Static Analysis by Abstract Interpretation of Functional Temporal Properties of Programs

Analyse Statique par Interprétation Abstraite de Propriétés Temporelles Fonctionnelles des Programmes

Caterina Urban

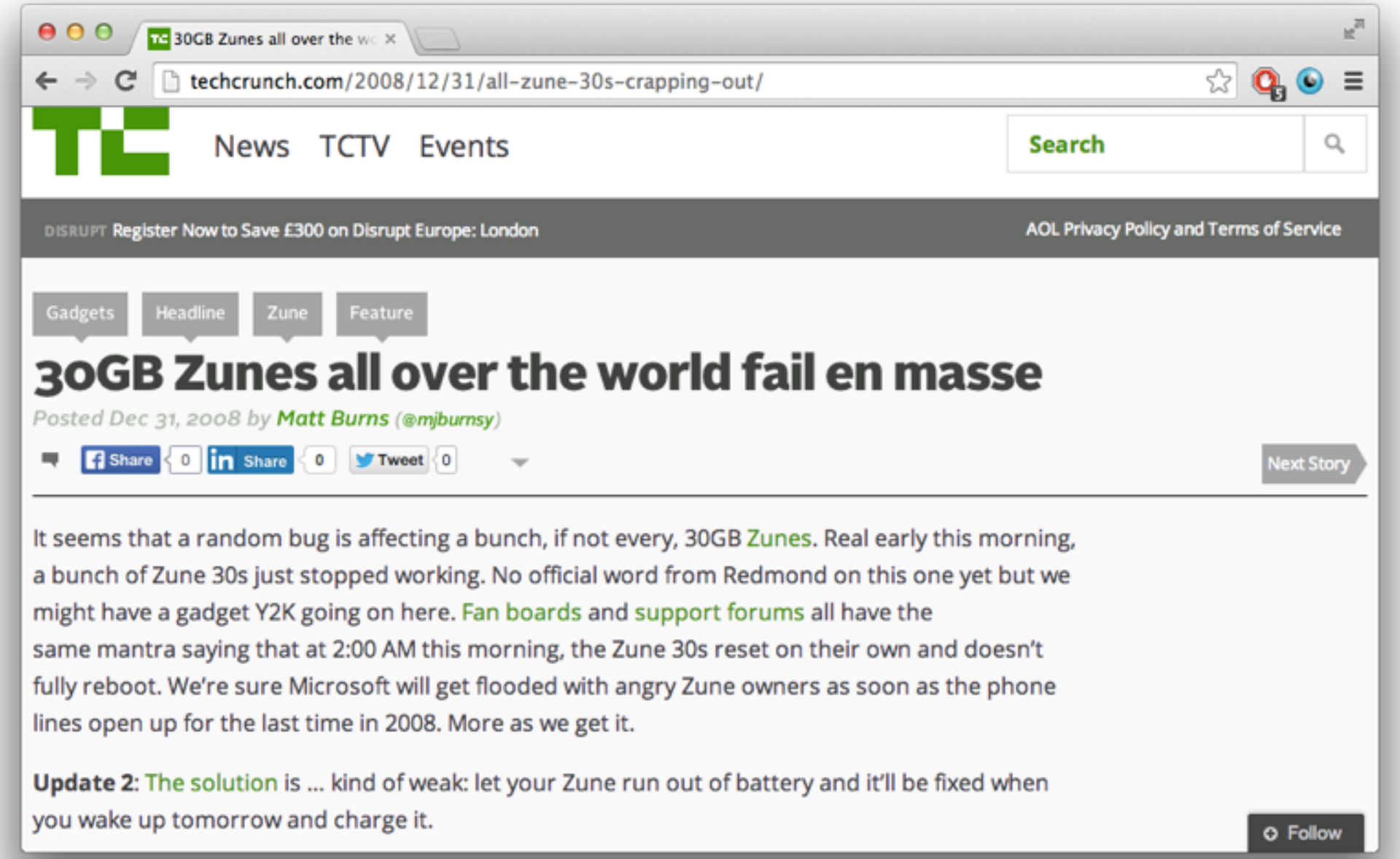
July 9th, 2015

PhD Defense

École Normale Supérieure, Paris

The Zune Bug

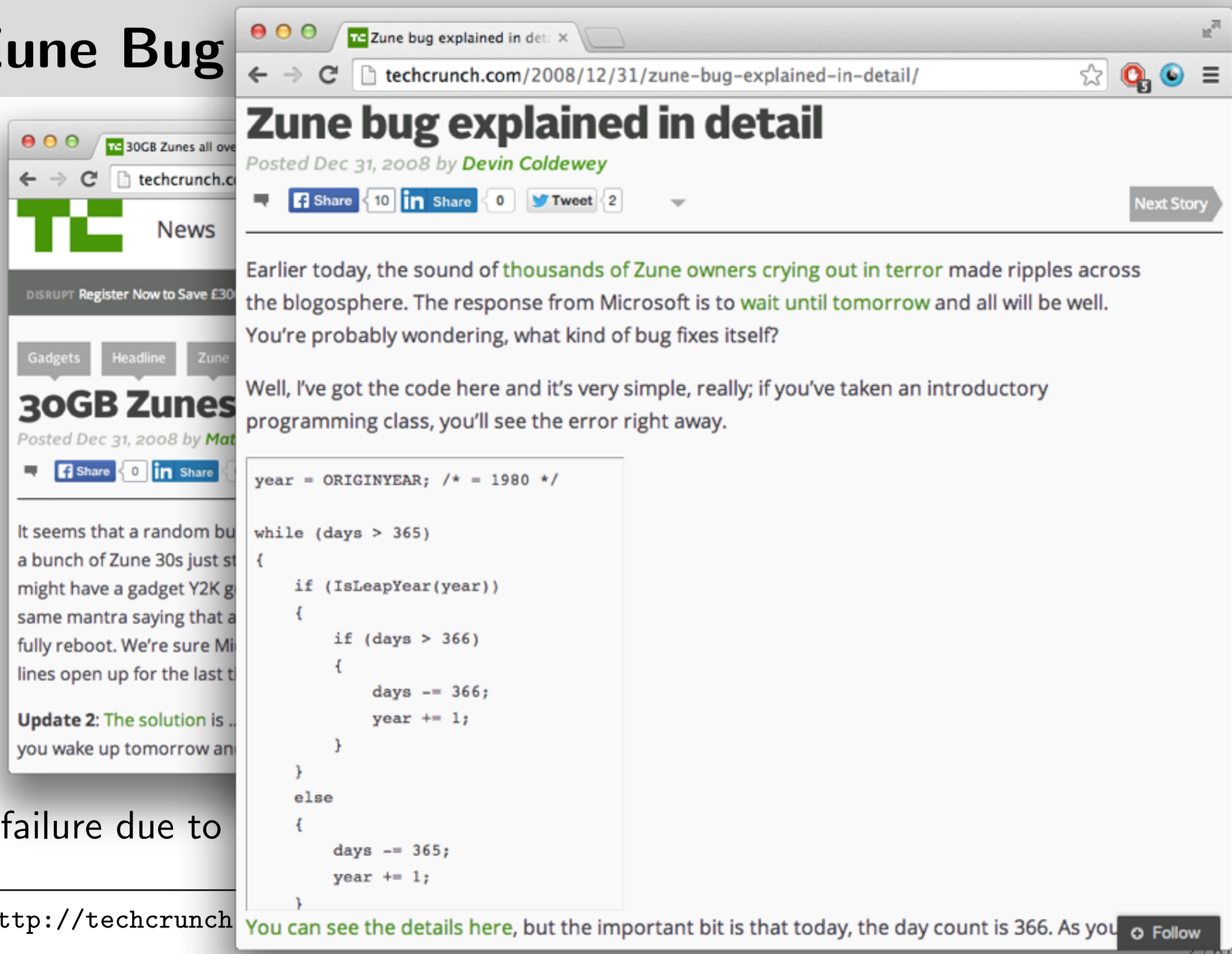
December 31st, 2008



- failure due to **non-termination**

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

The Zune Bug



- failure due to

`http://techcrunch`

- Microsoft **Zune** December 31st, 2008

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

- **Apache** HTTP Server Versions <2.3.3

`http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1890`

- Microsoft **Azure Storage Service** November 19th, 2014

`http://azure.microsoft.com/blog/2014/11/19/update-on-azure-storage-%20service-interruption`

- **safety properties**: *"something good always happens"* (e.g., partial correctness, mutual exclusion)

$$\Box\varphi$$

- **guarantee properties**: *"something good happens at least once"* (e.g., total correctness, termination)

$$\Diamond\varphi$$

- **recurrence properties**: *"something good happens infinitely often"* (e.g., starvation freedom)

$$\Box\Diamond\varphi$$

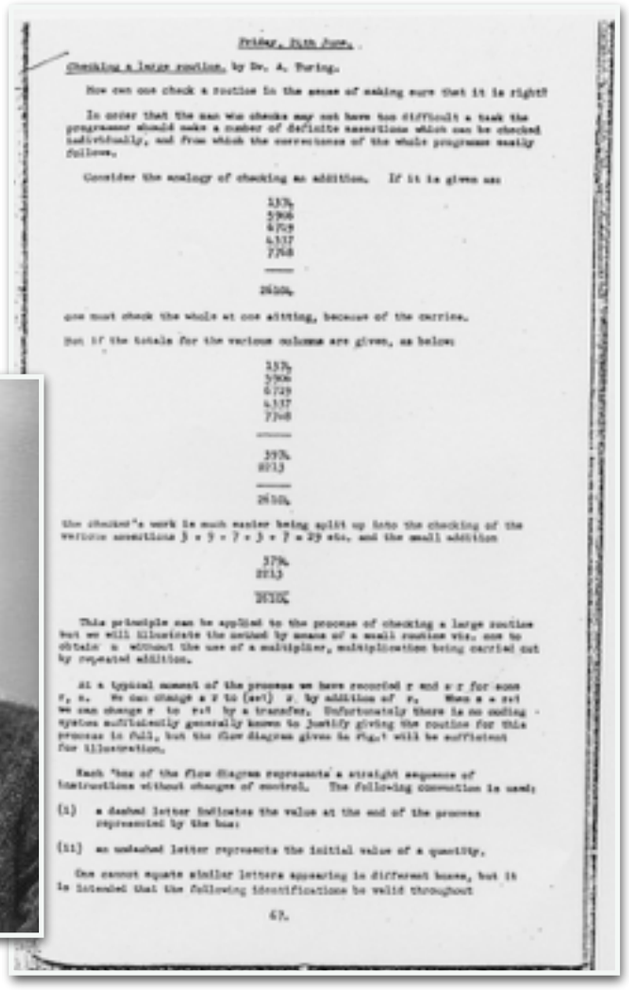
- **persistence properties**: *"something good eventually happens continuously"* (e.g., stabilization)

$$\Diamond\Box\varphi$$

Manna & Pnueli - *A Hierarchy of Temporal Properties* (PODC 1990)

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.

Consider the analogy of checking an addition. If it is given as:

137,
5906
6729
4337
7368

26536,

one must check the whole at one sitting, because of the carries.

But if the totals for the various columns are given, as below:

137,
5906
6729
4337
7368

378,
8713

26536,

the checker's work is much easier being split up into the checking of the various assertions $3 + 9 + 7 + 3 + 7 = 29$ etc., and the small addition

378,
8713

26536

This principle can be applied to the process of checking a large routine but we will illustrate the method by means of a small routine viz. one to obtain n without the use of a multiplier, multiplication being carried out by repeated addition.

At a typical moment in the process we have recorded r and $s \times r$ for some r, s. We can change $s \times r$ to $(s+1) \times r$ by addition of r. When $s = n$ we can change r to $r \times n$ by a transfer. Unfortunately there is no coding - system sufficiently generally known to justify giving the routine for this process in full, but the flow diagram given in Fig.1 will be sufficient for illustration.

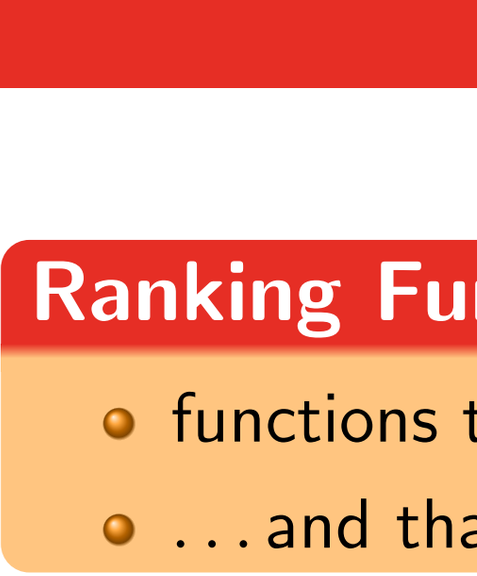
Each box of the flow diagram represents a straight sequence of instructions without changes of control. The following convention is used:

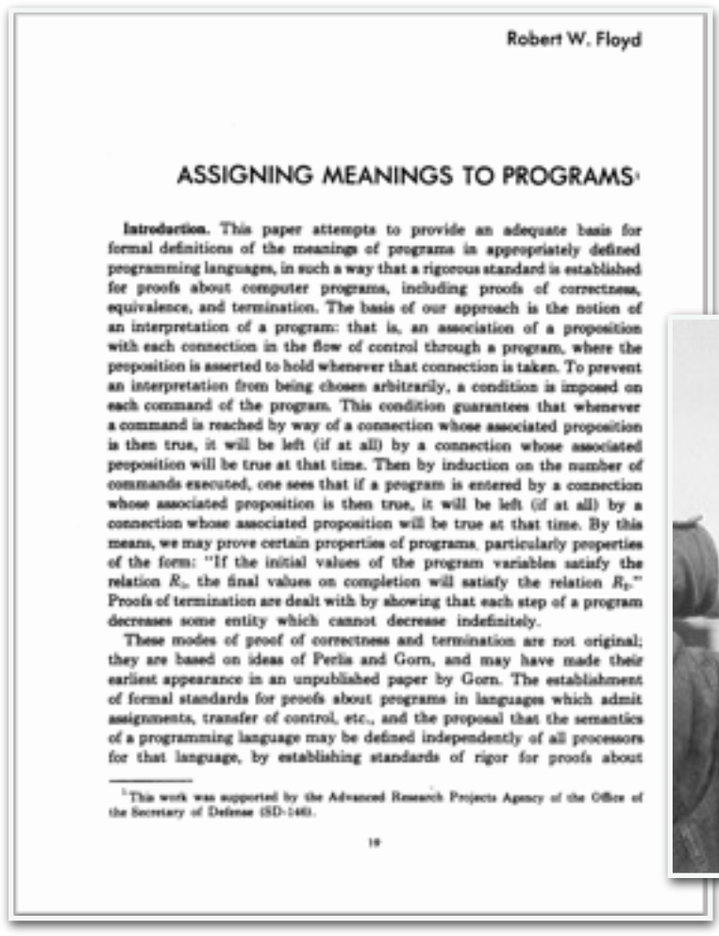
(i) a dashed letter indicates the value at the end of the process represented by the box;

(ii) an undashed letter represents the initial value of a quantity.

One cannot equate similar letters appearing in different boxes, but it is intended that the following identifications be valid throughout

67.





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)

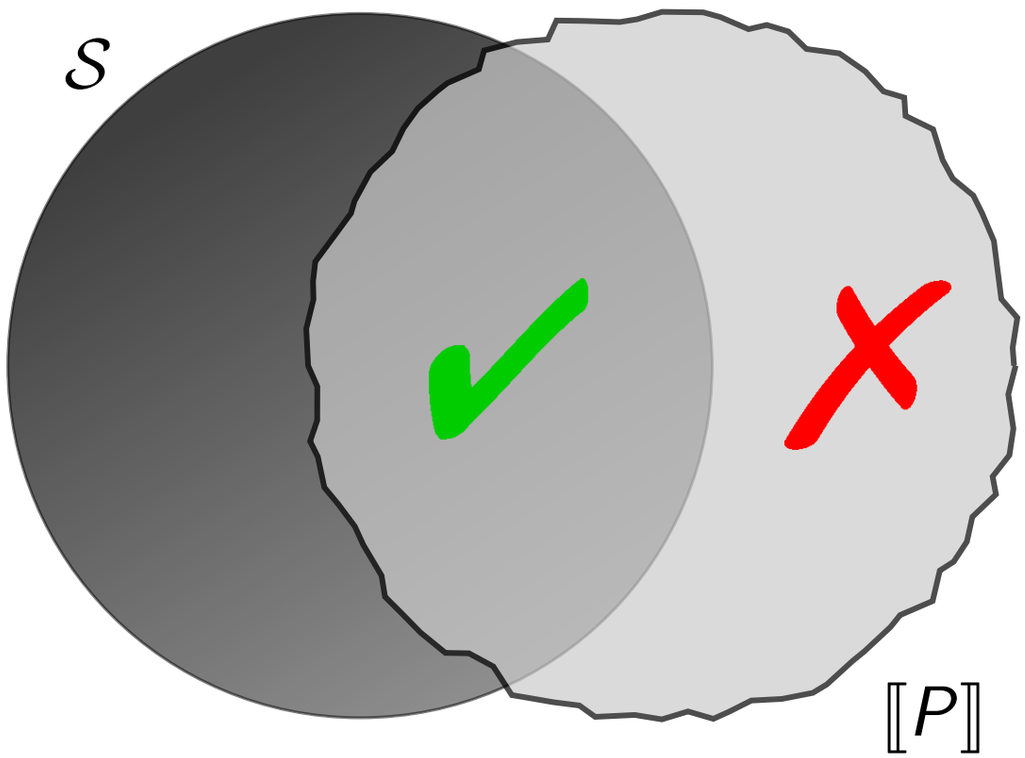
- **idea**: inference of ranking functions by **abstract interpretation**
  - **termination semantics**
  - **guarantee semantics**
  - **recurrence semantics**

Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)
Urban&Miné - *Proving Guarantee and Recurrence Temporal Properties by Abstract interpretation* (VMCAI 2015)

- **idea**: inference of ranking functions by **abstract interpretation**
  - **termination semantics**
  - **guarantee semantics**
  - **recurrence semantics**
- family of **abstract domains** for liveness properties
  - **piecewise-defined ranking functions**
  - **backward analysis**
  - **sufficient preconditions**

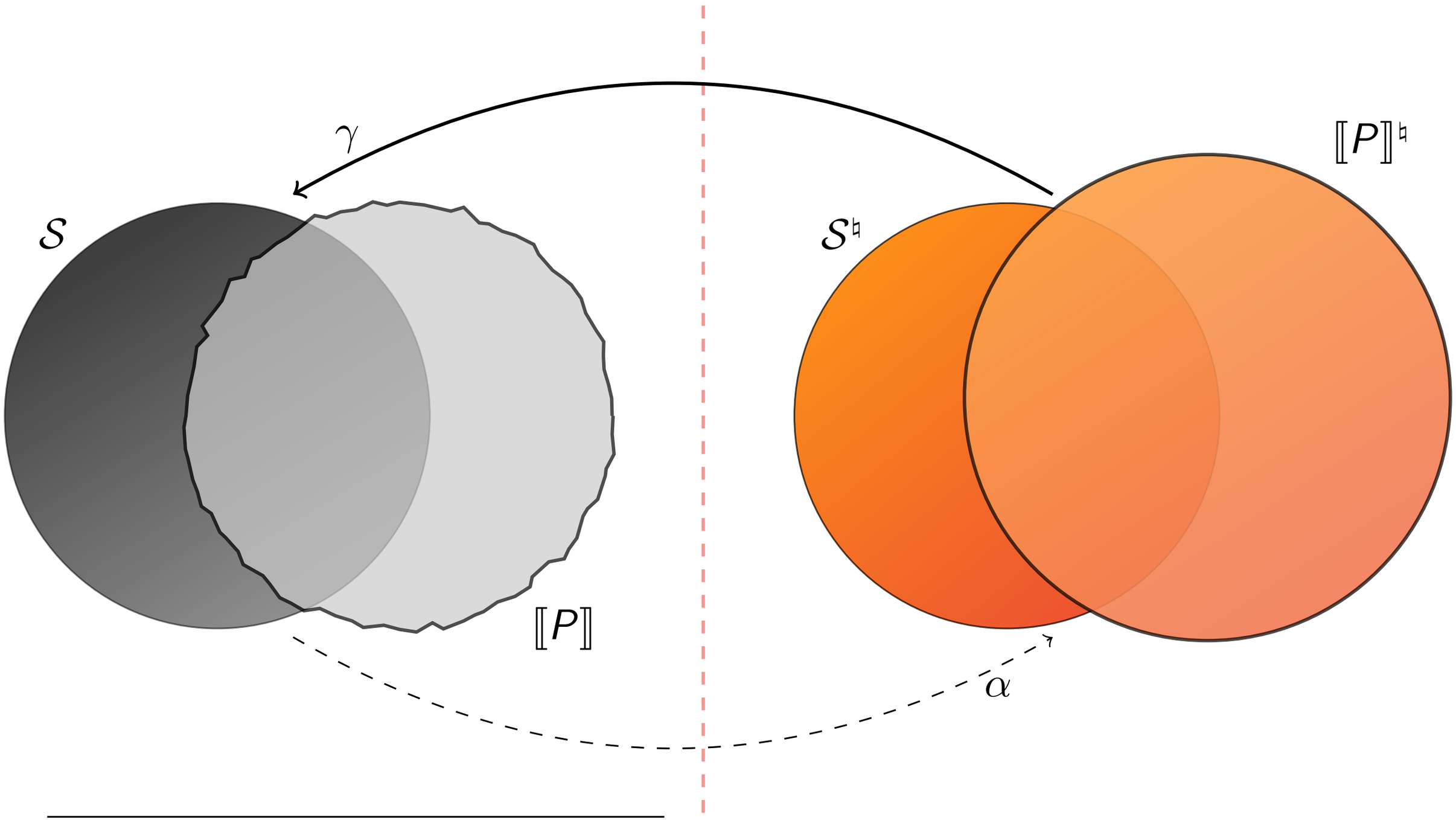
Urban - *The Abstract Domain of Segmented Ranking Functions* (SAS 2013)
Urban&Miné - *An Abstract Domain to Infer Ordinal-Valued Ranking Functions* (ESOP 2014)
Urban&Miné - *A Decision Tree Abstract Domain for Proving Conditional Termination* (SAS 2014)

Abstract Interpretation



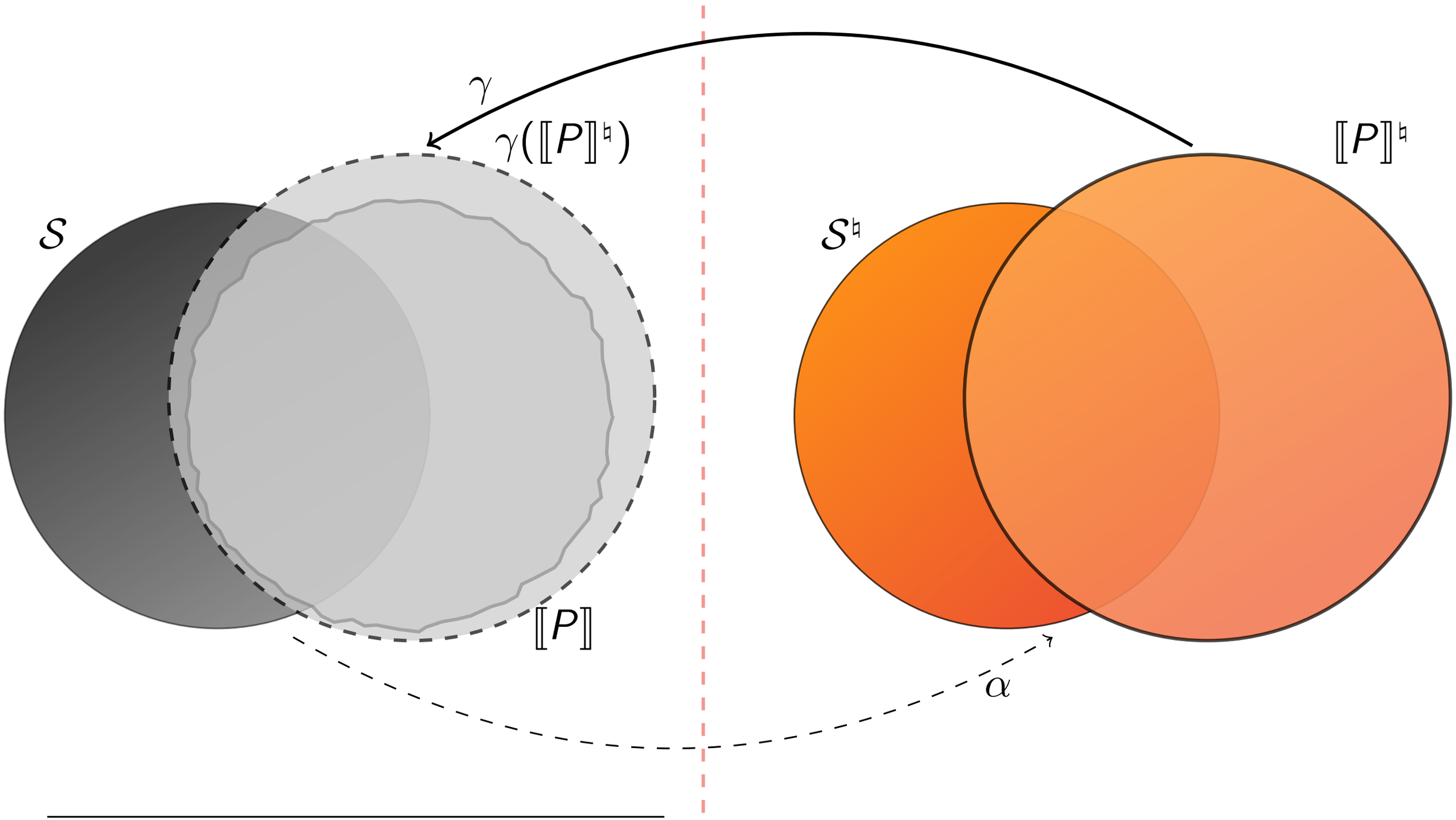
Cousot&Cousot - *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints* (POPL 1977)

Abstract Interpretation

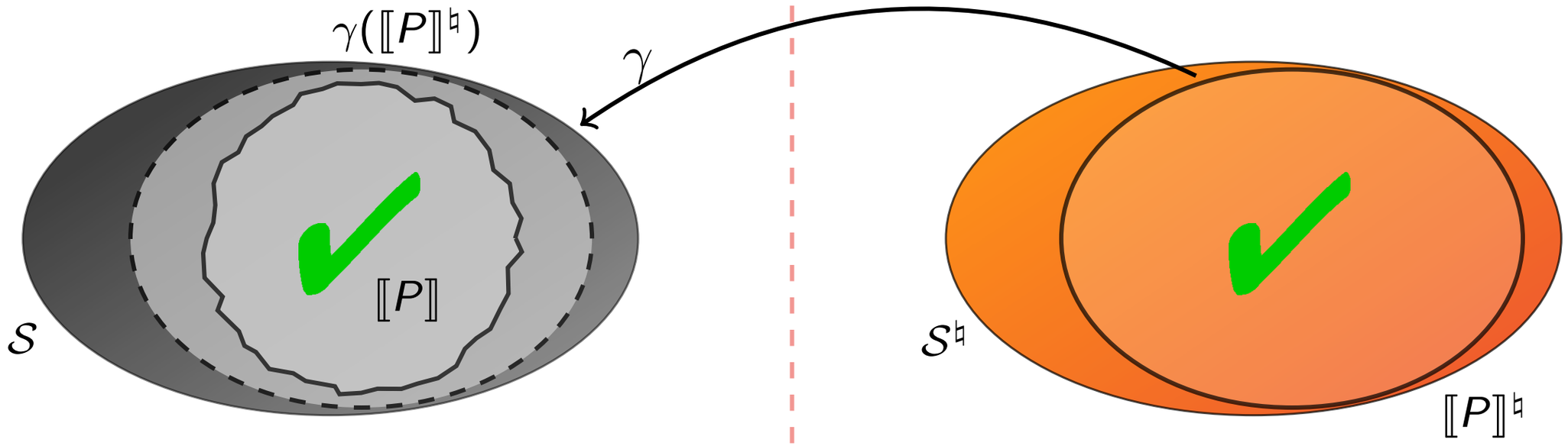


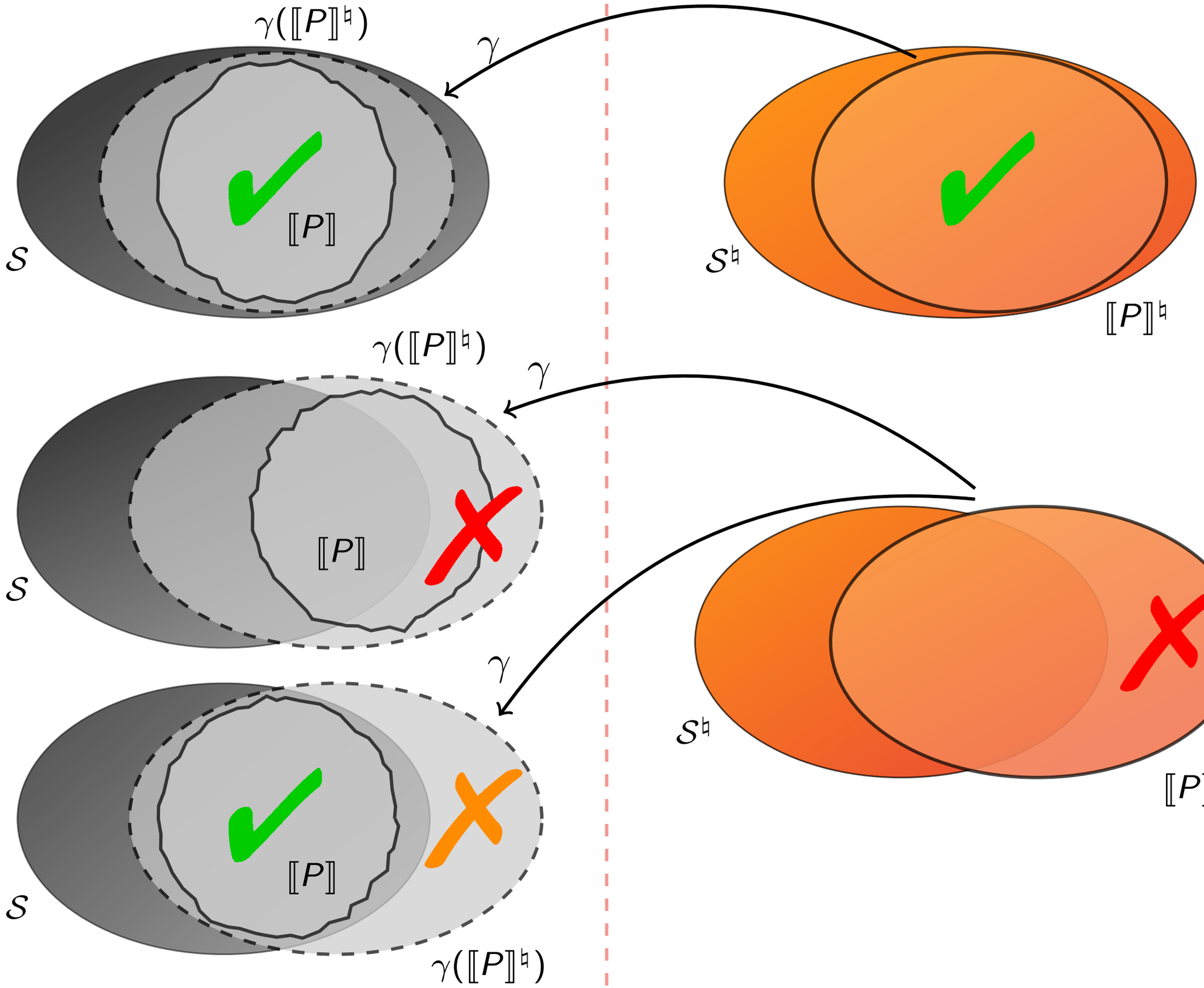
Cousot&Cousot - *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints* (POPL 1977)

Abstract Interpretation



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


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


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

Termination

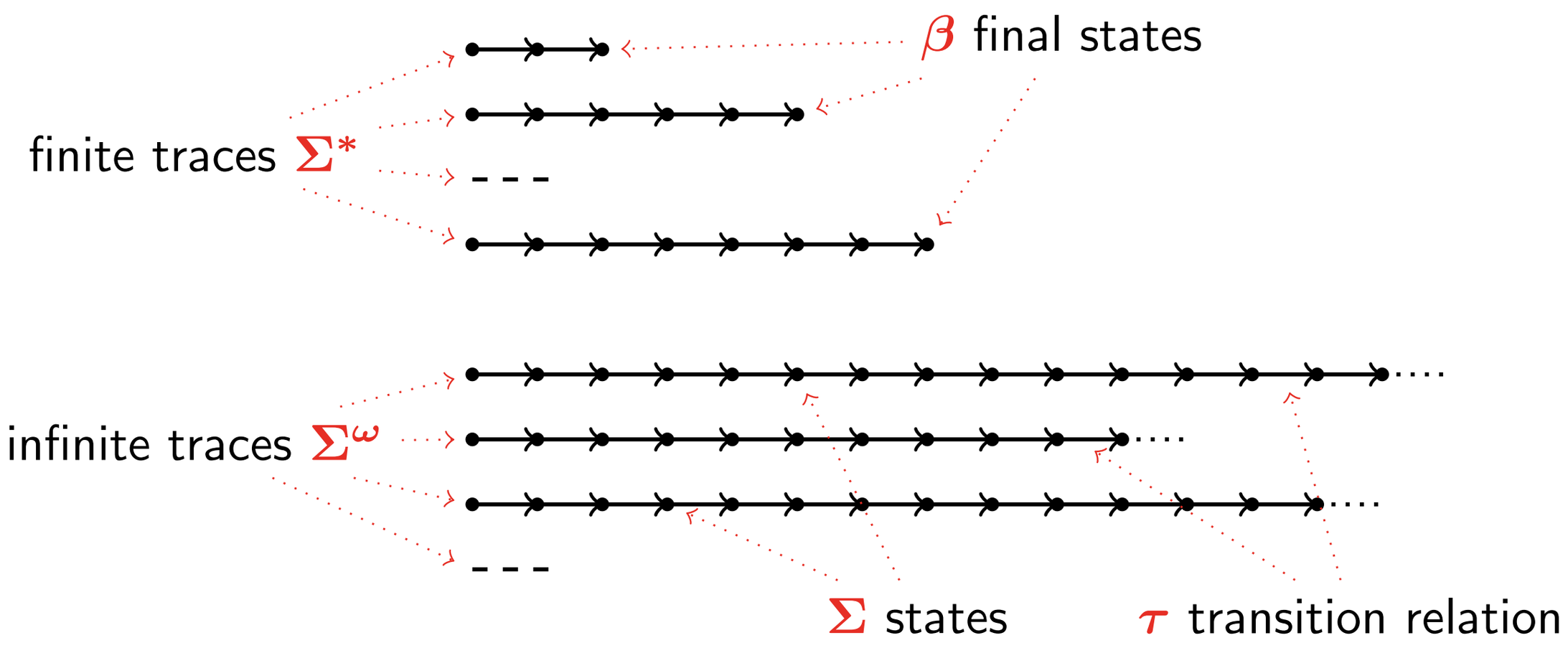
An Abstract Interpretation Framework for Termination

Patrick Cousot
CNRS, École Normale Supérieure, and INRIA, France
Courant Institute*, NYU, USA
pcousot@ens.fr, pcousot@cims.nyu.edu

Radhia Cousot
CNRS, École Normale Supérieure, and INRIA, France
rcousot@ens.fr

Abstract

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



Termination Semantics

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

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

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

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

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

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

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

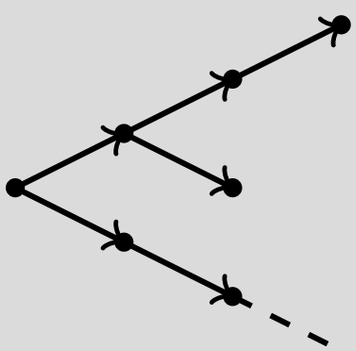
$$\mathcal{T}_t \in \Sigma \rightharpoonup \mathbb{O}$$

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

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

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

Example



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

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

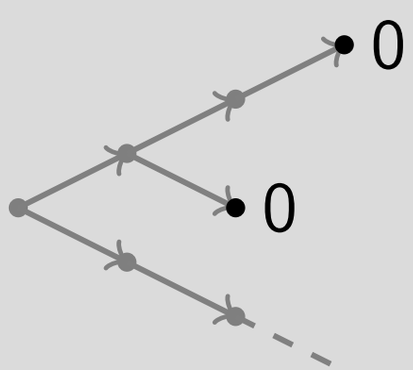
$$\mathcal{T}_t \in \Sigma \rightharpoonup \mathbb{O}$$

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

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

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

Example



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

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

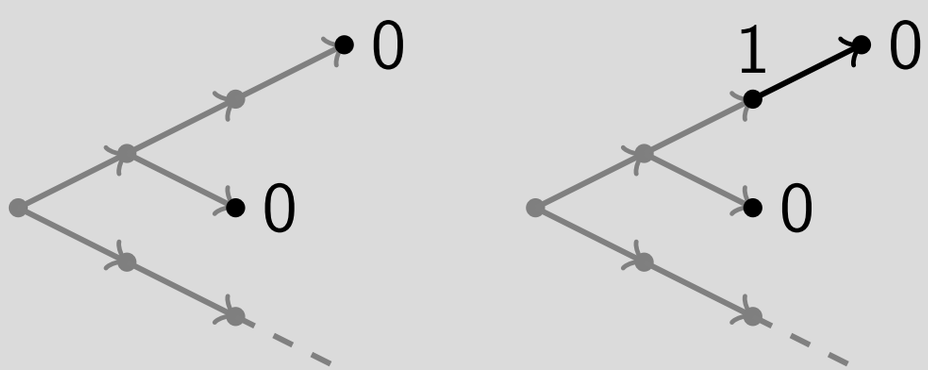
$$\mathcal{T}_t \in \Sigma \rightharpoonup \mathbb{O}$$

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

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

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

Example



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

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

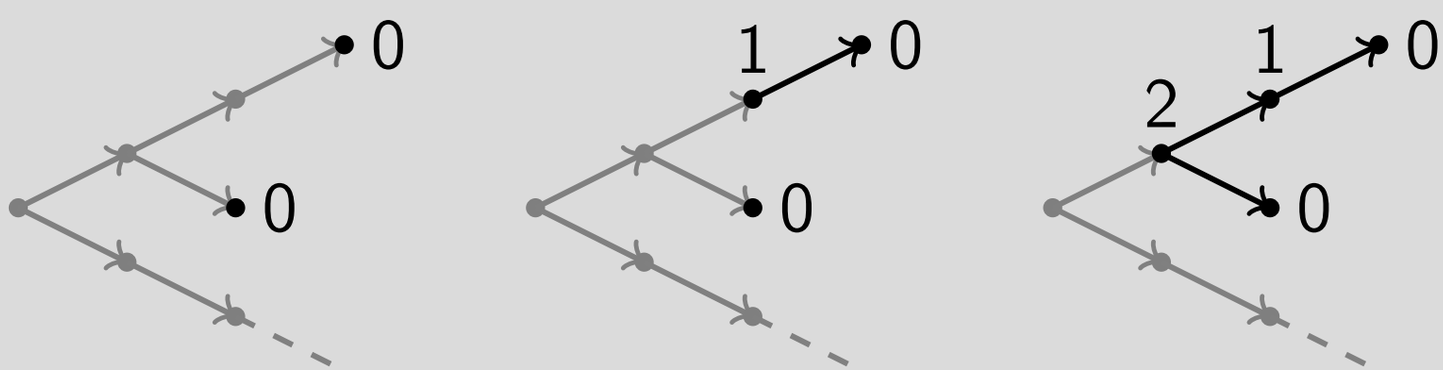
$$\mathcal{T}_t \in \Sigma \rightharpoonup \mathbb{O}$$

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

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

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

Example



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

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

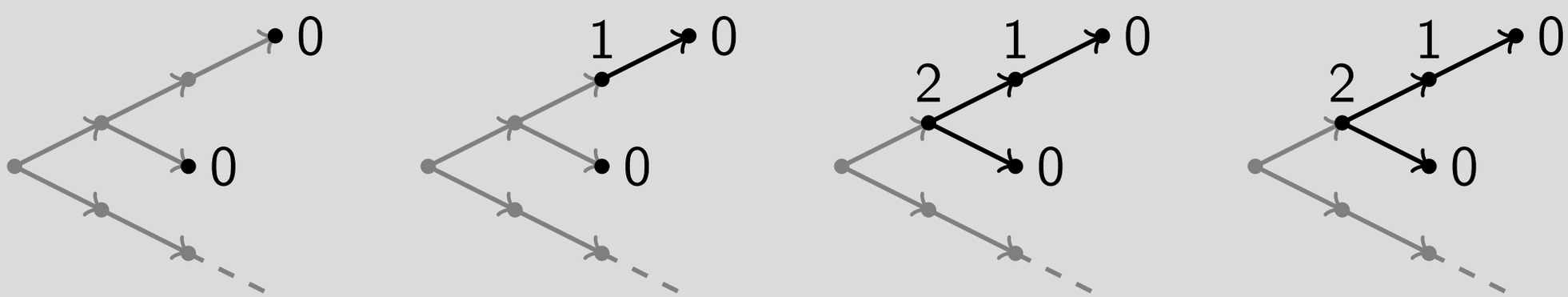
$$\mathcal{T}_t \in \Sigma \rightharpoonup \mathbb{O}$$

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

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

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

Example



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

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

Example

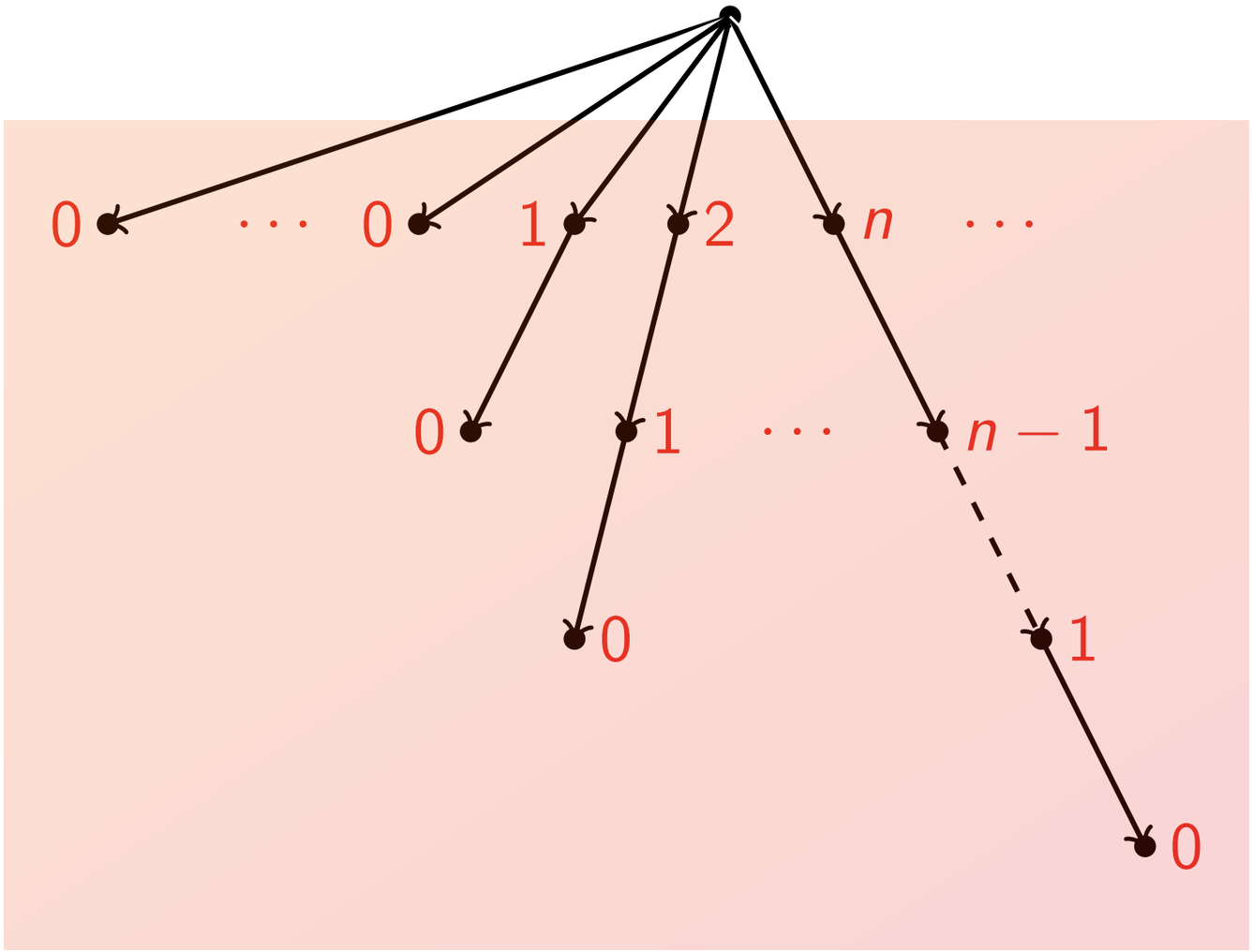
int : x

$x :=$?

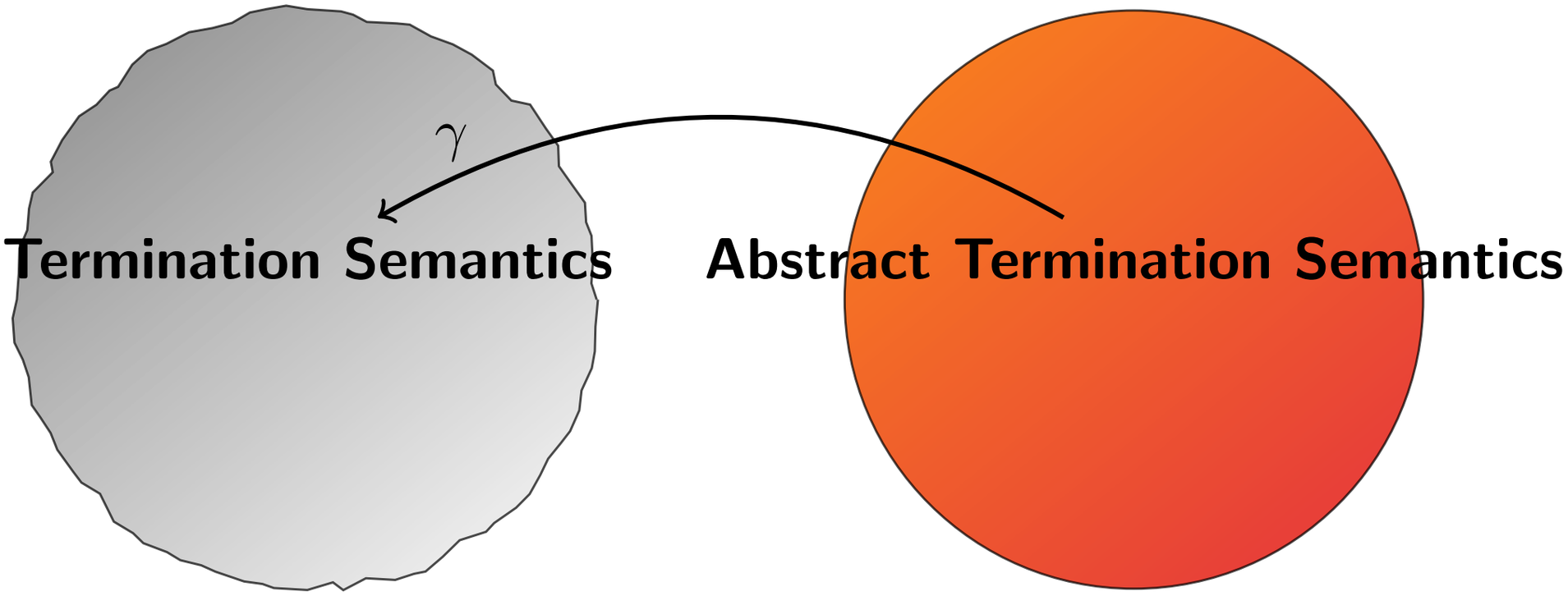
while $(x > 0)$ do

$x := x - 1$

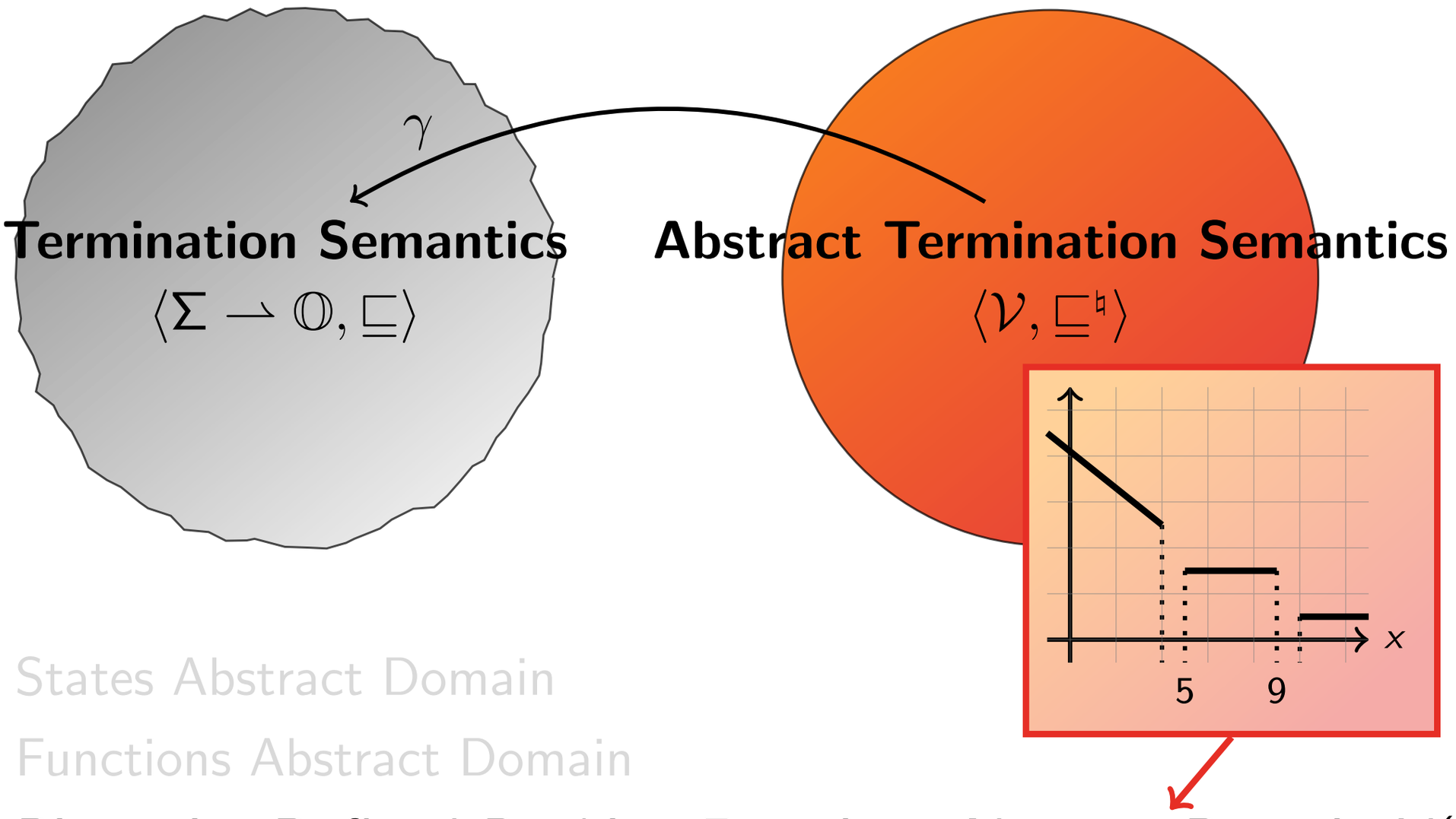
od



Piecewise-Defined Ranking Functions

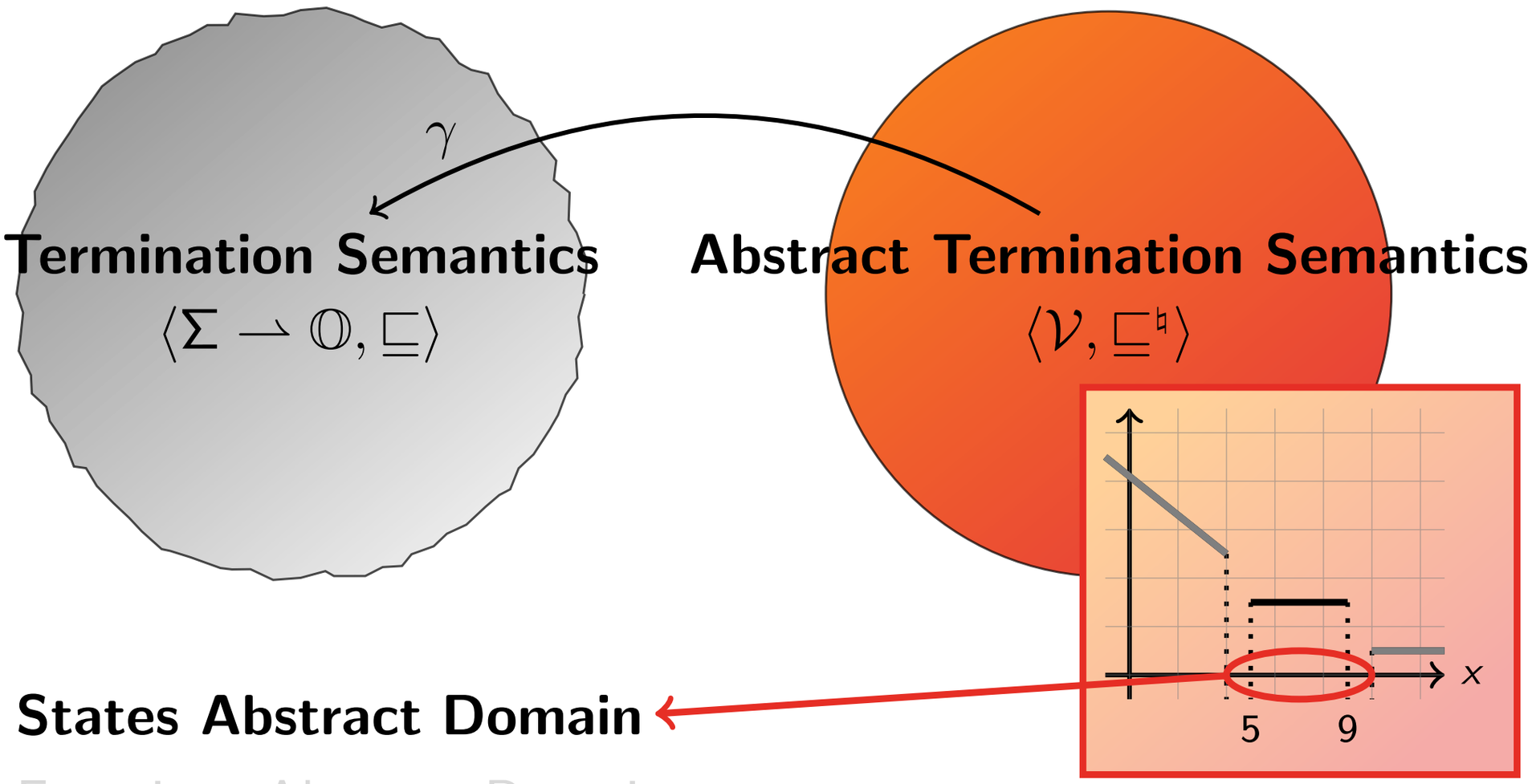


Urban - *The Abstract Domain of Segmented Ranking Functions* (SAS 2013)



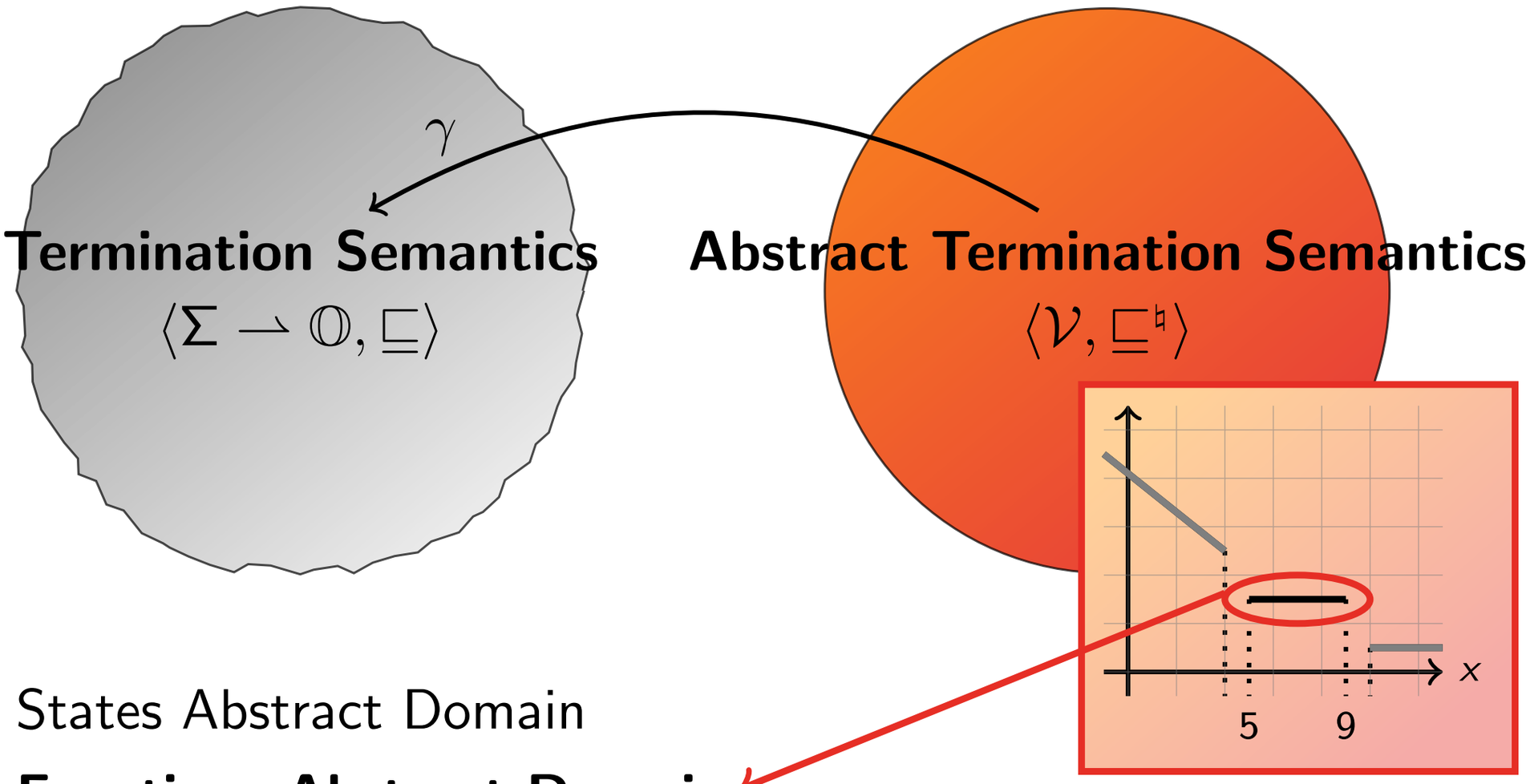
- States Abstract Domain S
- Functions Abstract Domain F
- **Piecewise-Defined Ranking Functions Abstract Domain** V(S, F)

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



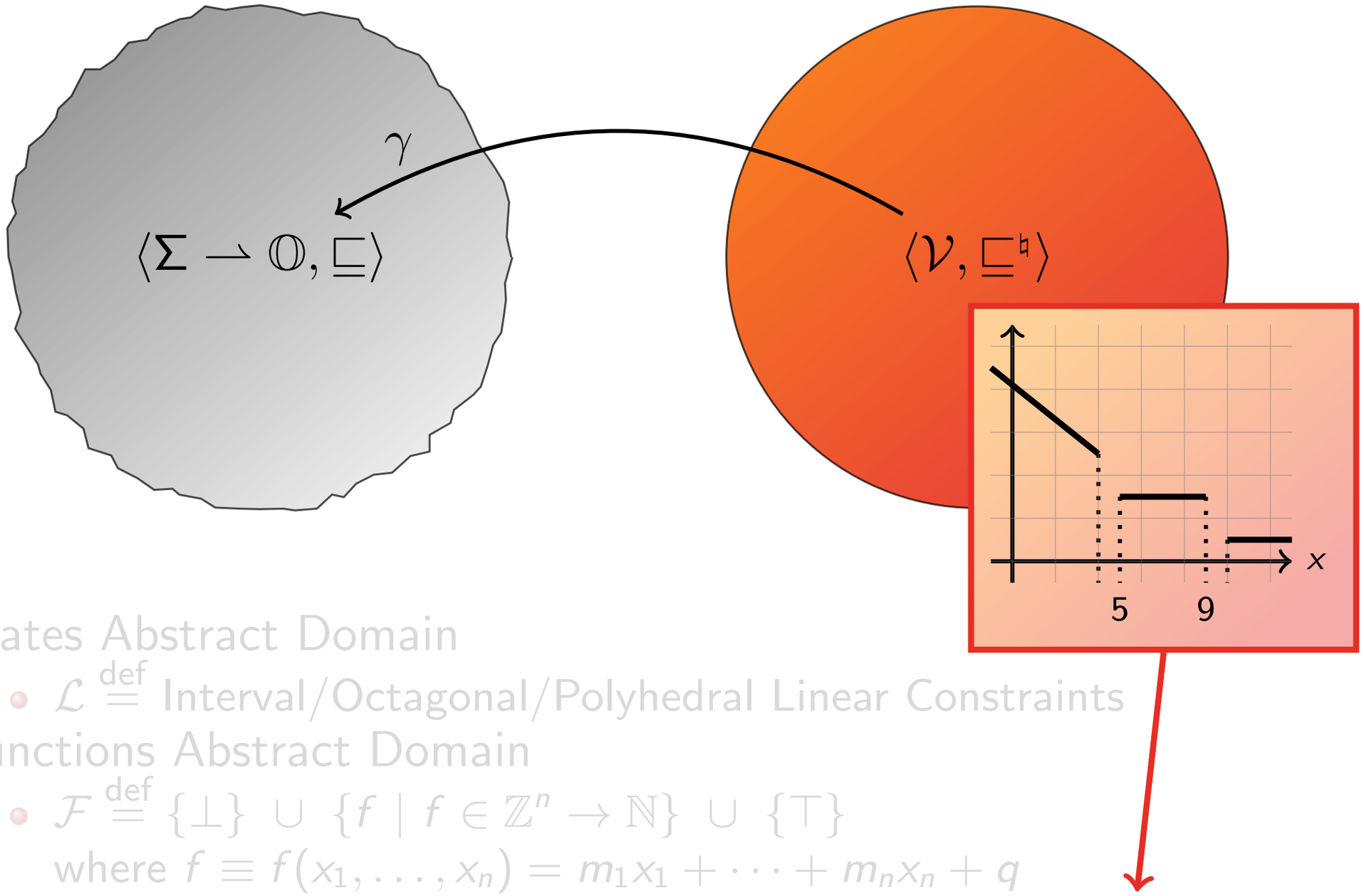
- **States Abstract Domain** S
- Functions Abstract Domain F
- Piecewise-Defined Ranking Functions Abstract Domain V(S, F)

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



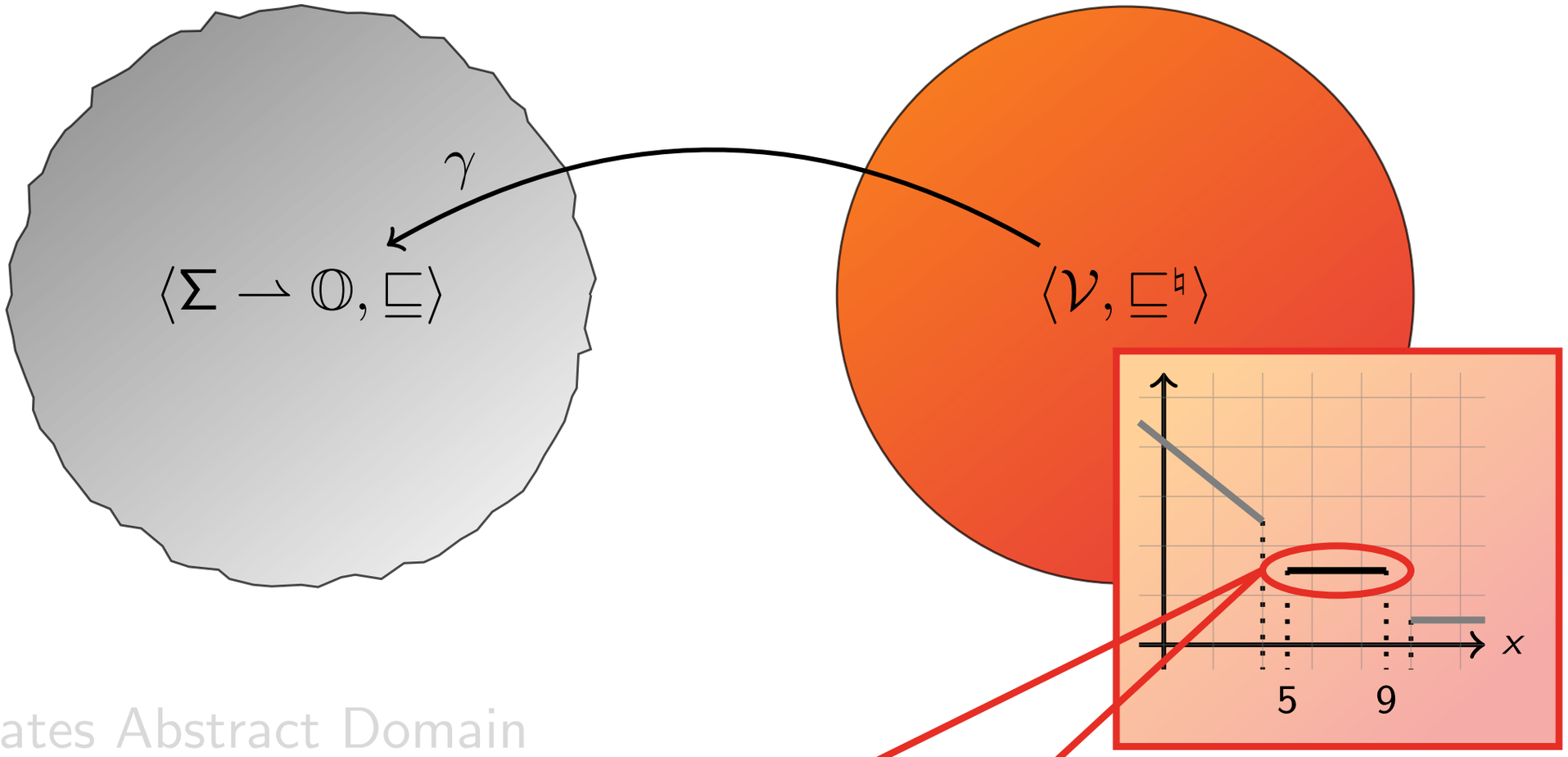
- States Abstract Domain S
- **Functions Abstract Domain** F
- Piecewise-Defined Ranking Functions Abstract Domain V(S, F)

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



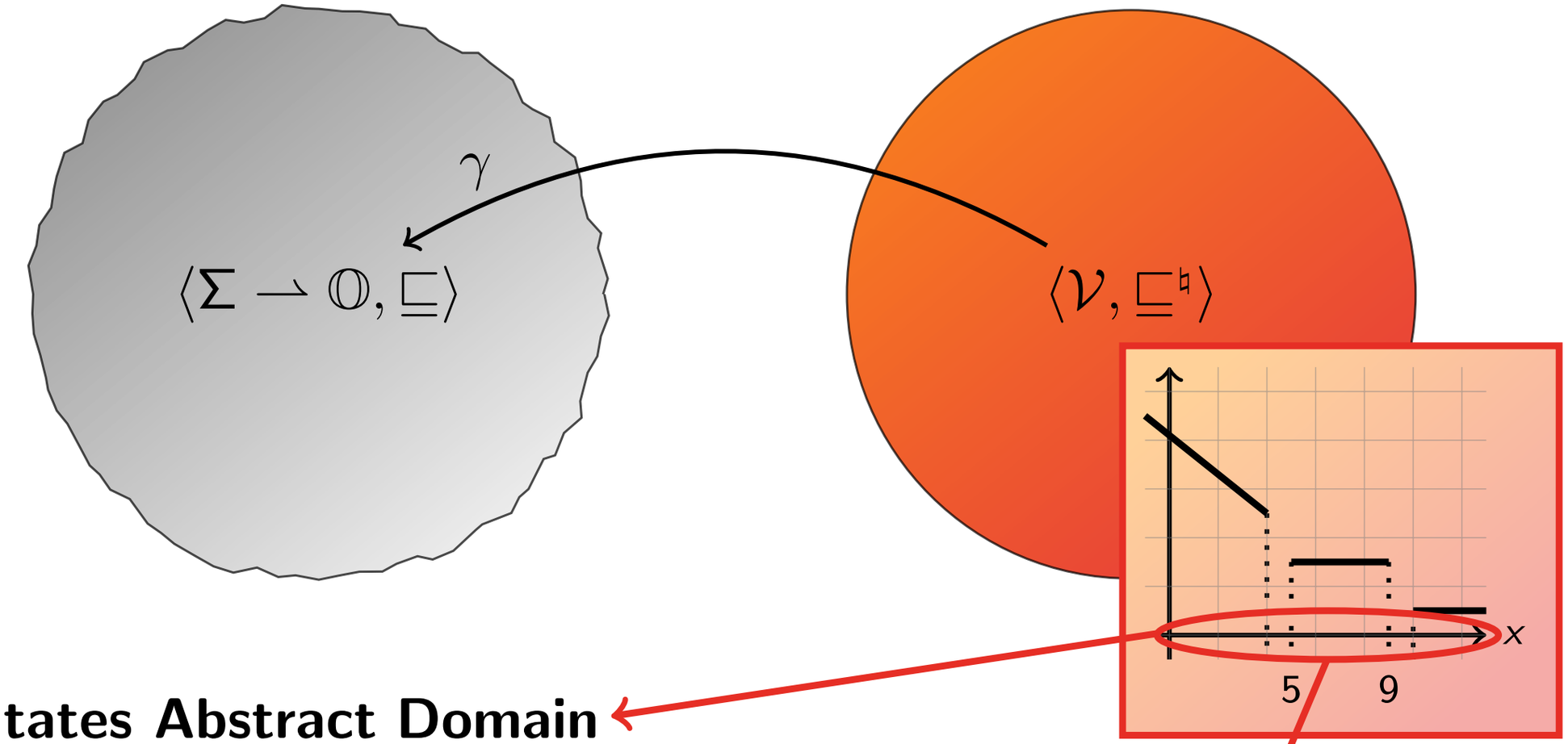
- States Abstract Domain
  - $\mathcal{L} \stackrel{\text{def}}{=}$ Interval/Octagonal/Polyhedral Linear Constraints
- Functions Abstract Domain
  - $\mathcal{F} \stackrel{\text{def}}{=} \{\bot\} \cup \{f \mid f \in \mathbb{Z}^n \to \mathbb{N}\} \cup \{\top\}$
    where $f \equiv f(x_1, \ldots, x_n)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

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



- States Abstract Domain
  - $\mathcal{L} \stackrel{\text{def}}{=}$ Interval/Octagonal/Polyhedral Linear Constraints
- **Functions Abstract Domain**
  - $\mathcal{F} \stackrel{\text{def}}{=} \{\bot\} \cup \{f \mid f \in \mathbb{Z}^n \to \mathbb{N}\} \cup \{\top\}$
    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
  - $\mathcal{V} \stackrel{\text{def}}{=} \{\textbf{LEAF} : f \mid f \in \mathcal{F}\} \cup \{\text{NODE}\{c\} : t_1, t_2 \mid c \in \mathcal{L} \wedge t_1, t_2 \in \mathcal{V}\}$

