A Decision Tree Abstract Domain for Proving Conditional Termination

Caterina Urban and **Antoine Miné**

École Normale Supérieure & CNRS & INRIA
Paris, France

SAS 2014
Munich, Germany

Outline

- **ranking functions**[1]
  - functions that strictly decrease at each program step...
  - ...and that are bounded from below
- **idea**: computation of ranking functions by abstract interpretation[2]
- family of **abstract domains** for program termination[3]
  - piecewise-defined ranking functions
  - backward analysis
  - sufficient conditions for termination
- instances based on **decision trees**

[1] Floyd - *Assigning Meanings to Programs* (1967)
[2] Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)
[3] Urban - *The Abstract Domain of Segmented Ranking Functions* (SAS 2013)

Outline

- **ranking functions**[1]
  - functions that strictly decrease at each program step...
  - ...and that are bounded from below
- **idea**: computation of ranking functions by abstract interpretation[2]
- family of **abstract domains** for program termination[3]
  - piecewise-defined ranking functions
  - backward analysis
  - sufficient conditions for termination
- instances based on **decision trees**

[1] Floyd - *Assigning Meanings to Programs* (1967)
[2] Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)
[3] Urban - *The Abstract Domain of Segmented Ranking Functions* (SAS 2013)

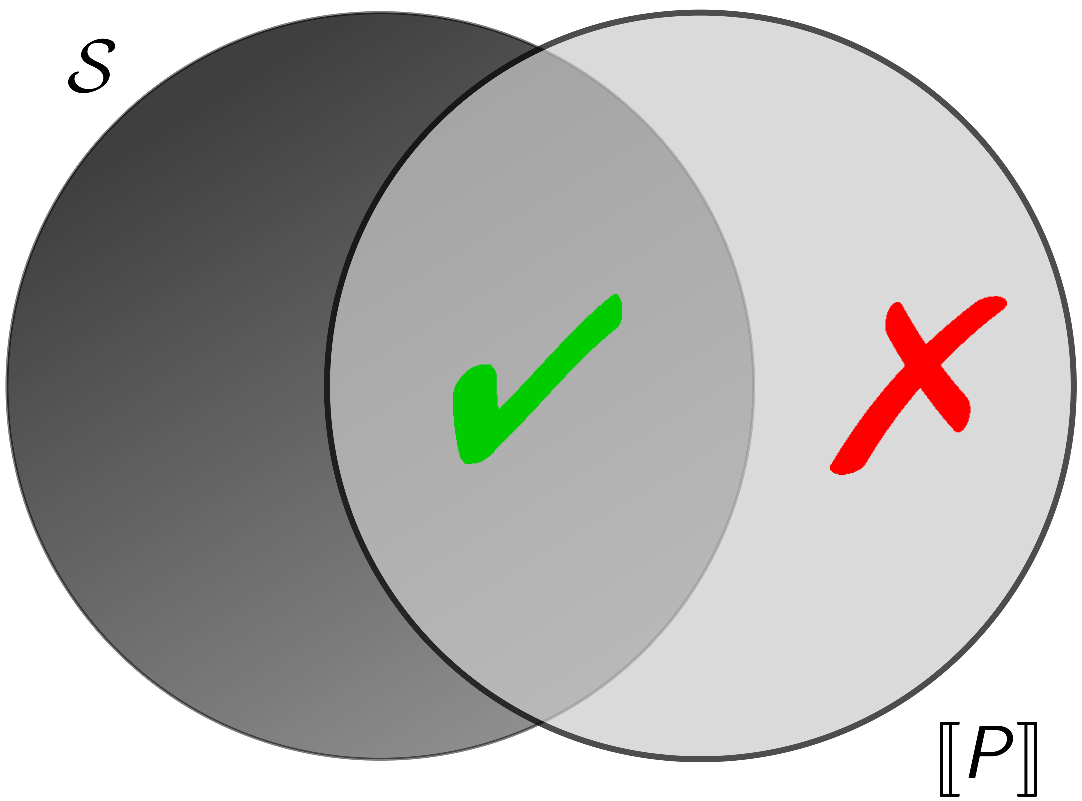
Outline

- **ranking functions**[1]
  - functions that strictly decrease at each program step...
  - ...and that are bounded from below
- **idea**: computation of ranking functions by abstract interpretation[2]
- family of **abstract domains** for program termination[3]
  - piecewise-defined ranking functions
  - backward analysis
  - sufficient conditions for termination
- instances based on **decision trees**

[1] Floyd - *Assigning Meanings to Programs* (1967)
[2] Cousot&Cousot - *An Abstract Interpretation Framework for Termination* (POPL 2012)
[3] Urban - *The Abstract Domain of Segmented Ranking Functions* (SAS 2013)

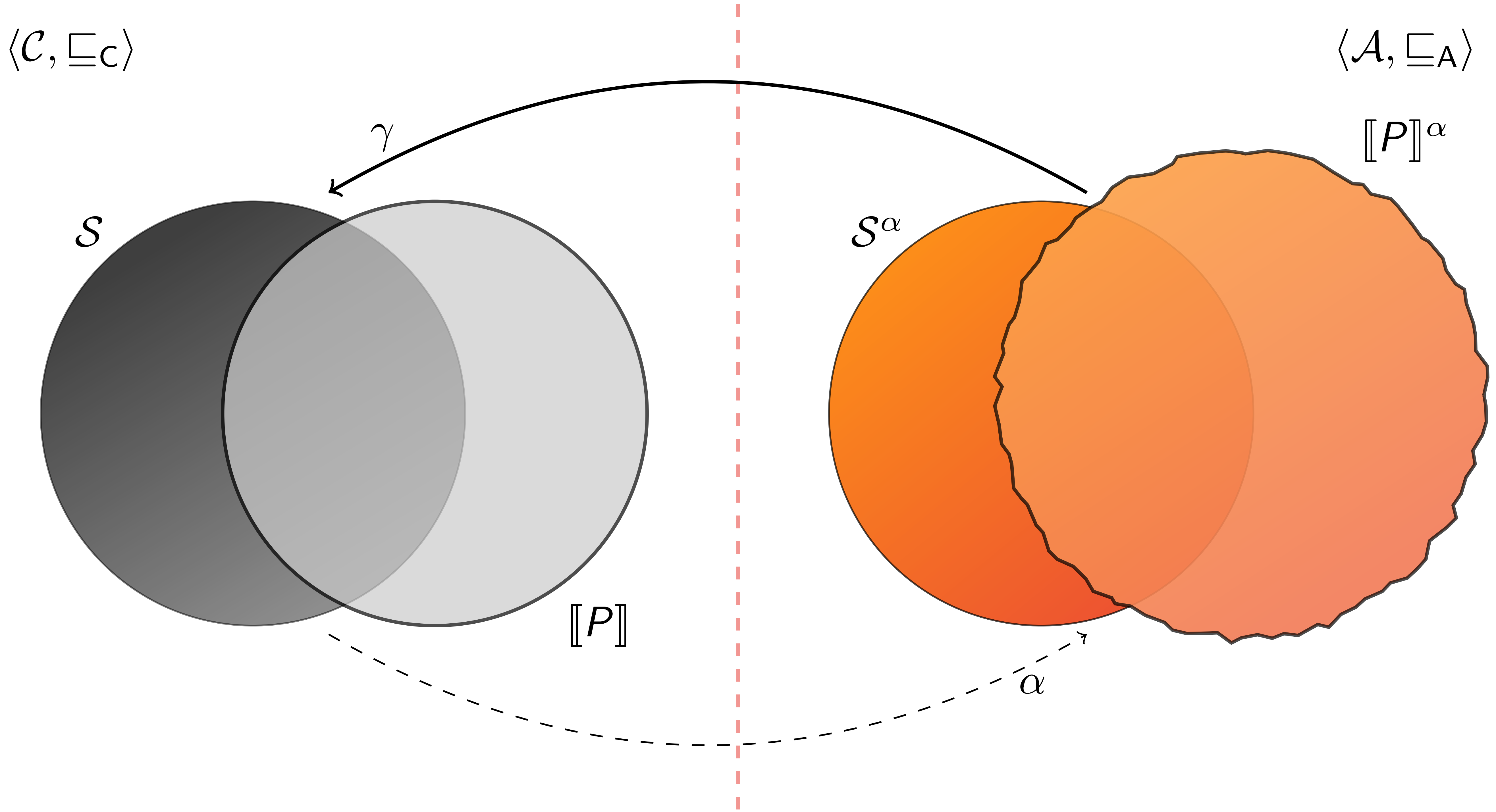
Abstract Interpretation[4]

$\langle \mathcal{C}, \sqsubseteq_\mathsf{C} \rangle$



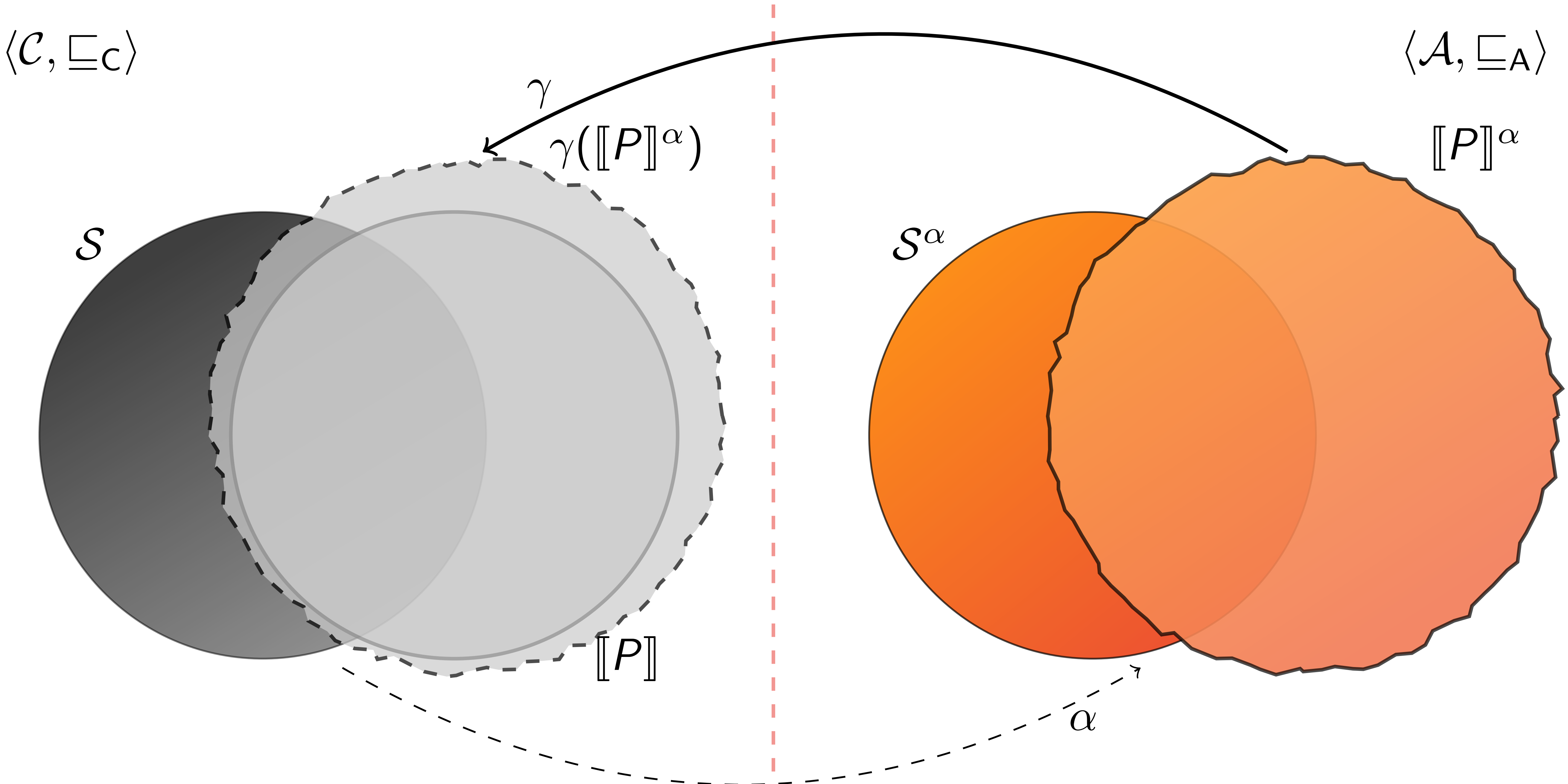
[4] Cousot&Cousot - *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints.* (POPL 1977)

Abstract Interpretation[4]

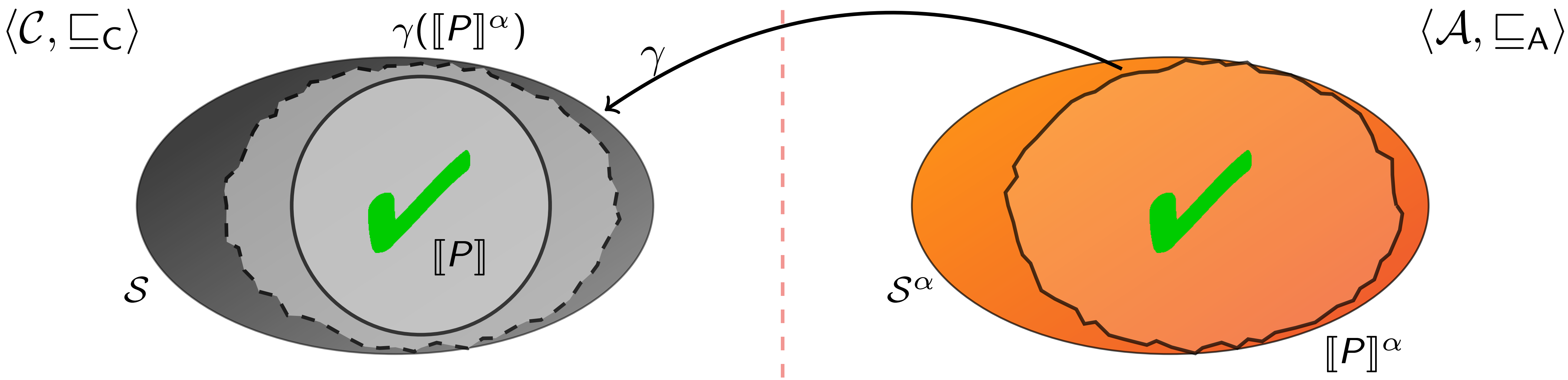


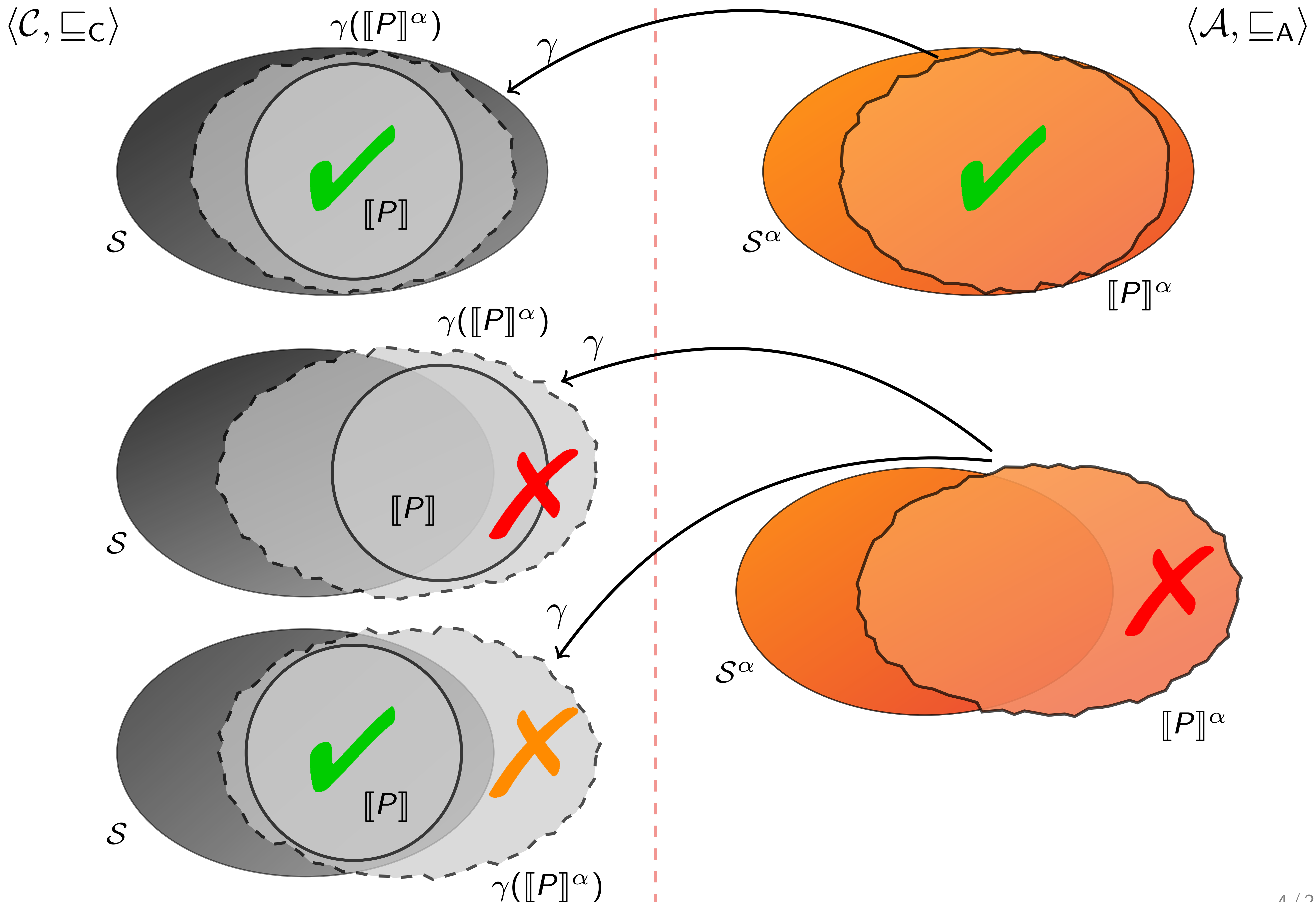
[4] Cousot&Cousot - *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints.* (POPL 1977)

Abstract Interpretation[4]



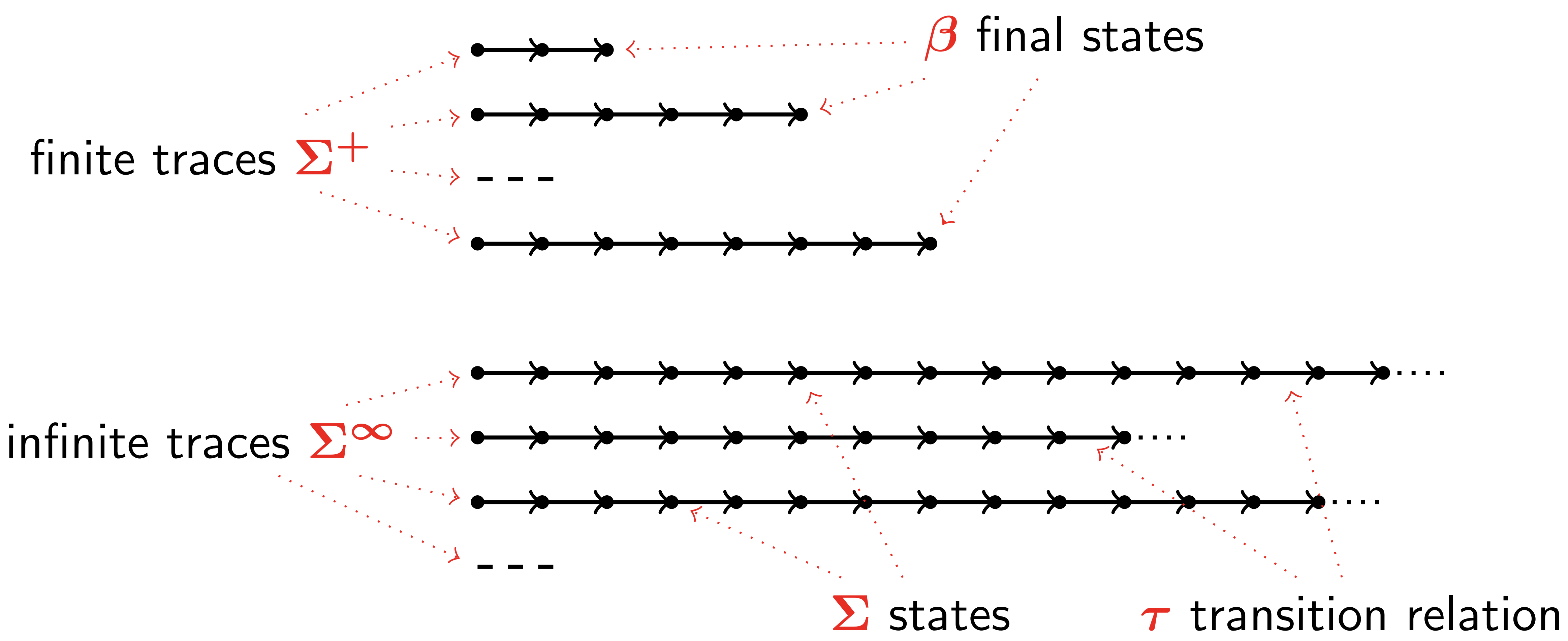
[4] Cousot&Cousot - *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints.* (POPL 1977)


$\langle \mathcal{C}, \sqsubseteq_C \rangle$
$\gamma(\llbracket P \rrbracket^\alpha)$
γ
$\langle \mathcal{A}, \sqsubseteq_A \rangle$
$\llbracket P \rrbracket$
$\mathcal{S}$
$\mathcal{S}^\alpha$
$\llbracket P \rrbracket^\alpha$


$\langle \mathcal{C}, \sqsubseteq_C \rangle$
$\langle \mathcal{A}, \sqsubseteq_A \rangle$
$\gamma(\llbracket P \rrbracket^\alpha)$
γ
$\llbracket P \rrbracket$
$\mathcal{S}$
$\mathcal{S}^\alpha$
$\llbracket P \rrbracket^\alpha$
$\gamma(\llbracket P \rrbracket^\alpha)$
γ
$\llbracket P \rrbracket$
$\mathcal{S}$
γ
$\llbracket P \rrbracket$
$\mathcal{S}$
$\gamma(\llbracket P \rrbracket^\alpha)$
$\mathcal{S}^\alpha$
$\llbracket P \rrbracket^\alpha$

Termination Semantics

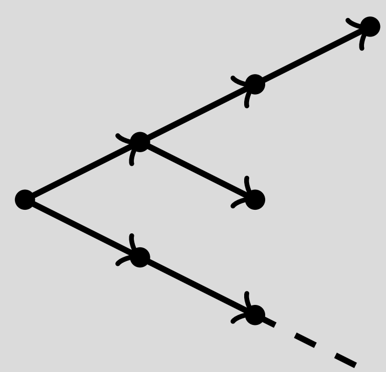
program $\mapsto$ **trace semantics**



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

idea = define a ranking function **counting the number of program steps** from the end of the program and **extracting the well-founded part** of the program transition relation

Example



Theorem (Soundness and Completeness)

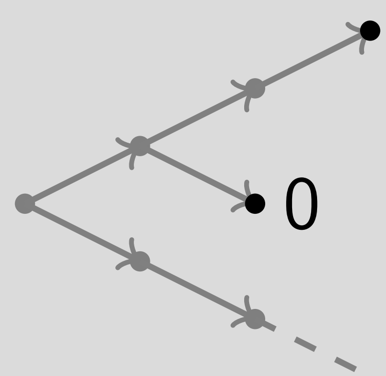
the termination semantics is **sound** *and* **complete** *to prove the termination of programs*

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

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

idea = define a ranking function **counting the number of program steps** from the end of the program and **extracting the well-founded part** of the program transition relation

Example



Theorem (Soundness and Completeness)

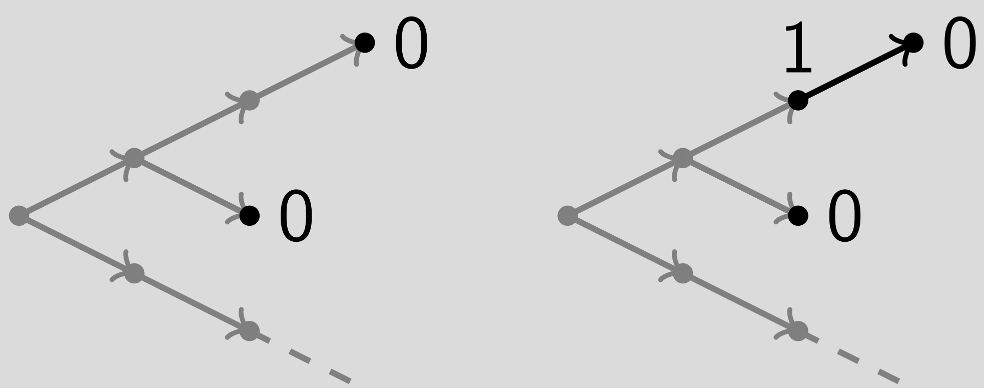
the termination semantics is **sound** *and* **complete** *to prove the termination of programs*

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

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

idea = define a ranking function **counting the number of program steps** from the end of the program and **extracting the well-founded part** of the program transition relation

Example



Theorem (Soundness and Completeness)

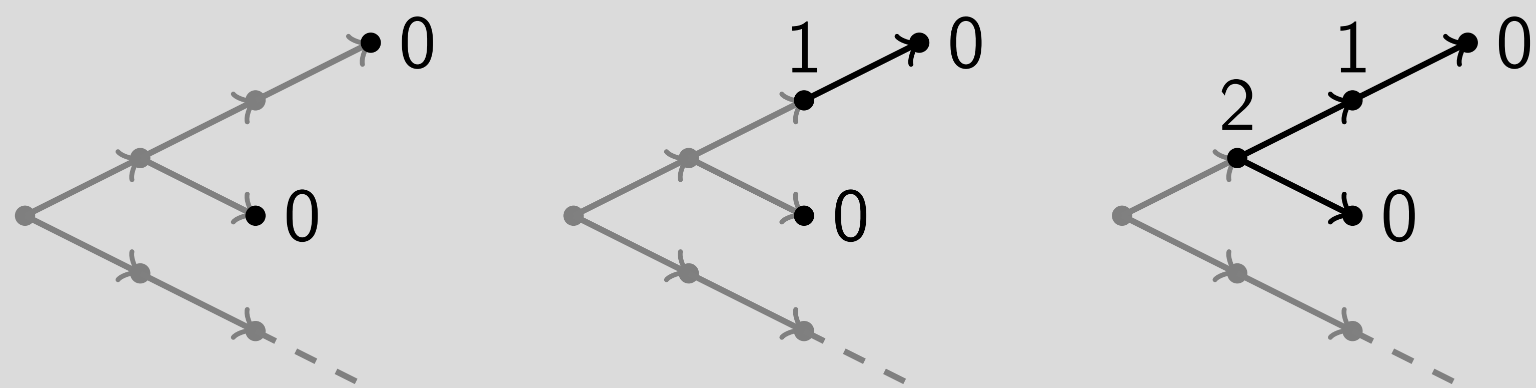
the termination semantics is **sound** *and* **complete** *to prove the termination of programs*

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

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

idea = define a ranking function **counting the number of program steps** from the end of the program and **extracting the well-founded part** of the program transition relation

Example



Theorem (Soundness and Completeness)

