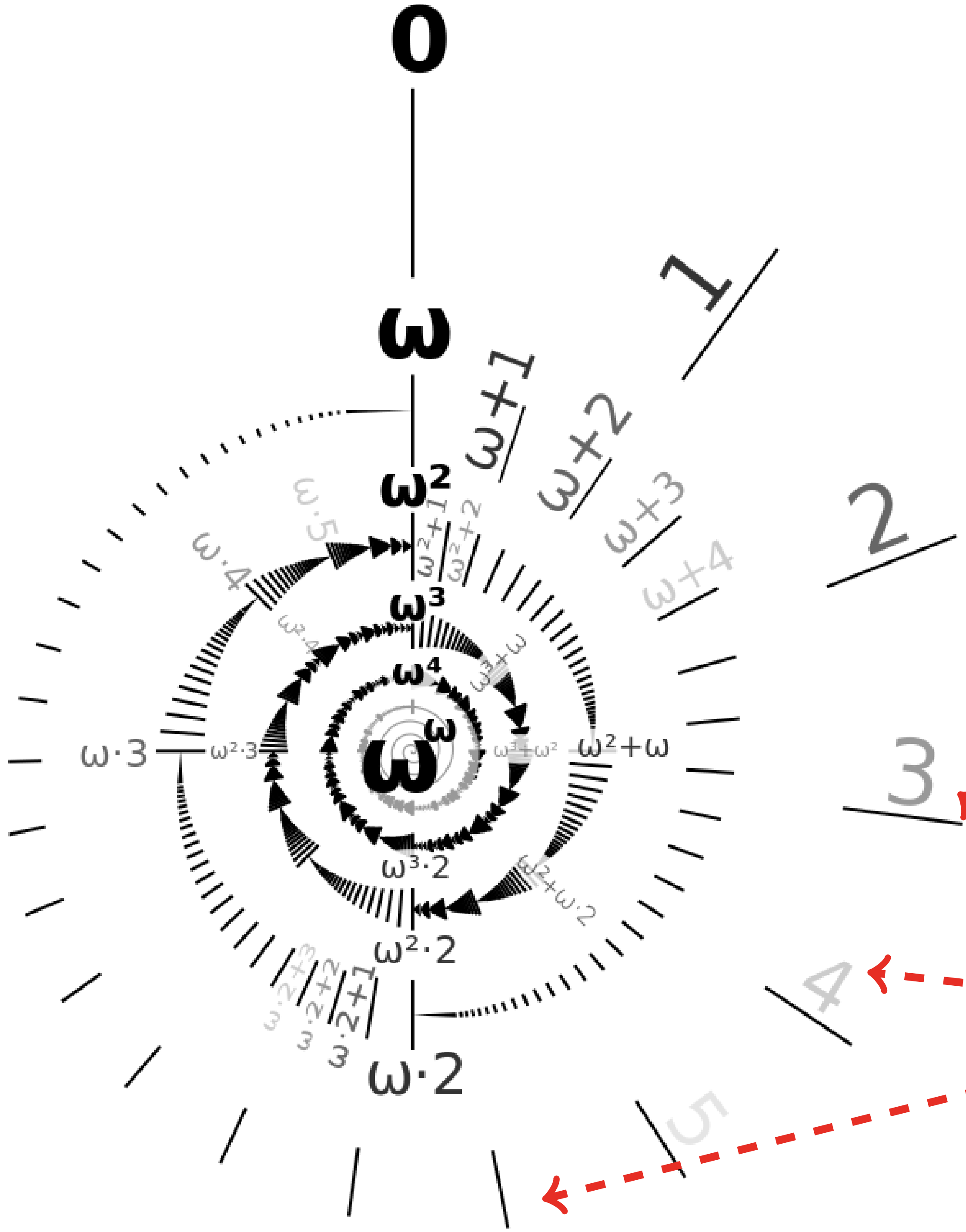
od



Ordinals

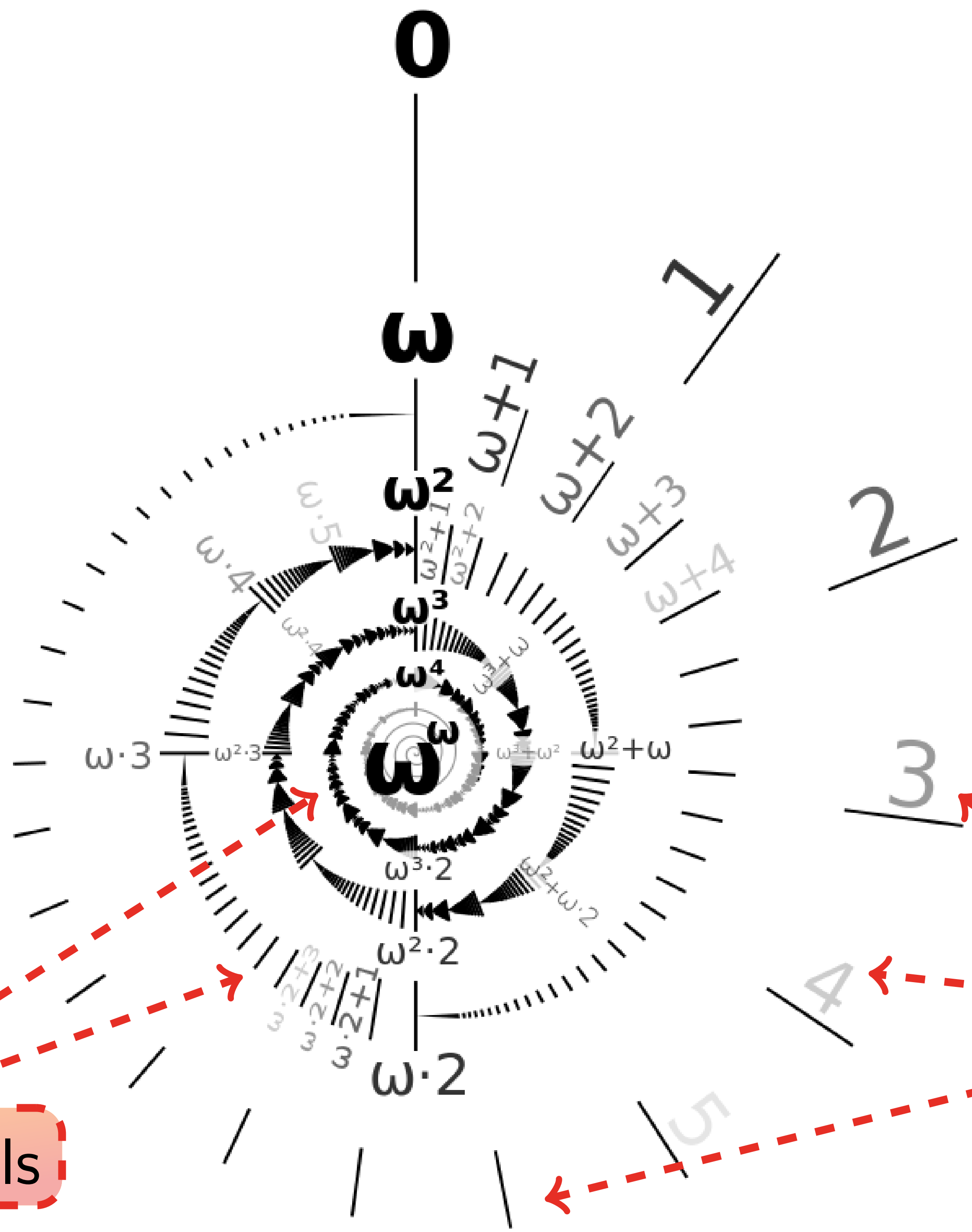

0
ω
1
$\omega+1$
$\omega+2$
$\omega+3$
$\omega+4$
2
ω^2
ω^2+1
ω^2+2
$\omega\cdot 5$
$\omega\cdot 4$
ω^3
$\omega^3+\omega$
ω^4
ω^ω
$\omega\cdot 3$
$\omega^2\cdot 3$
$\omega^3+\omega^2$
$\omega^2+\omega$
3
$\omega^3\cdot 2$
$\omega^2+\omega\cdot 2$
$\omega^2\cdot 2$
$\omega\cdot 2+3$
$\omega\cdot 2+2$
$\omega\cdot 2+1$
4
$\omega\cdot 2$
5