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



- **States Abstract Domain**
  - $\mathcal{L} \stackrel{\text{def}}{=}$ Interval/Octagonal/Polyhedral Linear Constraints
- Functions Abstract Domain
  - $\mathcal{F} \stackrel{\text{def}}{=} \{\bot\} \cup \{f \mid f \in \mathbb{Z}^n \to \mathbb{N}\} \cup \{\top\}$
    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
  - $\mathcal{V} \stackrel{\text{def}}{=} \{\text{LEAF} : f \mid f \in \mathcal{F}\} \cup \{\textbf{NODE}\{c\} : t_1, t_2 \mid c \in \mathcal{L} \wedge t_1, t_2 \in \mathcal{V}\}$

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

Example

int : $x,\ y$

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

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

od^{3}

the program terminates if and only if it starts with $\boldsymbol{x \leq 0 \vee y > 0}$

Example

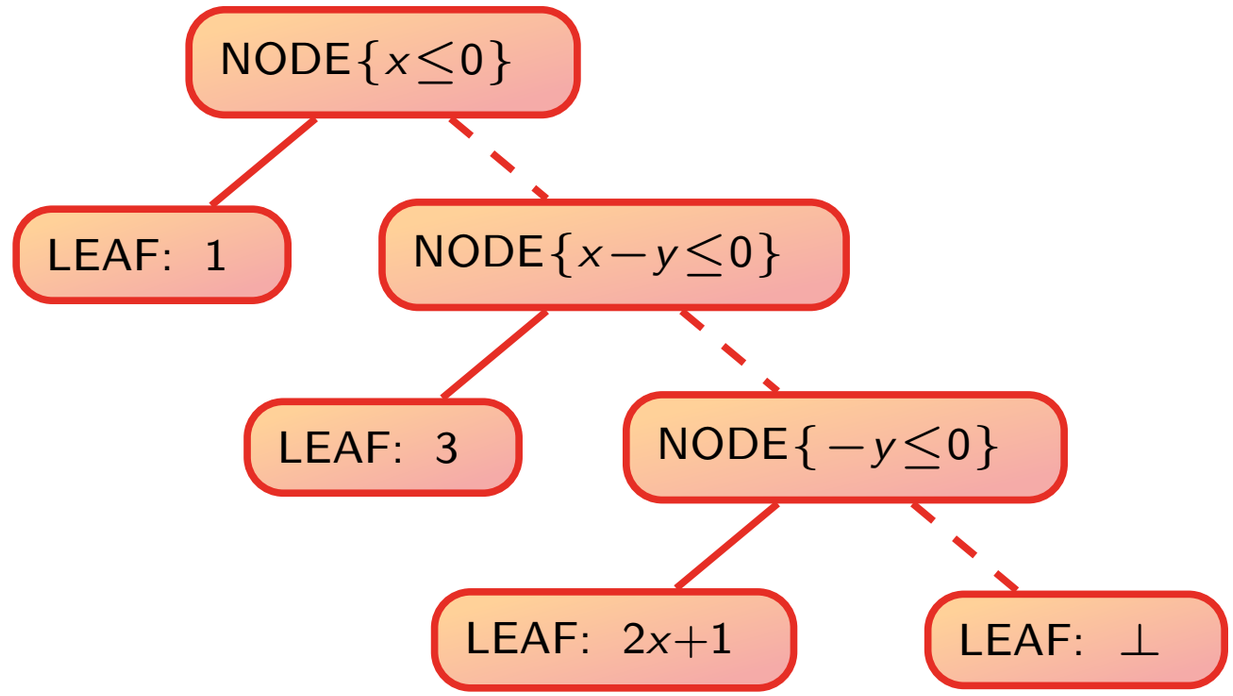
int : $x,\ y$

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

  [2]$x := x - y$

od[3]

the program terminates if and only if it starts with $\boldsymbol{x \leq 0 \vee y > 0}$



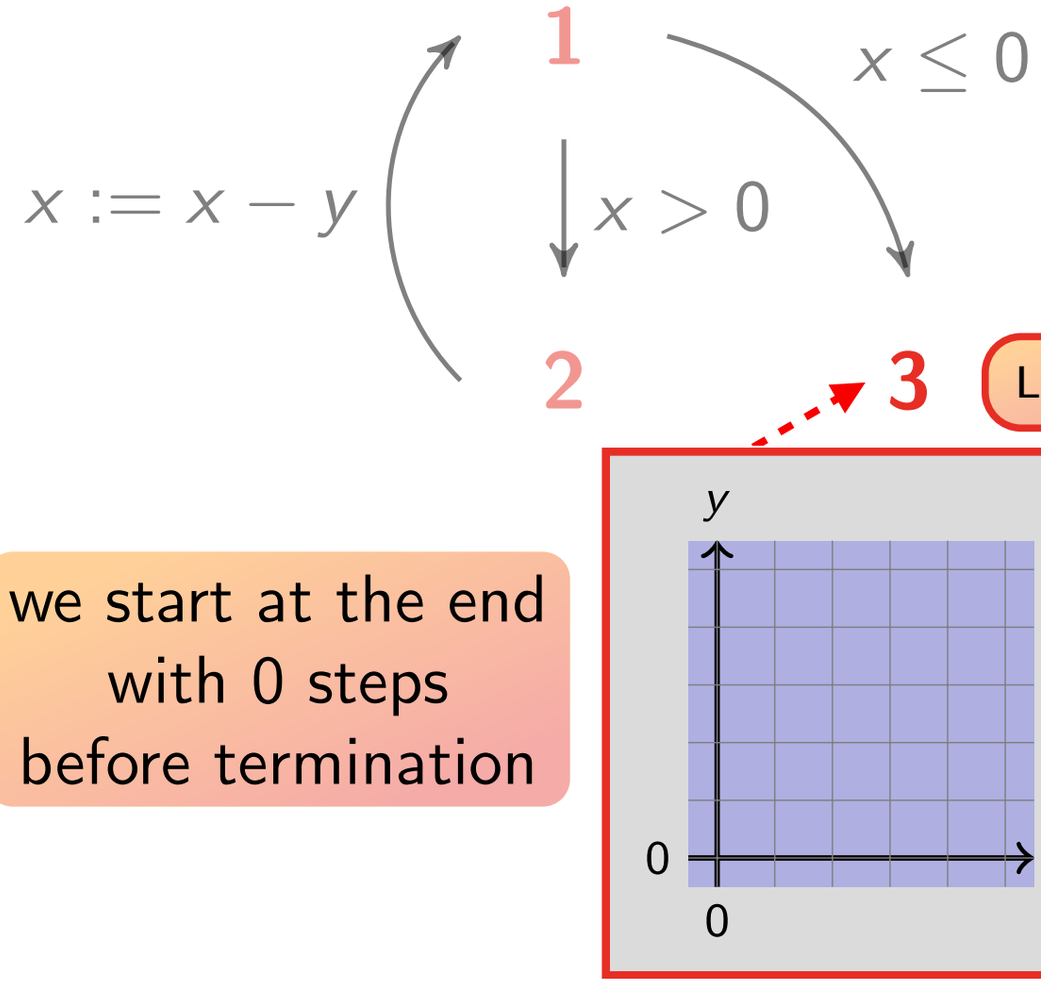
Example

int : $x,\ y$

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

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

od^{3}



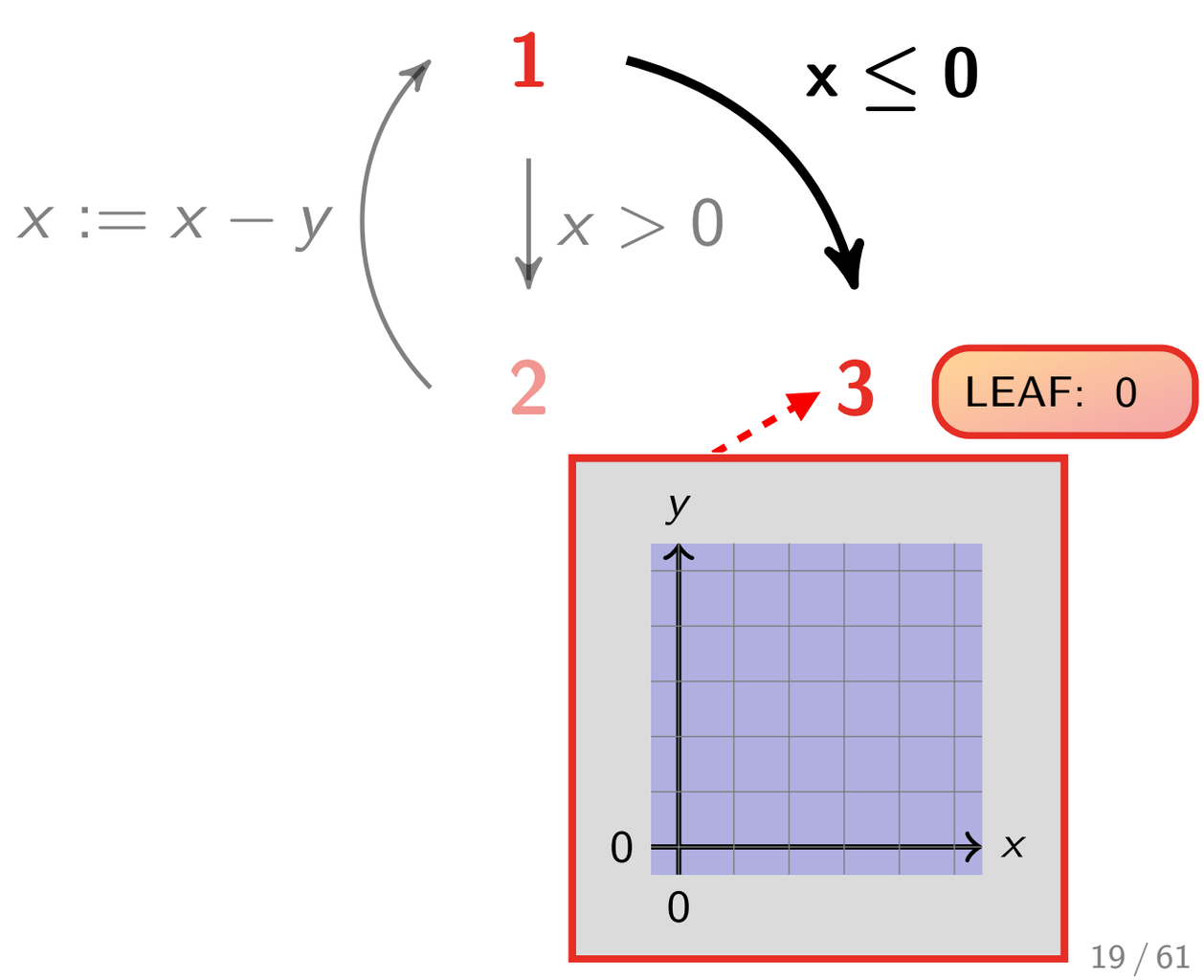
Example

int : $x,\ y$

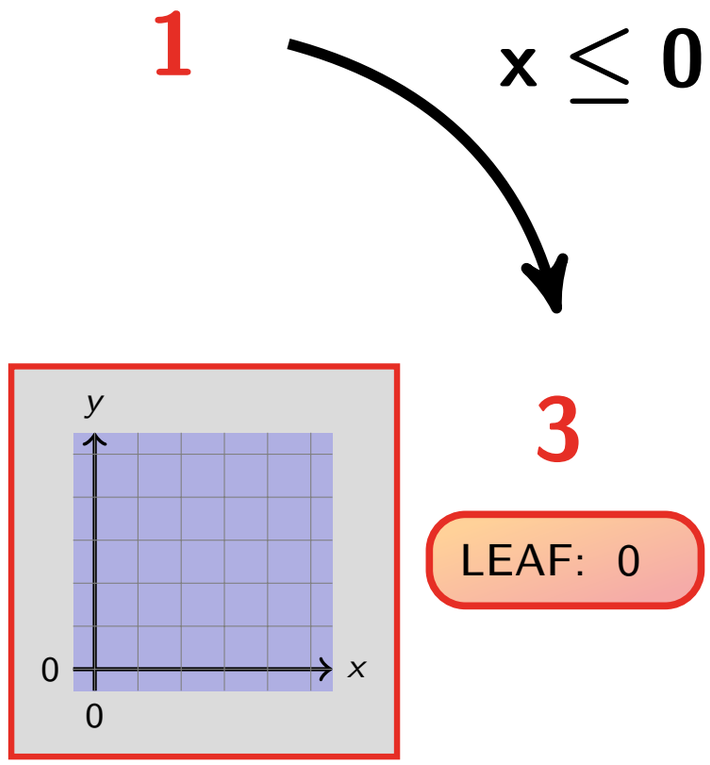
while $^{1}(x > 0)$ do

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

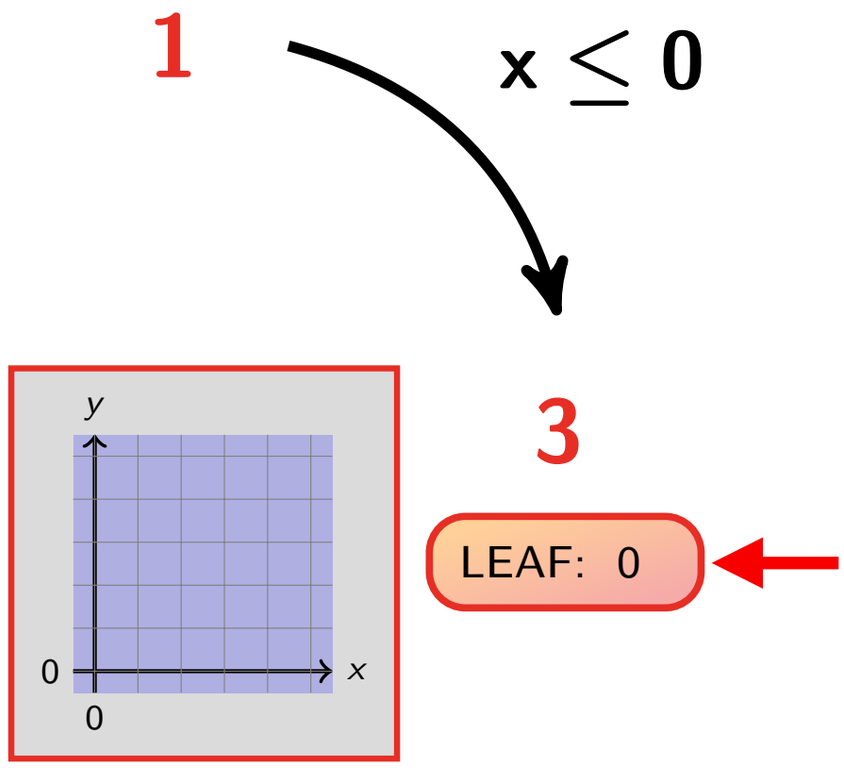
od^{3}



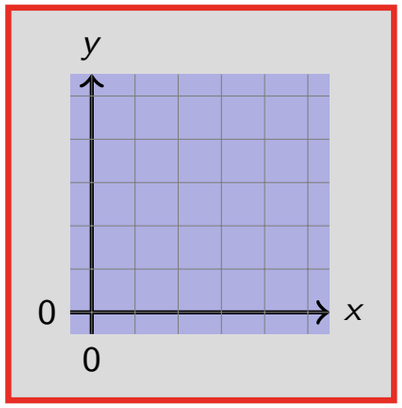
Tests



Tests



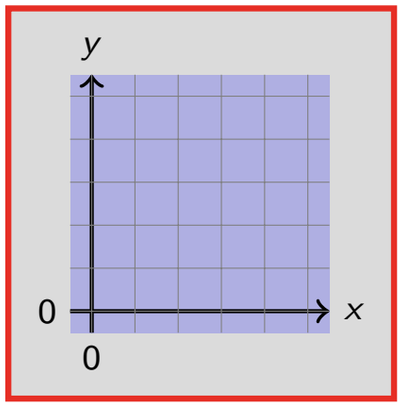
Tests



LEAF: 1

1

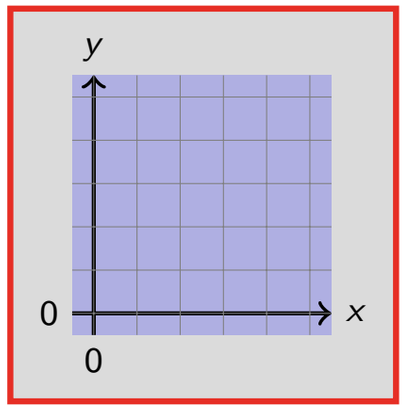
$x \leq 0$



3

LEAF: 0

Tests

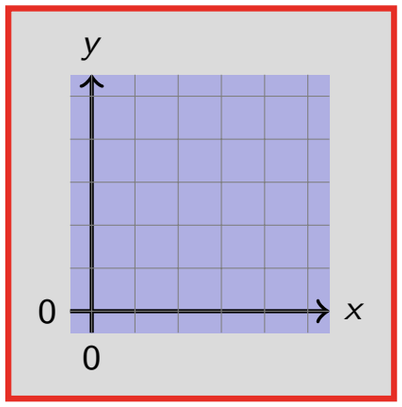


LEAF: 1

1

$x \leq 0$

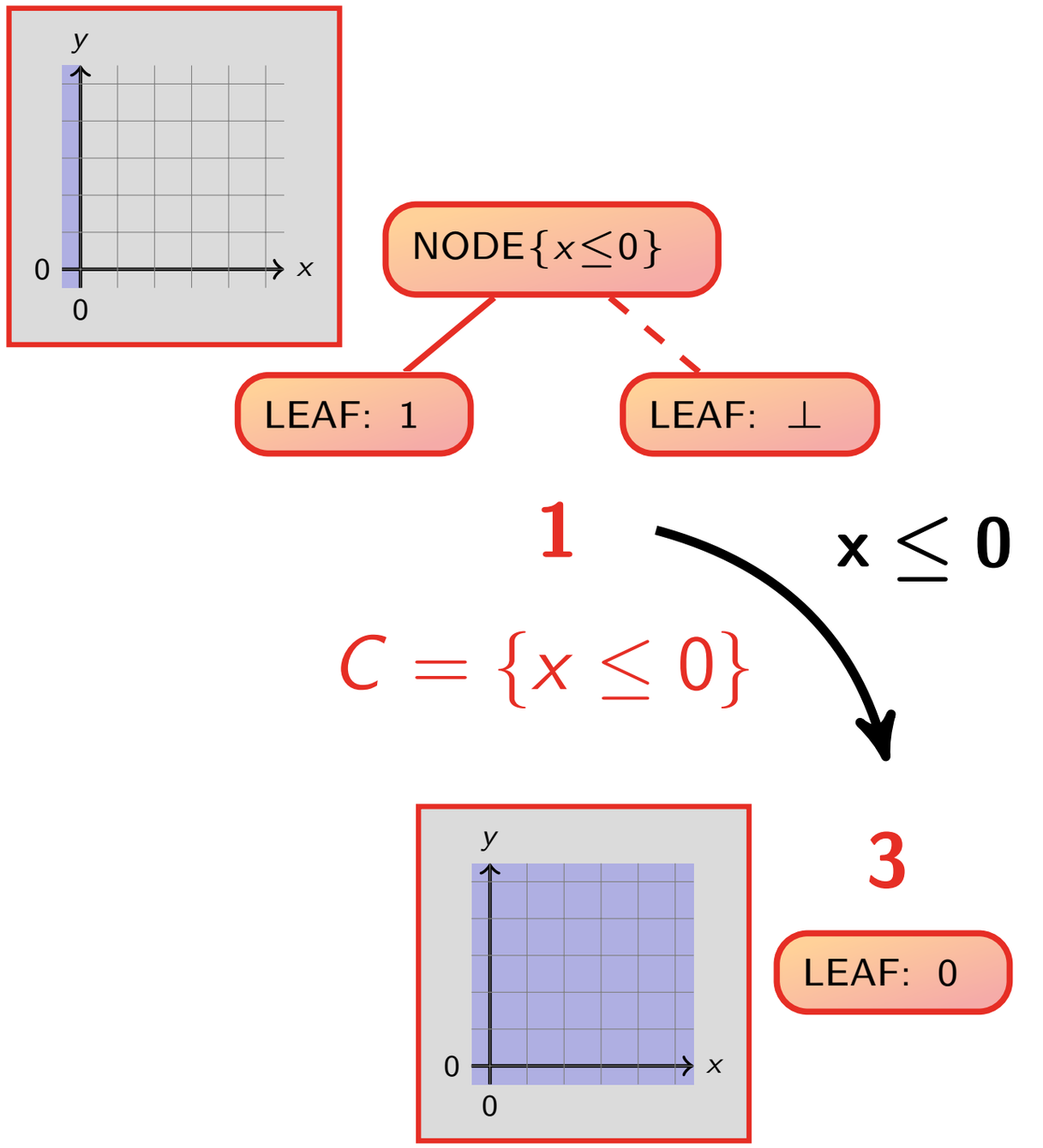
$C = \{x \leq 0\}$



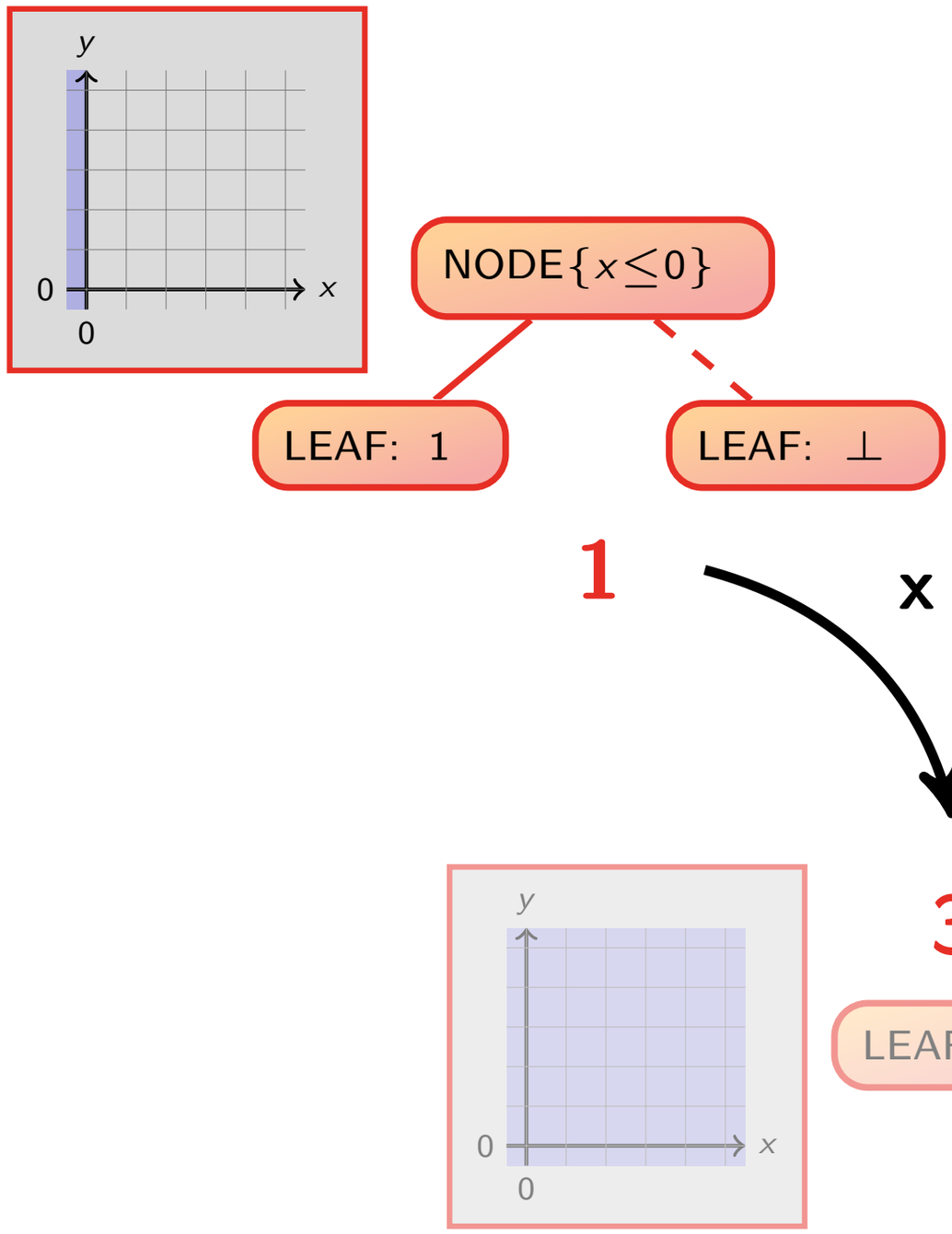
3

LEAF: 0

Tests



Tests



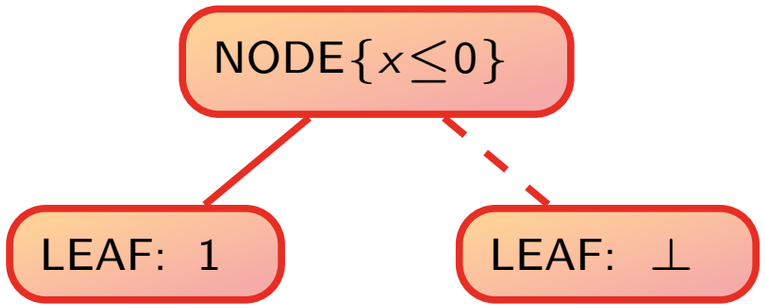
Example

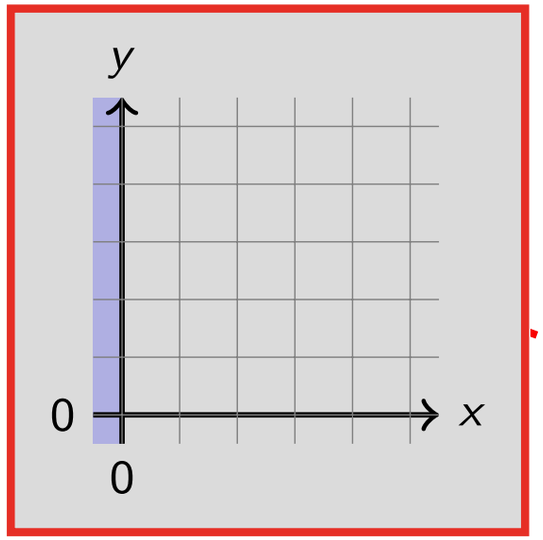
int : x, y

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

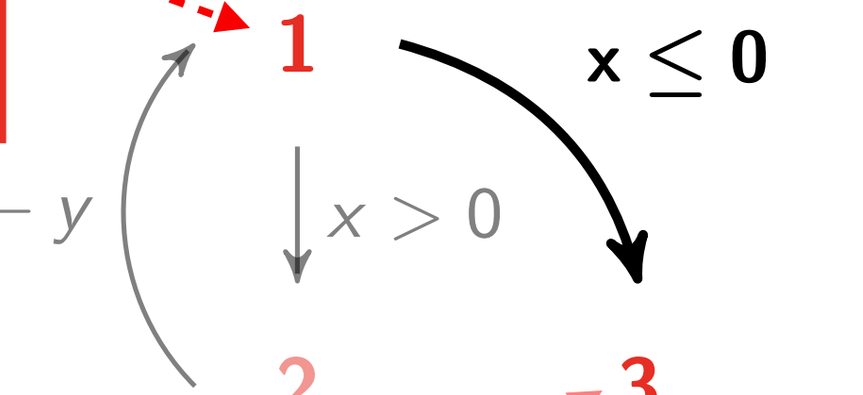
  [2] $x := x - y$

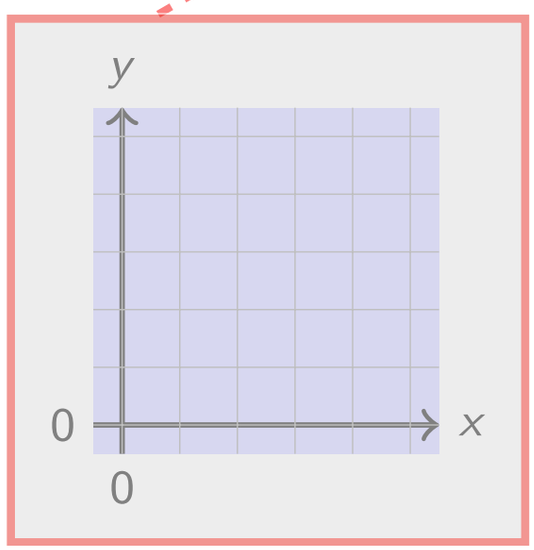
od [3]





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





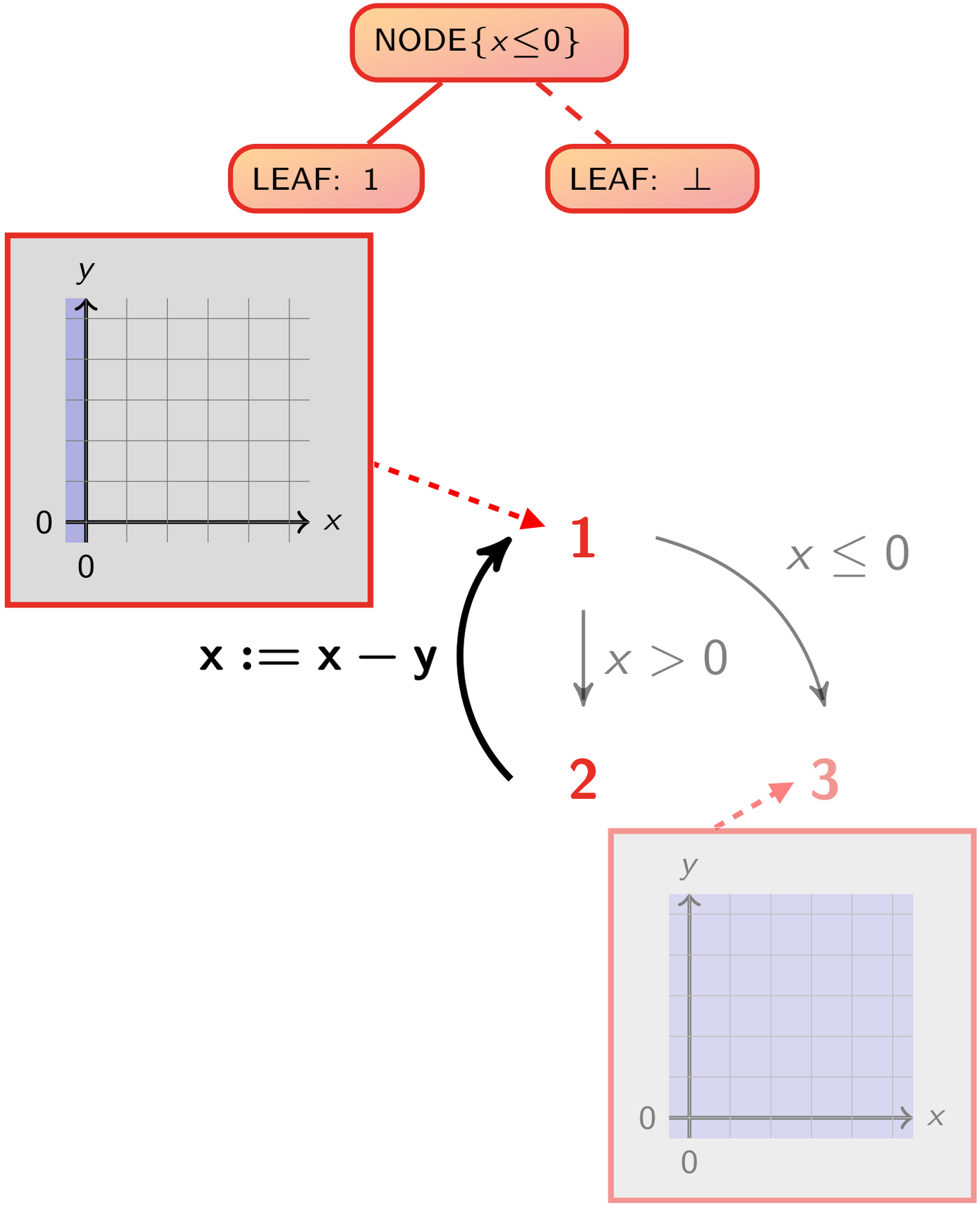
Example

int : x, y

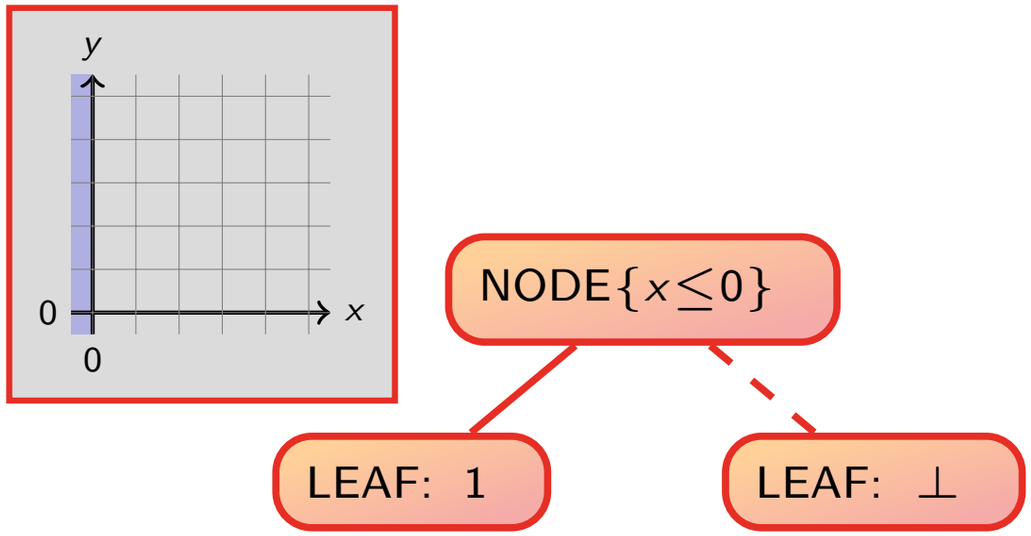
while [1] $(x > 0)$ do

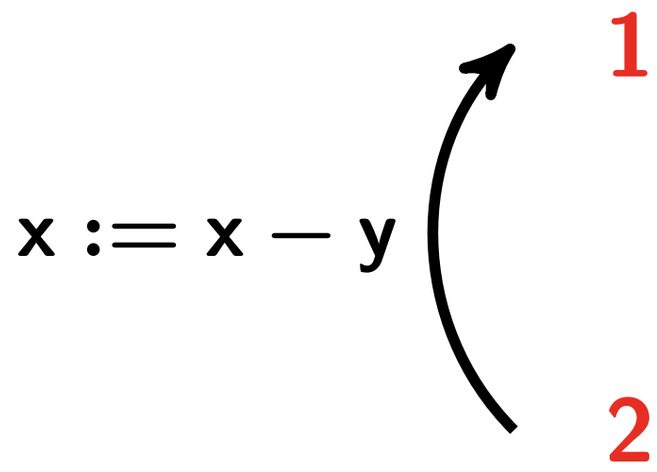
  [2] $x := x - y$

od [3]