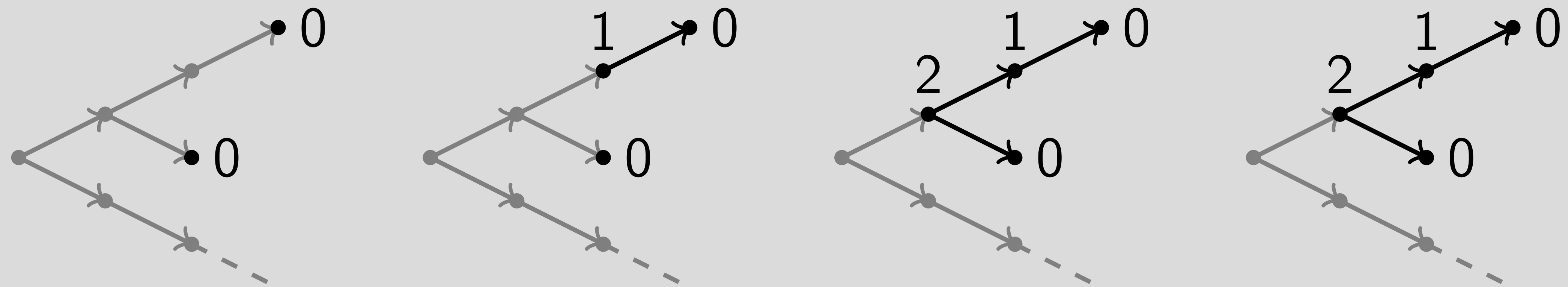
the termination semantics is **sound** *and* **complete** *to prove the termination of programs*

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

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

idea = define a ranking function **counting the number of program steps** from the end of the program and **extracting the well-founded part** of the program transition relation

Example



Theorem (Soundness and Completeness)

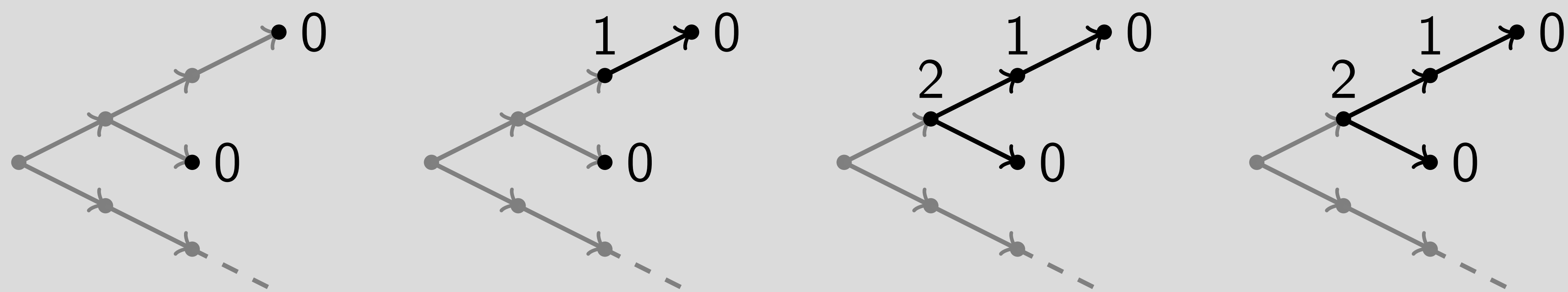
the termination semantics is **sound** *and* **complete** *to prove the termination of programs*

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

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

idea = define a ranking function **counting the number of program steps** from the end of the program and **extracting the well-founded part** of the program transition relation

Example



Theorem (Soundness and Completeness)

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

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

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

Example

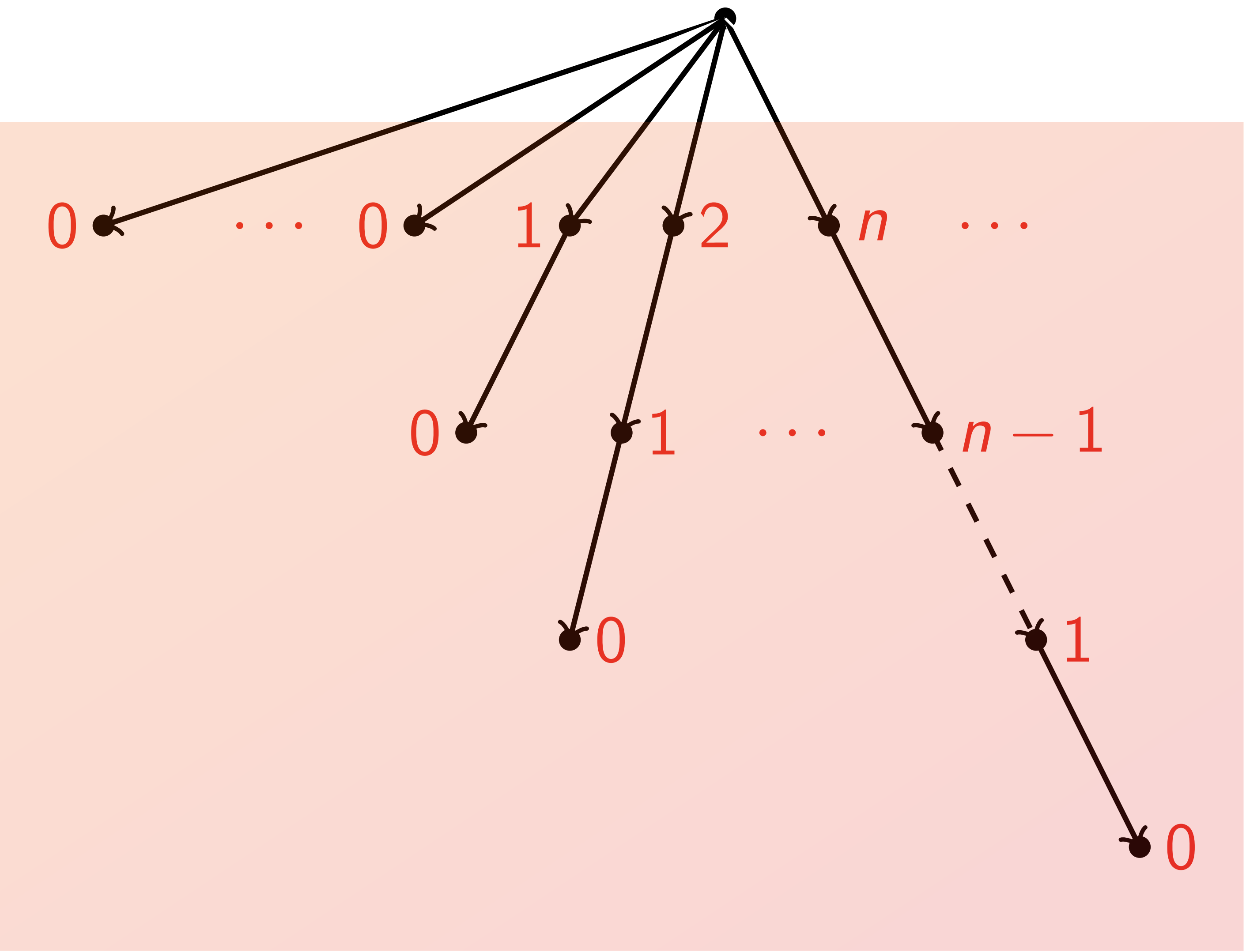
int : x

$x :=\ ?$

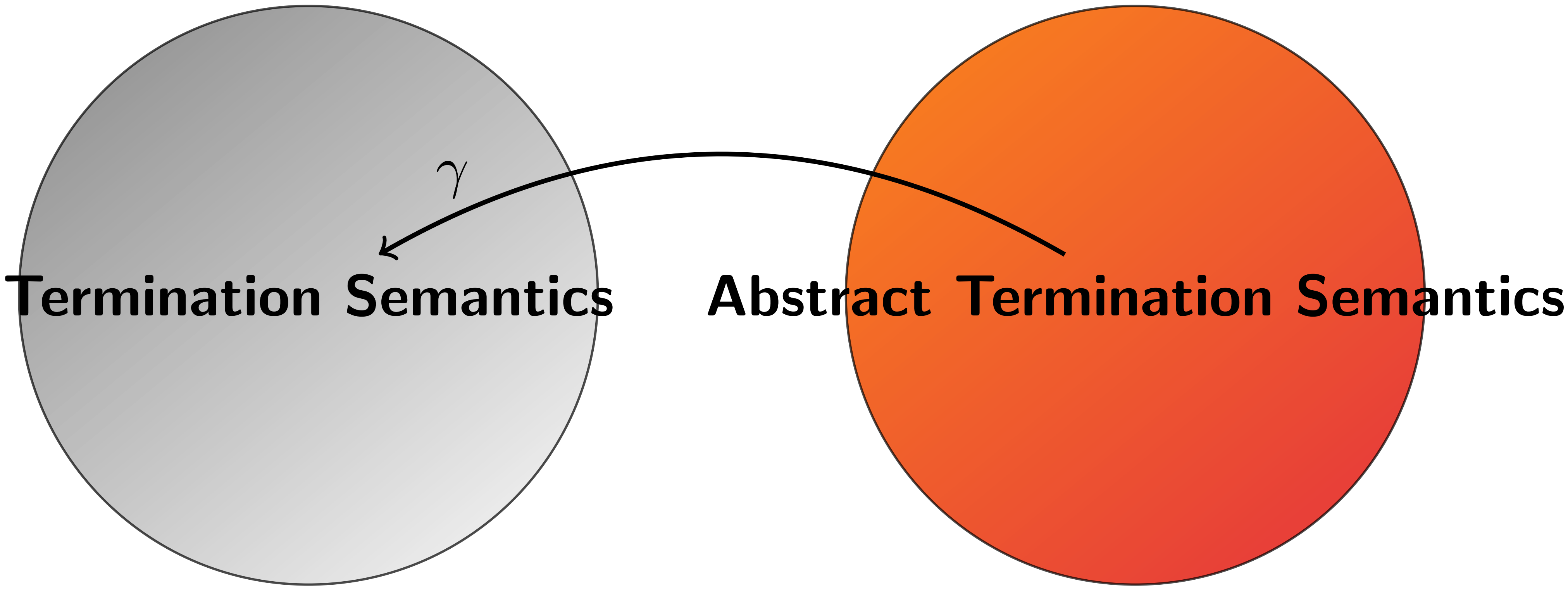
while $(x > 0)$ do

  $x := x - 1$

od

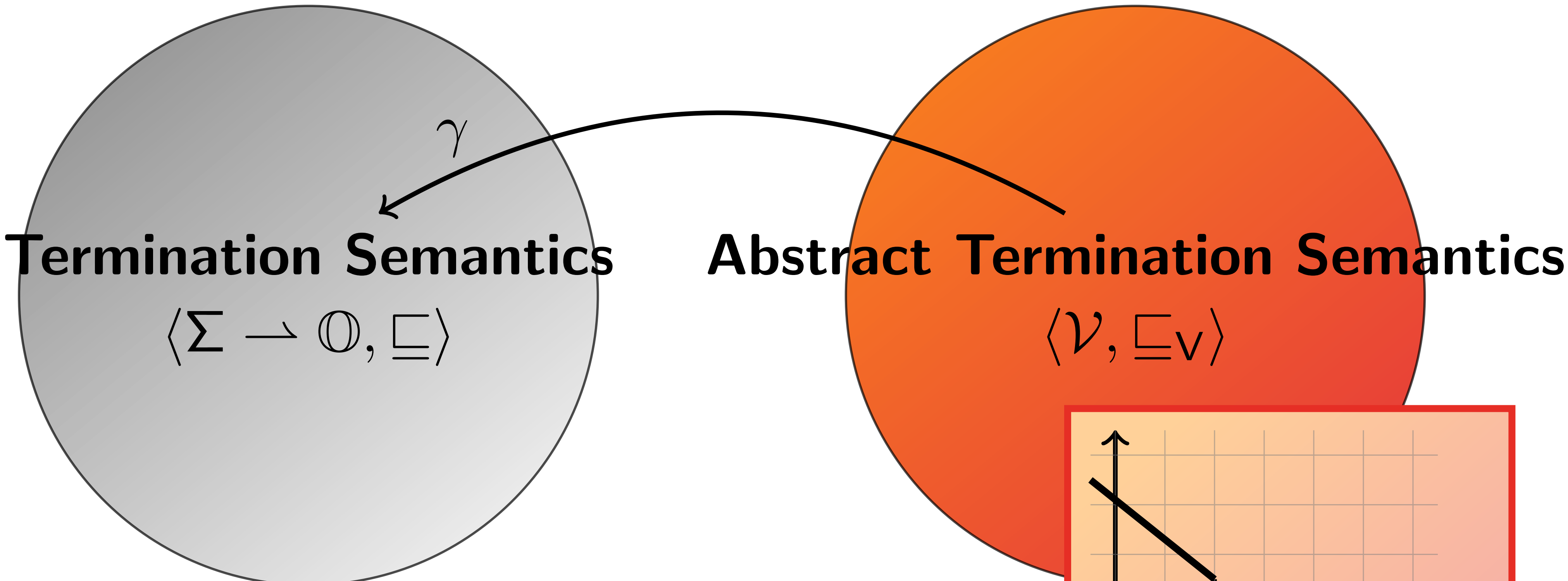


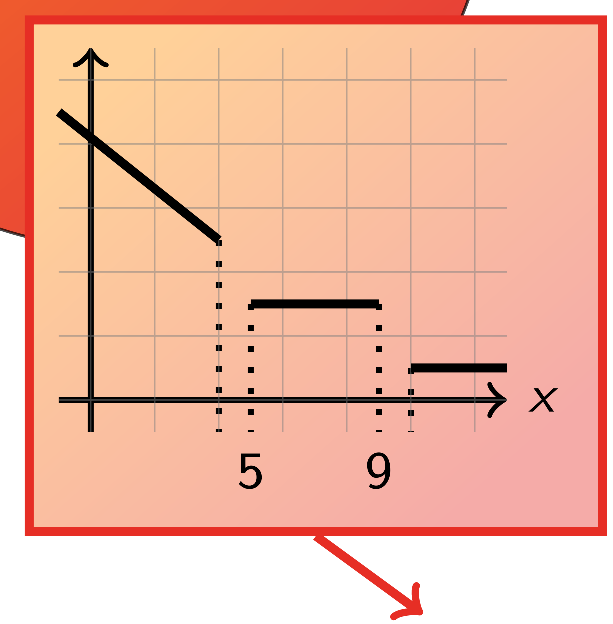
Piecewise-Defined Ranking Functions



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

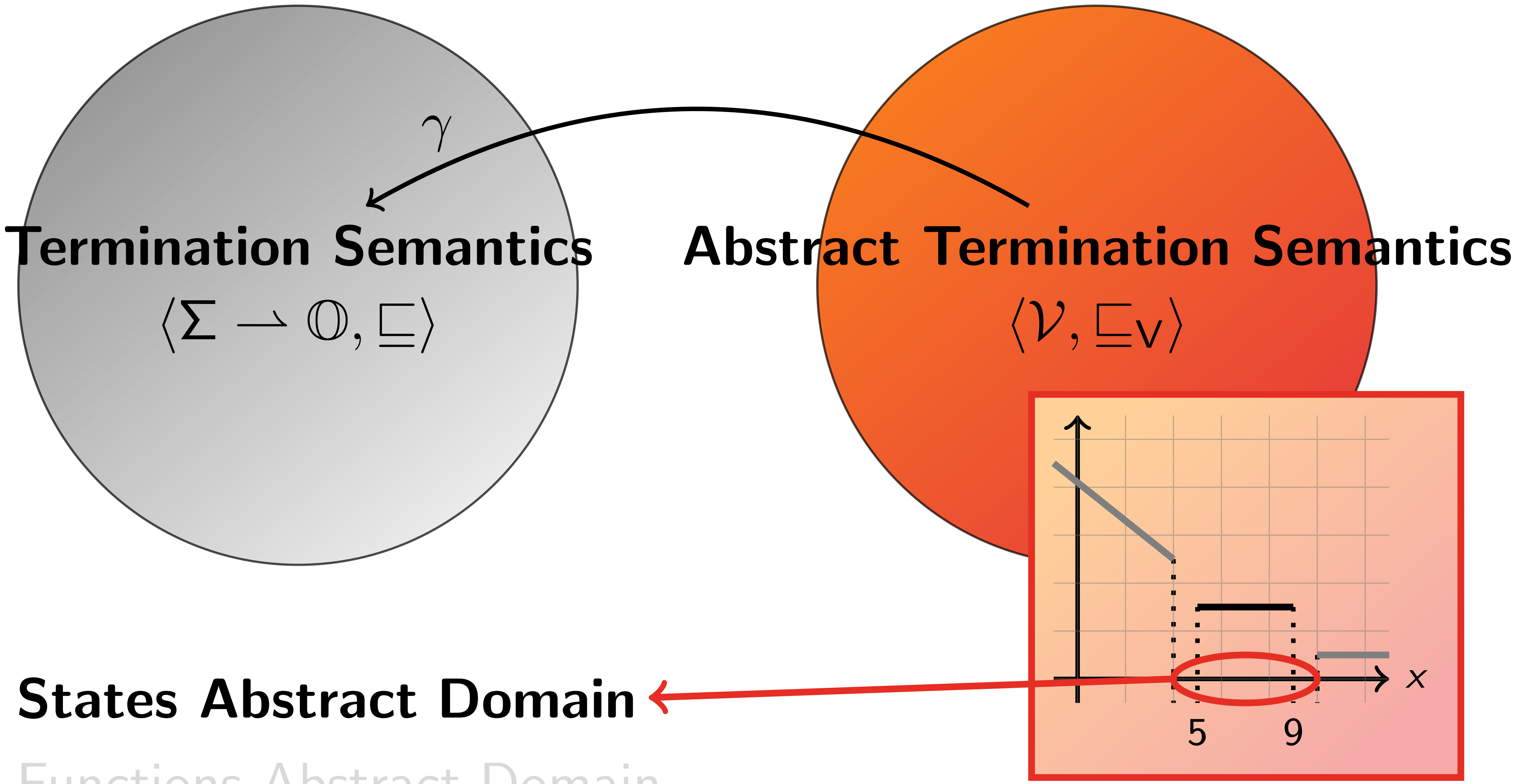
Urban - *The Abstract Domain of Segmented Ranking Functions* (SAS 2013)





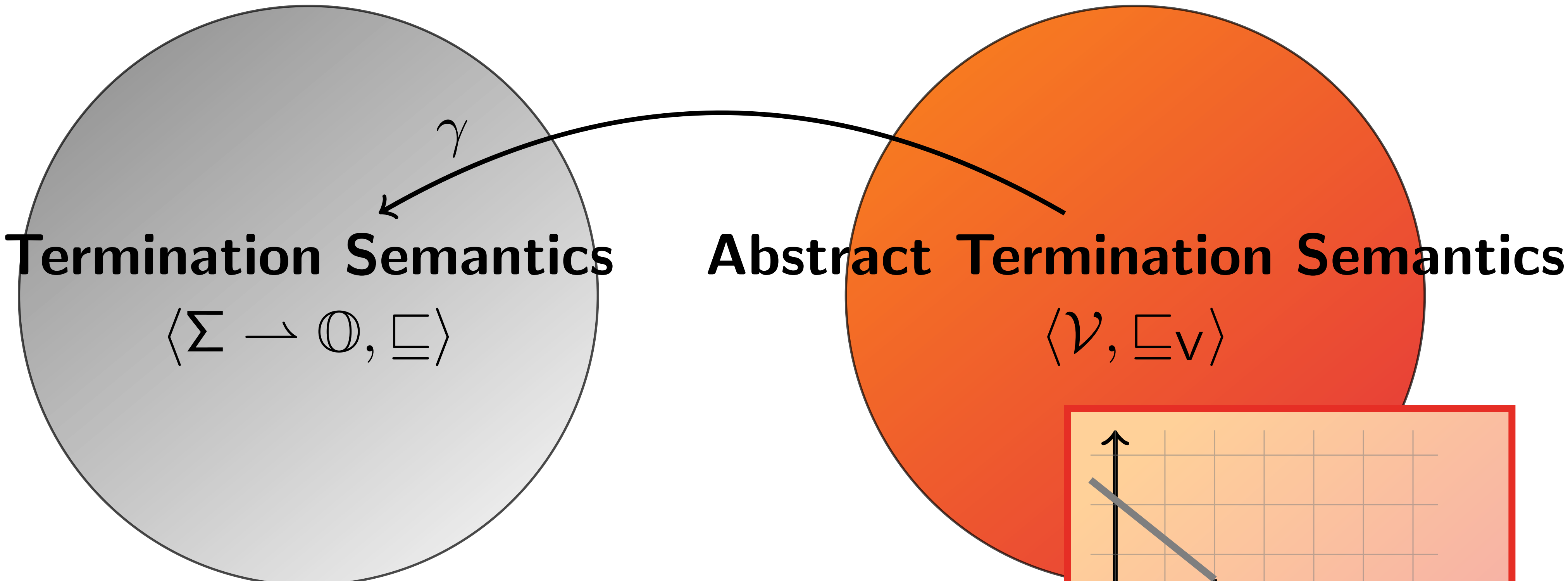
- States Abstract Domain S
- Functions Abstract Domain F
- **Piecewise-Defined Ranking Functions Abstract Domain** V(S, F)

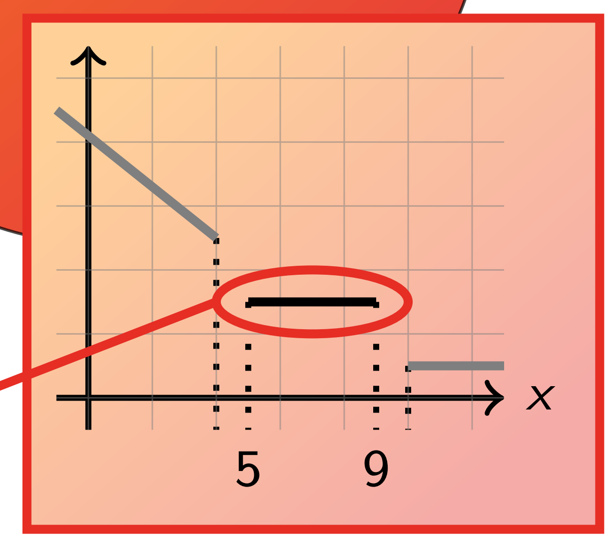
Urban - *The Abstract Domain of Segmented Ranking Functions* (SAS 2013)



- **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)

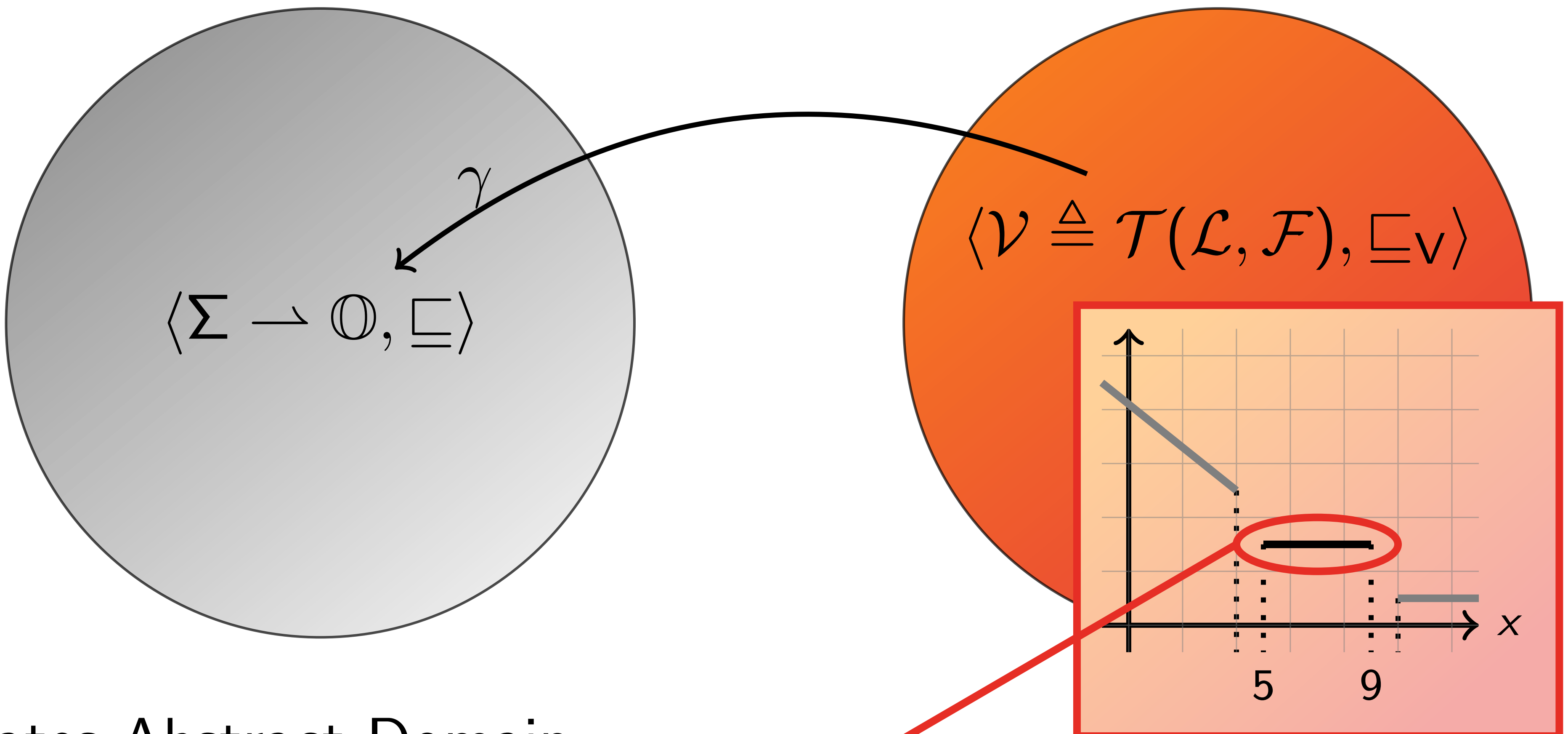




- States Abstract Domain — S
- **Functions Abstract Domain** — F
- Piecewise-Defined Ranking Functions Abstract Domain — V(S, F)