Ordinals



finite ordinals

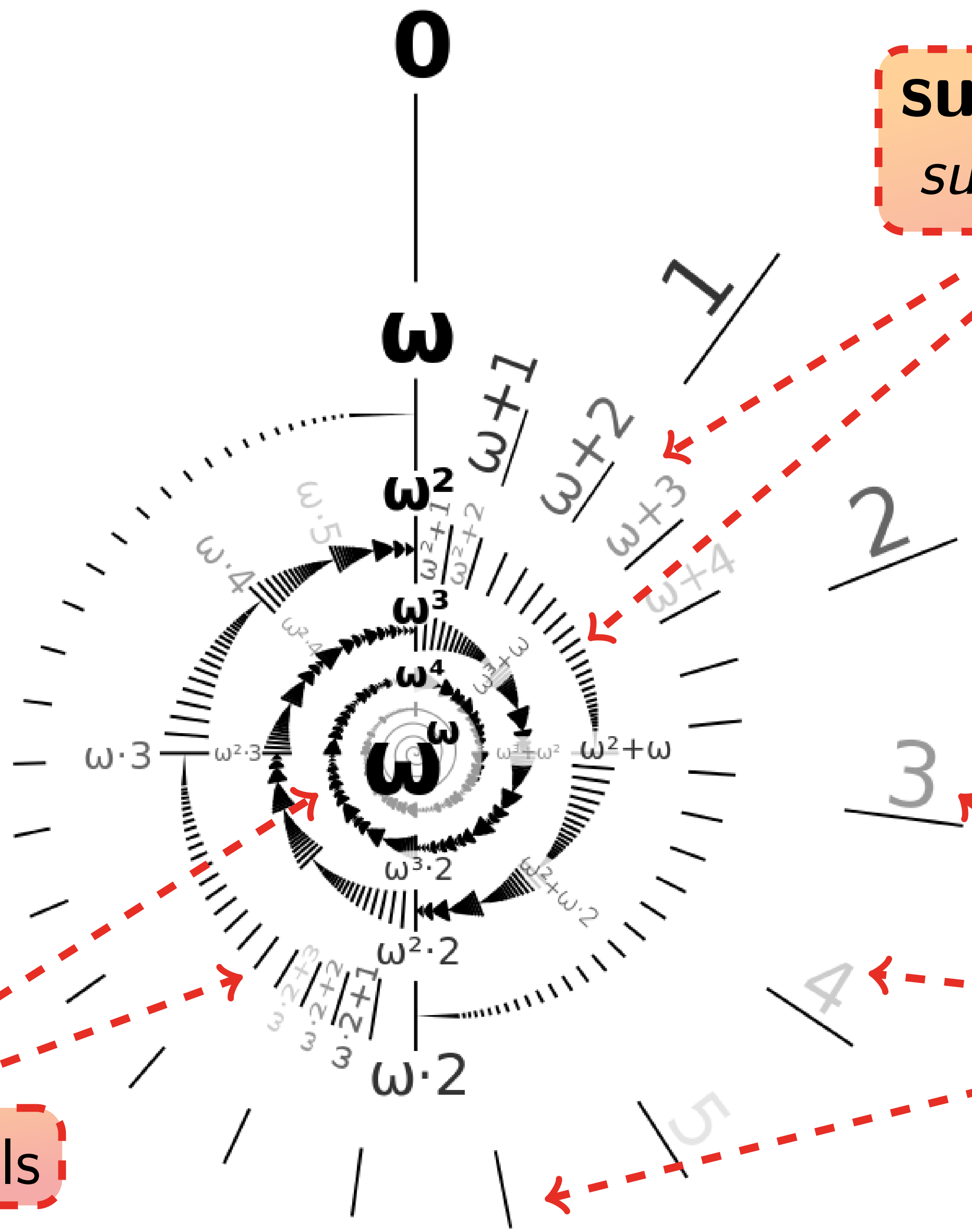
Ordinals



transfinite ordinals

finite ordinals

Ordinals



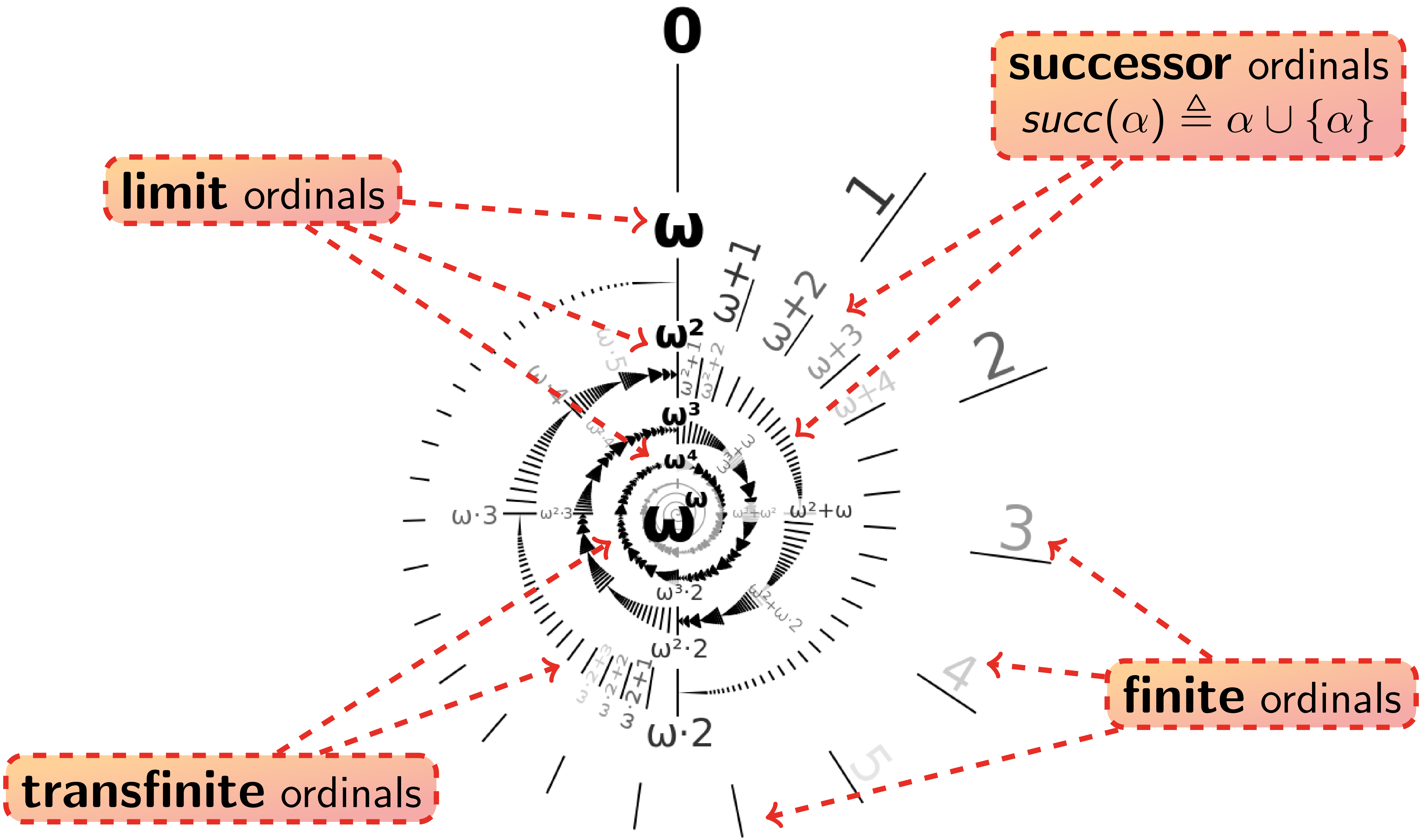
successor ordinals

$succ(\alpha) \triangleq \alpha \cup \{\alpha\}$

finite ordinals

transfinite ordinals

Ordinals



successor ordinals

$succ(\alpha) \triangleq \alpha \cup \{\alpha\}$

limit ordinals

finite ordinals

transfinite ordinals

Ordinal Arithmetic

- **addition**

$$
\begin{aligned}
\alpha + 0 &= \alpha && \text{(zero case)} \\
\alpha + \mathit{succ}(\beta) &= \mathit{succ}(\alpha + \beta) && \text{(successor case)} \\
\alpha + \beta &= \bigcup_{\gamma < \beta} (\alpha + \gamma) && \text{(limit case)}
\end{aligned}
$$

  - associative: $(\alpha + \beta) + \gamma = \alpha + (\beta + \gamma)$
  - not commutative: $1 + \omega = \omega \neq \omega + 1$
- **multiplication**