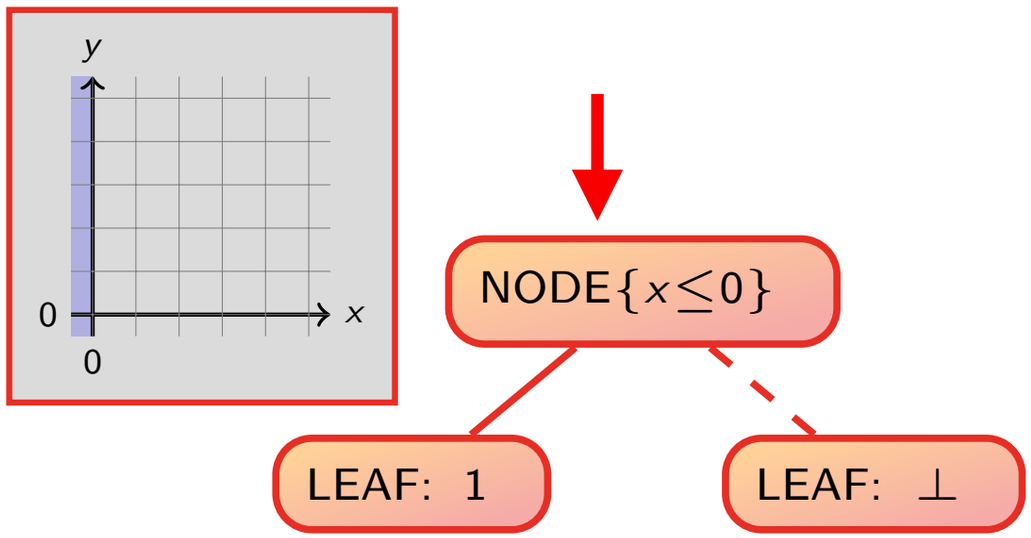


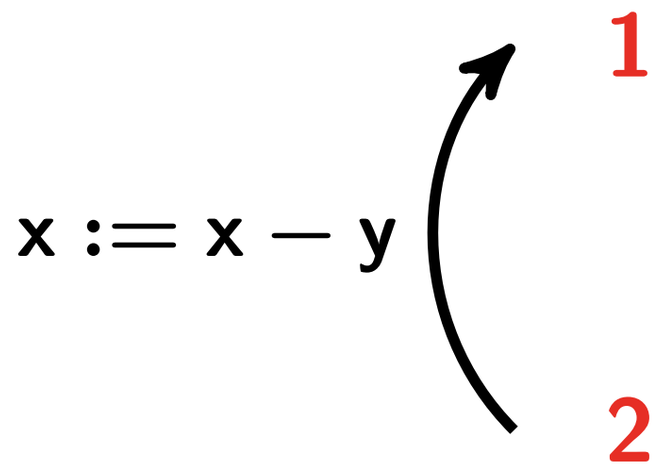
Assignments



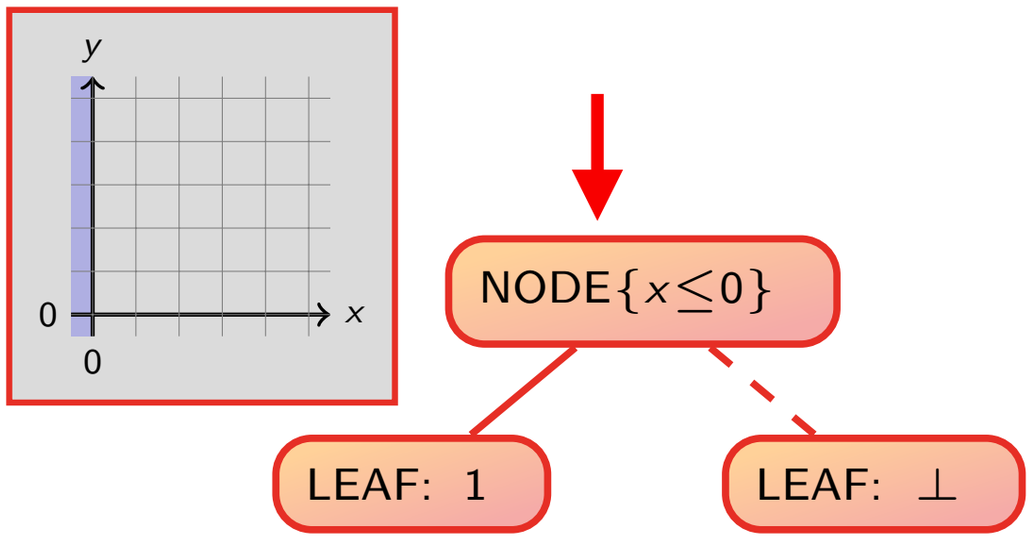


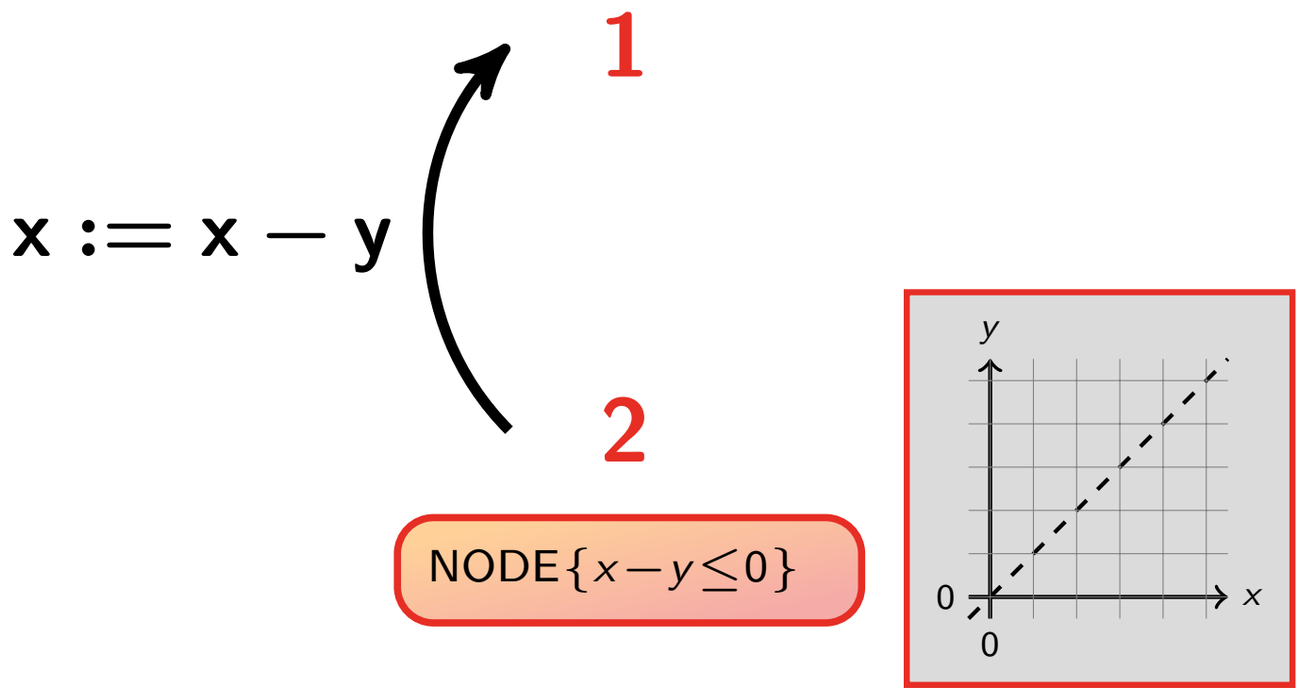
Assignments



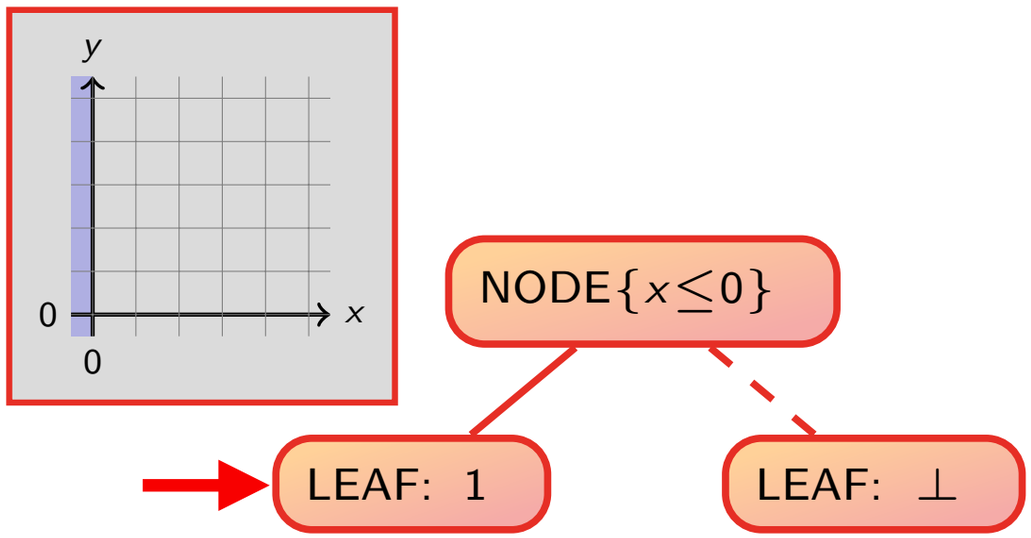


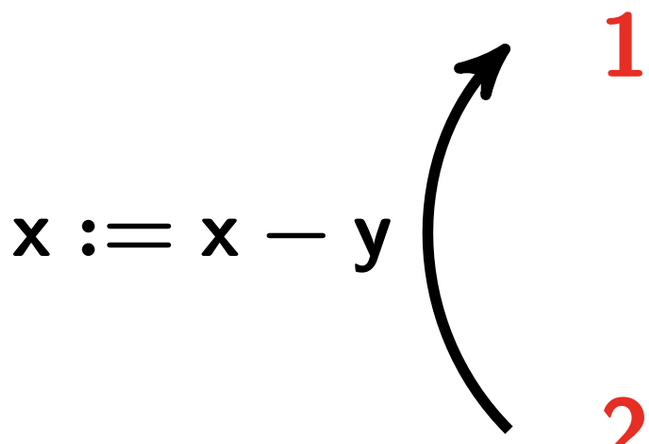
Assignments



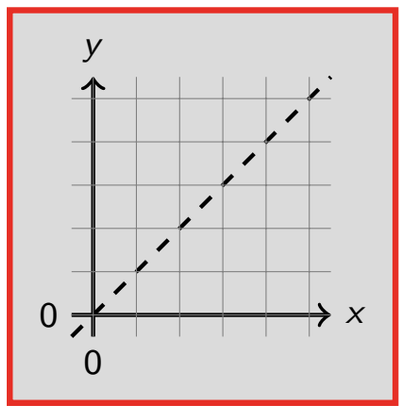


Assignments

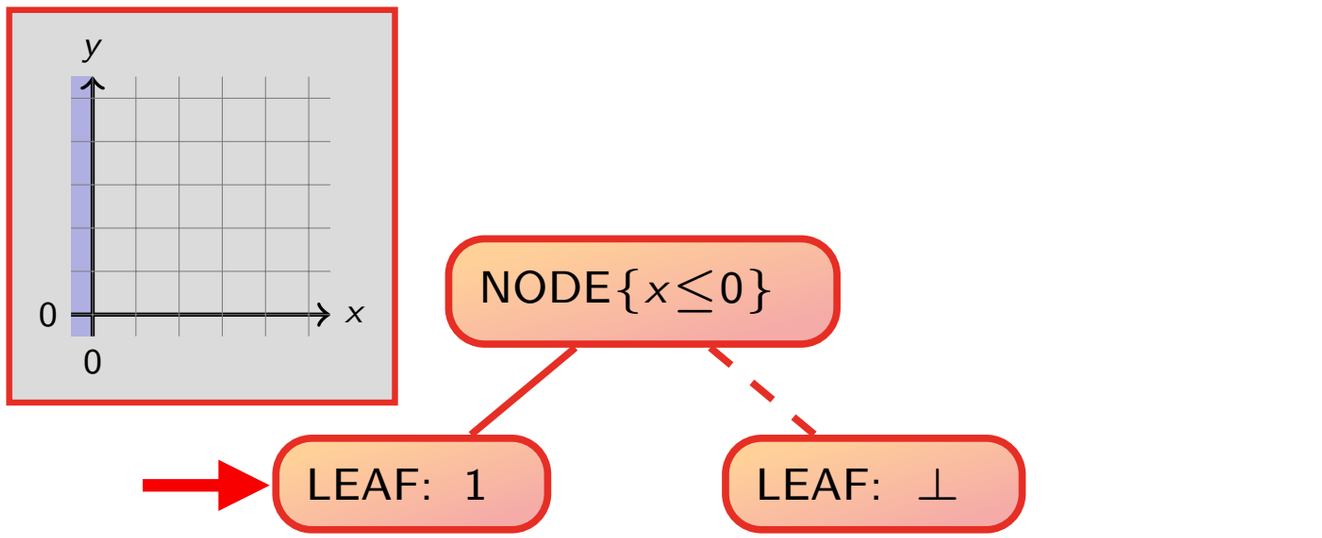


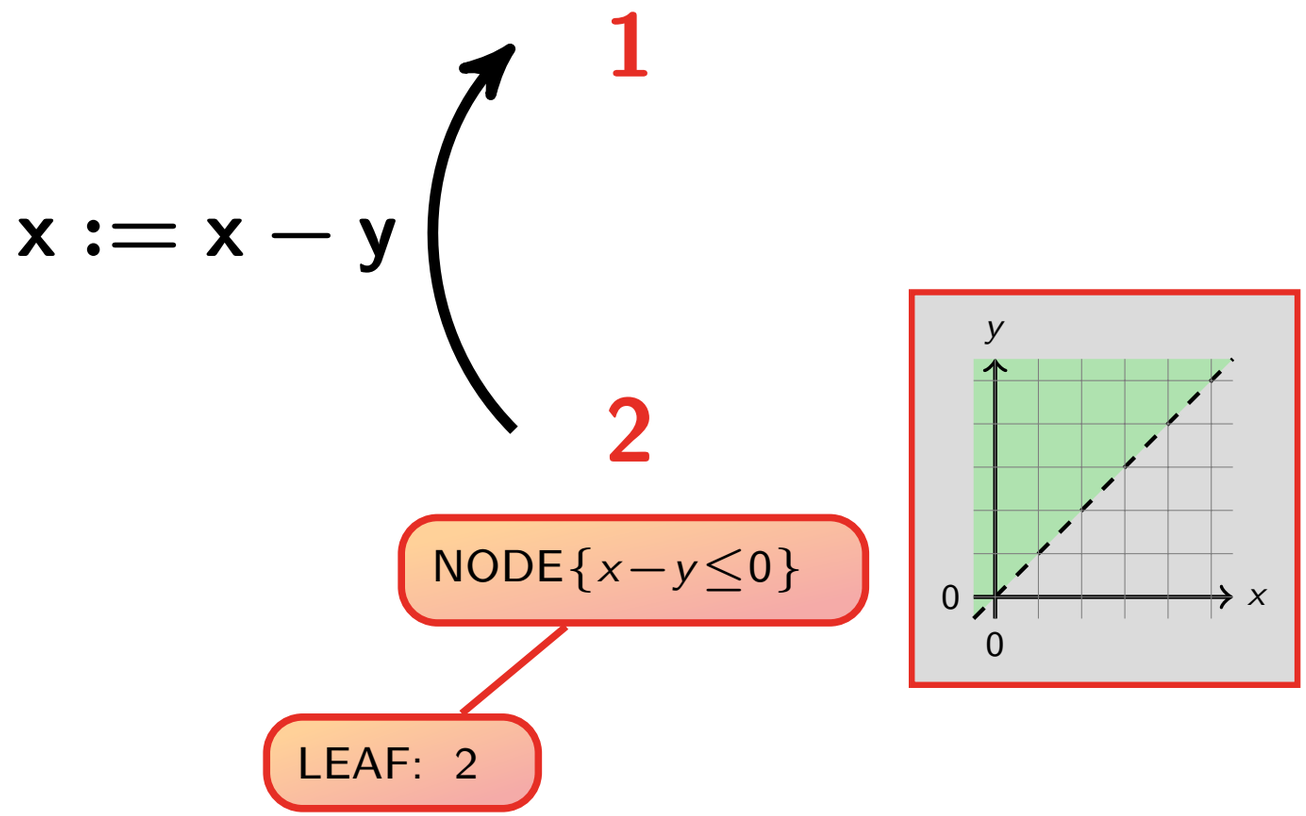


NODE{$x-y\leq 0$}

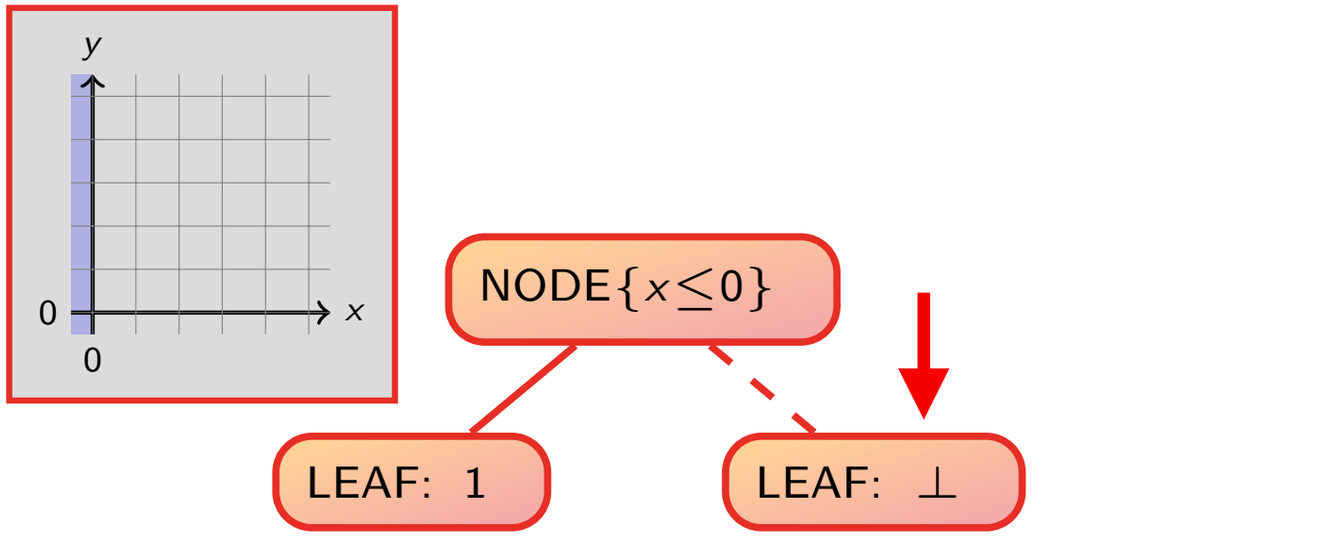


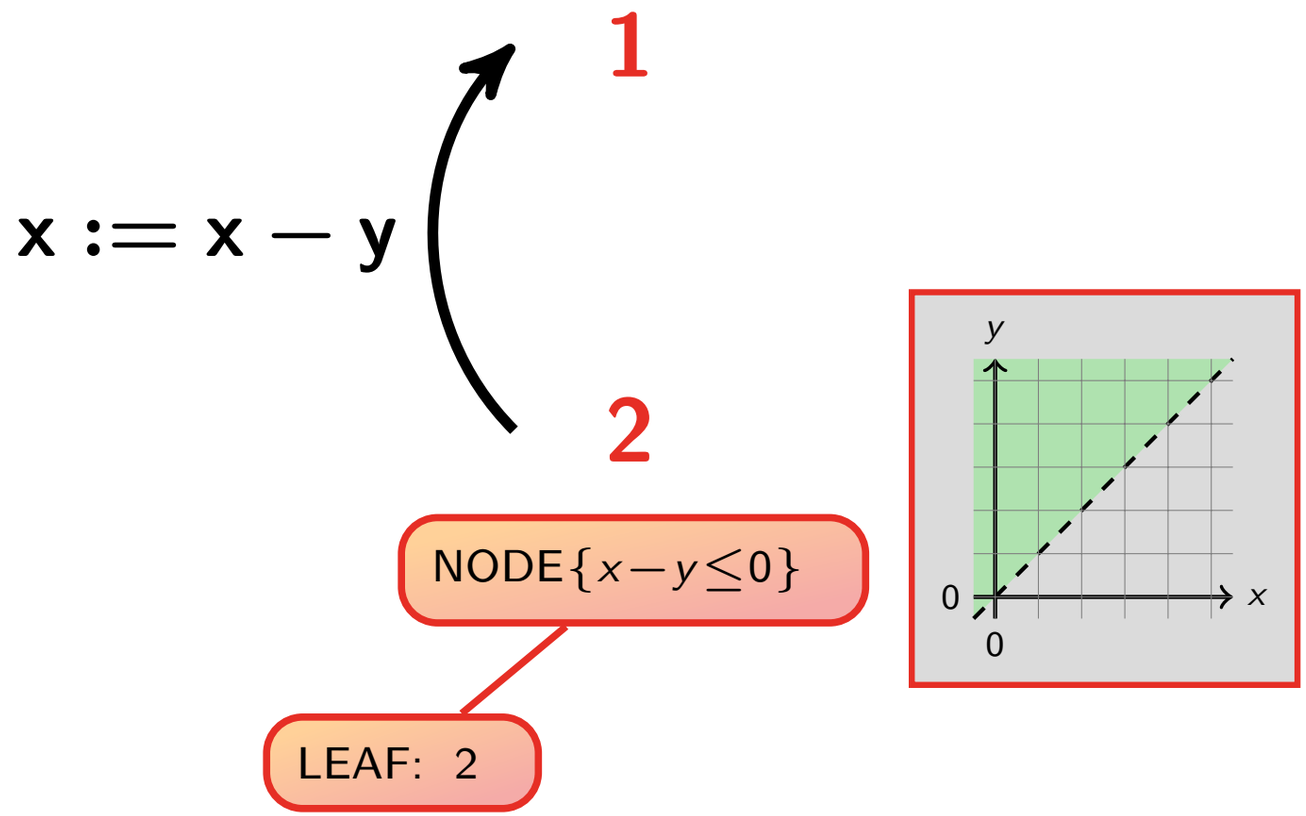
Assignments



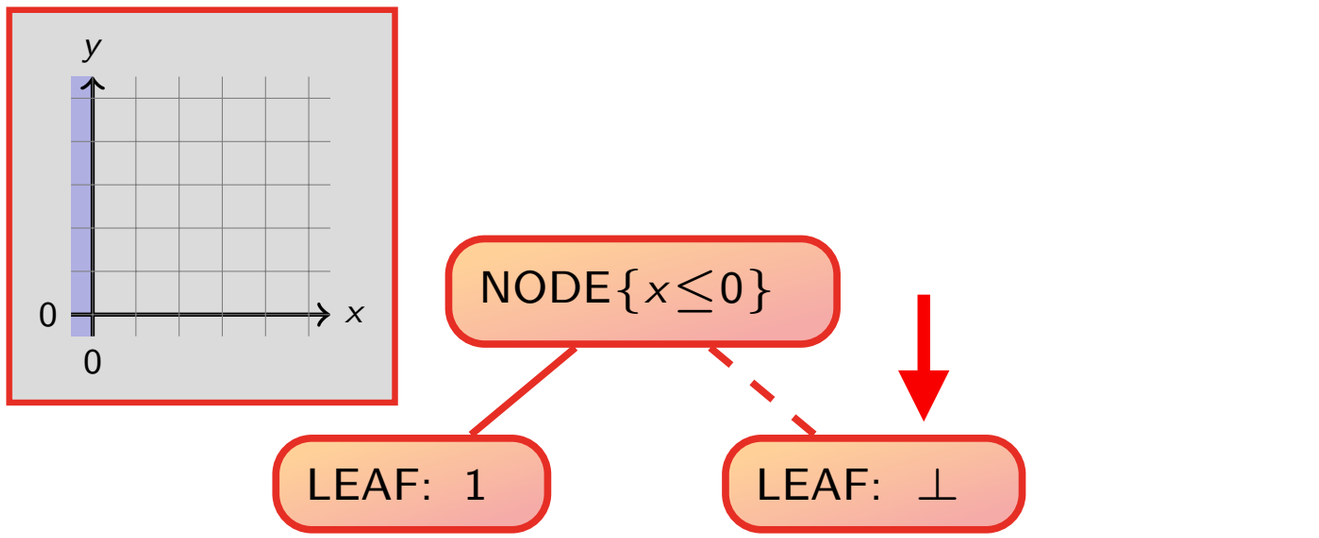


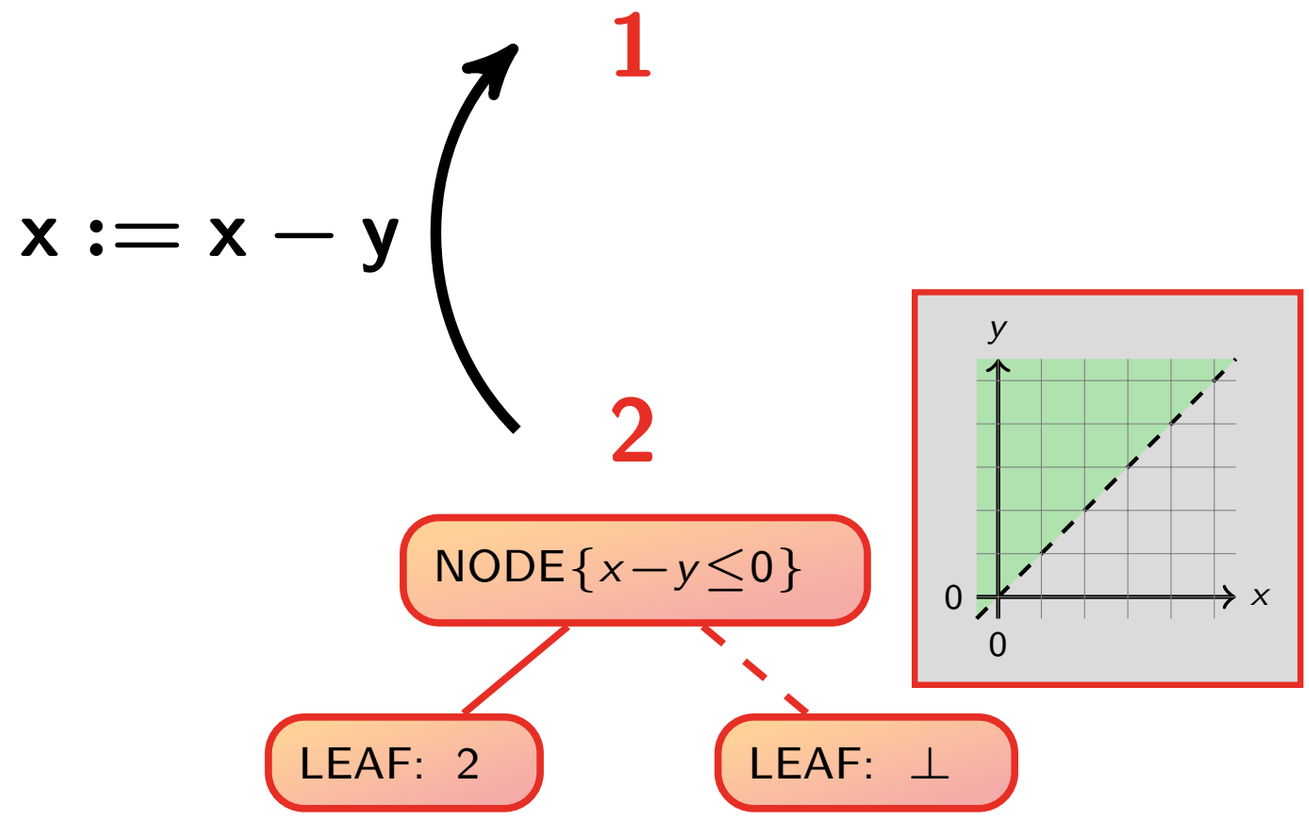
Assignments



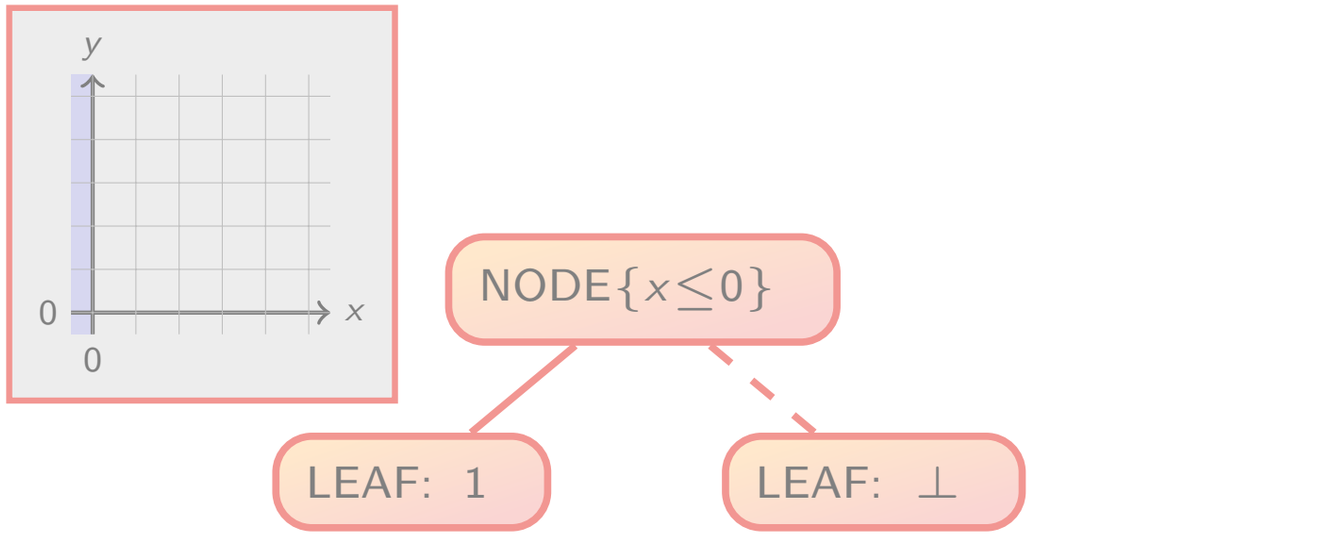


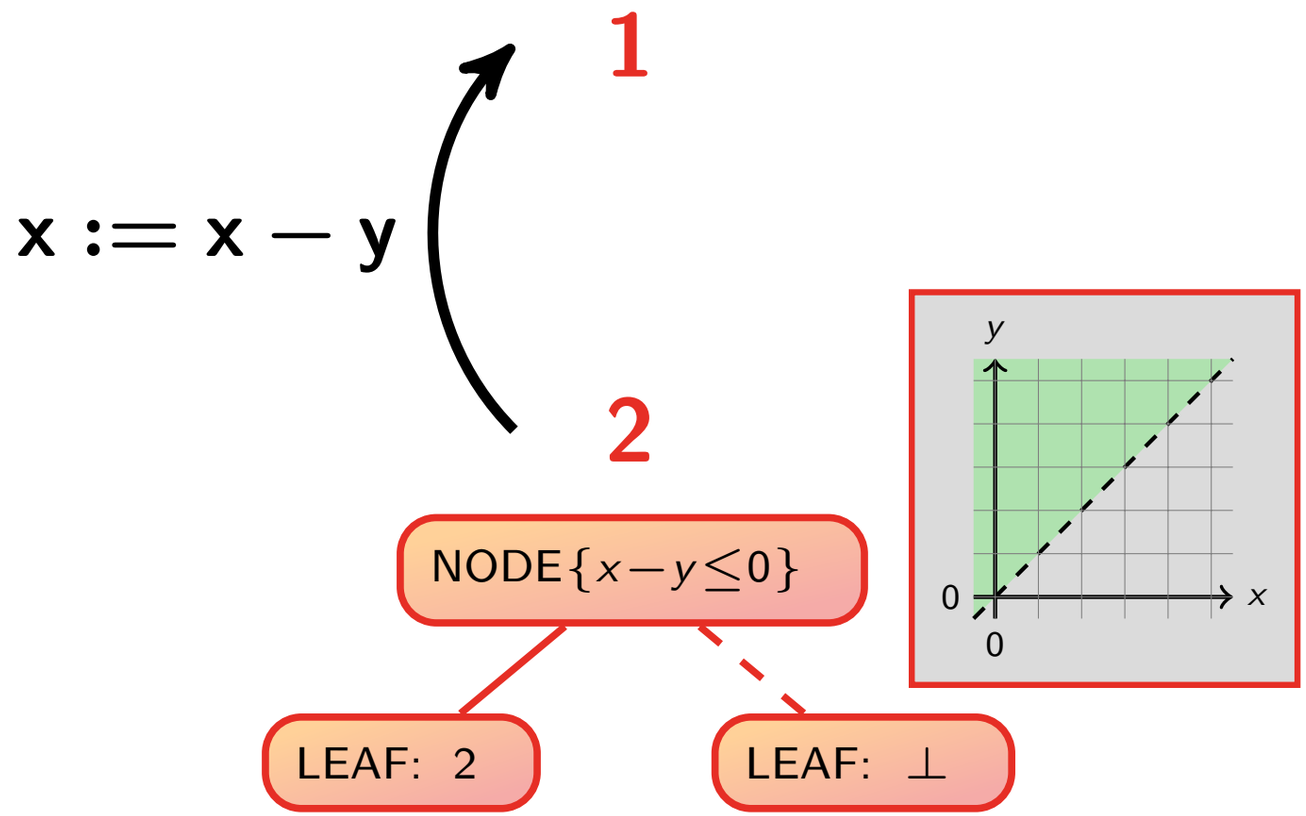
Assignments





Assignments





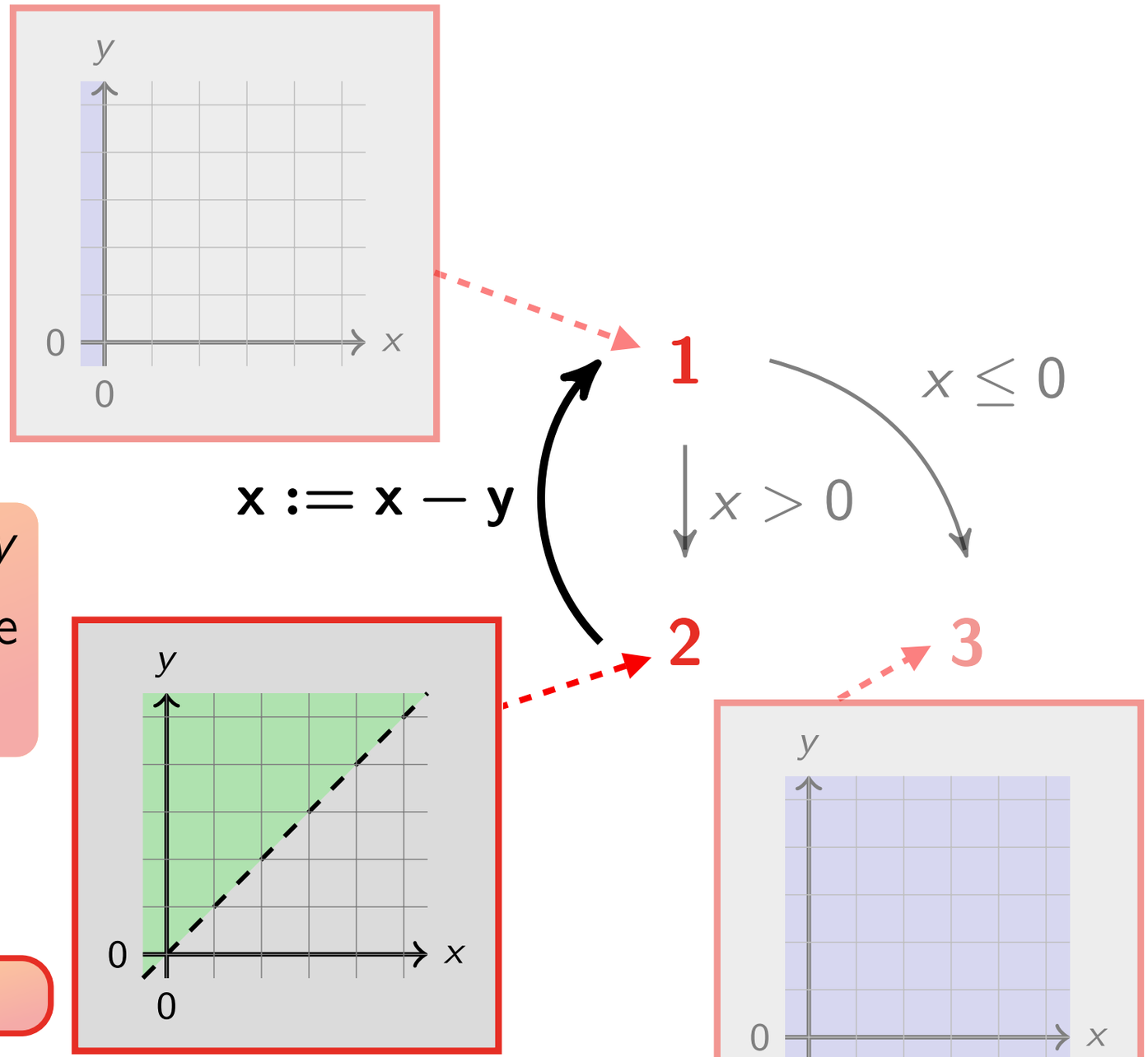
Example

int : $x,\ y$

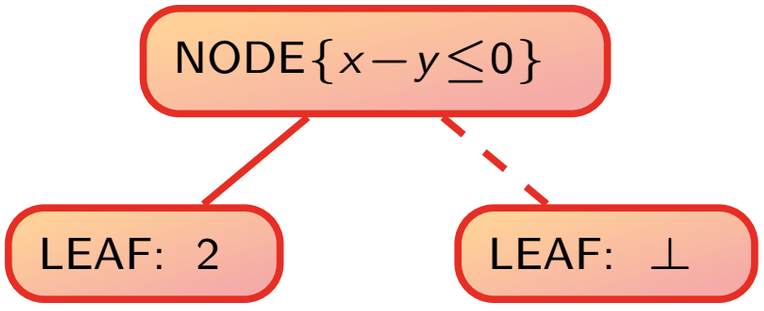
while $^{1}(x > 0)$ do

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

od^{3}



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



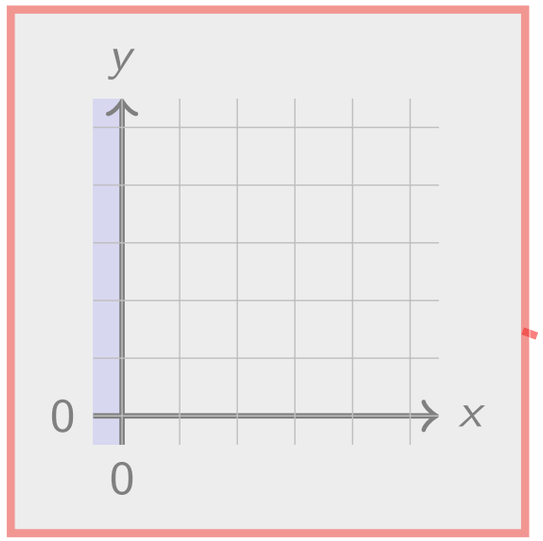
Example

int : $x,\ y$

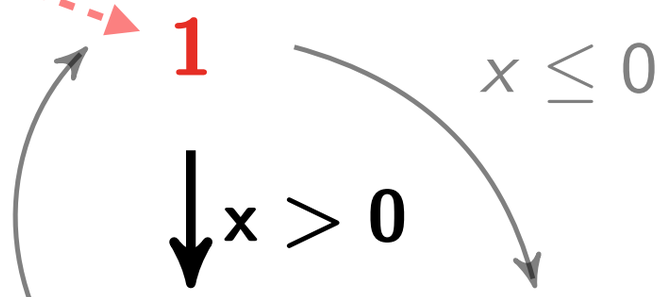
while $^{1}(x > 0)$ do

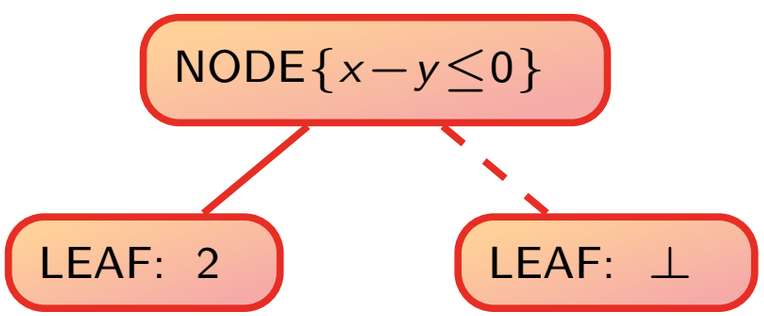
$\quad {}^{2}x := x - y$

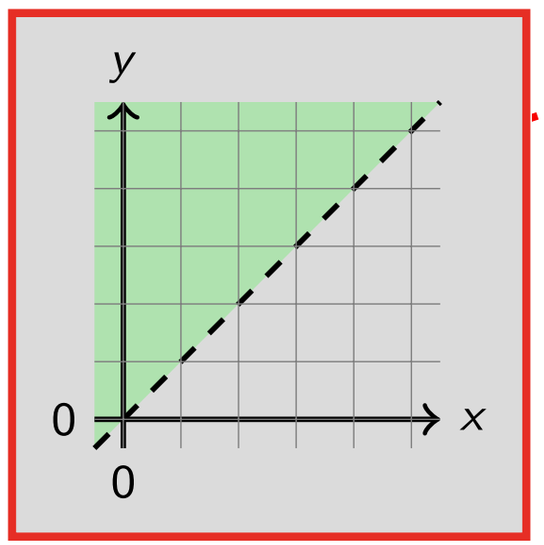
od^{3}

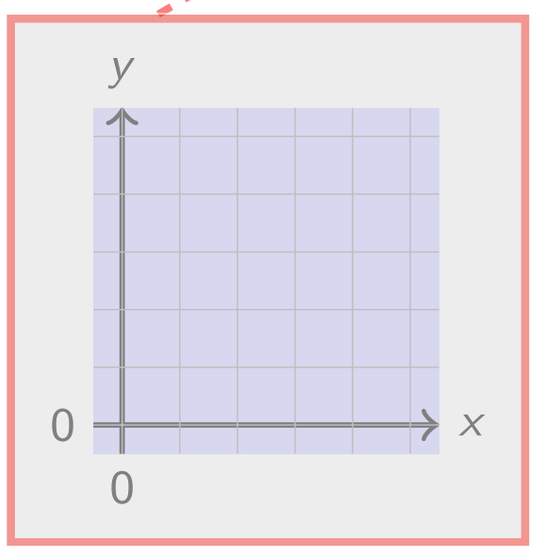


$x := x - y$

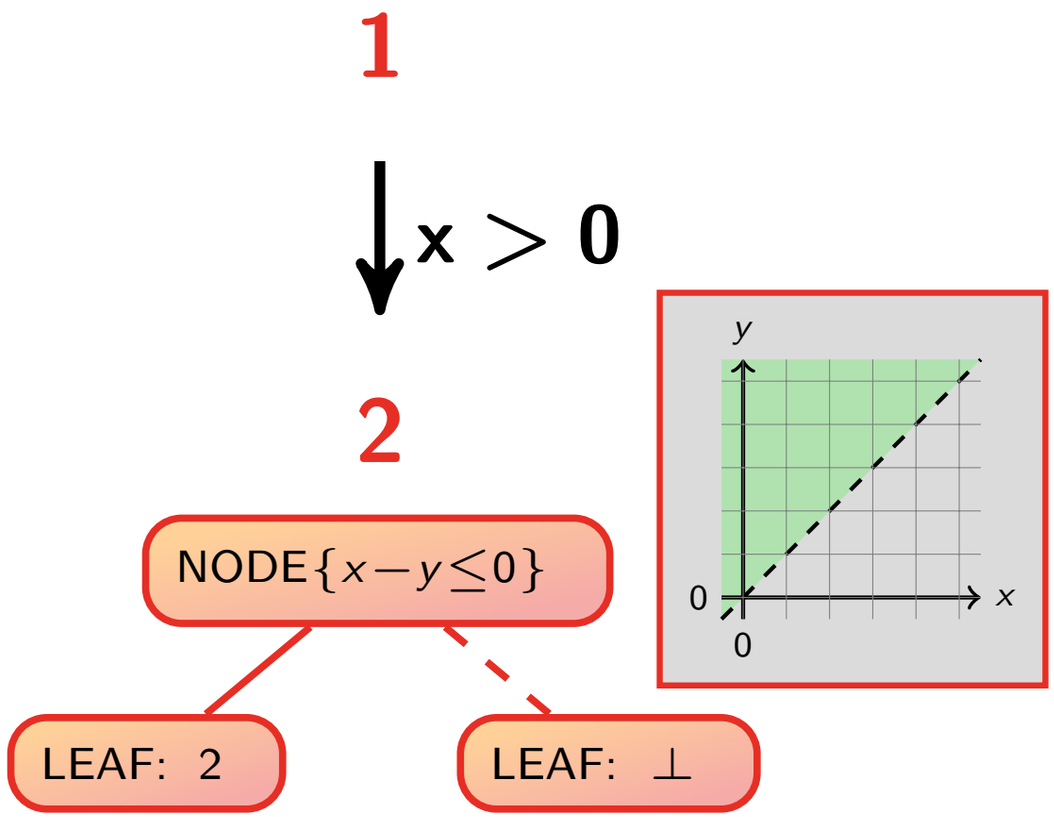




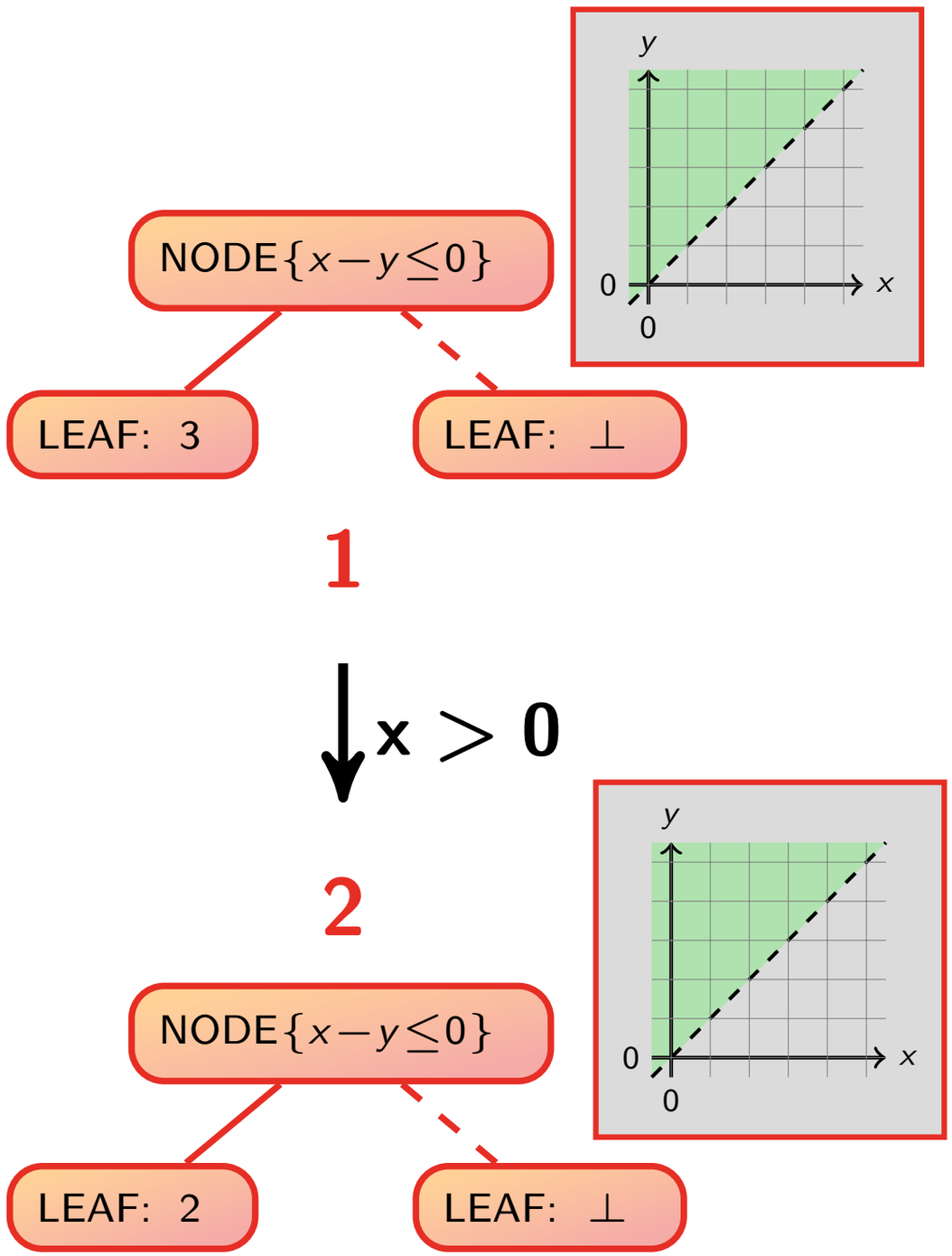




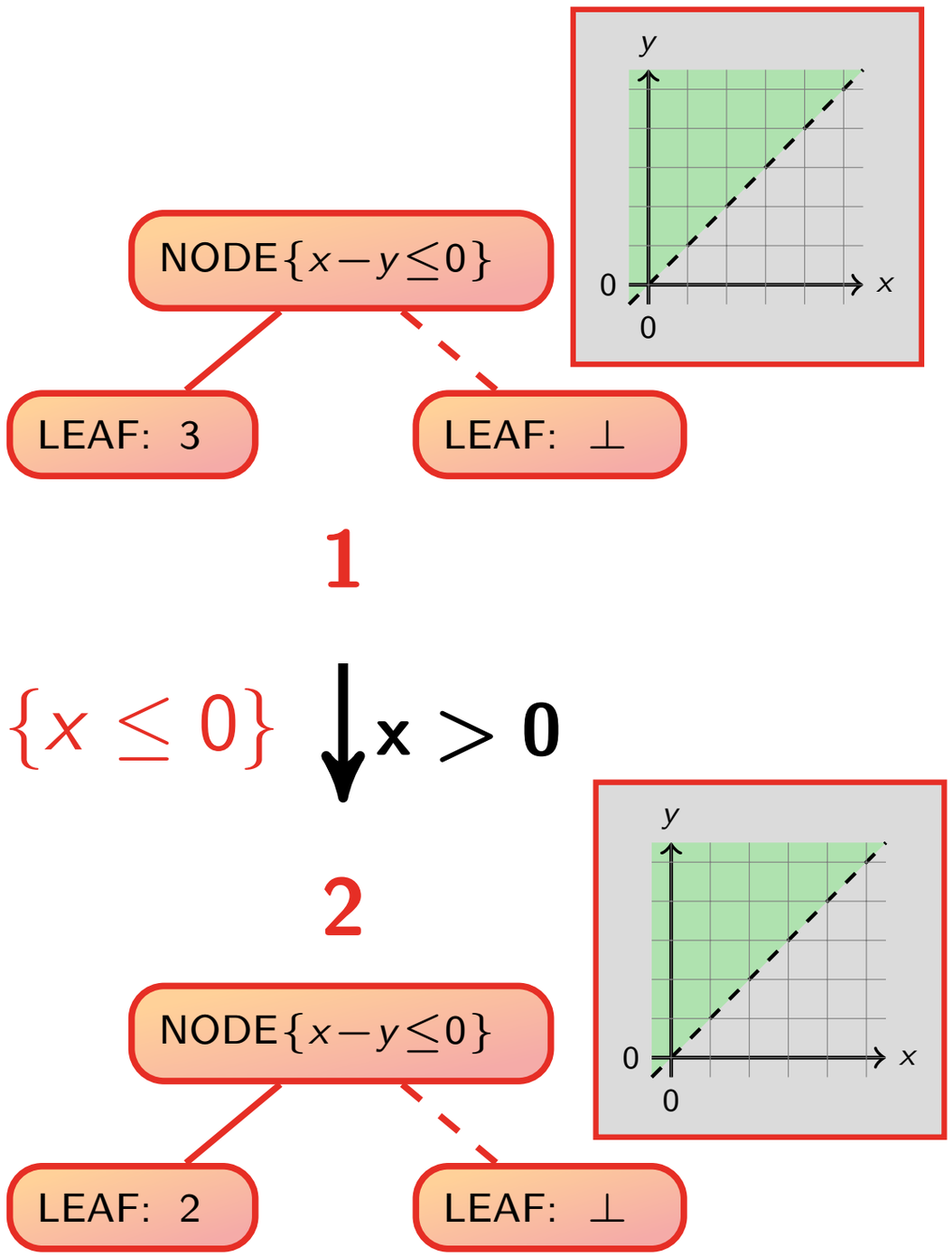
Tests



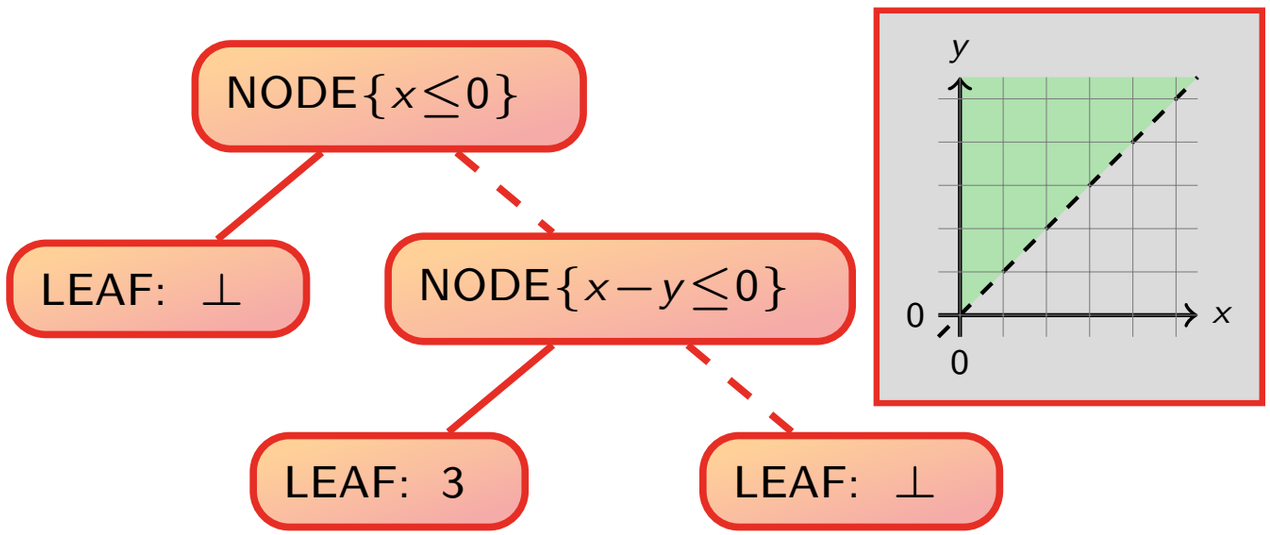
Tests



Tests



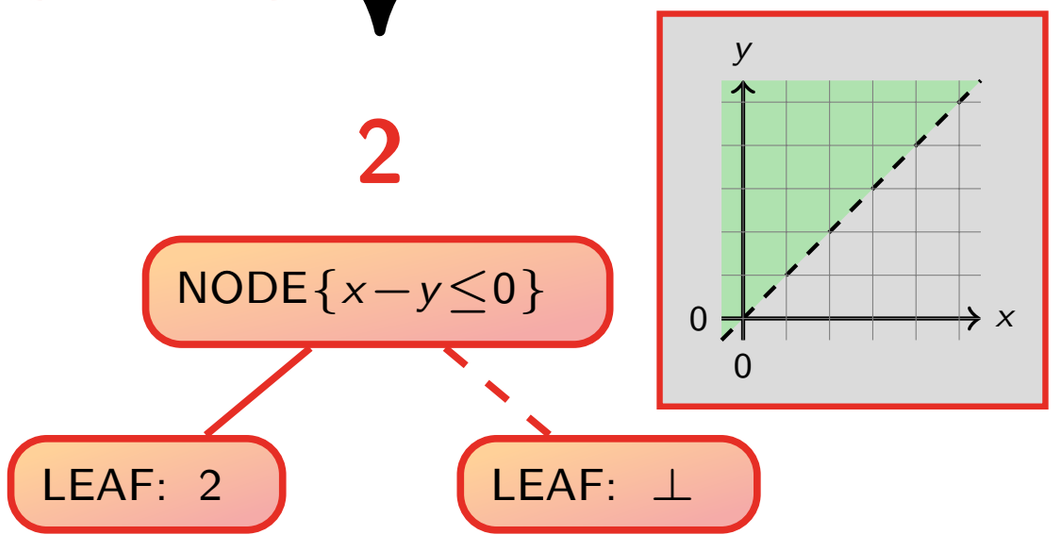
Tests



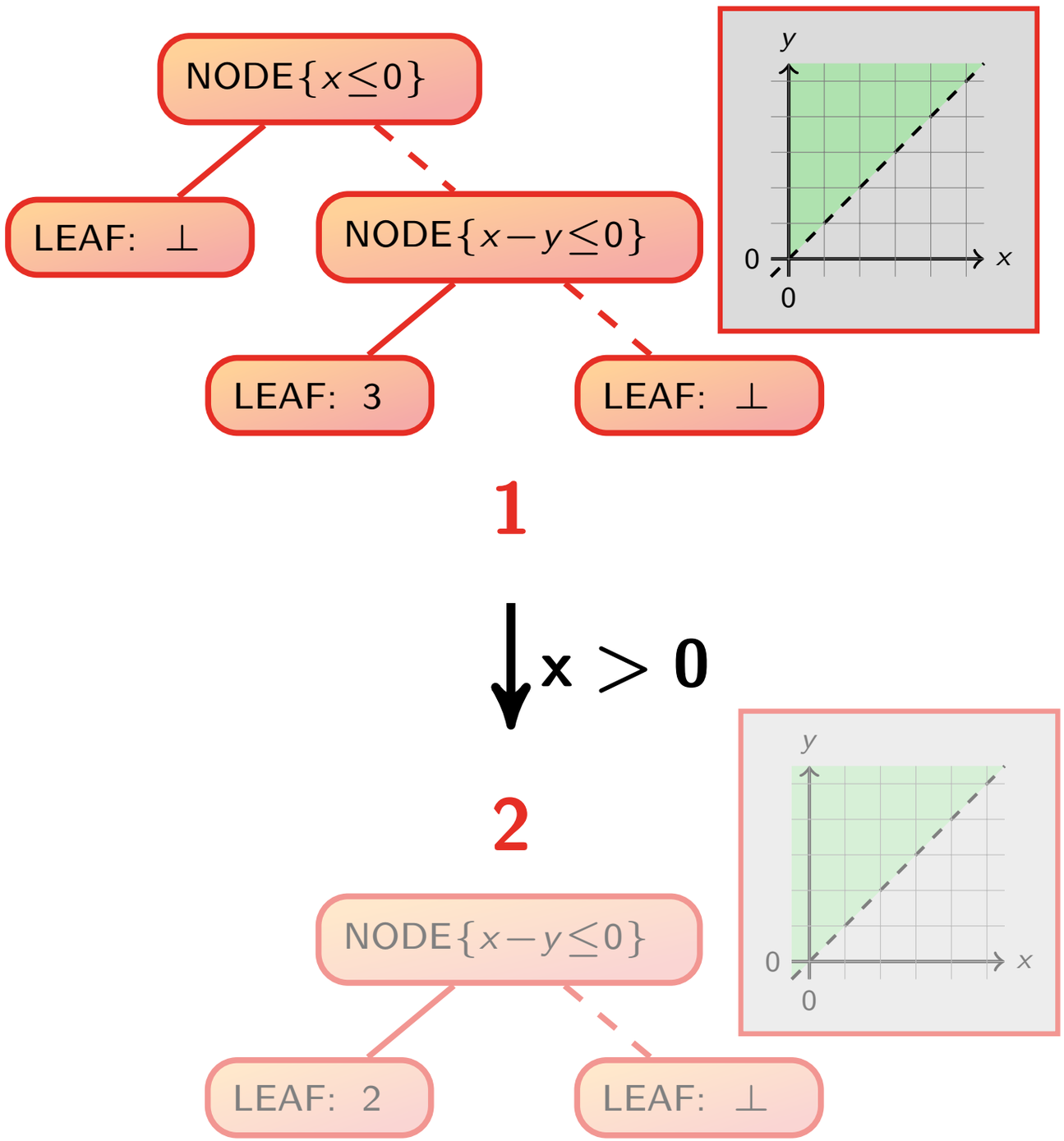
1

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

2



Tests


NODE{x≤0}
LEAF: ⊥
NODE{x−y≤0}
LEAF: 3
LEAF: ⊥
y
0
x
0
1
x > 0
2
NODE{x−y≤0}
LEAF: 2
LEAF: ⊥
y
0
x
0

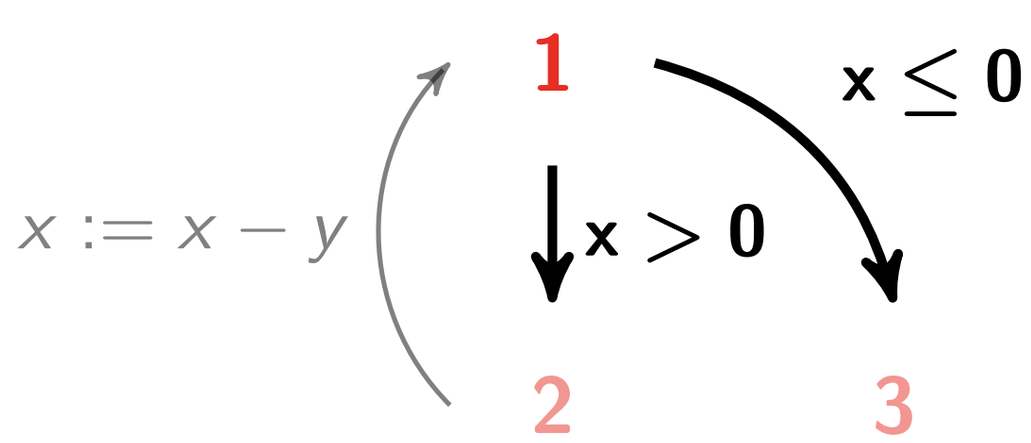
Example

int : $x,\ y$

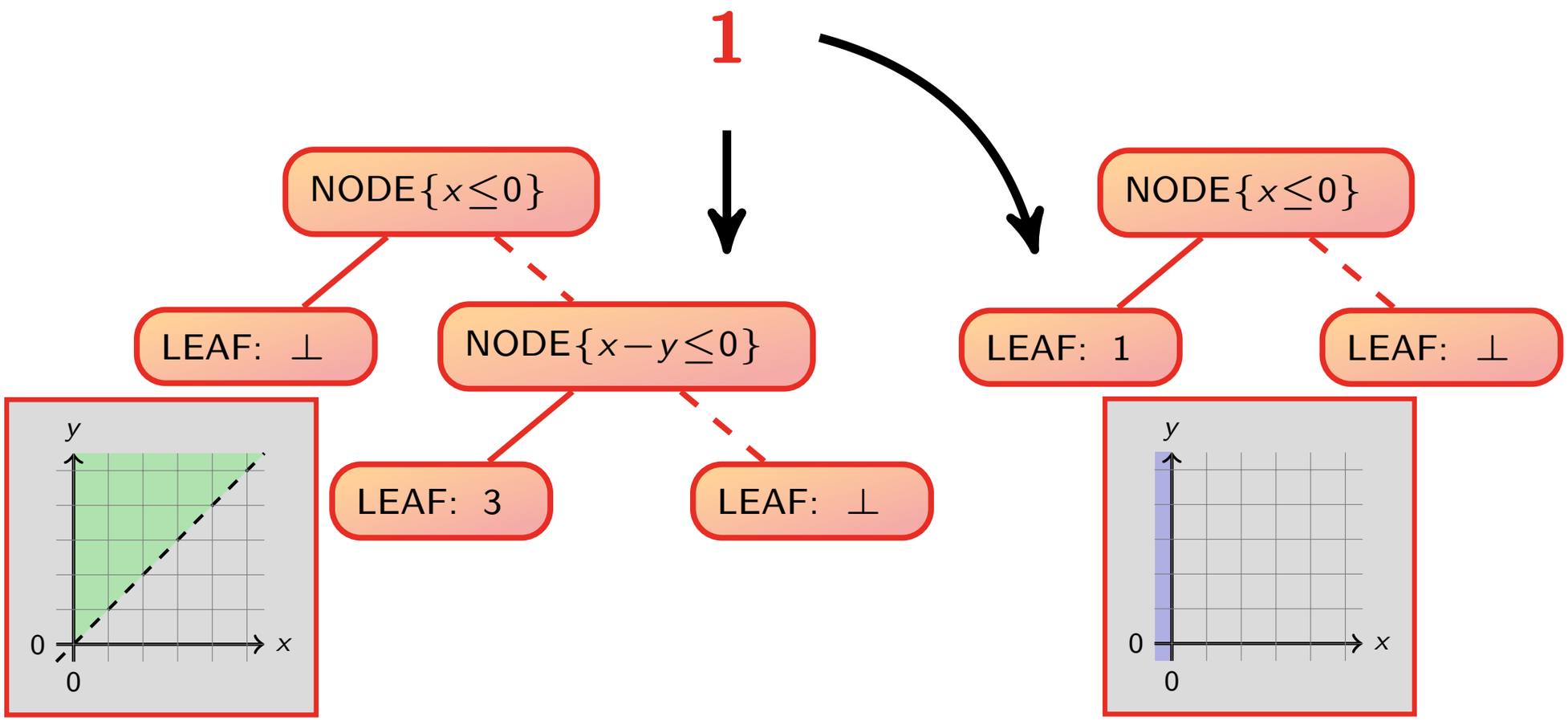
while $^{1}(x > 0)$ do

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

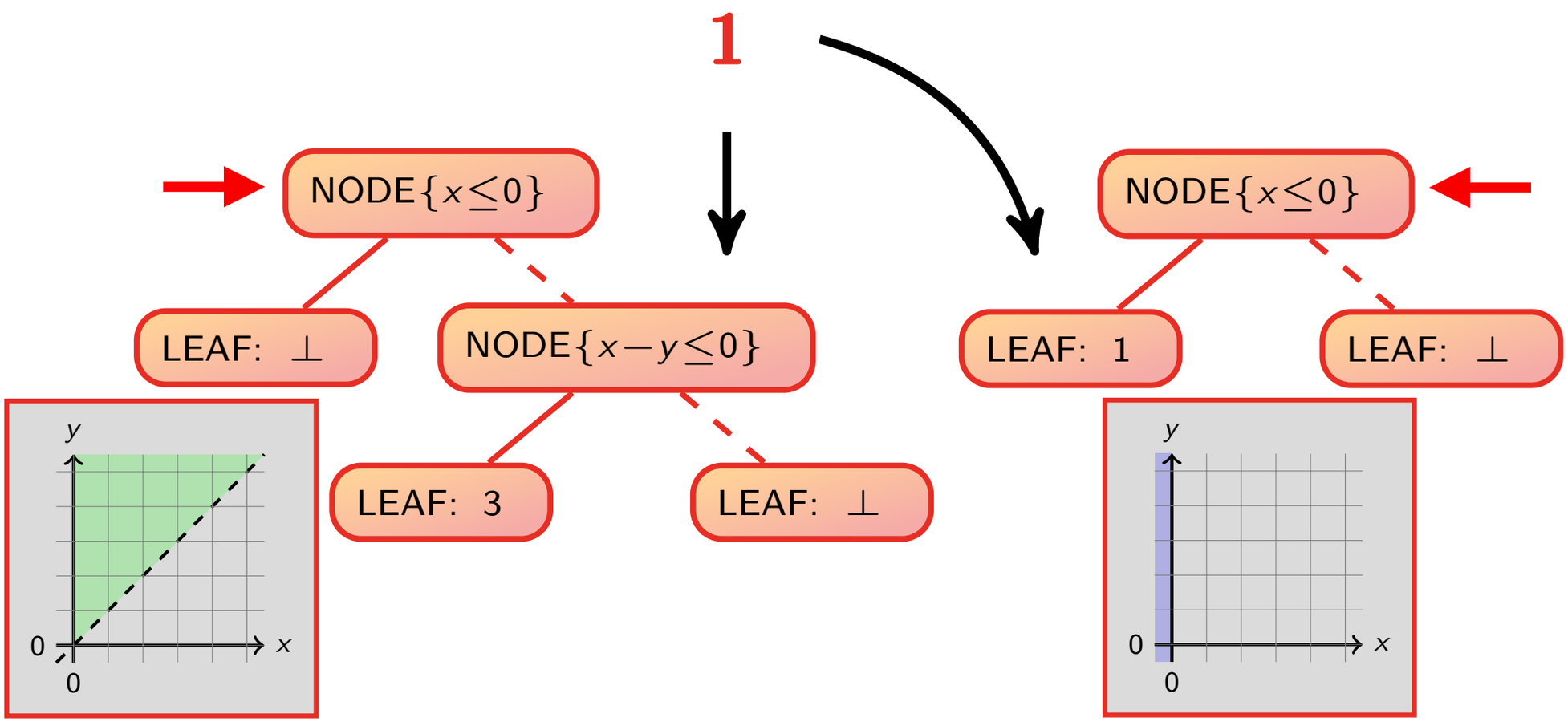
od^{3}



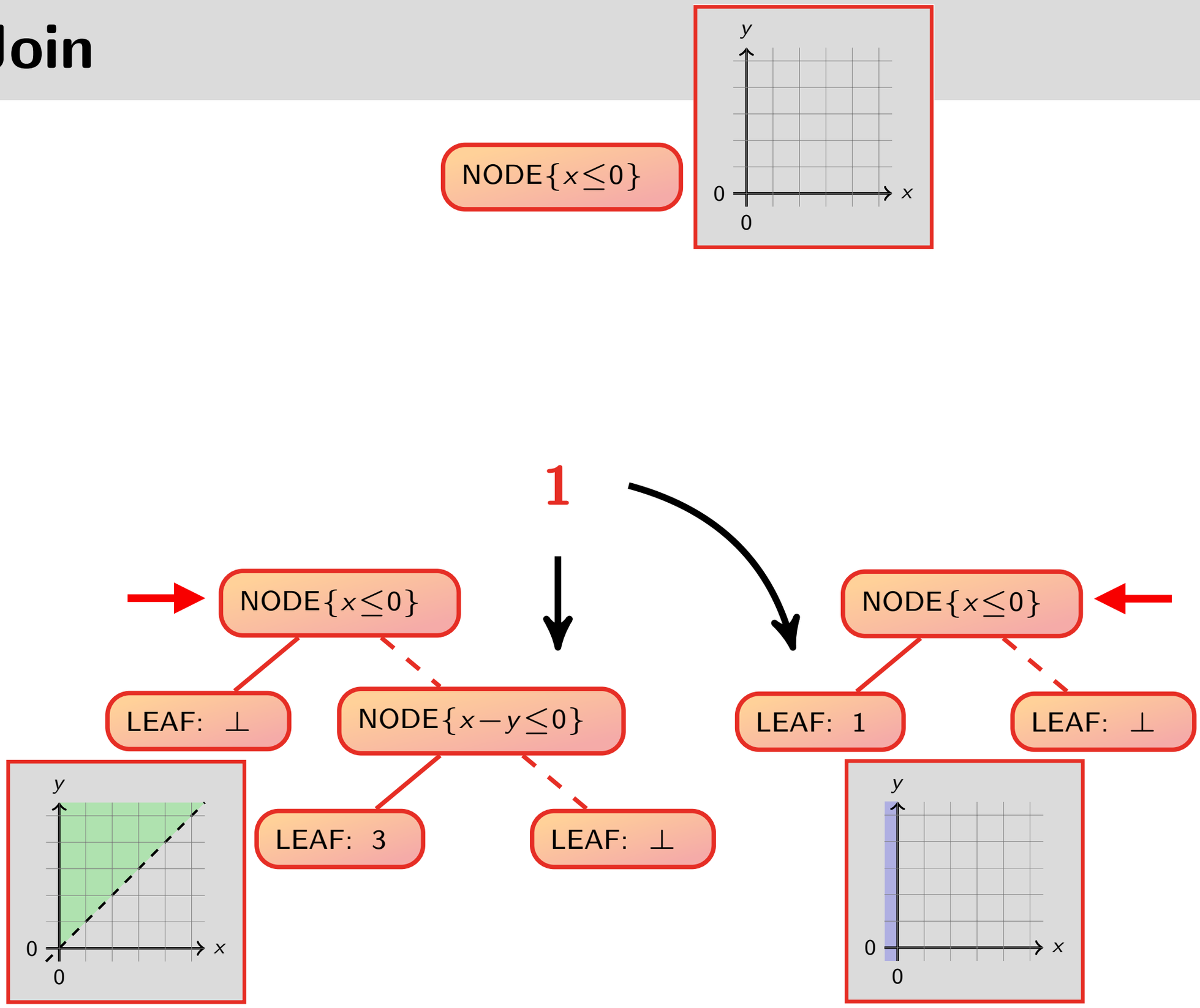
Join


1
NODE{x≤0}
LEAF: ⊥
NODE{x−y≤0}
LEAF: 3
LEAF: ⊥
y
0
x
0
NODE{x≤0}
LEAF: 1
LEAF: ⊥
y
0
x
0

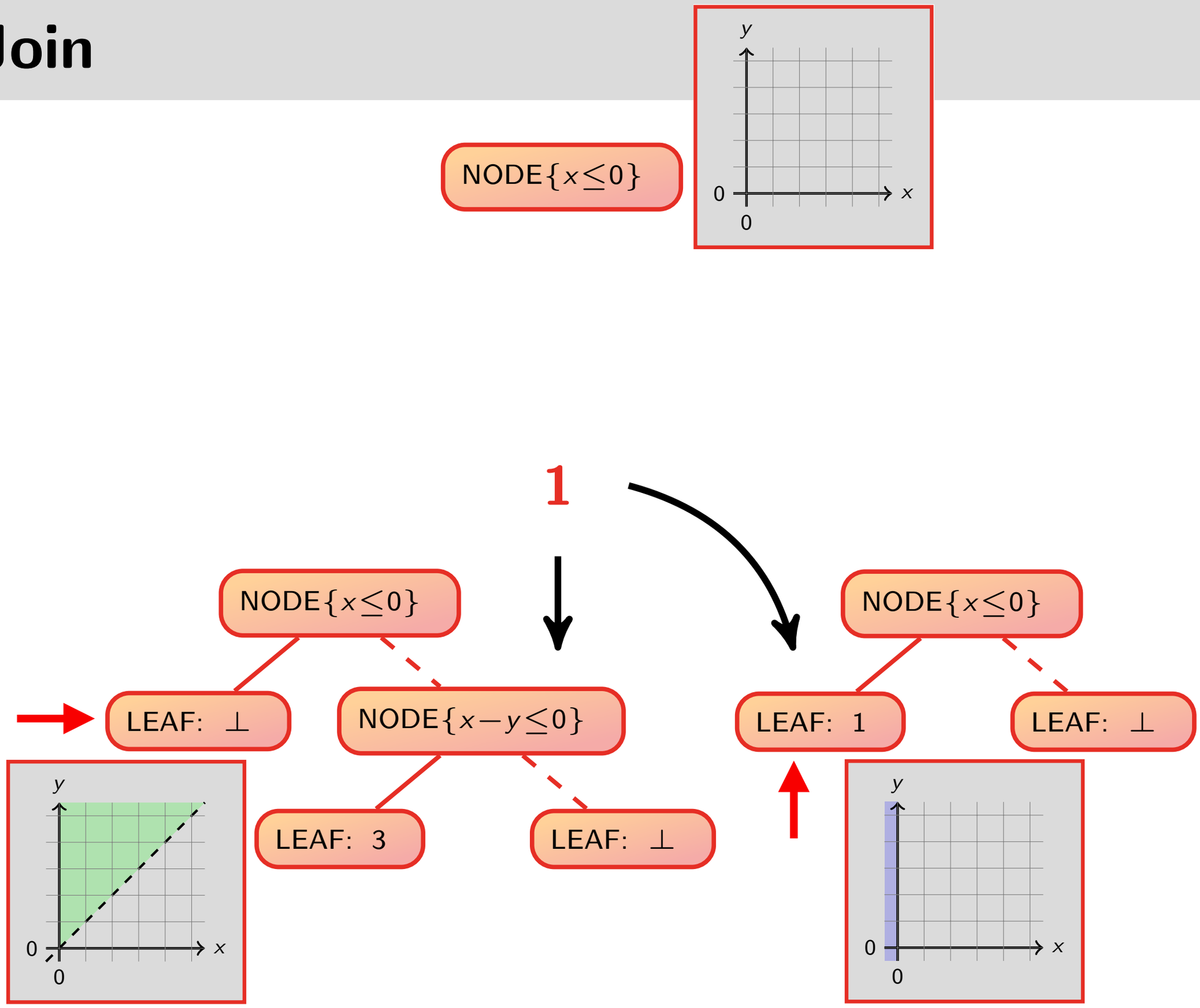
Join


1
NODE{x≤0}
LEAF: ⊥
NODE{x−y≤0}
LEAF: 3
LEAF: ⊥
y
0
x
0
NODE{x≤0}
LEAF: 1
LEAF: ⊥
y
0
x
0

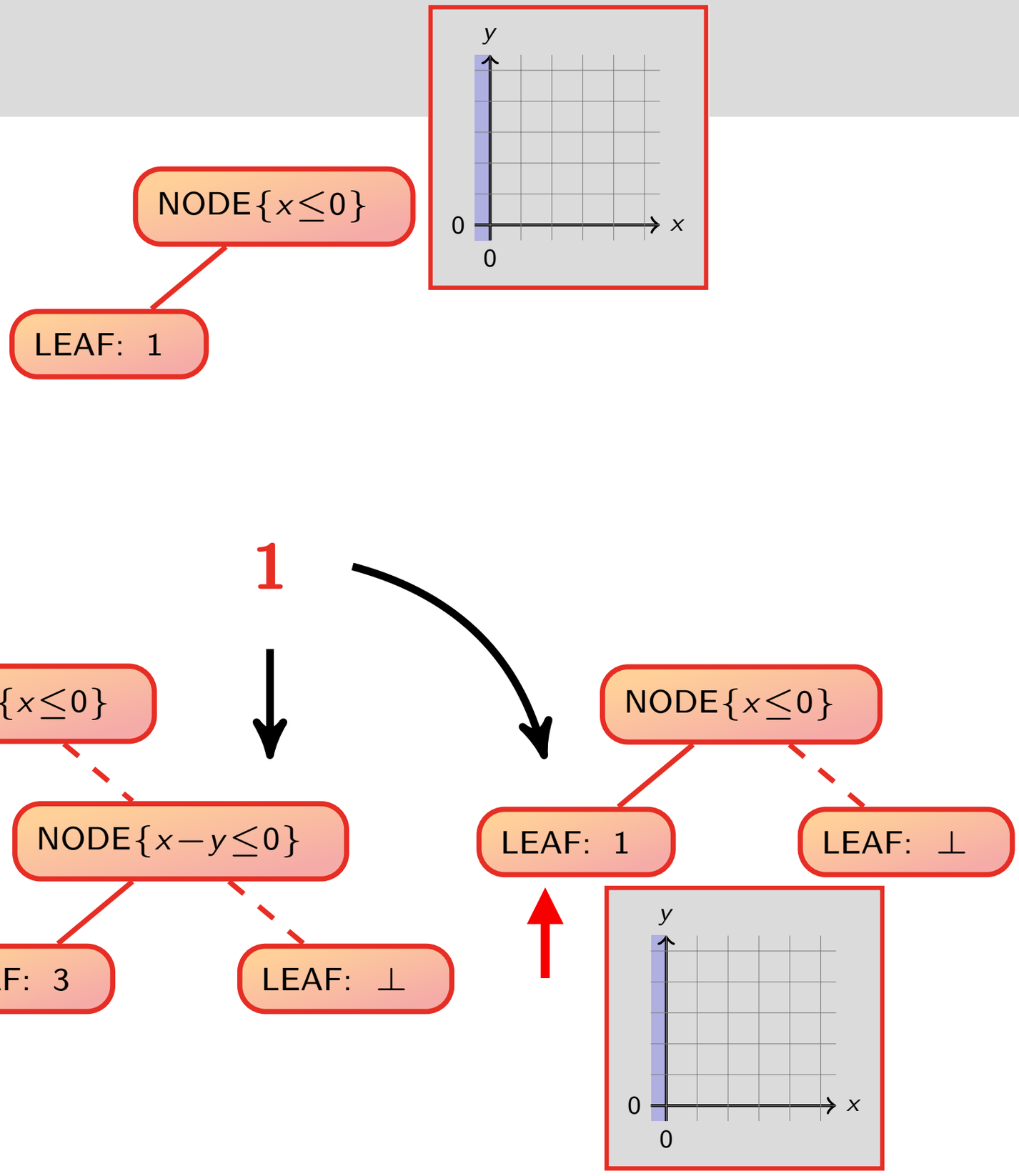
Join


NODE{x≤0}
y
0
x
0
1
NODE{x≤0}
LEAF: ⊥
NODE{x−y≤0}
LEAF: 3
LEAF: ⊥
y
0
x
0
NODE{x≤0}
LEAF: 1
LEAF: ⊥
y
0
x
0

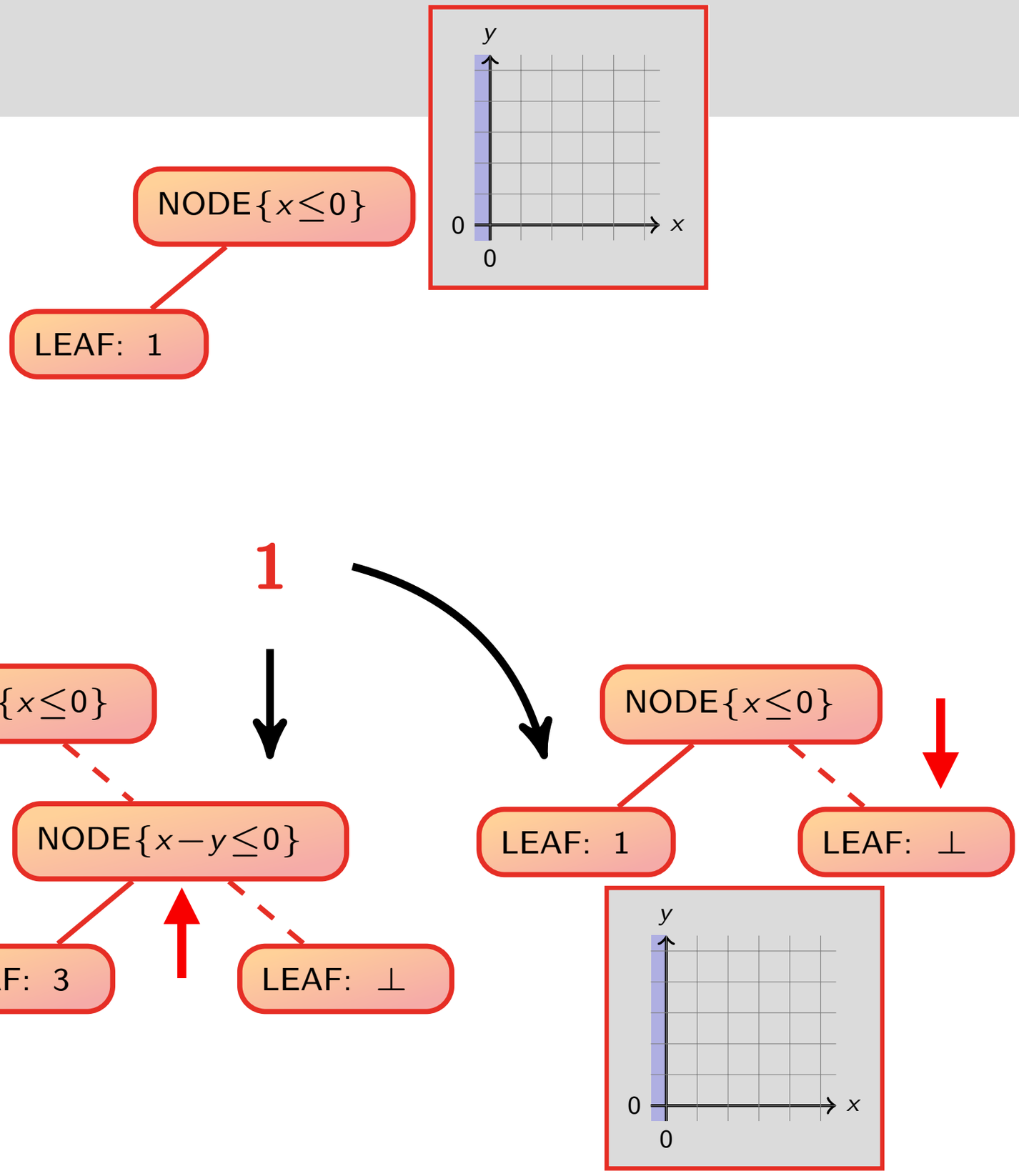
Join


NODE{x≤0}
y
0
x
0
1
NODE{x≤0}
LEAF: ⊥
NODE{x−y≤0}
LEAF: 3
LEAF: ⊥
y
0
x
0
NODE{x≤0}
LEAF: 1
LEAF: ⊥
y
0
x
0

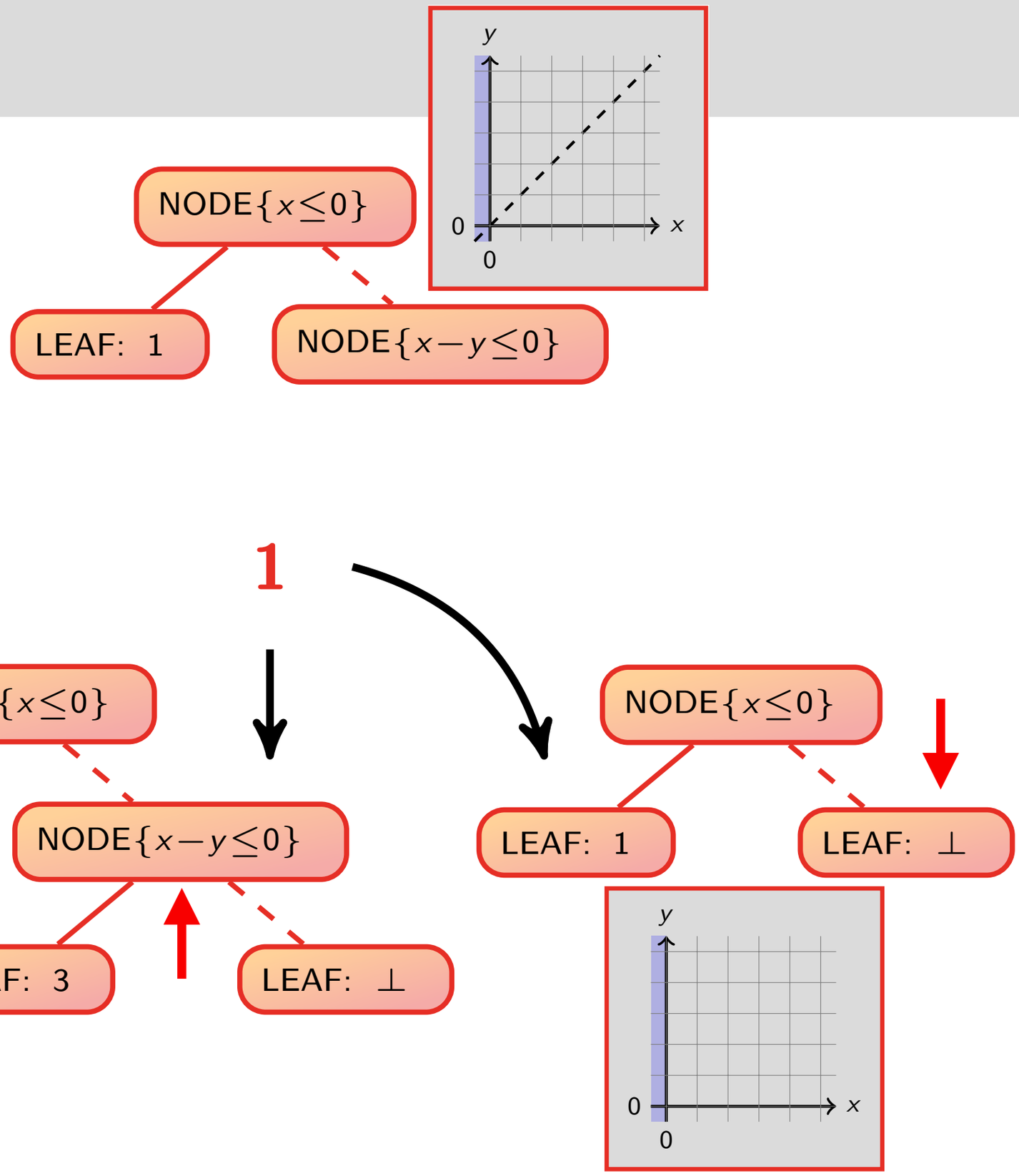
Join


NODE{x≤0}
y
0
x
0
LEAF: 1
1
NODE{x≤0}
LEAF: ⊥
NODE{x−y≤0}
LEAF: 3
LEAF: ⊥
y
0
x
0
NODE{x≤0}
LEAF: 1
LEAF: ⊥
y
0
x
0

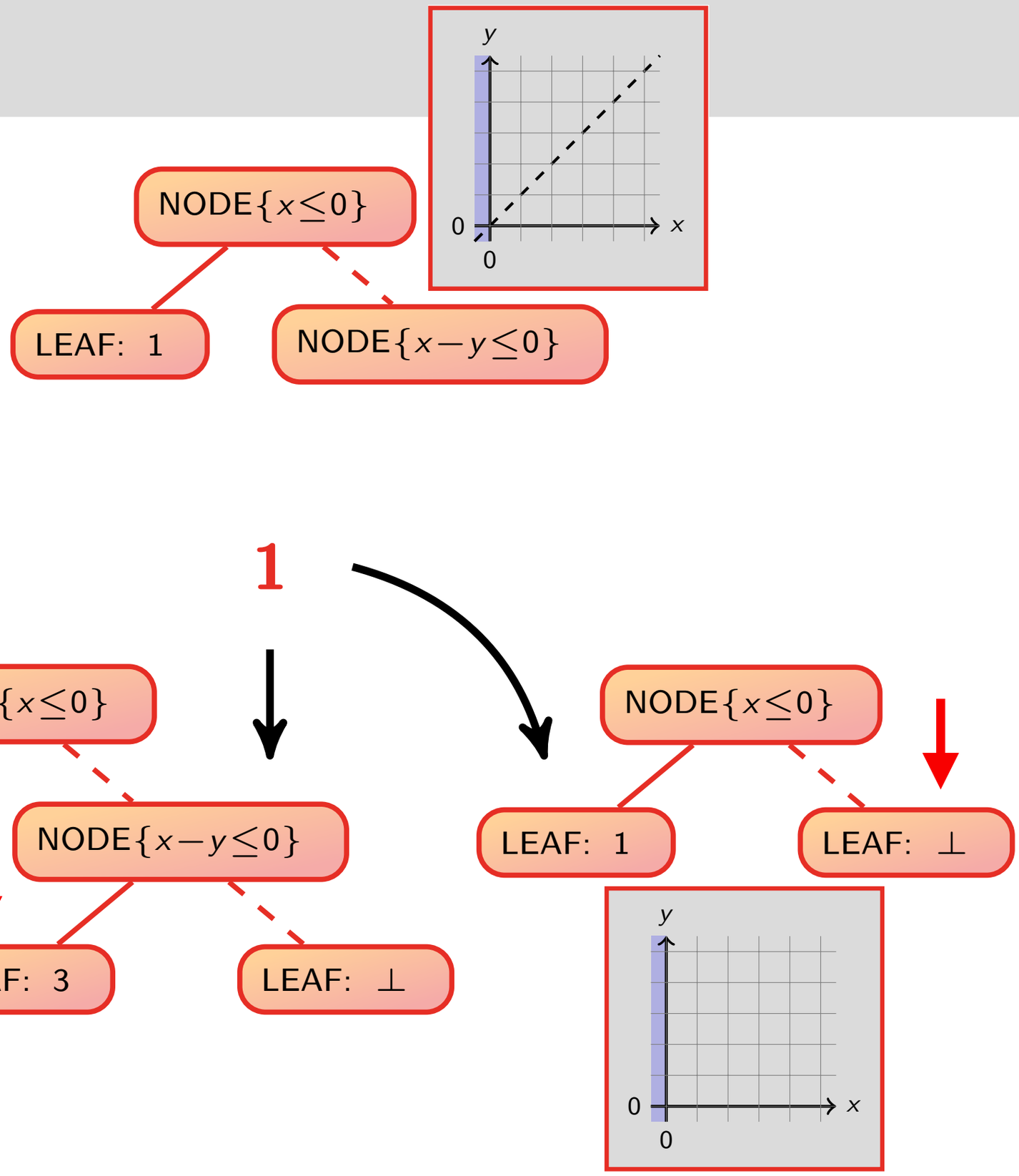
Join


NODE{x≤0}
LEAF: 1
y
0
x
0
1
NODE{x≤0}
LEAF: ⊥
NODE{x−y≤0}
LEAF: 3
LEAF: ⊥
y
0
x
0
NODE{x≤0}
LEAF: 1
LEAF: ⊥
y
0
x
0

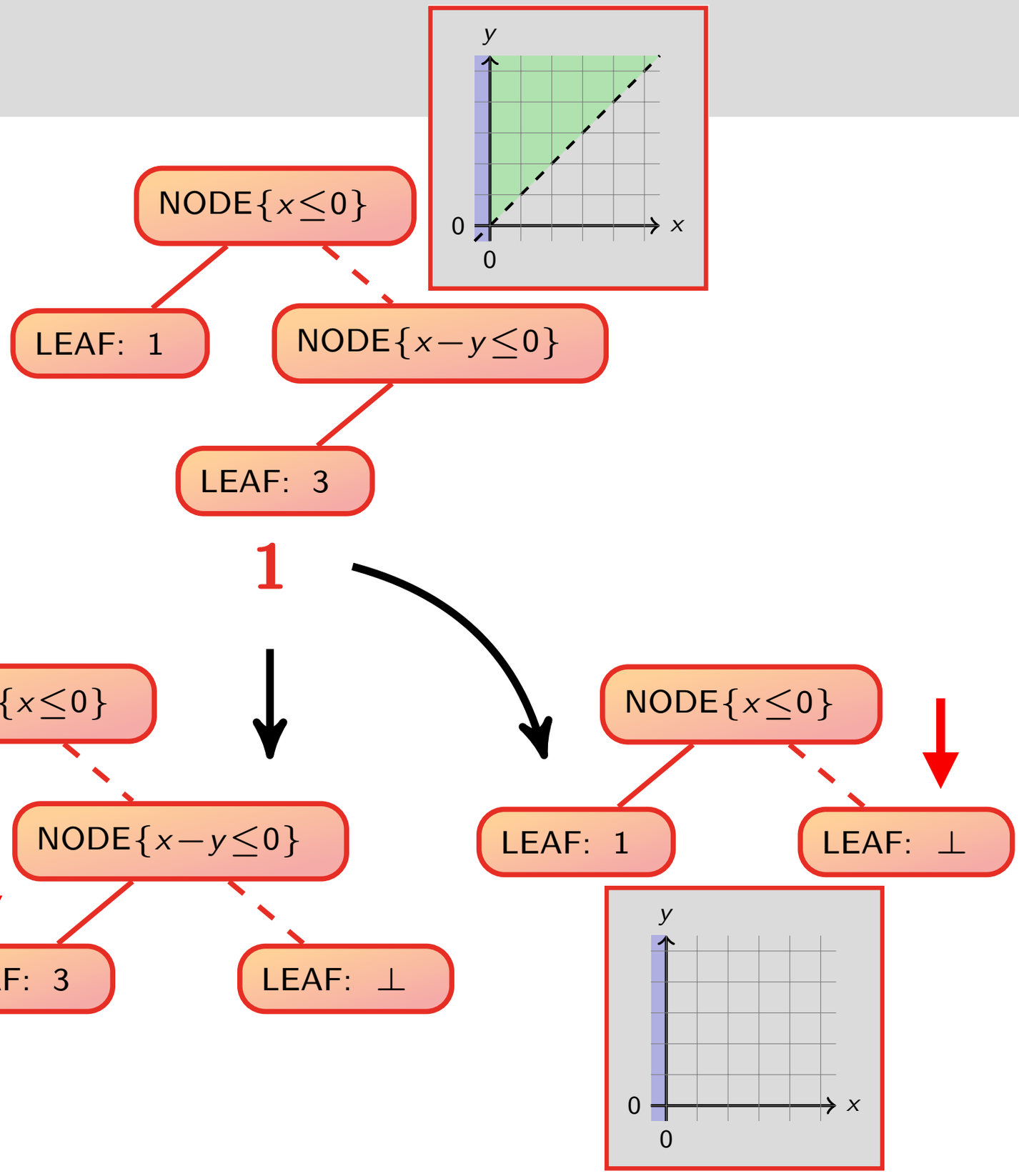
Join


NODE{x≤0}
y
0
x
0
LEAF: 1
NODE{x−y≤0}
1
NODE{x≤0}
LEAF: ⊥
NODE{x−y≤0}
LEAF: 3
LEAF: ⊥
NODE{x≤0}
LEAF: 1
LEAF: ⊥
y
0
x
0
y
0
x
0

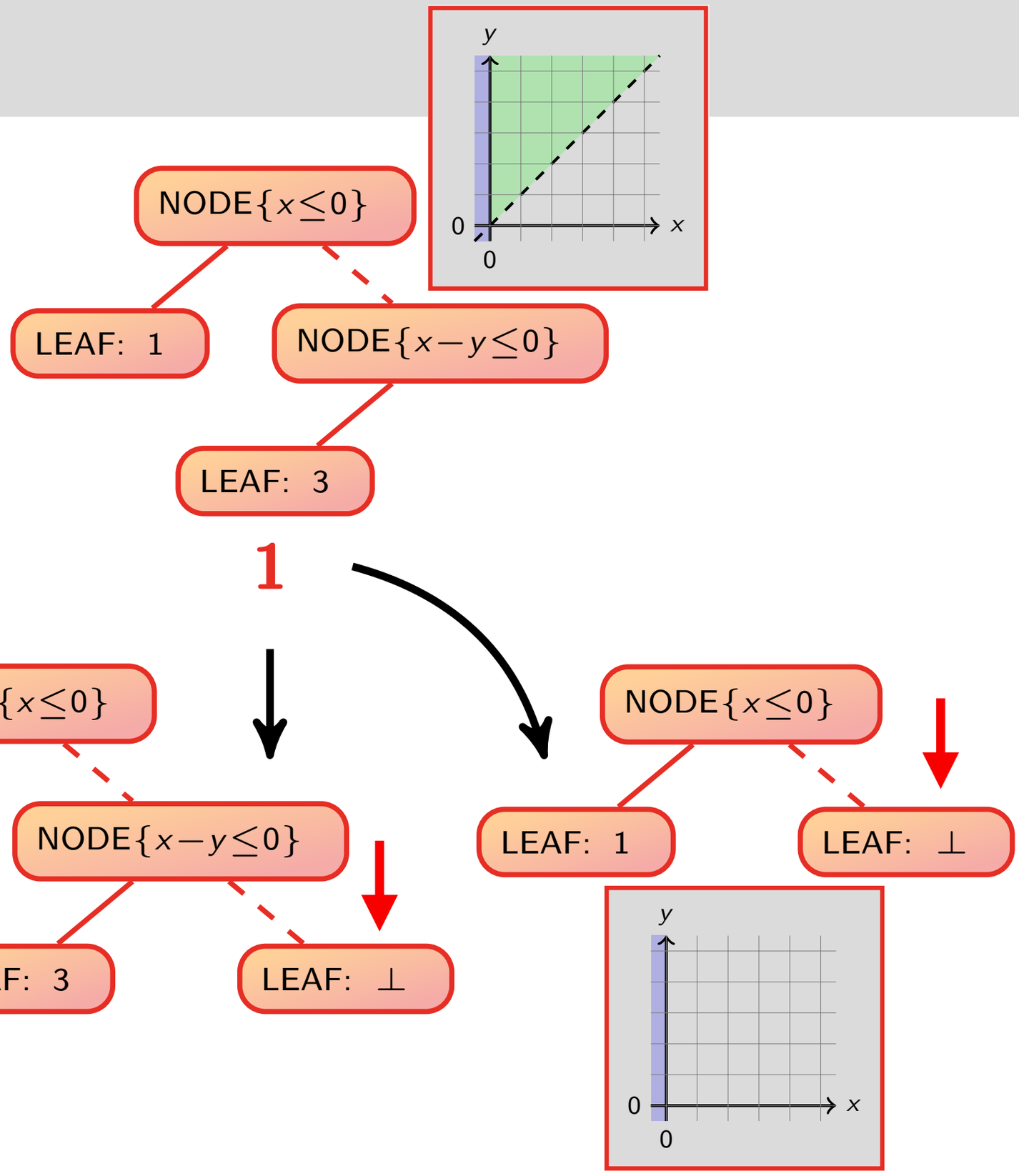
Join


NODE{x≤0}
y
0
x
0
LEAF: 1
NODE{x−y≤0}
1
NODE{x≤0}
LEAF: ⊥
NODE{x−y≤0}
LEAF: 3
LEAF: ⊥
NODE{x≤0}
LEAF: 1
LEAF: ⊥
y
0
x
0
y
0
x
0

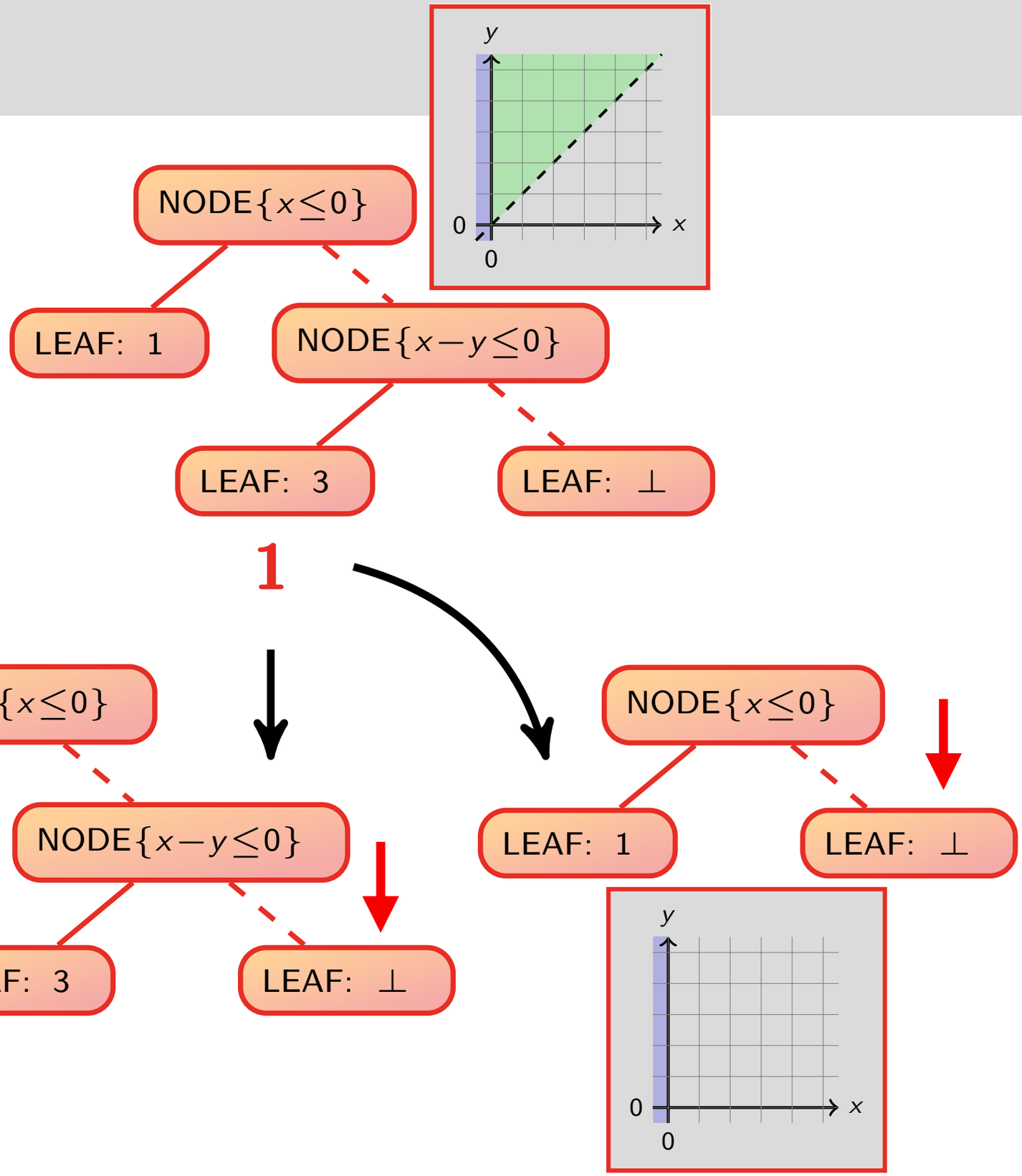
Join


NODE{x≤0}
LEAF: 1
NODE{x−y≤0}
LEAF: 3
LEAF: ⊥
y
0
x
0
1
NODE{x≤0}
LEAF: ⊥
NODE{x−y≤0}
LEAF: 3
LEAF: ⊥
y
0
x
0
NODE{x≤0}
LEAF: 1
LEAF: ⊥
y
0
x
0

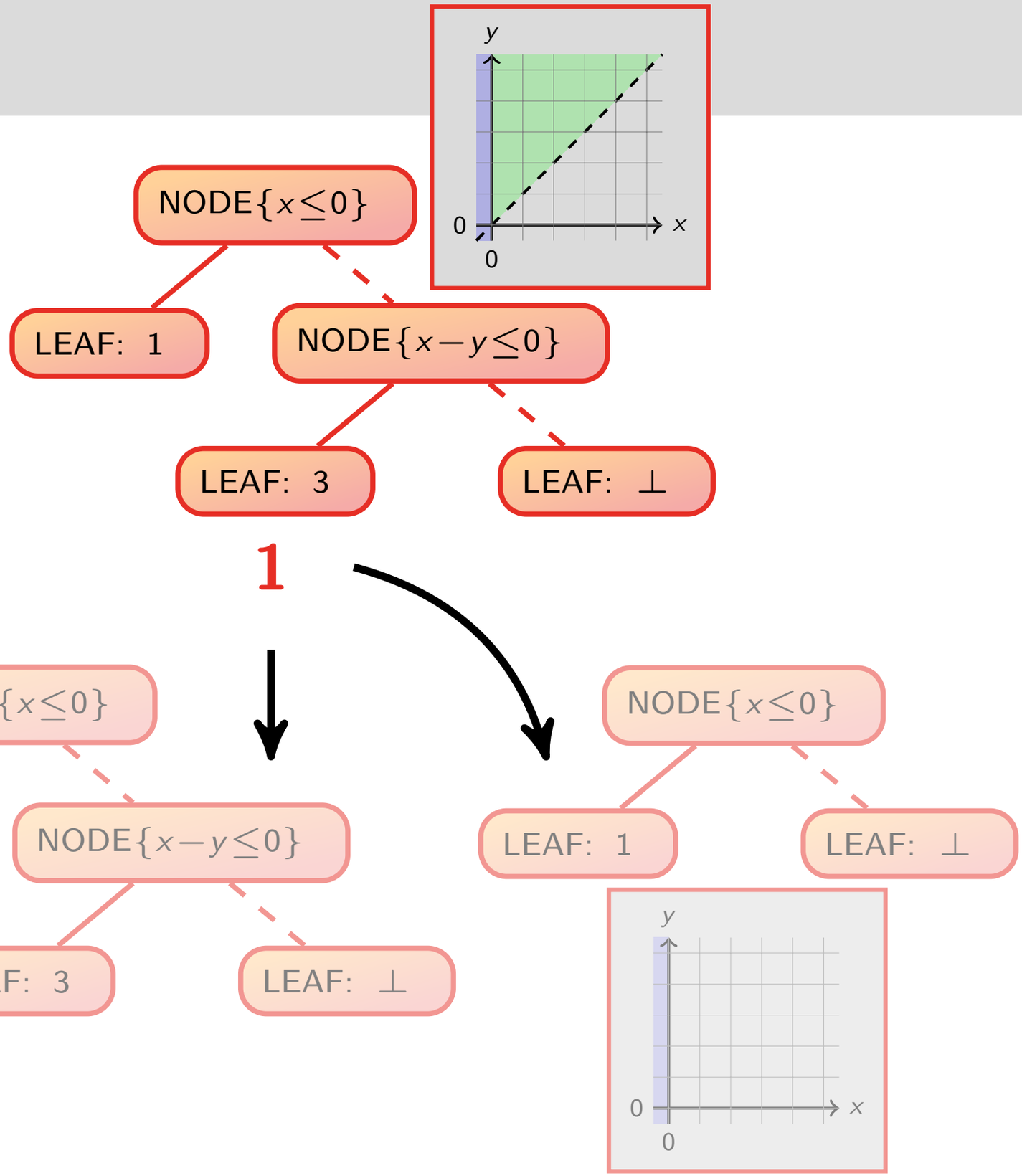
Join


NODE{x≤0}
LEAF: 1
NODE{x−y≤0}
LEAF: 3
LEAF: ⊥
y
0
x
0
1
NODE{x≤0}
LEAF: ⊥
NODE{x−y≤0}
LEAF: 3
LEAF: ⊥
y
0
x
0
NODE{x≤0}
LEAF: 1
LEAF: ⊥
y
0
x
0

Join


NODE{x≤0}
LEAF: 1
NODE{x−y≤0}
LEAF: 3
LEAF: ⊥
y
0
x
0
1
NODE{x≤0}
LEAF: ⊥
NODE{x−y≤0}
LEAF: 3
LEAF: ⊥
y
0
x
0
NODE{x≤0}
LEAF: 1
LEAF: ⊥
y
0
x
0