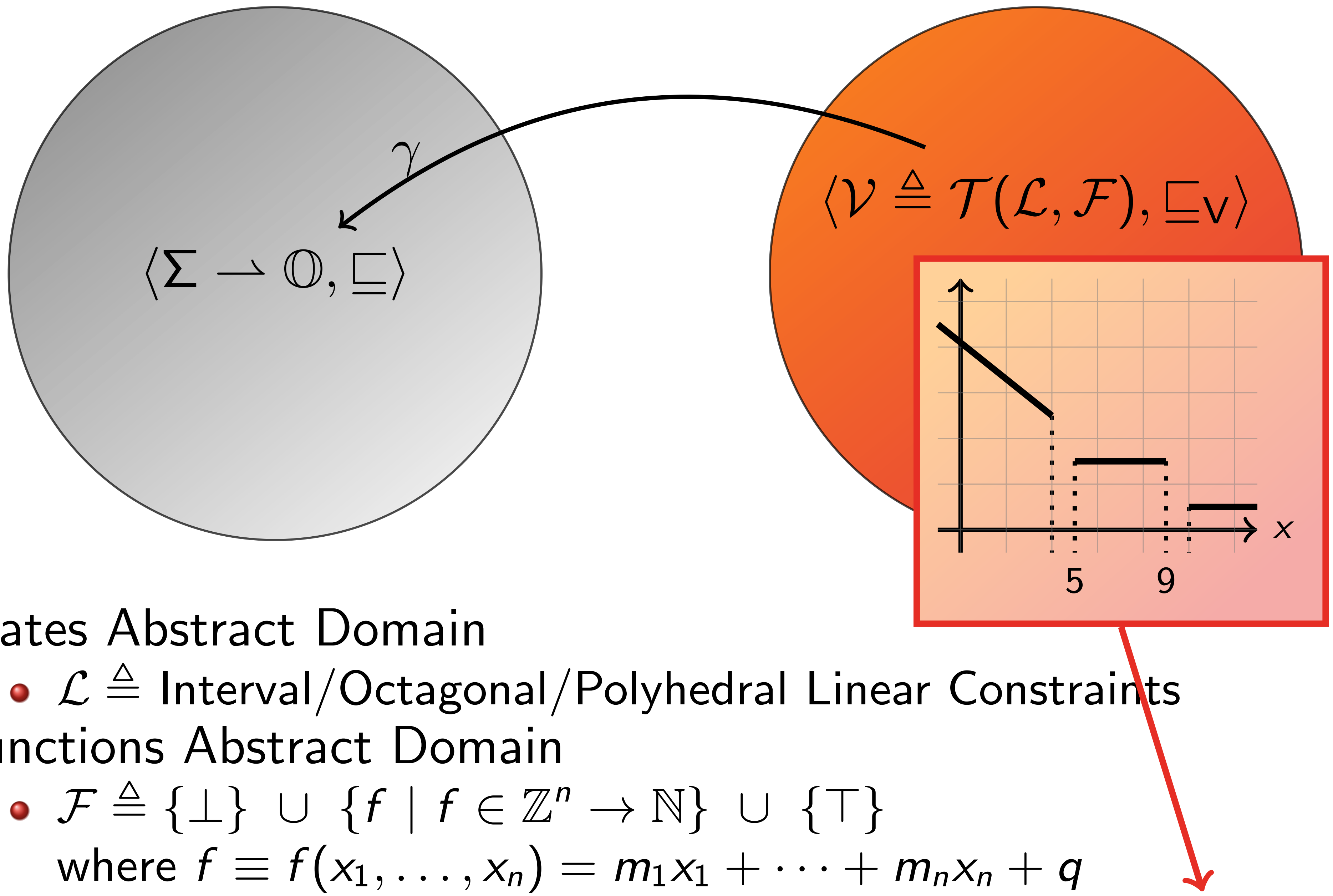
Urban - *The Abstract Domain of Segmented Ranking Functions* (SAS 2013)

Decision Tree Abstract Domain



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

Decision Tree Abstract Domain



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

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

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

od^{3}

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

Example

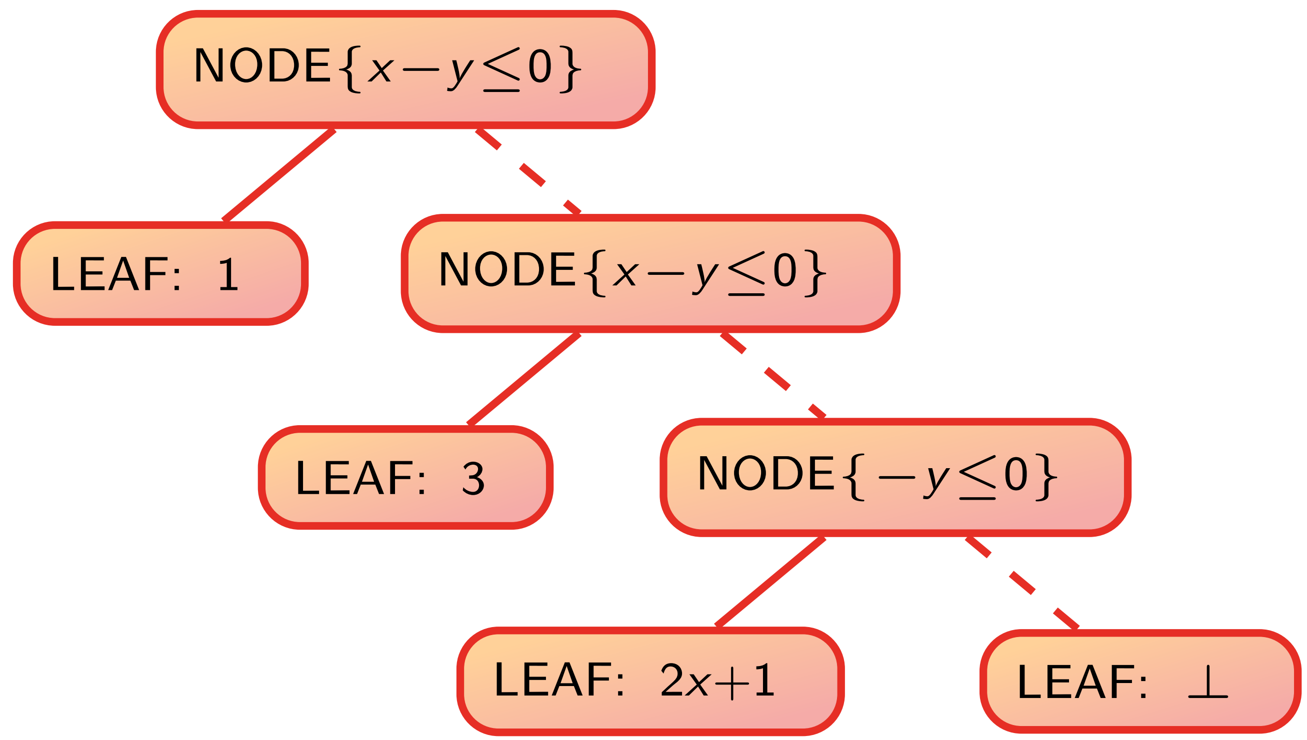
int : x, y

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

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

od^{3}

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



Example

int : x, y

while $^1(x > 0)$ do

$^2x := x - y$

od^3

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

$$
\begin{tikzcd}
\mathbf{1} \arrow[d, "x > 0"] \arrow[dr, bend left, "x \leq 0"] & \\
\mathbf{2} \arrow[u, bend left, "x := x - y"] & \mathbf{3}
\end{tikzcd}
$$

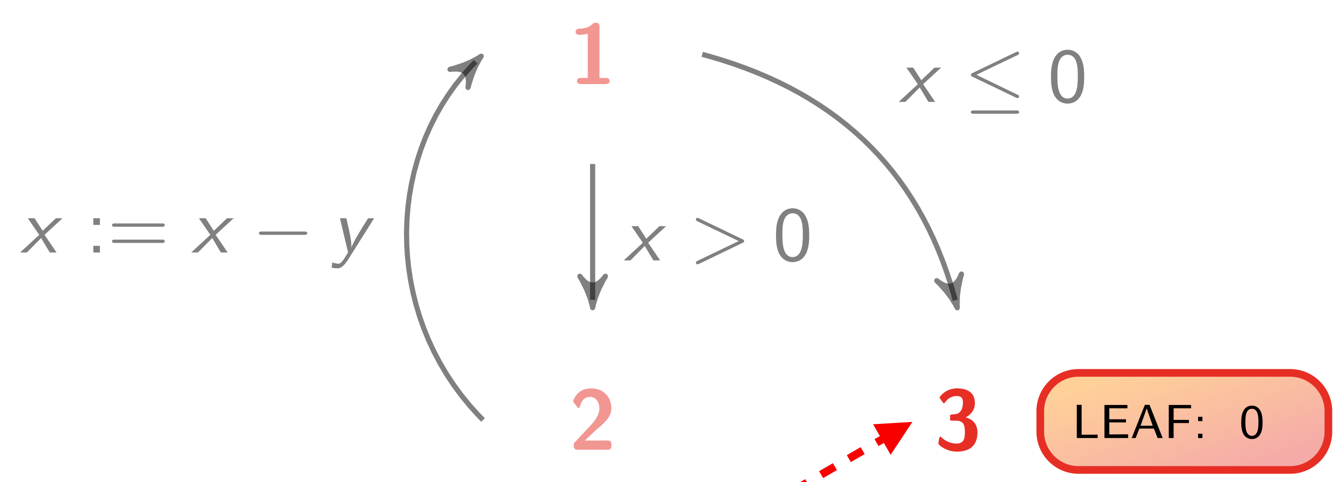
Example

int : x, y

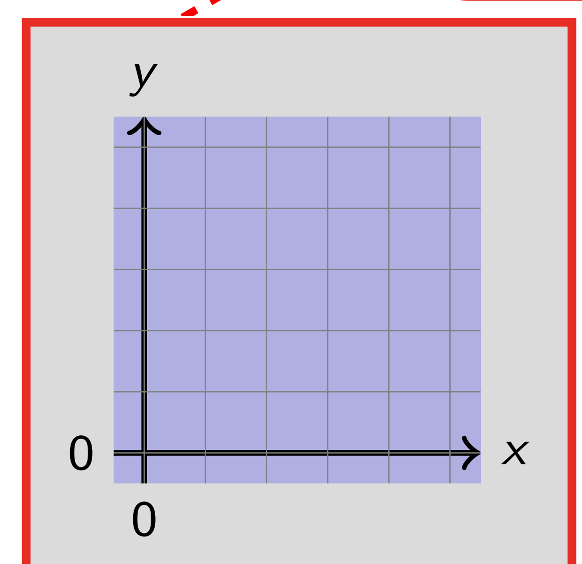
while $^1(x > 0)$ do

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

od^3



we start at the end
with 0 steps
before termination



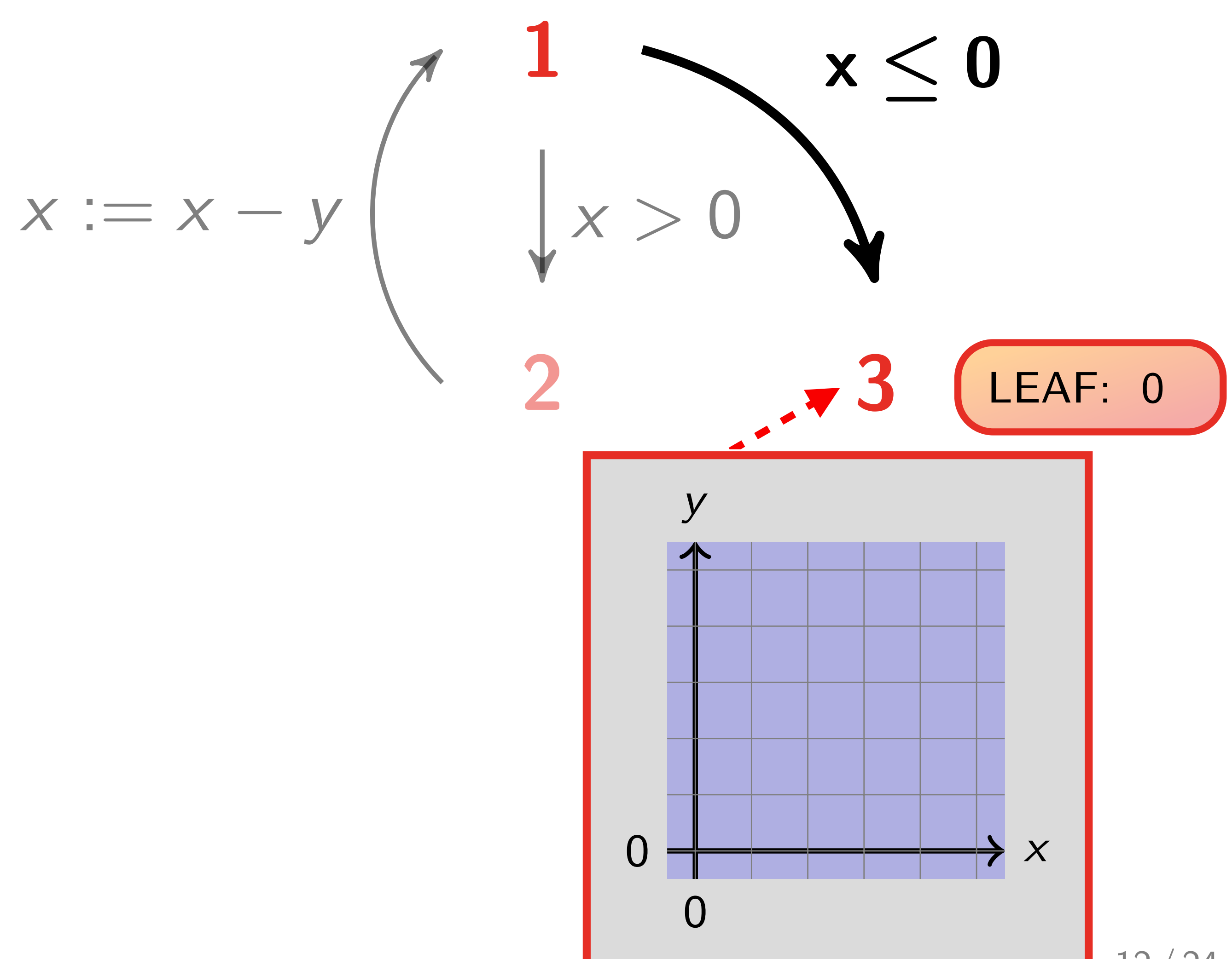
Example

$\text{int} : x, \ y$

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

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

od^3

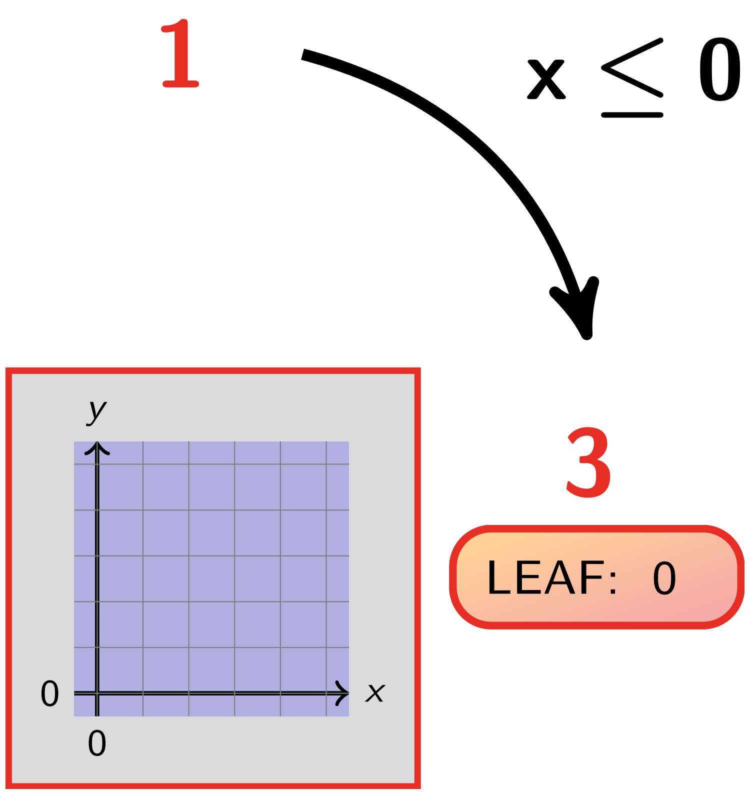


Tests

Algorithm 4 : Tree Filter

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

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

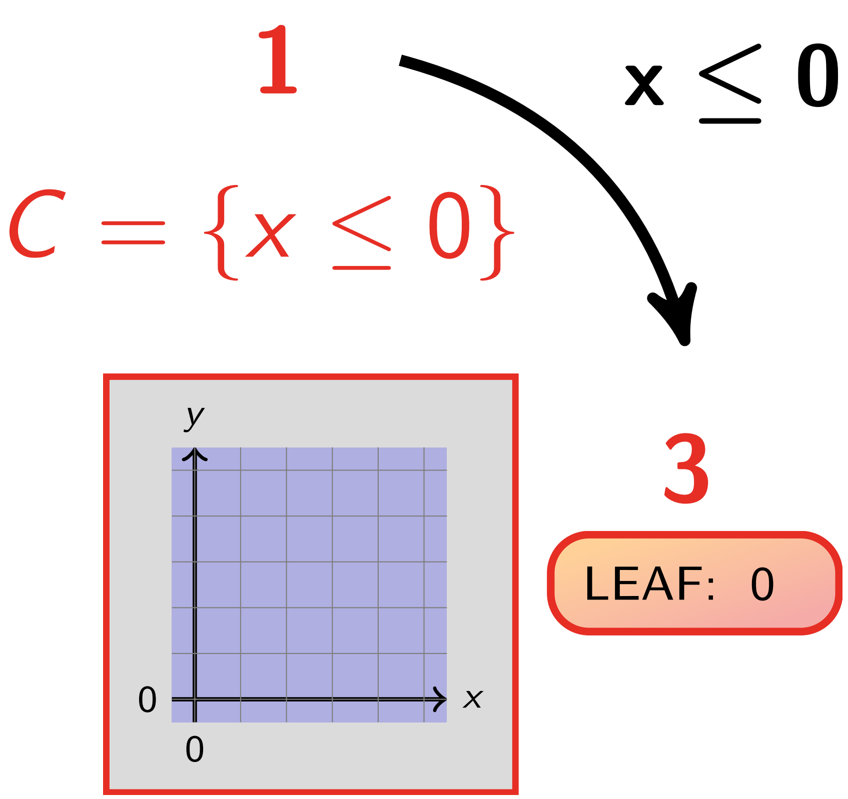


Tests

Algorithm 4 : Tree Filter

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

function FILTER(t,c)
  $C \leftarrow \text{FILTER}_\text{L}(c)$
  return AUGMENT(FILTER-AUX$(t, c), C)$

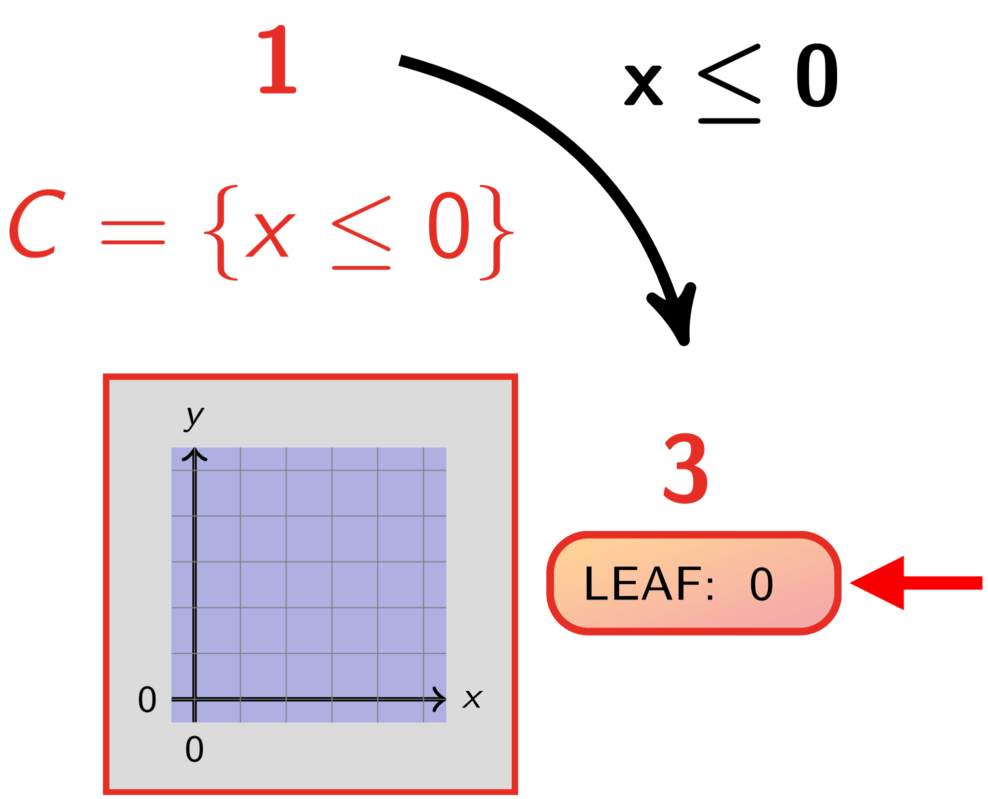


Tests

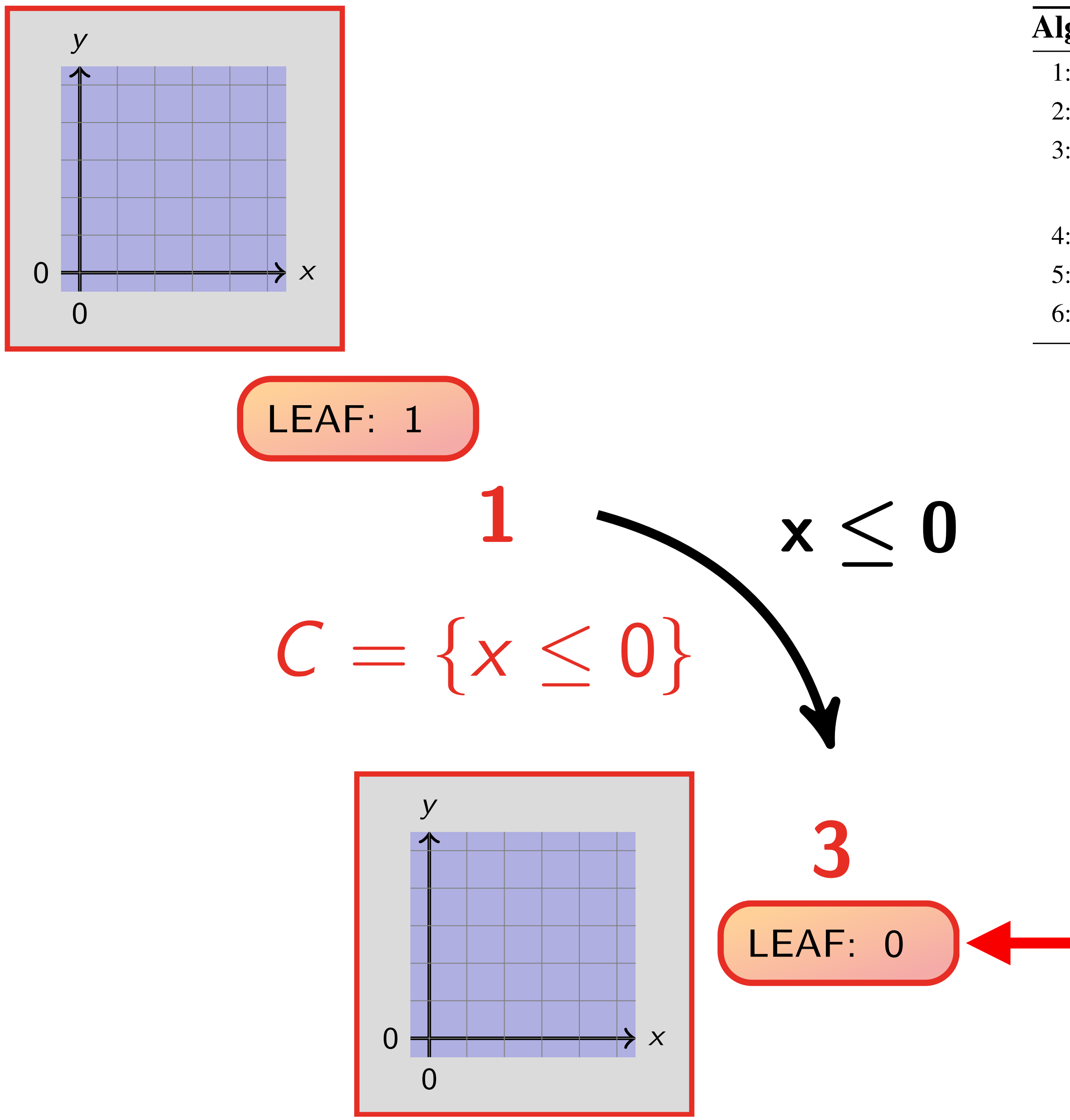
Algorithm 4 : Tree Filter

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

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



Tests

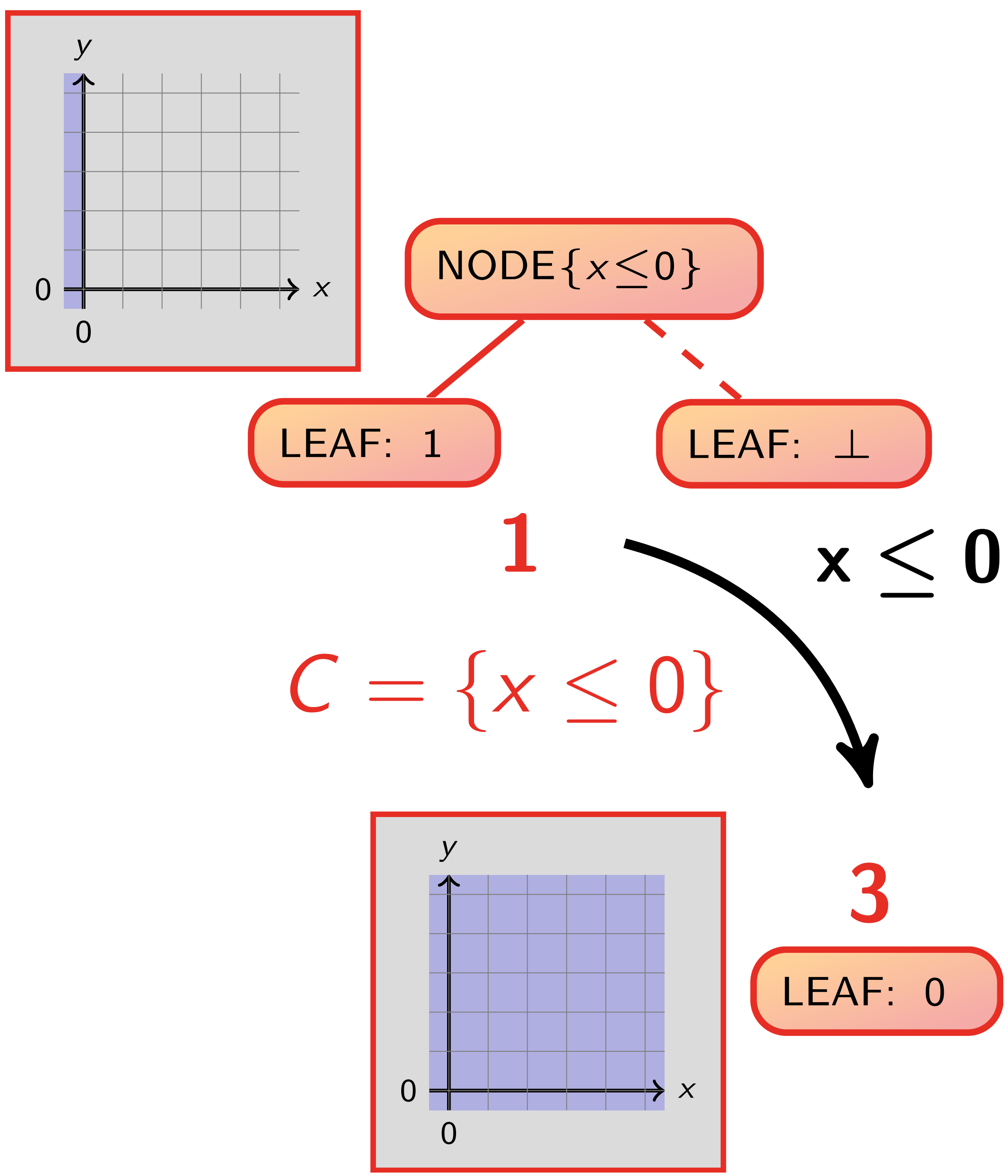


Algorithm 4 : Tree Filter

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

function FILTER(t,c)
  $C \leftarrow \mathsf{FILTER}_\mathsf{L}(c)$
  return AUGMENT(FILTER-AUX(t, c), C)