Ordinal Arithmetic

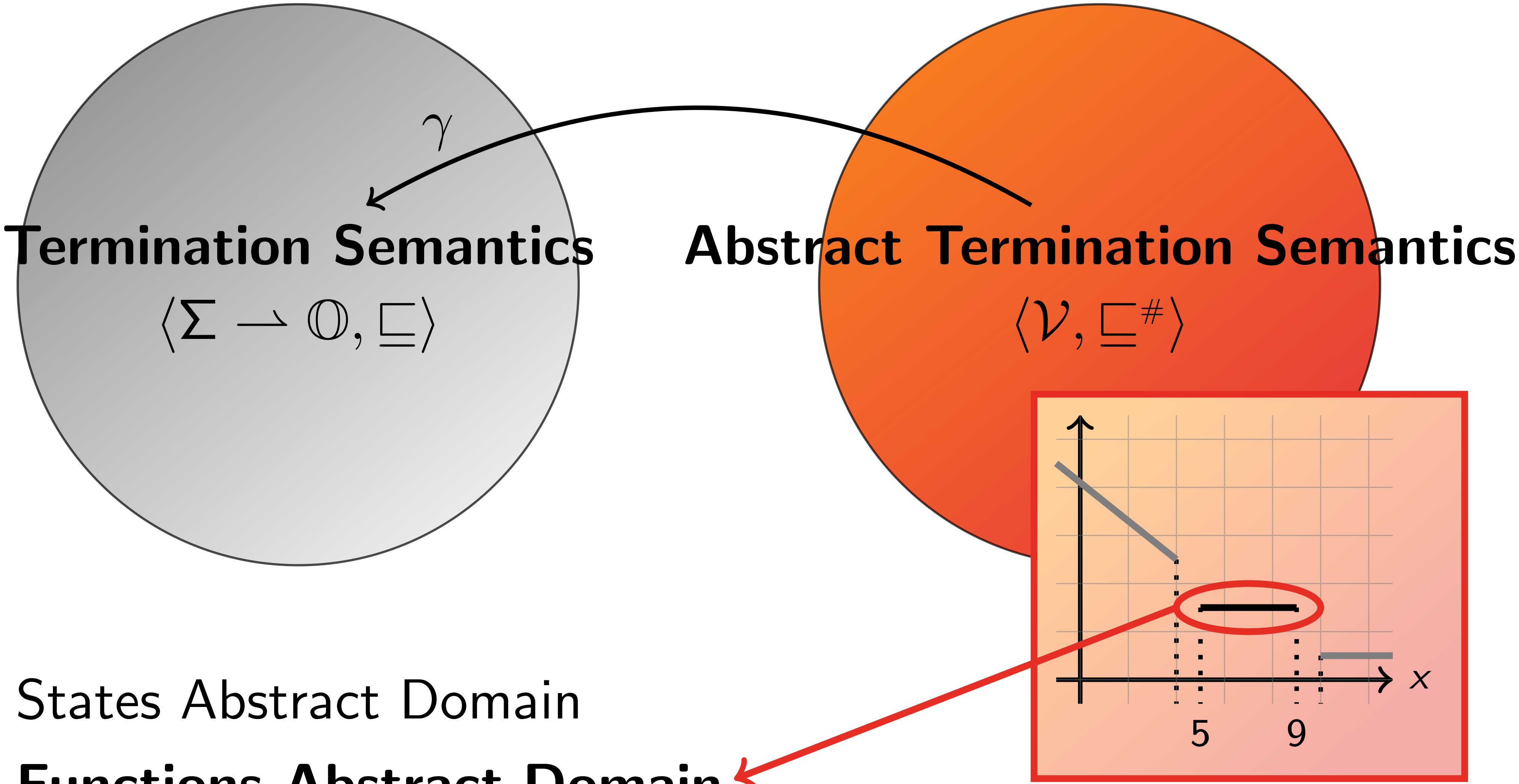
- addition
- multiplication

$$\alpha \cdot 0 = 0 \qquad \text{(zero case)}$$

$$\alpha \cdot succ(\beta) = (\alpha \cdot \beta) + \alpha \qquad \text{(successor case)}$$

$$\alpha \cdot \beta = \bigcup_{\gamma<\beta} (\alpha \cdot \gamma) \qquad \text{(limit case)}$$

  - associative: $(\alpha \cdot \beta) \cdot \gamma = \alpha \cdot (\beta \cdot \gamma)$
  - left distributive: $\alpha \cdot (\beta + \gamma) = (\alpha \cdot \beta) + (\alpha \cdot \gamma)$
  - not commutative: $2 \cdot \omega = \omega \neq \omega \cdot 2$
  - not right distributive: $(\omega + 1) \cdot \omega = \omega \cdot \omega \neq \omega \cdot \omega + \omega$



Termination Semantics

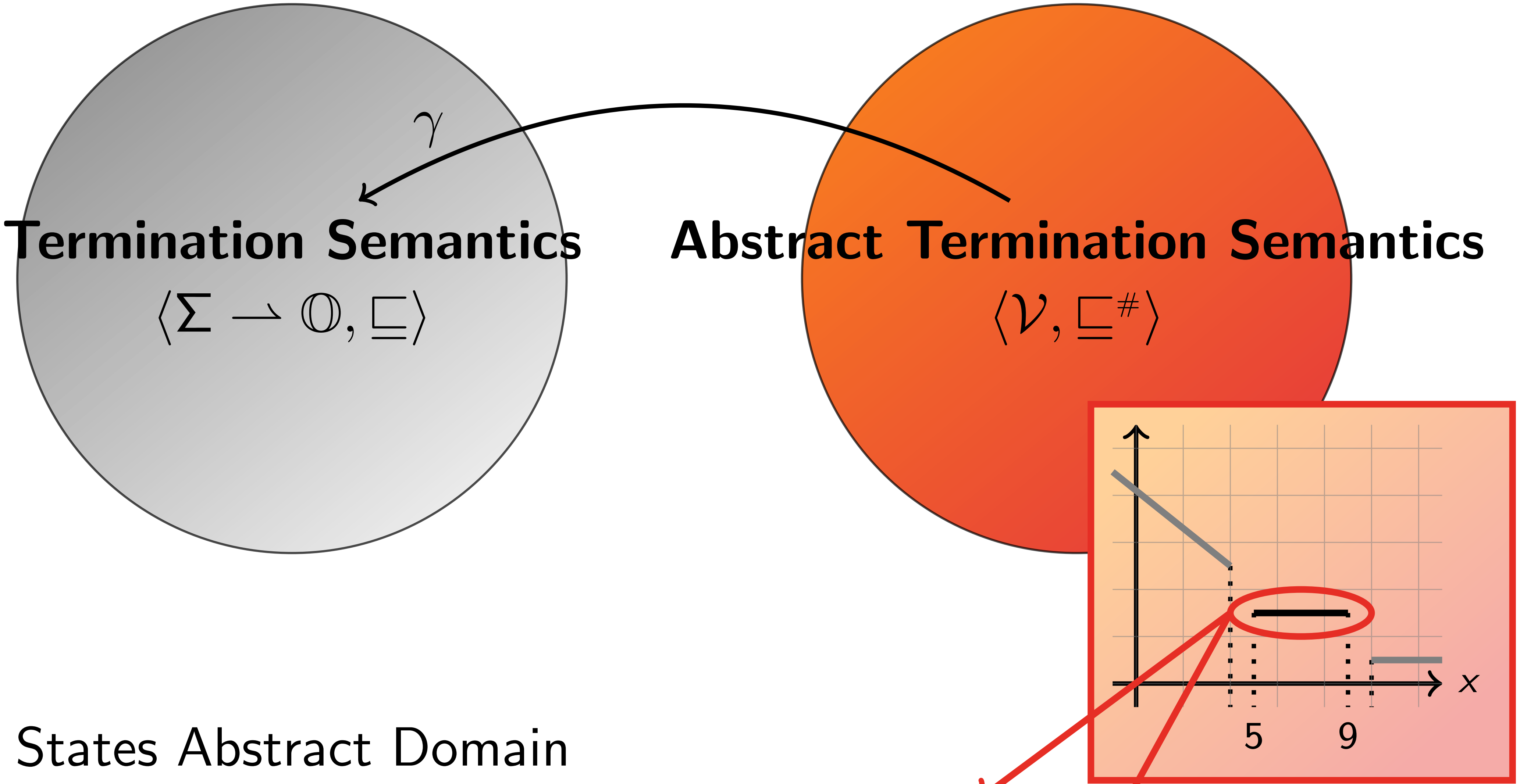
$\langle \Sigma \rightharpoonup \mathbb{O}, \sqsubseteq \rangle$

Abstract Termination Semantics

$\langle \mathcal{V}, \sqsubseteq^{\#} \rangle$

- States Abstract Domain S
- **Functions Abstract Domain** F
- Piecewise-Defined Ranking Functions Abstract Domain $\mathsf{V}(\mathsf{S}, \mathsf{F})$

Urban - *The Abstract Domain of Segmented Ranking Functions* (SAS 2013)