Join


NODE{x≤0}
y
0
x
0
LEAF: 1
NODE{x−y≤0}
LEAF: 3
LEAF: ⊥
1
NODE{x≤0}
LEAF: ⊥
NODE{x−y≤0}
LEAF: 3
LEAF: ⊥
y
0
x
0
NODE{x≤0}
LEAF: 1
LEAF: ⊥
y
0
x
0

Example

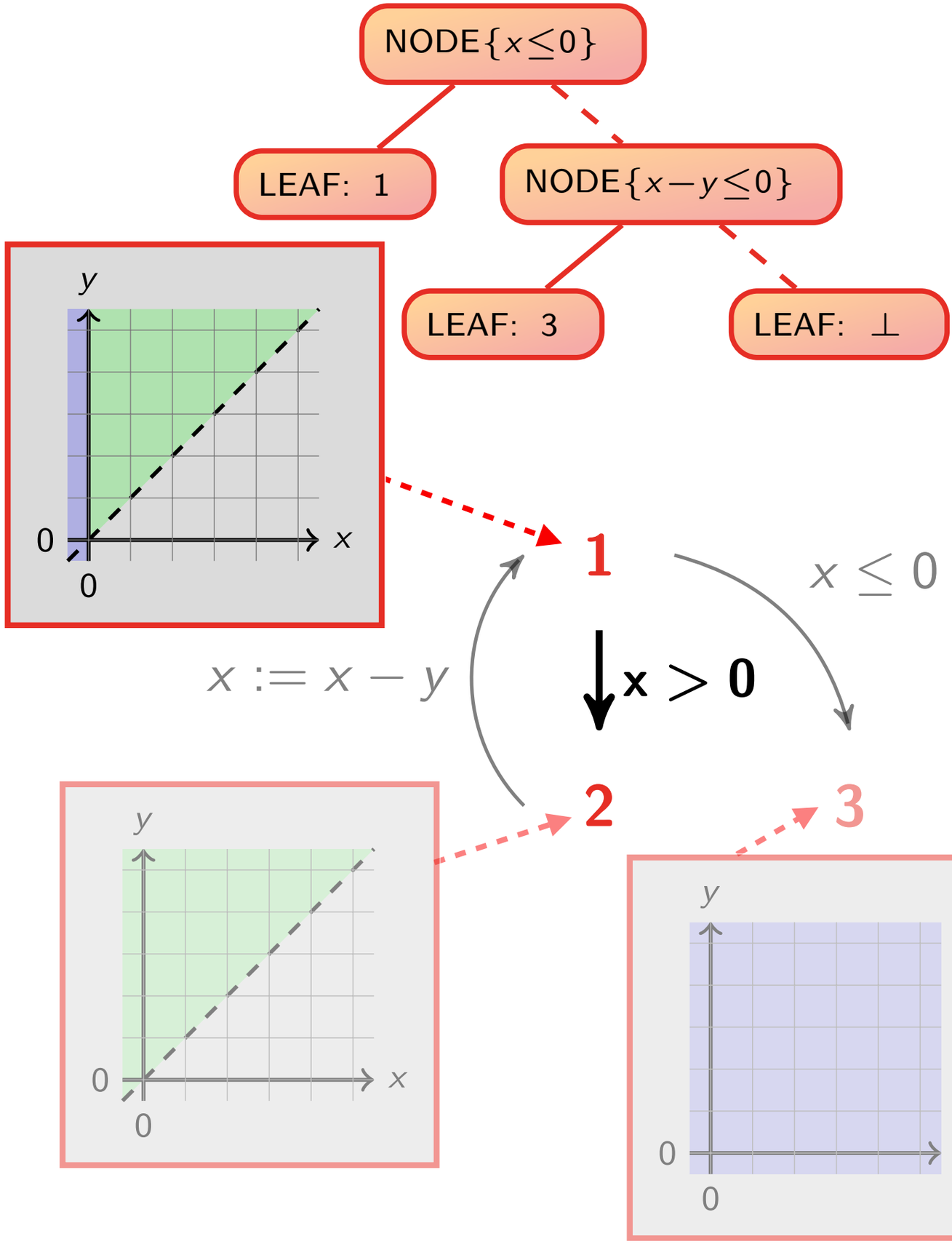
int : x, y

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

  [2] $x := x - y$

od [3]

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



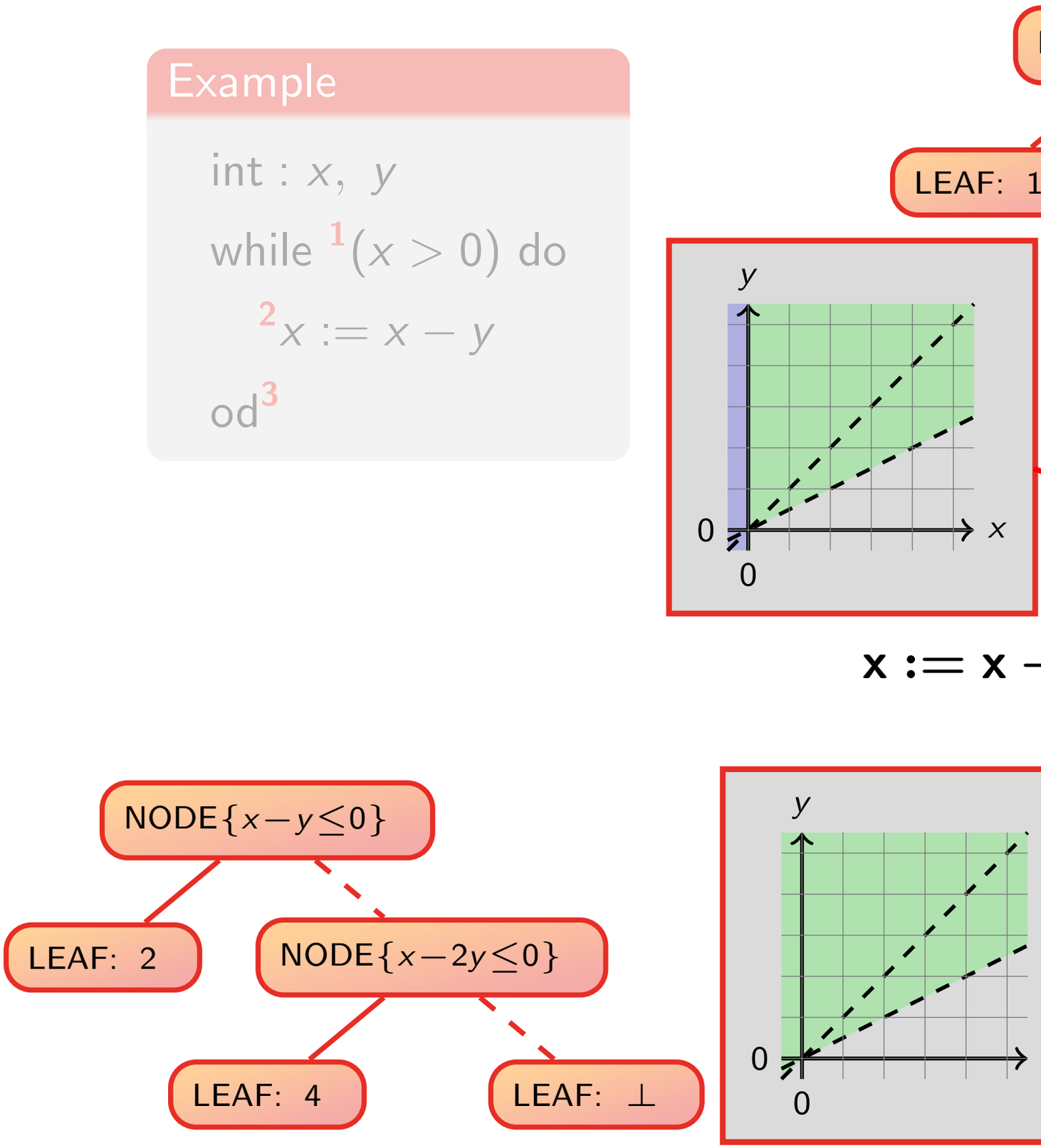
Example

int : x, y

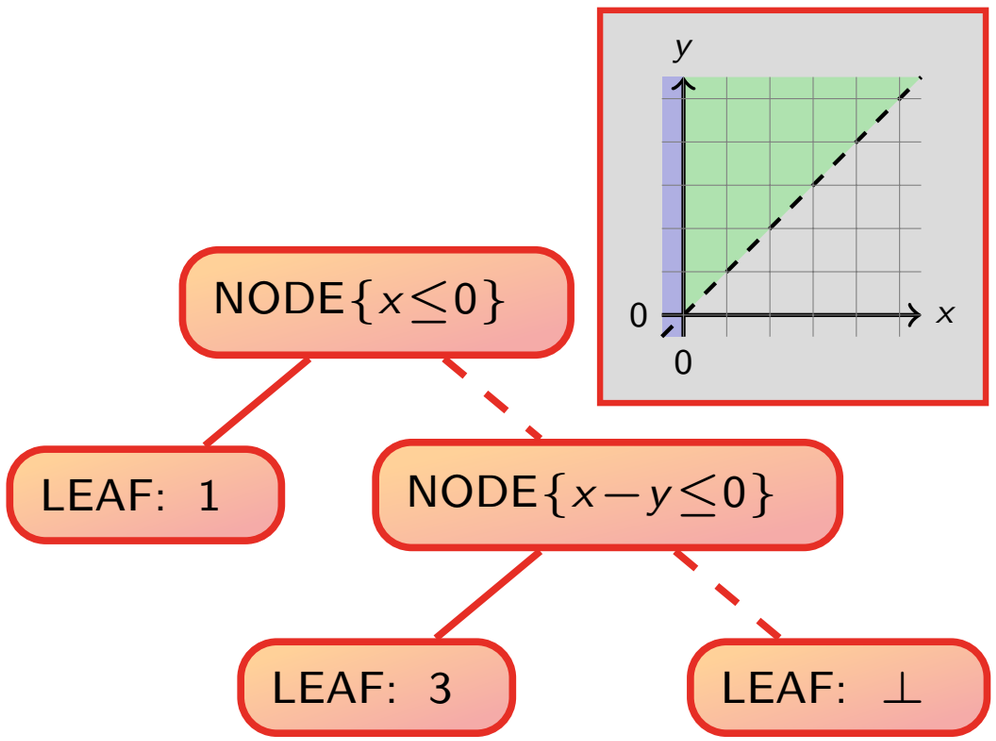
while [1] $(x > 0)$ do

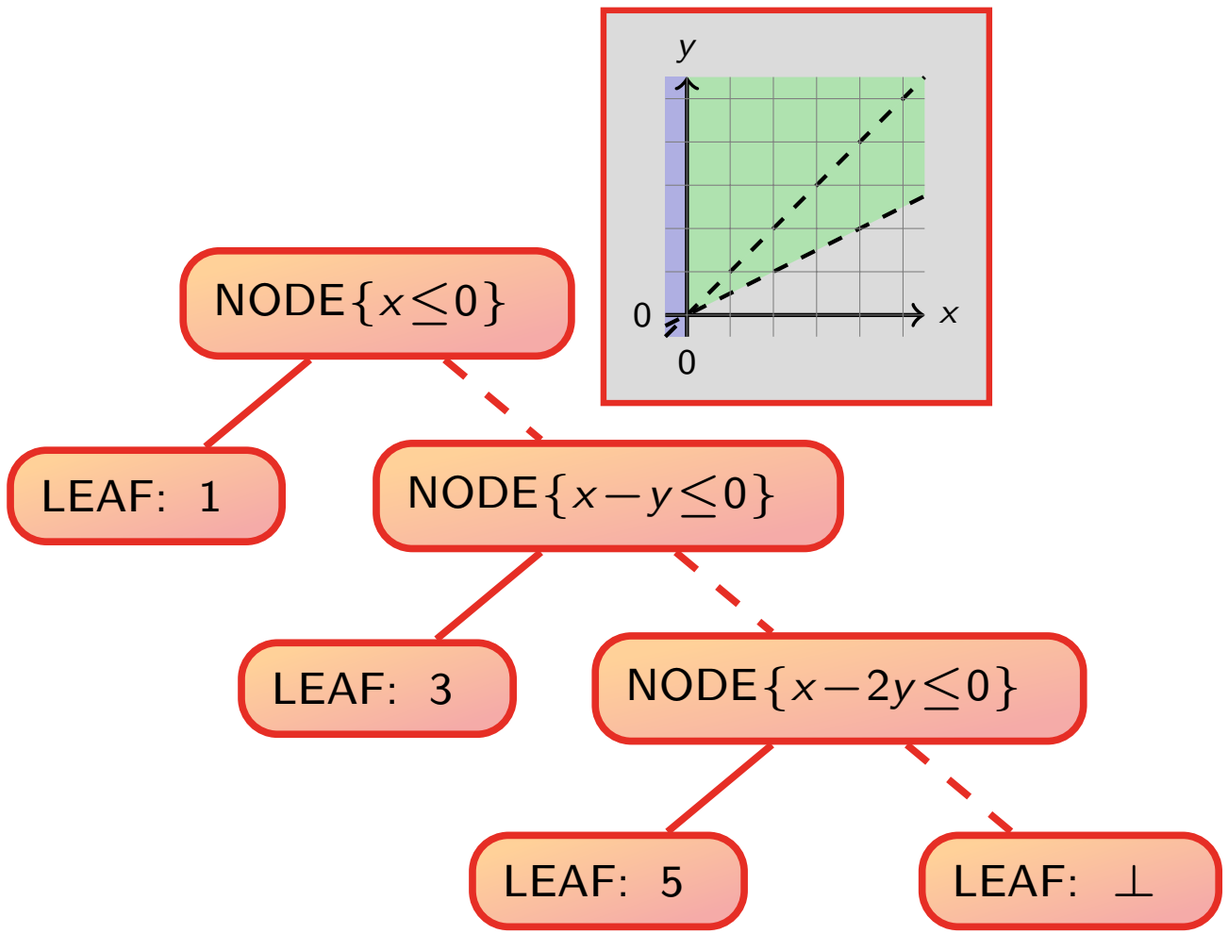
  [2] $x := x - y$

od [3]

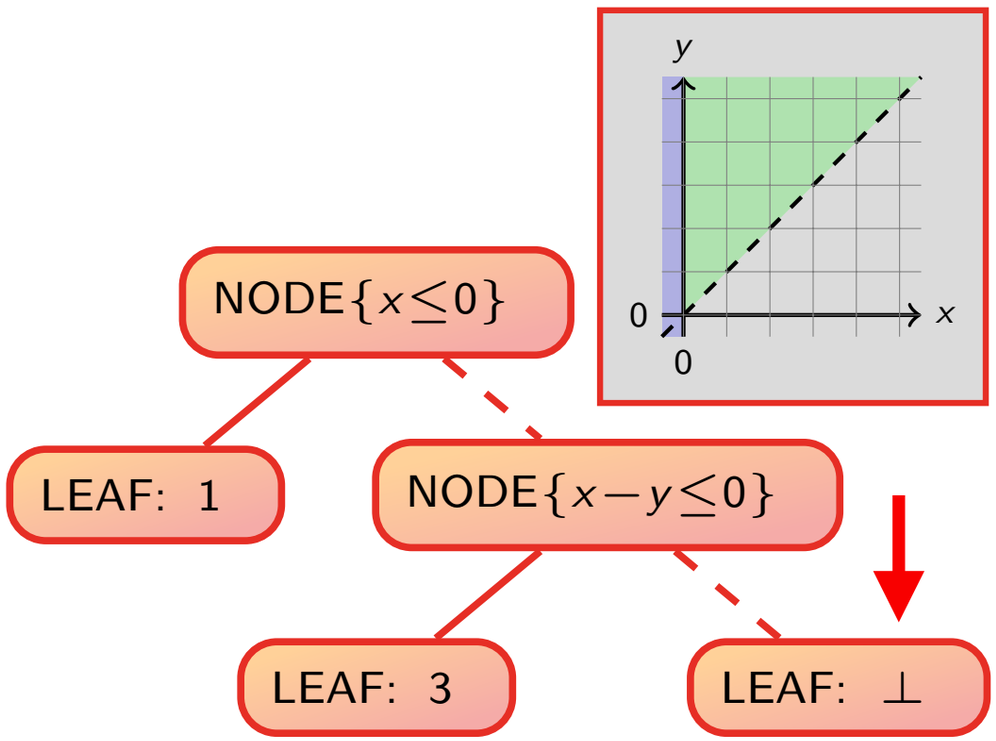


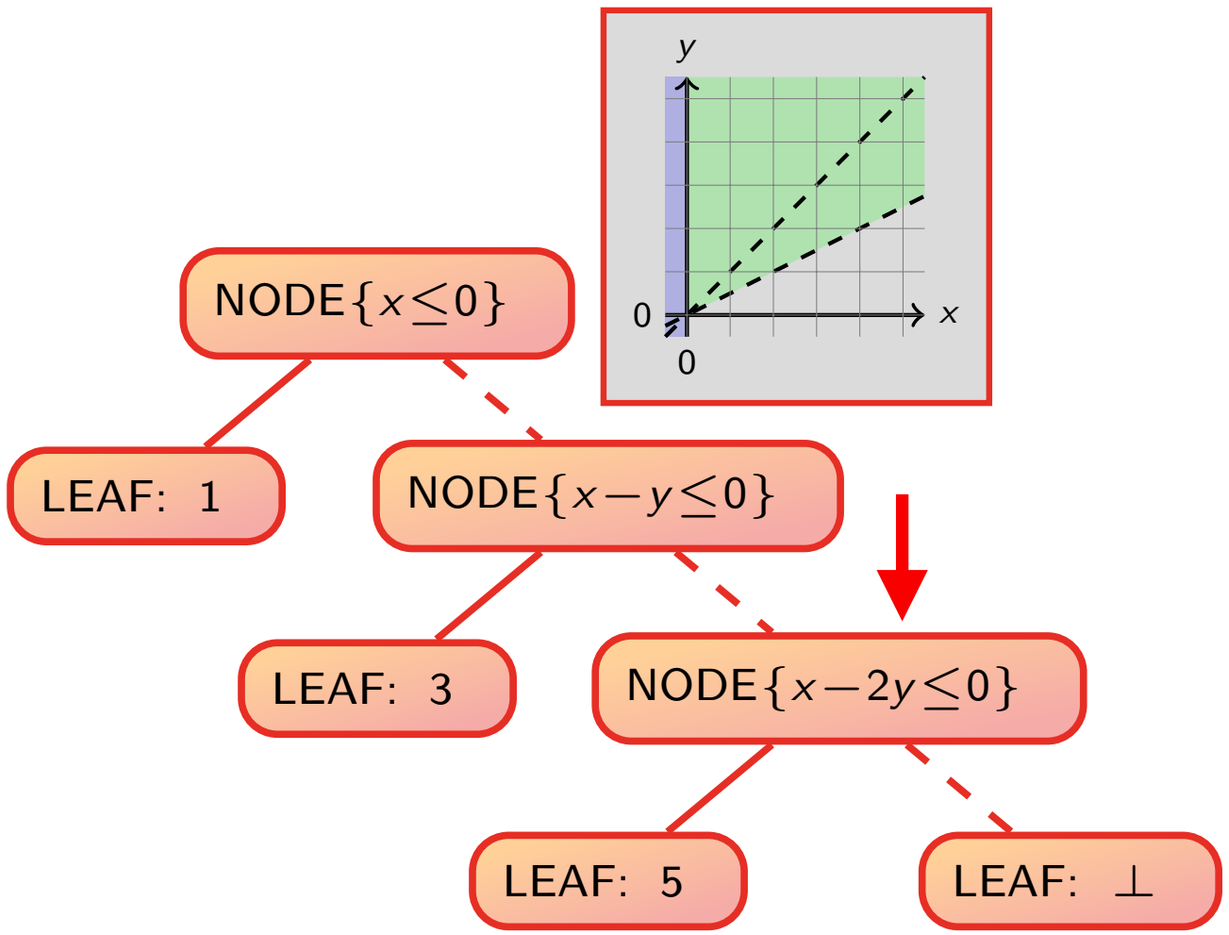
Widening


NODE{x≤0}
y
0
0
x
LEAF: 1
NODE{x−y≤0}
LEAF: 3
LEAF: ⊥

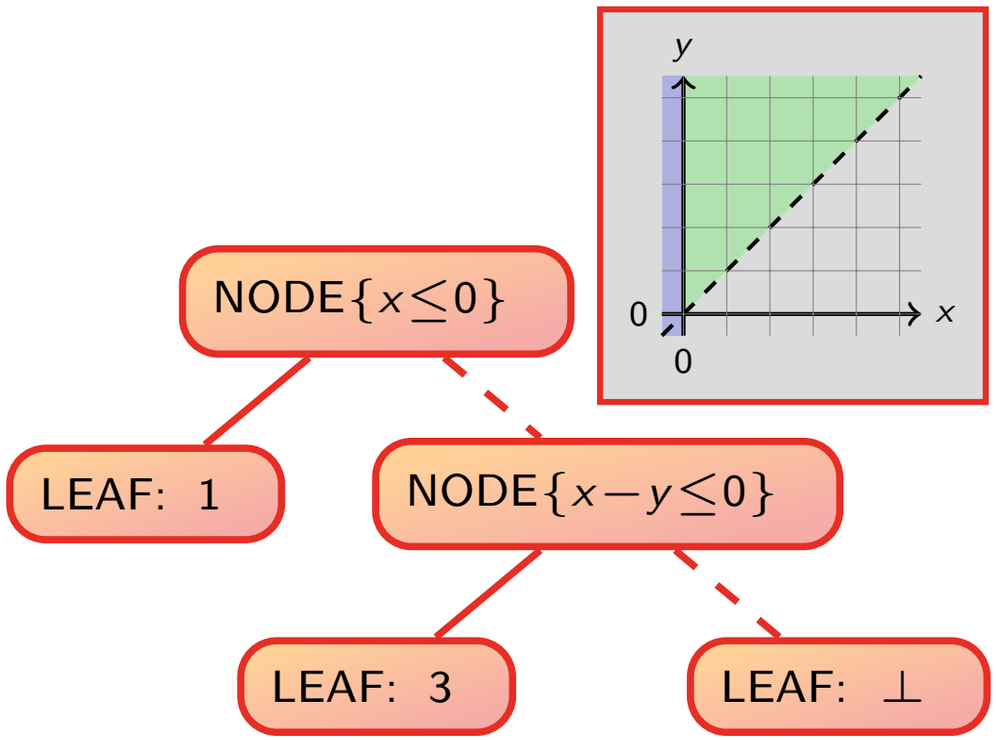

NODE{x≤0}
y
0
0
x
LEAF: 1
NODE{x−y≤0}
LEAF: 3
NODE{x−2y≤0}
LEAF: 5
LEAF: ⊥

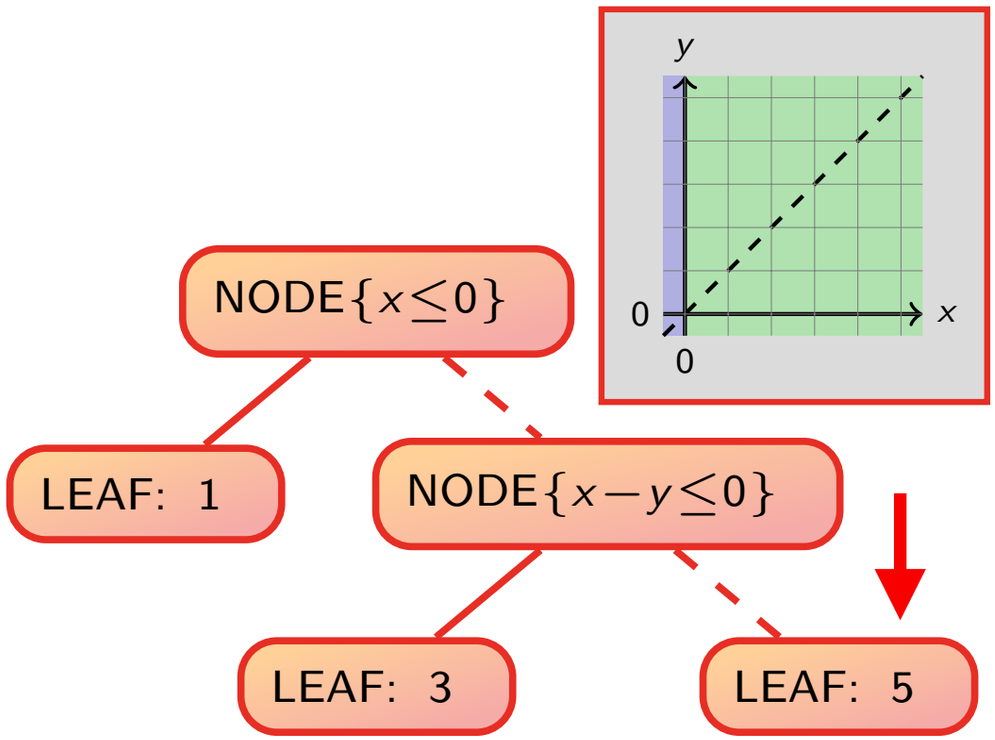
Widening


NODE{x≤0}
y
0
0
x
LEAF: 1
NODE{x−y≤0}
LEAF: 3
LEAF: ⊥

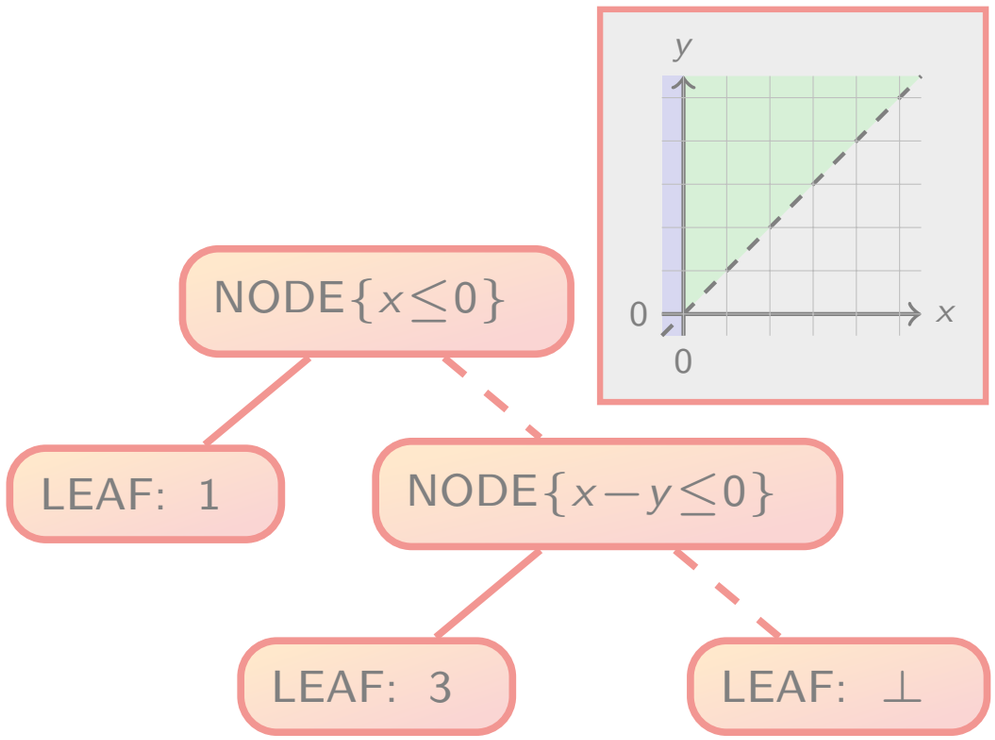

NODE{x≤0}
y
0
0
x
LEAF: 1
NODE{x−y≤0}
LEAF: 3
NODE{x−2y≤0}
LEAF: 5
LEAF: ⊥

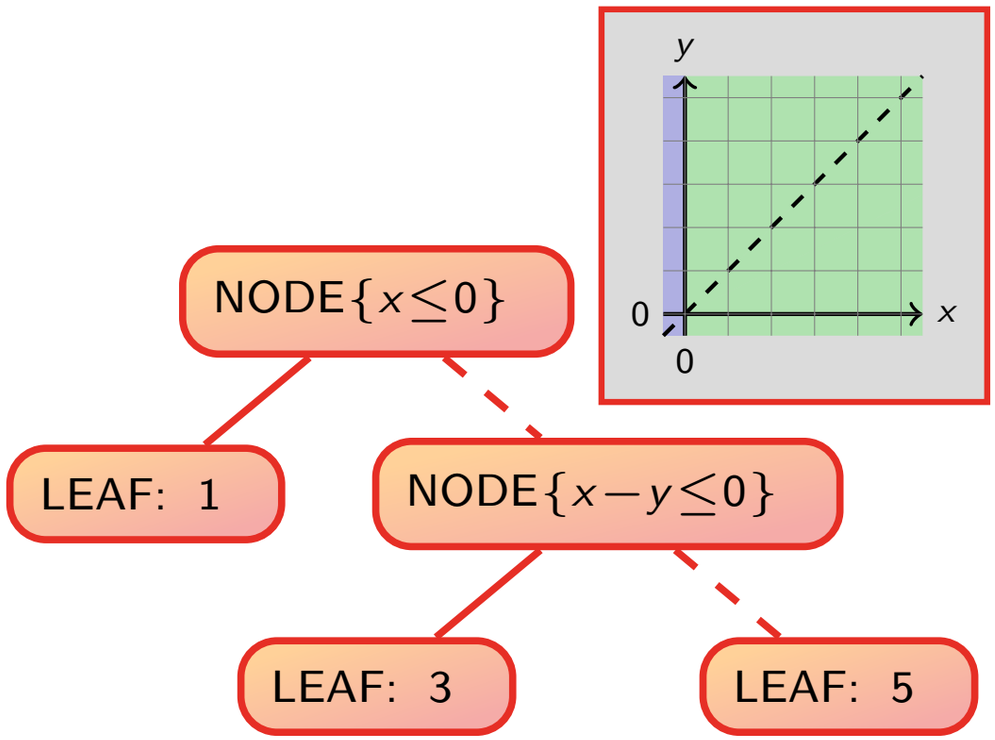
Widening


NODE{x≤0}
y
0
0
x
LEAF: 1
NODE{x−y≤0}
LEAF: 3
LEAF: ⊥


NODE{x≤0}
y
0
0
x
LEAF: 1
NODE{x−y≤0}
LEAF: 3
LEAF: 5

Widening


NODE{x≤0}
LEAF: 1
NODE{x−y≤0}
LEAF: 3
LEAF: ⊥
y
x
0
0


NODE{x≤0}
LEAF: 1
NODE{x−y≤0}
LEAF: 3
LEAF: 5
y
x
0
0

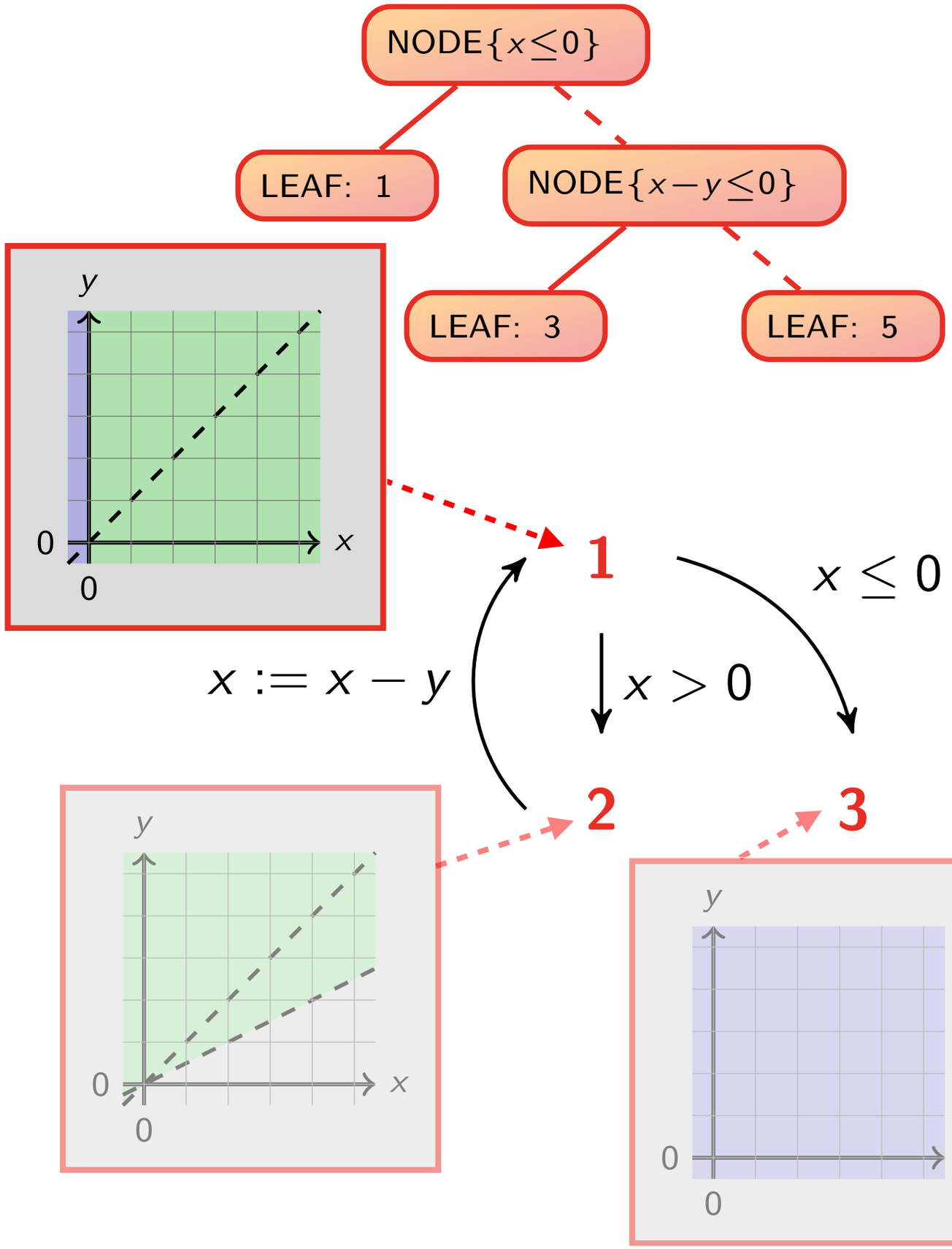
Example

int : $x,\ y$

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

  [2] $x := x - y$

od [3]



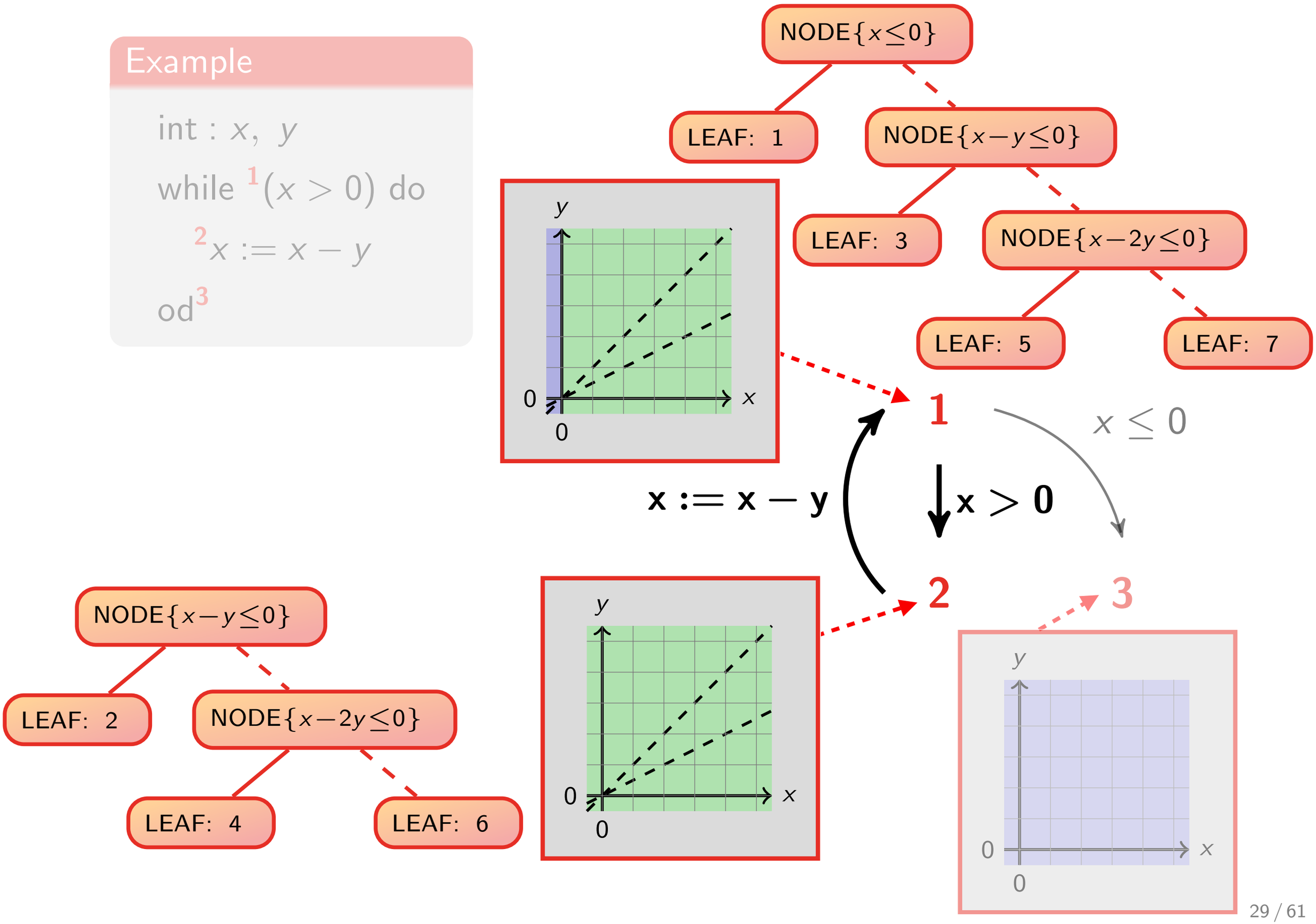
Example

int : x, y

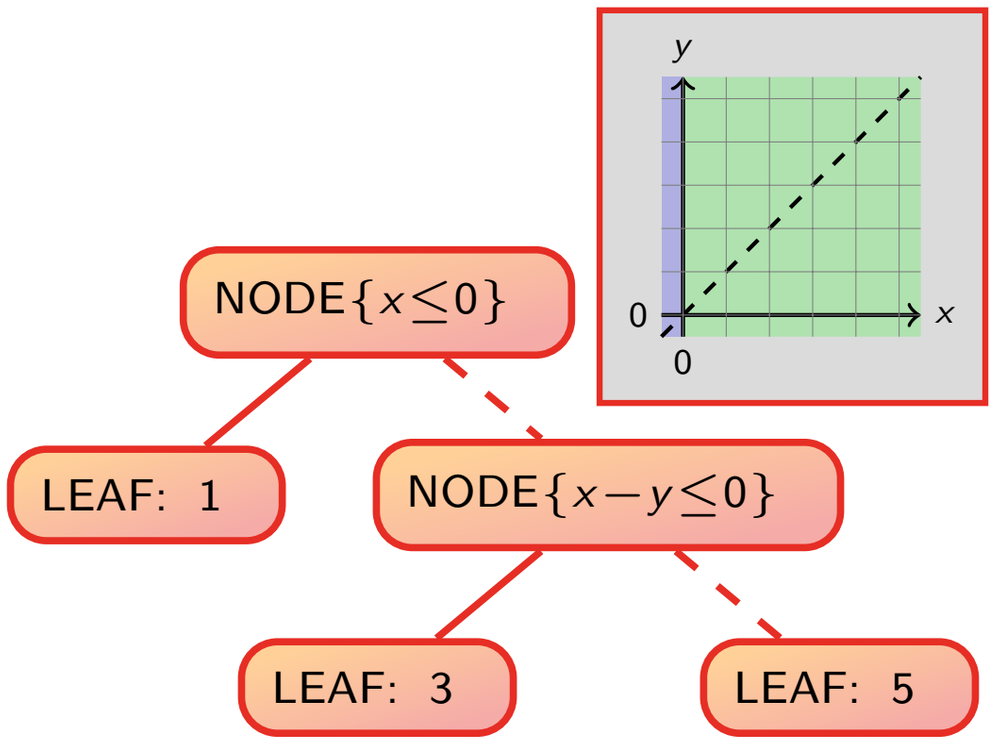
while [1] $(x > 0)$ do

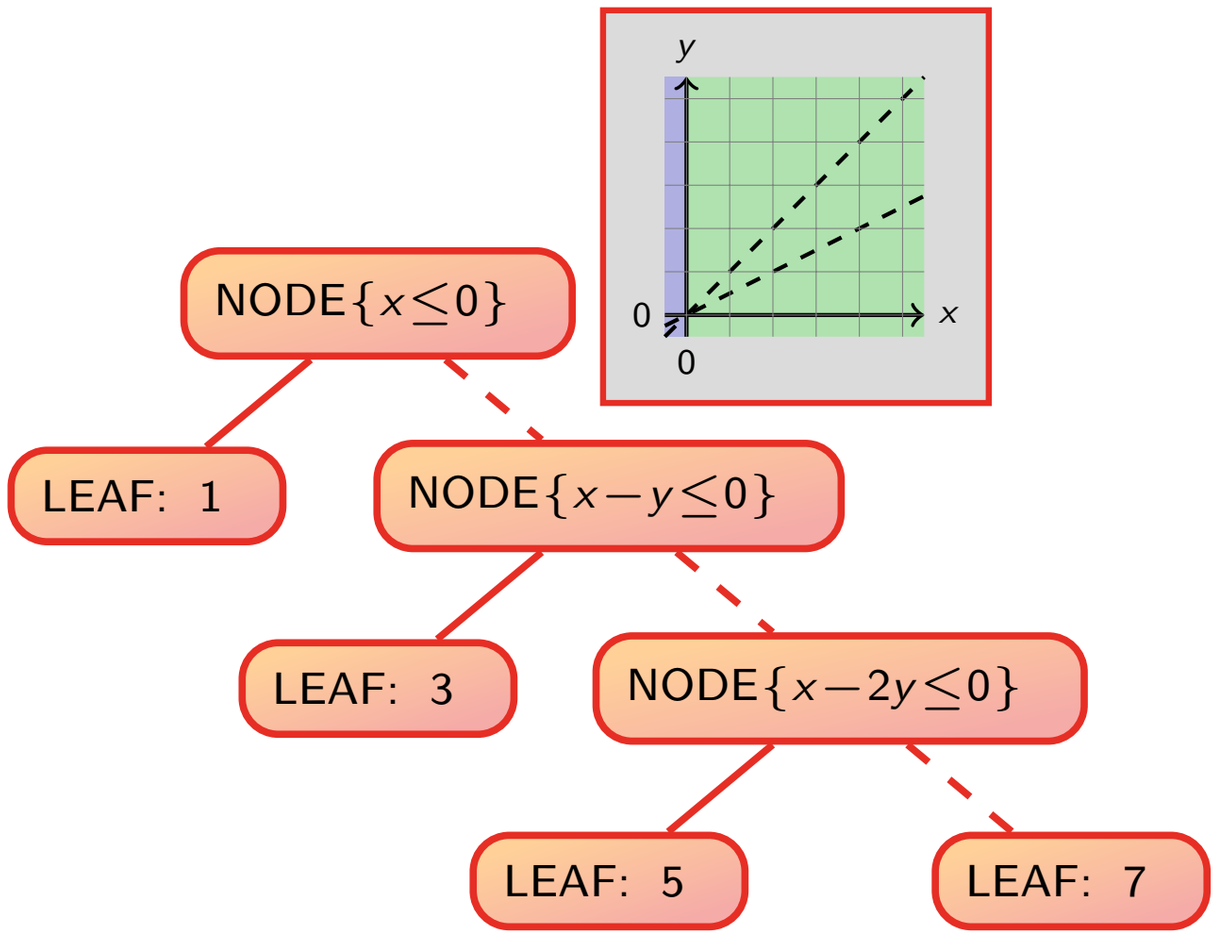
  [2] $x := x - y$

od [3]

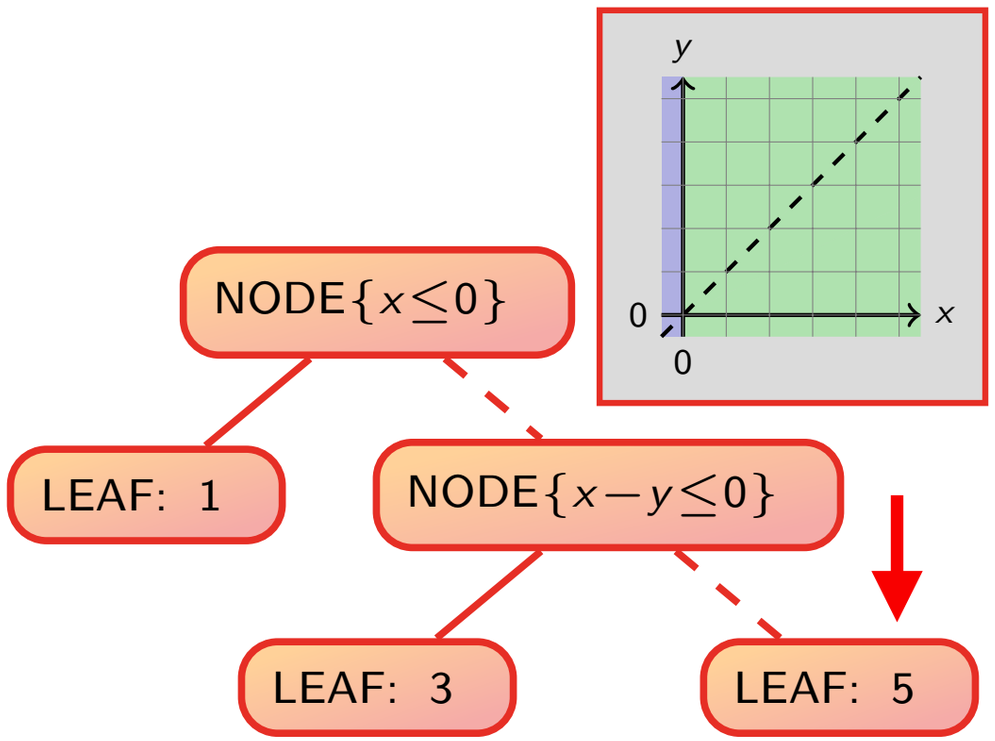


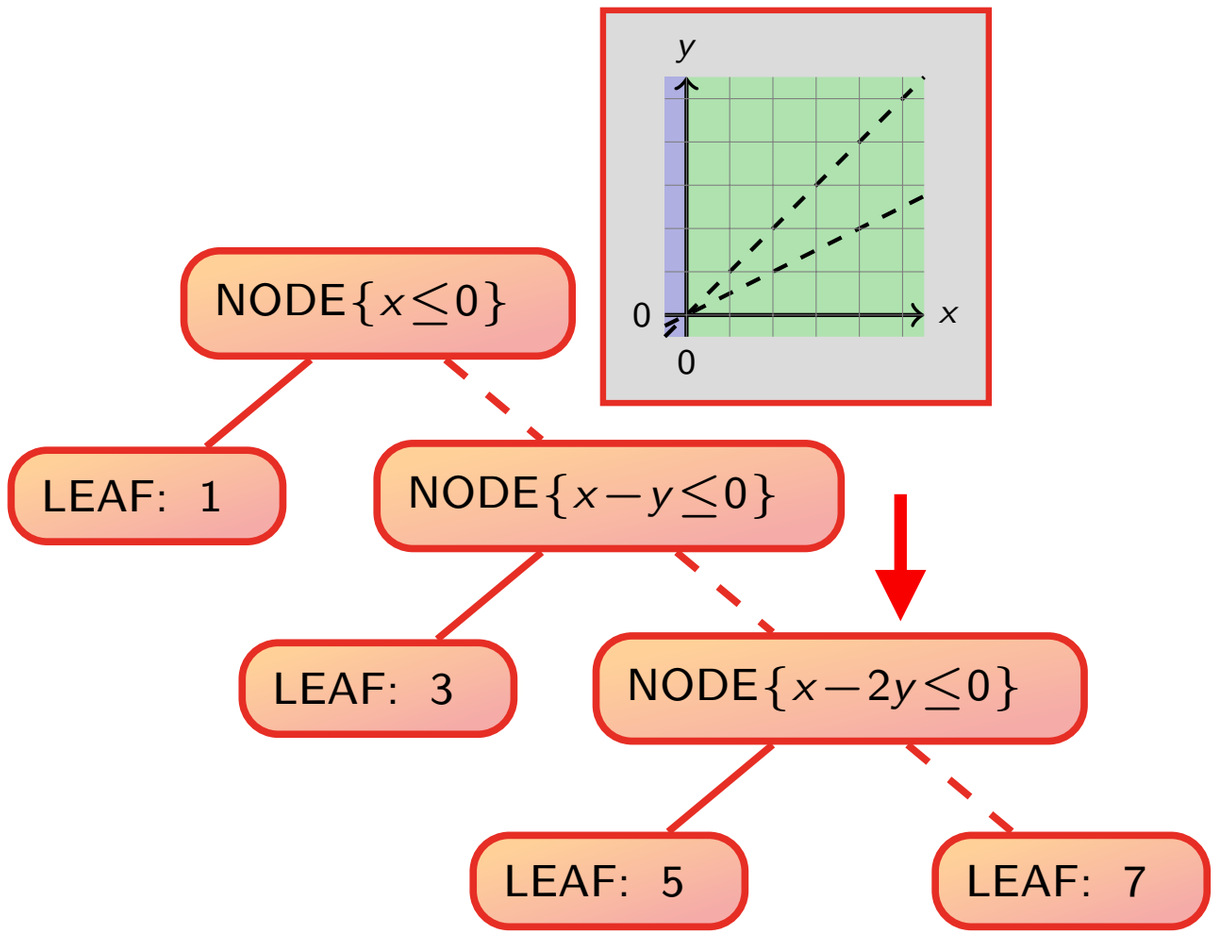
Widening


NODE{x≤0}
y
0
0
x
LEAF: 1
NODE{x−y≤0}
LEAF: 3
LEAF: 5

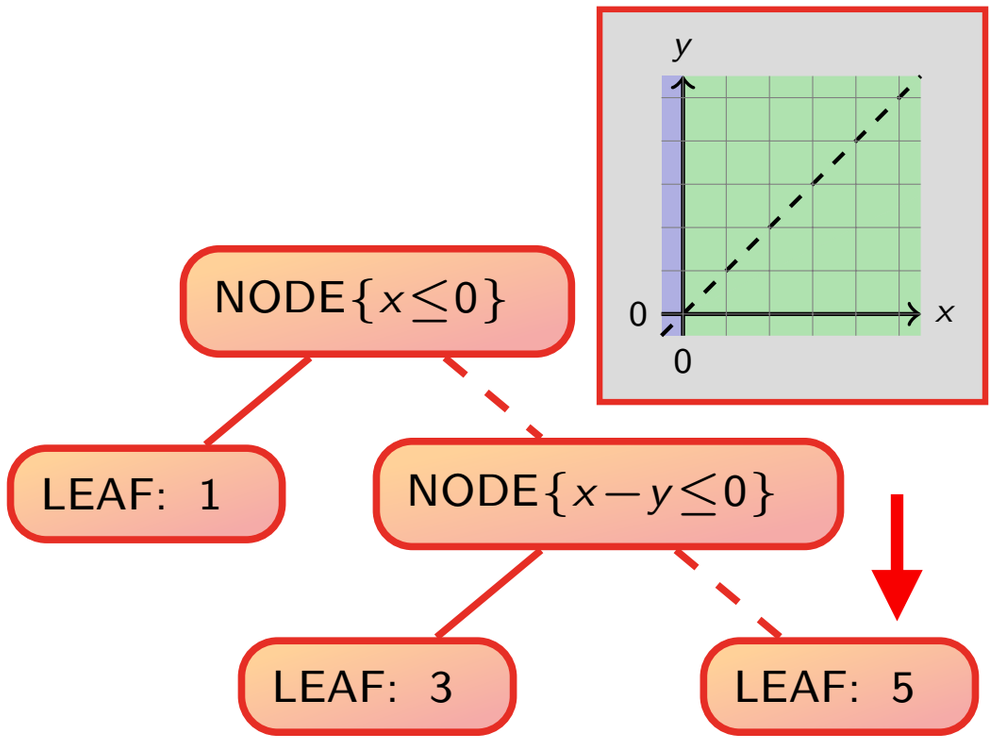

NODE{x≤0}
y
0
0
x
LEAF: 1
NODE{x−y≤0}
LEAF: 3
NODE{x−2y≤0}
LEAF: 5
LEAF: 7

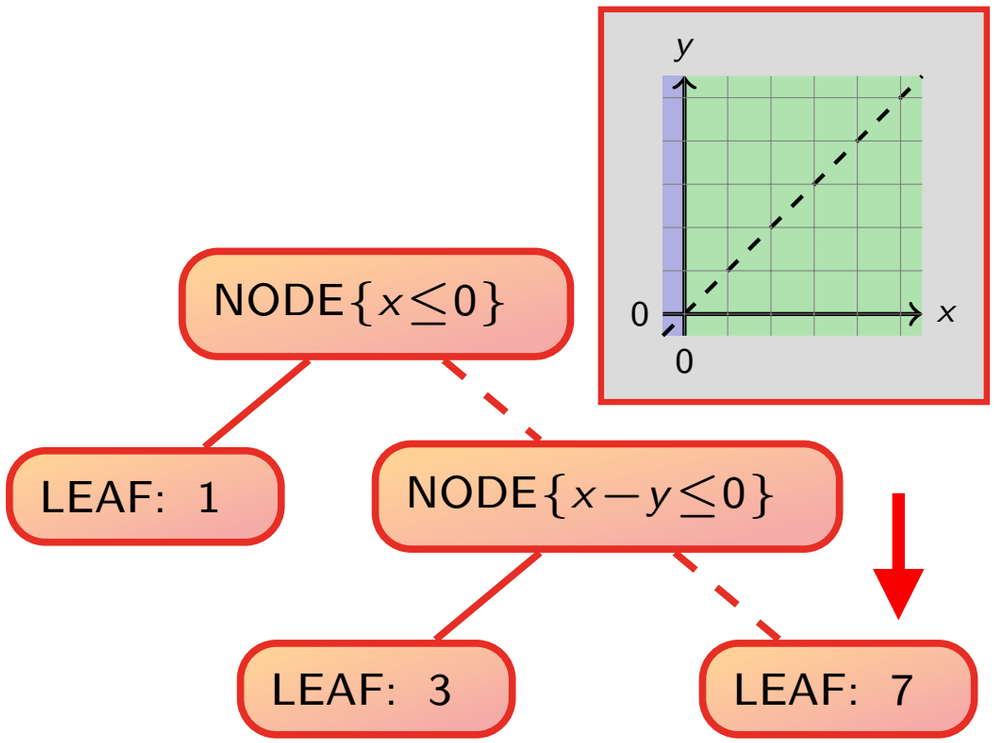
Widening


NODE{x≤0}
y
0
0
x
LEAF: 1
NODE{x−y≤0}
LEAF: 3
LEAF: 5

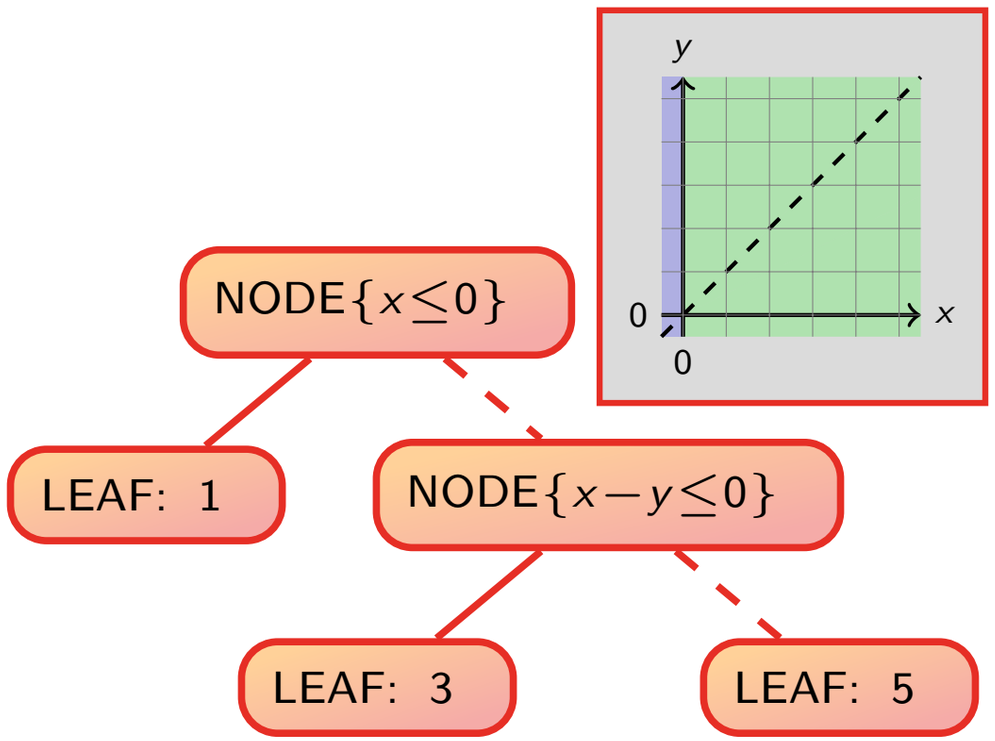

NODE{x≤0}
y
0
0
x
LEAF: 1
NODE{x−y≤0}
LEAF: 3
NODE{x−2y≤0}
LEAF: 5
LEAF: 7

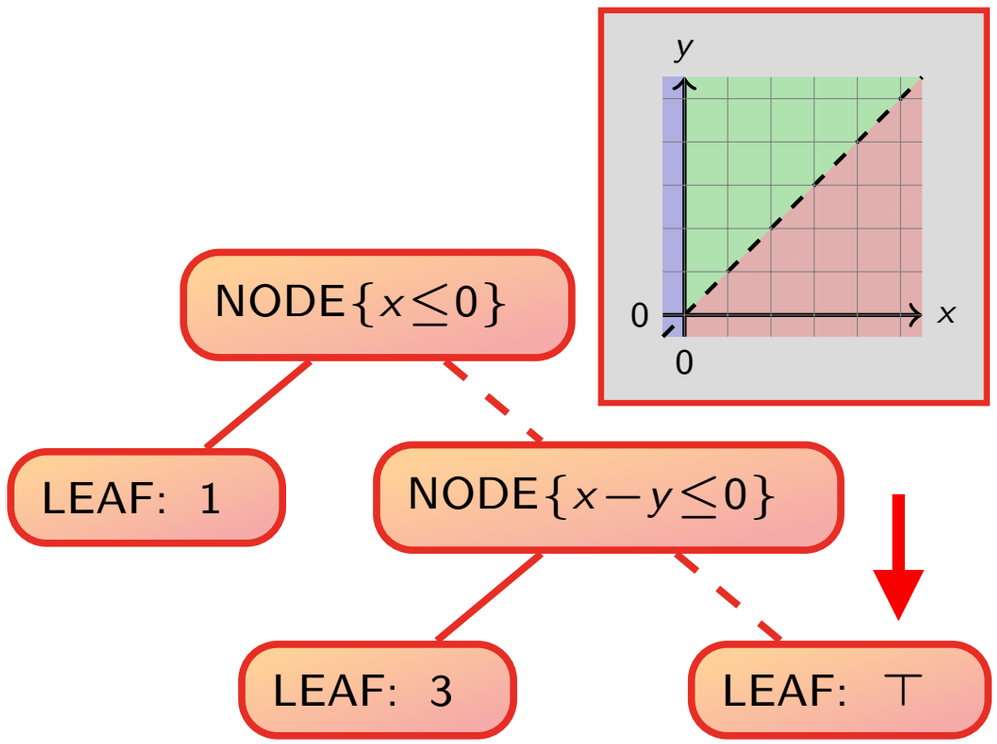
Widening


NODE{x≤0}
y
0
0
x
LEAF: 1
NODE{x−y≤0}
LEAF: 3
LEAF: 5

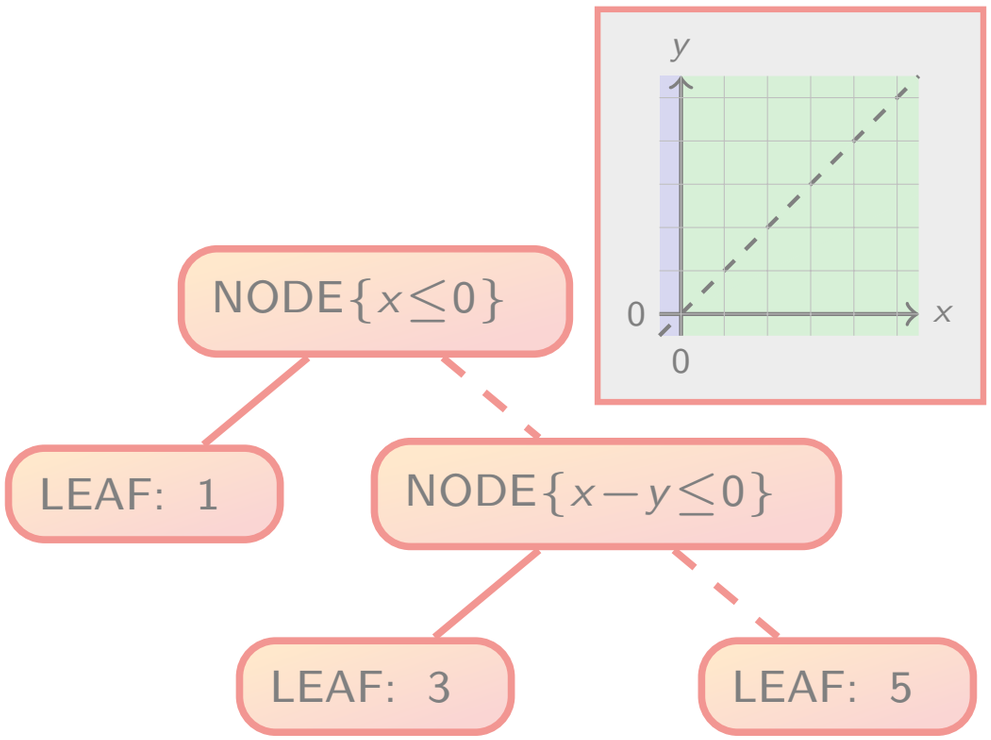

NODE{x≤0}
y
0
0
x
LEAF: 1
NODE{x−y≤0}
LEAF: 3
LEAF: 7

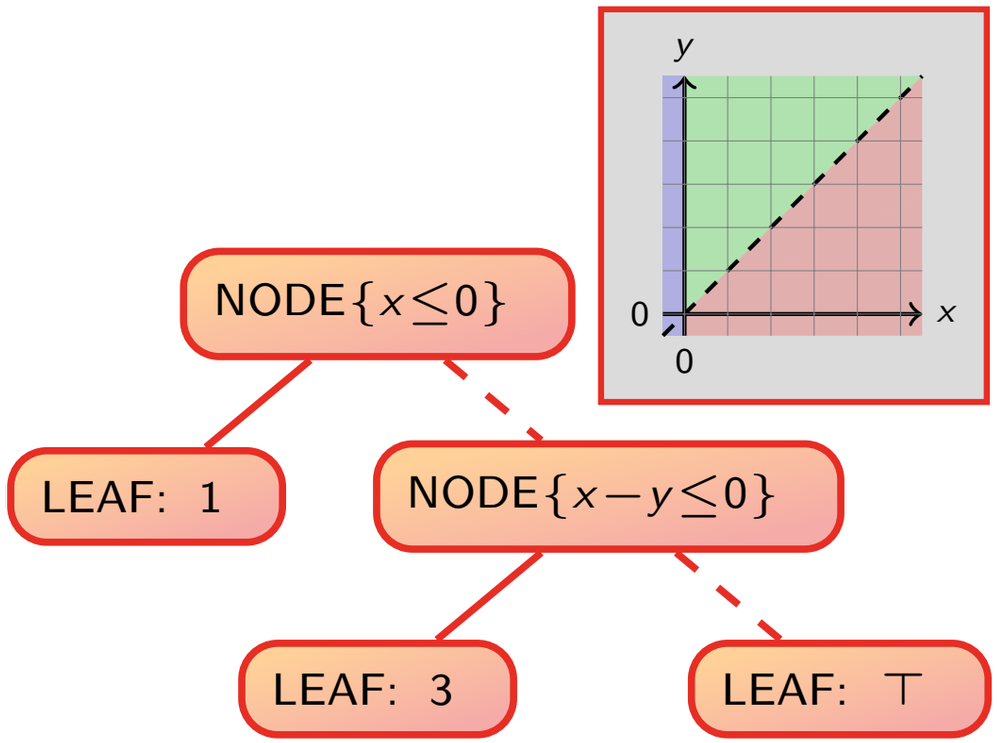
Widening


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


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

Widening


NODE{x≤0}
LEAF: 1
NODE{x−y≤0}
LEAF: 3
LEAF: 5
y
x
0
0


NODE{x≤0}
LEAF: 1
NODE{x−y≤0}
LEAF: 3
LEAF: ⊤
y
x
0
0

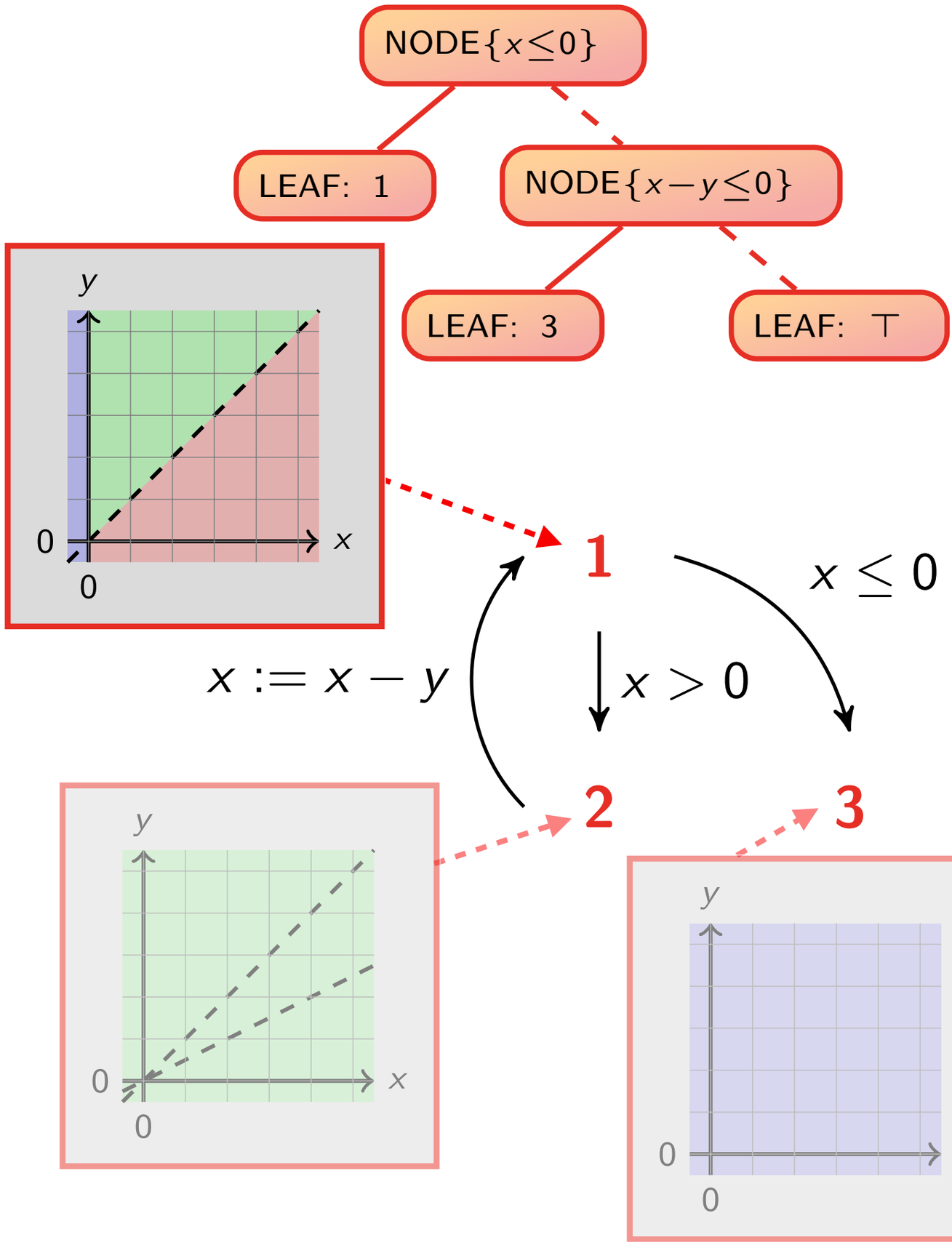
Example

int : x, y

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

  [2] $x := x - y$

od [3]



Example

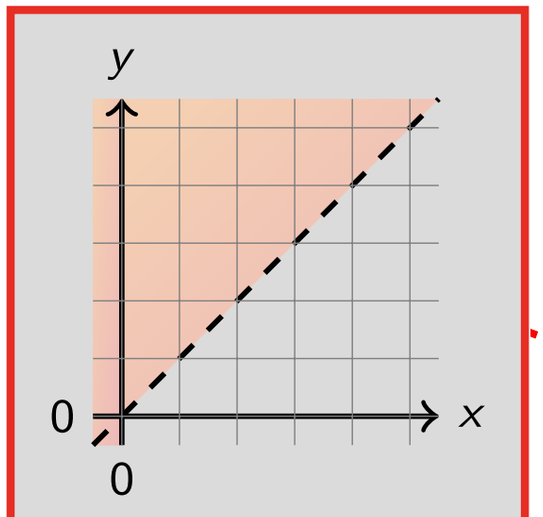
int : $x,\ y$

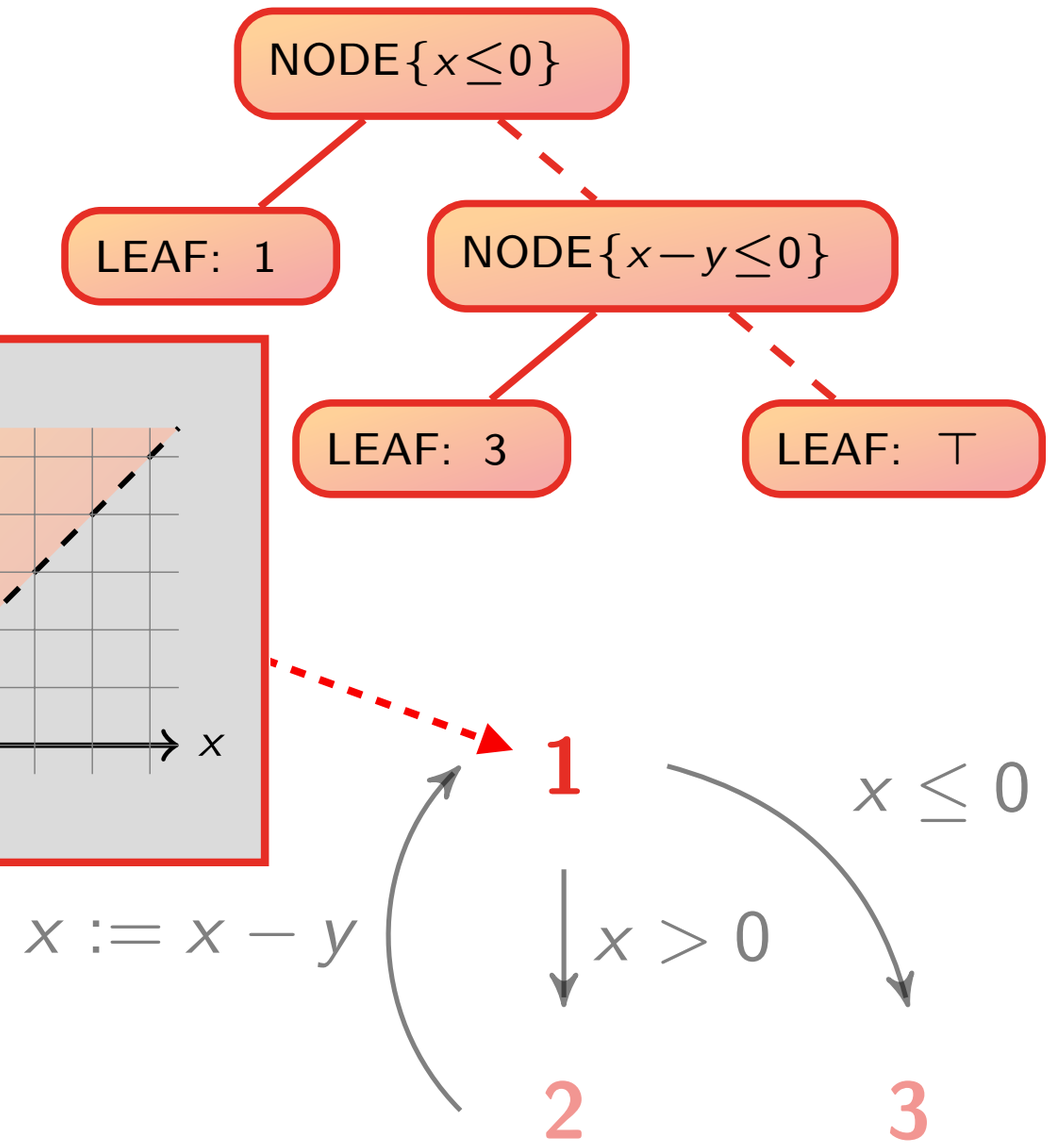
while [1] $(x > 0)$ do

  [2] $x := x - y$

od [3]