Tests



Algorithm 4 : Tree Filter

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

function FILTER(t,c)
    C ← FILTER_L(c)
    return AUGMENT(FILTER-AUX(t, c), C)
```

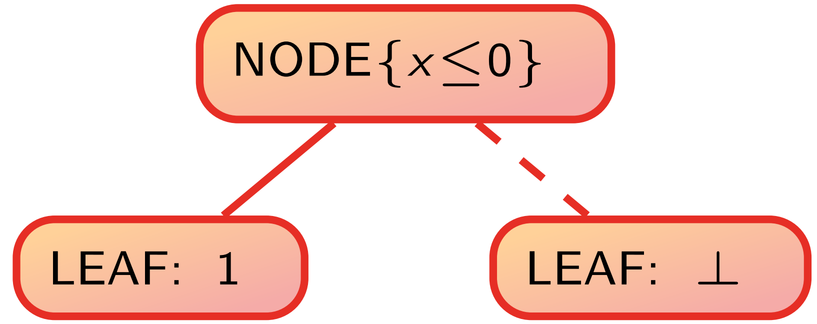
Example

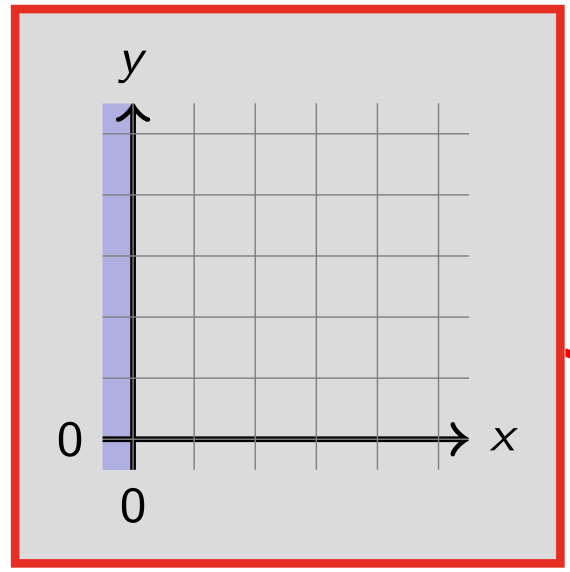
$\text{int} : x,\ y$

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

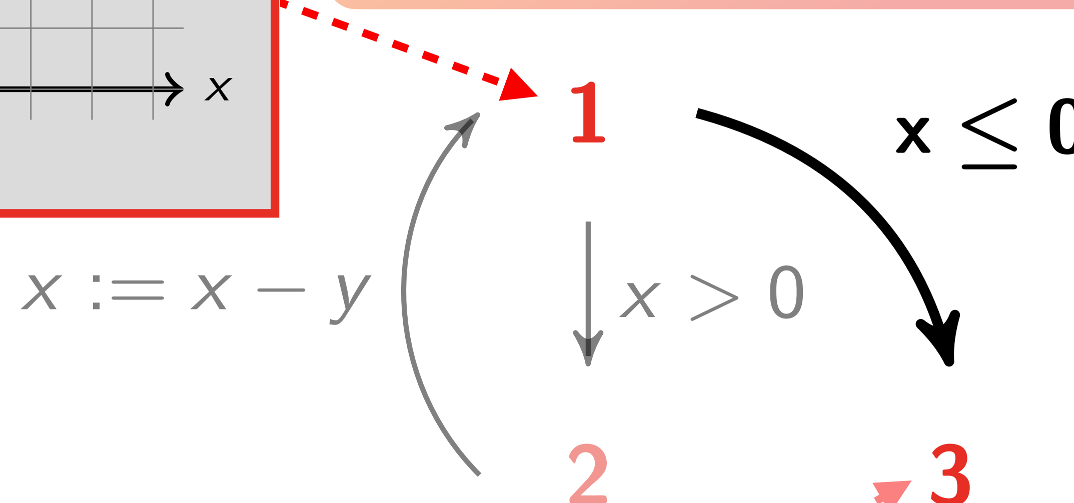
$\quad {}^{2}x := x - y$

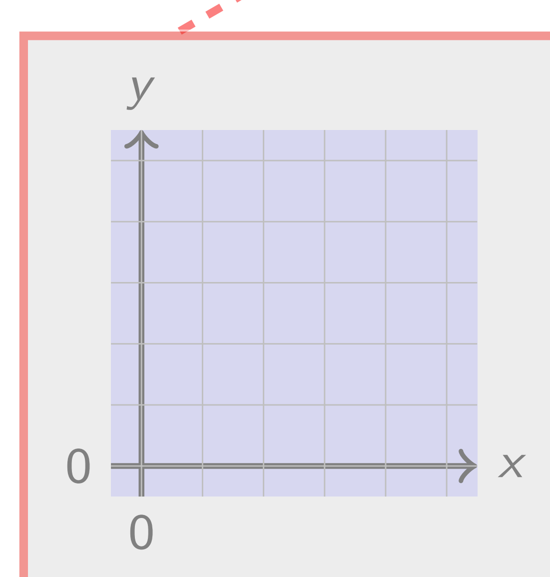
od^{3}





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





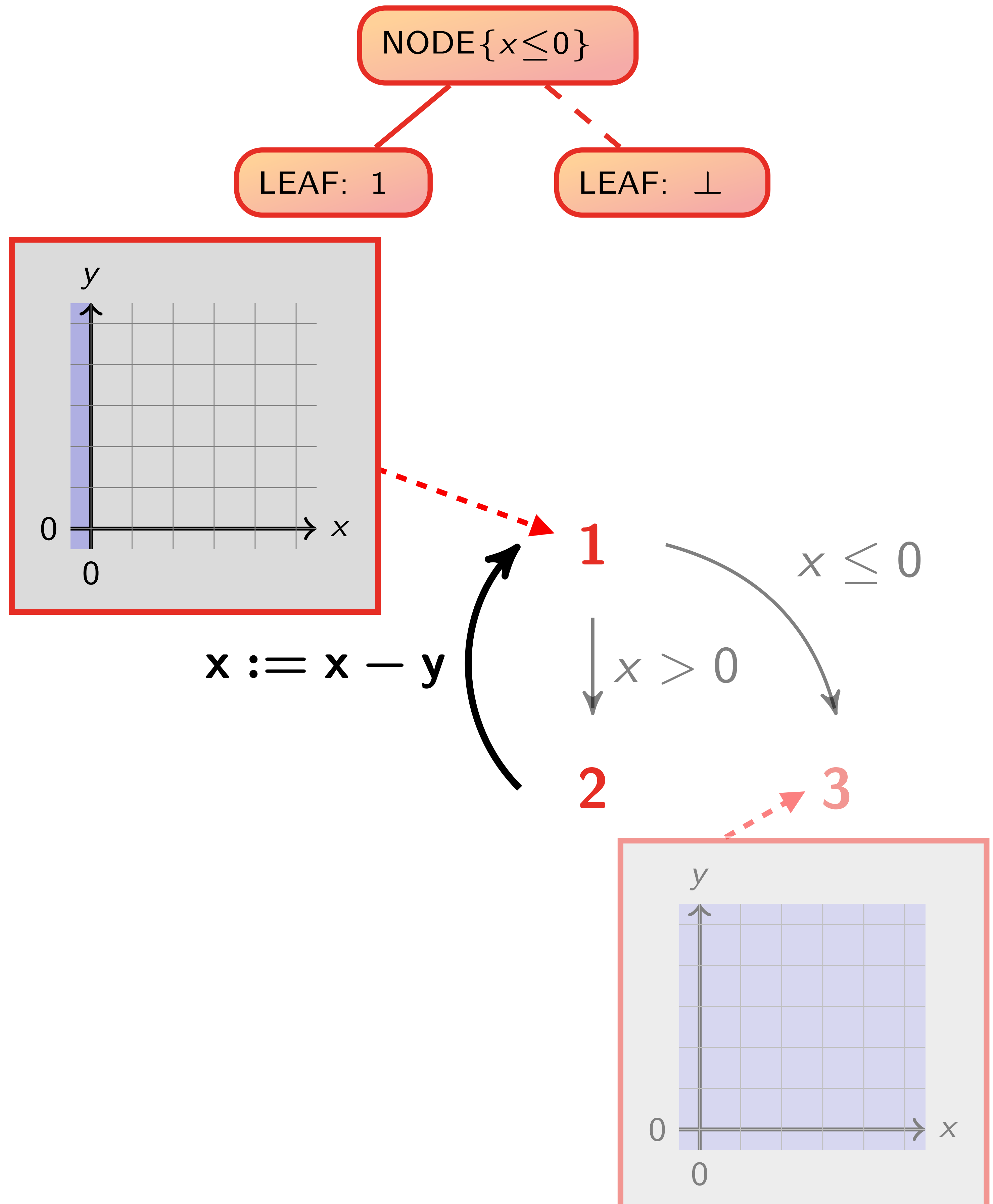
Example

int : x, y

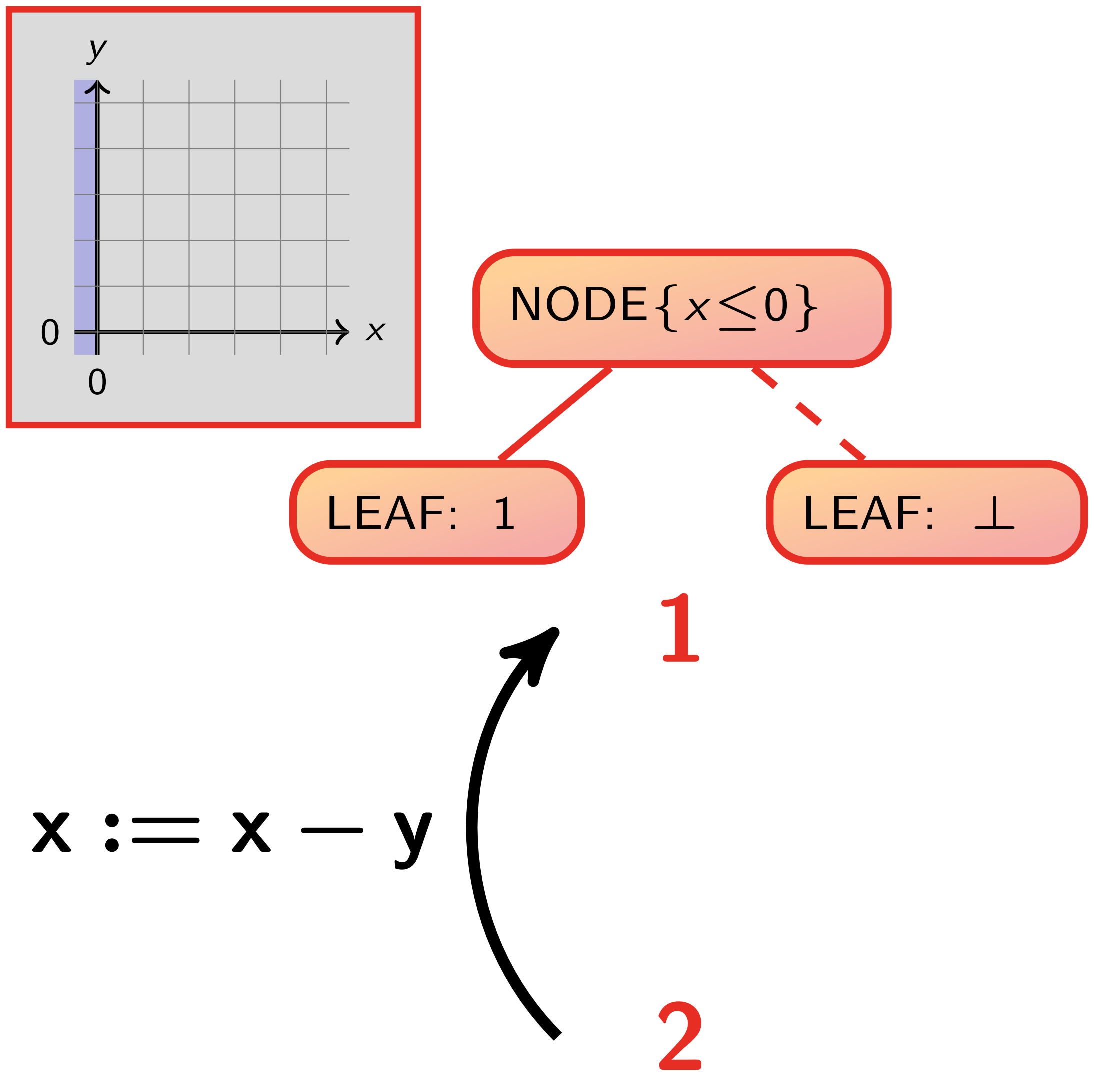
while [1]$(x > 0)$ do

[2]$x := x - y$

od[3]



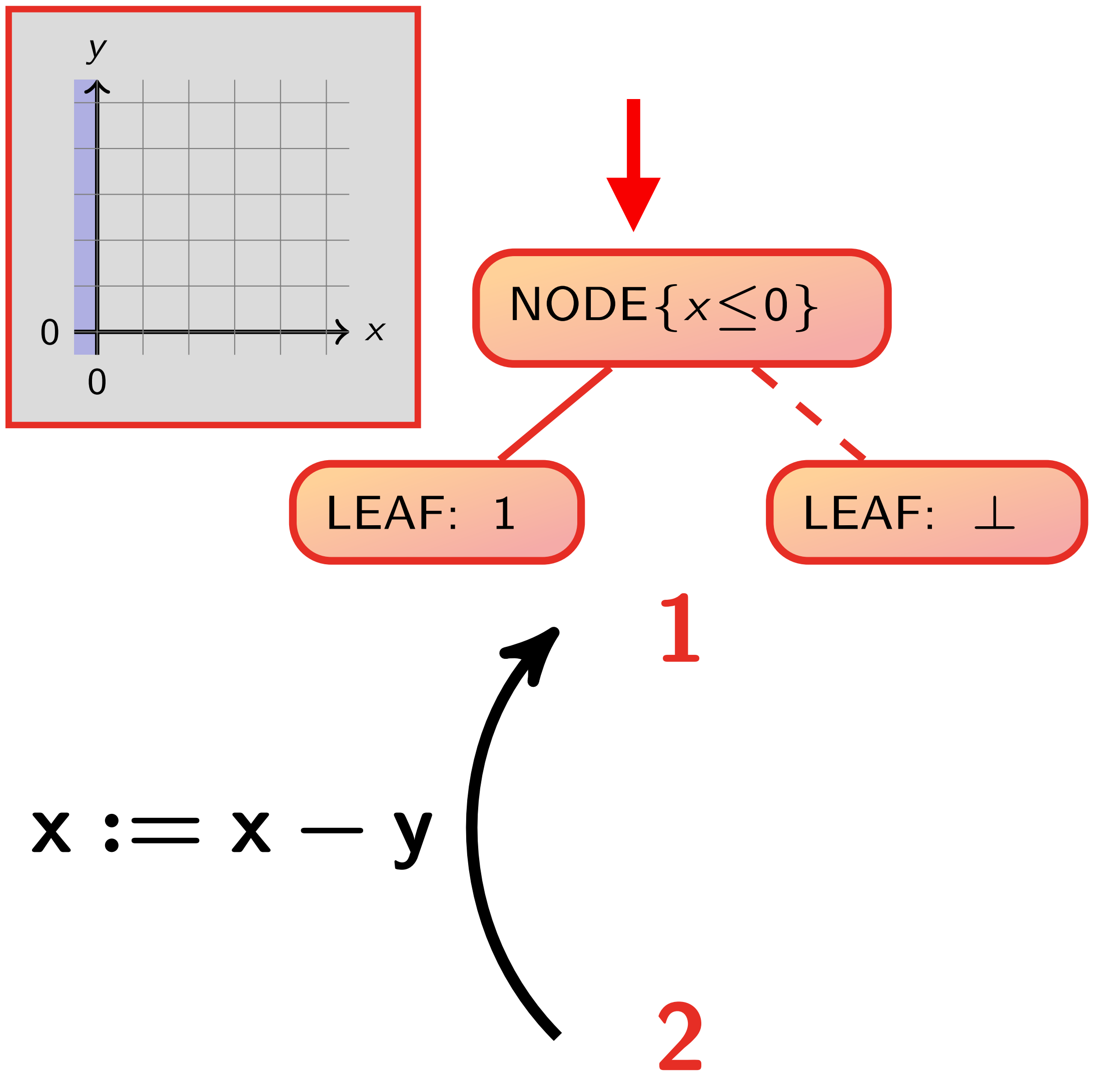
Assignments



Algorithm 3 : Tree Assign

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

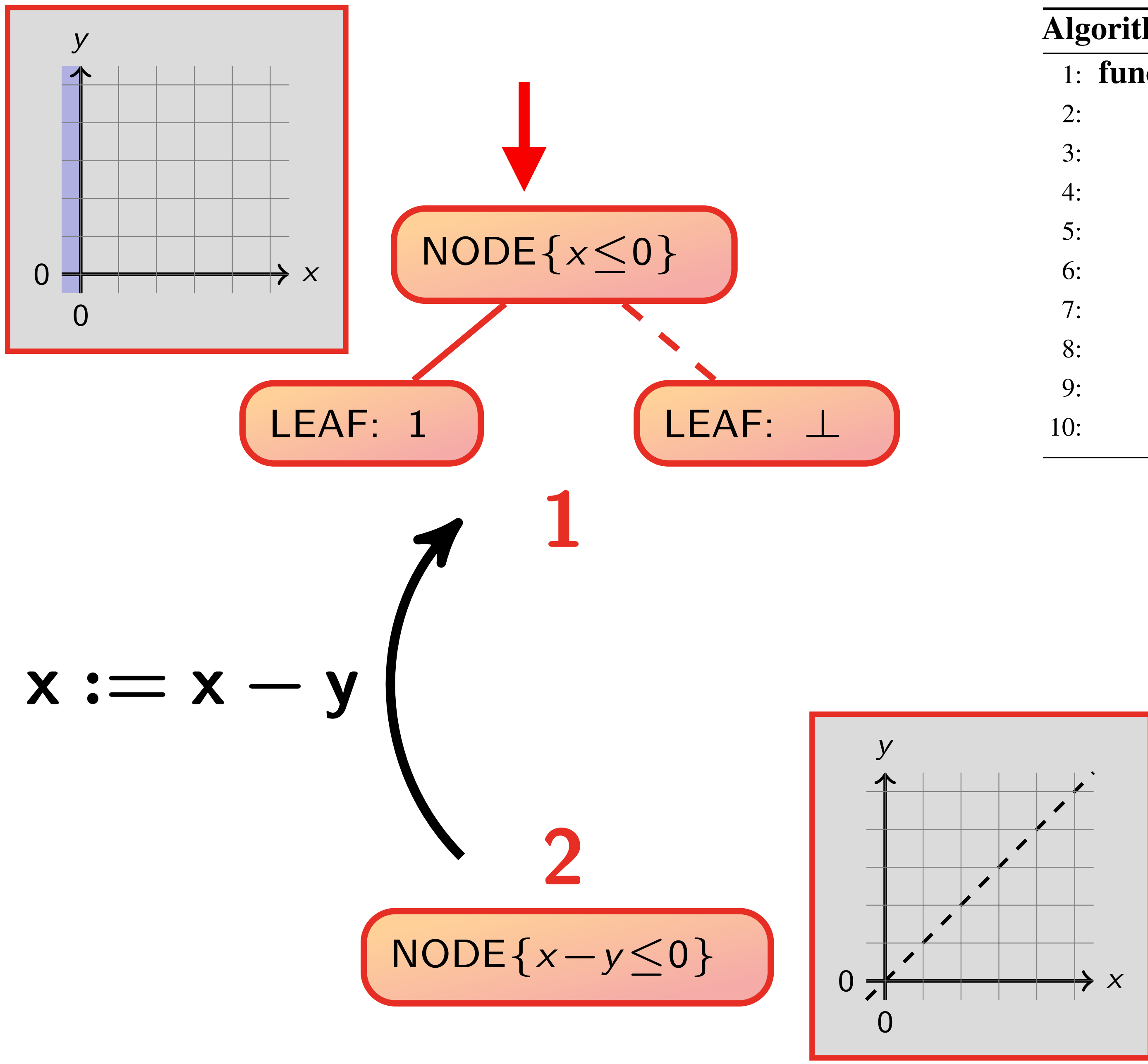
Assignments



Algorithm 3 : Tree Assign

function ASSIGN(t,$x := a$)
  if ISLEAF(t) **then return** LEAF : ASSIGN$_{\mathsf{F}}(f, x := a)$ $/* \ t \triangleq$ LEAF : f $*/$
  else
    $C \leftarrow$ ASSIGN$_{\mathsf{L}}(t.c, x := a)$
    if ISEMPTY(C) **then return** ASSIGN($t.l, x := a$) $\sqcup_{\mathsf{T}}$ ASSIGN($t.r, x := a$)
    else if ISUNSAT(C) **then return** ASSIGN($t.r, x := a$)
    else
      $l \leftarrow$ AUGMENT(ASSIGN($t.l, x := a$), C)
      $r \leftarrow$ AUGMENT(ASSIGN($t.r, x := a$), C)
      return NODE$\{l.c\}$: l; r

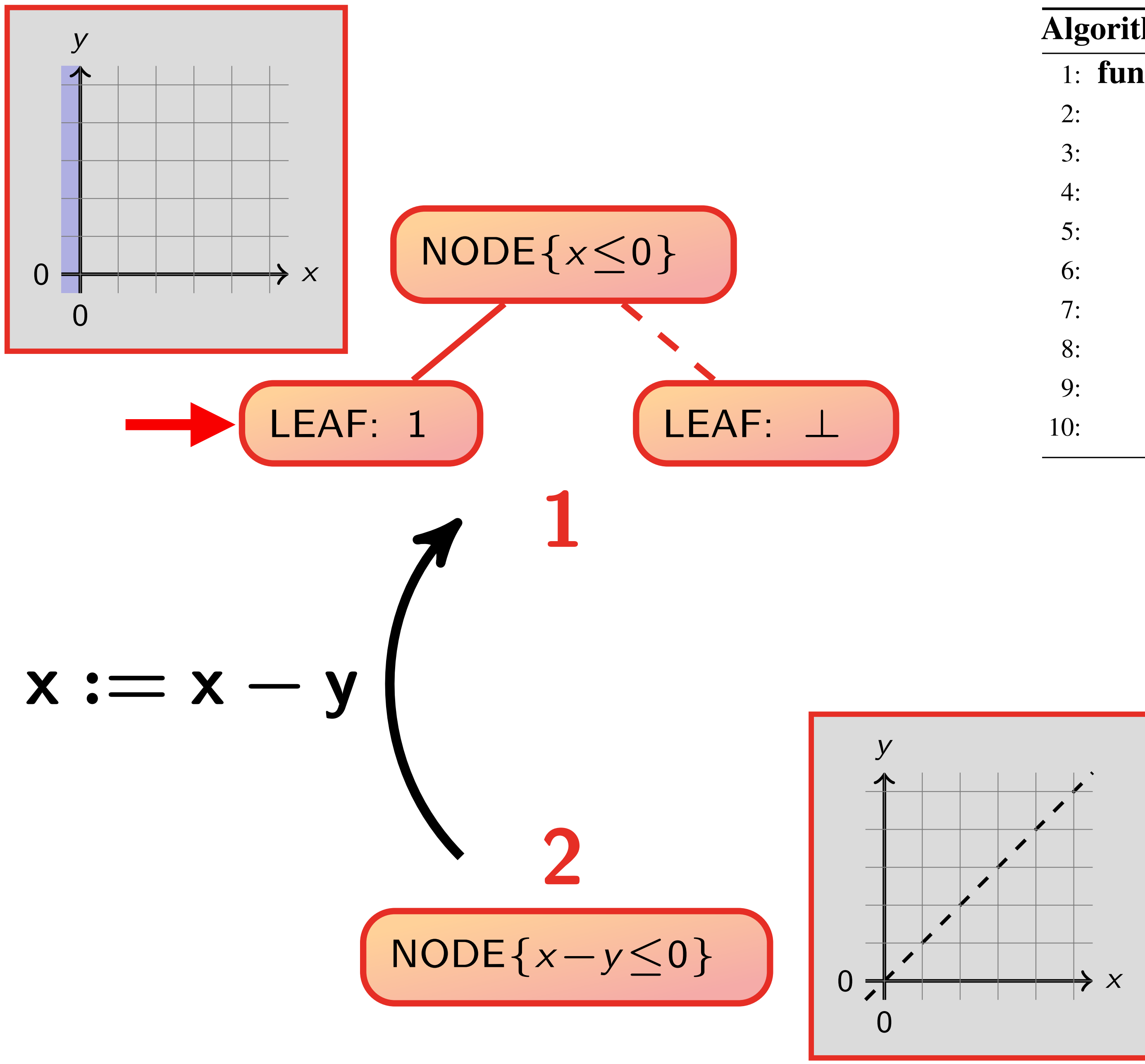
Assignments



Algorithm 3 : Tree Assign

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

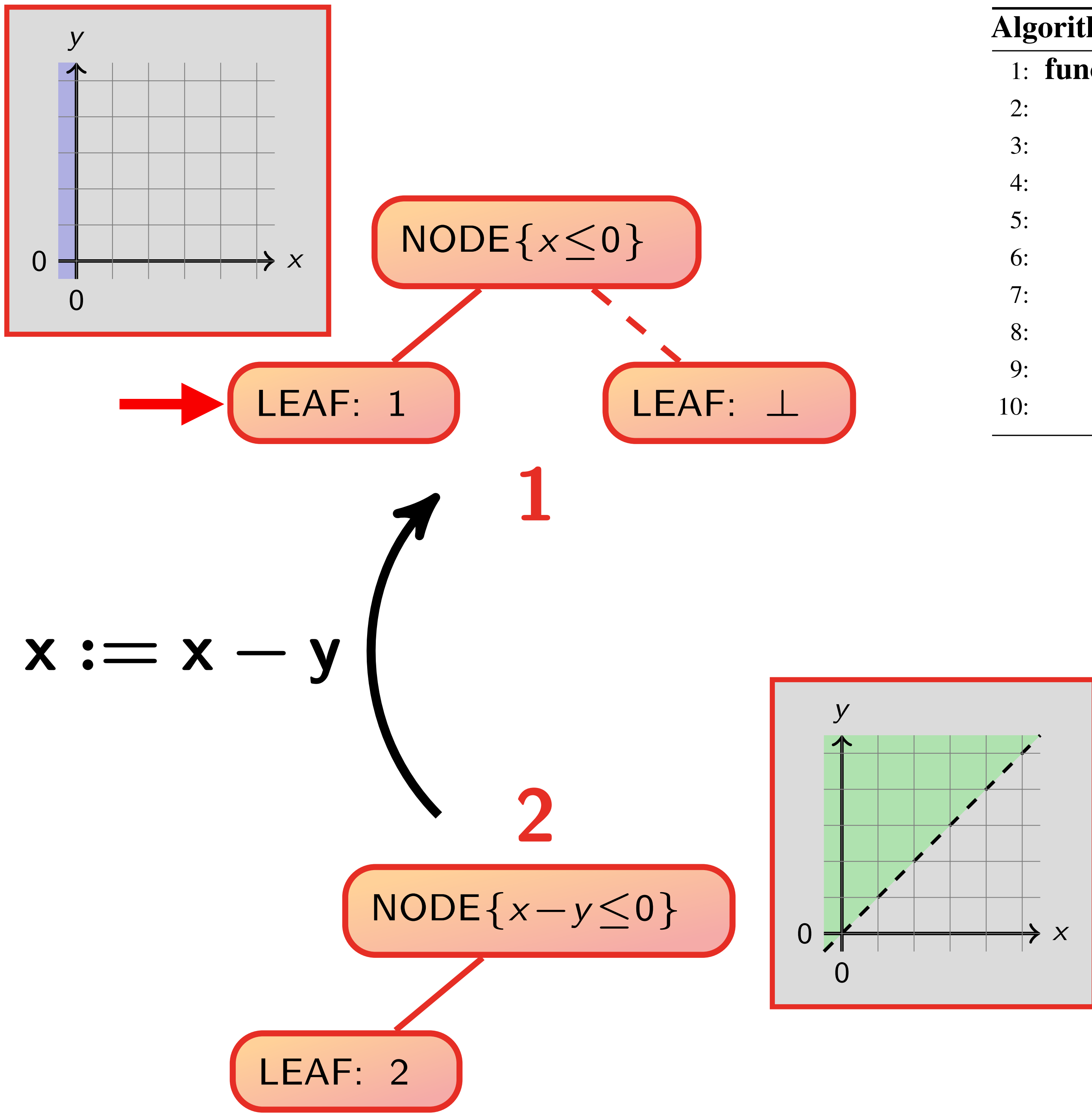
Assignments



Algorithm 3 : Tree Assign

function ASSIGN(t,$x := a$)
  if ISLEAF(t) **then return** LEAF : $\text{ASSIGN}_{\text{F}}(f, x := a)$ $/* \; t \triangleq$ LEAF : $f \; */$
  else
    $C \leftarrow \text{ASSIGN}_{\text{L}}(t.c, x := a)$
    if ISEMPTY(C) **then return** ASSIGN($t.l, x := a$) $\sqcup_{\text{T}}$ ASSIGN($t.r, x := a$)
    else if ISUNSAT(C) **then return** ASSIGN($t.r, x := a$)
    else
      $l \leftarrow$ AUGMENT(ASSIGN($t.l, x := a$), C)
      $r \leftarrow$ AUGMENT(ASSIGN($t.r, x := a$), C)
      return NODE$\{l.c\} : l; r$

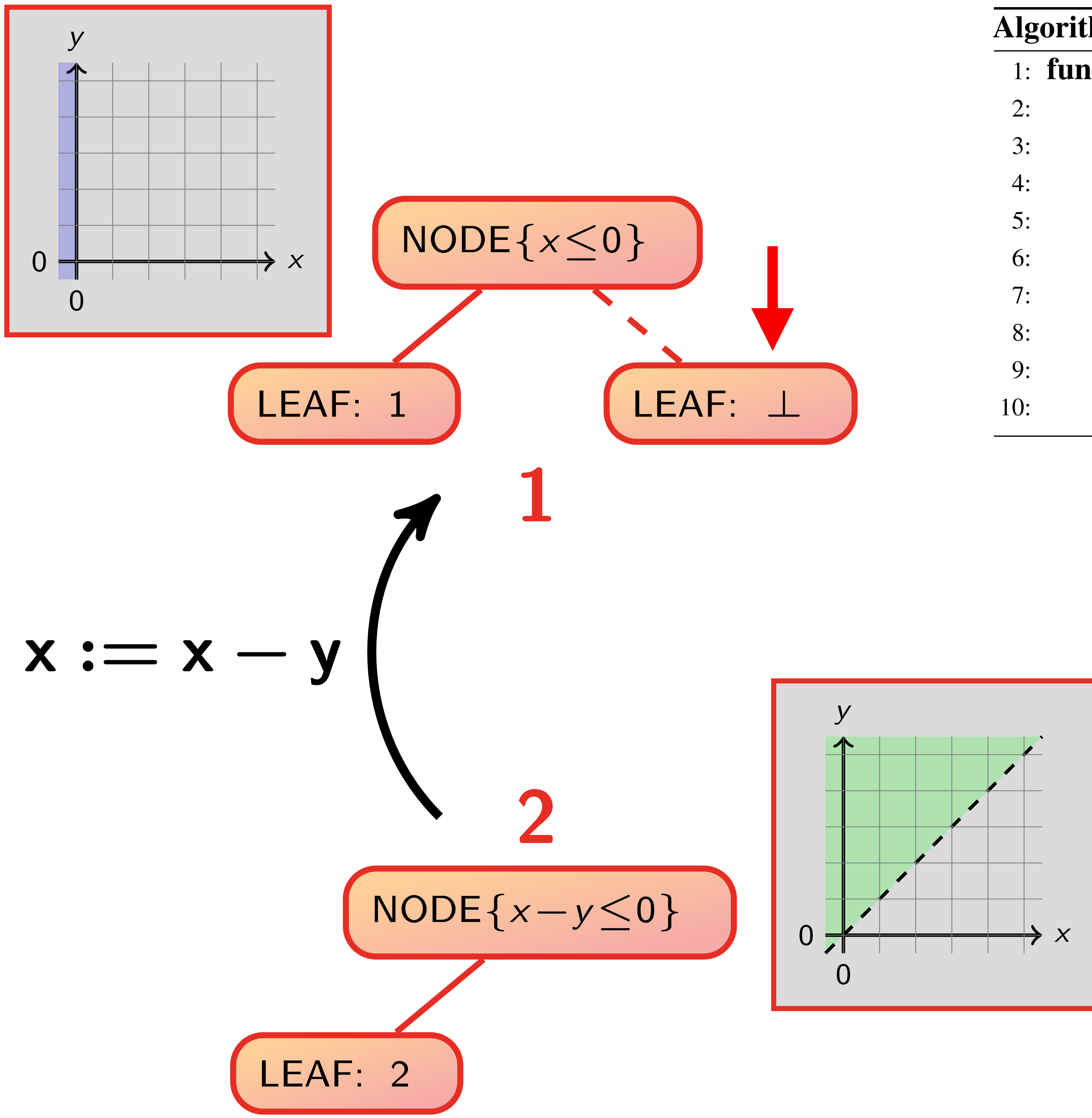
Assignments



Algorithm 3 : Tree Assign

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

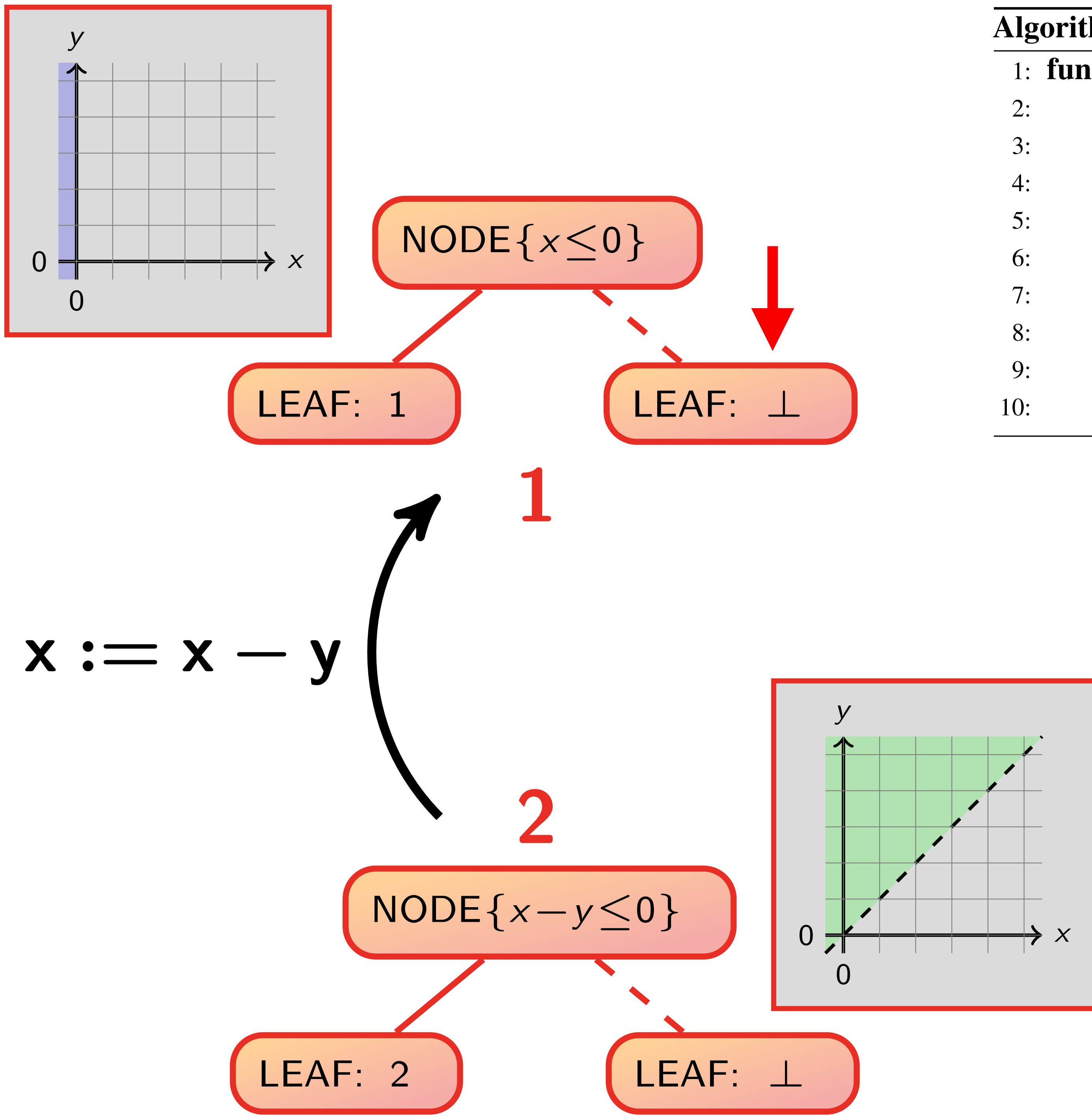
Assignments



Algorithm 3 : Tree Assign

function ASSIGN(t,$x := a$)