Termination Semantics

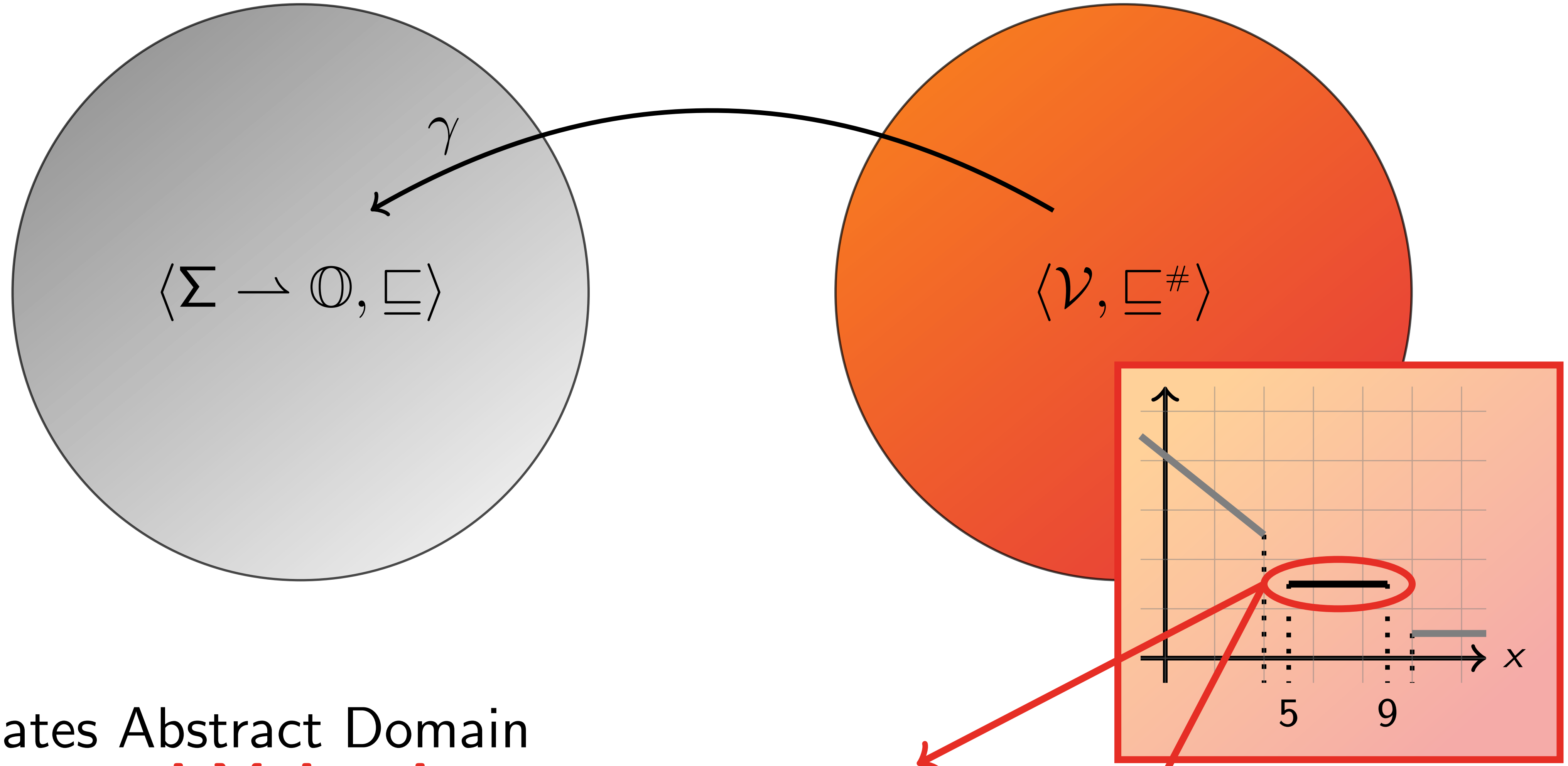
$\langle \Sigma \rightharpoonup \mathbb{O}, \sqsubseteq \rangle$

Abstract Termination Semantics

$\langle \mathcal{V}, \sqsubseteq^\# \rangle$

States Abstract Domain	S
Natural-Valued Functions Abstract Domain	F
Ordinal-Valued Functions Abstract Domain	**O**(F)
Piecewise-Defined Ranking Functions Abstract Domain	V(S, **O**(F))

Urban&Miné - *An Abstract Domain to Infer Ordinal-Valued Ranking Functions* (ESOP 2014)



- States Abstract Domain
- **Natural-Valued** Functions Abstract Domain
  - $\mathcal{F} \overset{\text{def}}{=} \{\bot\} \cup \{f \mid f \in \mathbb{Z}^n \to \mathbb{N}\} \cup \{\top\}$
    where $f \equiv f(x_1, \ldots, x_n) = m_1 x_1 + \cdots + m_n x_n + q$
- **Ordinal-Valued Functions Abstract Domain**
  - $\mathcal{O} \overset{\text{def}}{=} \{\bot\} \cup \{\sum_i \omega^i \cdot f_i \mid f_i \in \mathcal{F} \setminus \{\bot, \top\}\} \cup \{\top\}$
- Piecewise-Defined Ranking Functions Abstract Domain

Urban&Miné - *An Abstract Domain to Infer Ordinal-Valued Ranking Functions* (ESOP 2014)

Backward Non-Deterministic Assignments

- non-deterministic assignments are carried out in ascending powers of ω

Example

$$[-\infty, +\infty] \mapsto \omega \cdot x_1 + x_2$$

$$\Downarrow \quad \mathbf{x_1 := ?}$$

$$[-\infty, +\infty] \mapsto ?$$

Backward Non-Deterministic Assignments

- non-deterministic assignments are carried out in ascending powers of ω

Example

$$[-\infty, +\infty] \mapsto \omega \cdot x_1 + x_2$$

$$\Downarrow \quad \mathbf{x_1 := ?}$$

$$[-\infty, +\infty] \mapsto \quad + \quad \mathbf{1}$$

Backward Non-Deterministic Assignments

- non-deterministic assignments are carried out in ascending powers of ω

Example

$$[-\infty, +\infty] \mapsto \omega \cdot x_1 + \mathbf{x_2}$$

$$\Downarrow \quad \mathbf{x_1 := ?}$$

$$[-\infty, +\infty] \mapsto + \mathbf{x_2} + 1$$

Backward Non-Deterministic Assignments

- non-deterministic assignments are carried out in ascending powers of ω

Example

$$[-\infty, +\infty] \mapsto \omega \cdot \mathbf{x_1} + x_2$$

$$\Downarrow \quad \mathbf{x_1 := ?}$$

$$[-\infty, +\infty] \mapsto \boldsymbol{\omega^2 \cdot 1} + \omega \cdot \mathbf{0} + x_2 + 1$$

$$\omega \cdot \omega = \omega^2 \cdot 1 + \omega \cdot 0$$

Backward Non-Deterministic Assignments

- non-deterministic assignments are carried out in ascending powers of ω

Example

$$[-\infty,+\infty] \mapsto \omega \cdot x_1 + x_2$$

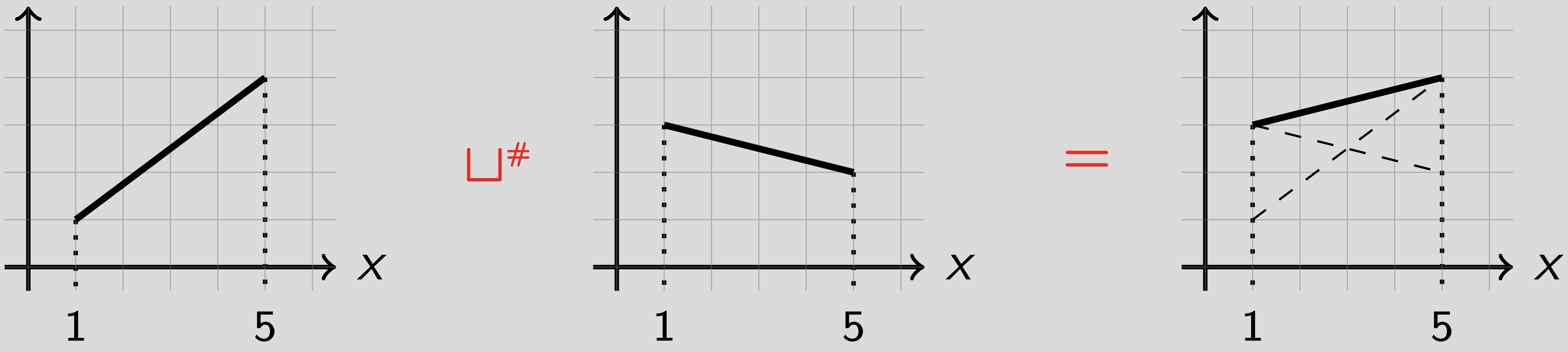
$$\Downarrow \quad \mathbf{x_1 := ?}$$

$$[-\infty,+\infty] \mapsto \omega^2 + x_2 + 1$$

Join

- join of natural-valued functions:

Example



- join of ordinal-valued functions:

Join

- join of natural-valued functions:
- join of ordinal-valued functions:
  - join of natural-valued functions in ascending powers of ω

Example

$$\begin{array}{lllllllllll}
[-\infty, +\infty] & \mapsto & o_1 & \equiv & \omega^2 \cdot x_1 & + & \omega \cdot x_2 & + & 3 \\
[-\infty, +\infty] & \mapsto & o_2 & \equiv & \omega^2 \cdot x_1 & + & \omega \cdot (-x_2) & + & 4 \\
\hline
[-\infty, +\infty] & \mapsto & o_1 \sqcup^{\#} o_2 & \equiv & ? & & & &
\end{array}$$

Join

- join of natural-valued functions:
- join of ordinal-valued functions:
    - join of natural-valued functions in ascending powers of ω

Example

$$\begin{array}{lclclclclcl}
[-\infty, +\infty] & \mapsto & o_1 & \equiv & \omega^2 \cdot x_1 & + & \omega \cdot x_2 & + & \mathbf{3} \\
[-\infty, +\infty] & \mapsto & o_2 & \equiv & \omega^2 \cdot x_1 & + & \omega \cdot (-x_2) & + & \mathbf{4} \\
\hline
[-\infty, +\infty] & \mapsto & o_1 \sqcup^\# o_2 & \equiv & & & & + & \mathbf{4}
\end{array}$$

Join

- join of natural-valued functions:
- join of ordinal-valued functions:
  - join of natural-valued functions in ascending powers of ω

Example

$$\begin{array}{lllcccccccccc}
[-\infty,+\infty] & \mapsto & o_1 & \equiv & \omega^2 & \cdot & x_1 & + & \omega & \cdot & x_2 & + & 3 \\
[-\infty,+\infty] & \mapsto & o_2 & \equiv & \omega^2 & \cdot & x_1 & + & \omega & \cdot & (-x_2) & + & 4 \\
\hline
[-\infty,+\infty] & \mapsto & o_1 \sqcup^{\#} o_2 & \equiv & & & \omega^2 \cdot 1 & + & \omega & \cdot & 0 & + & 4
\end{array}$$

$\omega \cdot \omega = \omega^2 \cdot 1 + \omega \cdot 0$

Join

- join of natural-valued functions:
- join of ordinal-valued functions:
  - join of natural-valued functions in ascending powers of ω

Example

$$
\begin{array}{lclcllclccl}
[-\infty, +\infty] & \mapsto & o_1 & \equiv & \omega^2 \cdot x_1 & & + & \omega \cdot x_2 & + & 3 \\
[-\infty, +\infty] & \mapsto & o_2 & \equiv & \omega^2 \cdot x_1 & & + & \omega \cdot (-x_2) & + & 4 \\
\hline
[-\infty, +\infty] & \mapsto & o_1 \sqcup^\# o_2 & \equiv & \omega^2 \cdot x_1 & \omega^2 \cdot 1 & & & + & 4
\end{array}
$$

Join

- join of natural-valued functions:
- join of ordinal-valued functions:
    - join of natural-valued functions in ascending powers of ω

Example

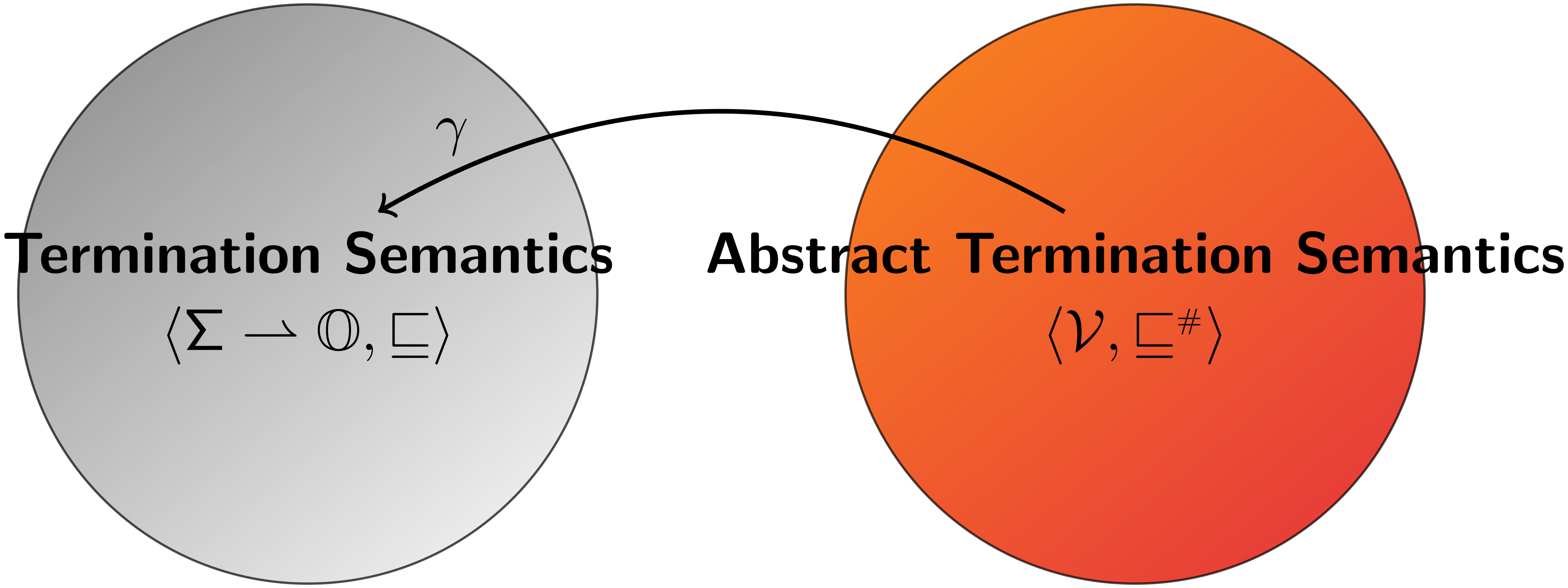
$$\begin{array}{lclclcccccccc}
[-\infty,+\infty] & \mapsto & o_1 & \equiv & \omega^2 & \cdot & \mathbf{x_1} & + & \omega & \cdot & x_2 & + & 3 \\
[-\infty,+\infty] & \mapsto & o_2 & \equiv & \omega^2 & \cdot & \mathbf{x_1} & + & \omega & \cdot & (-x_2) & + & 4 \\
\hline
[-\infty,+\infty] & \mapsto & o_1 \sqcup^\# o_2 & \equiv & \omega^2 & \cdot & (\mathbf{x_1}+\mathbf{1}) & & & & & + & 4
\end{array}$$

Join

- join of natural-valued functions:
- join of ordinal-valued functions:
    - join of natural-valued functions in ascending powers of ω

Example

$$
\begin{array}{lclcccccccc}
[-\infty, +\infty] & \mapsto & o_1 & \equiv & \omega^2 \cdot x_1 & + & \omega \cdot x_2 & + & 3 \\
[-\infty, +\infty] & \mapsto & o_2 & \equiv & \omega^2 \cdot x_1 & + & \omega \cdot (-x_2) & + & 4 \\
\hline
[-\infty, +\infty] & \mapsto & o_1 \sqcup^\# o_2 & \equiv & \omega^2 \cdot (x_1 + 1) & & & + & 4
\end{array}
$$



Termination Semantics

$\langle \Sigma \rightharpoonup \mathbb{O}, \sqsubseteq \rangle$

Abstract Termination Semantics

$\langle \mathcal{V}, \sqsubseteq^{\#} \rangle$

Theorem (Soundness)

the abstract termination semantics is **sound**
to prove the termination of programs

Example

```
int : $x_1, x_2$
while $^1(x_1 > 0 \wedge x_2 > 0)$ do
    if $^2($ ? $)$ then
        $^3x_1 := x_1 - 1$
        $^4x_2 :=$ ?
    else
        $^5x_2 := x_2 - 1$
od$^6$
```

$$f_1(x_1, x_2) = \begin{cases} 1 & x_1 \leq 0 \vee x_2 \leq 0 \\ \omega \cdot (x_1 - 1) + 7x_1 + 3x_2 - 5 & x_1 > 0 \wedge x_2 > 0 \end{cases}$$