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





Example

int : $x,\ y$

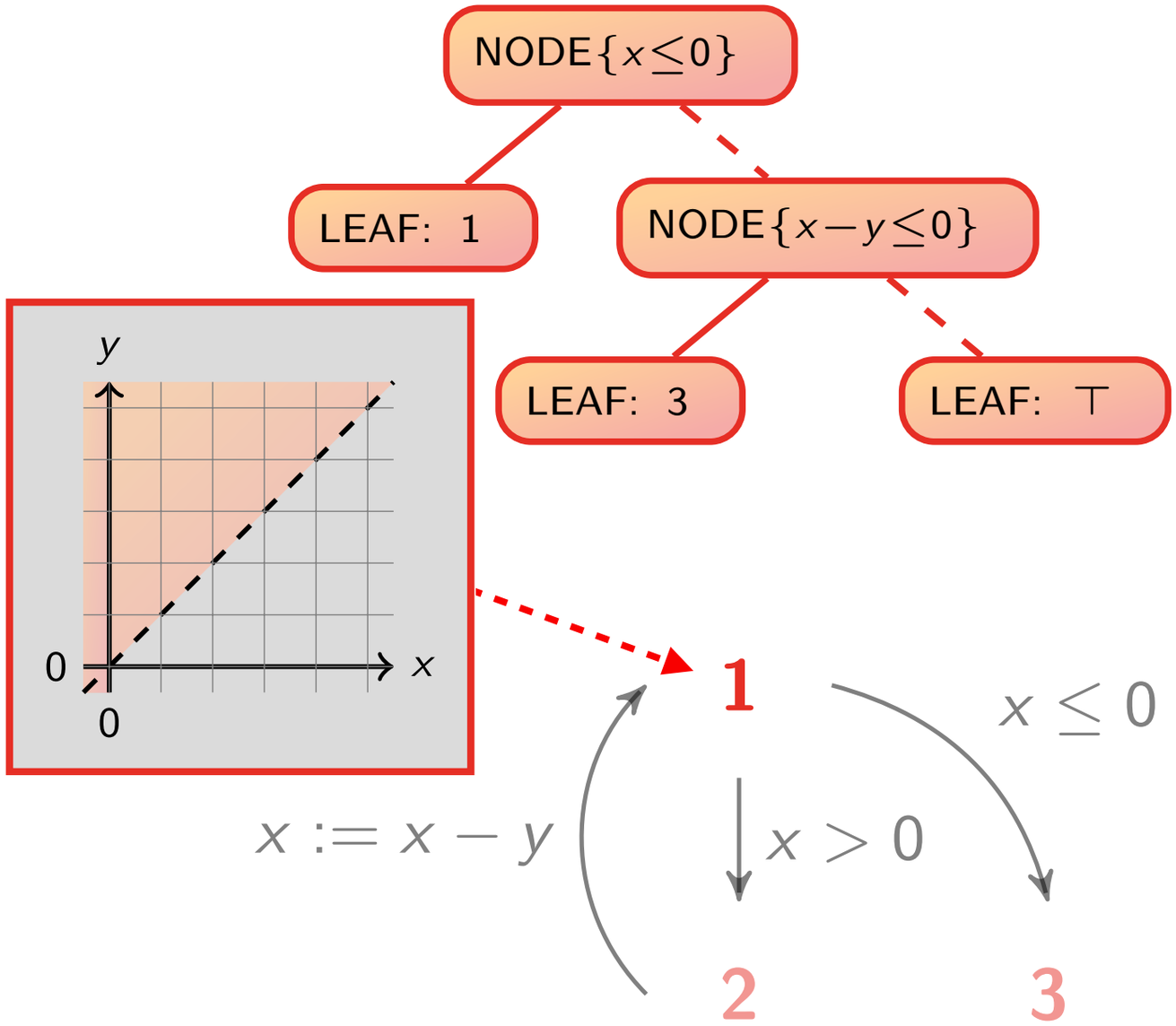
while [1] $(x > 0)$ do

  [2] $x := x - y$

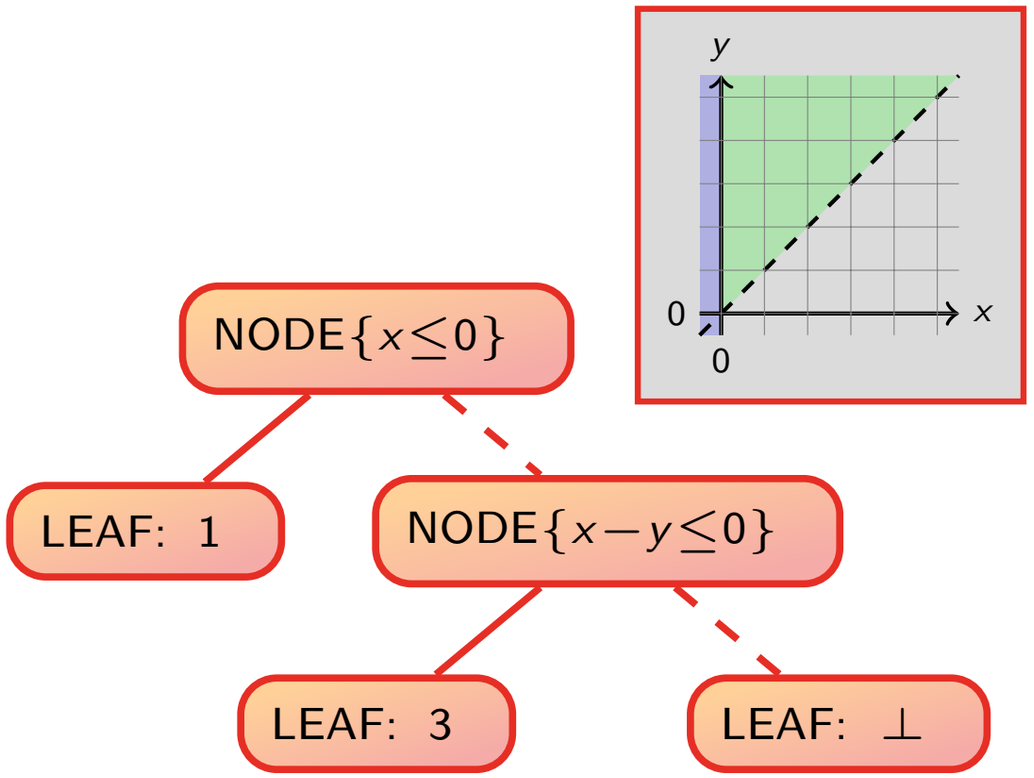
od [3]

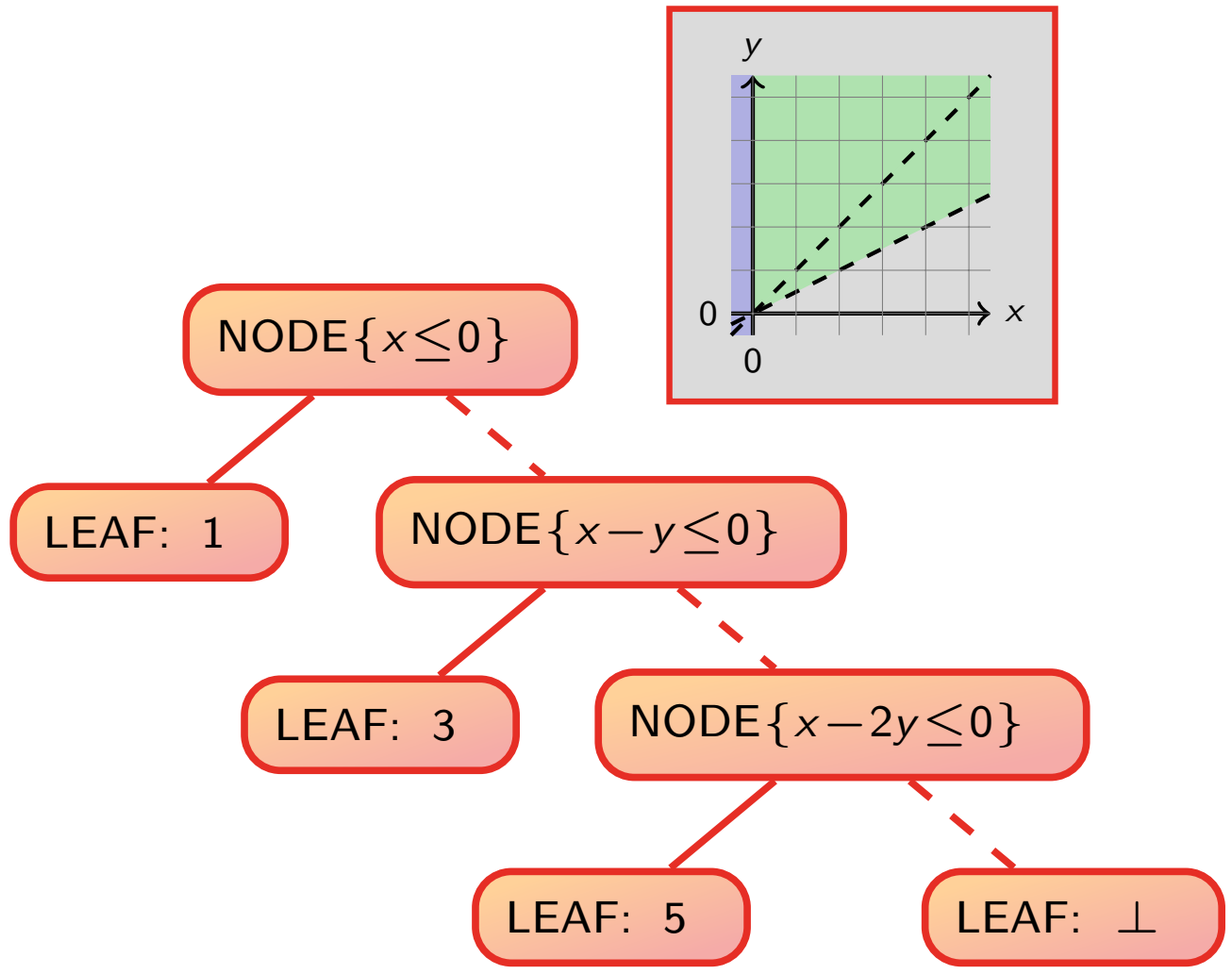
the analysis gives $x \leq 0 \vee x \leq y$ as **sufficient precondition**

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



Better Widening





Precise Widening Operators for Convex Polyhedra*

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

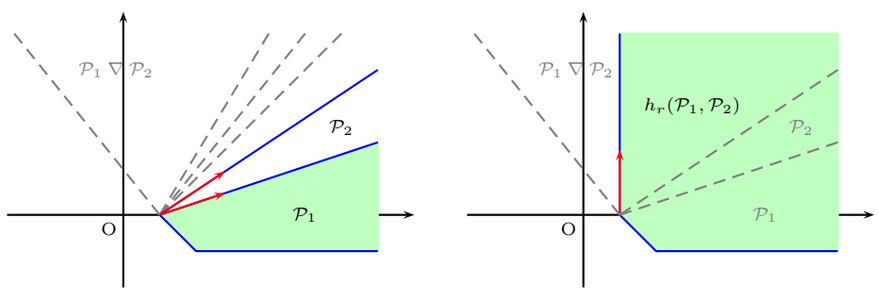
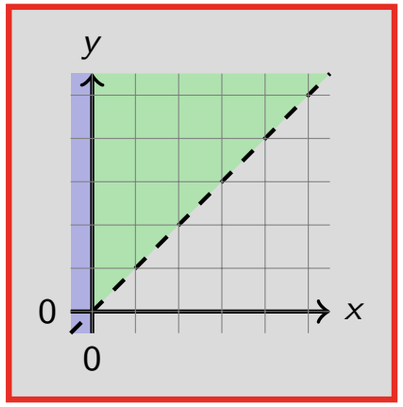
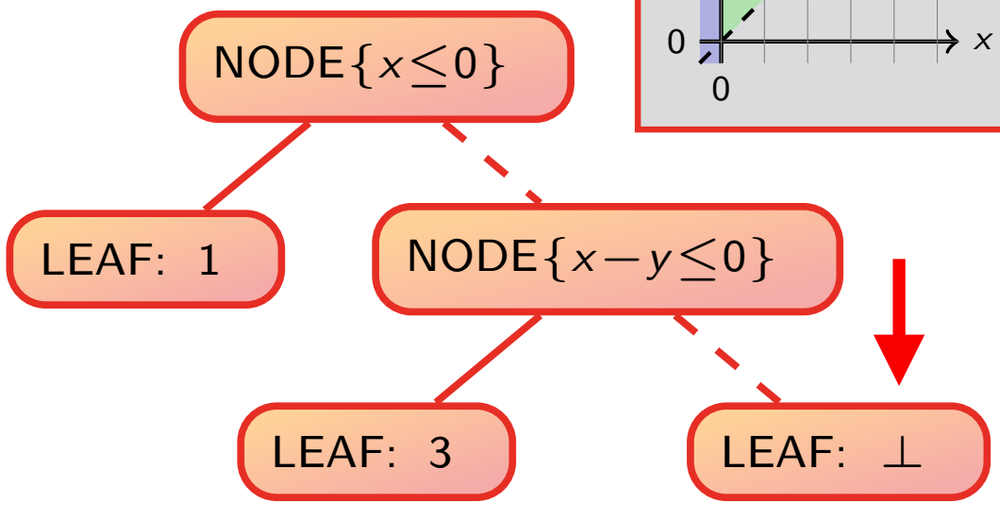
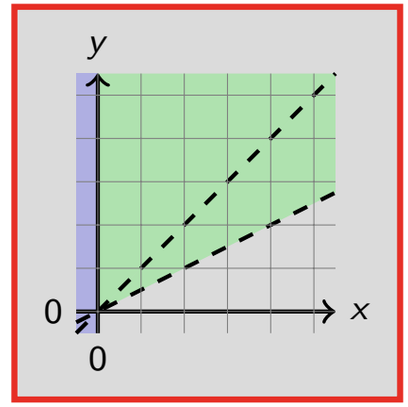


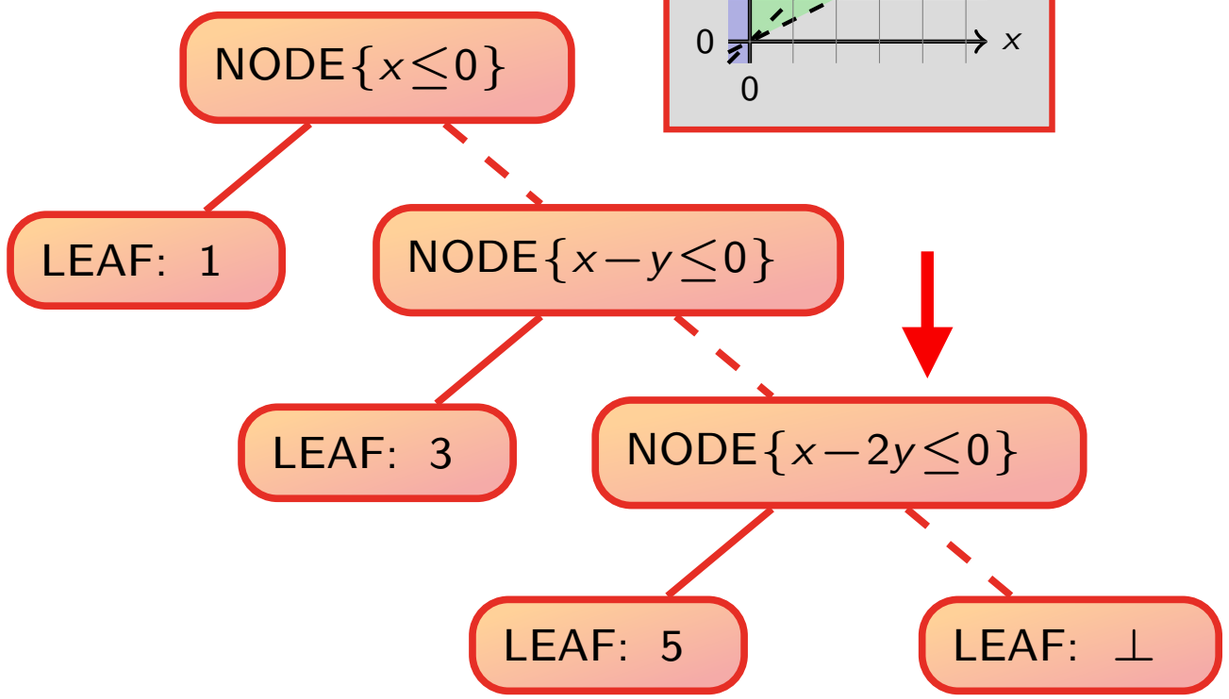
Fig. 2. The heuristics h_r improving on the standard widening.

Better Widening









Precise Widening Operators for Convex Polyhedra*

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

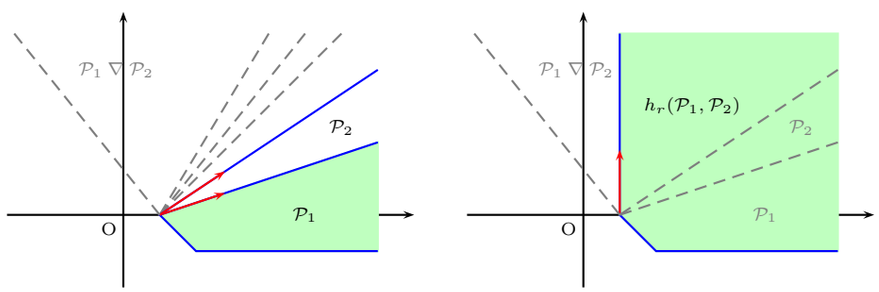
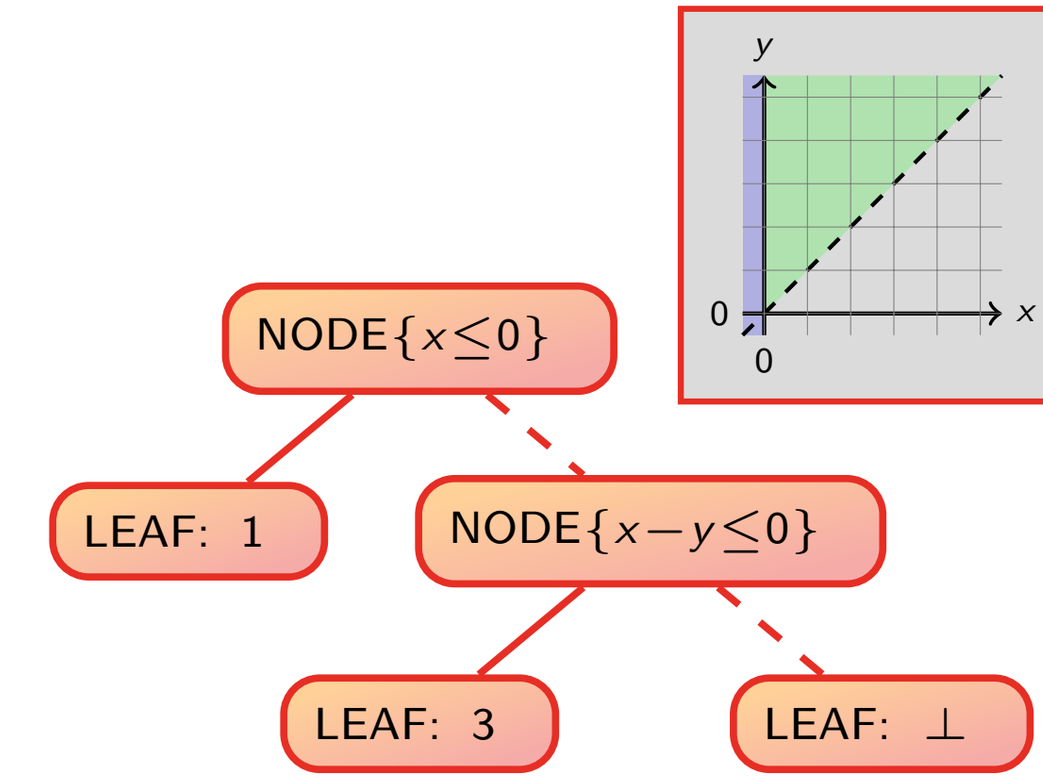
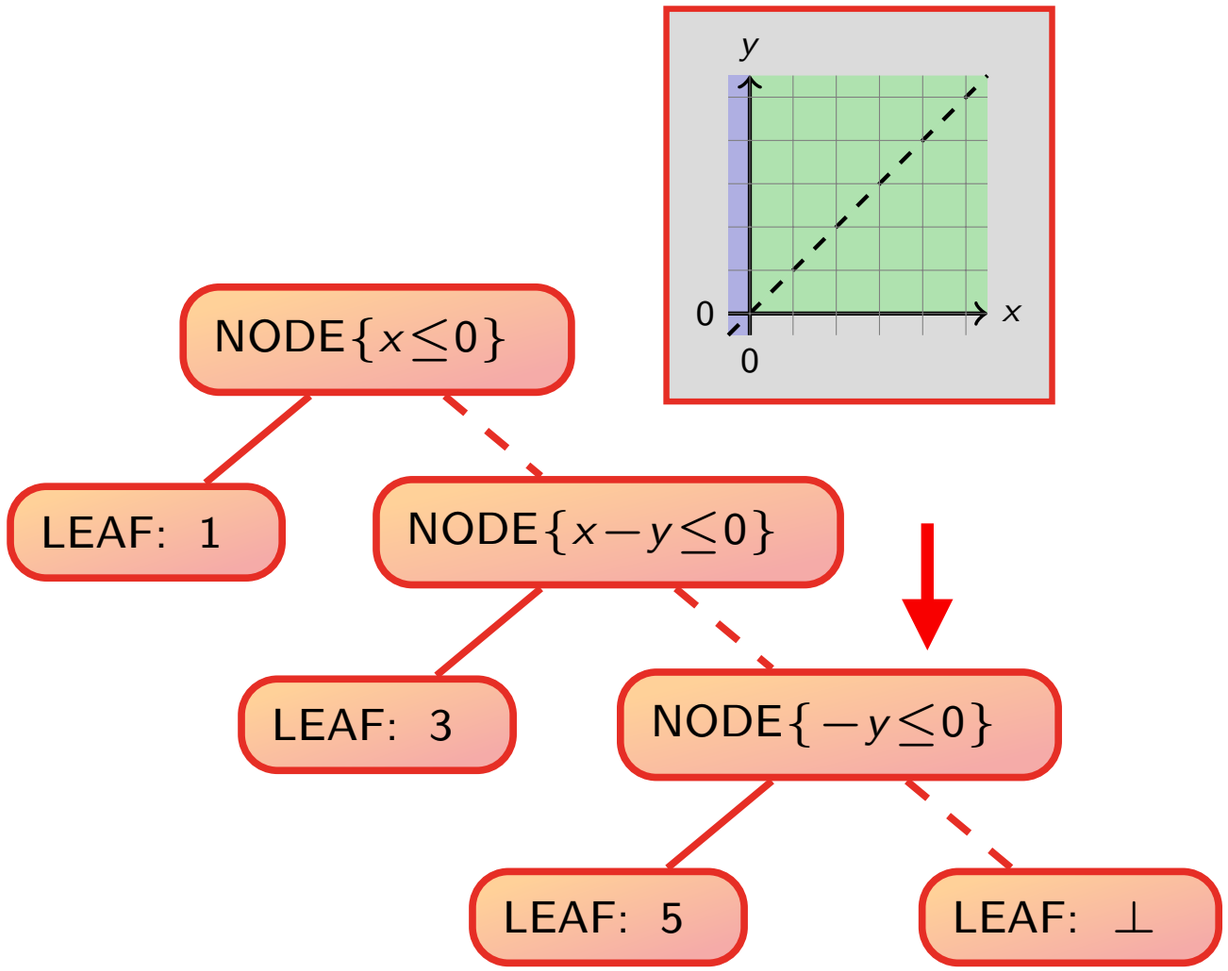


Fig. 2. The heuristics h_r improving on the standard widening.

Better Widening





Precise Widening Operators for Convex Polyhedra*

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

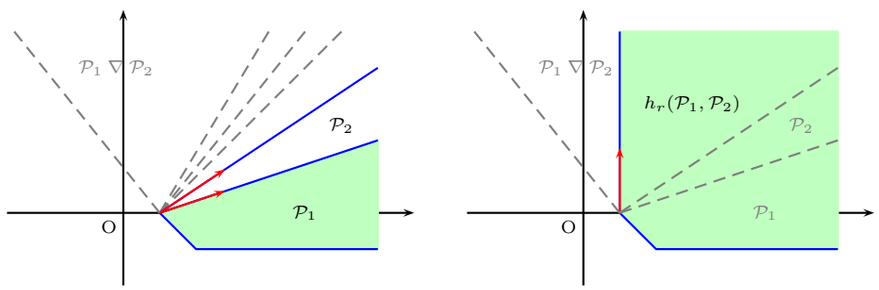
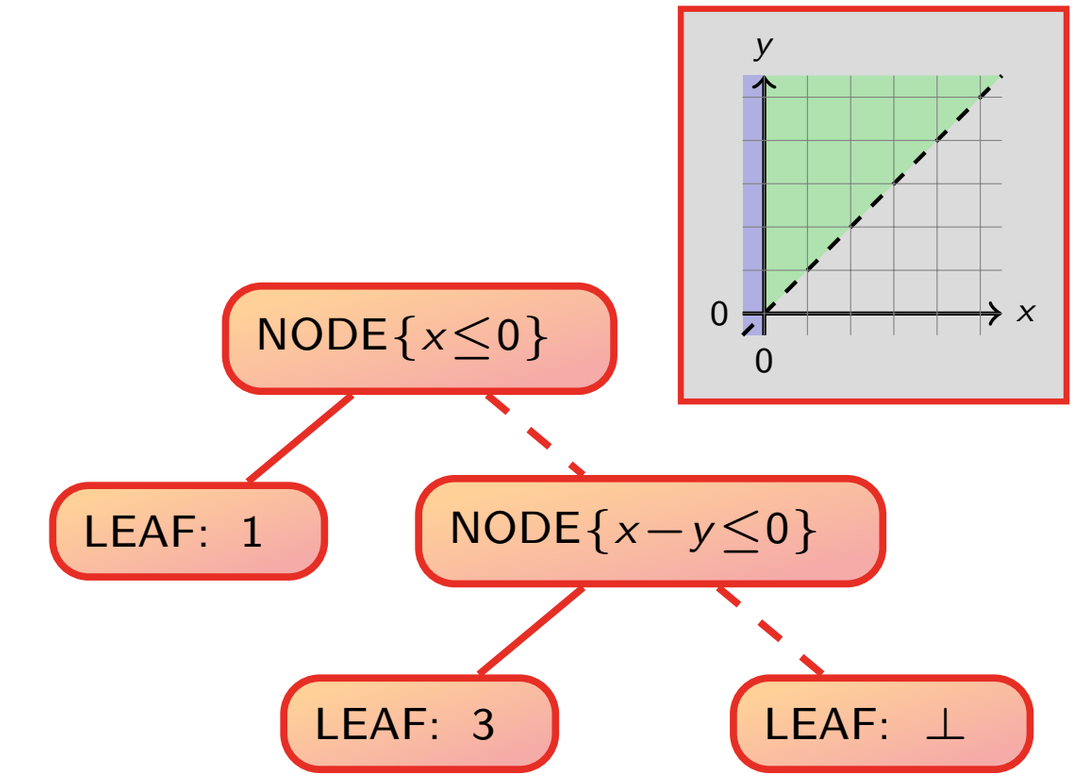
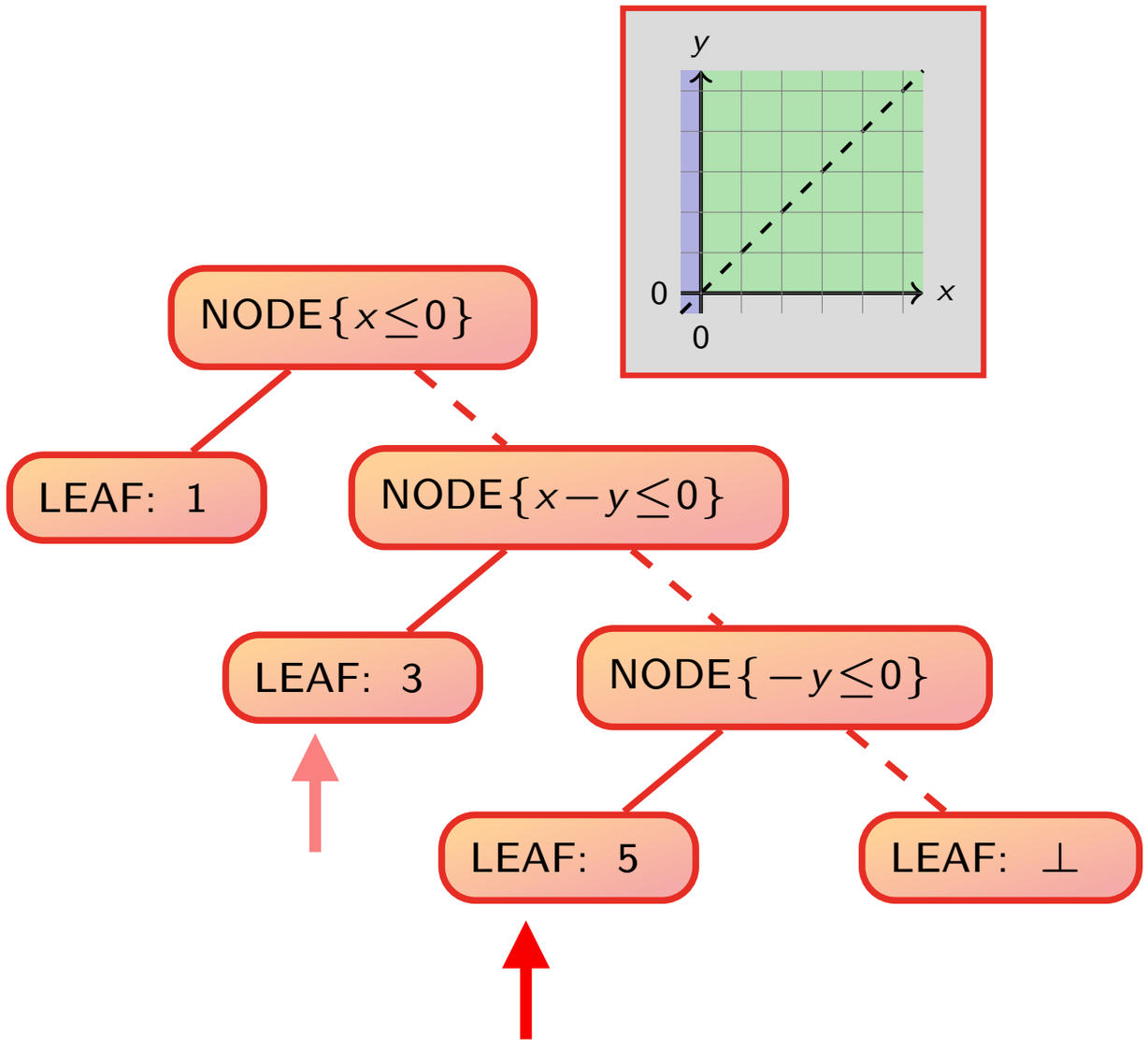


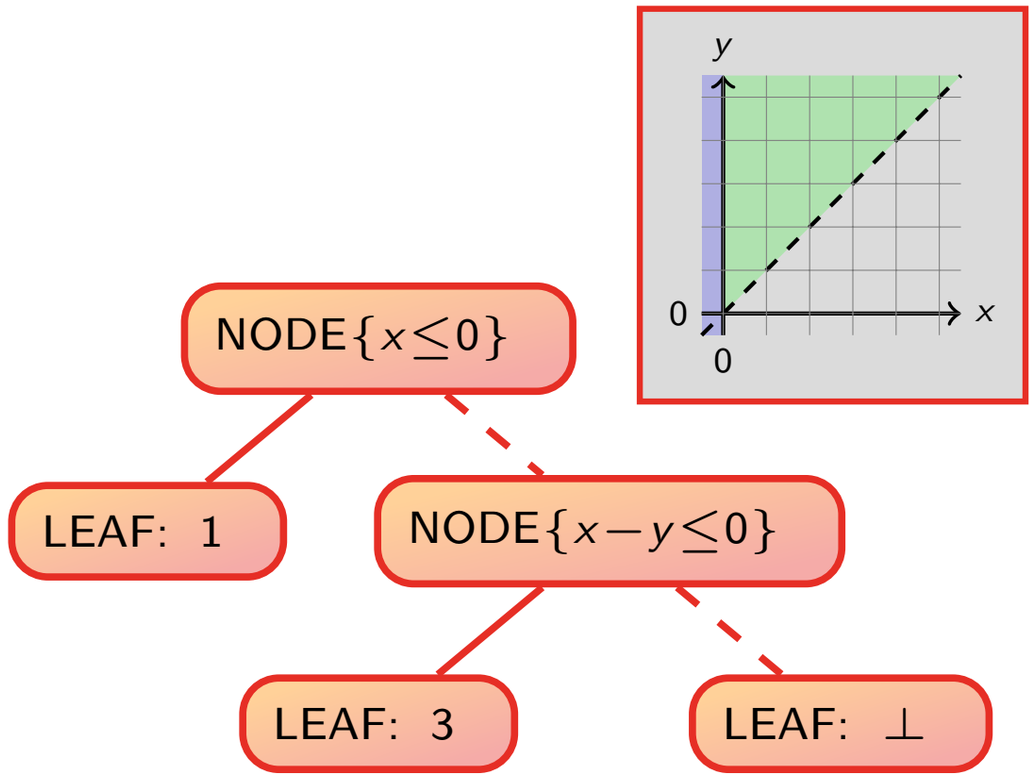
Fig. 2. The heuristics h_r improving on the standard widening.

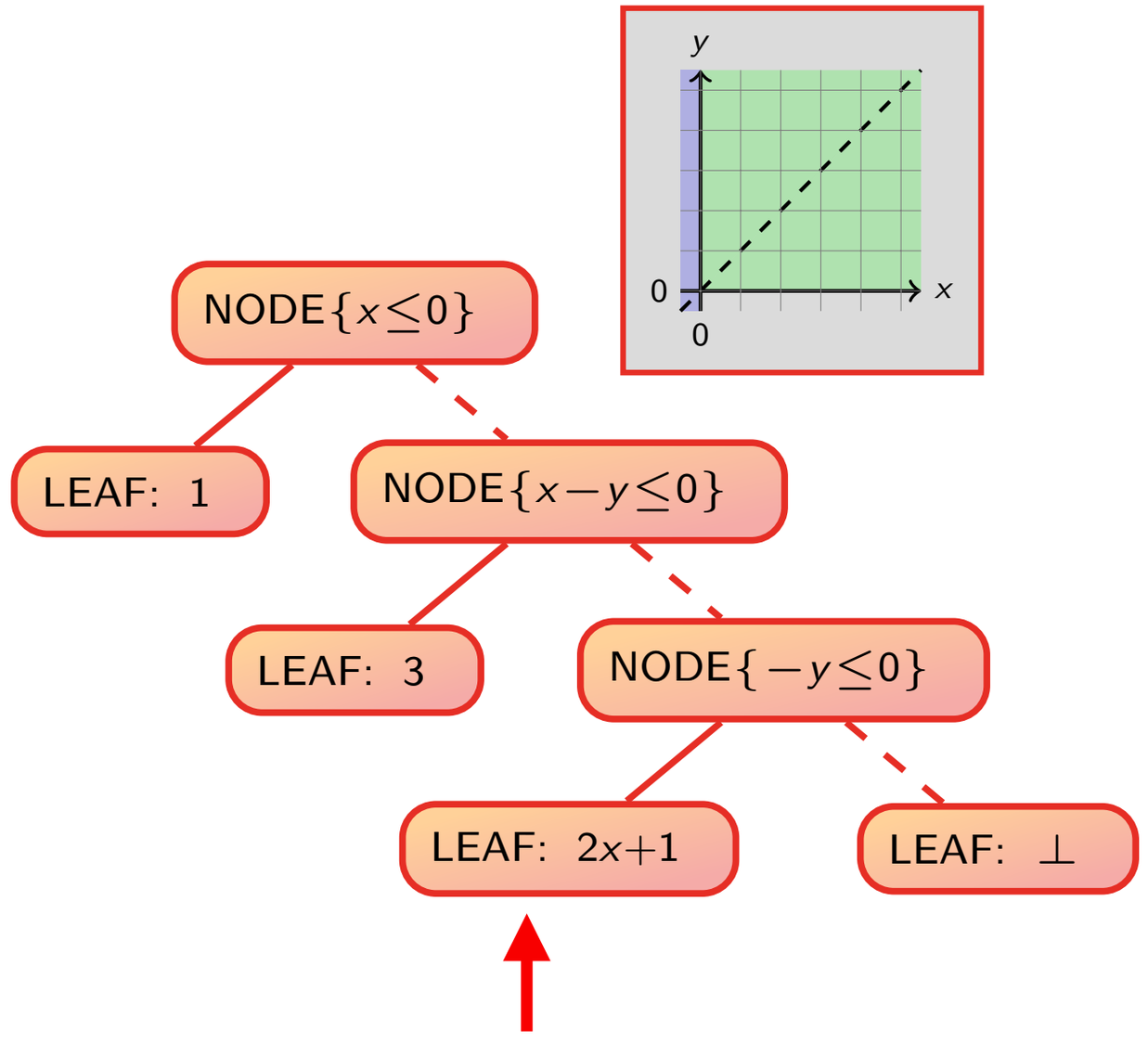
Better Widening



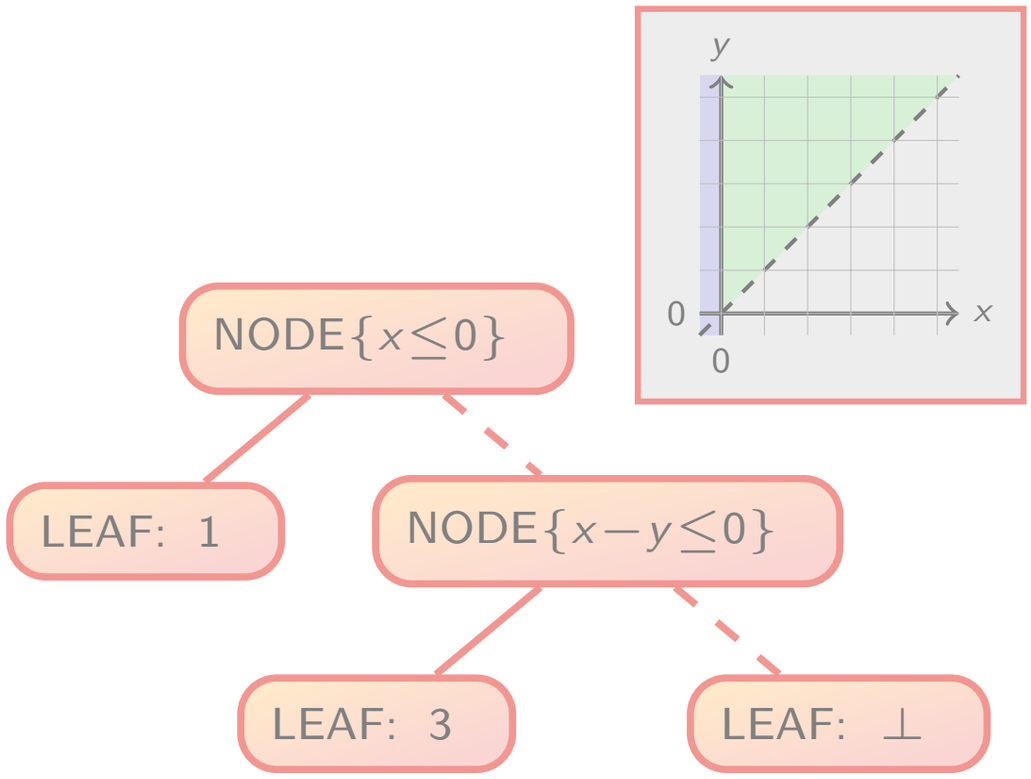


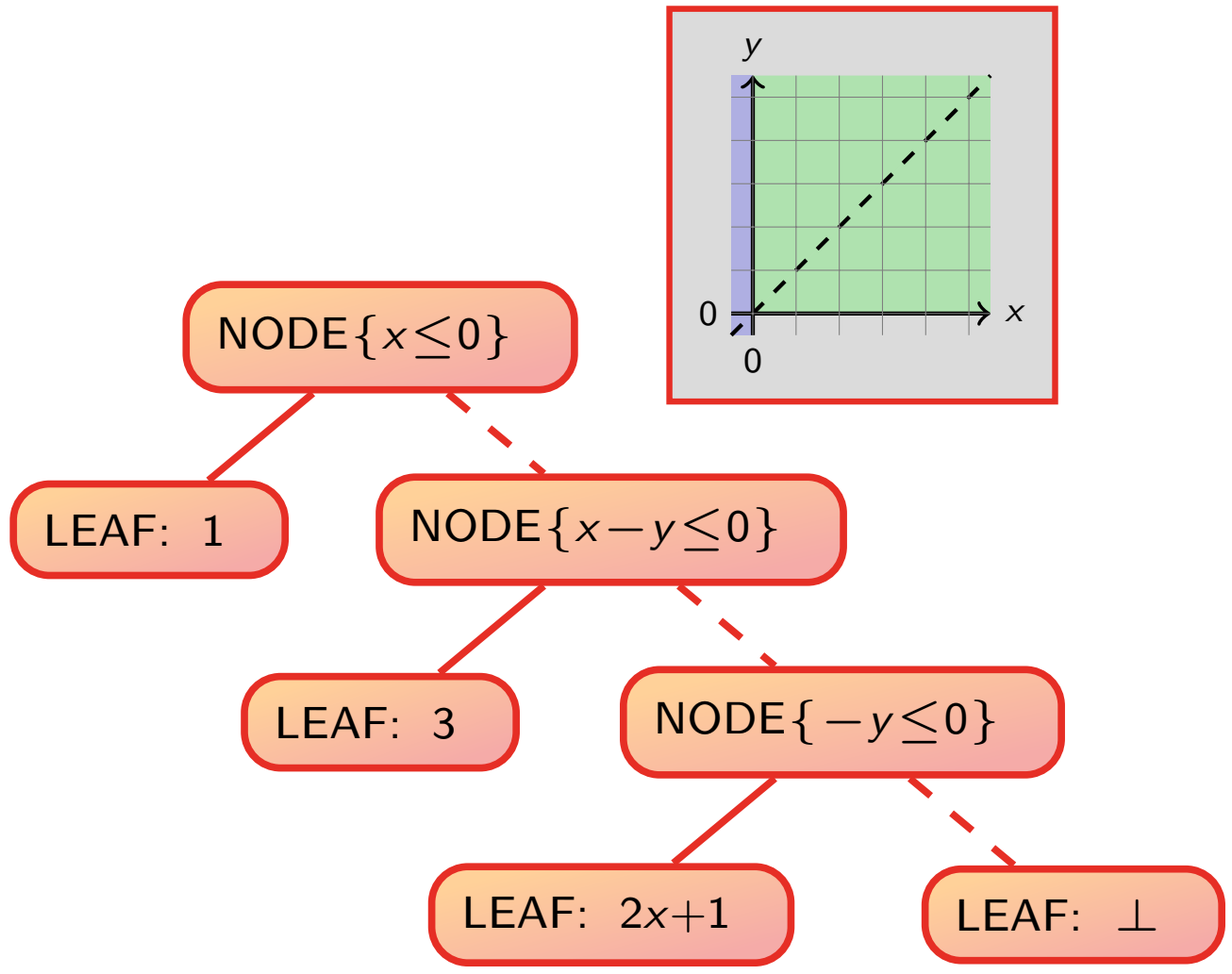
Better Widening





Better Widening





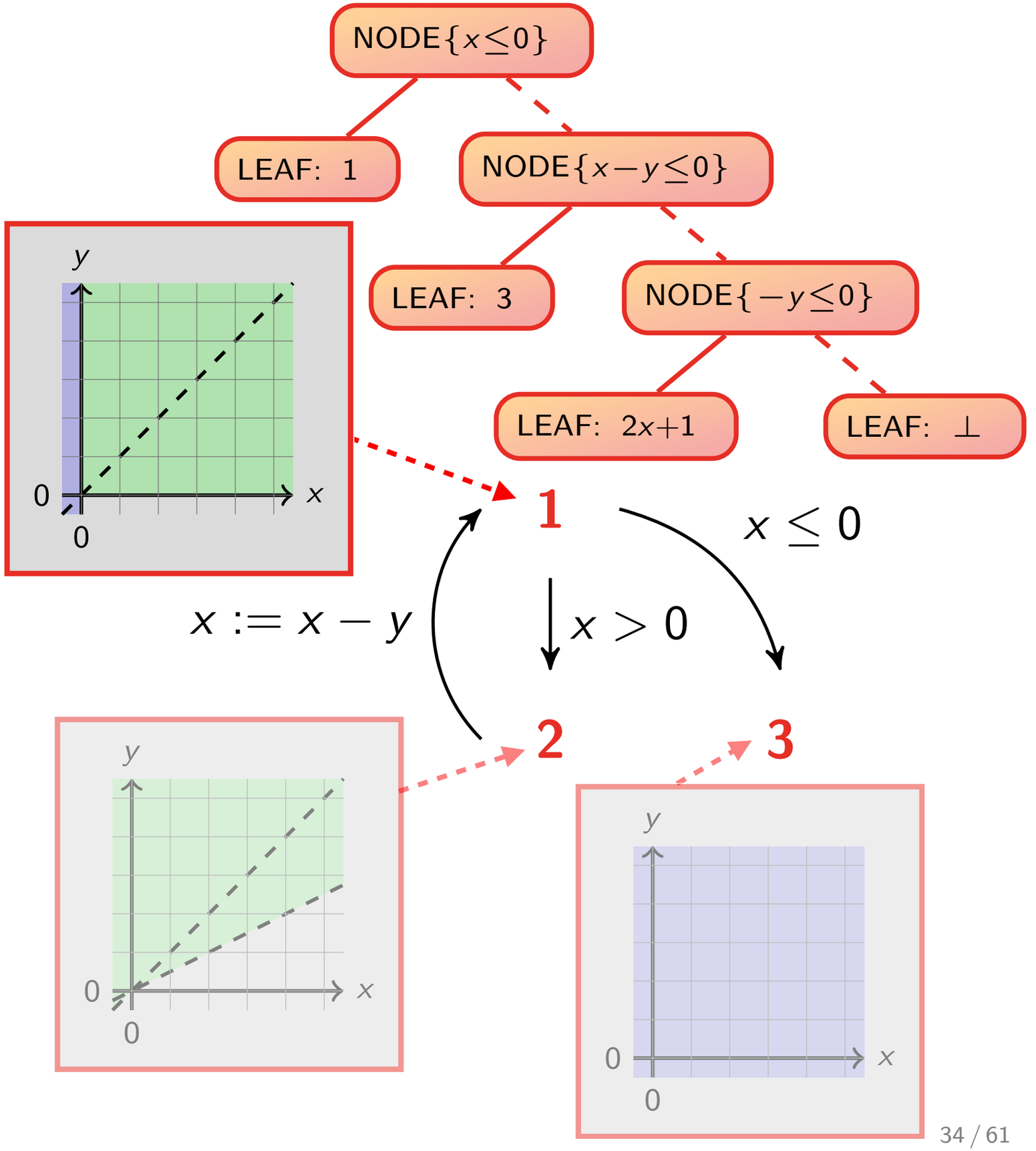
Example

int : $x,\ y$

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

  [2] $x := x - y$

od [3]



Example

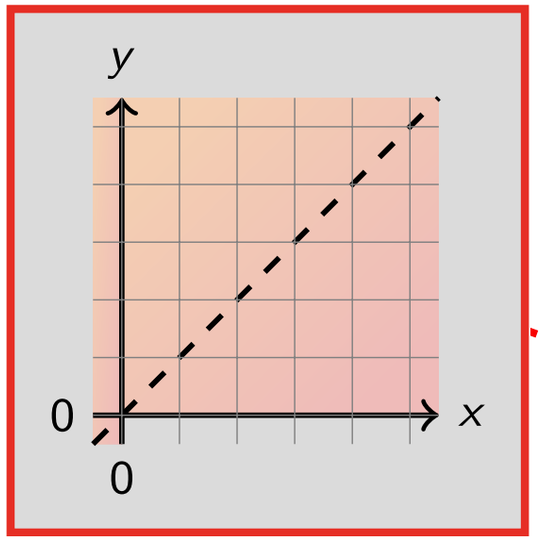
int : $x,\ y$

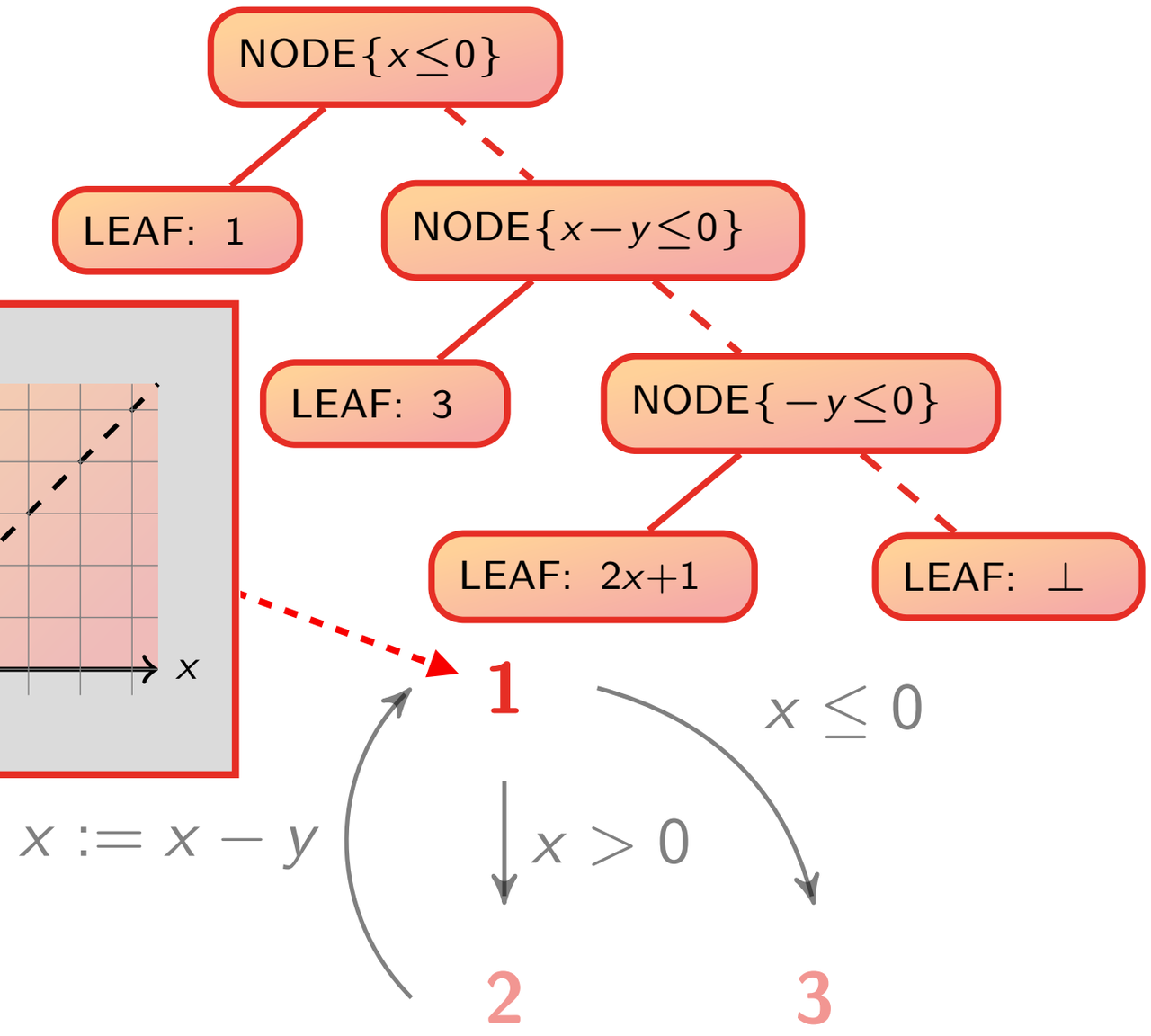
while [1] $(x > 0)$ do

  [2] $x := x - y$

od [3]

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





Example

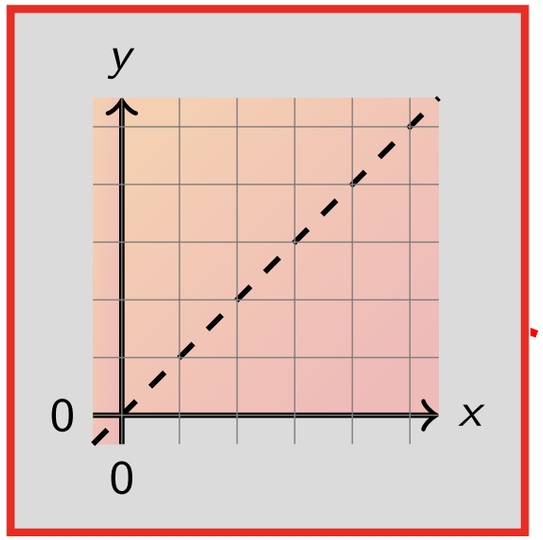
int : $x,\ y$

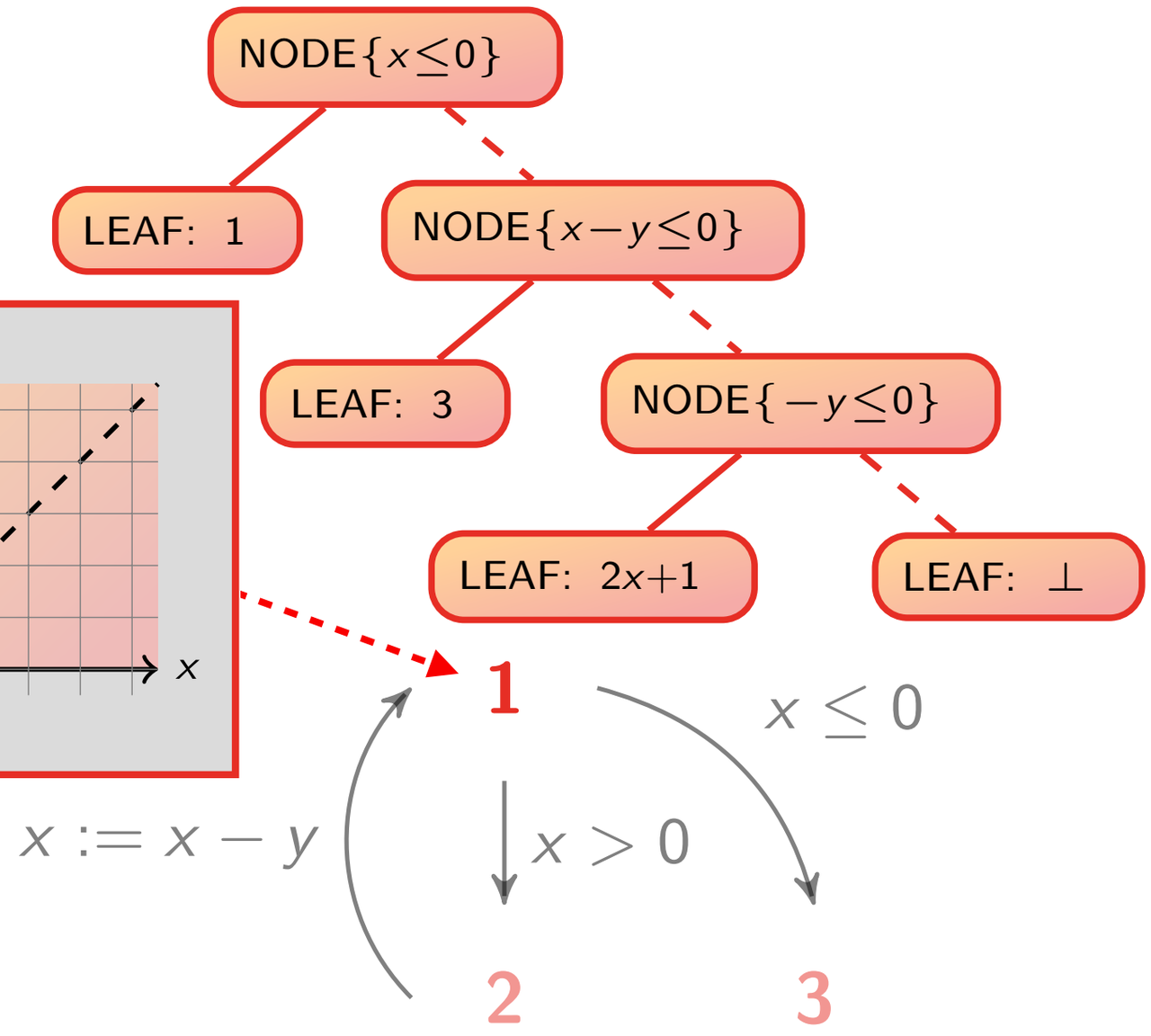
while [1] $(x > 0)$ do

  [2] $x := x - y$

od [3]

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





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

Example

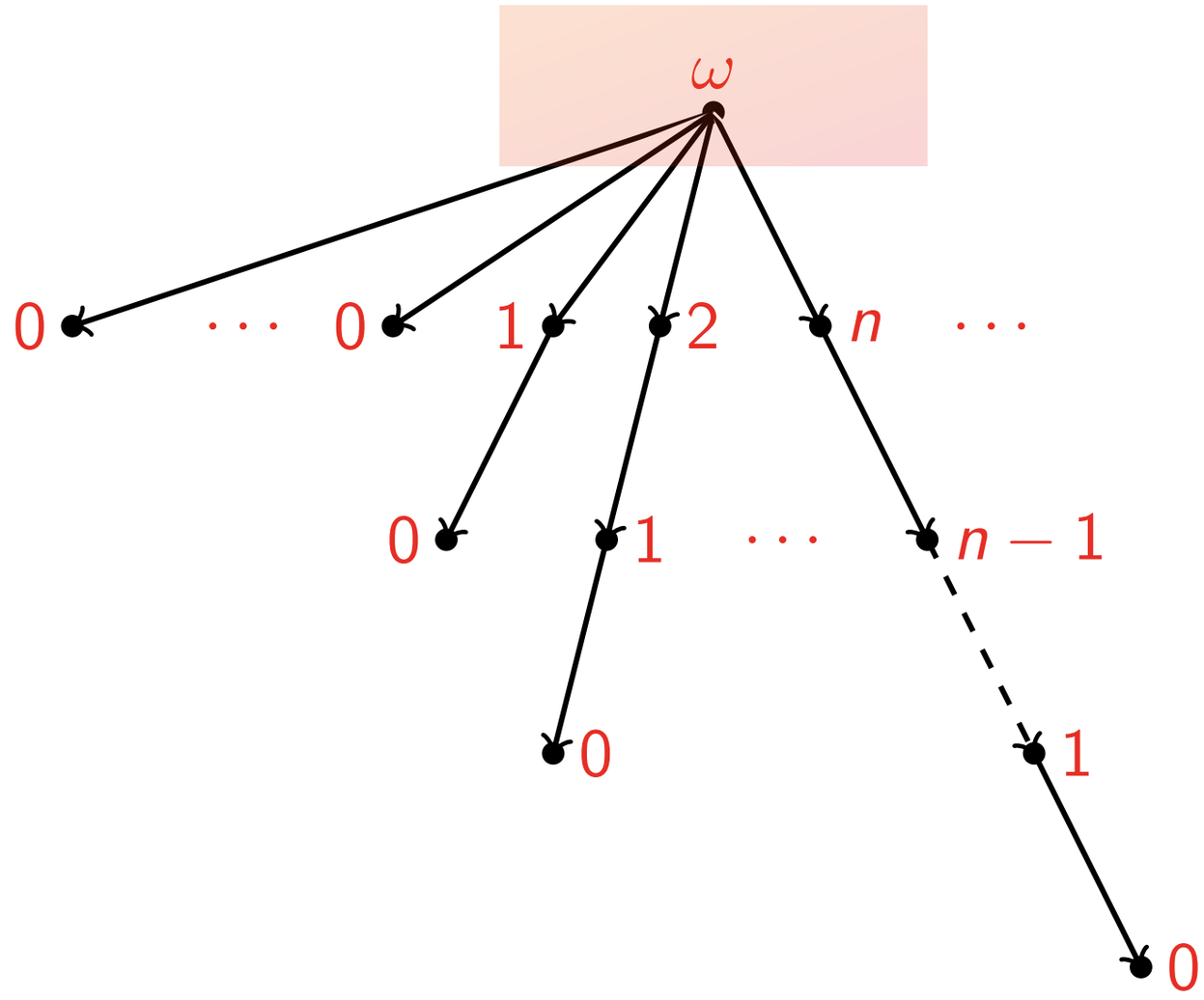
int : x

$x :=$?