  if ISLEAF(t) **then return** LEAF : ASSIGN$_\mathsf{F}(f, x := a)$ $/* \ t \triangleq$ LEAF : f $*/$
  else
    $C \leftarrow$ ASSIGN$_\mathsf{L}(t.c, x := a)$
    if ISEMPTY(C) **then return** ASSIGN($t.l, x := a$) $\sqcup_\mathsf{T}$ ASSIGN($t.r, x := a$)
    else if ISUNSAT(C) **then return** ASSIGN($t.r, x := a$)
    else
      $l \leftarrow$ AUGMENT(ASSIGN($t.l, x := a$), C)
      $r \leftarrow$ AUGMENT(ASSIGN($t.r, x := a$), C)
      return NODE$\{l.c\}$: $l; r$

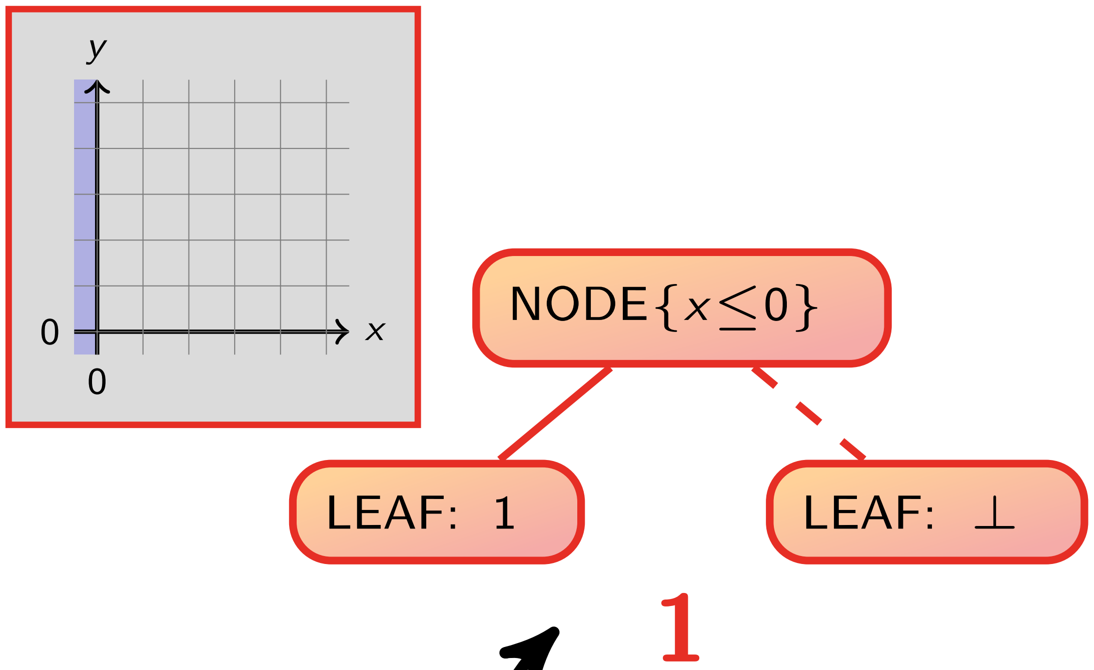
Assignments

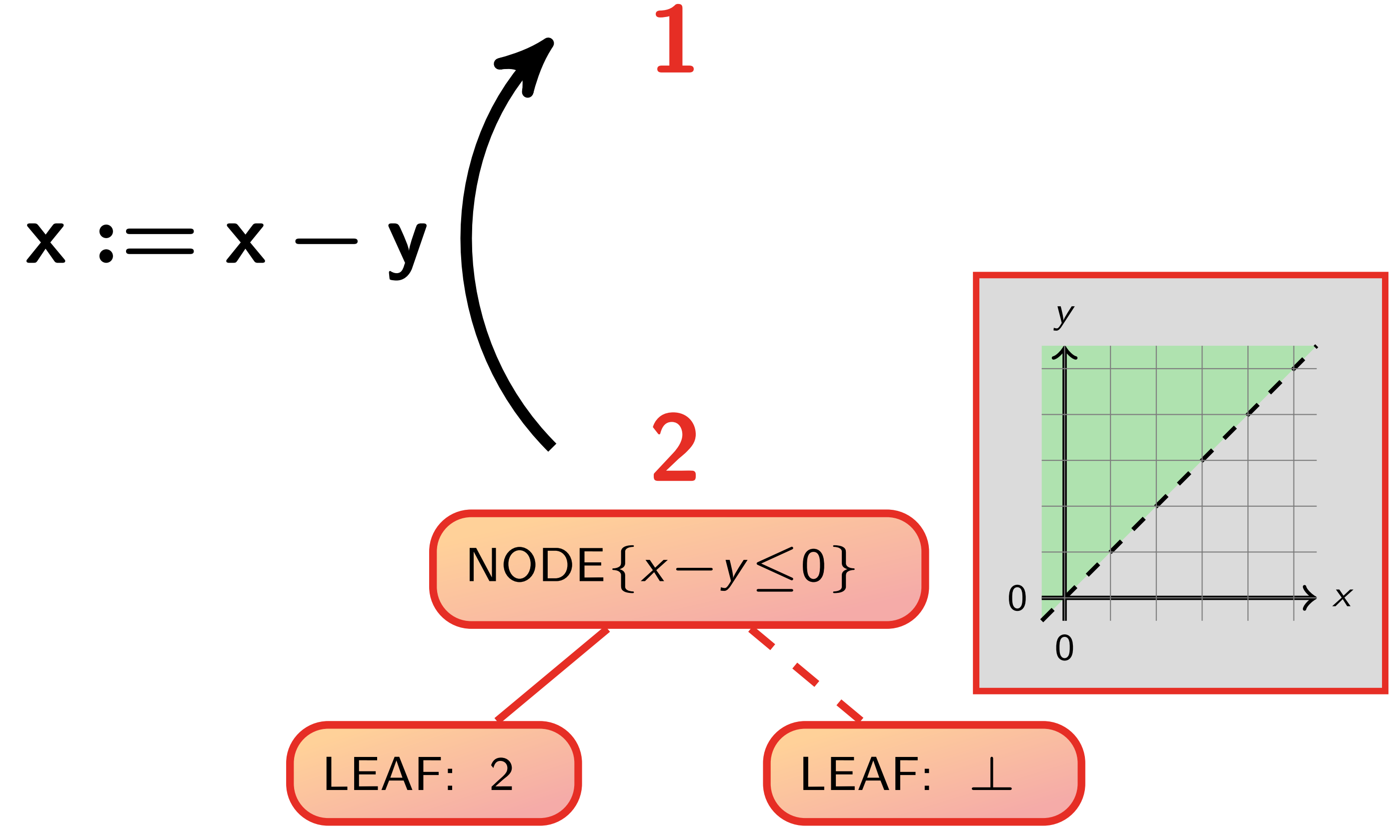


Algorithm 3 : Tree Assign

function ASSIGN(t,$x := a$)
  if ISLEAF(t) **then return** LEAF : ASSIGN$_{\mathsf{F}}(f, x := a)$ $/\!* \; t \triangleq$ LEAF : f $*/$
  else
    $C \leftarrow$ ASSIGN$_{\mathsf{L}}(t.c, x := a)$
    if ISEMPTY(C) **then return** ASSIGN($t.l, x := a$) $\sqcup_{\mathsf{T}}$ ASSIGN($t.r, x := a$)
    else if ISUNSAT(C) **then return** ASSIGN($t.r, x := a$)
    else
      $l \leftarrow$ AUGMENT(ASSIGN($t.l, x := a$), C)
      $r \leftarrow$ AUGMENT(ASSIGN($t.r, x := a$), C)
      return NODE$\{l.c\}$: l; r

Assignments





Algorithm 3 : Tree Assign

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

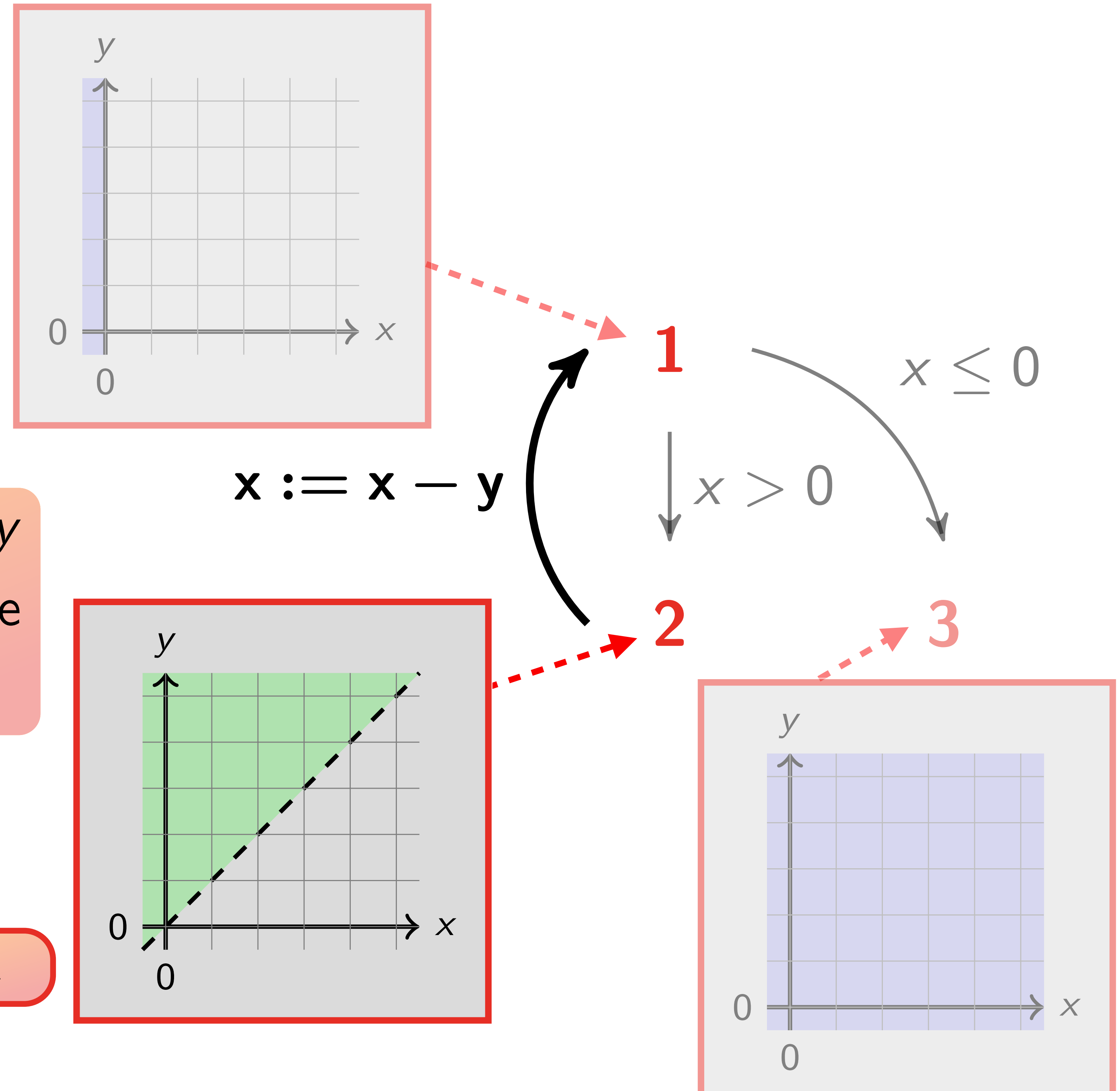
Example

int : x, y

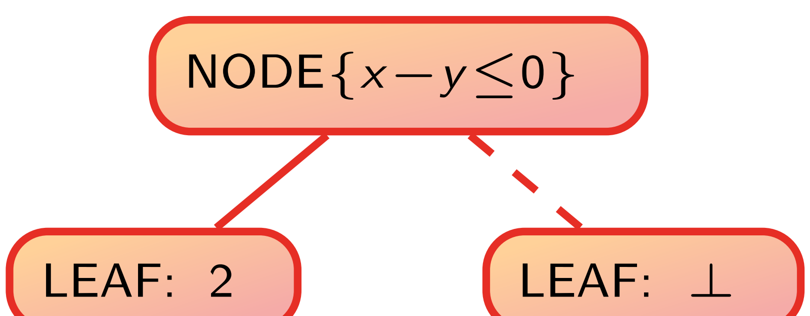
while [1]$(x > 0)$ do

[2]$x := x - y$

od[3]



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



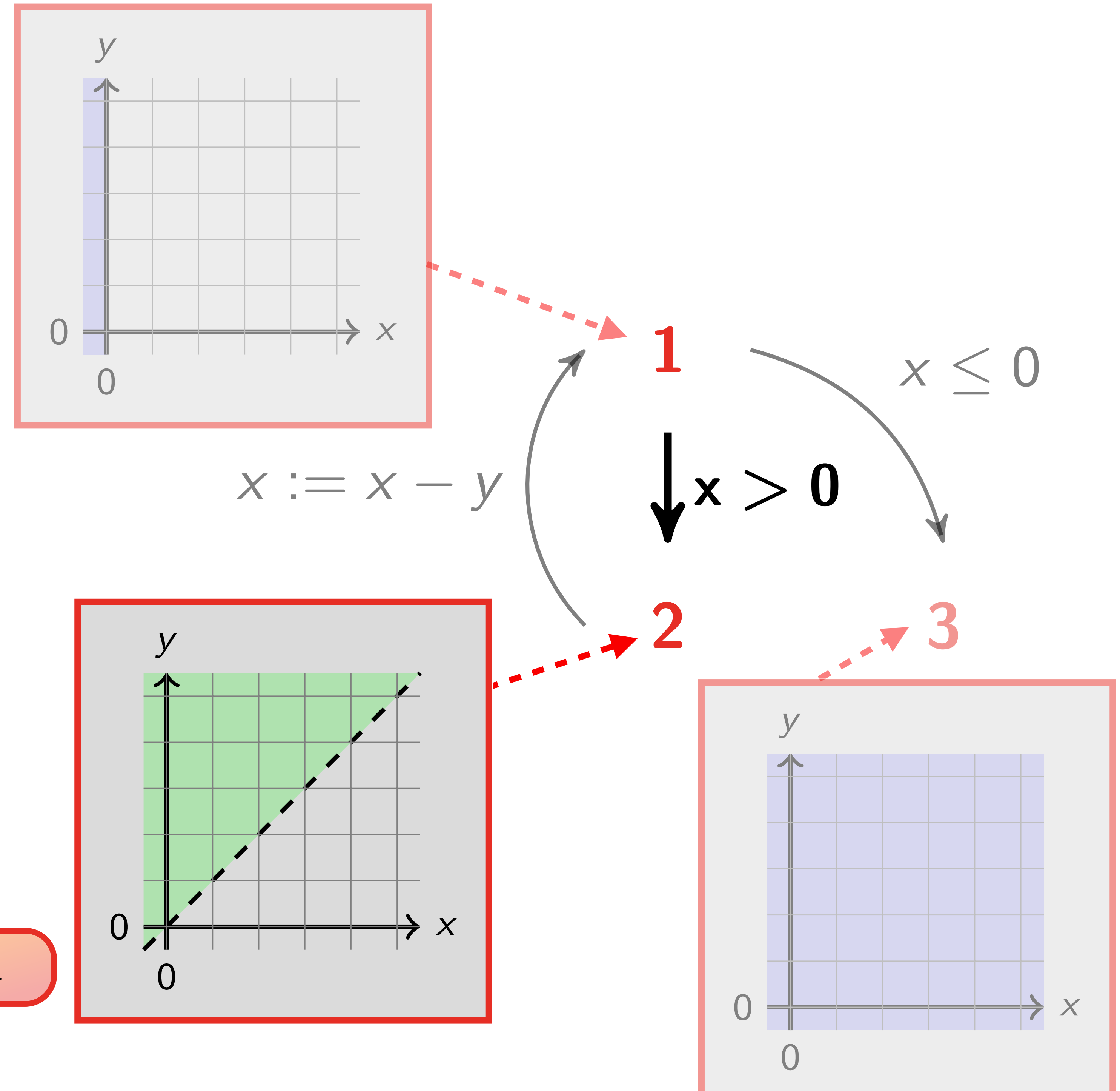
Example

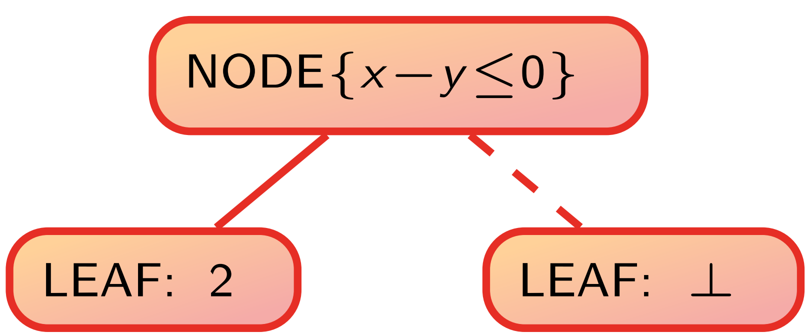
int : x, y

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

  [2]$x := x - y$

od[3]



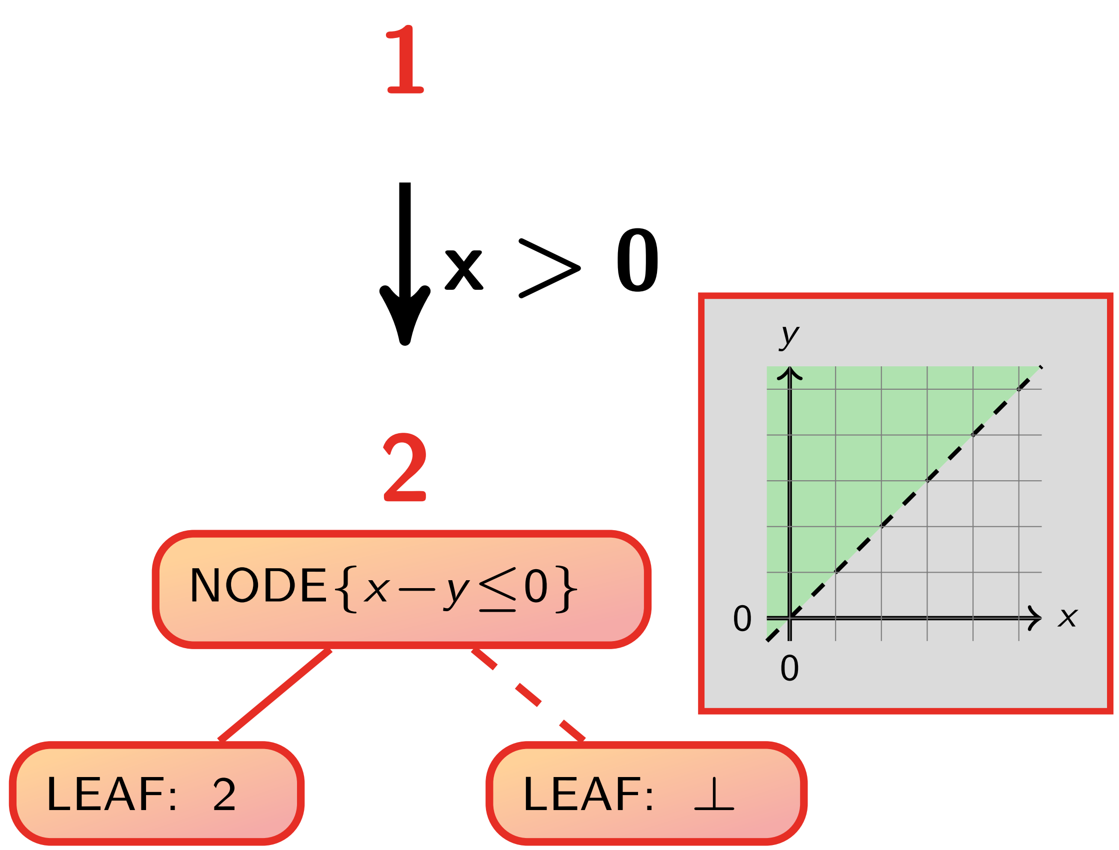


Tests

Algorithm 4 : Tree Filter

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

function FILTER(t,c)
  $C \leftarrow \text{FILTER}_\text{L}(c)$
  return AUGMENT(FILTER-AUX(t, c), C)

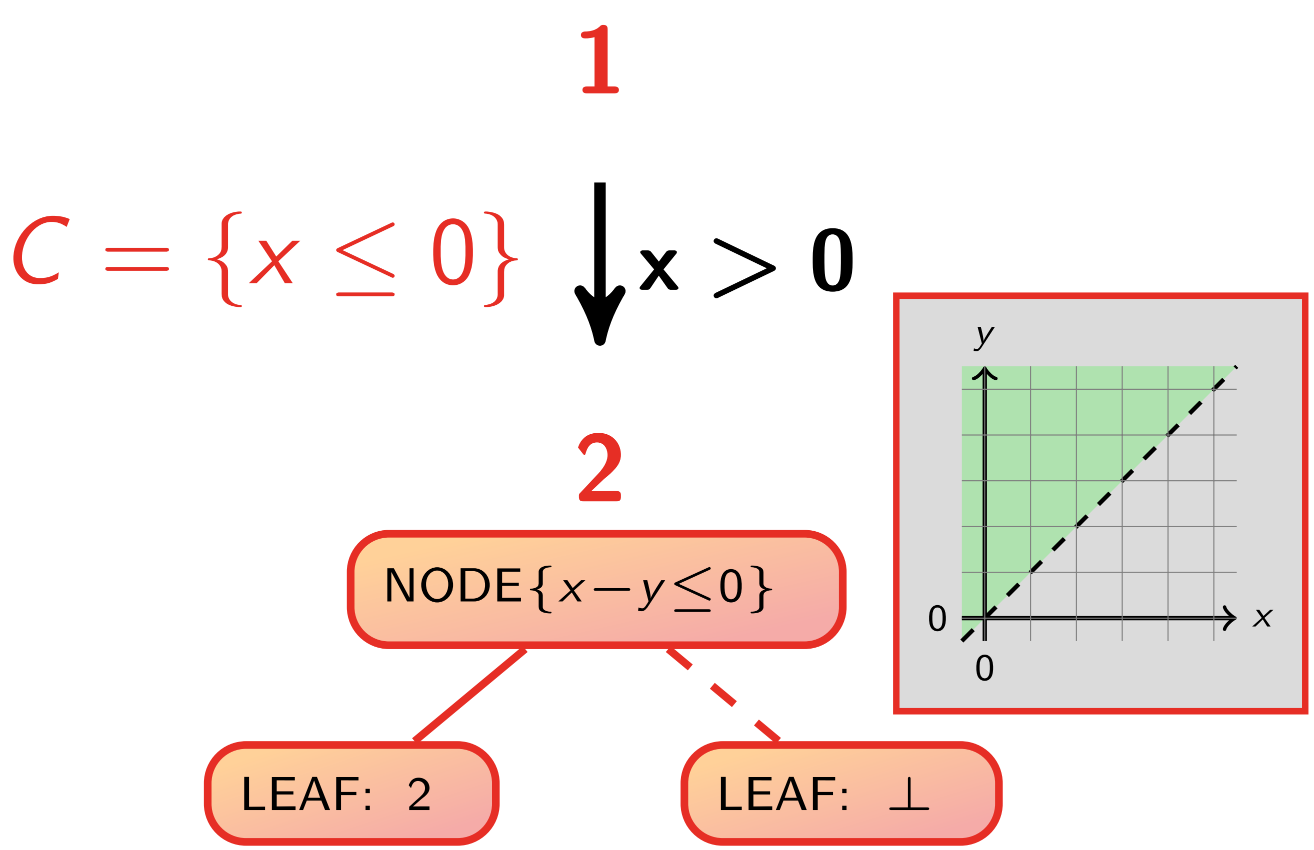


Tests

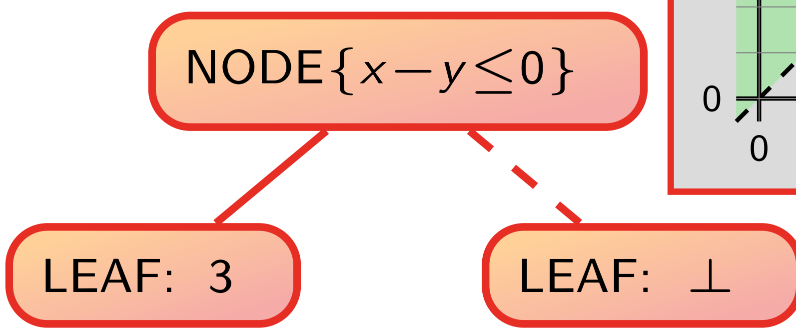
Algorithm 4 : Tree Filter

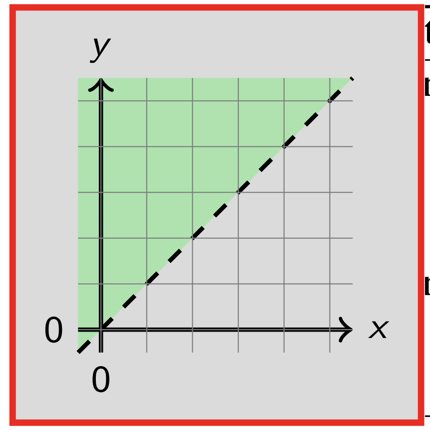
function FILTER-AUX(t,c)
  if ISLEAF(t) **then return** LEAF : FILTER$_F(f, c)$ $/* \; t \triangleq$ LEAF : $f \; */$
  else return NODE$\{t.c\}$: FILTER-AUX$(t.l, c)$; FILTER-AUX$(t.r, c)$

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



Tests





thm 4 : Tree Filter

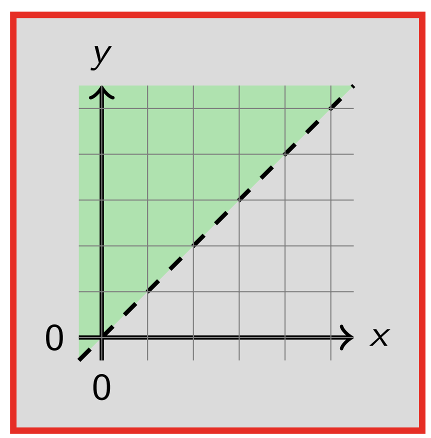
nction FILTER-AUX(t,c)