Example

```
int : $x_1, x_2$
while $^{1}(x_1 \neq 0 \wedge x_2 > 0)$ do
  if $^{2}(x_1 > 0)$ then
    if $^{3}(\ ?\ )$ then
      $^{4}x_1 := x_1 - 1$
      $^{5}x_2 :=\ ?$
    else
      $^{6}x_2 := x_2 - 1$
  else $/ *\ x_1 < 0\ * /$
    if $^{7}(\ ?\ )$ then
      $^{8}x_1 := x_1 + 1$
    else
      $^{9}x_2 := x_2 - 1$
      $^{10}x_1 :=\ ?$
od$^{11}$
```

$$f_1(x_1, x_2) = \begin{cases} \omega^2 + \omega \cdot (x_2 - 1) - 4x_1 + 9x_2 - 2 & x_1 < 0 \wedge x_2 > 0 \\ 1 & x_1 = 0 \vee x_2 \leq 0 \\ \omega \cdot (x_1 - 1) + 9x_1 + 4x_2 - 7 & x_1 > 0 \wedge x_2 > 0 \end{cases}$$

Example

```
int : $x_1, x_2$
while $^{1}(x_1 \neq 0 \wedge x_2 > 0)$ do
    if $^{2}(x_1 > 0)$ then
        if $^{3}(\ ?\ )$ then
            $^{4}x_1 := x_1 - 1$
            $^{5}x_2 := \ ?$
        else
            $^{6}x_2 := x_2 - 1$
    else /* $x_1 < 0$ */
        if $^{7}(\ ?\ )$ then
            $^{8}x_1 := x_1 + 1$
        else
            $^{9}x_2 := x_2 - 1$
            $^{10}x_1 := \ ?$
```

the coefficients and their **order** are **inferred by the analysis**

$$f_1(x_1, x_2) = \begin{cases} \omega^2 + \omega \cdot (x_2 - 1) + 4x_1 + 9x_2 - 2 & x_1 < 0 \wedge x_2 > 0 \\ 1 & x_1 = 0 \vee x_2 \leq 0 \\ \omega \cdot (x_1 - 1) + 9x_1 + 4x_2 - 7 & x_1 > 0 \wedge x_2 > 0 \end{cases}$$

Non-Linear Ranking Functions

Example

```
int : $N$,  $x_1$,  $x_2$
$^1 x_1 := N$
while $^2(x_1 \geq 0)$ do
    $^3 x_2 := N$
    while $^4(x_2 \geq 0)$ do
        $^5 x_2 := x_2 - 1$
    od$^6$
    $^7 x_1 := x_1 - 1$
od$^8$
```

$$f_1(x_1, x_2, N) = \begin{cases} 1 & x_1 < 0 \\ \omega \cdot (x_1 + 1) + 6x_1 + 7 & x_1 \geq 0 \end{cases}$$

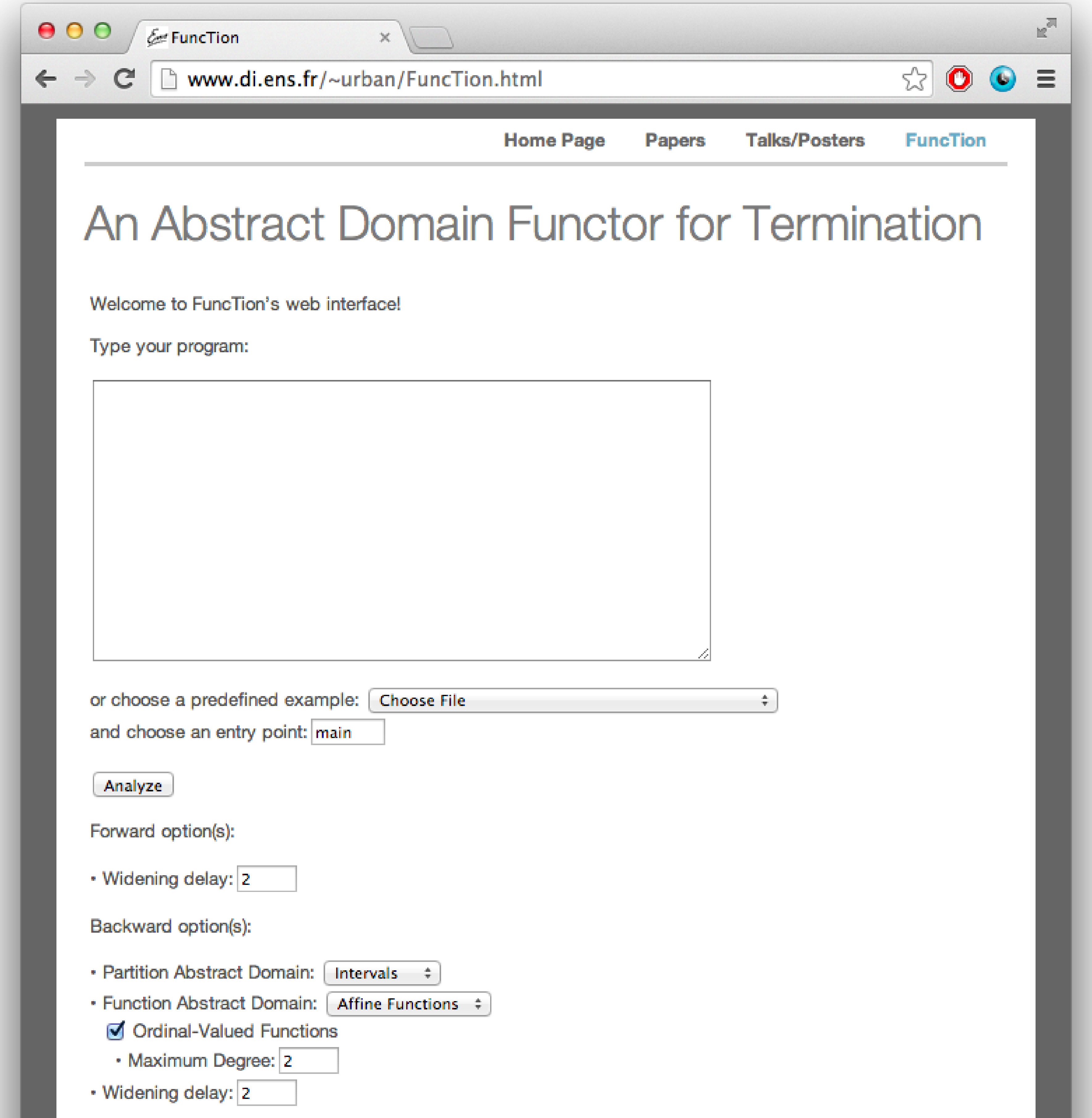
Non-Linear Ranking Functions

Example

```
int : $N, \; x_1, \; x_2$
$^{1}x_1 := N$
while $^{2}(x_1 \geq 0)$ do
    $^{3}x_2 := N$
    while $^{4}(x_2 \geq 0)$ do
        $^{5}x_2 := x_2 - 1$
    od$^{6}$
    $^{7}x_1 := x_1 - 1$
od$^{8}$
```

$$f_1(x_1, x_2, N) = \begin{cases} 1 & x_1 < 0 \\ \omega \cdot (x_1 + 1) + 6x_1 + 7 & x_1 \geq 0 \end{cases}$$

the loop terminates in a
finite number of iterations



An Abstract Domain Functor for Termination

Welcome to FuncTion's web interface!

Type your program:

or choose a predefined example: Choose File

and choose an entry point: main

Forward option(s):

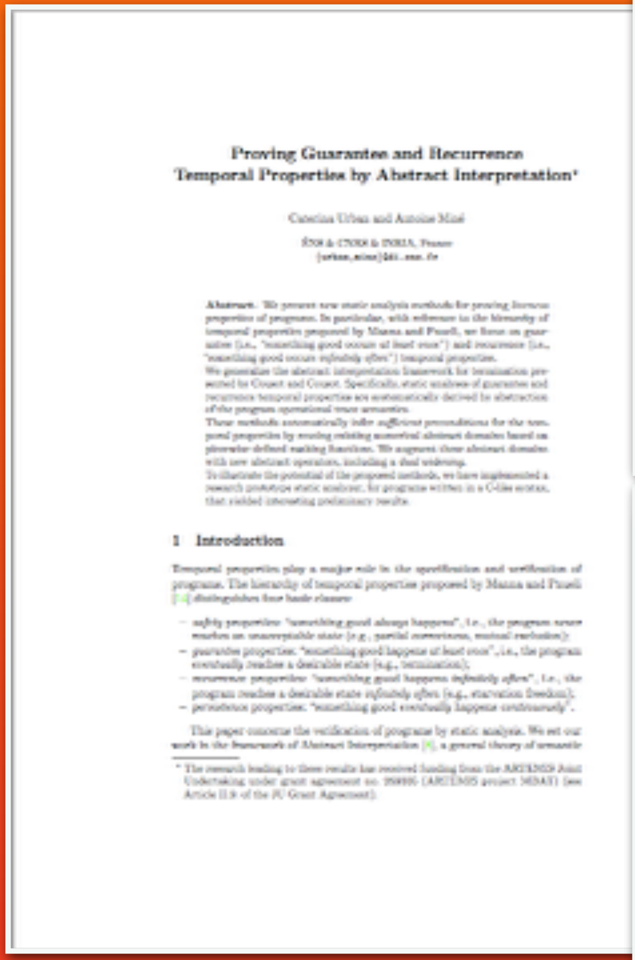
- Widening delay: 2

Backward option(s):