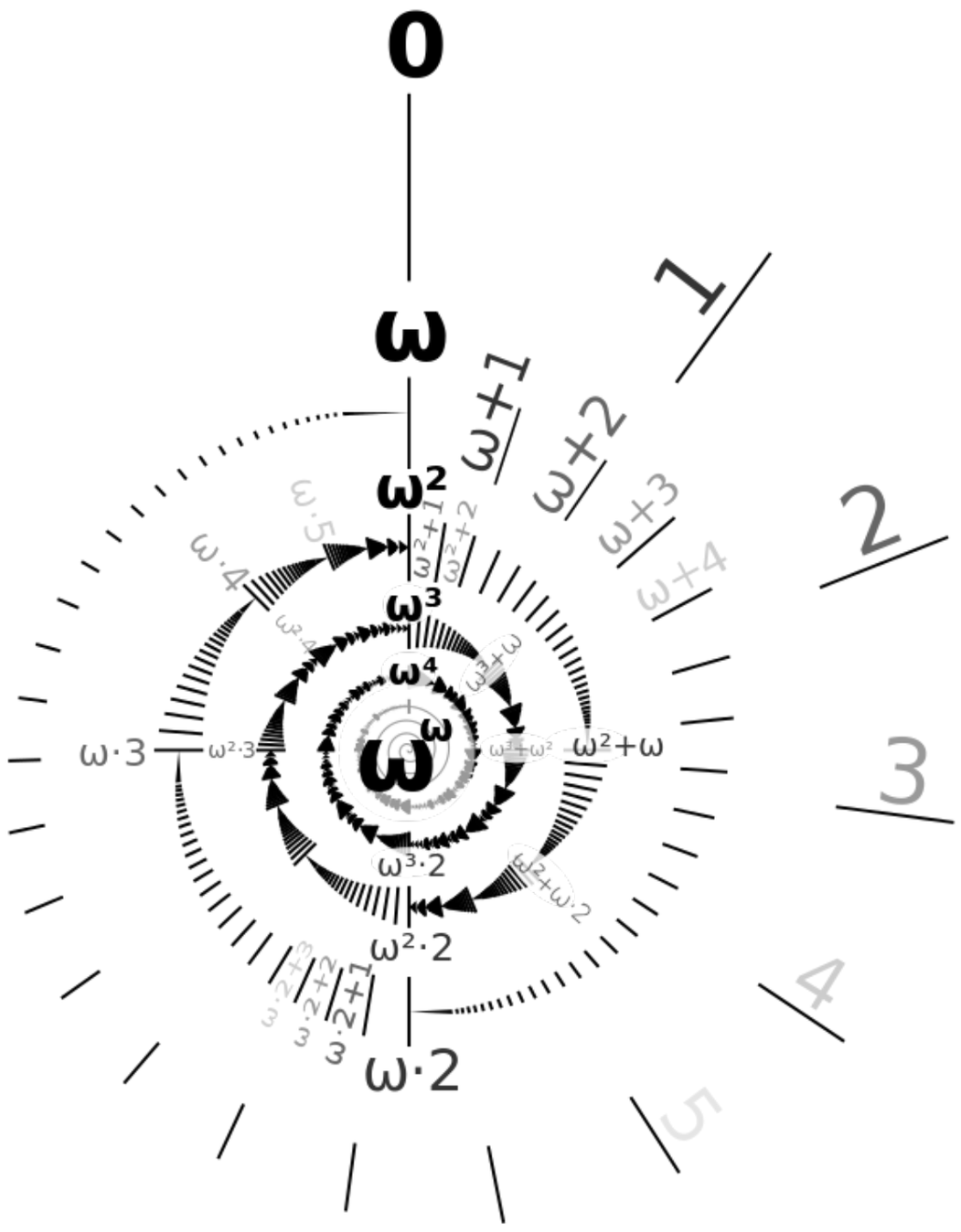
while $(x > 0)$ do

  $x := x - 1$

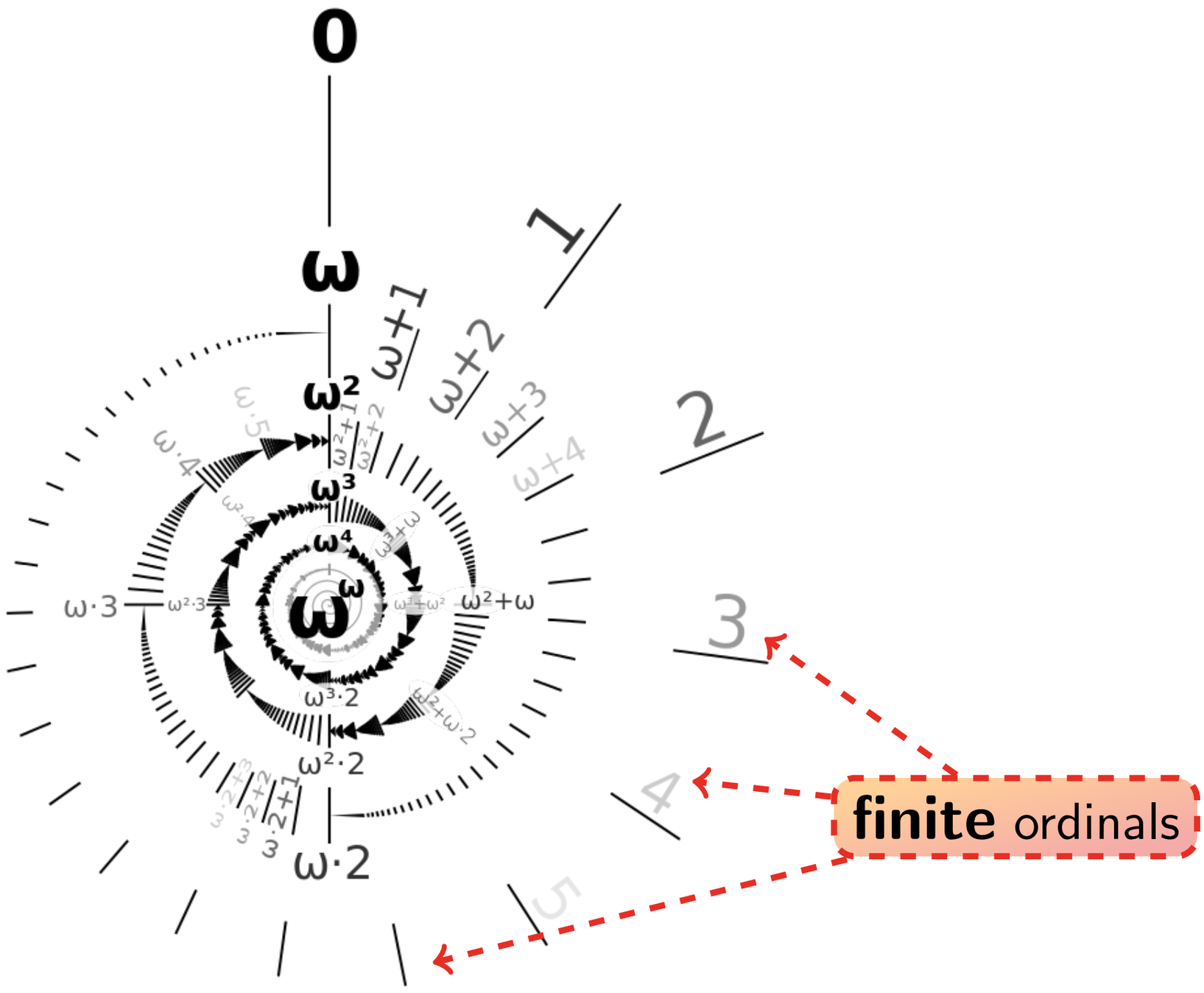
od



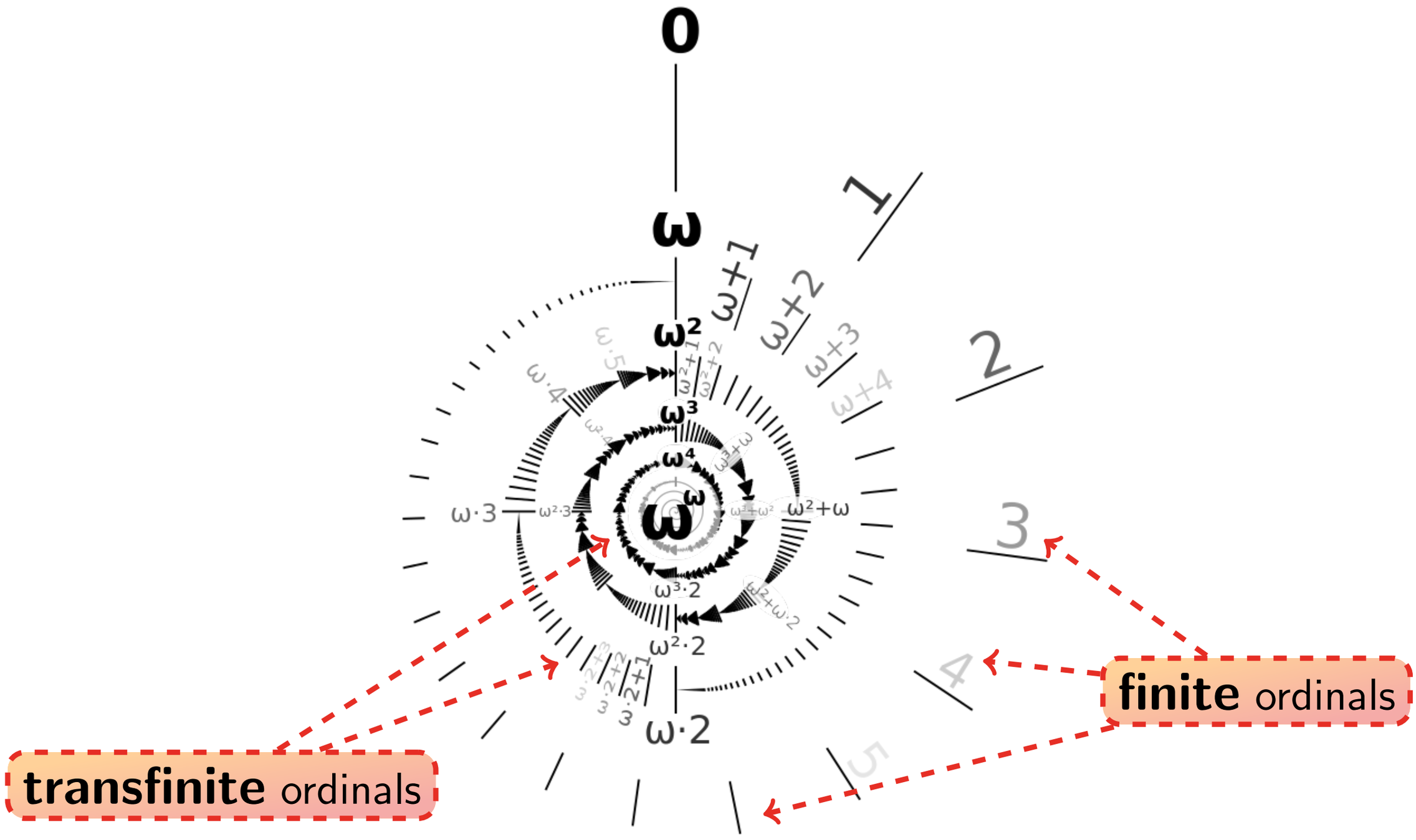
Ordinals


0
1
2
3
4
5
ω
ω+1
ω+2
ω+3
ω+4
ω·2
ω·3
ω·4
ω·5
ω²
ω²+1
ω²+2
ω²+ω
ω²·2
ω²·3
ω³
ω³+1
ω³+ω
ω³+ω²
ω³·2
ω⁴
ω^ω

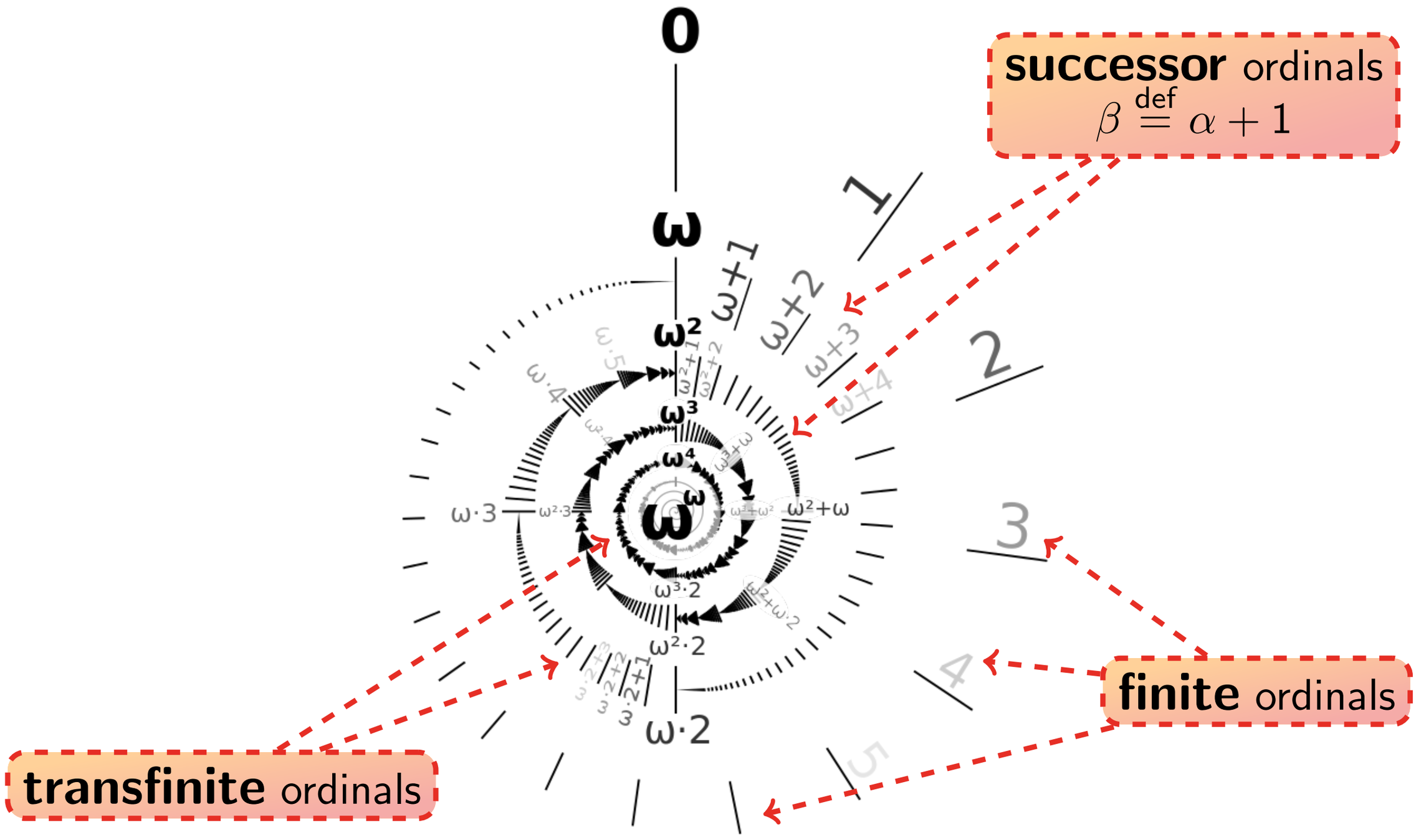
Ordinals


0
1
2
3
4
5
ω
ω+1
ω+2
ω+3
ω+4
ω·2
ω·3
ω·4
ω·5
ω²
ω²+1
ω²+2
ω²+3
ω²+ω
ω²·2
ω³
ω³+1
ω³+ω
ω³+ω²
ω³·2
ω⁴
ω^ω
finite ordinals

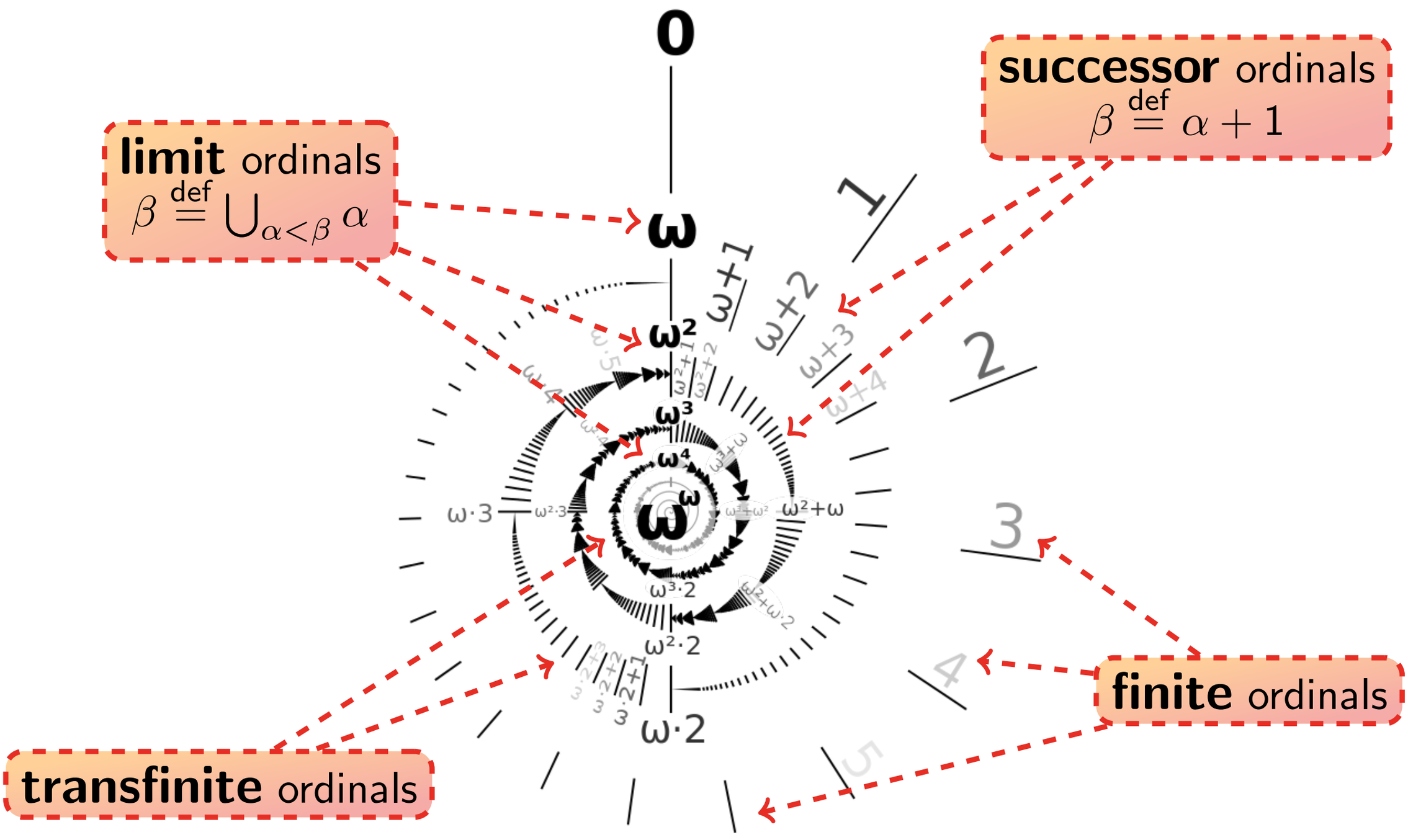
Ordinals

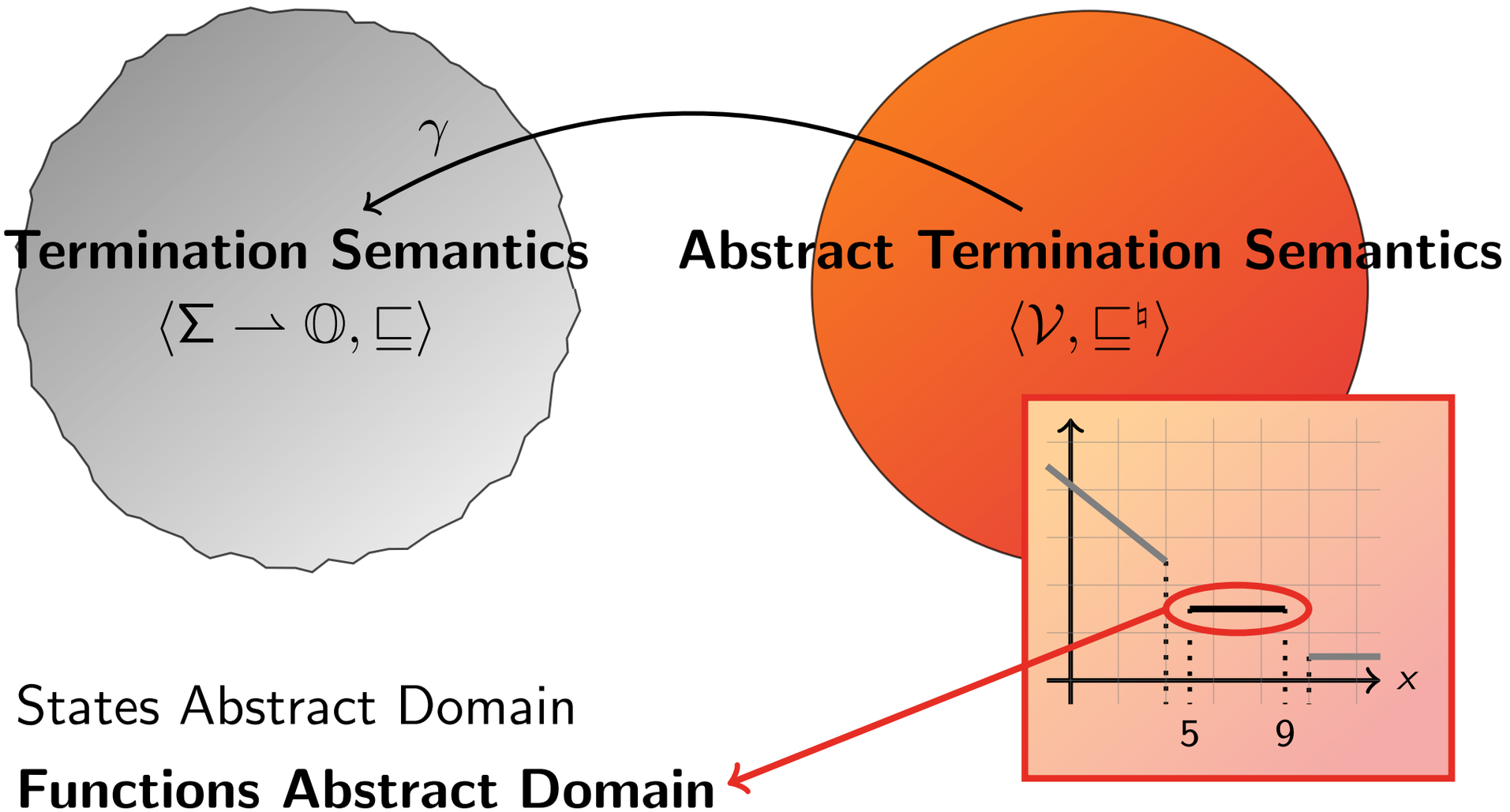

0
1
2
3
4
5
ω
ω+1
ω+2
ω+3
ω+4
ω·2
ω·3
ω·4
ω·5
ω²
ω²+1
ω²+2
ω²+3
ω²+4
ω²+ω
ω²·2
ω²·3
ω³
ω³+1
ω³+2
ω³+ω
ω³+ω²
ω³·2
ω⁴
ω^ω
transfinite ordinals
finite ordinals

Ordinals


0
1
2
3
4
5
ω
ω+1
ω+2
ω+3
ω+4
ω·2
ω·3
ω·4
ω·5
ω²
ω²+1
ω²+2
ω²+3
ω²+ω
ω²·2
ω³
ω³+1
ω³+2
ω³+ω
ω³+ω²
ω³·2
ω⁴
ω^ω
successor ordinals
$\beta \stackrel{\text{def}}{=} \alpha + 1$
finite ordinals
transfinite ordinals

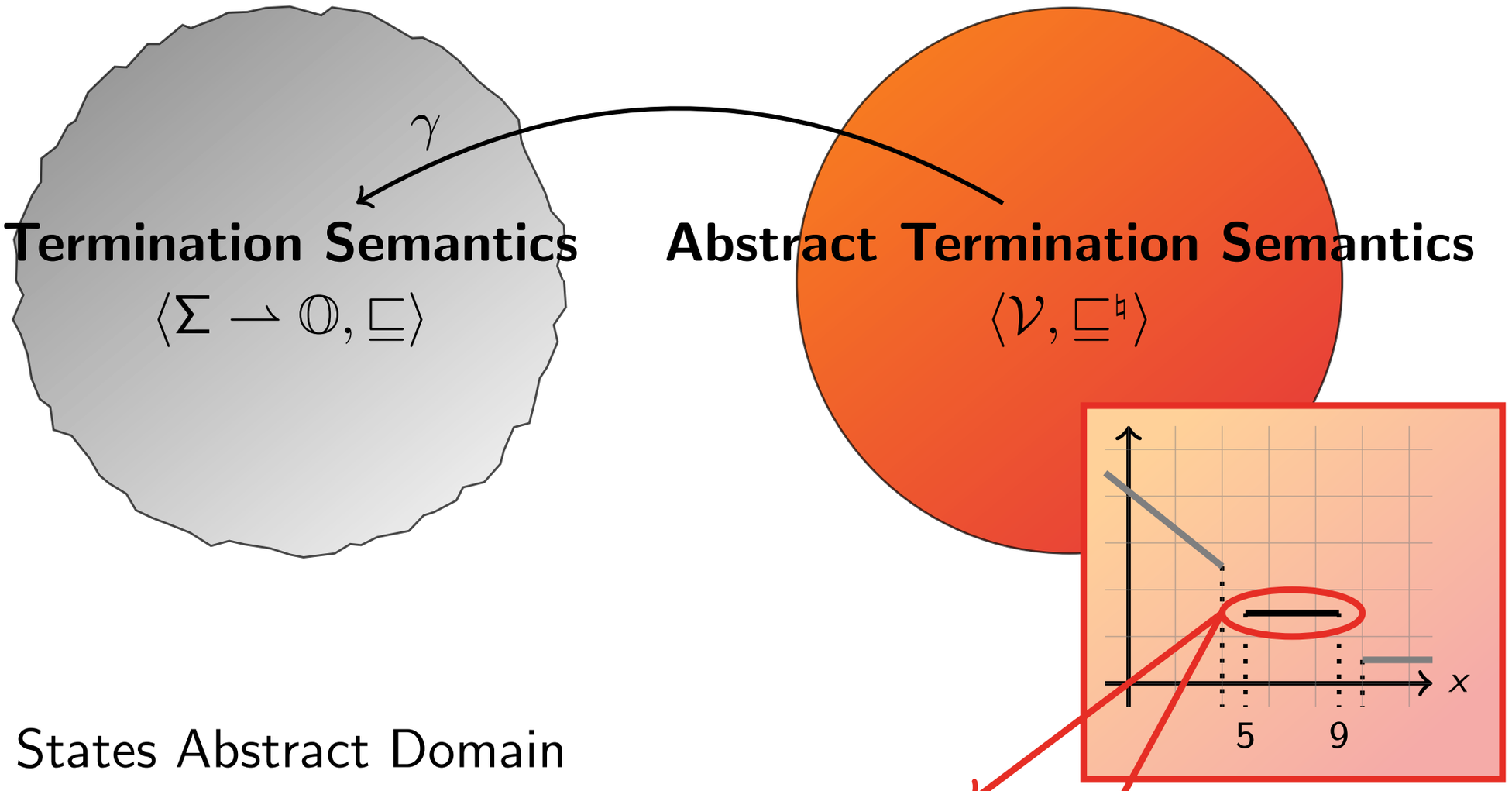
Ordinals


0
ω
ω+1
ω+2
ω+3
ω+4
1
2
3
4
5
ω·2
ω·3
ω²
ω²+1
ω²+2
ω²+3
ω²+4
ω²+ω
ω²·2
ω²·3
ω³
ω³+1
ω³+2
ω³+ω
ω³+ω²
ω³·2
ω⁴
ω^ω
limit ordinals
$\beta \stackrel{\text{def}}{=} \bigcup_{\alpha<\beta} \alpha$
successor ordinals
$\beta \stackrel{\text{def}}{=} \alpha + 1$
finite ordinals
transfinite ordinals



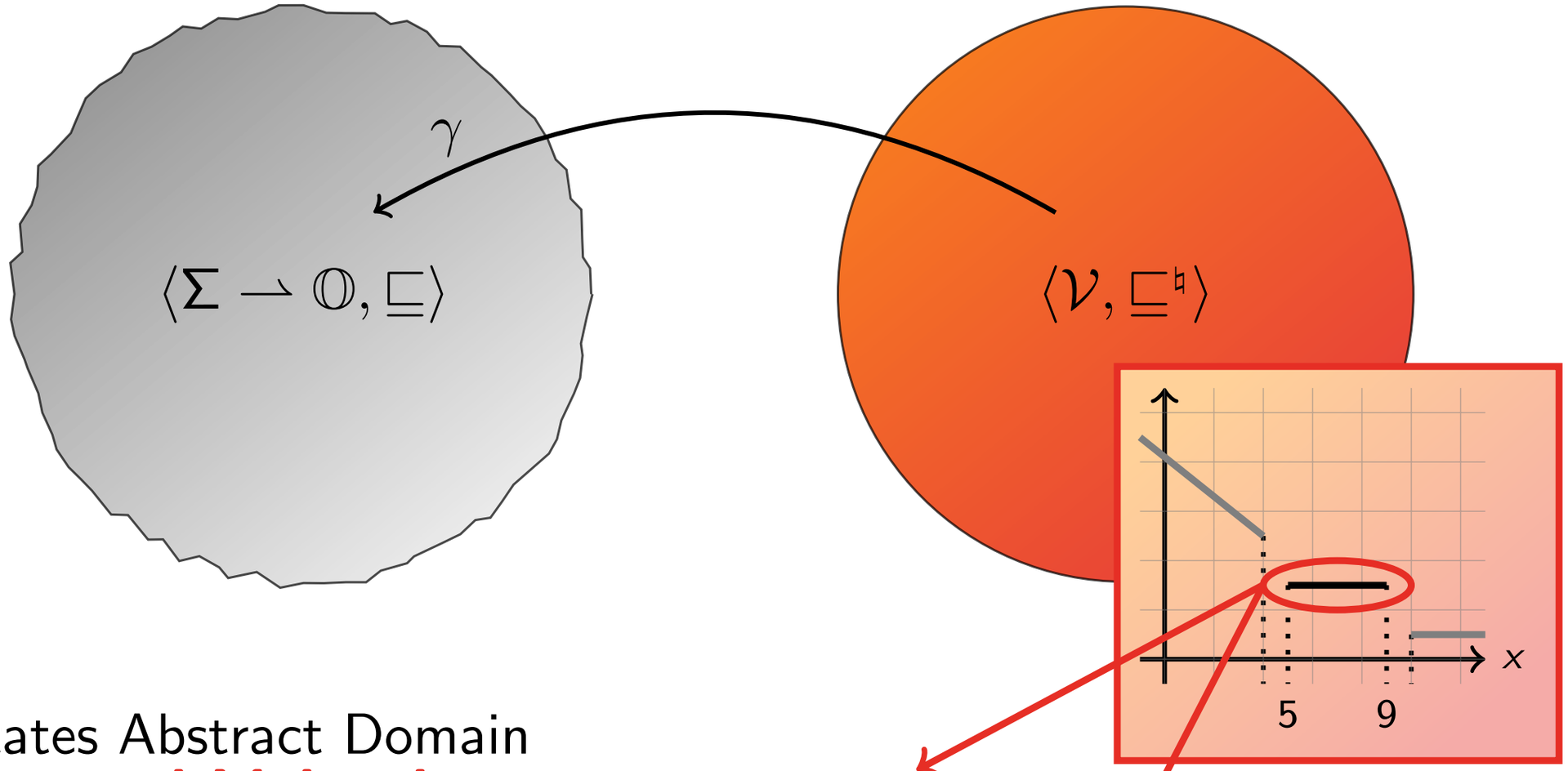
- States Abstract Domain S
- **Functions Abstract Domain** F
- Piecewise-Defined Ranking Functions Abstract Domain V(S, F)

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



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

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

$$\Downarrow \quad x_1 := ?$$

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

Backward Non-Deterministic Assignments

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

Example

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

$$\Downarrow \quad x_1 := ?$$

$$[-\infty, +\infty] \mapsto \quad + \quad 1$$

Backward Non-Deterministic Assignments

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

Example

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

$$\Downarrow \quad x_1 := ?$$

$$[-\infty, +\infty] \mapsto + x_2 + 1$$

Backward Non-Deterministic Assignments

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

Example

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

$$\Downarrow \quad x_1 := ?$$

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

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

Backward Non-Deterministic Assignments

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

Example

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

$$\Downarrow \quad x_1 := ?$$

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

Example

int : x_1, x_2

while [1] $(x_1 > 0 \wedge x_2 > 0)$ do

  if [2] (?) then

    [3] $x_1 := x_1 - 1$

    [4] $x_2 :=$?

  else

    [5] $x_2 := x_2 - 1$

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**

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

Guarantee and Recurrence Properties

Proving Guarantee and Recurrence
Temporal Properties by Abstract Interpretation*

Caterina Urban and Antoine Miné

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

Abstract. We present new static analysis methods for proving liveness properties of programs. In particular, with reference to the hierarchy of temporal properties proposed by Manna and Pnueli, we focus on guarantee (i.e., "something good occurs *at least once*") and recurrence (i.e., "something good occurs *infinitely often*") temporal properties.
We generalize the abstract interpretation framework for termination presented by Cousot and Cousot. Specifically, static analyses of guarantee and recurrence temporal properties are systematically derived by abstraction of the program operational trace semantics.
These methods automatically infer *sufficient* preconditions for the temporal properties by *creating* existing numerical abstract domains based on piecewise-defined ranking functions. We augment these abstract domains with new abstract operators, including a *dual* widening.
To illustrate the potential of the proposed methods, we have implemented a research prototype static analyzer, for programs written in a C-like syntax, that yielded interesting preliminary results.

1 Introduction

Temporal properties play a major role in the specification and verification of programs. The hierarchy of temporal properties proposed by Manna and Pnueli [17] distinguishes four basic classes:

- *safety* properties: "something good *always* happens", i.e., the program never reaches an unacceptable state (e.g., partial correctness, mutual exclusion);
- *guarantee* properties: "something good happens *at least once*", i.e., the program *eventually* reaches a desirable state (e.g., termination);
- **recurrence properties: "something good happens *infinitely often*"**, i.e., the program reaches a desirable state *infinitely often* (e.g., starvation freedom);
- *persistence* properties: "something good *eventually* happens *continuously*".

This paper concerns the verification of programs by static analysis. We set our work in the framework of Abstract Interpretation [8], a general theory of semantic

* The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement no. 269335 (ARTEMIS project MBAT) (see Article II.9 of the JU Grant Agreement).

Example

int : x, y

while [1] $(x \geq 0)$ do

  [2] $x := x + 1$

od

while [3] (true) do

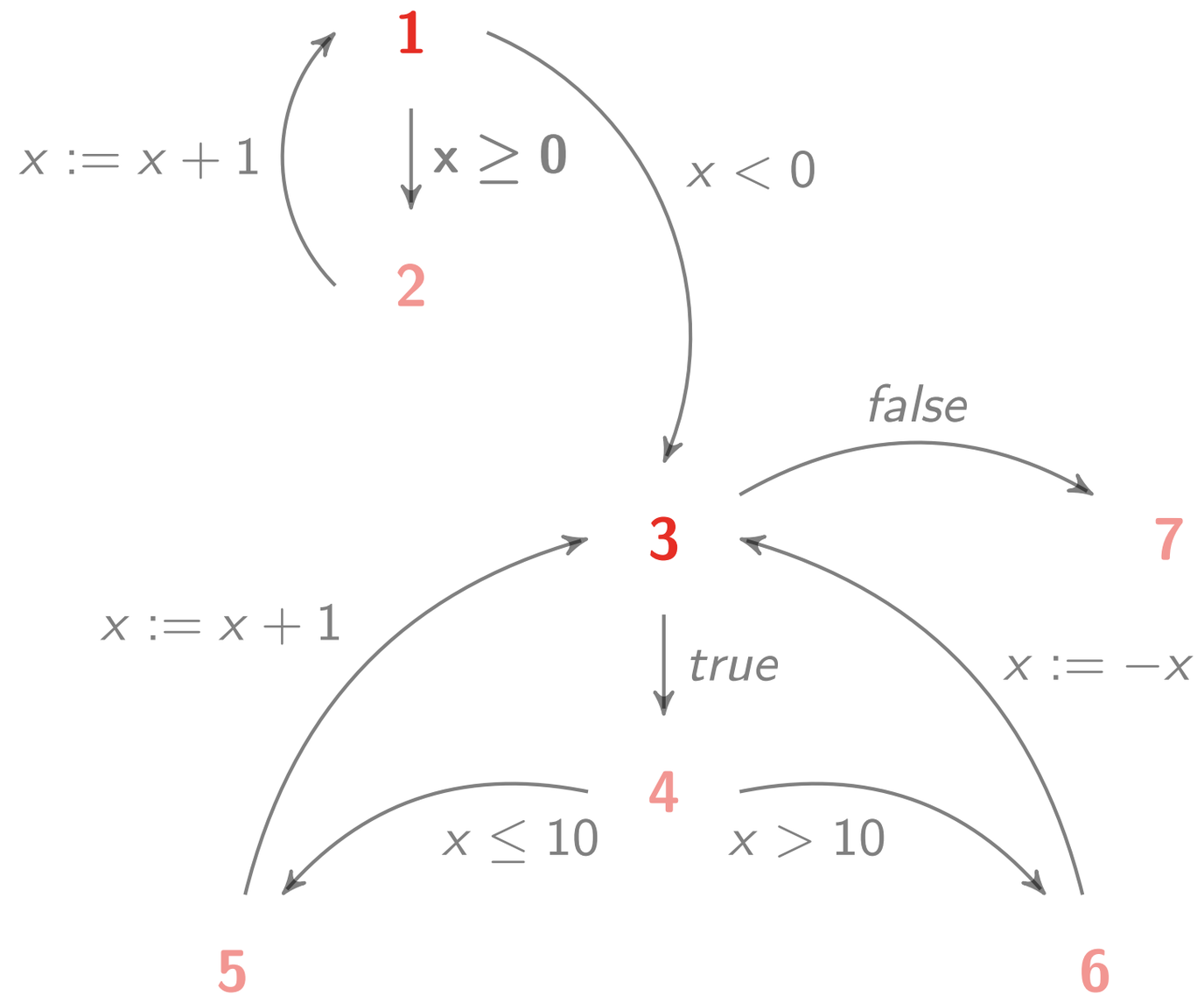
  if [4] ($x \leq 10$)

    [5] $x := x + 1$

  else

    [6] $x := -x$

od[7]



Example

int : x, y

while [1]$(x \geq 0)$ do

  [2]$x := x + 1$

od

while [3](true) do

  if [4]($x \leq 10$)

    [5]$x := x + 1$

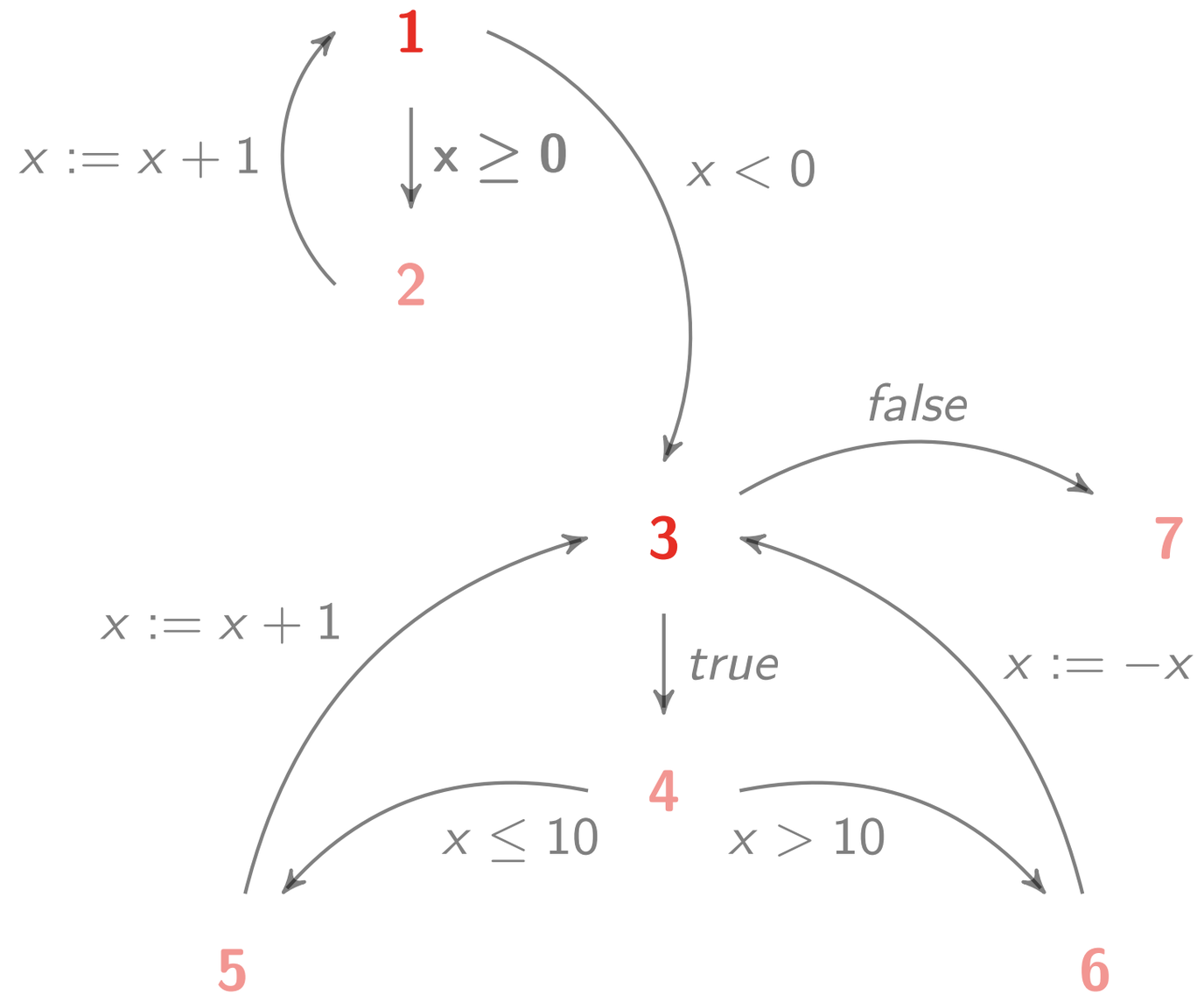
  else

    [6]$x := -x$

od[7]

Property

$\Diamond\ x = 3$



Example

int : x, y

while [1]$(x \geq 0)$ do

  [2]$x := x + 1$

od

while [3](true) do

  if [4]($x \leq 10$)

    [5]$x := x + 1$

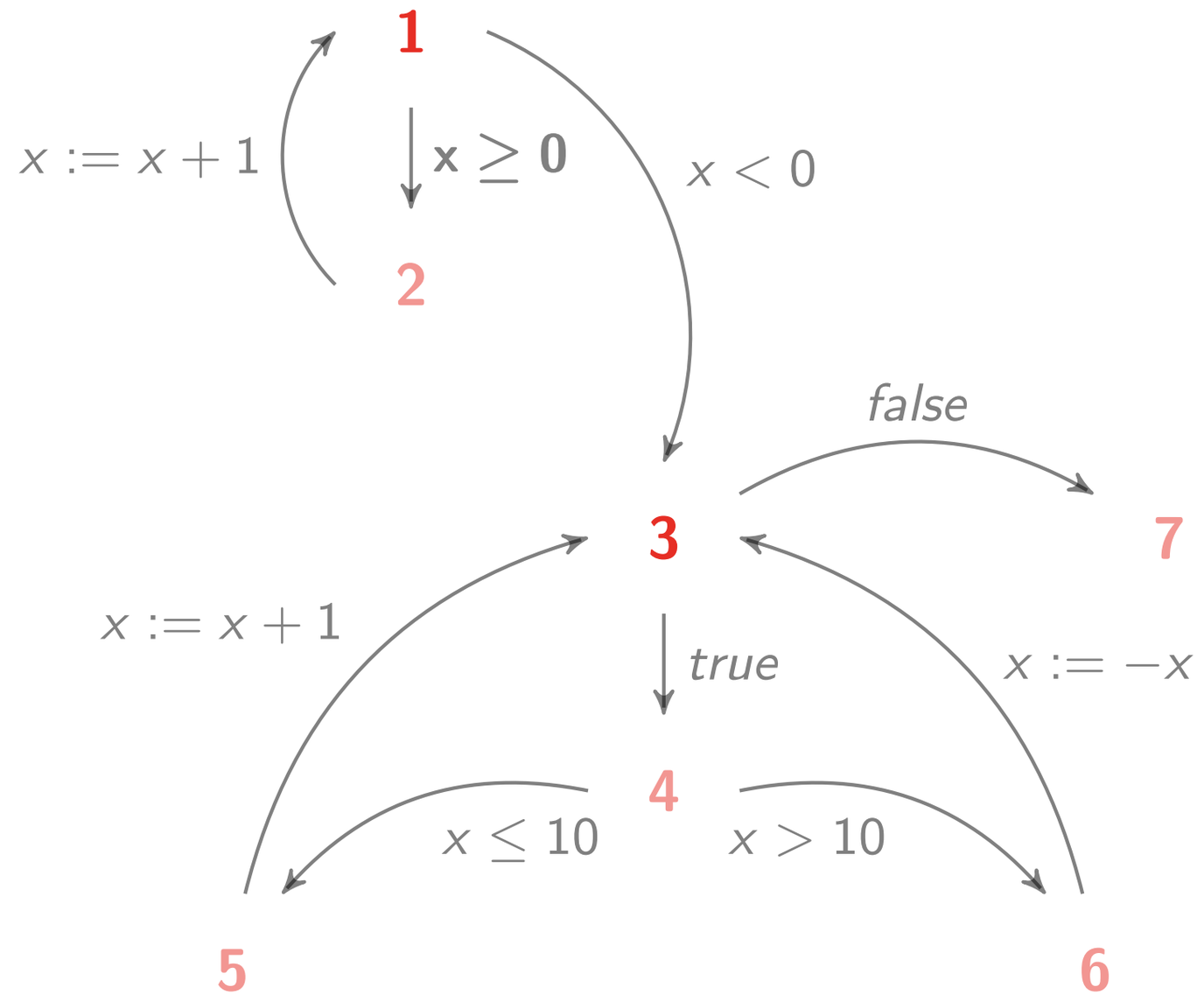
  else

    [6]$x := -x$

od[7]

Property

$\Diamond\ x = 3$



Example

int : x, y

while [1] $(x \geq 0)$ do

  [2] $x := x + 1$

od

while [3] (true) do

  if [4] ($x \leq 10$)

    [5] $x := x + 1$

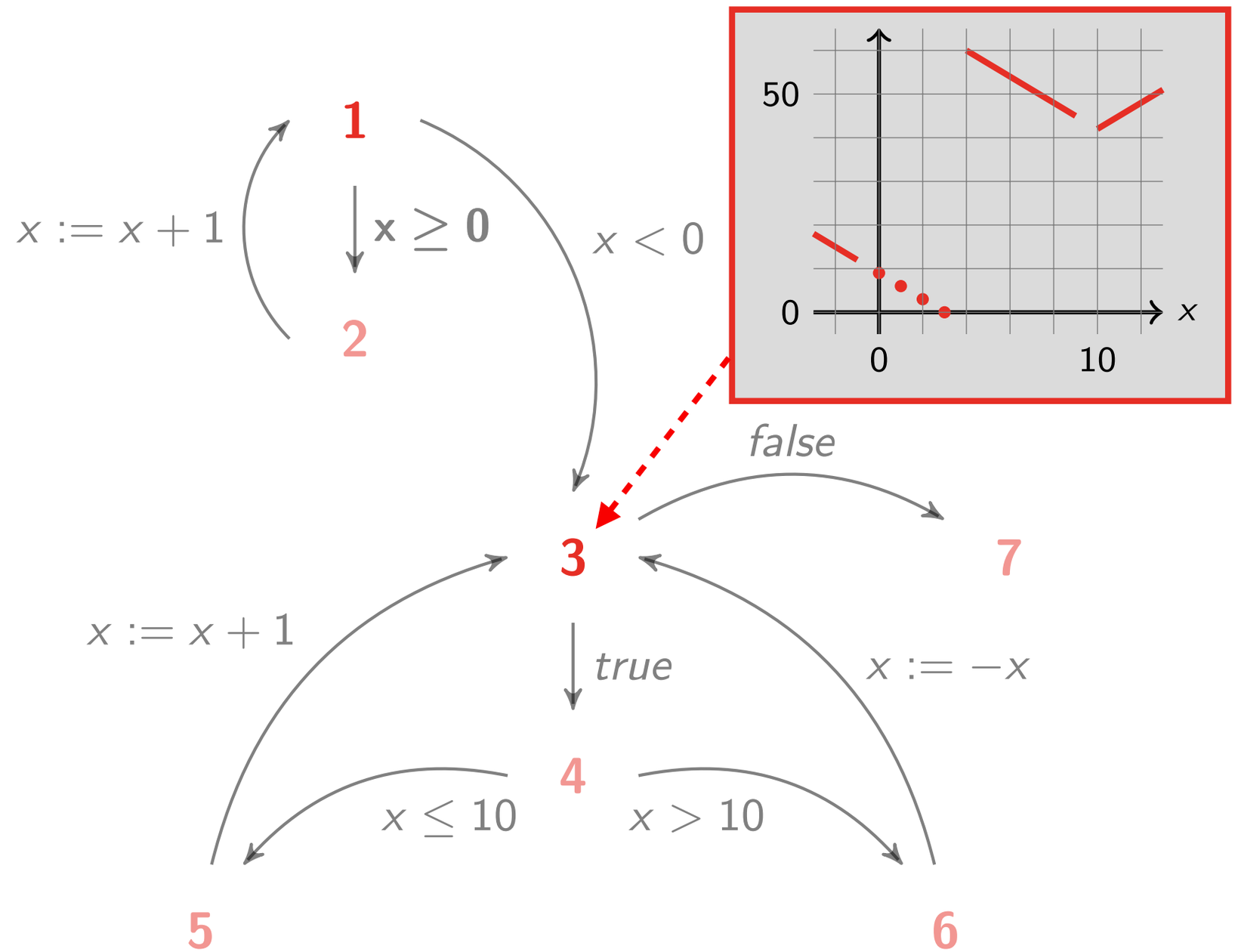
  else

    [6] $x := -x$

od [7]

Property

$\Diamond\ x = 3$



Example

int : x, y

while [1] $(x \geq 0)$

  [2] $x := x + 1$

od

while [3] (true) do

  if [4] ($x \leq 10$)

    [5] $x := x + 1$

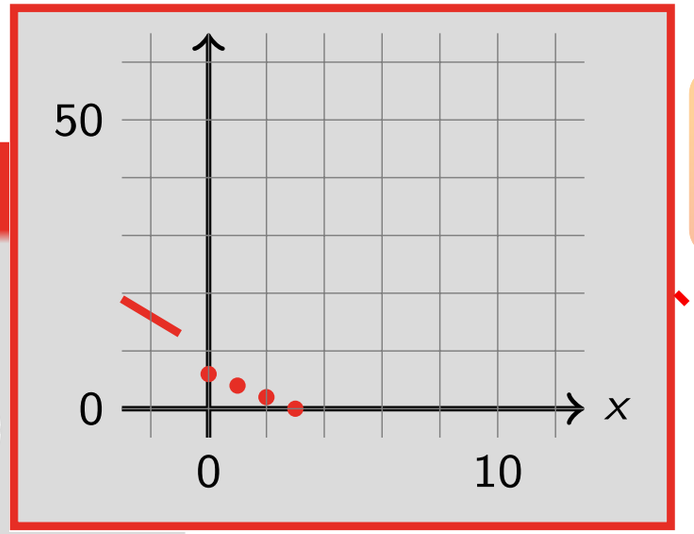
  else

    [6] $x := -x$

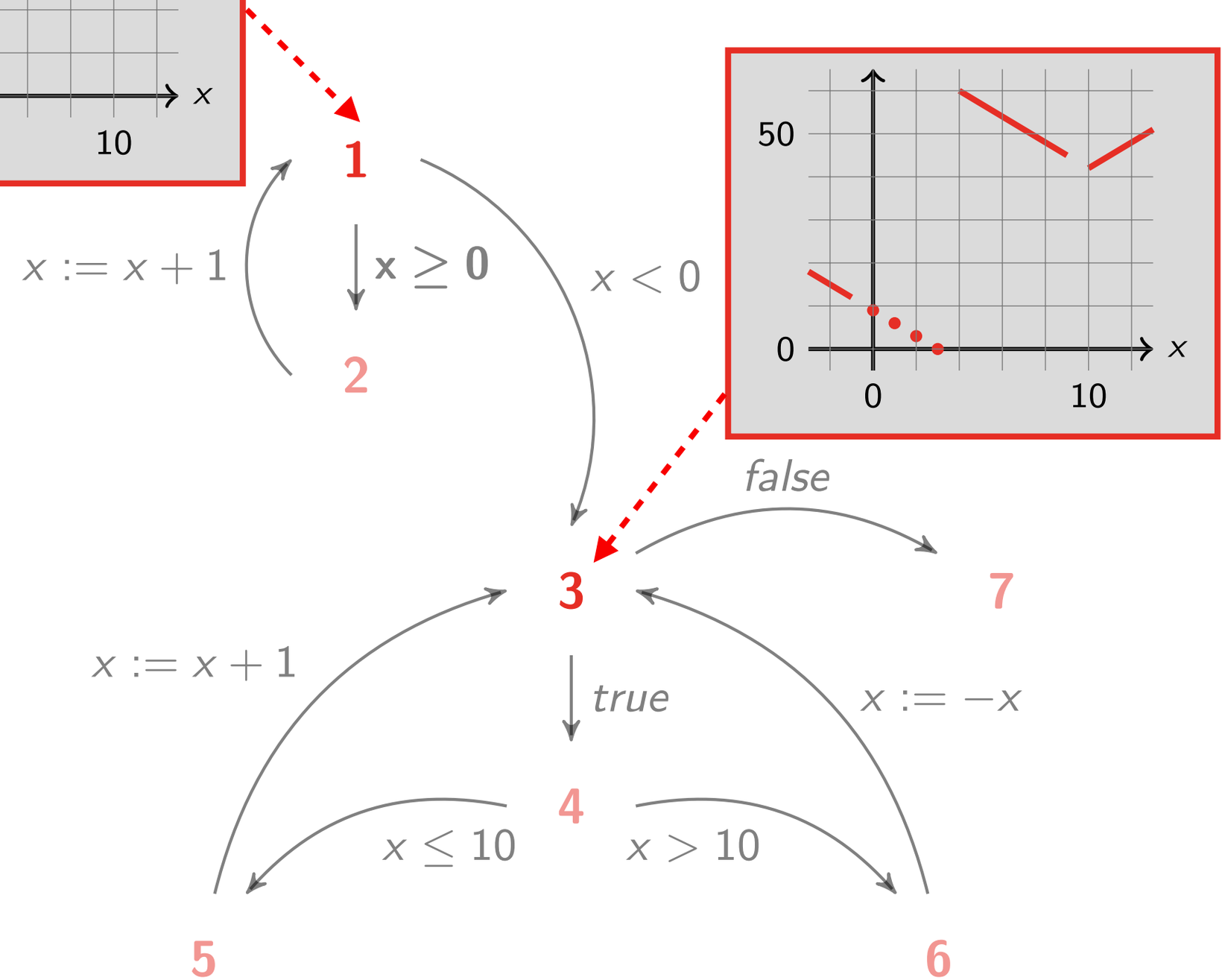
od [7]

Property

$\Diamond\ x = 3$



the analysis gives $x \leq 3$ as **sufficient precondition**



Example

int : x, y

while [1] $(x \geq 0)$ do

  [2] $x := x + 1$

od

while [3] (true) do

  if [4] ($x \leq 10$)

    [5] $x := x + 1$

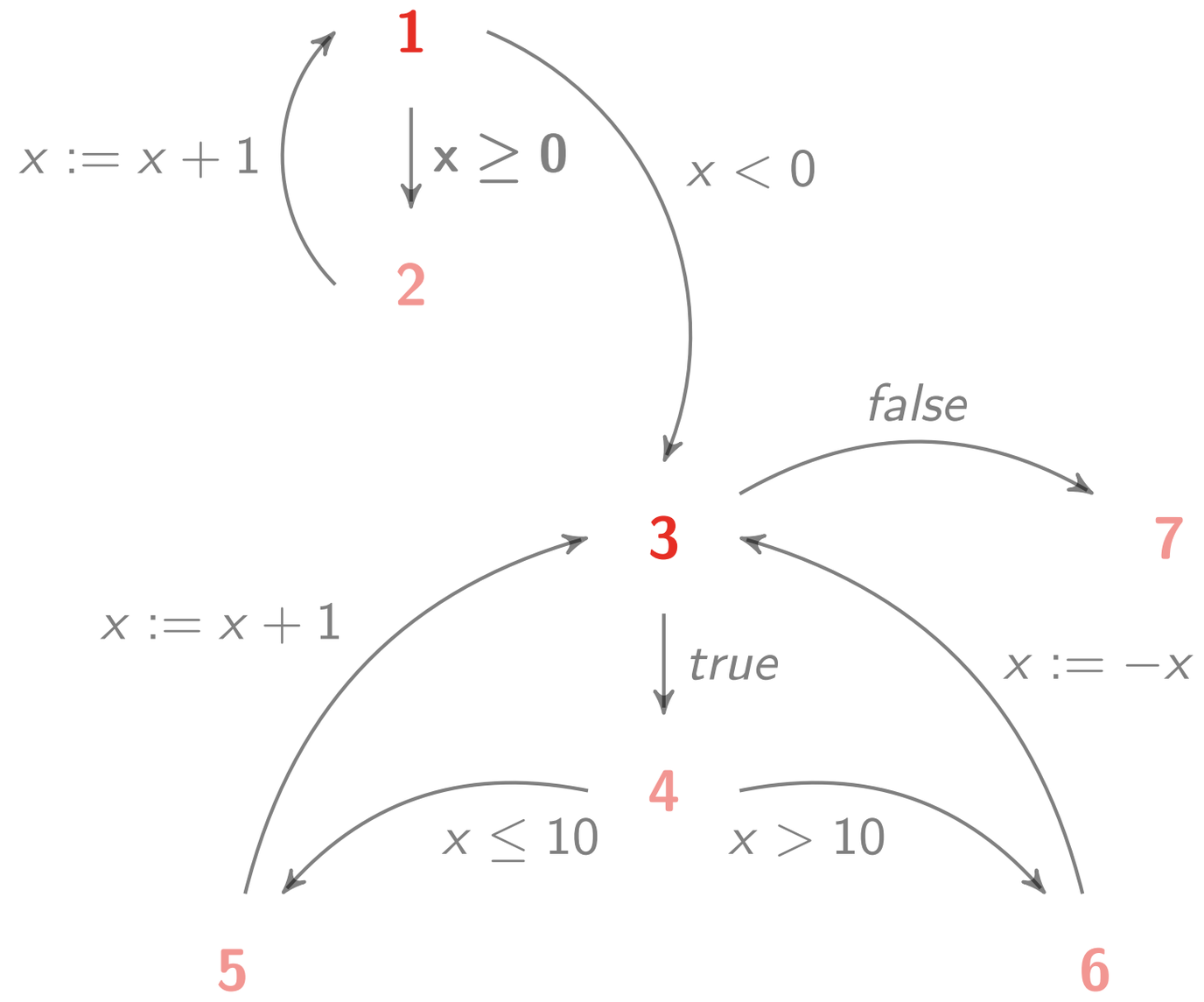
  else

    [6] $x := -x$

od [7]

Property

$\Box\Diamond\ x = 3$



Example

int : x, y

while [1] $(x \geq 0)$ do

  [2] $x := x + 1$

od

while [3] (true) do

  if [4] ($x \leq 10$)

    [5] $x := x + 1$

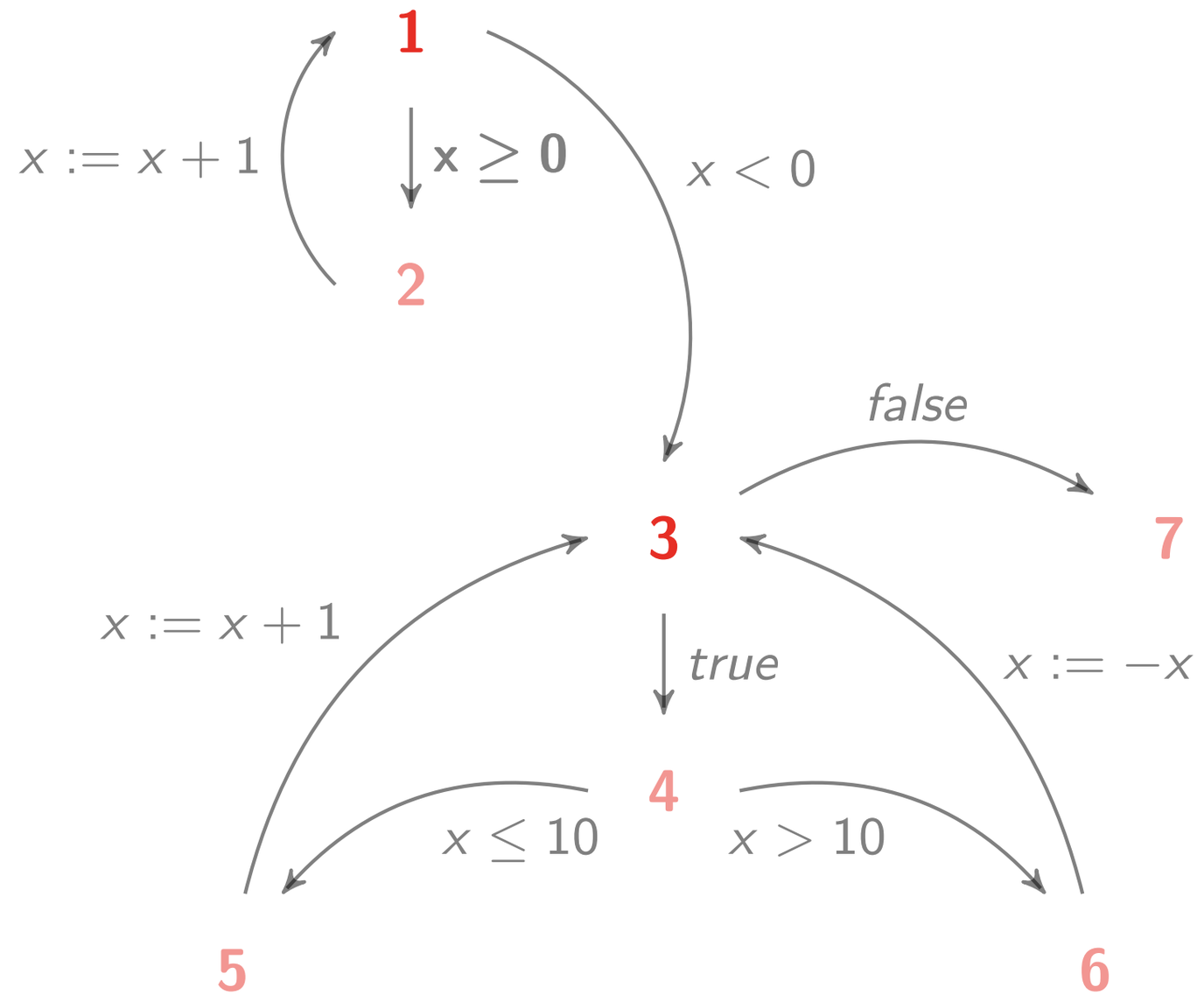
  else

    [6] $x := -x$

od [7]

Property

$\Box\Diamond\ x = 3$



Example

int : x, y

while [1] $(x \geq 0)$ do

  [2] $x := x + 1$

od

while [3] (true) do

  if [4] ($x \leq 10$)

    [5] $x := x + 1$

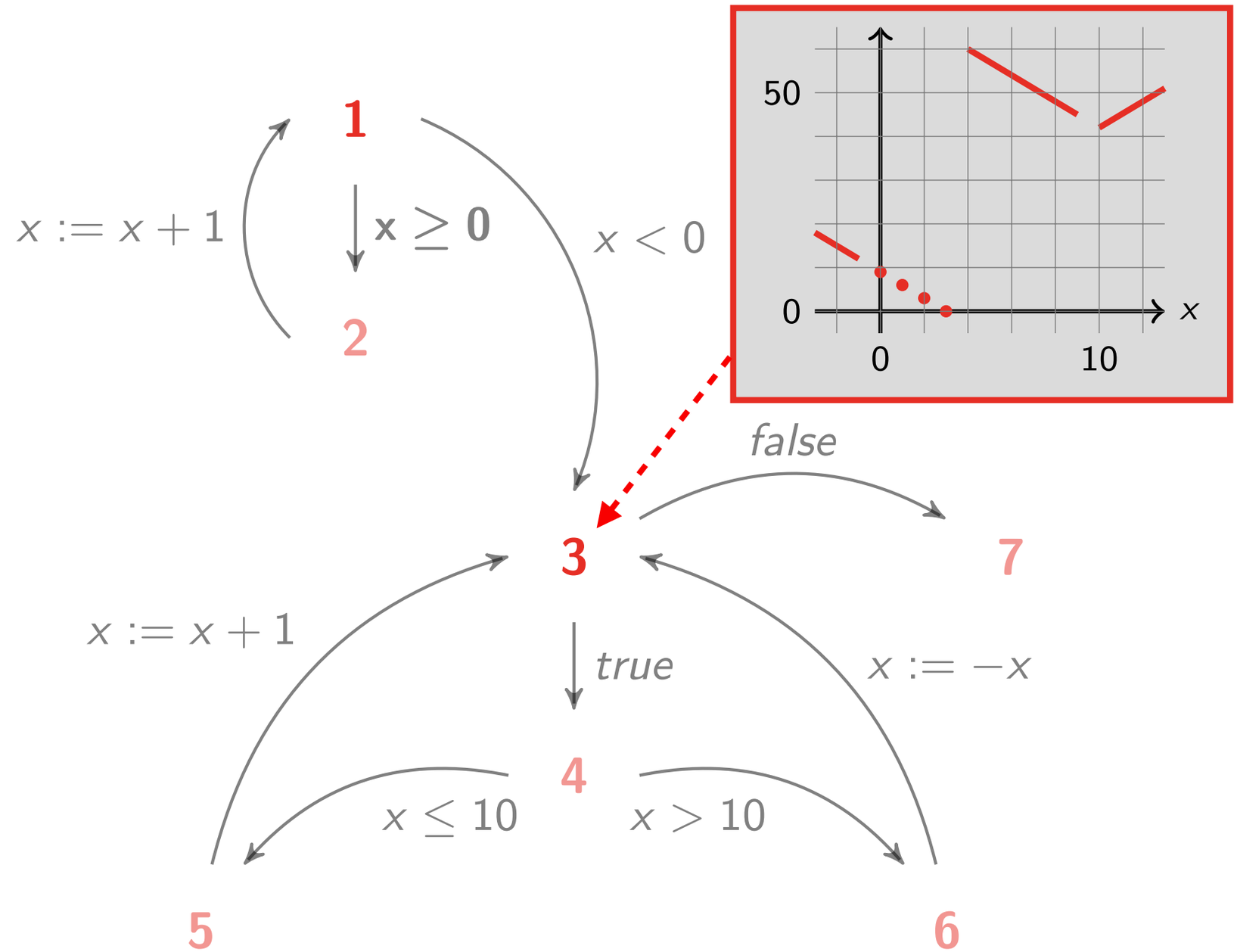
  else

    [6] $x := -x$

od [7]

Property

$\Box\Diamond\ x = 3$



Example

int : x, y

while [1] $(x \geq 0)$

  [2] $x := x + 1$

od

while [3] (true) do

  if [4] ($x \leq 10$)

    [5] $x := x + 1$

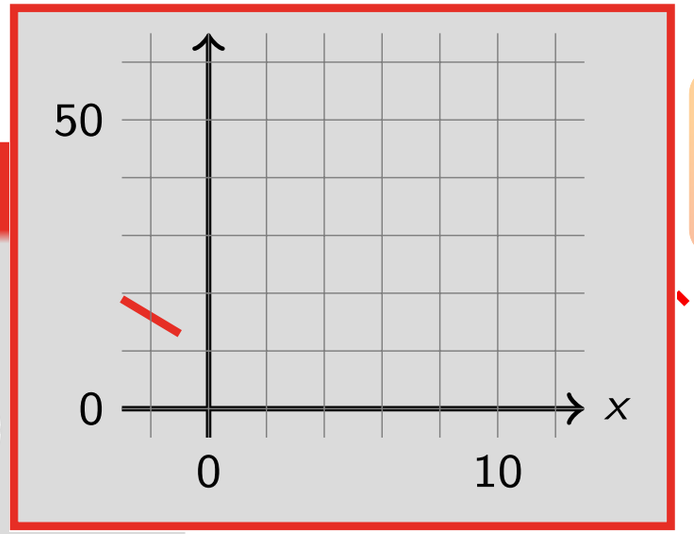
  else

    [6] $x := -x$

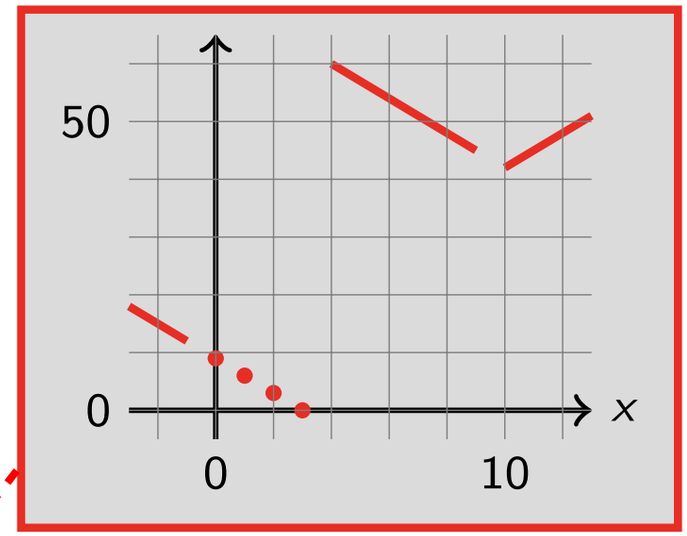
od [7]

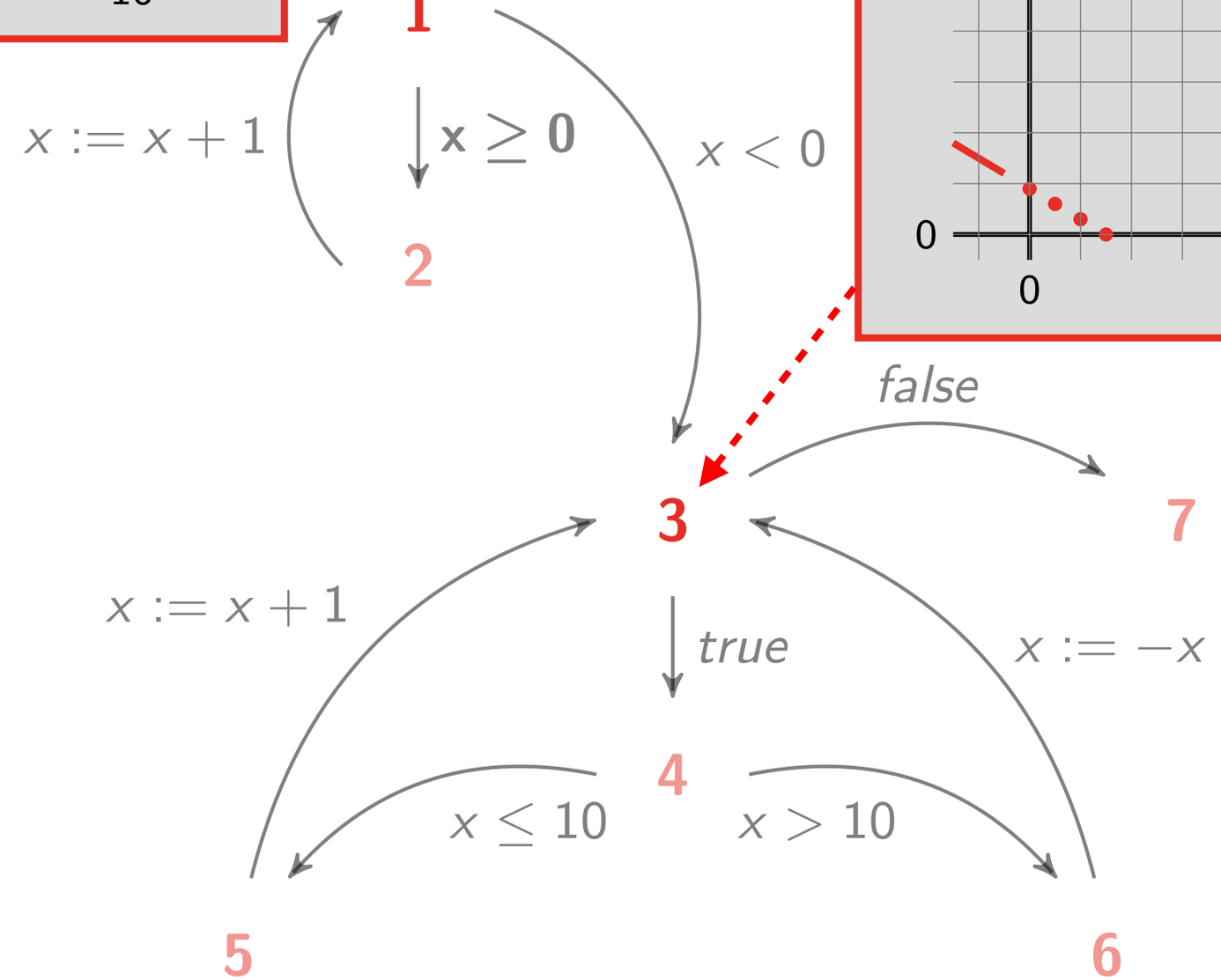
Property

$\Box\Diamond\ x = 3$



the analysis gives $x < 0$ as
sufficient precondition





Guarantee Properties

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

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

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

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

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

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

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

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

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

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

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

$$\mathcal{T}_g^\varphi \in \Sigma \rightharpoonup \mathbb{O}$$

$$\mathcal{T}_g^\varphi \stackrel{\text{def}}{=} \mathbf{lfp}\ F_g^\varphi$$

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

$$F_g^\varphi(v)s \stackrel{\text{def}}{=} \begin{cases} \mathbf{0} & s \in \varphi \\ \mathbf{sup}\{\ v(s') + 1 \mid s \to s'\ \} & s \in \widetilde{\mathbf{pre}}(\mathbf{dom}(v)) \ \wedge\ s \notin \varphi \\ \mathbf{undefined} & \mathbf{otherwise} \end{cases}$$

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

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

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

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

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

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

$$\mathcal{T}_g^\varphi \in \Sigma \rightharpoonup \mathbb{O}$$

$$\mathcal{T}_g^\varphi \stackrel{\text{def}}{=} \mathbf{lfp}\ F_g^\varphi$$

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

$$F_g^\varphi(v)s \stackrel{\text{def}}{=} \begin{cases} \mathbf{0} & s \in \varphi \\ \mathbf{sup}\{\ v(s') + 1 \mid s \to s'\ \} & s \in \widetilde{\mathbf{pre}}(\mathbf{dom}(v)) \ \wedge\ s \notin \varphi \\ \mathbf{undefined} & \text{otherwise} \end{cases}$$

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

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

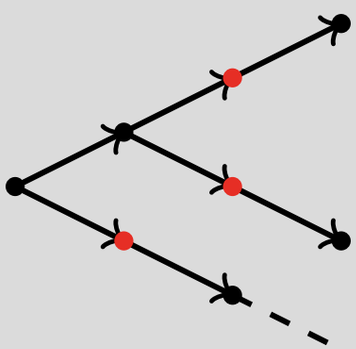
$$\mathcal{T}^{\varphi}_{g} \in \Sigma \rightharpoonup \mathbb{O}$$

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

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

$$F^{\varphi}_{g}(v)s \stackrel{\text{def}}{=} \begin{cases} \mathbf{0} & s \in \varphi \\ \mathbf{sup}\{\ v(s') + \mathbf{1} \mid s \to s'\ \} & s \in \widetilde{\mathbf{pre}}(\mathbf{dom}(v)) \ \wedge\ s \notin \varphi \\ \mathbf{undefined} & \mathbf{otherwise} \end{cases}$$

Example



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

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

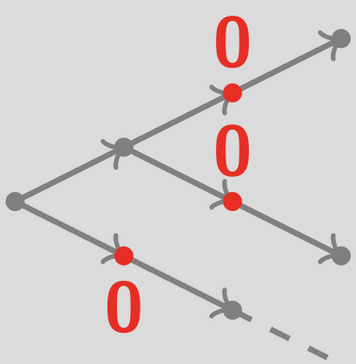
$$\mathcal{T}^{\varphi}_{g} \in \Sigma \rightharpoonup \mathbb{O}$$

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

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

$$F^{\varphi}_{g}(v)s \stackrel{\text{def}}{=} \begin{cases} \mathbf{0} & s \in \varphi \\ \mathbf{sup}\{\ v(s') + \mathbf{1} \mid s \to s'\ \} & s \in \widetilde{\mathbf{pre}}(\mathbf{dom}(v)) \ \wedge\ s \notin \varphi \\ \mathbf{undefined} & \mathbf{otherwise} \end{cases}$$

Example



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

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

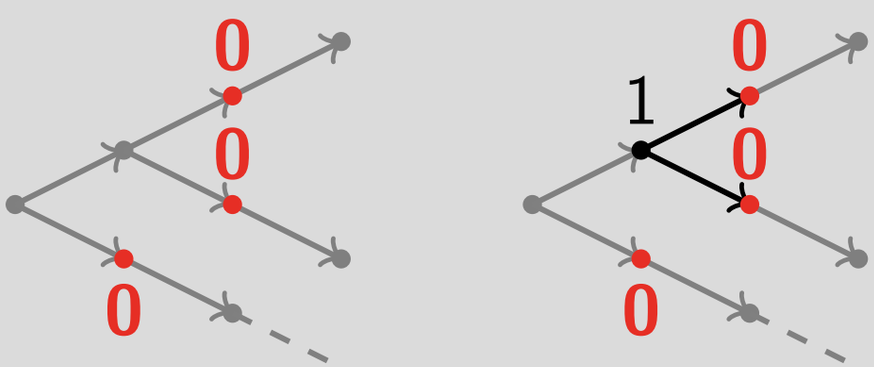
$$\mathcal{T}^{\varphi}_{g} \in \Sigma \rightharpoonup \mathbb{O}$$

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

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

$$F^{\varphi}_{g}(v)s \stackrel{\text{def}}{=} \begin{cases} \mathbf{0} & s \in \varphi \\ \mathbf{sup}\{\ v(s') + \mathbf{1} \mid s \rightarrow s'\ \} & s \in \widetilde{\mathbf{pre}}(\mathbf{dom}(v)) \ \wedge\ s \notin \varphi \\ \mathbf{undefined} & \text{otherwise} \end{cases}$$

Example



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

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

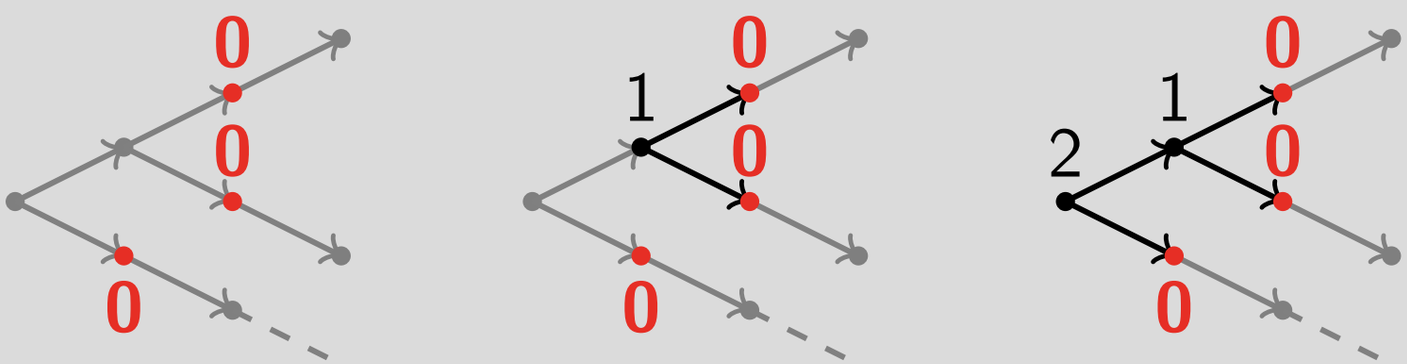
$$\mathcal{T}_g^{\varphi} \in \Sigma \rightharpoonup \mathbb{O}$$

$$\mathcal{T}_g^{\varphi} \stackrel{\text{def}}{=} \mathbf{lfp}\ F_g^{\varphi}$$

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

$$F_g^{\varphi}(v)s \stackrel{\text{def}}{=} \begin{cases} \mathbf{0} & s \in \varphi \\ \mathbf{sup}\{\ v(s') + \mathbf{1} \mid s \rightarrow s'\ \} & s \in \widetilde{\mathbf{pre}}(\mathbf{dom}(v)) \ \wedge\ s \notin \varphi \\ \mathbf{undefined} & \text{otherwise} \end{cases}$$