  if ISLEAF(t) **then return** LEAF : FILTER$_F(f, c)$ $/* \; t \triangleq$ LEAF : $f \; */$
  else return NODE$\{t.c\}$: FILTER-AUX$(t.l, c)$; FILTER-AUX$(t.r, c)$

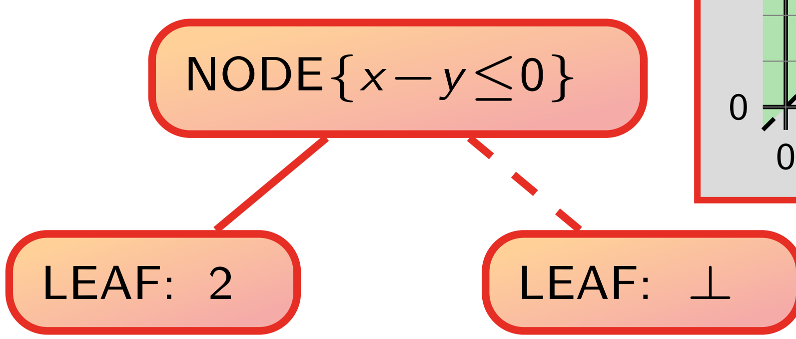
nction FILTER(t,c)
  $C \leftarrow$ FILTER$_L(c)$
  return AUGMENT(FILTER-AUX$(t, c), C)$

1

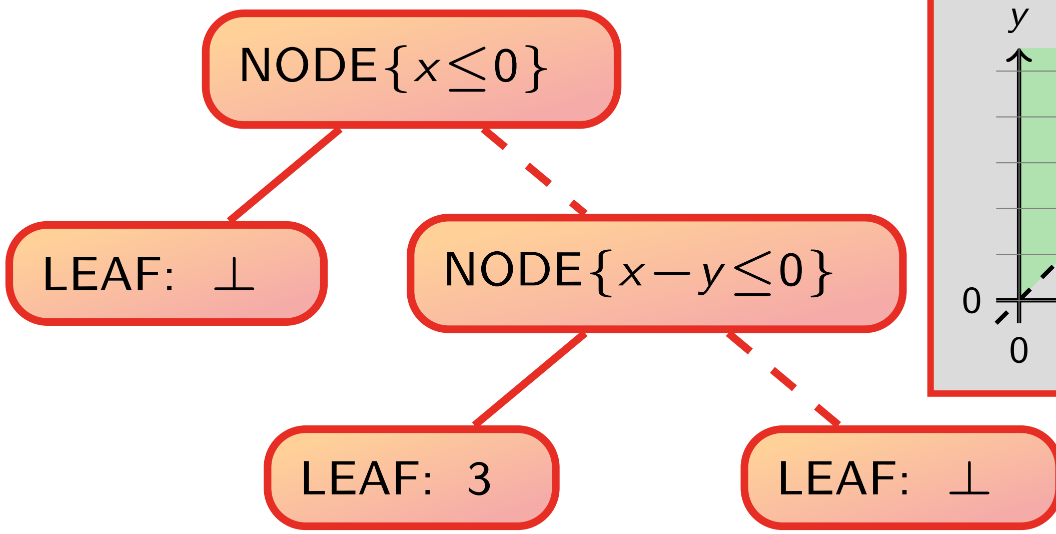
$C = \{x \leq 0\}$ $\downarrow$ $\mathbf{x > 0}$

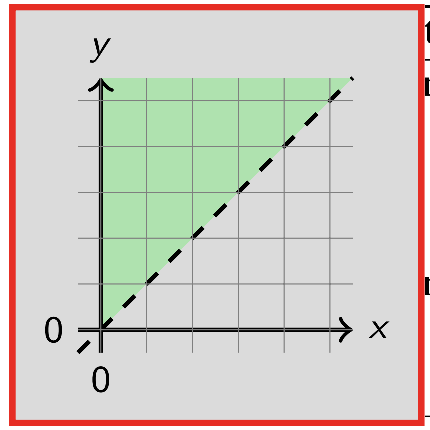
2





Tests





thm 4 : Tree Filter

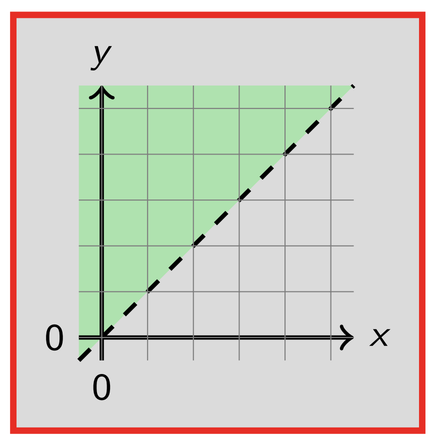
nction FILTER-AUX(t,c)
  if ISLEAF(t) **then return** LEAF : FILTER$_F(f, c)$ $/* \; t \triangleq$ LEAF : $f \; */$
  else return NODE$\{t.c\}$: FILTER-AUX$(t.l, c)$; FILTER-AUX$(t.r, c)$

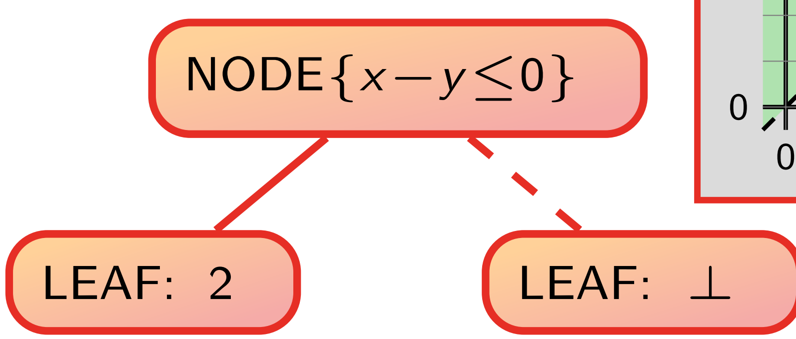
nction FILTER(t,c)
  $C \leftarrow$ FILTER$_L(c)$
  return AUGMENT(FILTER-AUX$(t, c), C)$

1

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

2





Example

int : x, y

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

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

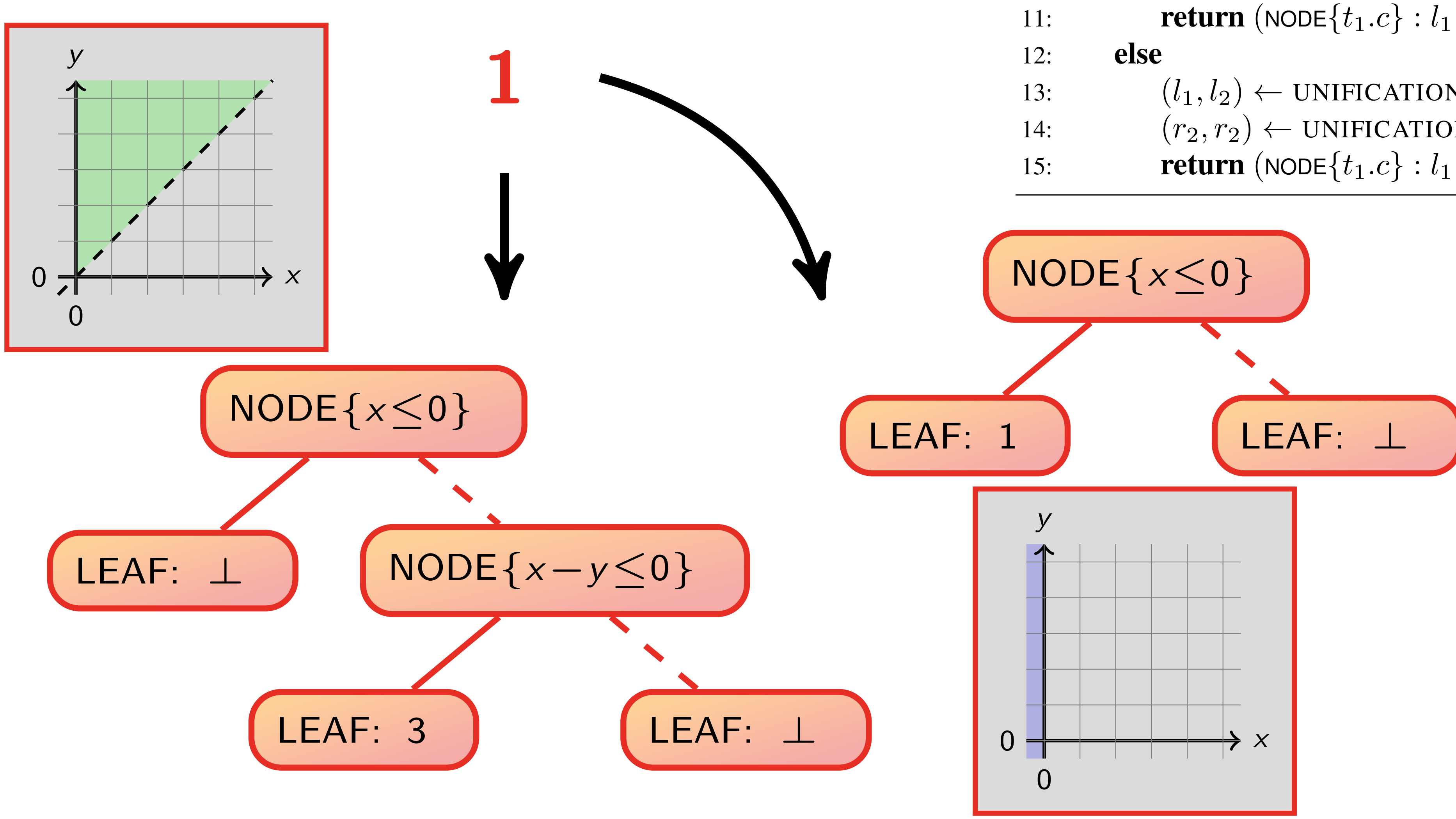
od^{3}

$$
\begin{tikzcd}
 & 1 \arrow[d, "\mathbf{x > 0}"] \arrow[dr, bend left, "\mathbf{x \leq 0}"] & \\
 & 2 \arrow[u, bend left, "x := x - y"] & 3
\end{tikzcd}
$$

Join

Algorithm 1 : Tree Unification

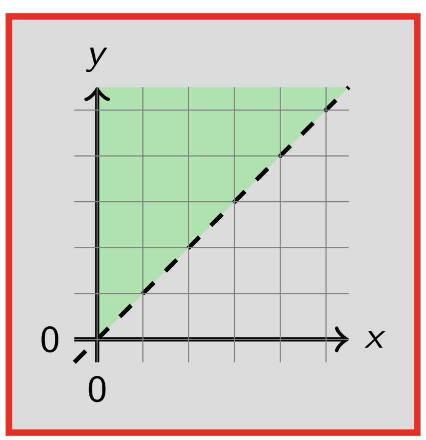
function UNIFICATION(t_1,t_2)
    if ISLEAF(t_1) $\wedge$ ISLEAF(t_2) **then**
        return $(t_1,\ t_2)$
    else if ISLEAF(t_1) $\vee$ (ISNODE(t_1) $\wedge$ ISNODE(t_2) $\wedge$ $t_2.c <_{\mathsf{L}} t_1.c$) **then**
        $(l_1, l_2) \leftarrow$ UNIFICATION($t_1, t_2.l$)
        $(r_1, r_2) \leftarrow$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

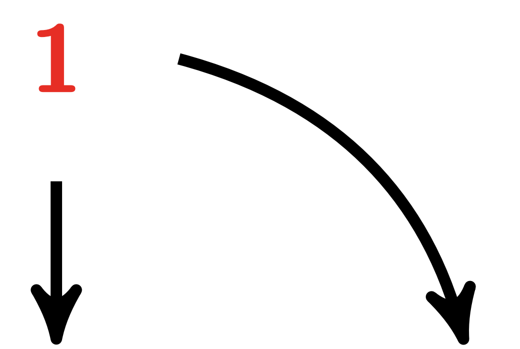


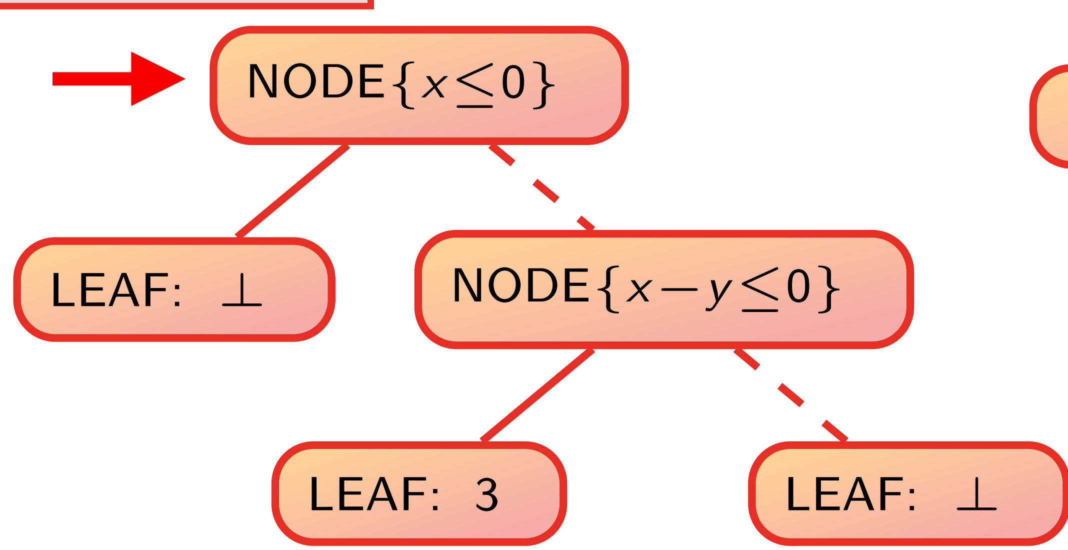
Join

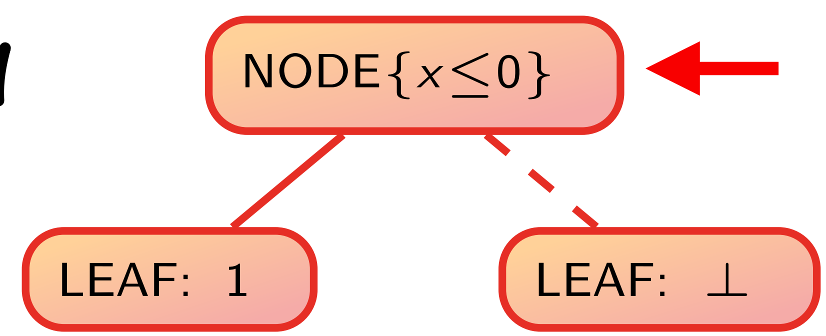
Algorithm 1 : Tree Unification

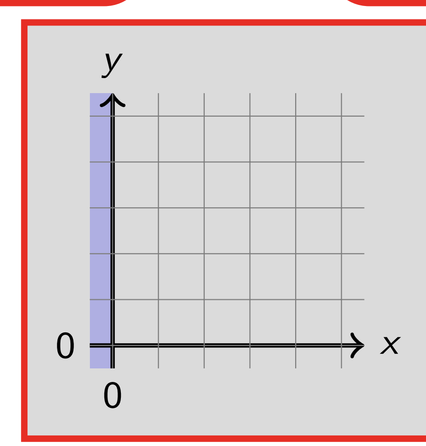
function UNIFICATION(t_1,t_2)
  if ISLEAF(t_1) $\wedge$ ISLEAF(t_2) **then**
    return $(t_1,\ t_2)$
  else if ISLEAF(t_1) $\vee$ (ISNODE(t_1) $\wedge$ ISNODE(t_2) $\wedge\ t_2.c <_{\mathsf{L}} t_1.c$) **then**
    $(l_1, l_2) \leftarrow$ UNIFICATION$(t_1, t_2.l)$
    $(r_1, r_2) \leftarrow$ UNIFICATION$(t_1,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





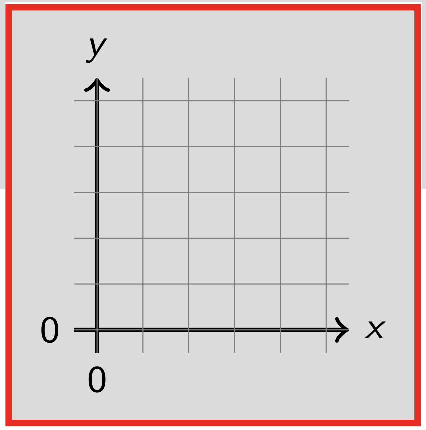






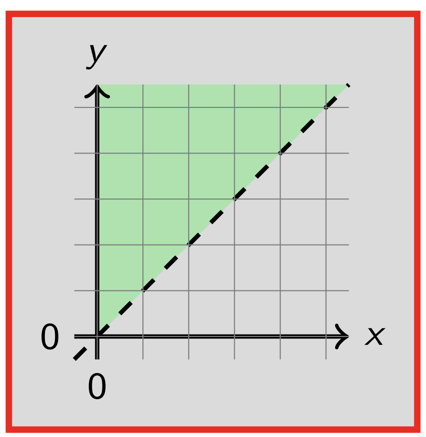
Join

NODE$\{x \leq 0\}$



Algorithm 1 : Tree Unification