- Partition Abstract Domain: Intervals
- Function Abstract Domain: Affine Functions

  ☑ Ordinal-Valued Functions
  - Maximum Degree: 2
- Widening delay: 2

Guarantee Semantics

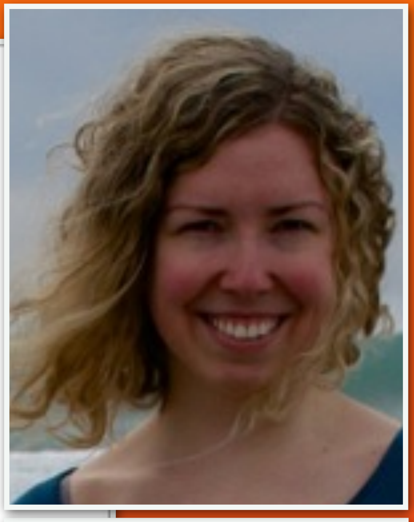


Proving Guarantee and Recurrence Temporal Properties by Abstract Interpretation*

Caterina Urban and Antoine Miné

ÉNS & CNRS & INRIA, France
{urban,mine}@di.ens.fr

1 Introduction



program $\mapsto$ maximal trace semantics $\rightarrow$ φ-**guarantee semantics**

idea = define a ranking function **counting the number of program steps** from the property φ

$$\mathcal{T}_{\mathbf{g}}^{\varphi} \in \Sigma \rightharpoonup \mathbb{O}$$

$$\mathcal{T}_{\mathbf{g}}^{\varphi} \overset{\text{def}}{=} \mathbf{lfp}\ \mathbf{F}_{\mathbf{g}}^{\varphi}$$

$$\mathbf{F}_{\mathbf{g}}^{\varphi}(\mathbf{v})\mathbf{s} \overset{\text{def}}{=} \begin{cases} \mathbf{0} & \mathbf{s} \models \varphi \\ \mathbf{sup}\{\ \mathbf{v}(\mathbf{s}') + \mathbf{1} \mid \mathbf{s} \rightarrow \mathbf{s}'\ \} & \mathbf{s} \in \widetilde{\mathbf{pre}}(\mathbf{dom}(\mathbf{v}))\ \wedge\ \mathbf{s} \not\models \varphi \\ \mathbf{undefined} & \mathbf{otherwise} \end{cases}$$

Urban&Mine - *Proving Guarantee and Recurrence Temporal Properties by Abstract Interpretation* (VMCAI 2015)

program ↦ maximal trace semantics → **φ-guarantee semantics**

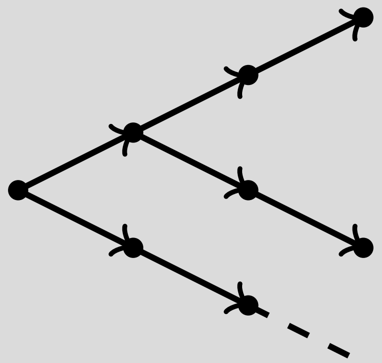
idea = define a ranking function **counting the number of program steps** from the property φ

$$\mathcal{T}_{\mathbf{g}}^{\varphi} \in \Sigma \rightharpoonup \mathbb{O}$$

$$\mathcal{T}_{\mathbf{g}}^{\varphi} \overset{\text{def}}{=} \mathbf{lfp}\ \mathbf{F}_{\mathbf{g}}^{\varphi}$$

$$\mathbf{F}_{\mathbf{g}}^{\varphi}(\mathbf{v})\mathbf{s} \overset{\text{def}}{=} \begin{cases} \mathbf{0} & \mathbf{s} \models \varphi \\ \mathbf{sup}\{\ \mathbf{v}(\mathbf{s}') + \mathbf{1} \mid \mathbf{s} \to \mathbf{s}'\ \} & \mathbf{s} \in \widetilde{\mathbf{pre}}(\mathbf{dom}(\mathbf{v})) \ \wedge\ \mathbf{s} \not\models \varphi \\ \mathbf{undefined} & \mathbf{otherwise} \end{cases}$$

Example



Urban&Mine - *Proving Guarantee and Recurrence Temporal Properties by Abstract Interpretation* (VMCAI 2015)

program ↦ maximal trace semantics → **φ-guarantee semantics**

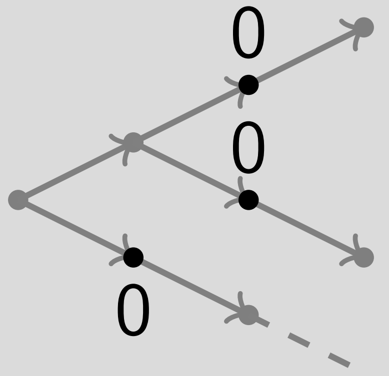
idea = define a ranking function **counting the number of program steps** from the property φ

$$\mathcal{T}_{\mathbf{g}}^{\varphi} \in \Sigma \rightharpoonup \mathbb{O}$$

$$\mathcal{T}_{\mathbf{g}}^{\varphi} \overset{\text{def}}{=} \mathbf{lfp}\ \mathbf{F}_{\mathbf{g}}^{\varphi}$$

$$\mathbf{F}_{\mathbf{g}}^{\varphi}(\mathbf{v})\mathbf{s} \overset{\text{def}}{=} \begin{cases} \mathbf{0} & \mathbf{s} \models \varphi \longleftarrow \\ \mathbf{sup}\{\ \mathbf{v}(\mathbf{s}') + \mathbf{1} \mid \mathbf{s} \to \mathbf{s}'\ \} & \mathbf{s} \in \widetilde{\mathbf{pre}}(\mathbf{dom}(\mathbf{v})) \ \wedge\ \mathbf{s} \not\models \varphi \\ \mathbf{undefined} & \mathbf{otherwise} \end{cases}$$

Example



Urban&Mine - *Proving Guarantee and Recurrence Temporal Properties by Abstract Interpretation* (VMCAI 2015)

program $\mapsto$ maximal trace semantics $\to$ φ-**guarantee semantics**

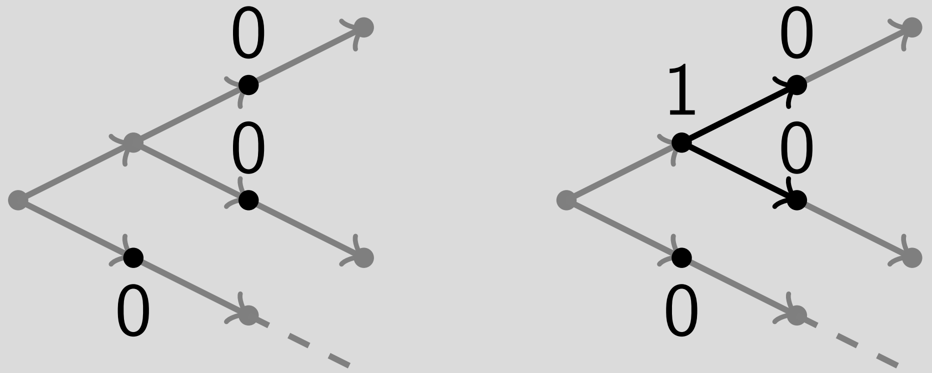
idea = define a ranking function **counting the number of program steps** from the property φ

$$\mathcal{T}_{\mathbf{g}}^{\varphi} \in \Sigma \rightharpoonup \mathbb{O}$$

$$\mathcal{T}_{\mathbf{g}}^{\varphi} \overset{\text{def}}{=} \mathbf{lfp}\ \mathbf{F}_{\mathbf{g}}^{\varphi}$$

$$\mathbf{F}_{\mathbf{g}}^{\varphi}(\mathbf{v})\mathbf{s} \overset{\text{def}}{=} \begin{cases} \mathbf{0} & \mathbf{s} \models \varphi \\ \mathbf{sup}\{\ \mathbf{v}(\mathbf{s}') + \mathbf{1} \mid \mathbf{s} \to \mathbf{s}'\ \} & \mathbf{s} \in \widetilde{\mathbf{pre}}(\mathbf{dom}(\mathbf{v}))\ \wedge\ \mathbf{s} \not\models \varphi \leftarrow \\ \mathbf{undefined} & \mathbf{otherwise} \end{cases}$$