Example



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

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

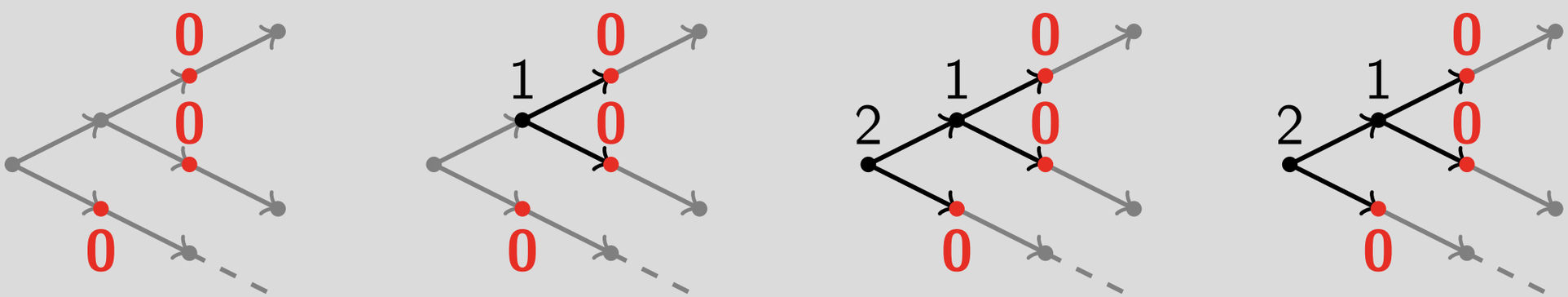
$$\mathcal{T}_g^{\varphi} \in \Sigma \rightharpoonup \mathbb{O}$$

$$\mathcal{T}_g^{\varphi} \stackrel{\text{def}}{=} \mathbf{lfp}\ F_g^{\varphi}$$

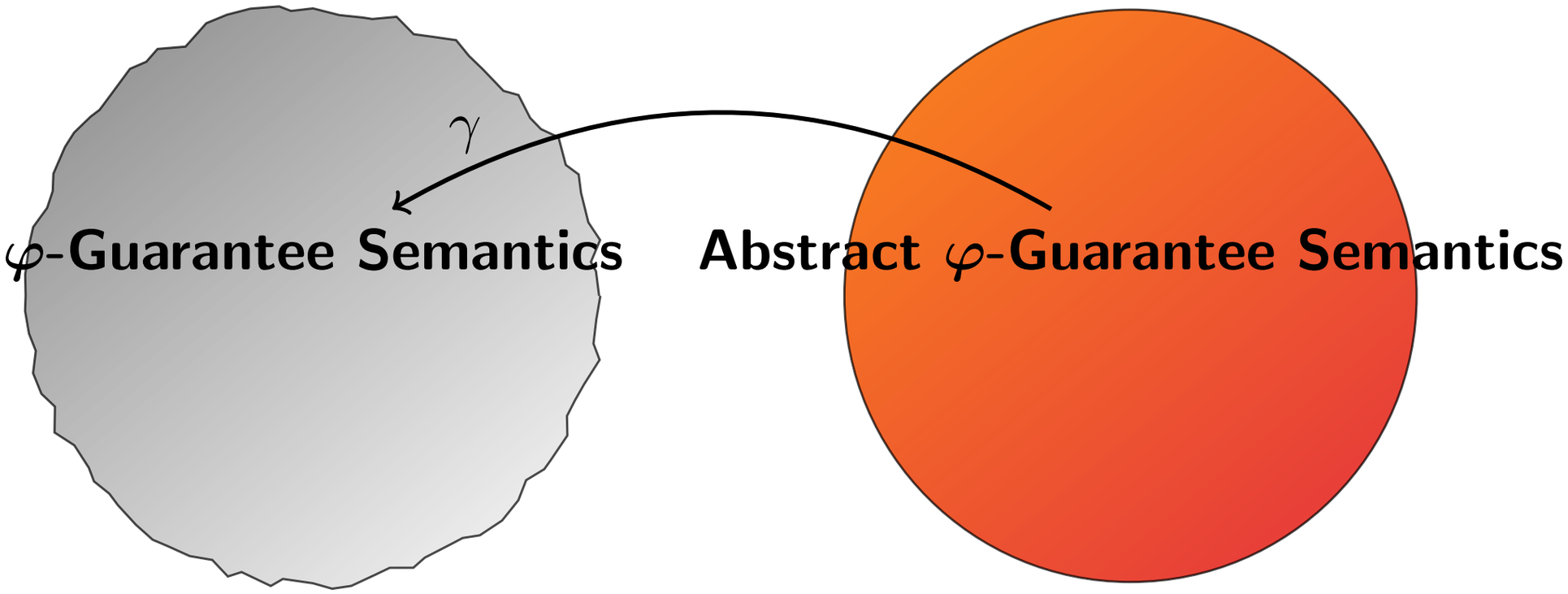
idea = define a ranking function **counting the number of program steps** from the next occurrence of the property φ

$$F_g^{\varphi}(v)s \stackrel{\text{def}}{=} \begin{cases} \mathbf{0} & s \in \varphi \\ \mathbf{sup}\{\ v(s') + \mathbf{1} \mid s \to s'\ \} & s \in \widetilde{\mathbf{pre}}(\mathbf{dom}(v)) \ \wedge\ s \notin \varphi \\ \mathbf{undefined} & \text{otherwise} \end{cases}$$

Example



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


γ
φ-Guarantee Semantics
Abstract φ-Guarantee Semantics

Example

int : $x,\ y$

while [1] $(x \geq 0)$ do

  [2] $x := x + 1$

od

while [3] (true) do

  if [4] ($x \leq 10$)

    [5] $x := x + 1$

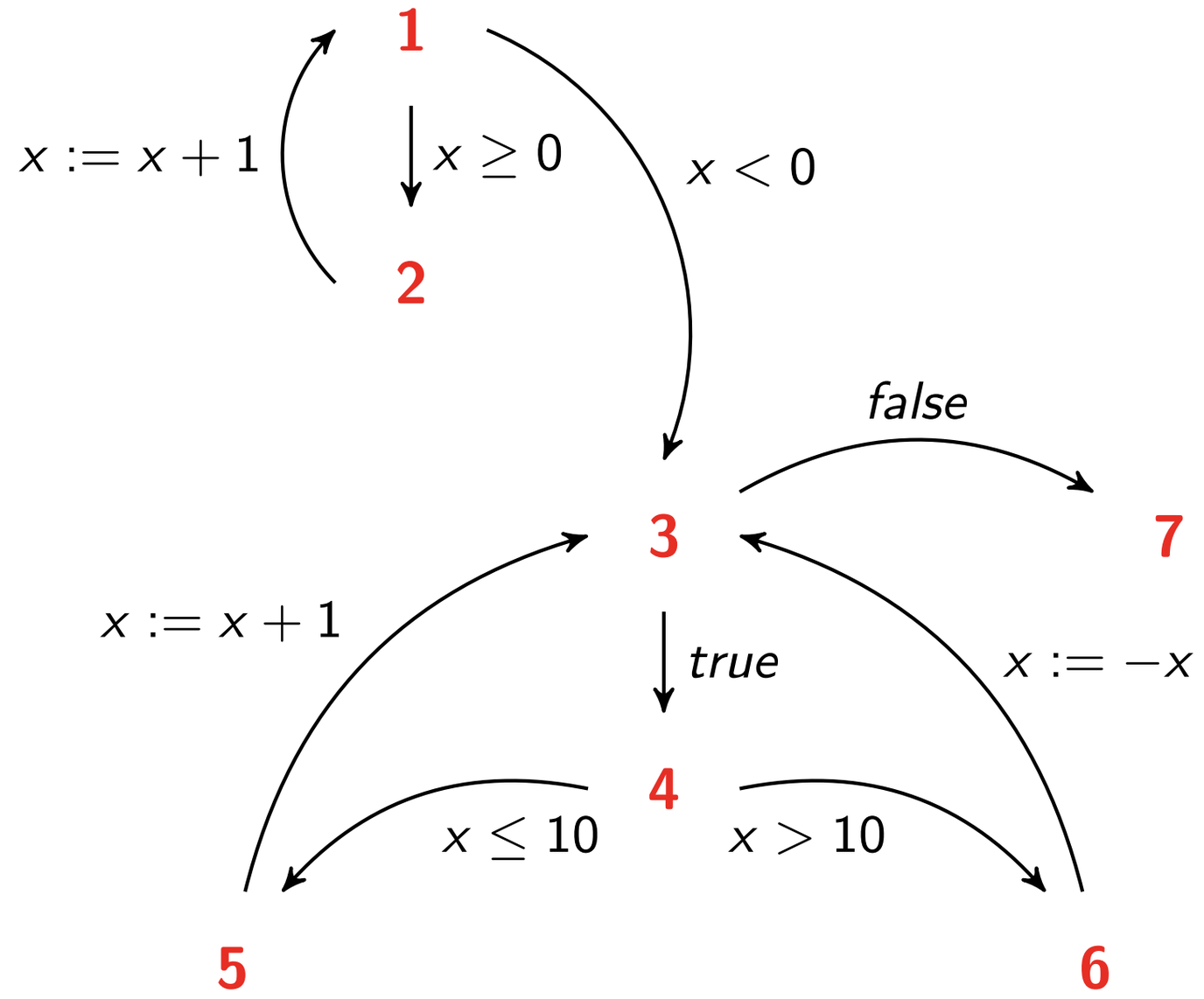
  else

    [6] $x := -x$

od [7]

Property

$\Diamond\ x = 3$



Example

int : $x,\ y$

while [1] $(x \geq 0)$ do

  [2] $x := x + 1$

od

while [3] (true) do

  if [4] ($x \leq 10$)

    [5] $x := x + 1$

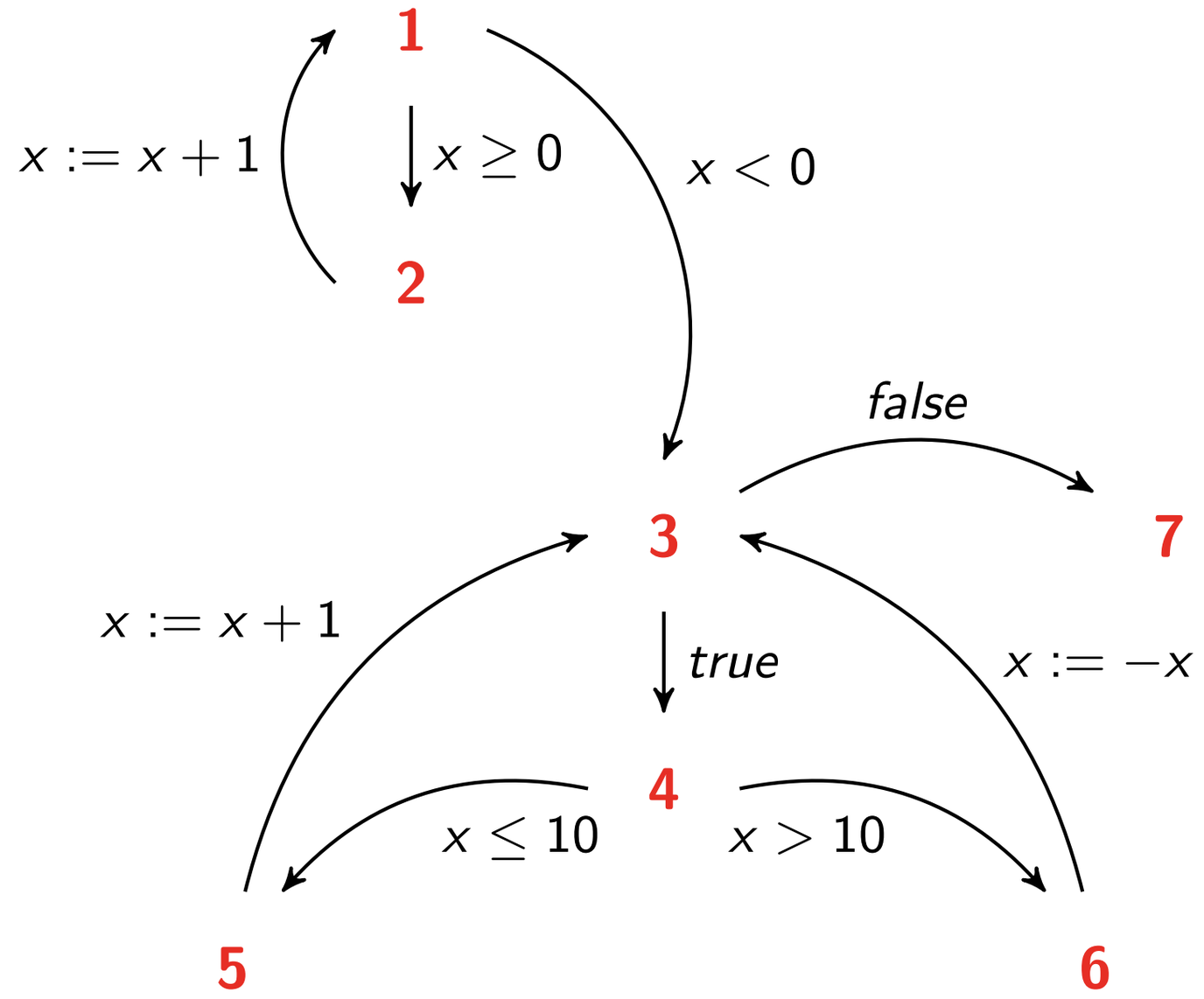
  else

    [6] $x := -x$

od [7]

Property

$\Diamond\ x = 3$



Example

int : x, y

while [1] $(x \geq 0)$ do

  [2] $x := x + 1$

od

while [3] (true) do

  if [4] ($x \leq 10$)

    [5] $x := x + 1$

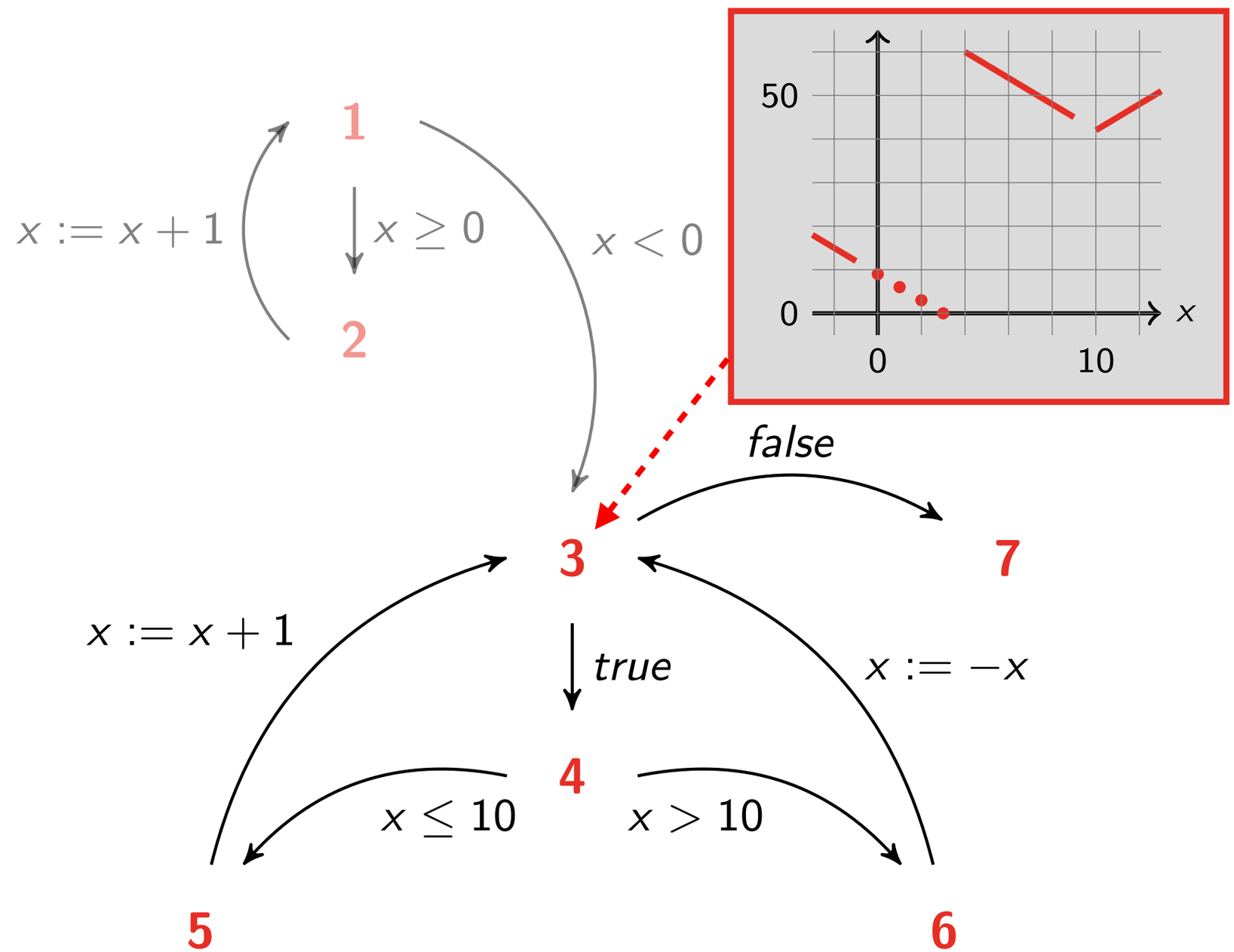
  else

    [6] $x := -x$

od [7]

Property

$\Diamond\ x = 3$



Example

int : x, y

while [1] $(x \geq 0)$ do

  [2] $x := x + 1$

od

while [3] (true) do

  if [4] ($x \leq 10$)

    [5] $x := x + 1$

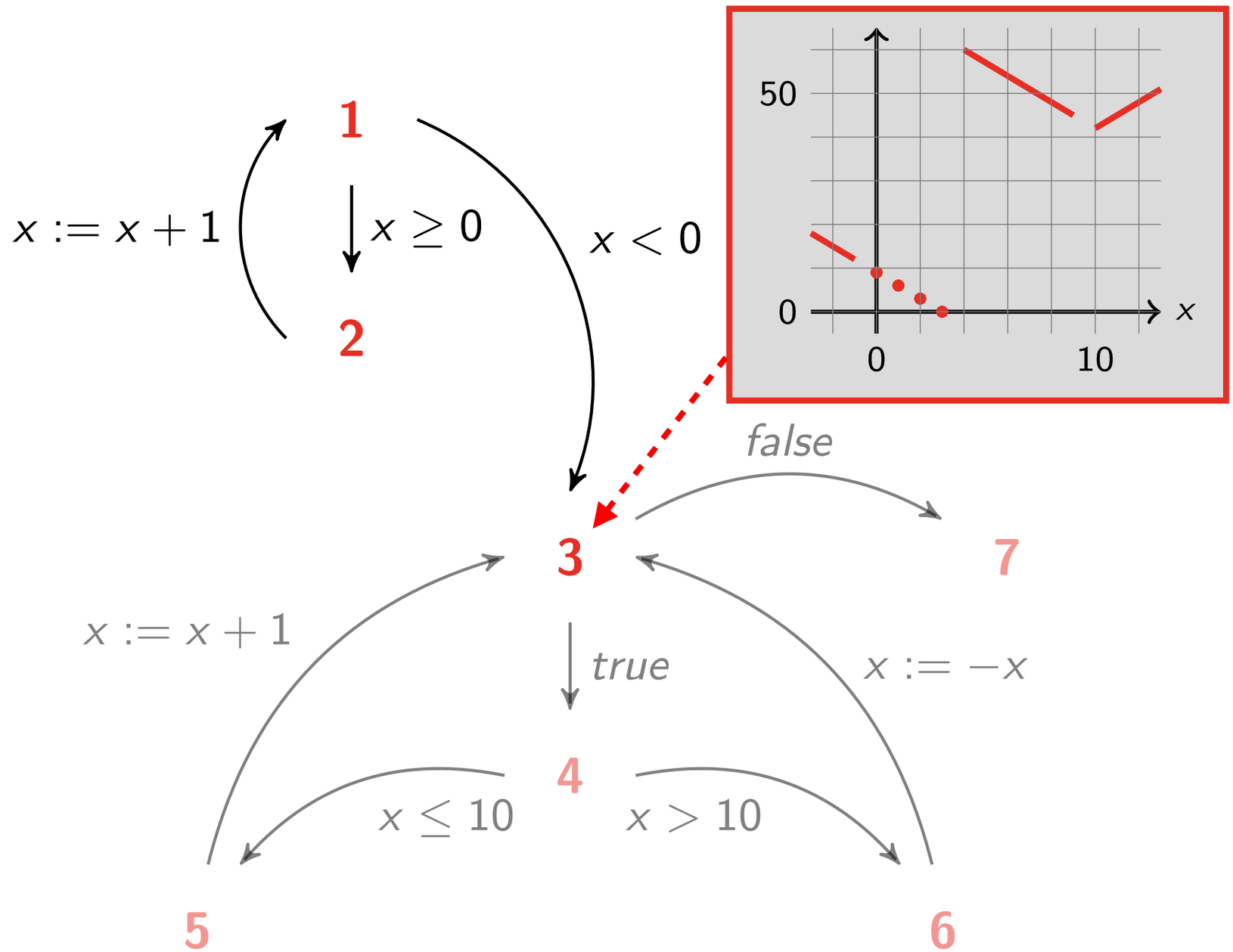
  else

    [6] $x := -x$

od [7]

Property

$\Diamond\ x = 3$



Example

int : x, y

[1] while $(x \geq 0)$

  [2] $x := x + 1$

od

[3] while (true) do

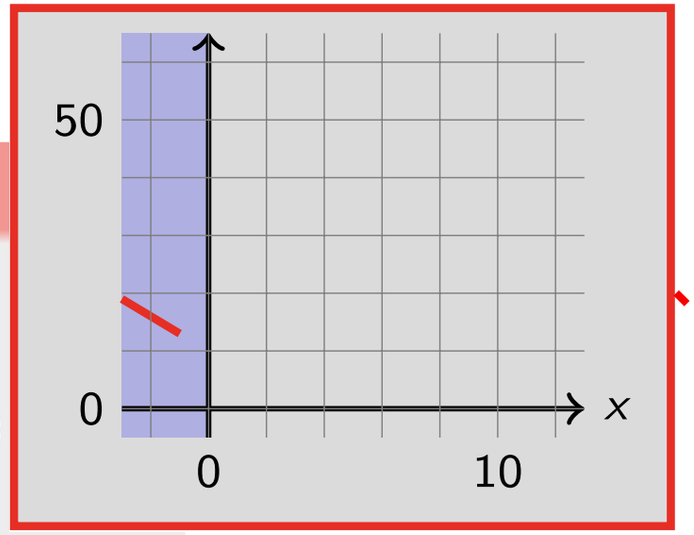
  [4] if $(x \leq 10)$

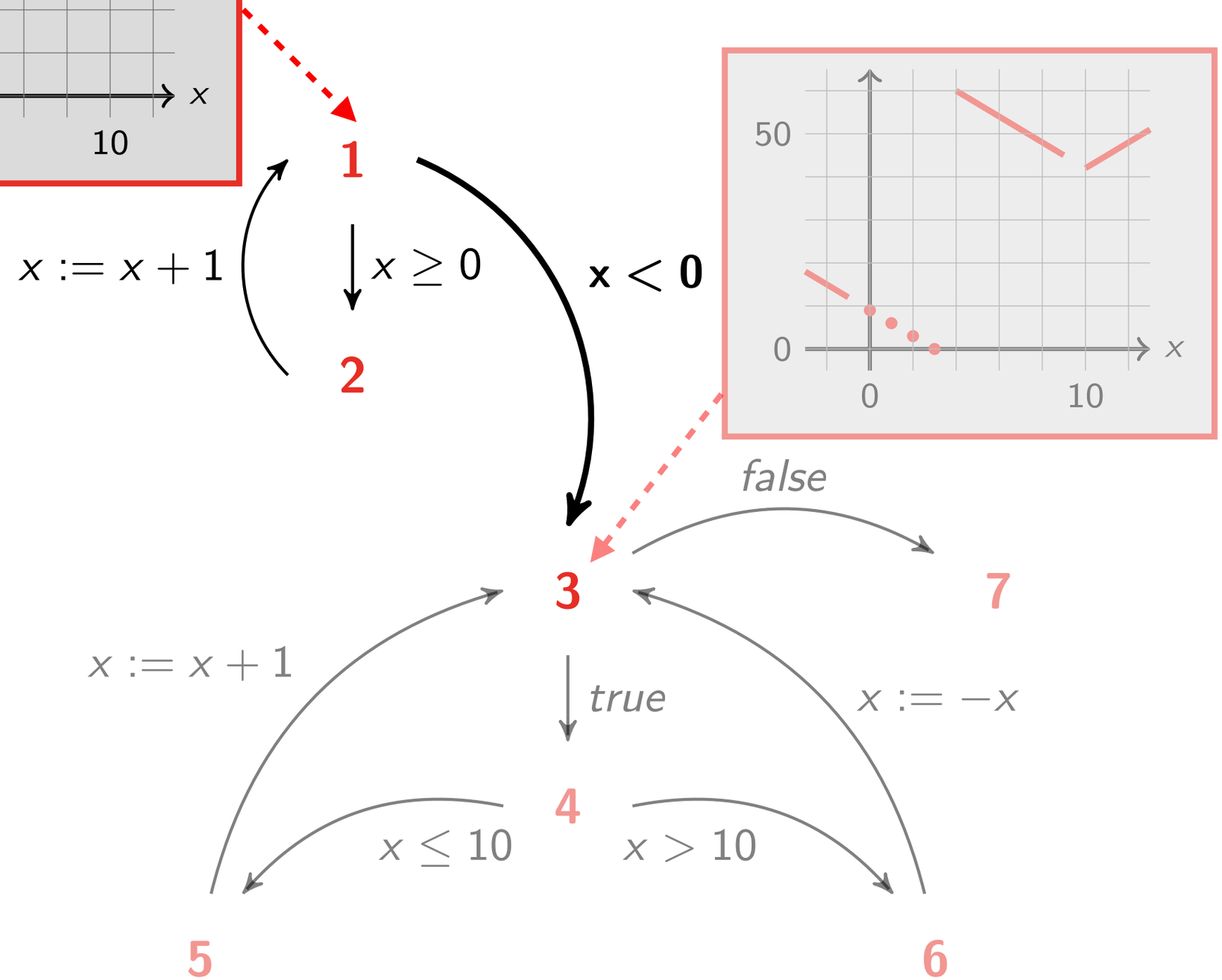
    [5] $x := x + 1$

  else

    [6] $x := -x$

od [7]





Property

$\Diamond\ x = 3$

Example

int : x, y

[1] while $(x \geq 0)$

  [2] $x := x + 1$

od

[3] while (true) do

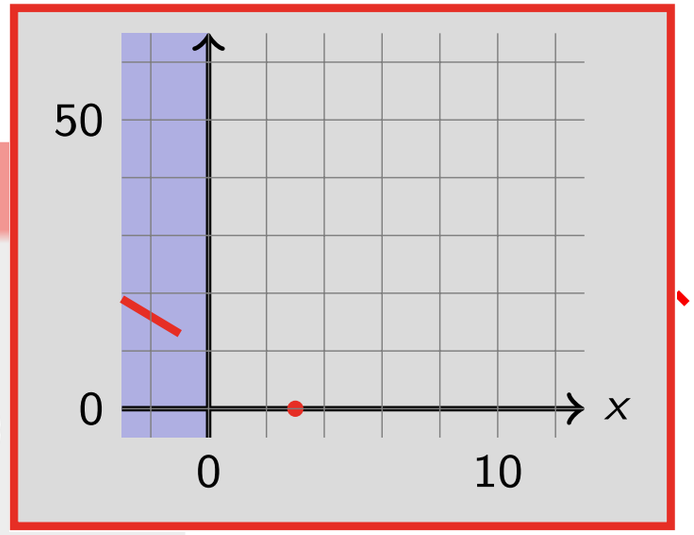
  [4] if $(x \leq 10)$

    [5] $x := x + 1$

  else

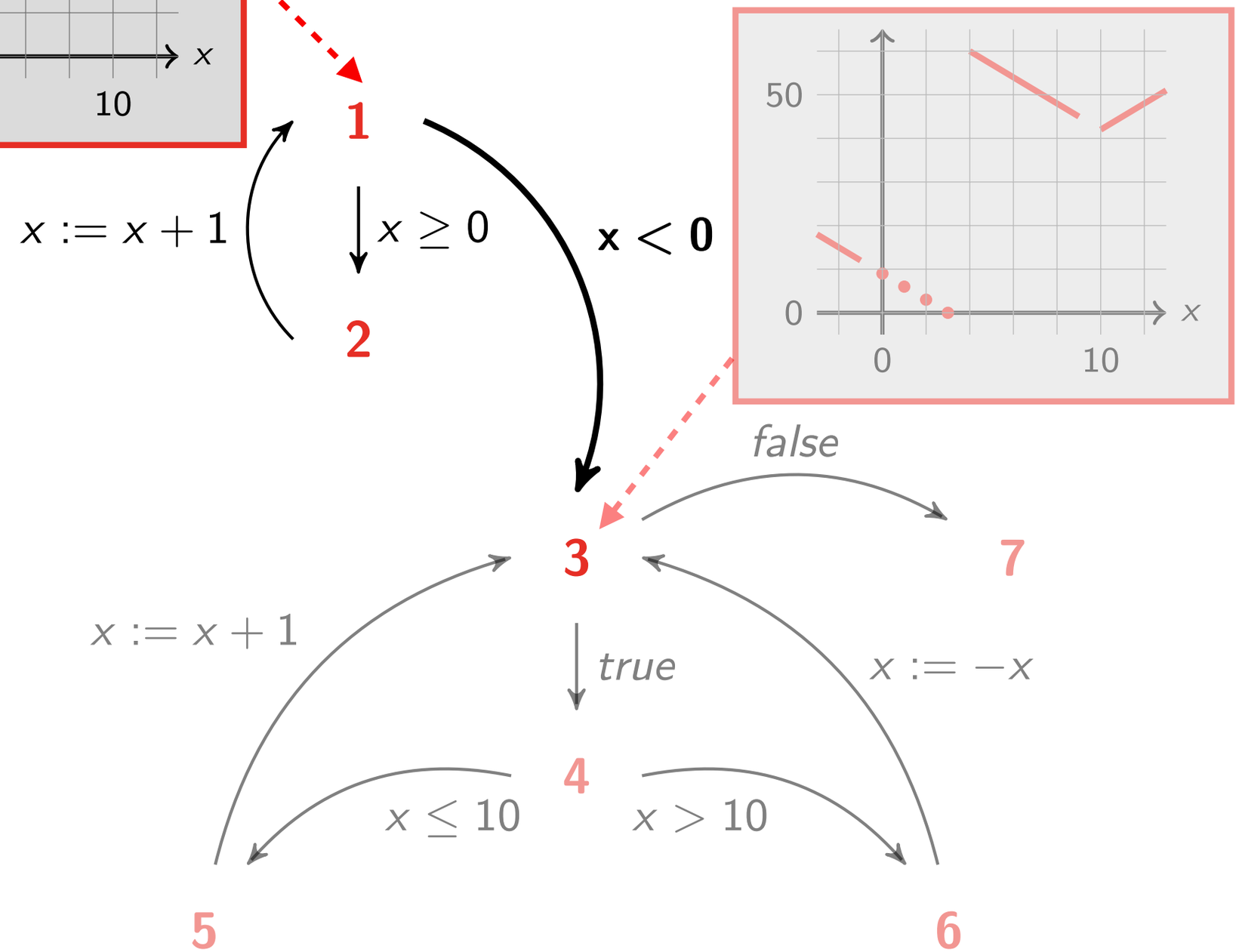
    [6] $x := -x$

od [7]



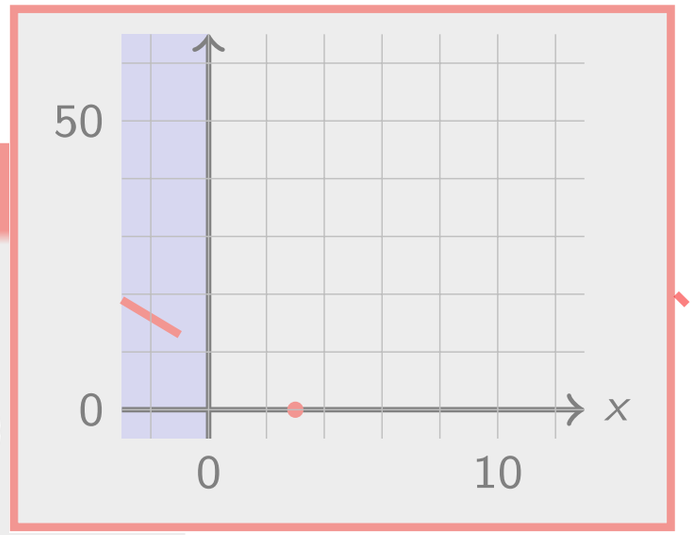
Property

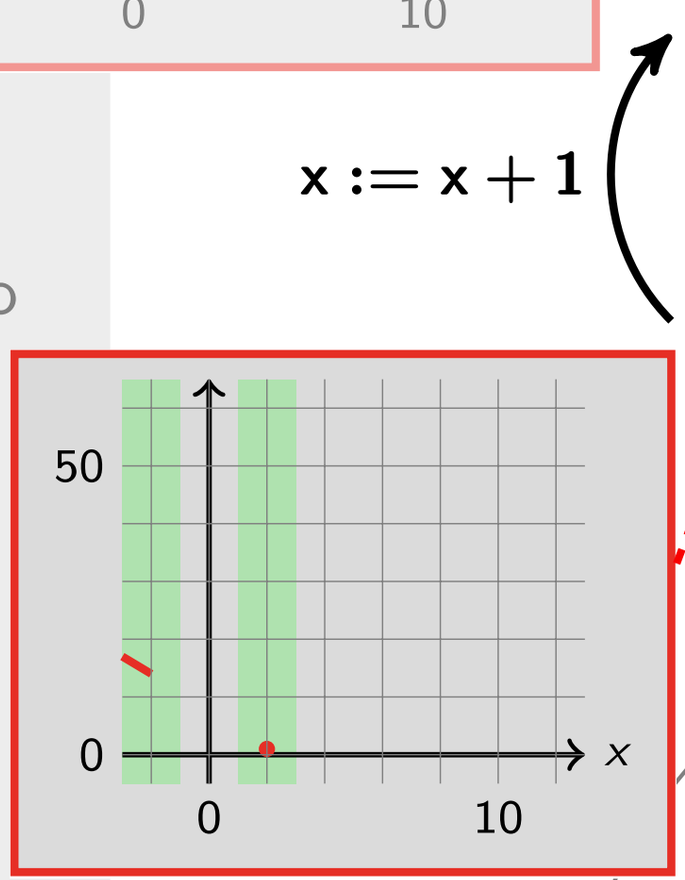
$\Diamond\ x = 3$



Example

```
int : x, y
while [1](x ≥ 0)
    [2]x := x + 1
od
while [3]( true ) do
    if [4]( x ≤ 10 )
        [5]x := x + 1
    else
        [6]x := −x
od[7]
```



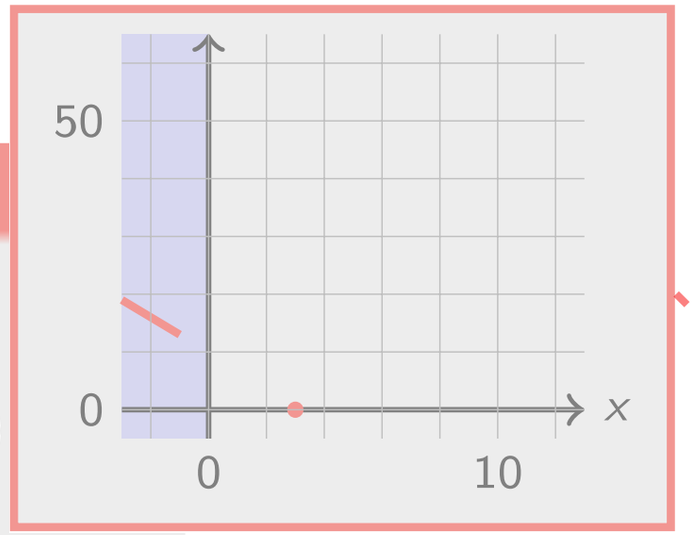


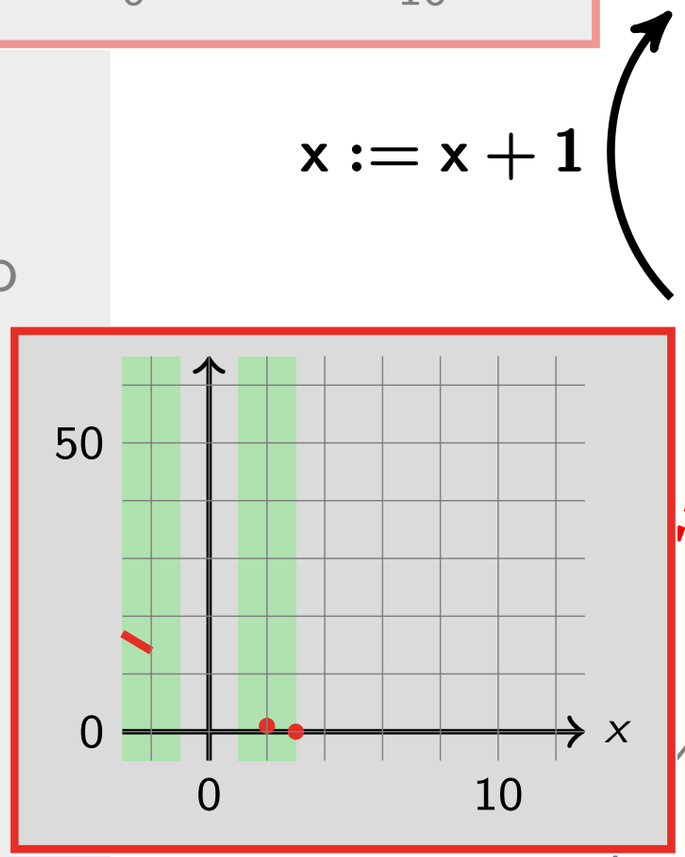
Property

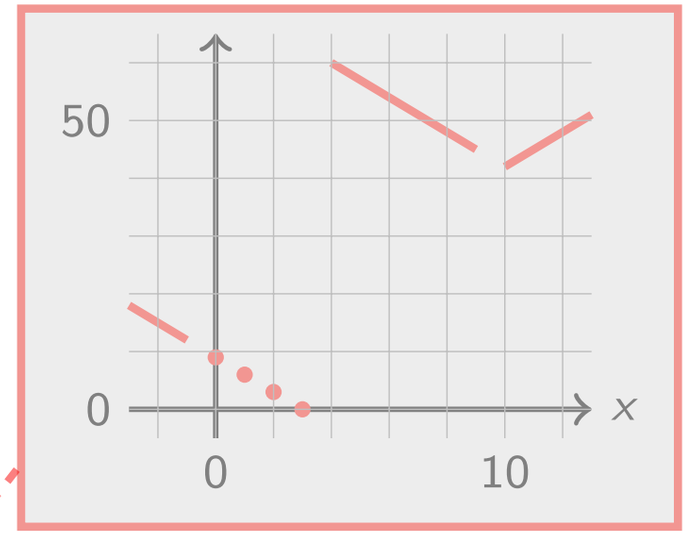
$\Diamond\ x = 3$

Example

```
int : x, y
while [1](x ≥ 0)
    [2]x := x + 1
od
while [3]( true ) do
    if [4]( x ≤ 10 )
        [5]x := x + 1
    else
        [6]x := −x
od[7]
```







Property

$\Diamond\ x = 3$

Example

int : x, y

[1] while $(x \geq 0)$

  [2] $x := x + 1$

od

[3] while (true) do

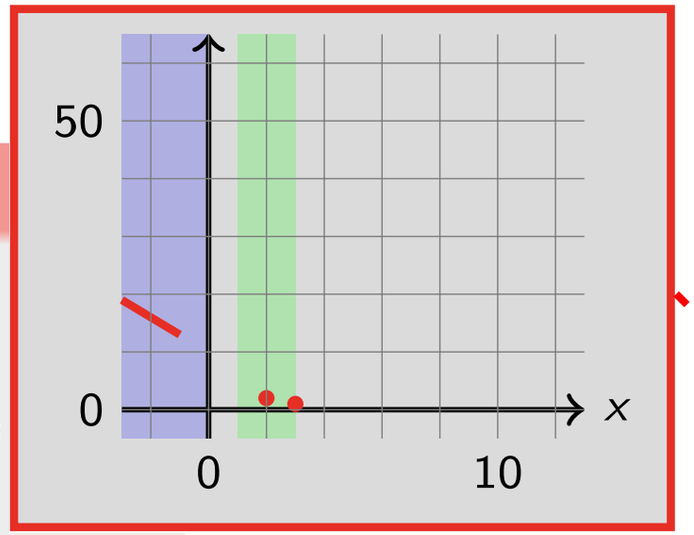
  [4] if $(x \leq 10)$

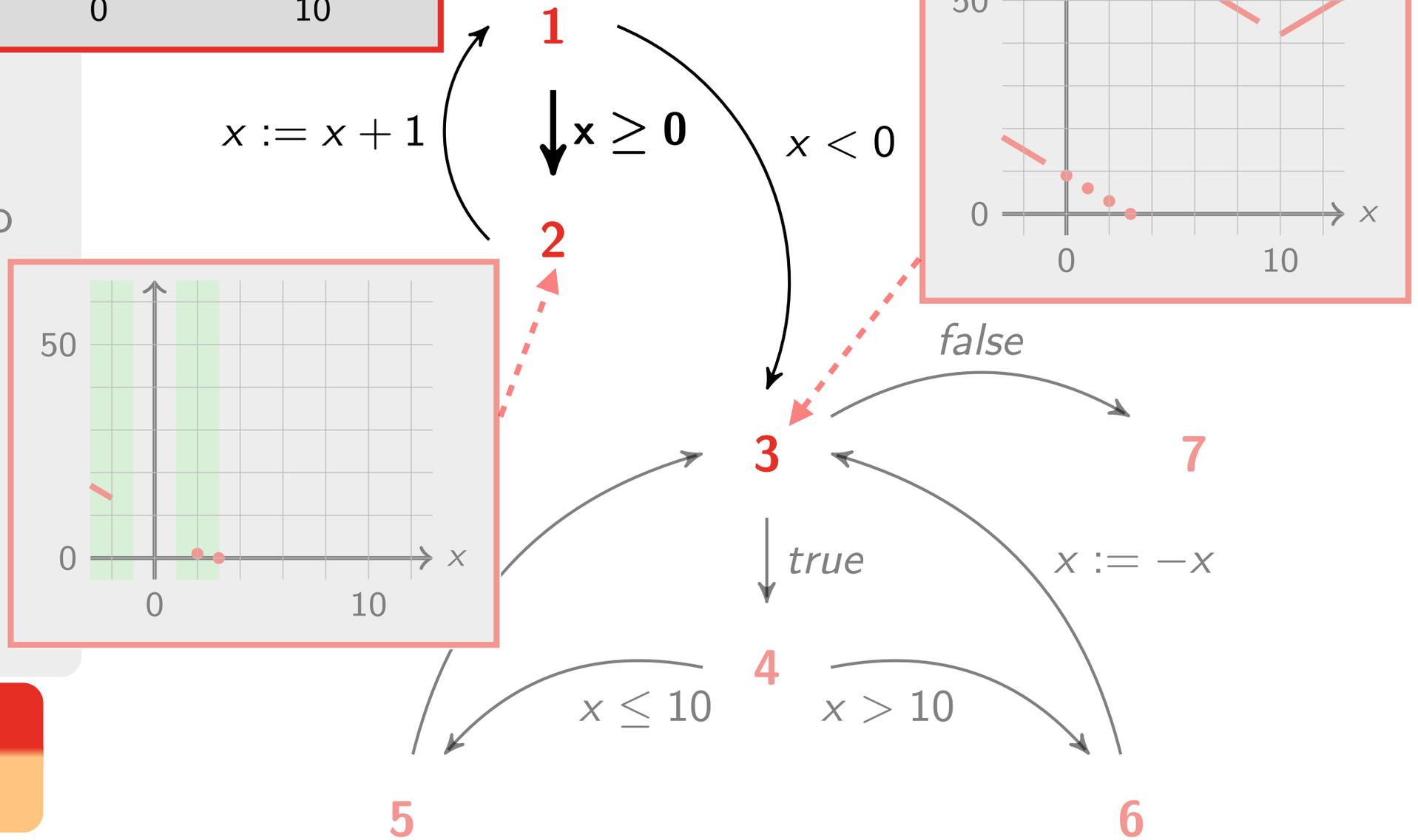
    [5] $x := x + 1$

  else

    [6] $x := -x$

od [7]





Property

$\Diamond\ x = 3$

Example

int : x, y

[1] while $(x \geq 0)$

  [2] $x := x + 1$

od

[3] while (true) do

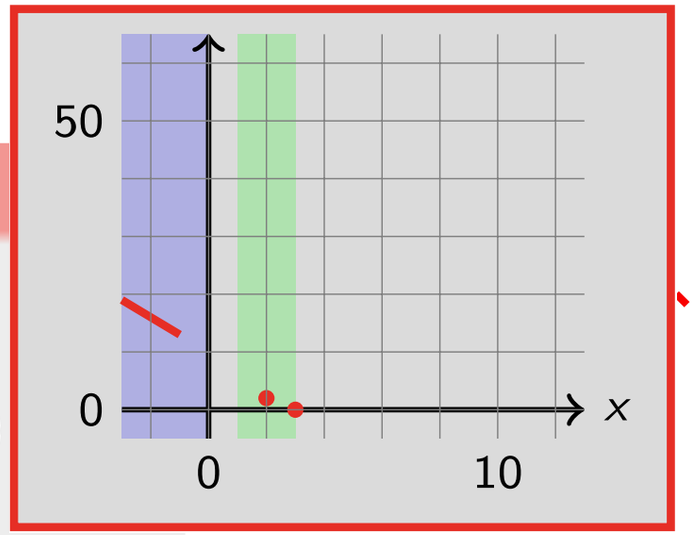
  [4] if $(x \leq 10)$

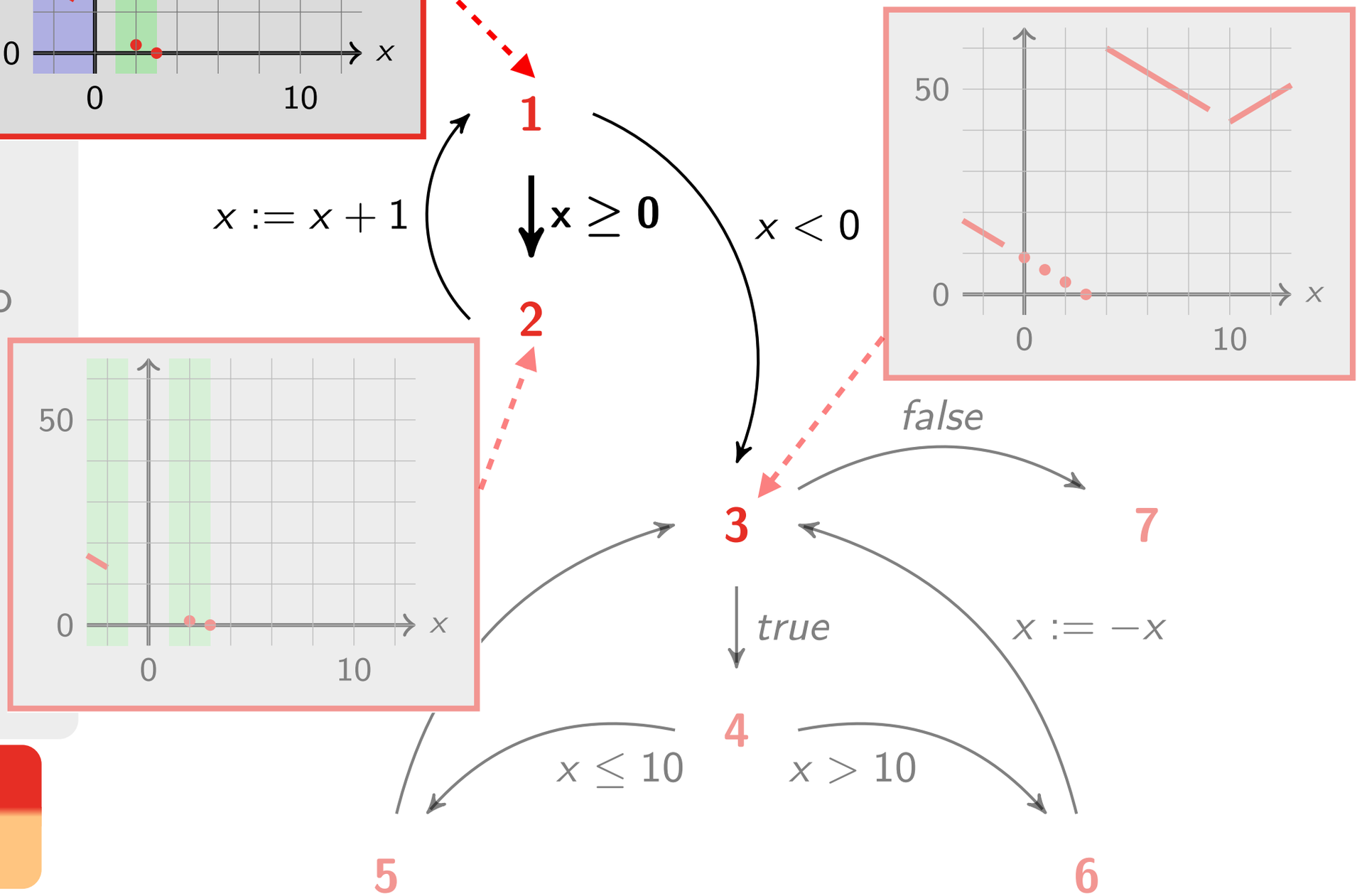
    [5] $x := x + 1$

  else

    [6] $x := -x$

od [7]





Property

$\Diamond\ x = 3$

Example

int : x, y

[1] while $(x \geq 0)$

  [2] $x := x + 1$

od

[3] while (true) do

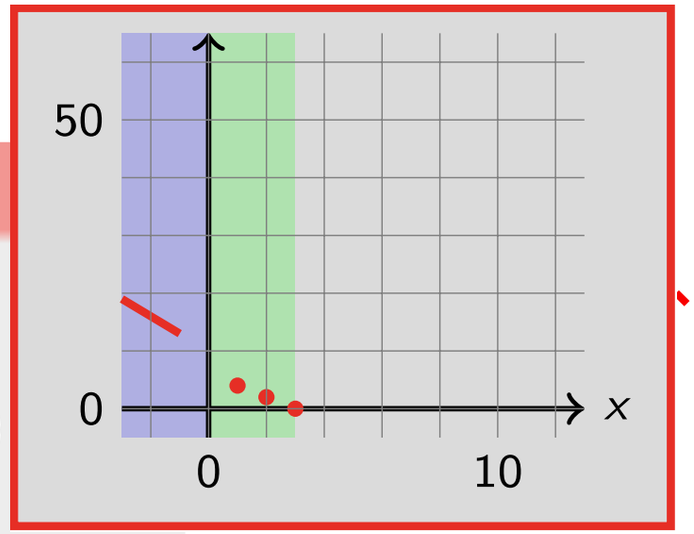
  [4] if $(x \leq 10)$

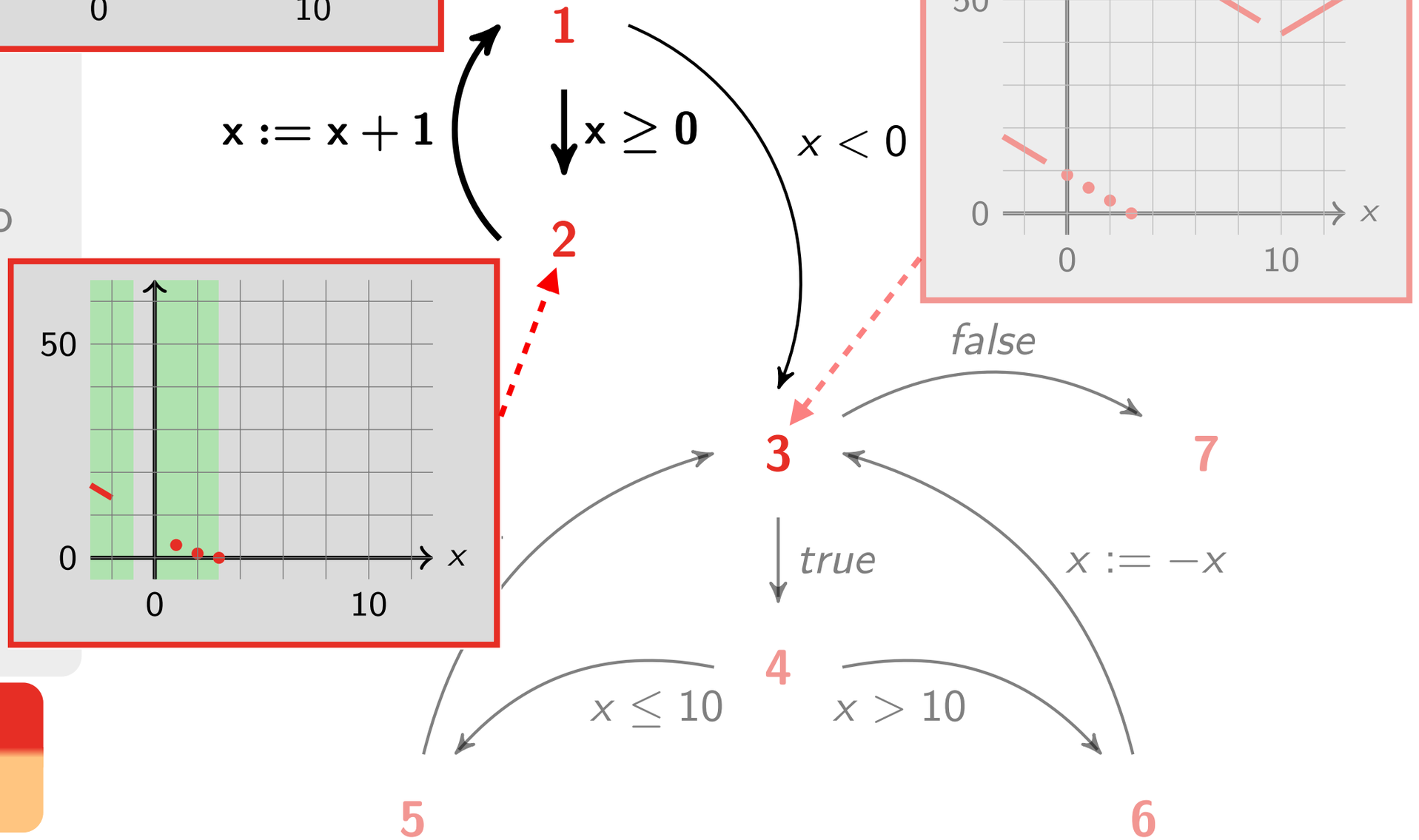
    [5] $x := x + 1$

  else

    [6] $x := -x$

od [7]





Property

$\Diamond\ x = 3$

Example

int : x, y

[1] while $(x \geq 0)$

  [2] $x := x + 1$

od

[3] while (true) do

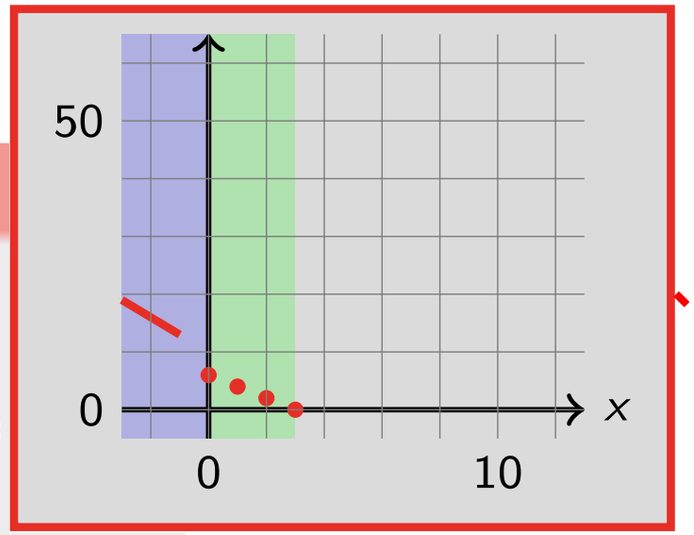
  [4] if $(x \leq 10)$

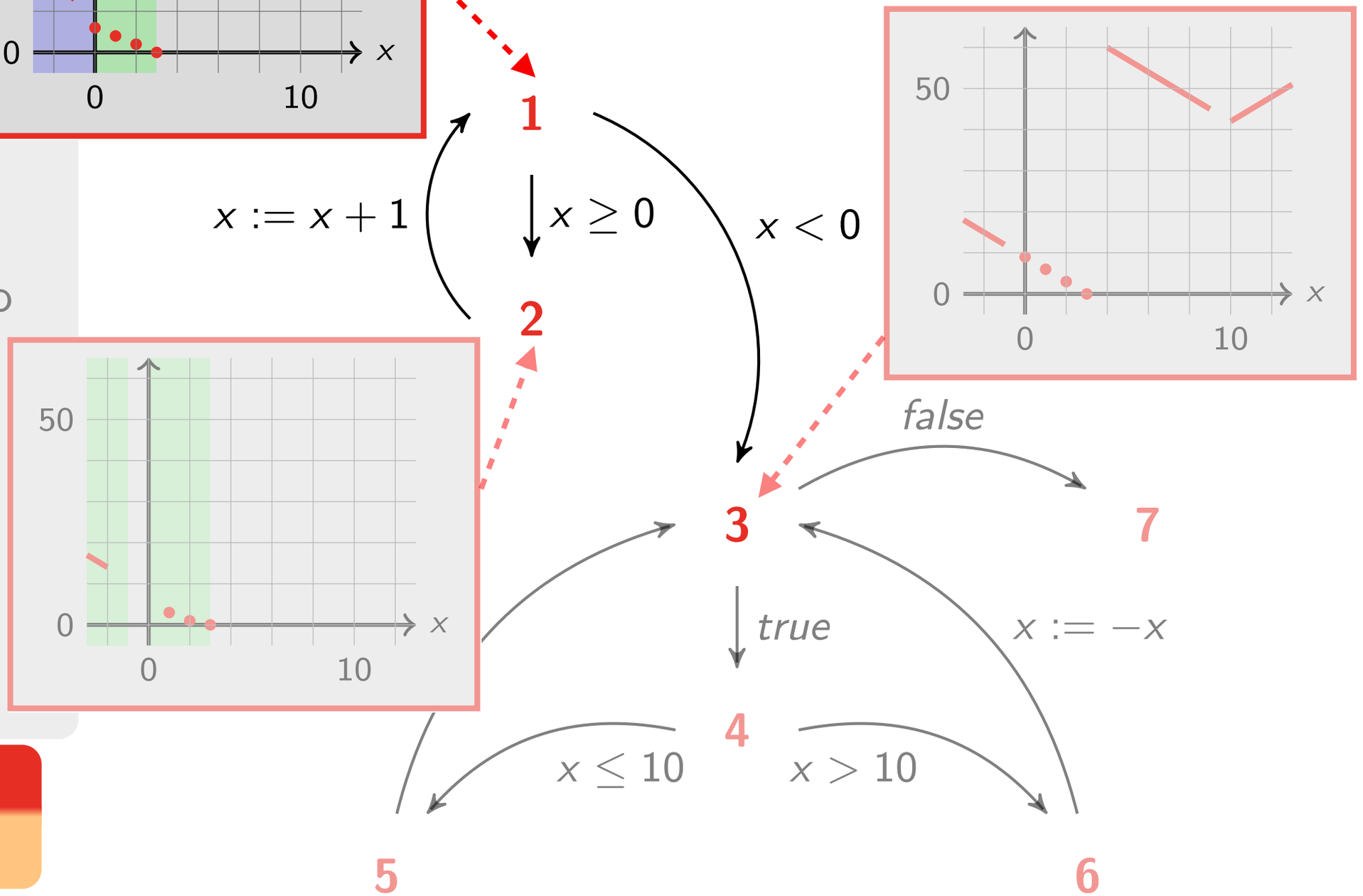
    [5] $x := x + 1$

  else

    [6] $x := -x$

od [7]



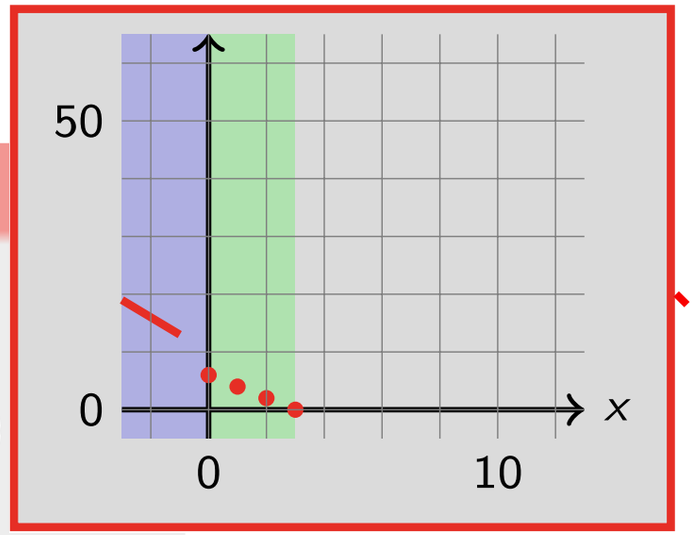


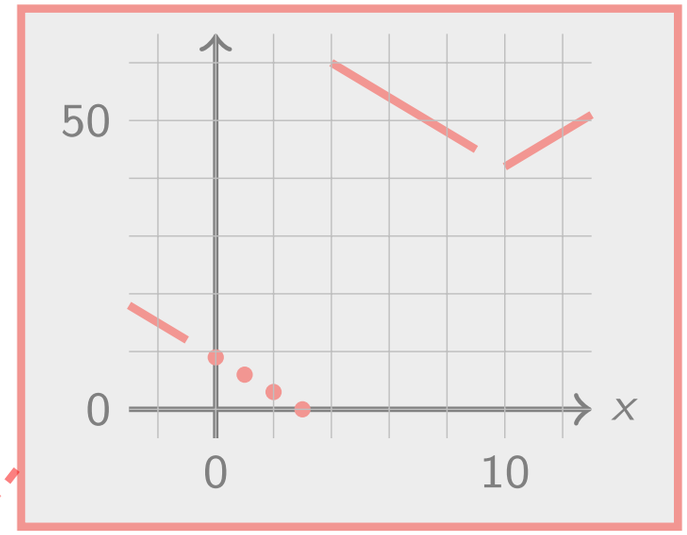
Property

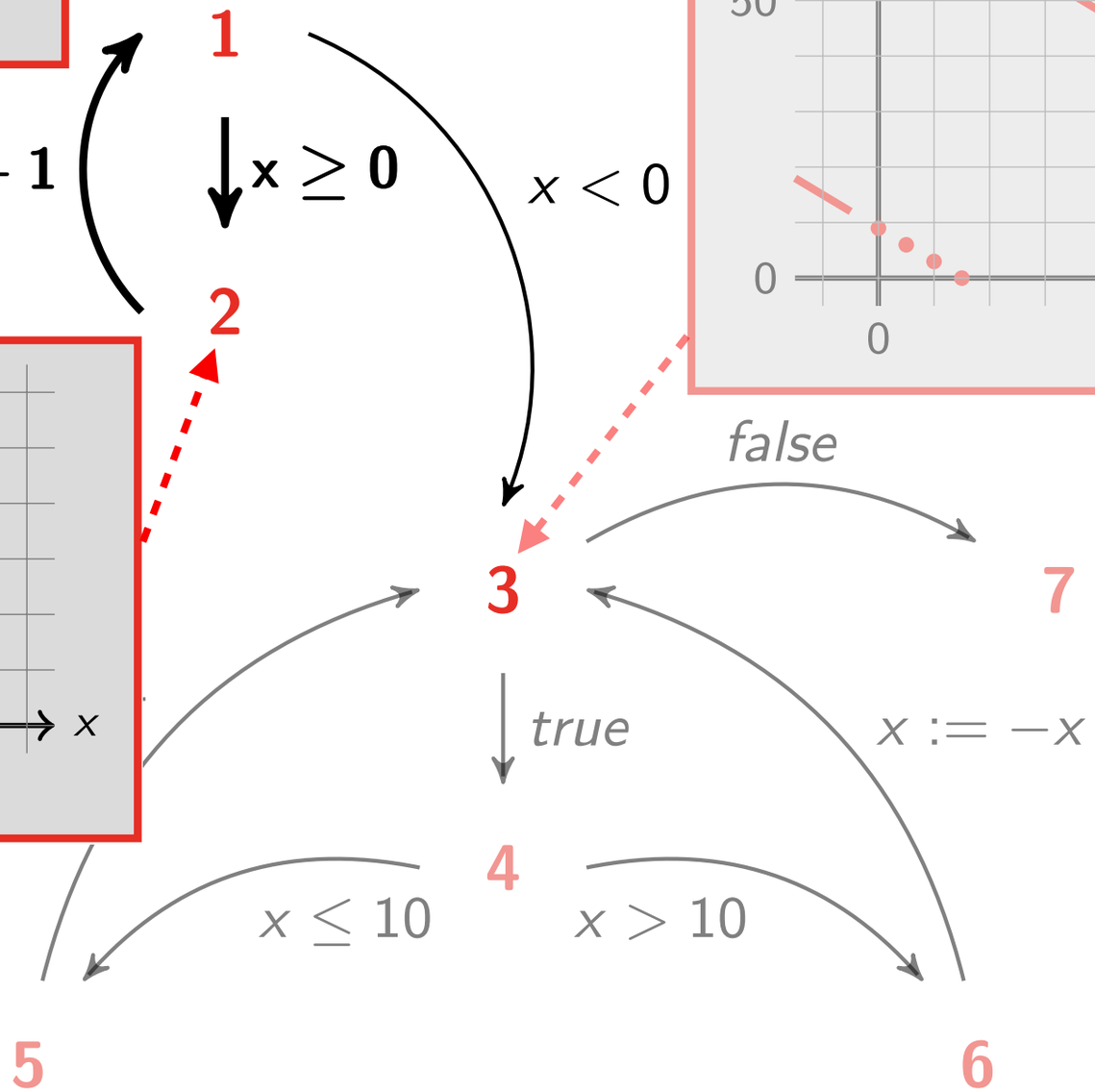
$\Diamond\ x = 3$

Example

```
int : x, y
while [1](x ≥ 0)
    [2]x := x + 1
od
while [3]( true ) do
    if [4]( x ≤ 10 )
        [5]x := x + 1
    else
        [6]x := −x
od[7]
```







Property

$\Diamond\ x = 3$

Example

int : x, y

while [1] $(x \geq 0)$

  [2] $x := x + 1$

od

while [3] (true) do

  if [4] ($x \leq 10$)

    [5] $x := x + 1$

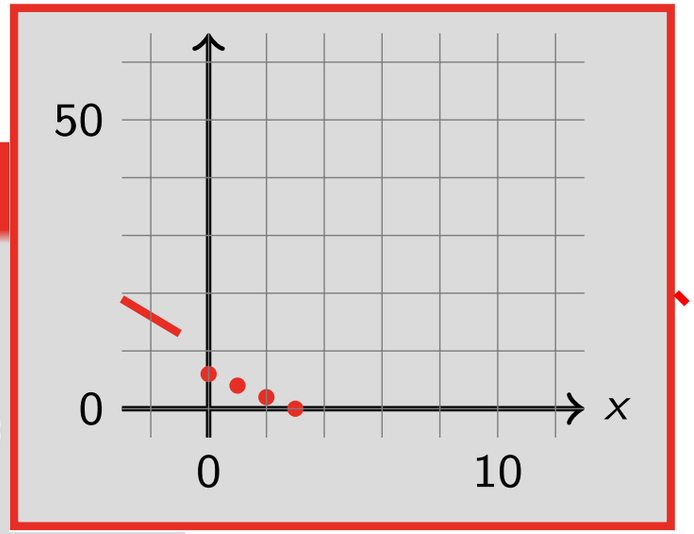
  else

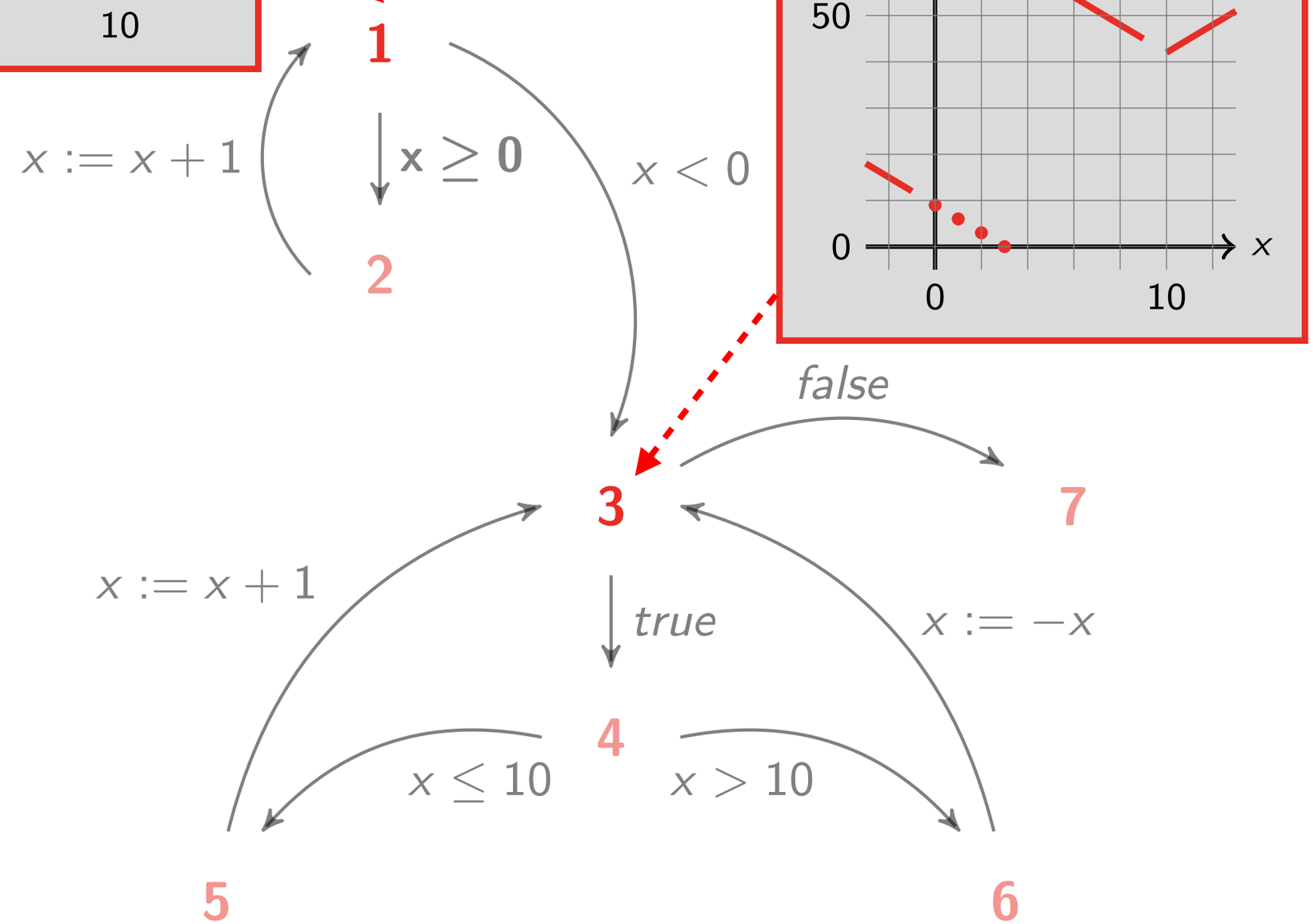
    [6] $x := -x$

od [7]

Property

$\Diamond\ x = 3$





Example

int : x, y

while [1] $(x \geq 0)$

  [2] $x := x + 1$

od

while [3] (true) do

  if [4] ($x \leq 10$)