```
function UNIFICATION(t1,t2)
    if ISLEAF(t1) ∧ ISLEAF(t2) then
        return (t1, t2)
    else if ISLEAF(t1) ∨ (ISNODE(t1) ∧ ISNODE(t2) ∧ t2.c <_L t1.c) then
        (l1, l2) ← UNIFICATION(t1, t2.l)
        (r1, r2) ← UNIFICATION(t1, t2.r)
        return (NODE{t2.c} : l1; r1, NODE{t2.c} : l2; r2)
    else if ISLEAF(t2) ∨ (ISNODE(t1) ∧ ISNODE(t2) ∧ t1.c <_L t2.c) then
        (l1, l2) ← UNIFICATION(t1.l, t2)
        (r1, r2) ← UNIFICATION(t1.r, t2)
        return (NODE{t1.c} : l1; r1, NODE{t1.c} : l2; r2)
    else
        (l1, l2) ← UNIFICATION(t1.l, t2.l)
        (r2, r2) ← UNIFICATION(t1.r, t2.r)
        return (NODE{t1.c} : l1; r1, NODE{t2.c} : l2; r2)
```



1

NODE$\{x \leq 0\}$

LEAF: $\perp$

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

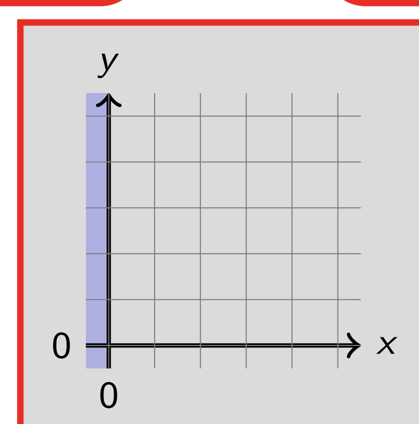
LEAF: 3

LEAF: $\perp$

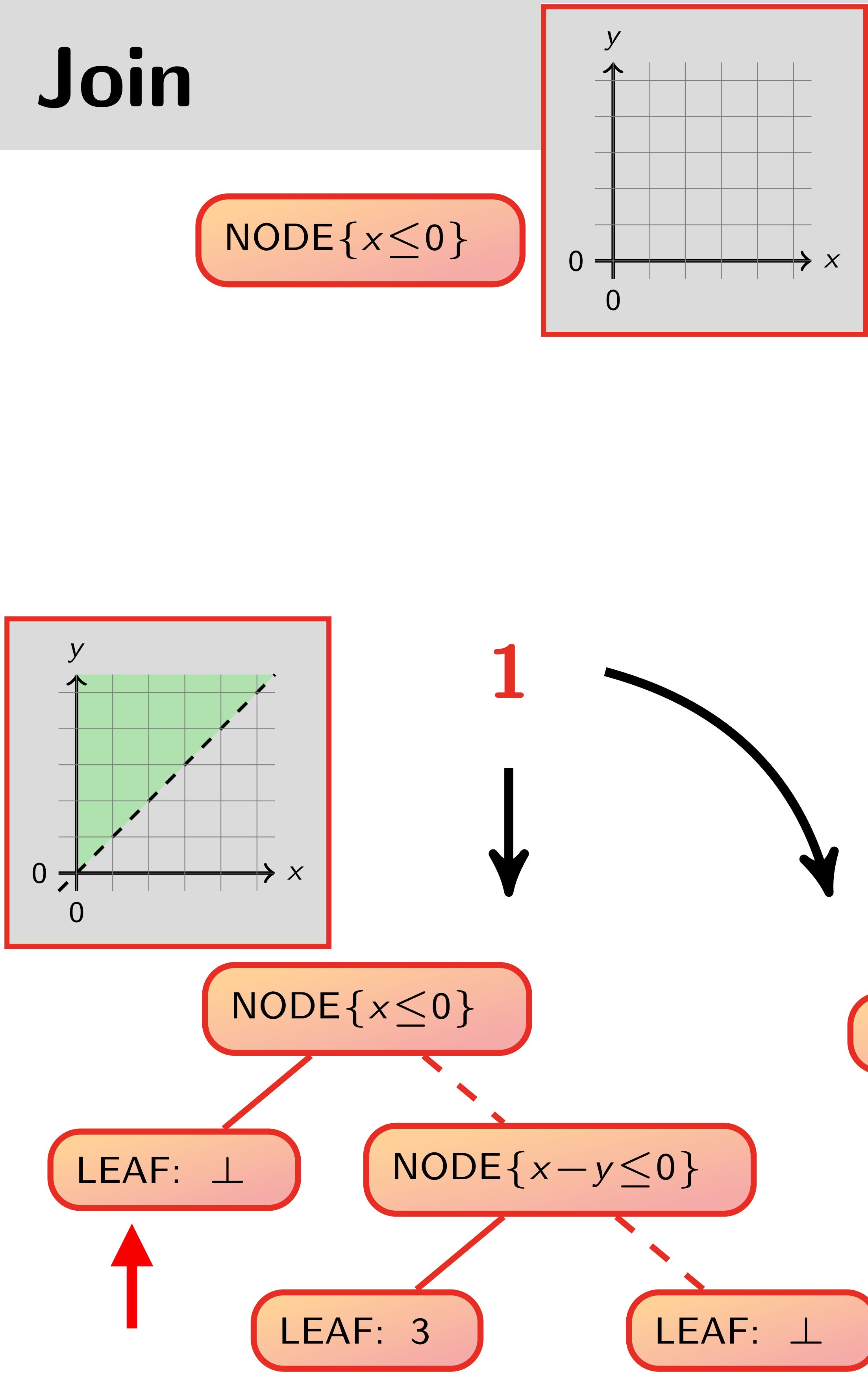
NODE$\{x \leq 0\}$

LEAF: 1

LEAF: $\perp$

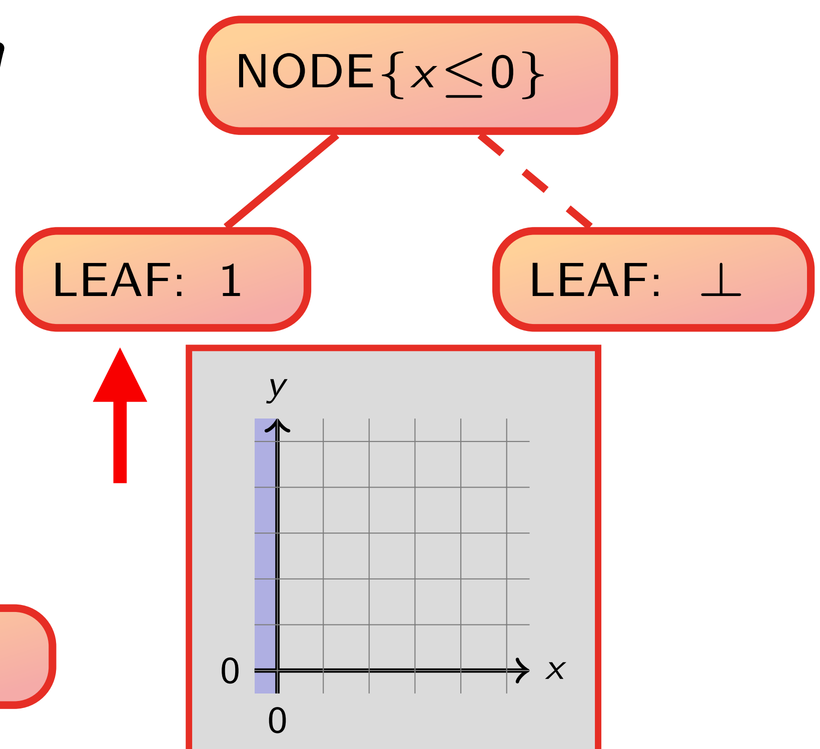


Join

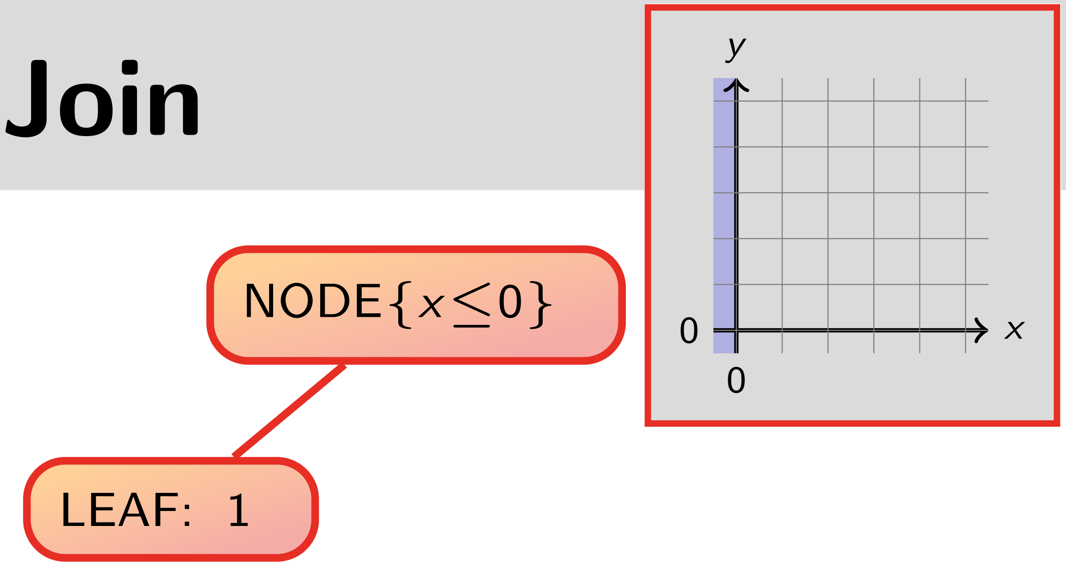


Algorithm 1 : Tree Unification

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

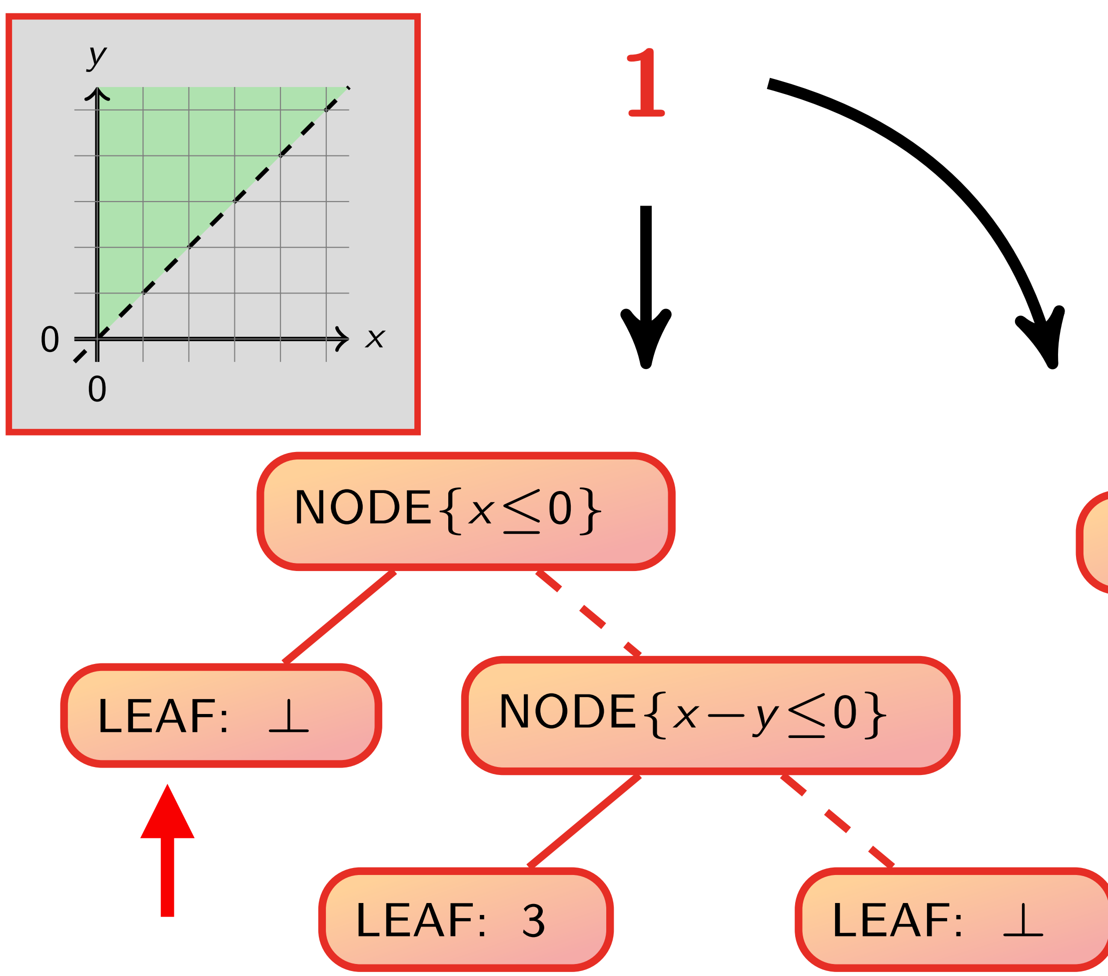


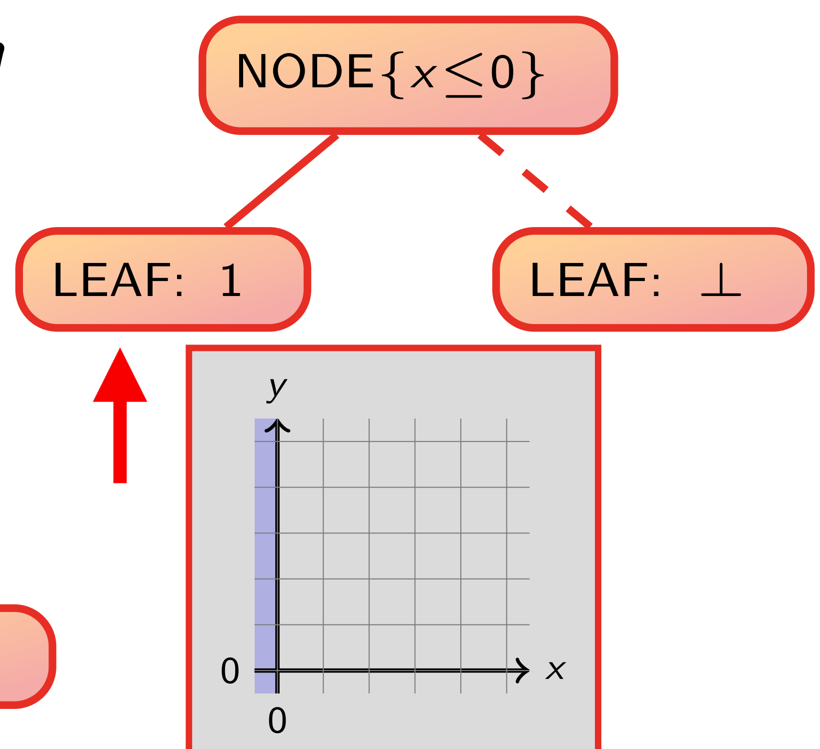
Join



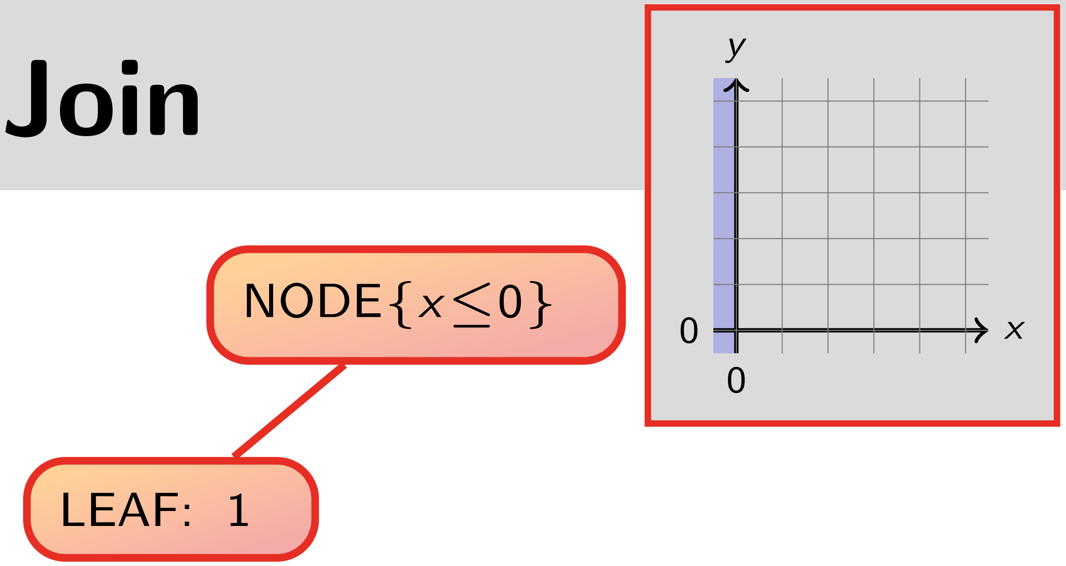
Algorithm 1 : Tree Unification

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



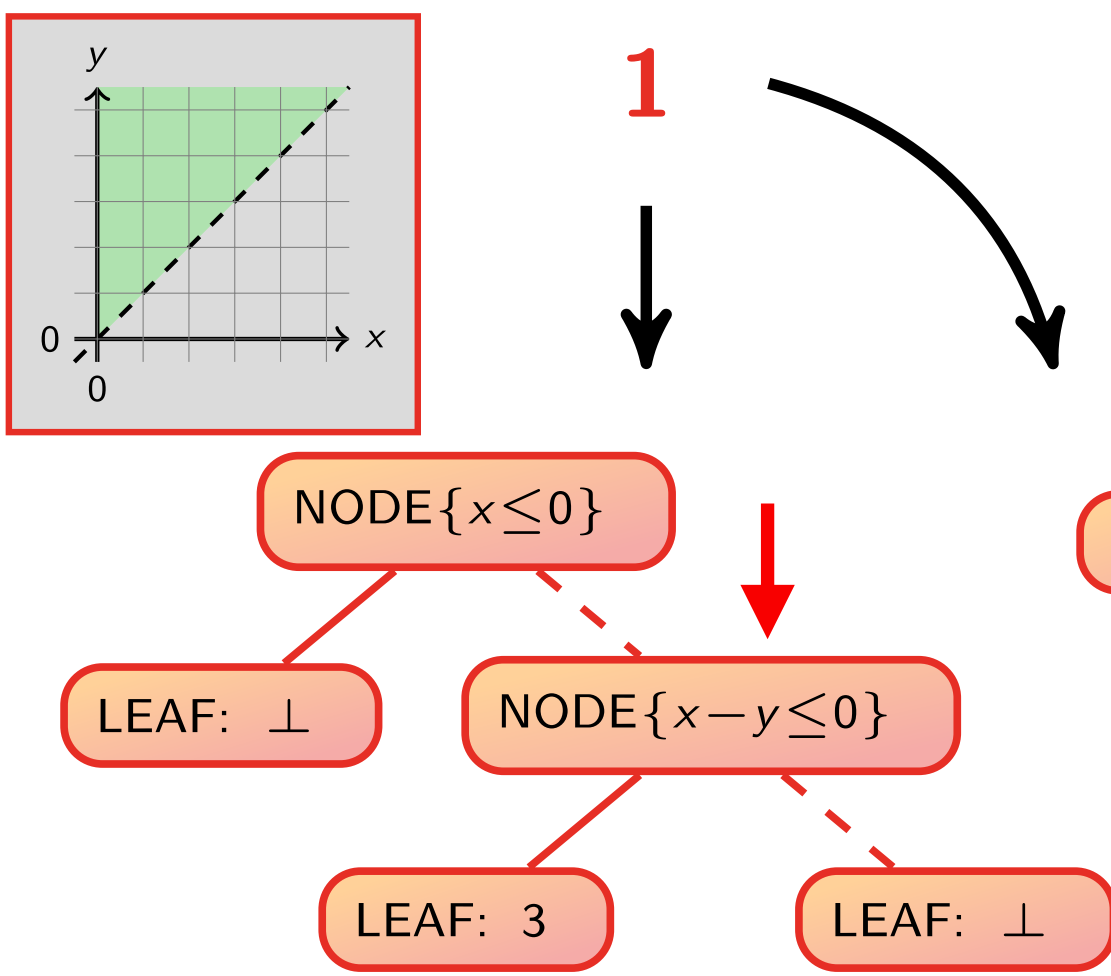


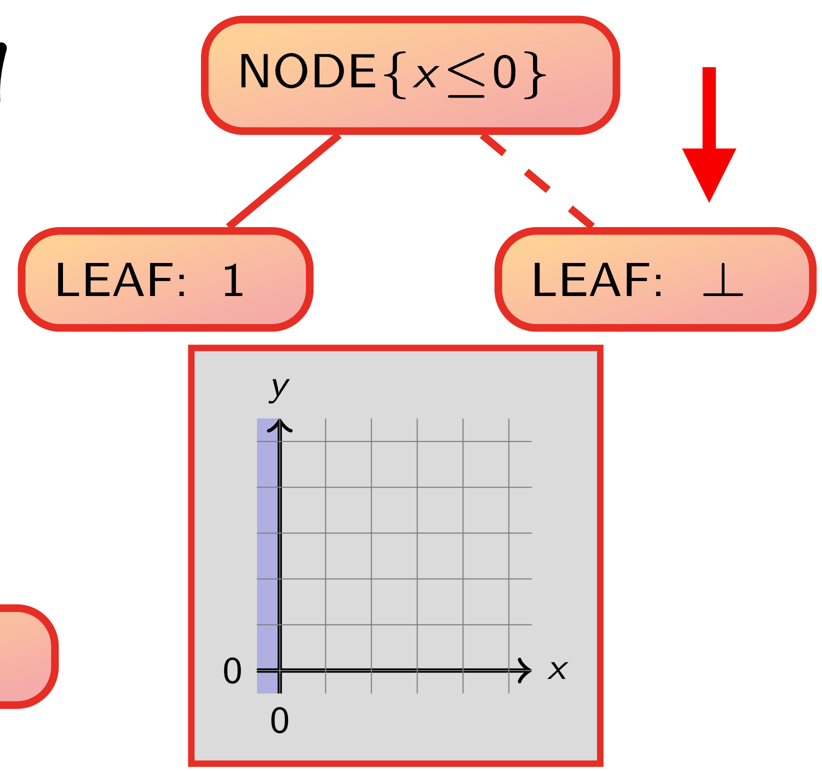
Join



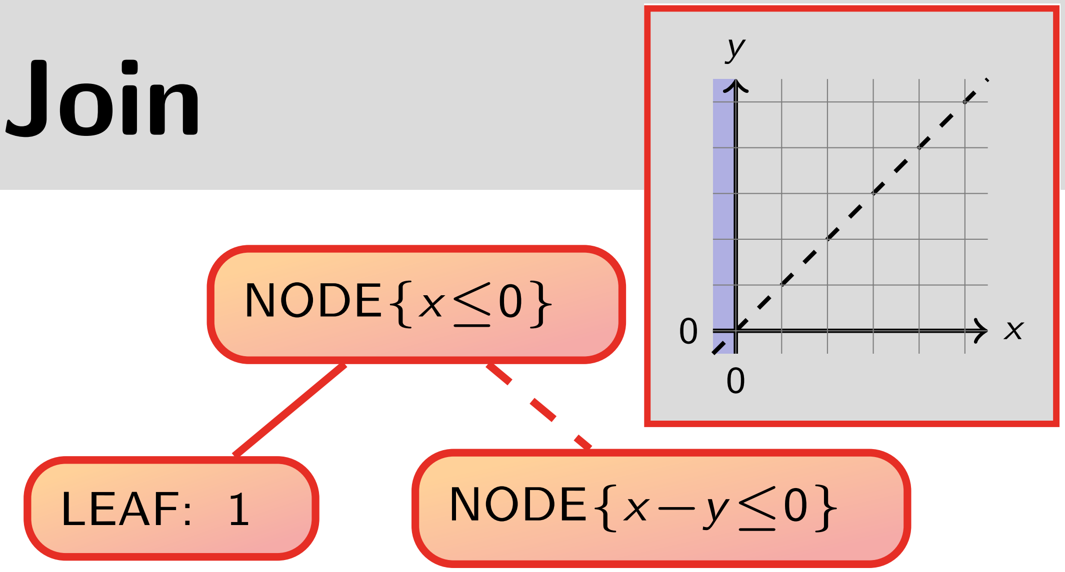
Algorithm 1 : Tree Unification

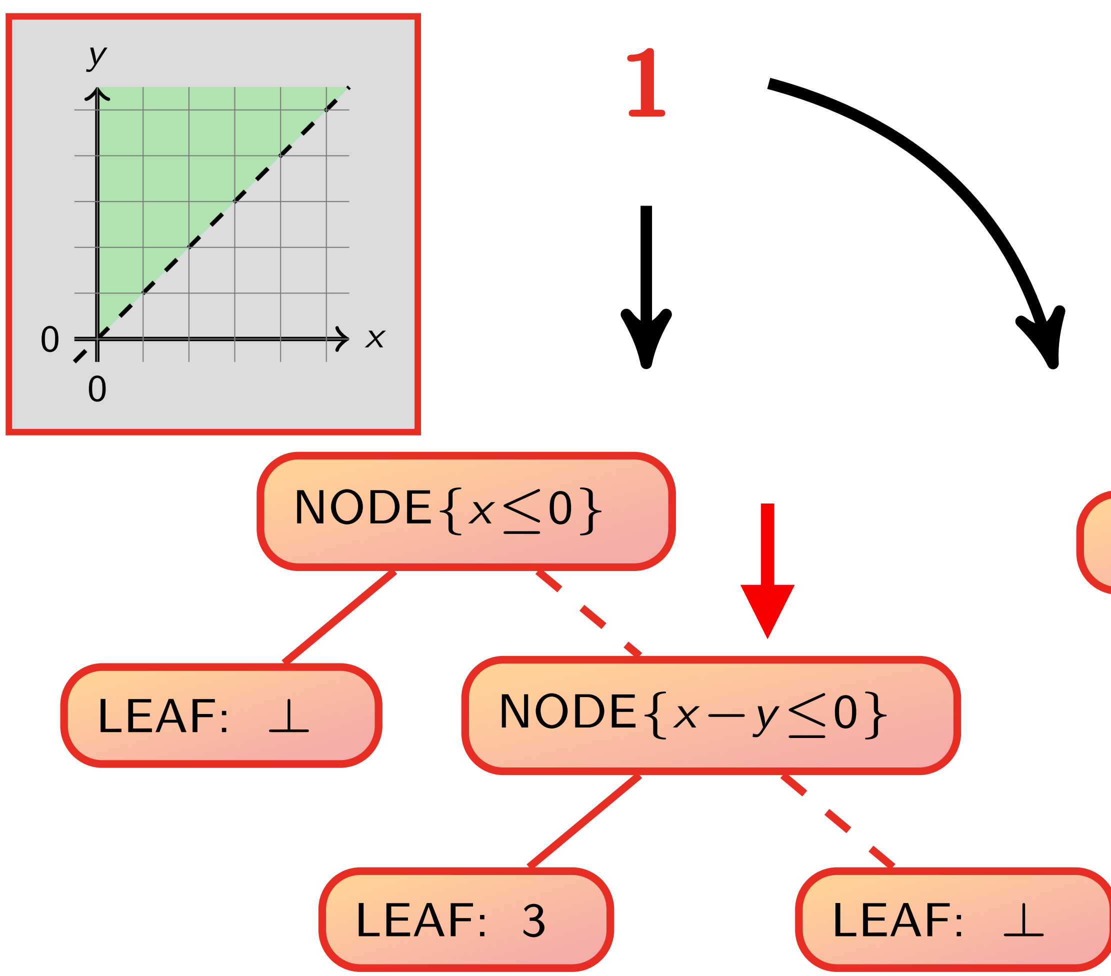
function UNIFICATION(t_1,t_2)
    if ISLEAF(t_1) $\wedge$ ISLEAF(t_2) **then**