Example



Urban&Mine - *Proving Guarantee and Recurrence Temporal Properties by Abstract Interpretation* (VMCAI 2015)

program $\mapsto$ maximal trace semantics $\rightarrow$ **φ-guarantee semantics**

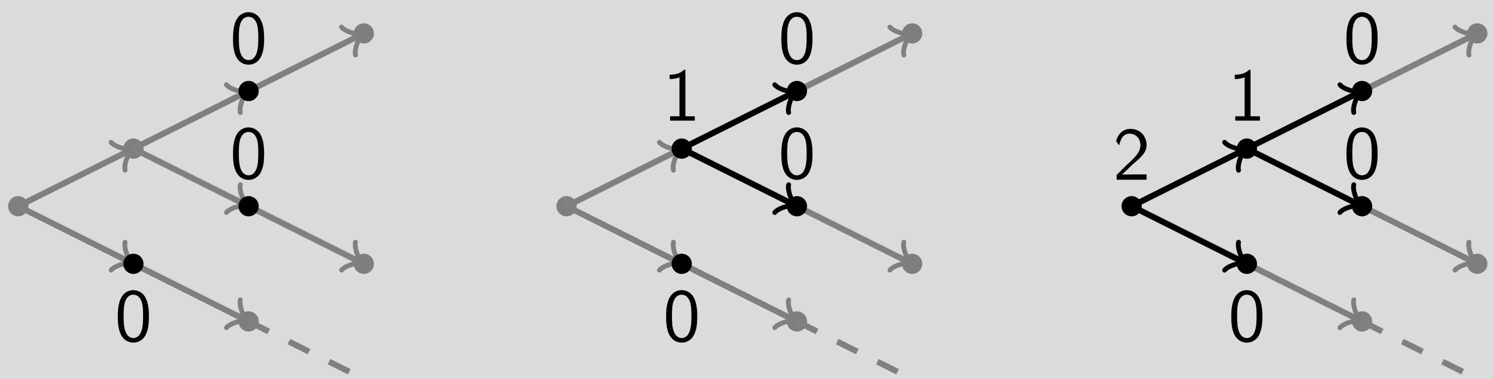
$$\mathcal{T}_{\mathbf{g}}^{\varphi} \in \Sigma \rightharpoonup \mathbb{O}$$

$$\mathcal{T}_{\mathbf{g}}^{\varphi} \overset{\text{def}}{=} \mathbf{lfp}\ \mathbf{F}_{\mathbf{g}}^{\varphi}$$

idea = define a ranking function **counting the number of program steps** from the property φ

$$\mathbf{F}_{\mathbf{g}}^{\varphi}(\mathbf{v})\mathbf{s} \overset{\text{def}}{=} \begin{cases} \mathbf{0} & \mathbf{s} \models \varphi \\ \mathbf{sup}\{\ \mathbf{v}(\mathbf{s}') + 1 \mid \mathbf{s} \rightarrow \mathbf{s}'\ \} & \mathbf{s} \in \widetilde{\mathbf{pre}}(\mathbf{dom}(\mathbf{v}))\ \wedge\ \mathbf{s} \not\models \varphi \\ \mathbf{undefined} & \mathbf{otherwise} \end{cases}$$

Example



Urban&Mine - *Proving Guarantee and Recurrence Temporal Properties by Abstract Interpretation* (VMCAI 2015)

program $\mapsto$ maximal trace semantics $\rightarrow$ **φ-guarantee semantics**

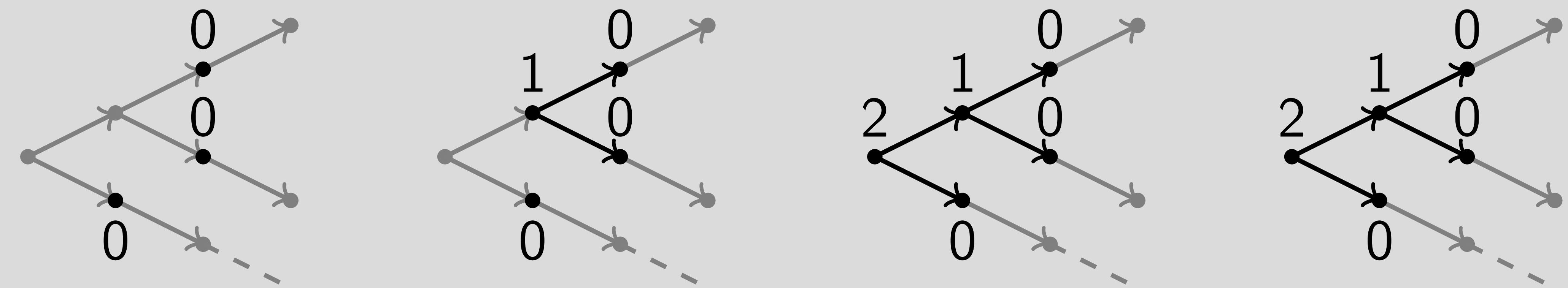
$$\mathcal{T}_{\mathbf{g}}^{\varphi} \in \Sigma \rightharpoonup \mathbb{O}$$

$$\mathcal{T}_{\mathbf{g}}^{\varphi} \overset{\text{def}}{=} \mathbf{lfp}\ \mathbf{F}_{\mathbf{g}}^{\varphi}$$

idea = define a ranking function **counting the number of program steps** from the property φ

$$\mathbf{F}_{\mathbf{g}}^{\varphi}(\mathbf{v})\mathbf{s} \overset{\text{def}}{=} \begin{cases} \mathbf{0} & \mathbf{s} \models \varphi \\ \mathbf{sup}\{\ \mathbf{v}(\mathbf{s}') + \mathbf{1} \mid \mathbf{s} \rightarrow \mathbf{s}'\ \} & \mathbf{s} \in \widetilde{\mathbf{pre}}(\mathbf{dom}(\mathbf{v}))\ \wedge\ \mathbf{s} \not\models \varphi \\ \mathbf{undefined} & \mathbf{otherwise} \end{cases}$$

Example



Urban&Mine - *Proving Guarantee and Recurrence Temporal Properties by Abstract Interpretation* (VMCAI 2015)

program $\mapsto$ maximal trace semantics $\rightarrow$ **φ-guarantee semantics**

$$\mathcal{T}_{\mathbf{g}}^{\varphi} \in \Sigma \rightharpoonup \mathbb{O}$$

$$\mathcal{T}_{\mathbf{g}}^{\varphi} \overset{\text{def}}{=} \mathbf{lfp}\ \mathbf{F}_{\mathbf{g}}^{\varphi}$$

$$\mathbf{F}_{\mathbf{g}}^{\varphi}(v)s \overset{\text{def}}{=} \begin{cases} 0 & s \models \varphi \\ \sup\{\, v(s') + 1 \mid s \rightarrow s' \,\} & s \in \widetilde{\mathrm{pre}}(\mathrm{dom}(v)) \ \wedge \ s \not\models \varphi \\ \text{undefined} & \text{otherwise} \end{cases}$$

Theorem (Soundness and Completeness)

the φ-guarantee semantics is **sound** *and* **complete**
to prove the guarantee property $\Diamond\varphi$

Urban&Mine - *Proving Guarantee and Recurrence Temporal Properties by Abstract Interpretation* (VMCAI 2015)

Bibliography

An Abstract Interpretation Framework f

Patrick Cousot

The Abstract Domain of Segmented Ranking Functions

Caterina Urban

An Abstract Domain to Infer Ordinal-Valued Ranking Functions

Caterina Urban and Antoine Miné

A Decision Tree Abstract Domain for Proving Conditional Termination

Caterina Urban and Antoine Miné

Proving Guarantee and Recurrence Temporal Properties by Abstract Interpretation

Caterina Urban and Antoine Miné

- **Cousot & Cousot** - *An Abstract Interpretation Framework for Termination* (POPL 2012)
- **Urban** - *The Abstract Domain of Segmented Ranking Functions* (SAS 2013)
- **Urban & Miné** - *An Abstract Domain to Infer Ordinal-Valued Ranking Functions* (ESOP 2014)
- **Urban & Miné** - *A Decision Tree Abstract Domain for Proving Conditional Termination* (SAS 2014)
- **Urban & Miné** - *Proving Guarantee and Recurrence Temporal Properties by Abstract Interpretation* (VMCAI 2015)