    [5] $x := x + 1$

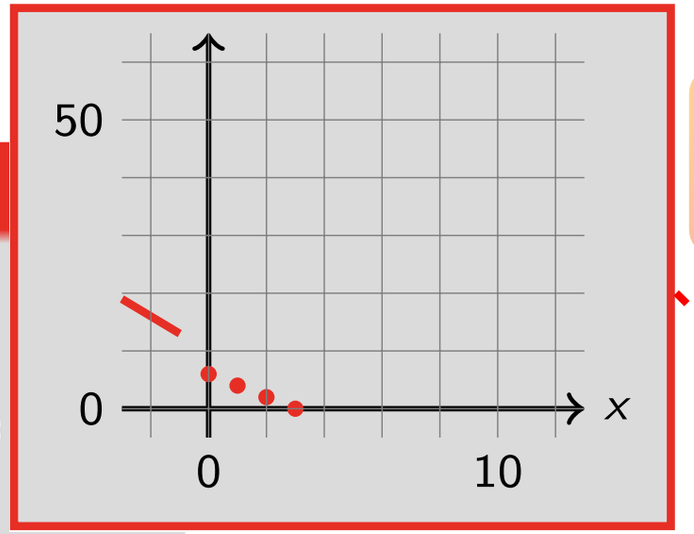
  else

    [6] $x := -x$

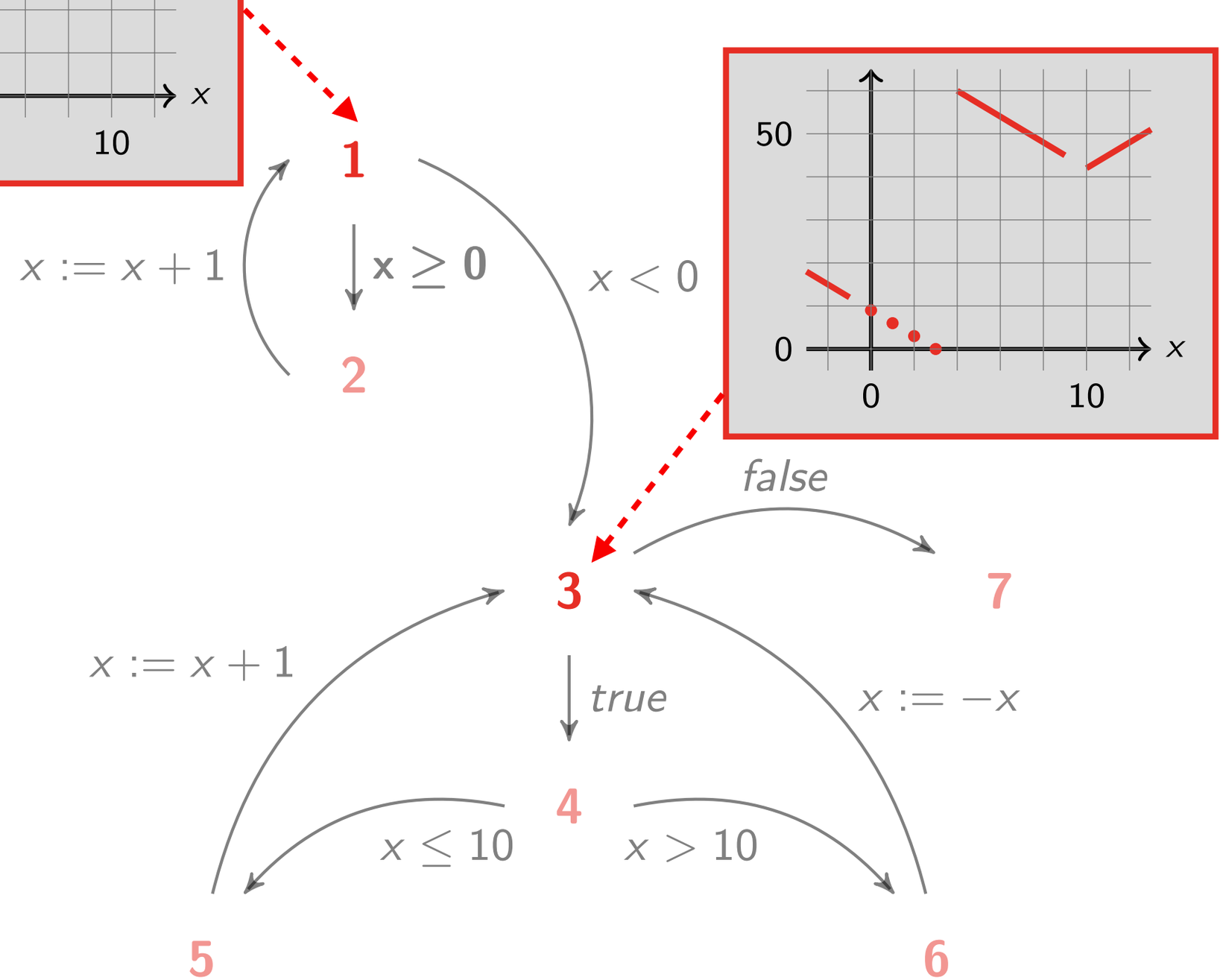
od [7]

Property

$\Diamond\ x = 3$



the analysis gives $x \leq 3$ as **sufficient precondition**



Recurrence Properties

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

$$\mathcal{T}_r^\varphi \in \Sigma \rightharpoonup \mathbb{O}$$

$$\mathcal{T}_r^\varphi \stackrel{\text{def}}{=} \mathbf{gfp}_{\mathcal{T}_g^\varphi} F_r^\varphi$$

$$F_r^\varphi(v)s \stackrel{\text{def}}{=} \begin{cases} v(s) & s \in \mathbf{dom}(\mathcal{T}_g^{\varphi \cap \widetilde{\mathrm{pre}}(\mathrm{dom}(v))}) \\ \mathbf{undefined} & \mathbf{otherwise} \end{cases}$$

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

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

$$\mathcal{T}_r^\varphi \in \Sigma \rightharpoonup \mathbb{O}$$

$$\mathcal{T}_r^\varphi \overset{\text{def}}{=} \mathbf{gfp}_{\mathcal{T}_g^\varphi} F_r^\varphi = \bigcap_{i \in \mathbb{N}} (F_r^\varphi)^i$$

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

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

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

$$\mathcal{T}^{\varphi}_{r} \stackrel{\text{def}}{=} \mathbf{gfp}_{\mathcal{T}^{\varphi}_{g}} F^{\varphi}_{r} = \bigcap_{i \in \mathbb{N}} (F^{\varphi}_{r})^{i}$$

$$(F^{\varphi}_{r})^{0} = \mathcal{T}^{\varphi}_{g}$$

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

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

$$\mathcal{T}_r^\varphi \in \Sigma \rightharpoonup \mathbb{O}$$

$$\mathcal{T}_r^\varphi \stackrel{\text{def}}{=} \mathbf{gfp}_{\mathcal{T}_g^\varphi} F_r^\varphi = \bigcap_{i \in \mathbb{N}} (F_r^\varphi)^i$$

$$(F_r^\varphi)^0 = \mathcal{T}_g^\varphi$$

$$(F_r^\varphi)^1 = \mathcal{T}_g^{\varphi \cap \widetilde{\text{pre}}(\text{dom}(\mathcal{T}_g^\varphi))}$$

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

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

$$\mathcal{T}_r^\varphi \in \Sigma \rightharpoonup \mathbb{O}$$

$$\mathcal{T}_r^\varphi \stackrel{\text{def}}{=} \mathbf{gfp}_{\mathcal{T}_g^\varphi} F_r^\varphi = \bigcap_{i \in \mathbb{N}} (F_r^\varphi)^i$$

$$(F_r^\varphi)^0 = \mathcal{T}_g^\varphi$$

$$(F_r^\varphi)^1 = \mathcal{T}_g^{\varphi \cap \widetilde{\text{pre}}(\text{dom}(\mathcal{T}_g^\varphi))}$$

$$(F_r^\varphi)^2 = \mathcal{T}_g^{\varphi \cap \widetilde{\text{pre}}(\text{dom}(\mathcal{T}_g^{\varphi \cap \widetilde{\text{pre}}(\text{dom}(\mathcal{T}_g^\varphi))}))}$$

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

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

$$\mathcal{T}_r^\varphi \in \Sigma \rightharpoonup \mathbb{O}$$

$$\mathcal{T}_r^\varphi \stackrel{\text{def}}{=} \mathbf{gfp}_{\mathcal{T}_g^\varphi} F_r^\varphi = \bigcap_{i \in \mathbb{N}} (F_r^\varphi)^i$$

$$(F_r^\varphi)^0 = \mathcal{T}_g^\varphi$$

$$(F_r^\varphi)^1 = \mathcal{T}_g^{\varphi \cap \widetilde{\text{pre}}(\text{dom}(\mathcal{T}_g^\varphi))}$$

$$(F_r^\varphi)^2 = \mathcal{T}_g^{\varphi \cap \widetilde{\text{pre}}(\text{dom}(\mathcal{T}_g^{\varphi \cap \widetilde{\text{pre}}(\text{dom}(\mathcal{T}_g^\varphi))}))}$$

$$\vdots$$

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

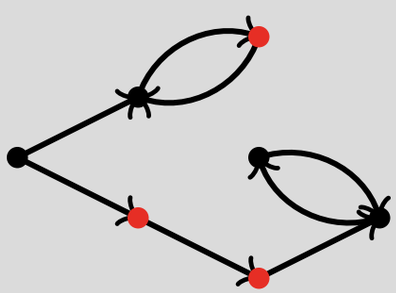
program $\mapsto$ maximal trace semantics $\rightarrow$ **φ-recurrence semantics**

$$\mathcal{T}_r^\varphi \in \Sigma \rightharpoonup \mathbb{O}$$

$$\mathcal{T}_r^\varphi \stackrel{\text{def}}{=} \mathbf{gfp}_{\mathcal{T}_g^\varphi} F_r^\varphi$$

$$F_r^\varphi(v)s \stackrel{\text{def}}{=} \begin{cases} v(s) & s \in \mathbf{dom}(\mathcal{T}_g^{\varphi \cap \widetilde{\mathrm{pre}}(\mathrm{dom}(v))}) \\ \mathbf{undefined} & \mathbf{otherwise} \end{cases}$$

Example



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

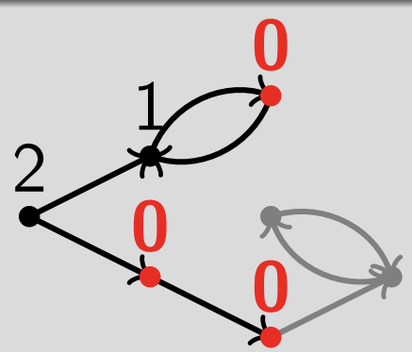
program $\mapsto$ maximal trace semantics $\rightarrow$ **φ-recurrence semantics**

$$\mathcal{T}_r^\varphi \in \Sigma \rightharpoonup \mathbb{O}$$

$$\mathcal{T}_r^\varphi \stackrel{\text{def}}{=} \mathbf{gfp}_{\mathcal{T}_g^\varphi} F_r^\varphi$$

$$F_r^\varphi(v)s \stackrel{\text{def}}{=} \begin{cases} v(s) & s \in \mathbf{dom}(\mathcal{T}_g^{\varphi \cap \widetilde{\mathrm{pre}}(\mathrm{dom}(v))}) \\ \mathbf{undefined} & \mathbf{otherwise} \end{cases}$$

Example



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

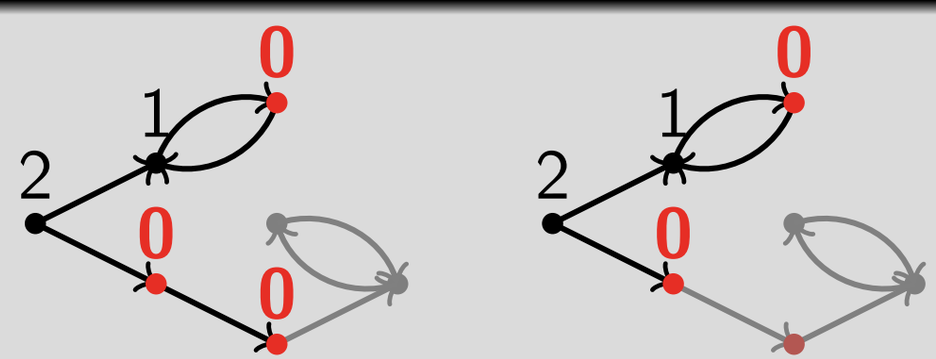
program $\mapsto$ maximal trace semantics $\rightarrow$ **φ-recurrence semantics**

$$\mathcal{T}_r^\varphi \in \Sigma \rightharpoonup \mathbb{O}$$

$$\mathcal{T}_r^\varphi \stackrel{\text{def}}{=} \mathbf{gfp}_{\mathcal{T}_g^\varphi} F_r^\varphi$$

$$F_r^\varphi(v)s \stackrel{\text{def}}{=} \begin{cases} v(s) & s \in \mathbf{dom}(\mathcal{T}_g^{\varphi \cap \widetilde{\mathrm{pre}}(\mathrm{dom}(v))}) \\ \mathbf{undefined} & \mathbf{otherwise} \end{cases}$$

Example



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

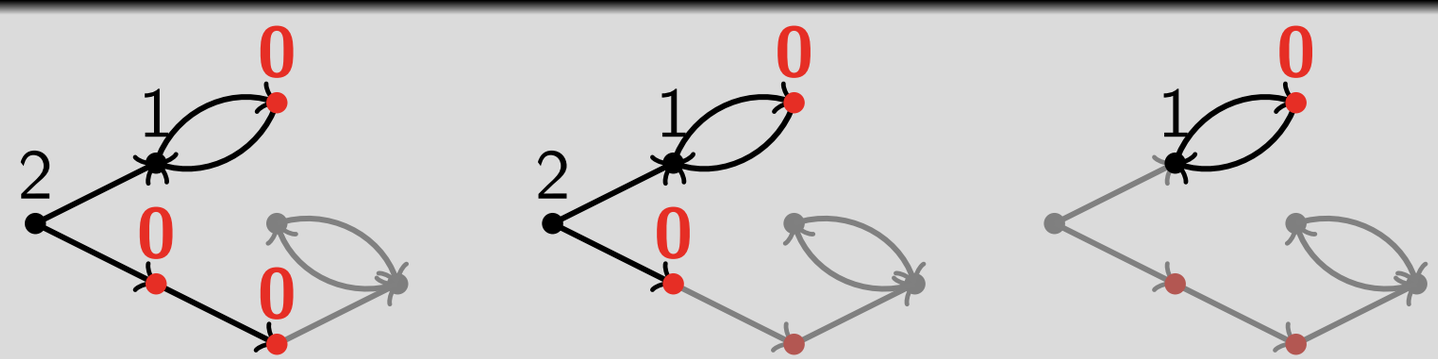
program $\mapsto$ maximal trace semantics $\rightarrow$ **φ-recurrence semantics**

$$\mathcal{T}_r^\varphi \in \Sigma \rightharpoonup \mathbb{O}$$

$$\mathcal{T}_r^\varphi \stackrel{\text{def}}{=} \mathbf{gfp}_{\mathcal{T}_g^\varphi} F_r^\varphi$$

$$F_r^\varphi(v)s \stackrel{\text{def}}{=} \begin{cases} v(s) & s \in \mathbf{dom}(\mathcal{T}_g^{\varphi \cap \widetilde{\mathrm{pre}}(\mathrm{dom}(v))}) \\ \mathbf{undefined} & \mathbf{otherwise} \end{cases}$$

Example



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

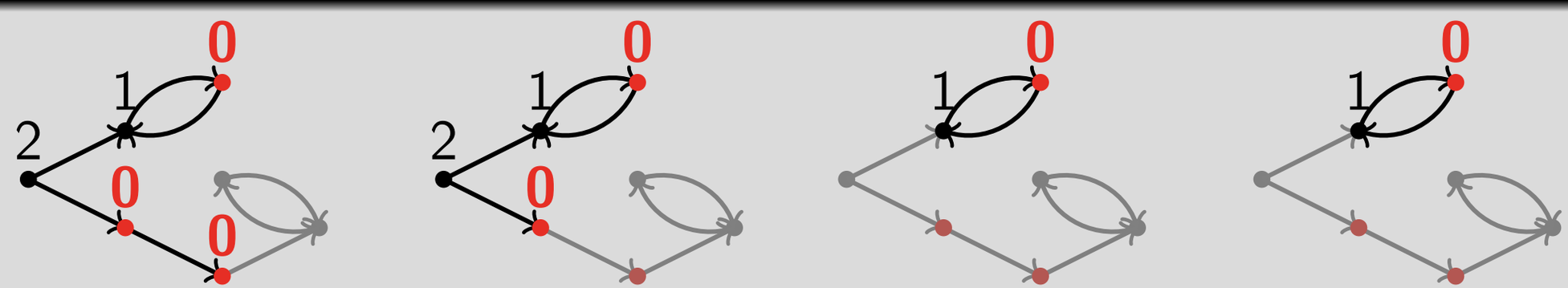
program $\mapsto$ maximal trace semantics $\rightarrow$ **φ-recurrence semantics**

$$\mathcal{T}_r^\varphi \in \Sigma \rightharpoonup \mathbb{O}$$

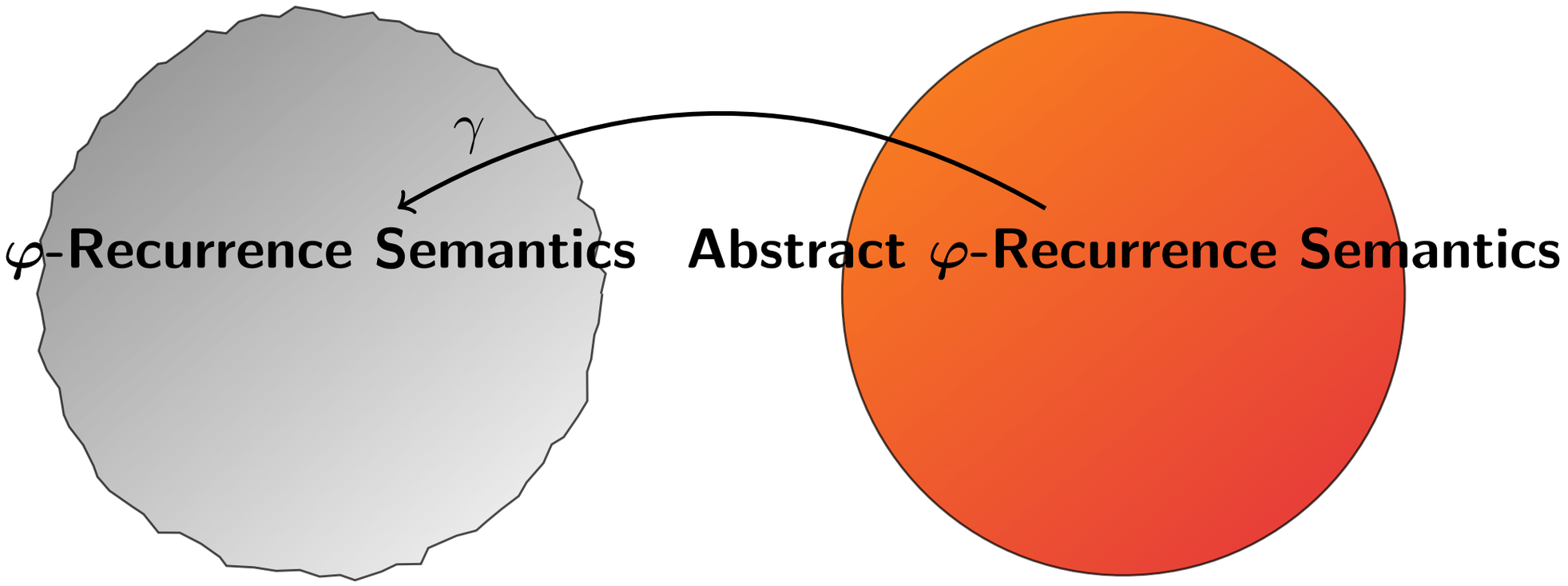
$$\mathcal{T}_r^\varphi \stackrel{\text{def}}{=} \mathbf{gfp}_{\mathcal{T}_g^\varphi} F_r^\varphi$$

$$F_r^\varphi(v)s \stackrel{\text{def}}{=} \begin{cases} v(s) & s \in \mathbf{dom}(\mathcal{T}_g^{\varphi \cap \widetilde{\mathrm{pre}}(\mathrm{dom}(v))}) \\ \mathbf{undefined} & \mathbf{otherwise} \end{cases}$$

Example



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


γ
φ-Recurrence Semantics
Abstract φ-Recurrence Semantics

Example

int : $x,\ y$

while [1]$(x \geq 0)$ do

  [2]$x := x + 1$

od

while [3](true) do

  if [4]($x \leq 10$)

    [5]$x := x + 1$

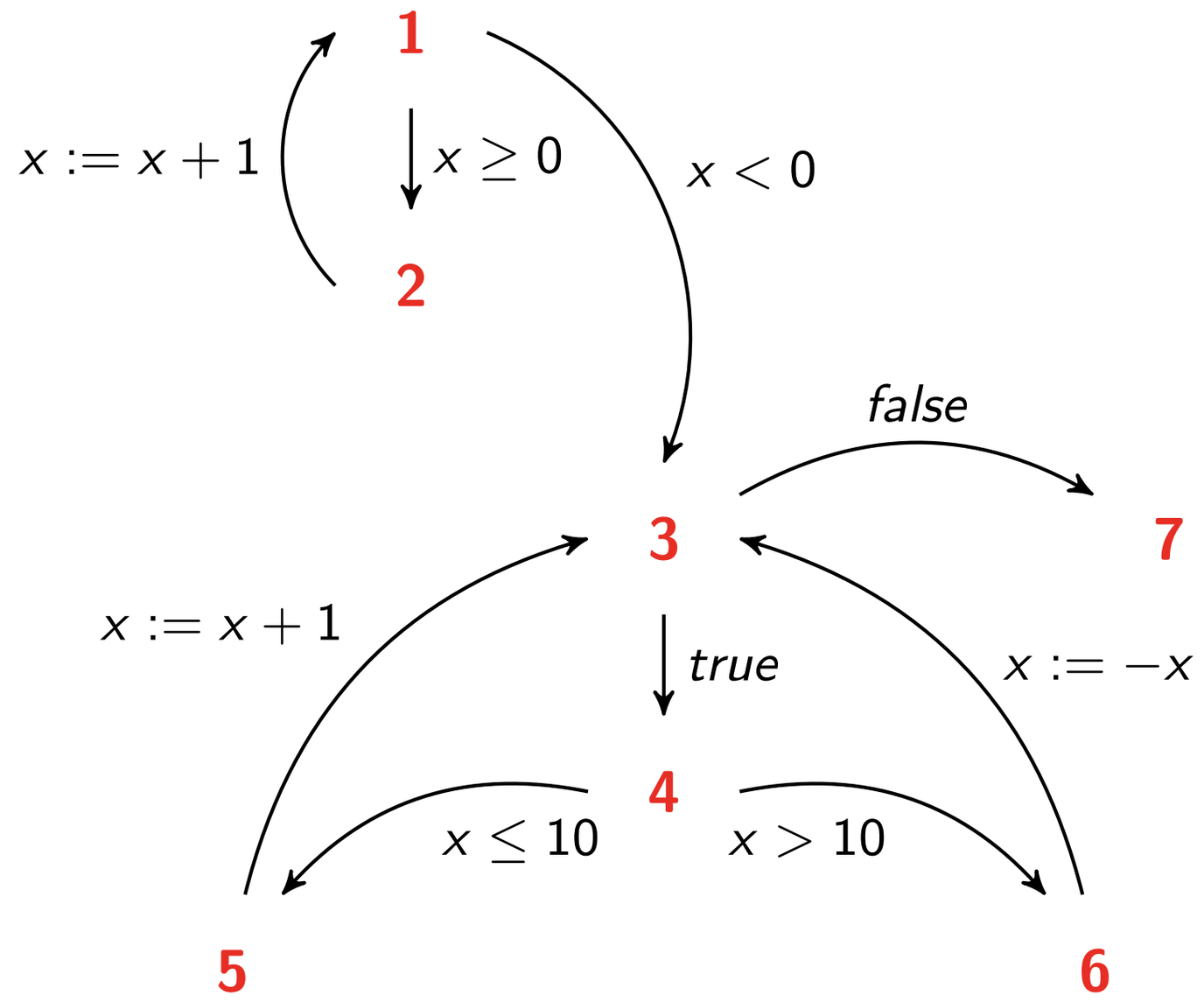
  else

    [6]$x := -x$

od[7]

Property

$\Box\Diamond\ x = 3$



Example

int : $x,\ y$

while [1]$(x \geq 0)$ do

  [2]$x := x + 1$

od

while [3](true) do

  if [4]($x \leq 10$)

    [5]$x := x + 1$

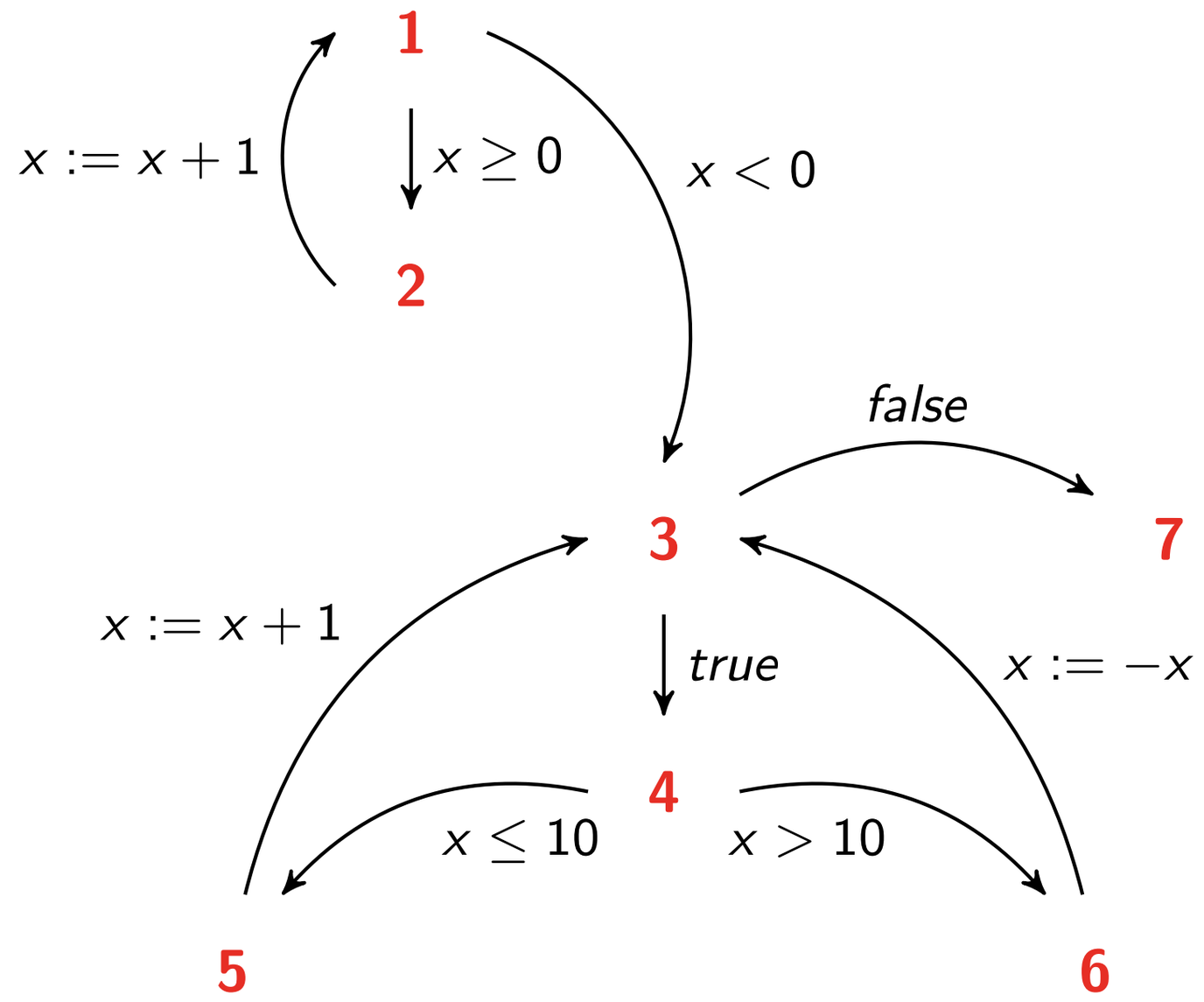
  else

    [6]$x := -x$

od[7]

Property

$\Box\Diamond\ x = 3$



Example

int : x, y

while [1] $(x \geq 0)$ do

  [2] $x := x + 1$

od

while [3] (true) do

  if [4] ($x \leq 10$)

    [5] $x := x + 1$

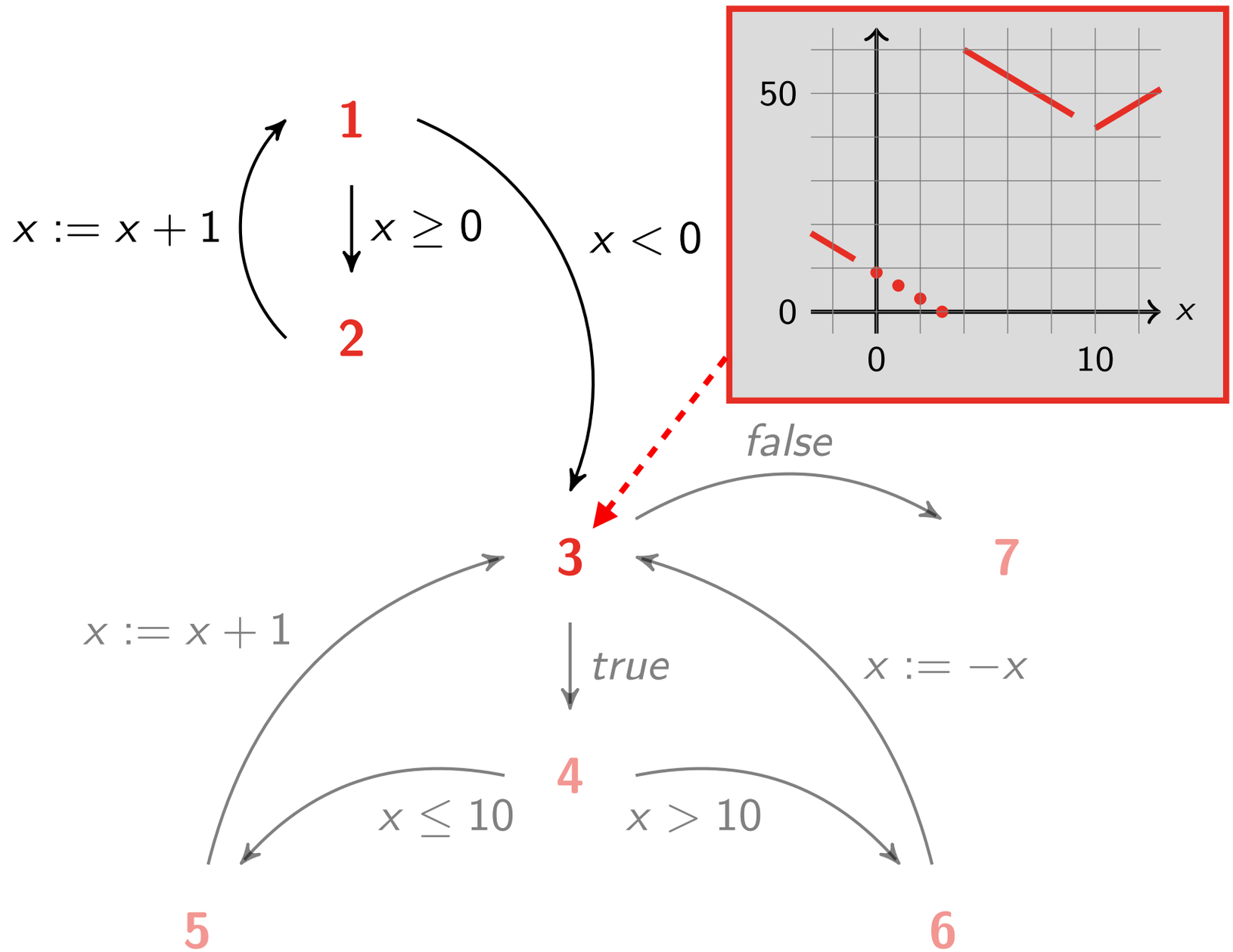
  else

    [6] $x := -x$

od [7]

Property

$\square\Diamond\ x = 3$



Example

int : x, y

[1] while $(x \geq 0)$

  [2] $x := x + 1$

od

[3] while (true) do

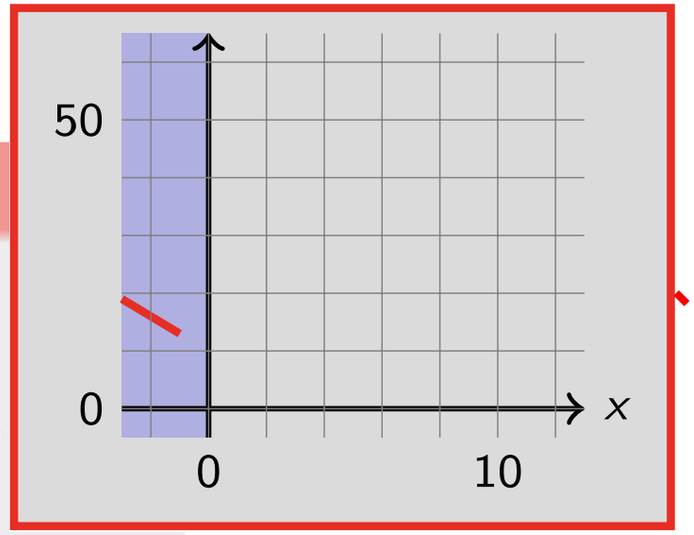
  [4] if $(x \leq 10)$

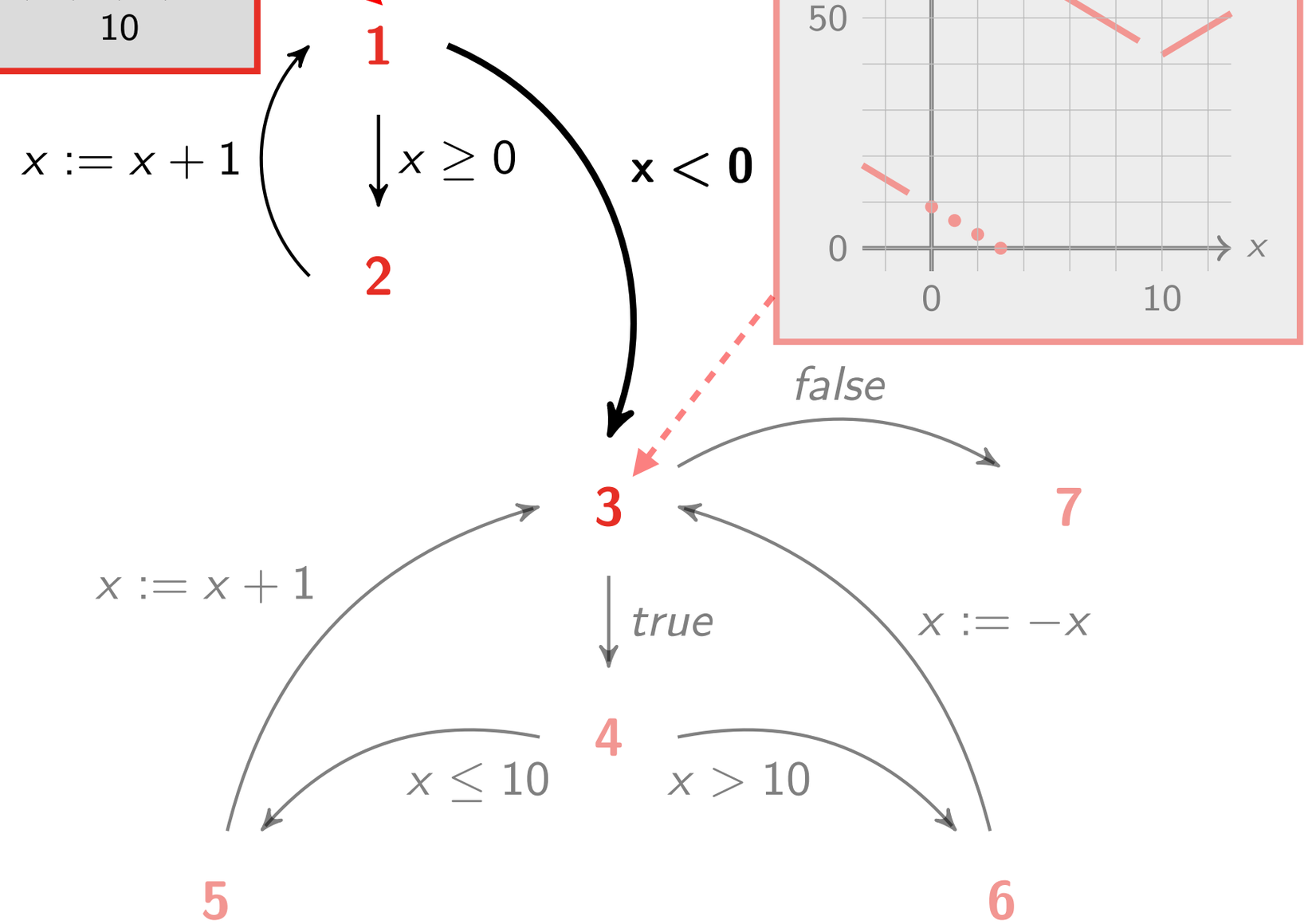
    [5] $x := x + 1$

  else

    [6] $x := -x$

od [7]



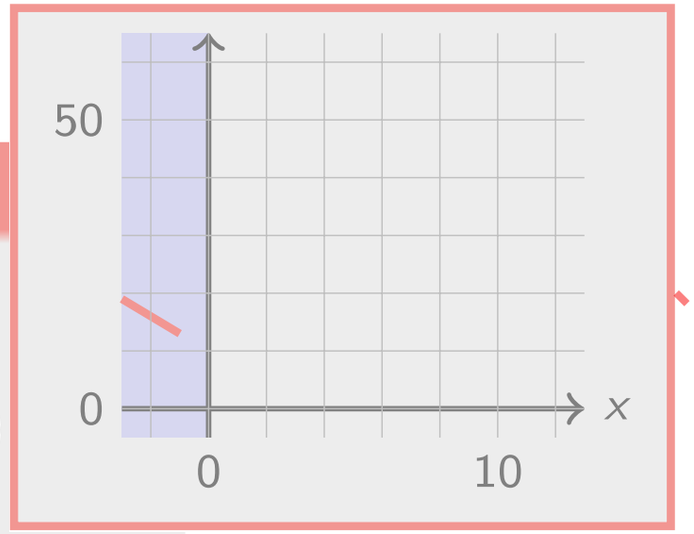


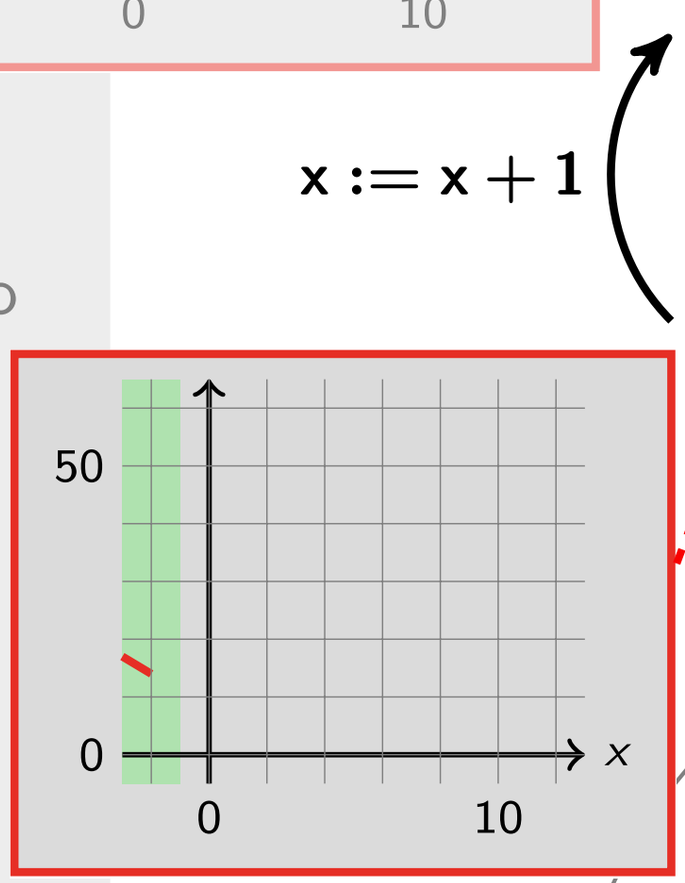
Property

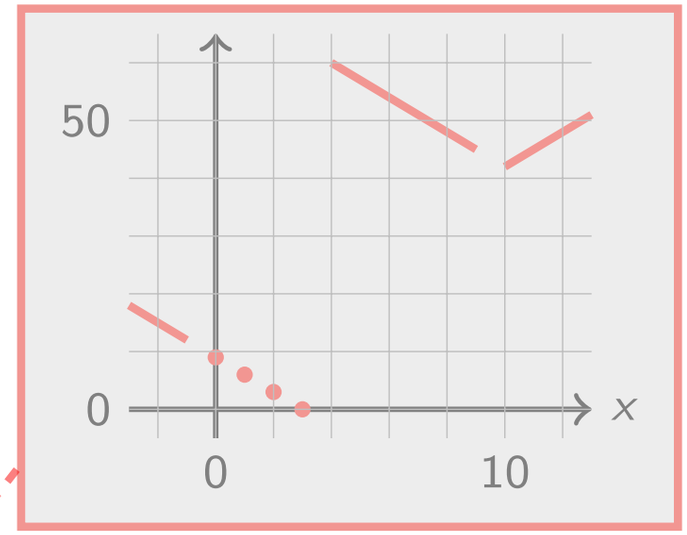
$\square\diamond\ x = 3$

Example

```
int : x, y
while [1](x ≥ 0)
    [2]x := x + 1
od
while [3]( true ) do
    if [4]( x ≤ 10 )
        [5]x := x + 1
    else
        [6]x := −x
od[7]
```







Property

$\Box\Diamond\ x = 3$

Example

int : x, y

[1] while $(x \geq 0)$

  [2] $x := x + 1$

od

[3] while (true) do

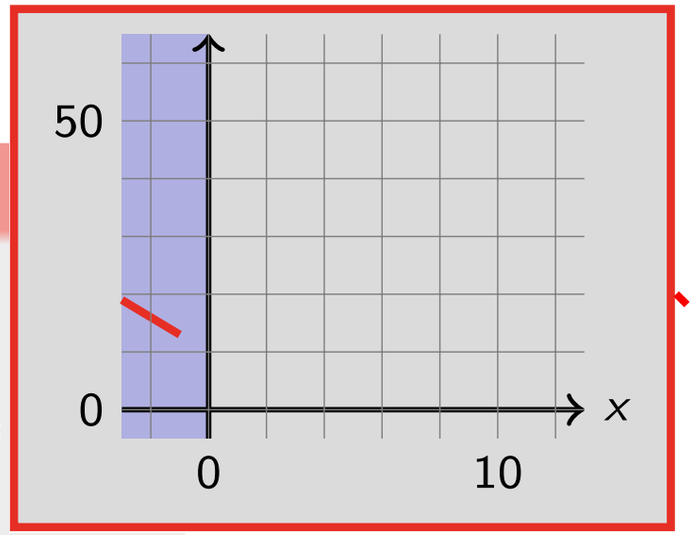
  [4] if $(x \leq 10)$

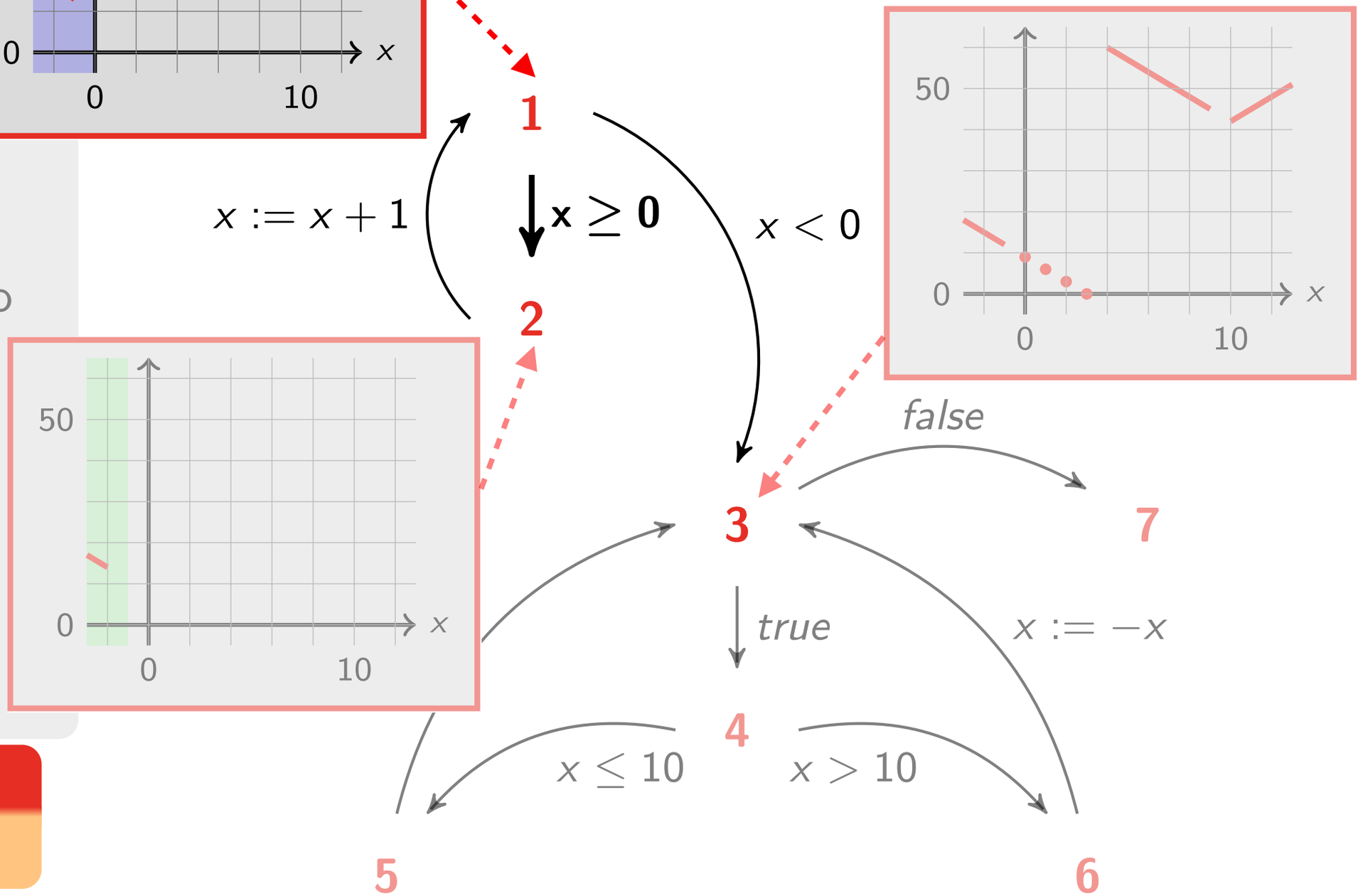
    [5] $x := x + 1$

  else

    [6] $x := -x$

od [7]





Property

$\square\Diamond\ x = 3$

Example

int : x, y

[1] while $(x \geq 0)$

  [2] $x := x + 1$

od

[3] while (true) do

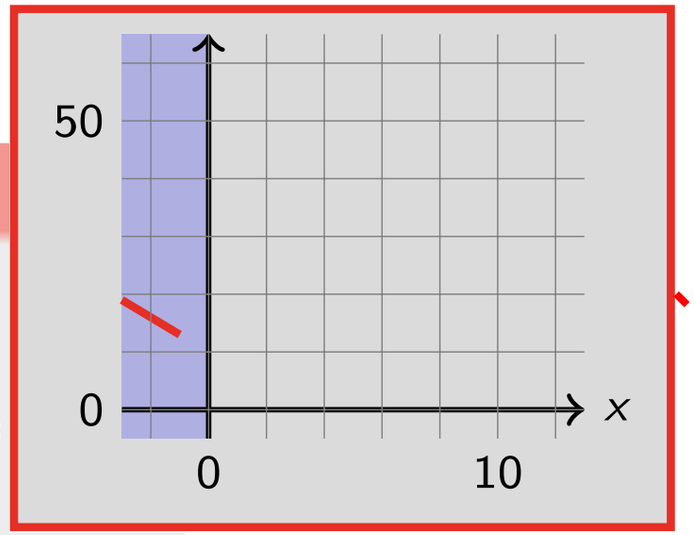
  [4] if $(x \leq 10)$

    [5] $x := x + 1$

  else

    [6] $x := -x$

od [7]



Property

$\square\Diamond\ x = 3$

Example

int : x, y

while [1] $(x \geq 0)$

  [2] $x := x + 1$

od

while [3] (true) do

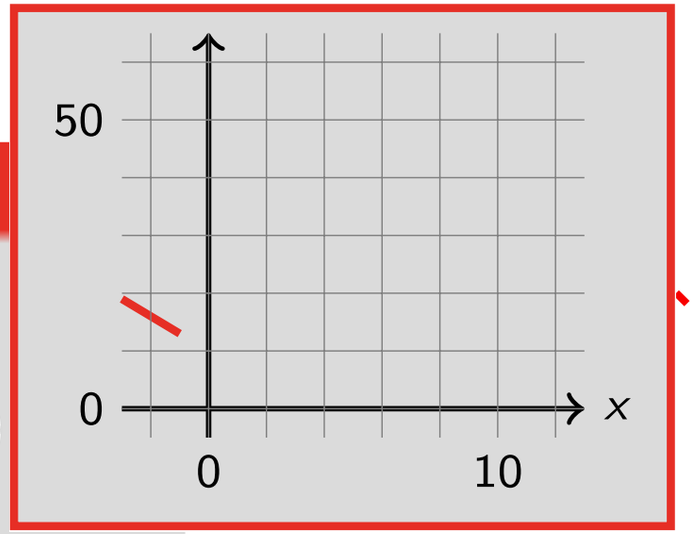
  if [4] ($x \leq 10$)

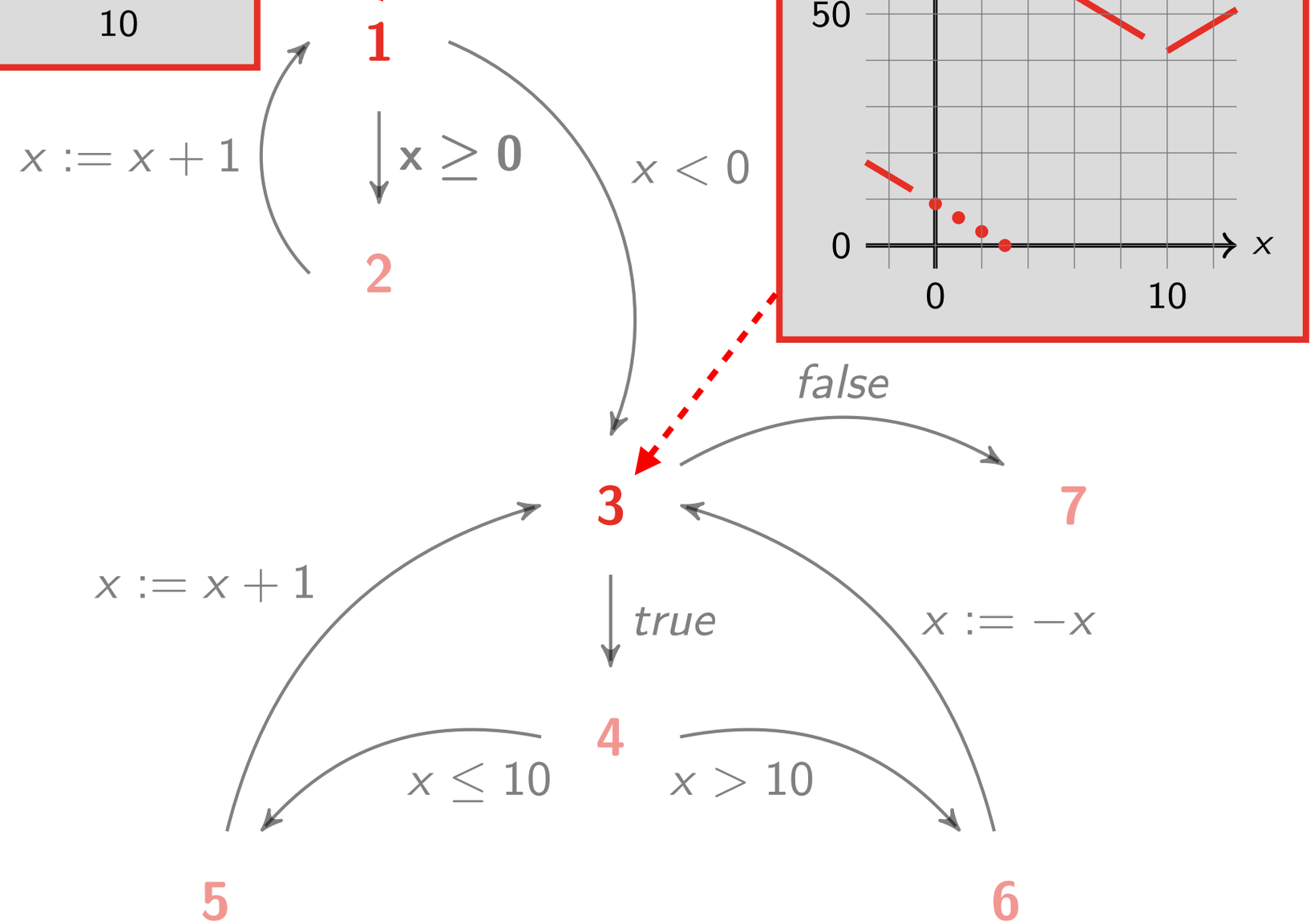
    [5] $x := x + 1$

  else

    [6] $x := -x$

od [7]





Property

$\Box\Diamond\ x = 3$

Example

int : x, y

while [1] $(x \geq 0)$

  [2] $x := x + 1$

od

while [3] (true) do

  if [4] ($x \leq 10$)

    [5] $x := x + 1$

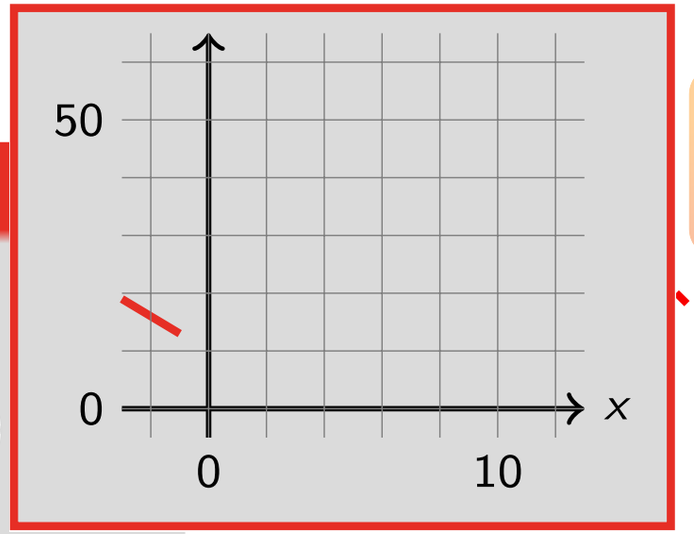
  else

    [6] $x := -x$

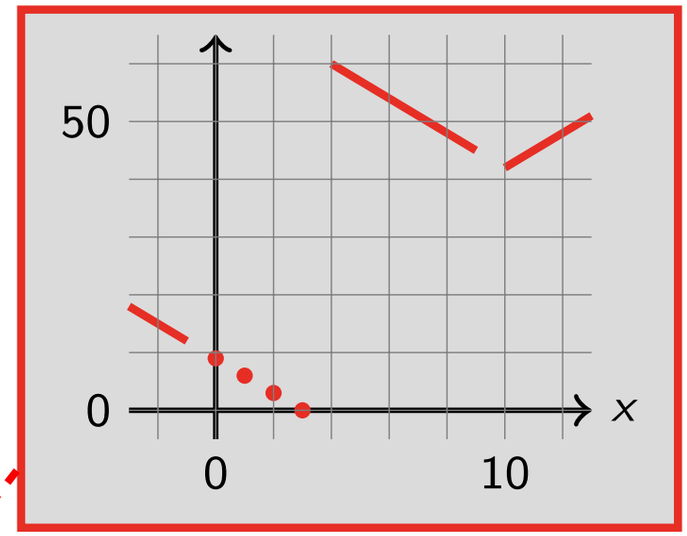
od [7]

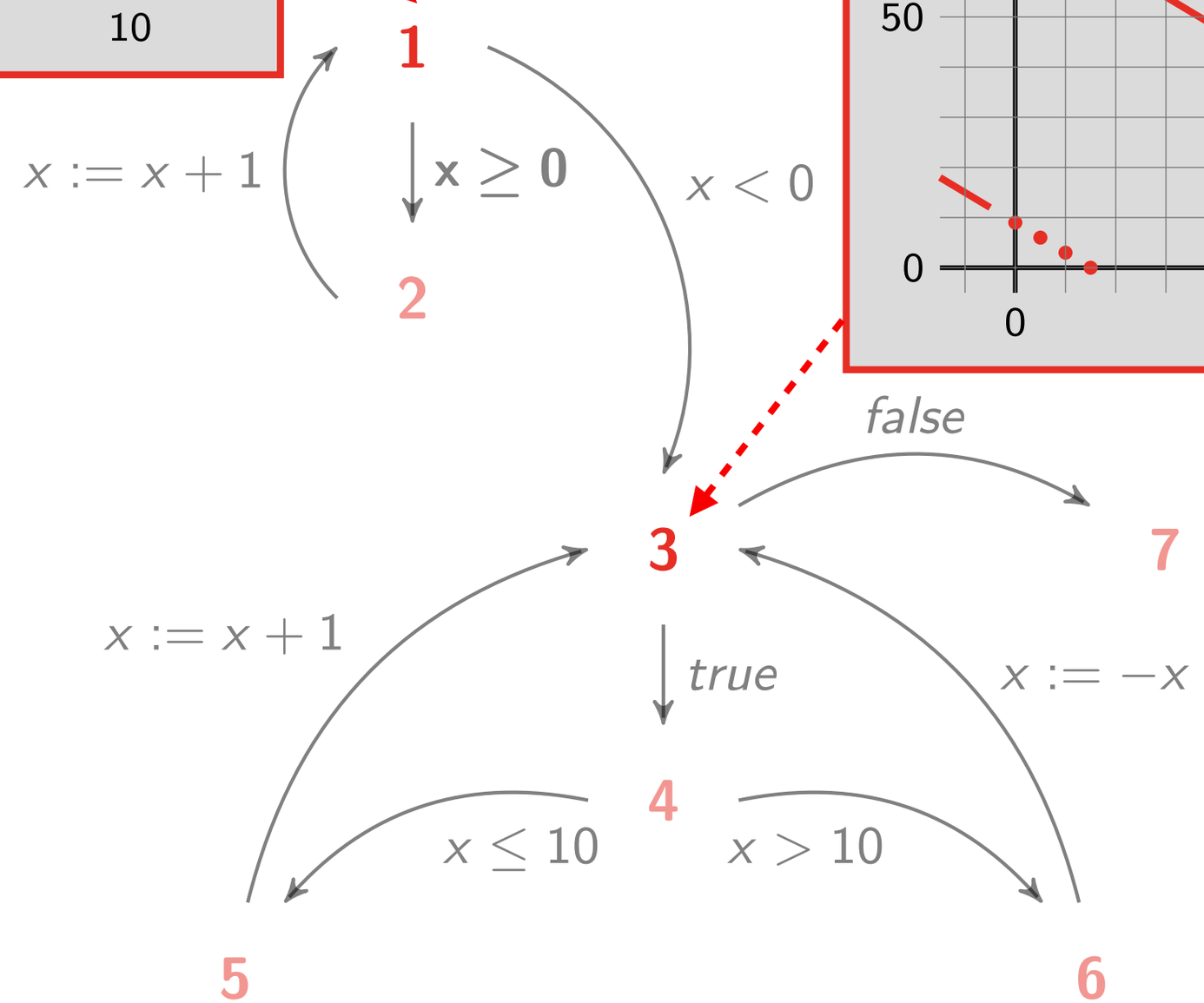
Property

$\Box\Diamond\ x = 3$



the analysis gives $x < 0$ as
sufficient precondition





Peterson's Algorithm

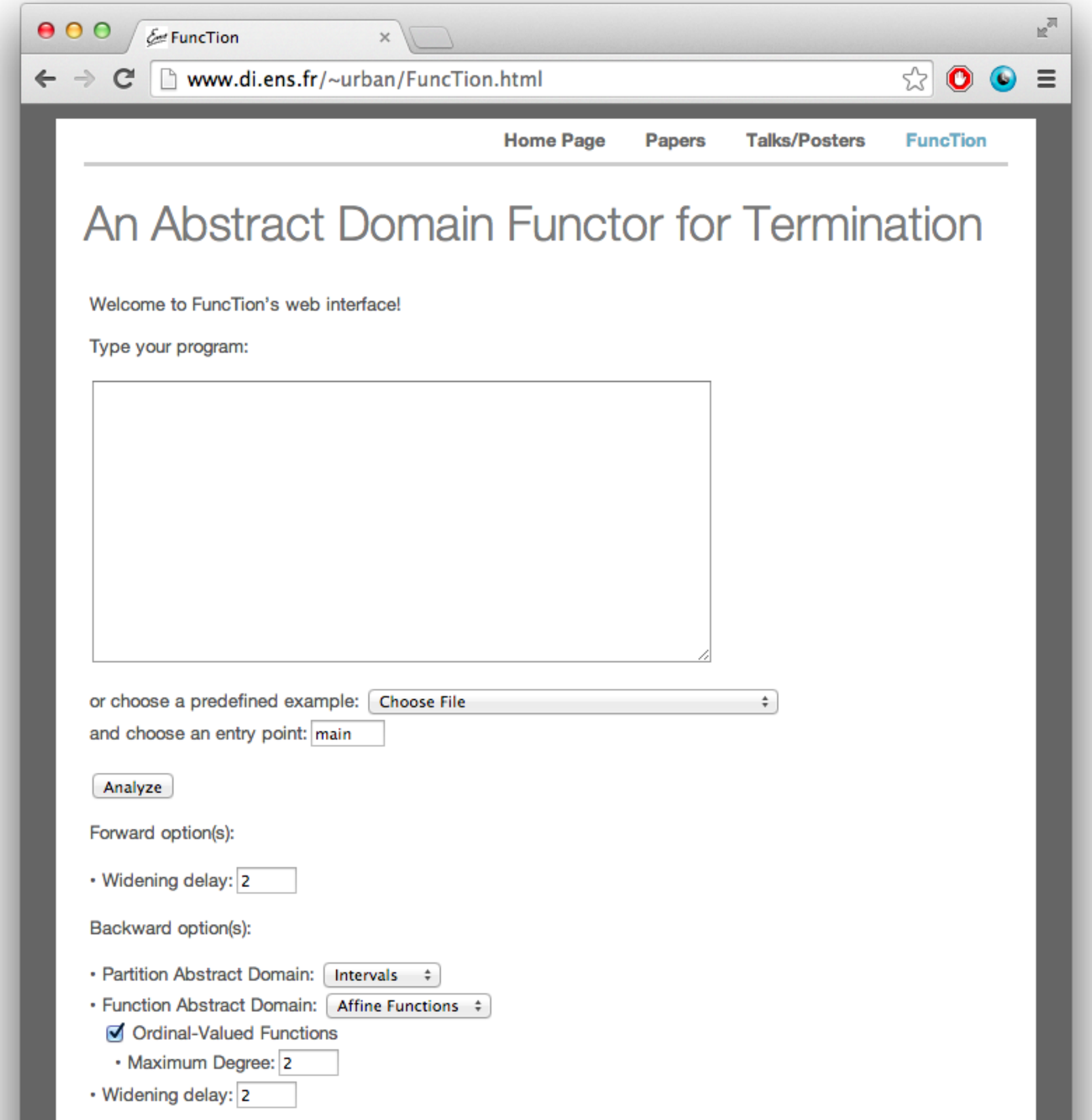
Example

$$flag_1 := 0;\ \ flag_2 := 0;$$

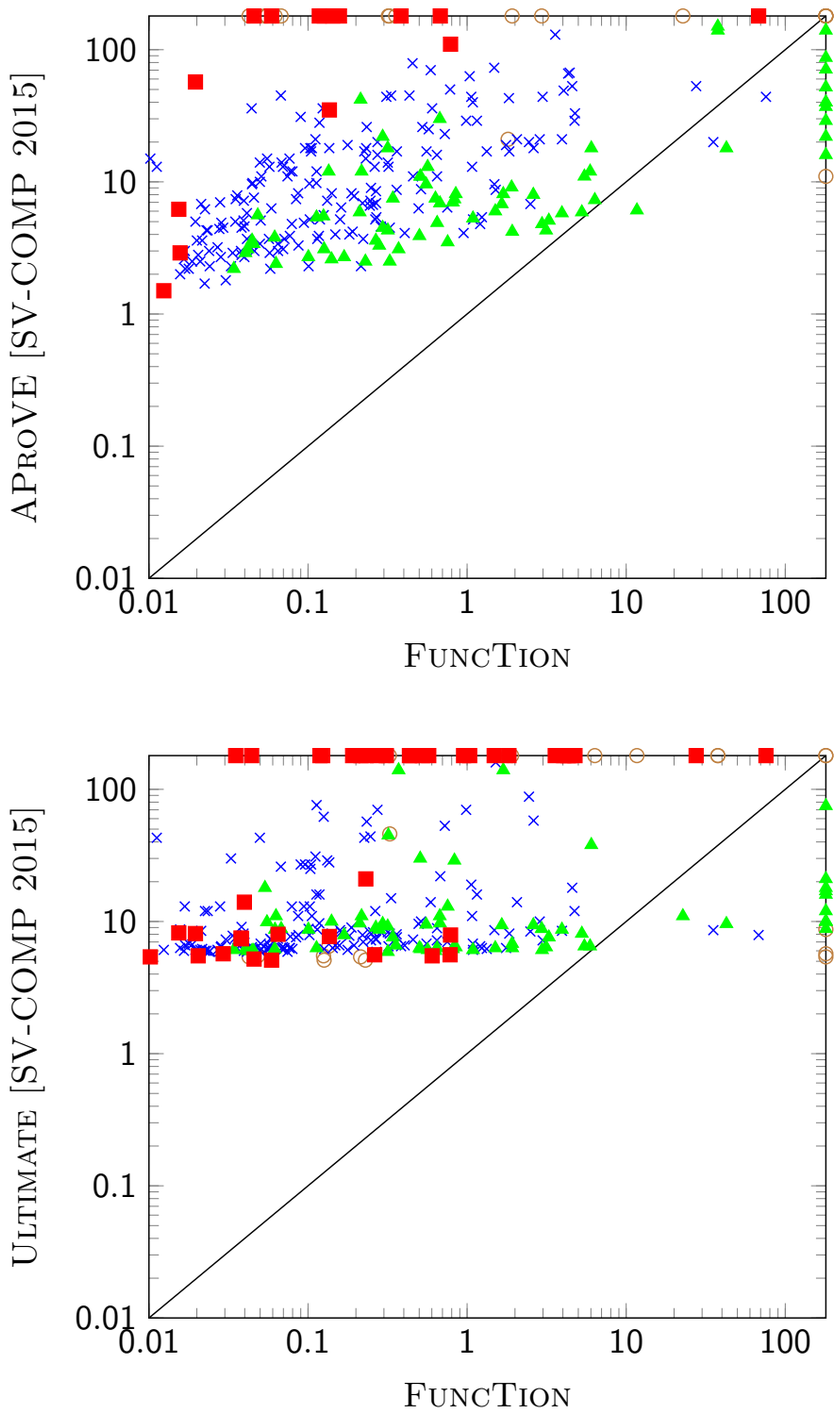
$$\left[\begin{array}{l}\text{while } {}^{1}(\ true\ ) \text{ do}\\ \quad {}^{2}flag_1 := 1\\ \quad {}^{3}turn := 2\\ \quad \text{await } {}^{4}(\ flag_2 = 0 \vee turn = 1\ )\\ \quad {}^{5}\textsc{critical\_section}\\ \quad {}^{6}flag_1 := 0\end{array}\right] \parallel \left[\begin{array}{l}\text{while } {}^{1}(\ true\ ) \text{ do}\\ \quad {}^{2}flag_2 := 1\\ \quad {}^{3}turn := 1\\ \quad \text{await } {}^{4}(\ flag_1 = 0 \vee turn = 2\ )\\ \quad {}^{5}\textsc{critical\_section}\\ \quad {}^{6}flag_2 := 0\end{array}\right]$$

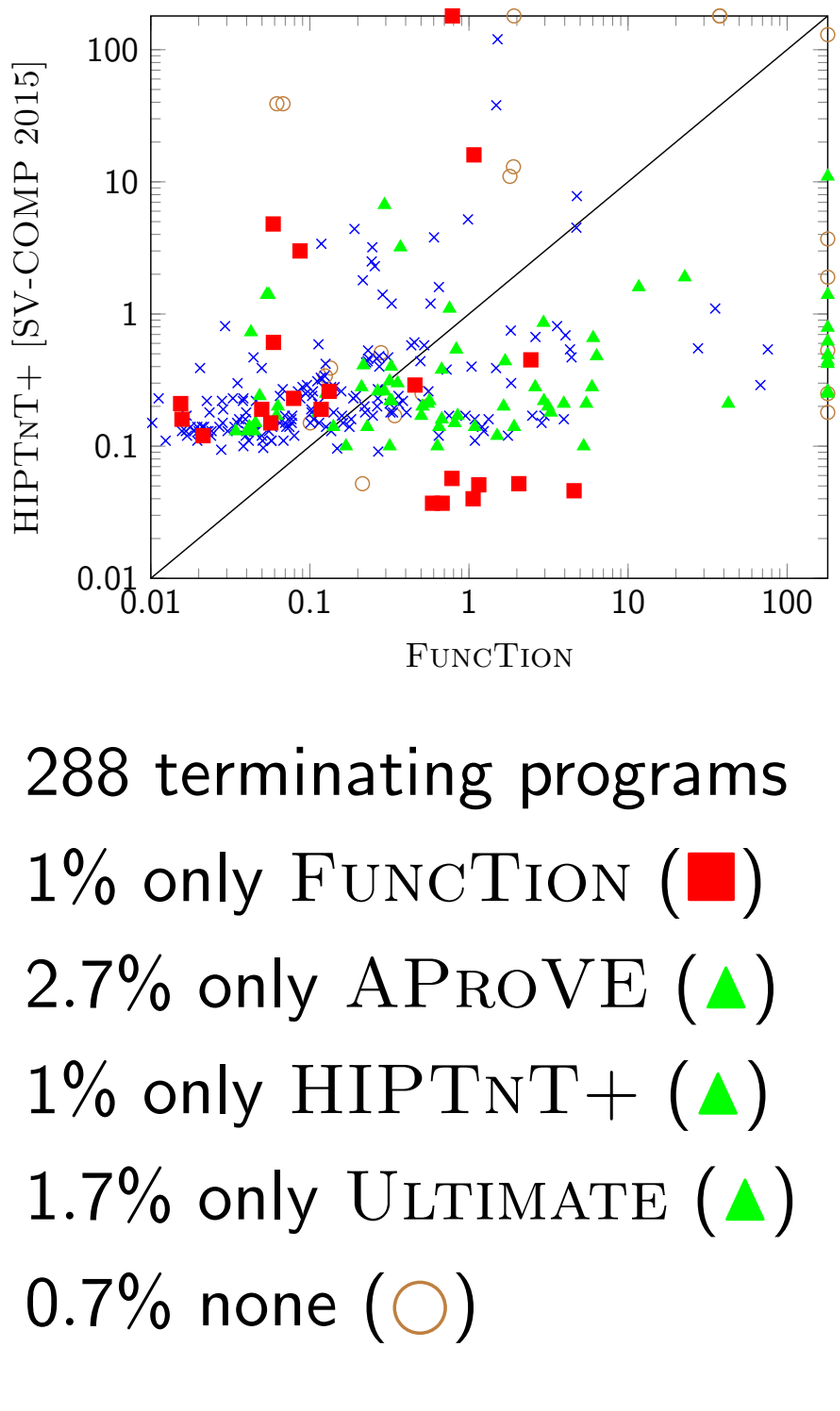
Property

$\square\diamond$ 5 : true


FuncTion
www.di.ens.fr/~urban/FuncTion.html
Home Page
Papers
Talks/Posters
FuncTion
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
Analyze
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

- accepts programs written in a small **subset of C**
- the only basic data type are **mathematical integers**
- implemented in **OCaml**
- abstract domains implemented on top of the **APRON library**
- participating to **SV-COMP 2014** (demo) and **SV-COMP 2015**





- 288 terminating programs
- 1% only FUNCTION (■)
- 2.7% only APROVE (▲)
- 1% only HIPTNT+ (▲)
- 1.7% only ULTIMATE (▲)
- 0.7% none (○)

D'Silva & Urban - *Conflict-Driven Conditional Termination* (CAV 2015)

Conclusions

- family of **abstract domains** for liveness properties
  - piecewise-defined ranking functions
  - sufficient preconditions for liveness properties
- instances based on affine and **ordinal-valued functions**

To Infinity...

Future Work

- **potential** termination and **non-termination**
- more **abstract domains**
  - non-linear ranking functions
  - better widening
  - machine integers and floats
- **fair termination** and other **liveness** properties
- heap-manipulating and **concurrent** programs

...and Beyond!

Thank You!

- **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)
- **D'Silva & Urban** - *Conflict-Driven Conditional Termination* (CAV 2015)

THE BEST THESIS DEFENSE IS A GOOD THESIS OFFENSE.