        return $(t_1,\ t_2)$
    else if ISLEAF(t_1) $\vee$ (ISNODE(t_1) $\wedge$ ISNODE(t_2) $\wedge$ $t_2.c <_{\mathsf{L}} t_1.c$) **then**
        $(l_1, l_2) \leftarrow$ UNIFICATION($t_1, t_2.l$)
        $(r_1, r_2) \leftarrow$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





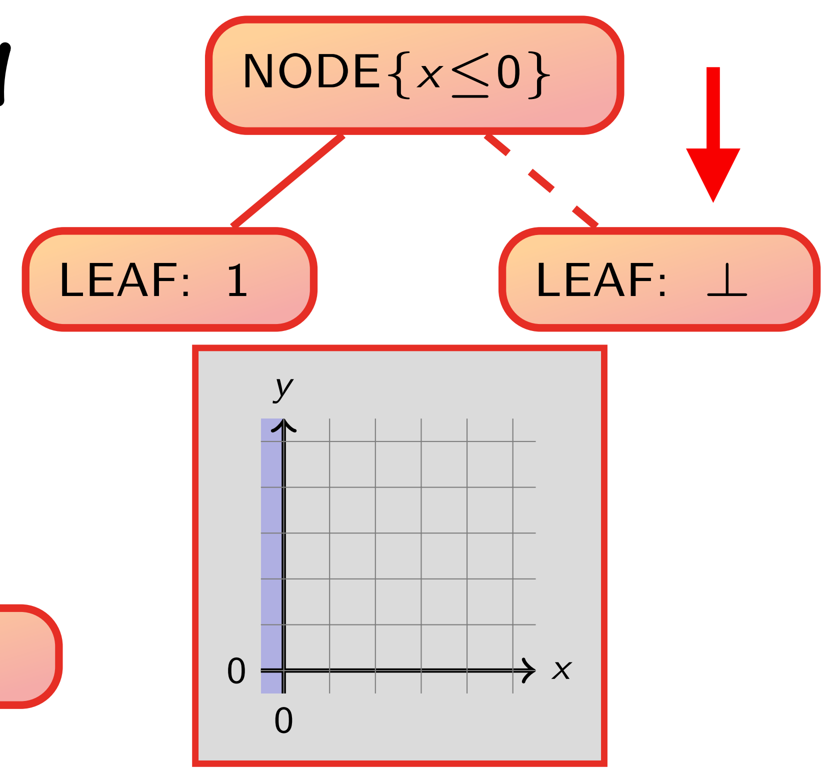
Join



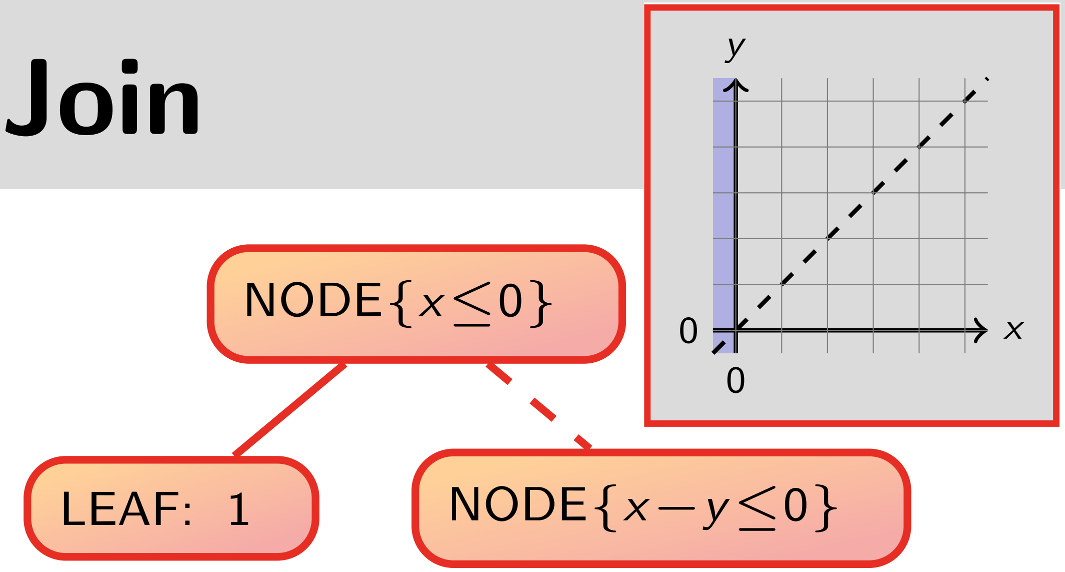


Algorithm 1 : Tree Unification

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

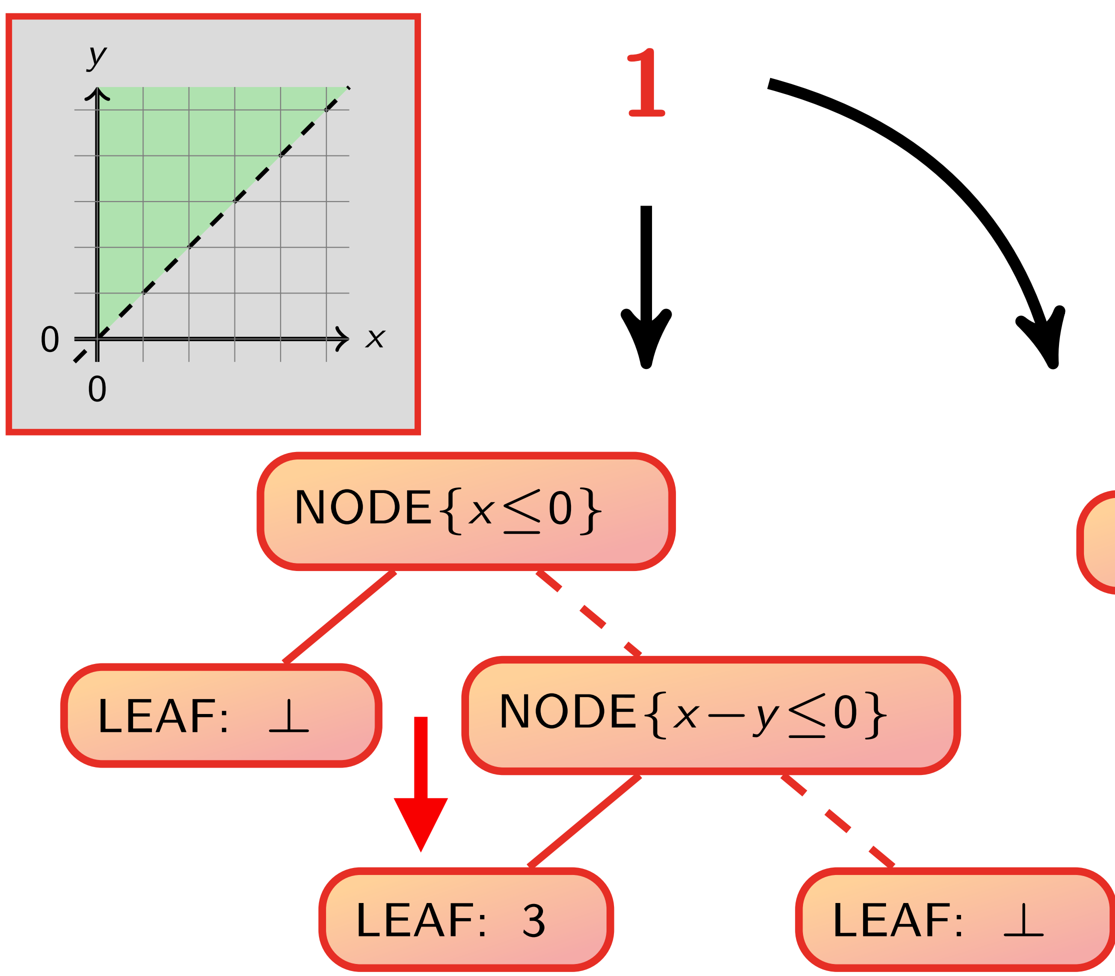


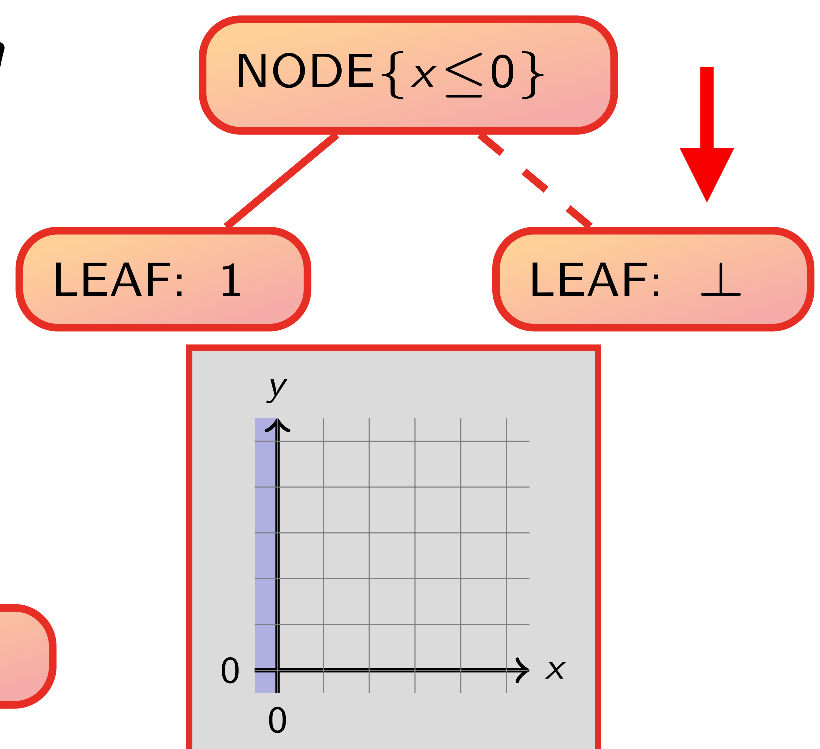
Join



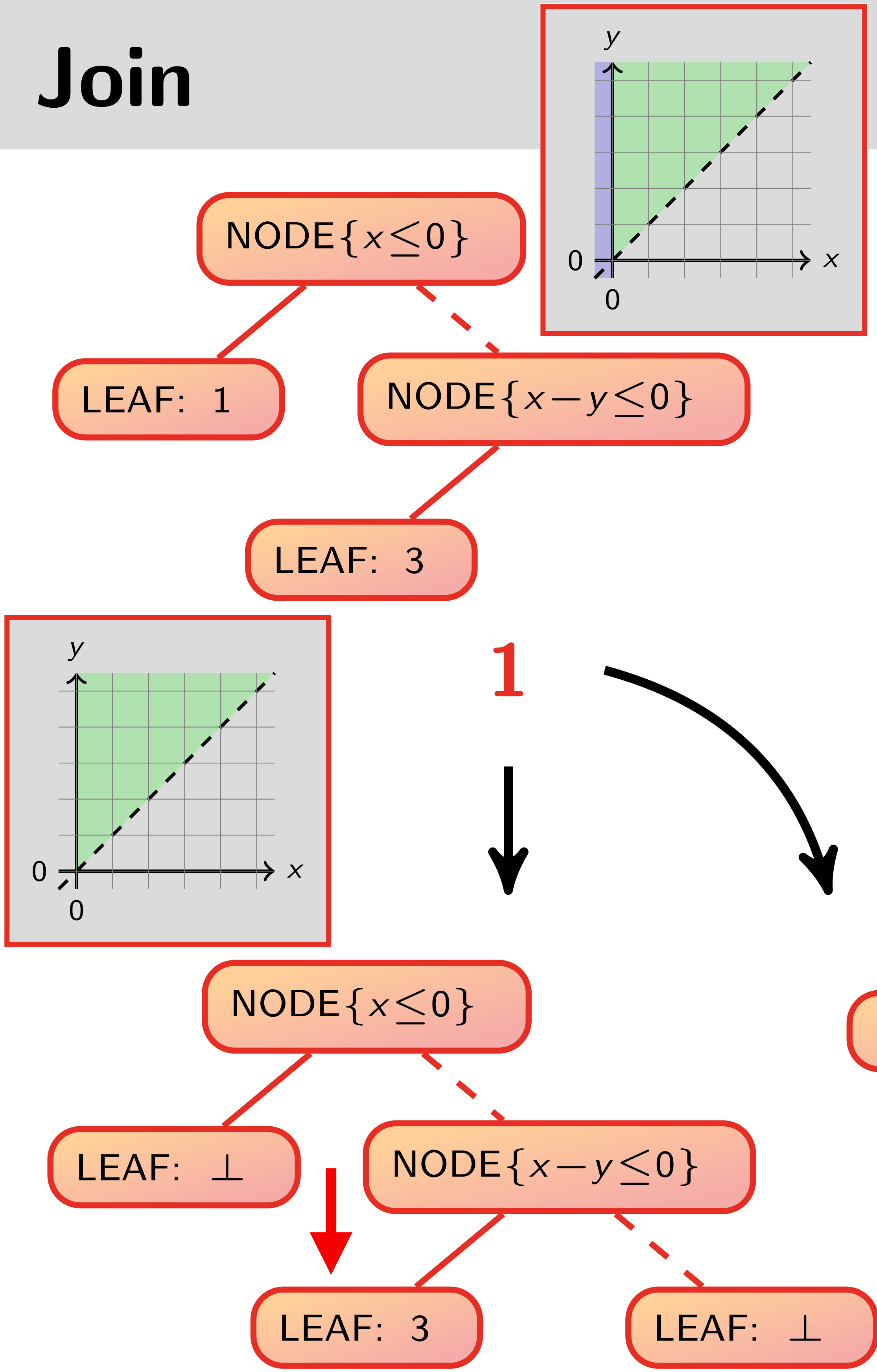
Algorithm 1 : Tree Unification

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



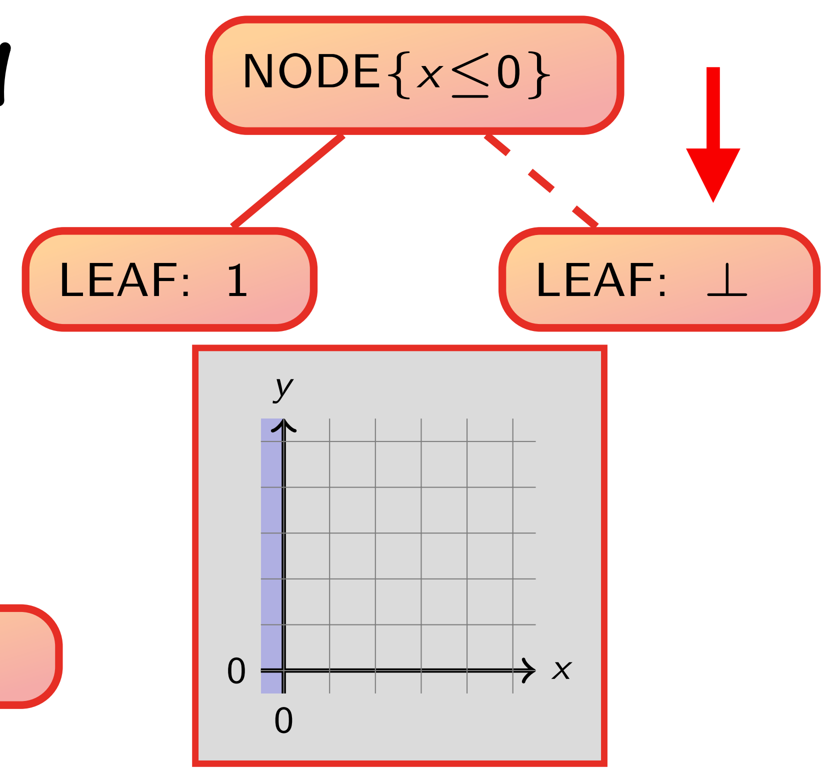


Join

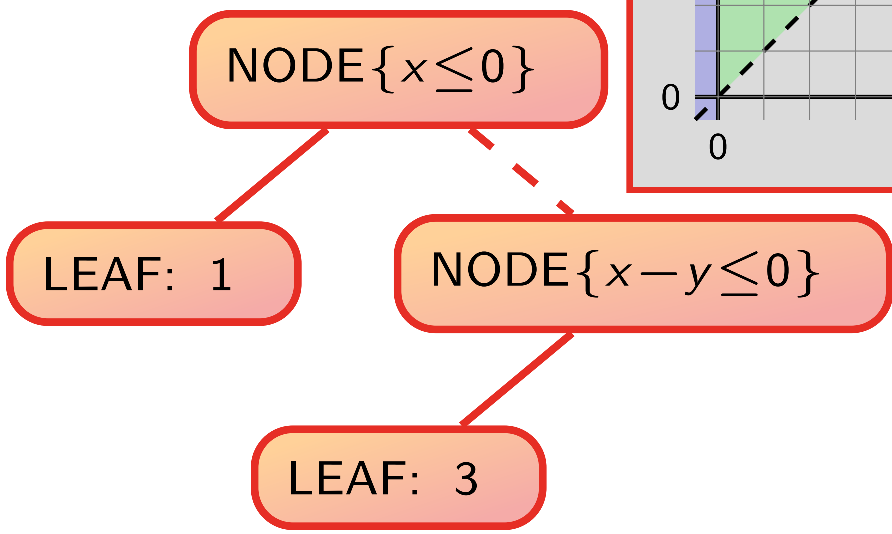


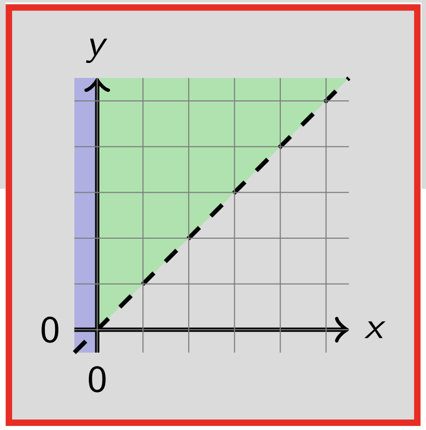
Algorithm 1 : Tree Unification

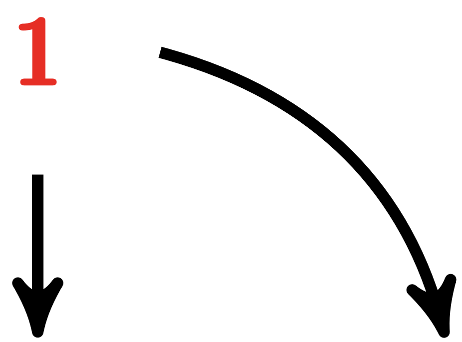
function UNIFICATION(t_1,t_2)
  if ISLEAF(t_1) $\wedge$ ISLEAF(t_2) **then**
    return $(t_1,\ t_2)$
  else if ISLEAF(t_1) $\vee$ (ISNODE(t_1) $\wedge$ ISNODE(t_2) $\wedge$ $t_2.c <_{\mathrm{L}} t_1.c$) **then**
    $(l_1, l_2) \leftarrow$ UNIFICATION($t_1, t_2.l$)
    $(r_1, r_2) \leftarrow$ UNIFICATION($t_1,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

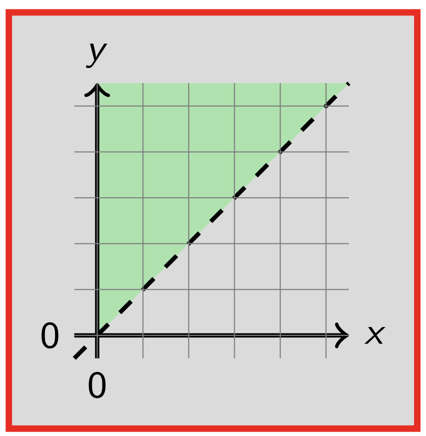


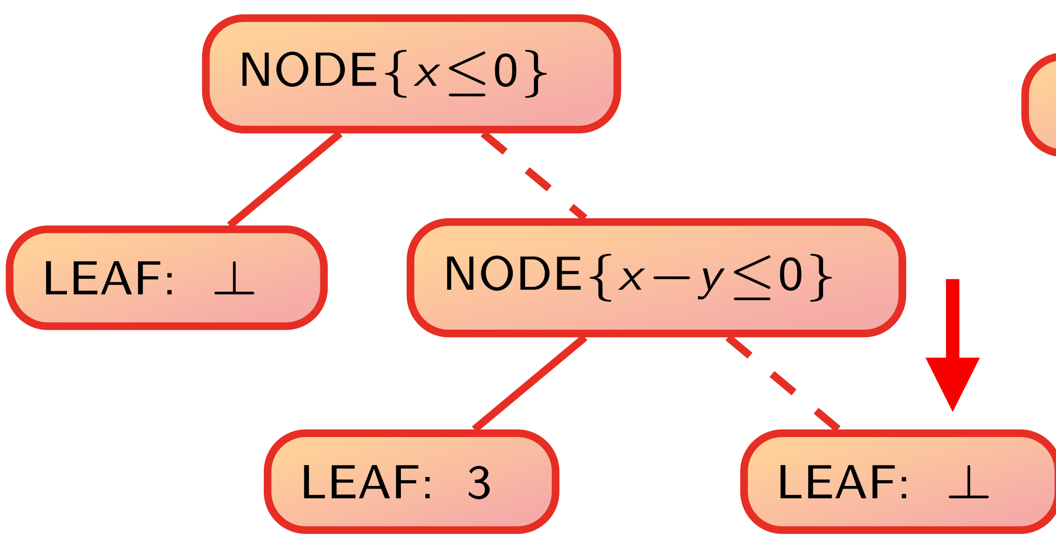
Join





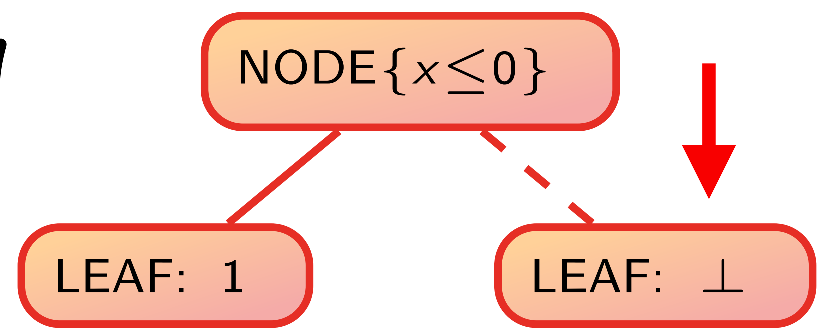


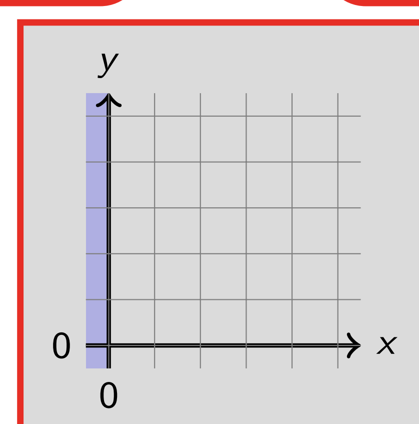




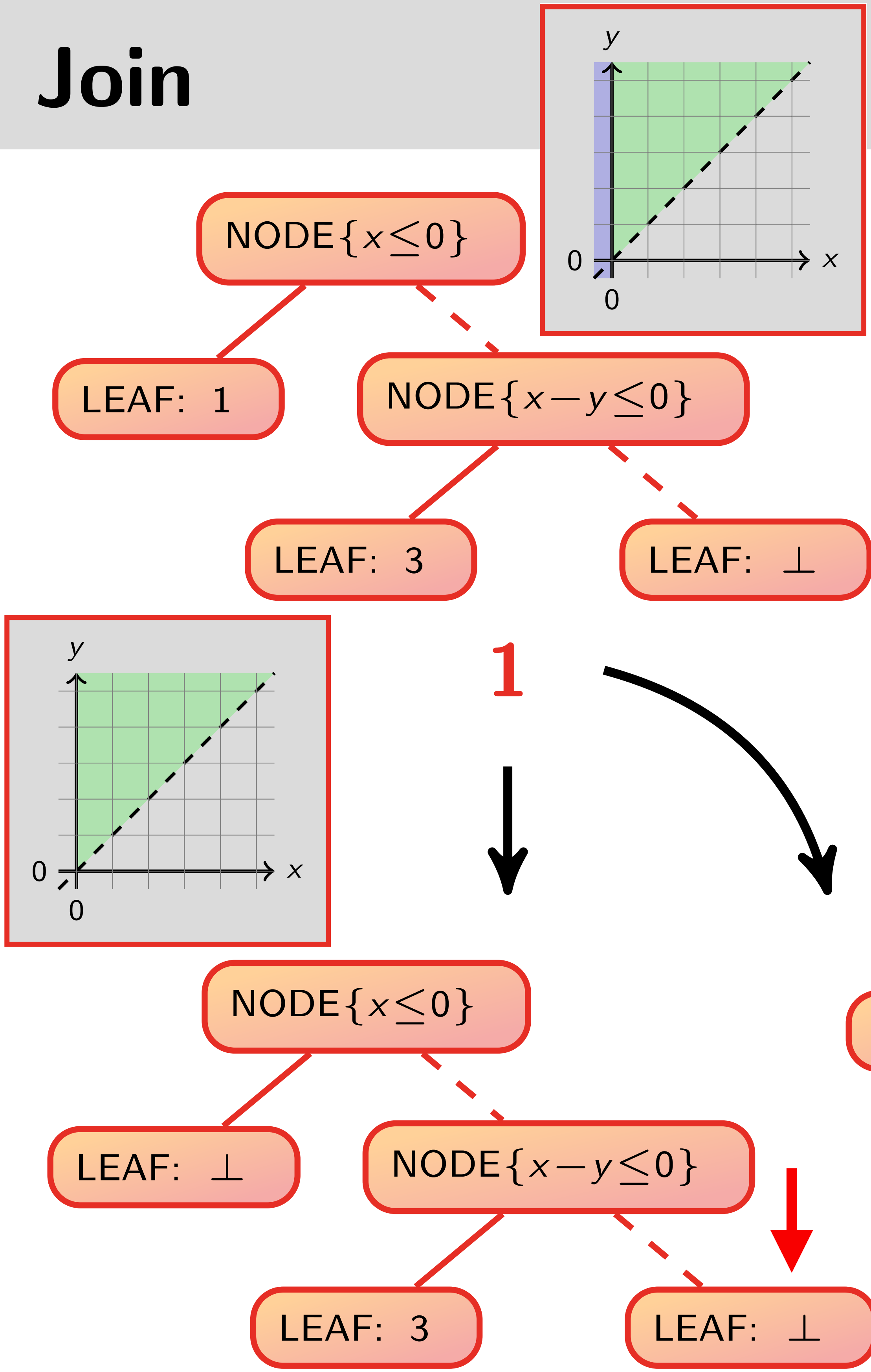
Algorithm 1 : Tree Unification

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



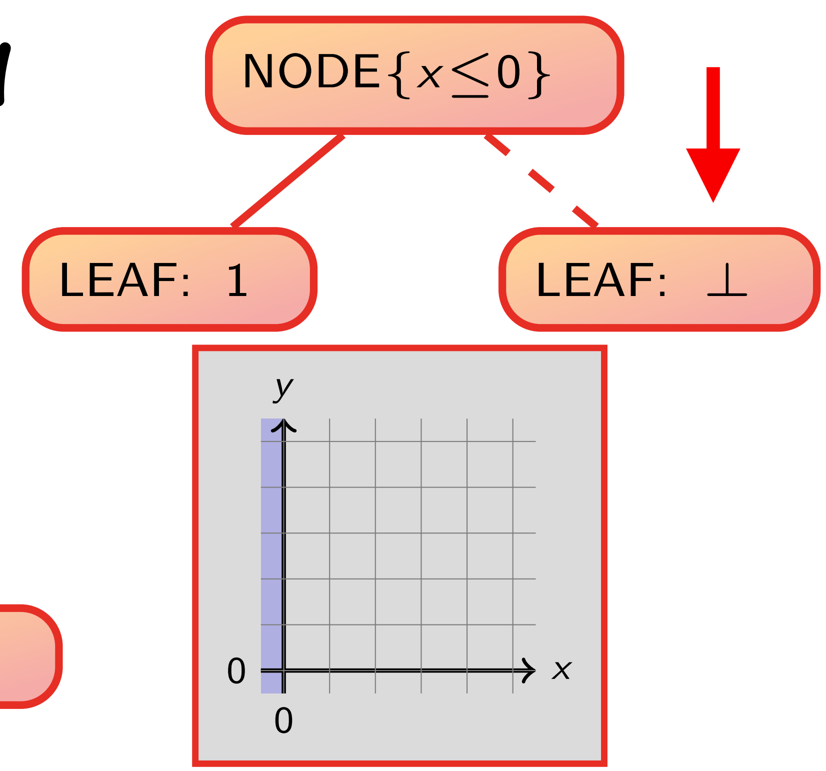


Join

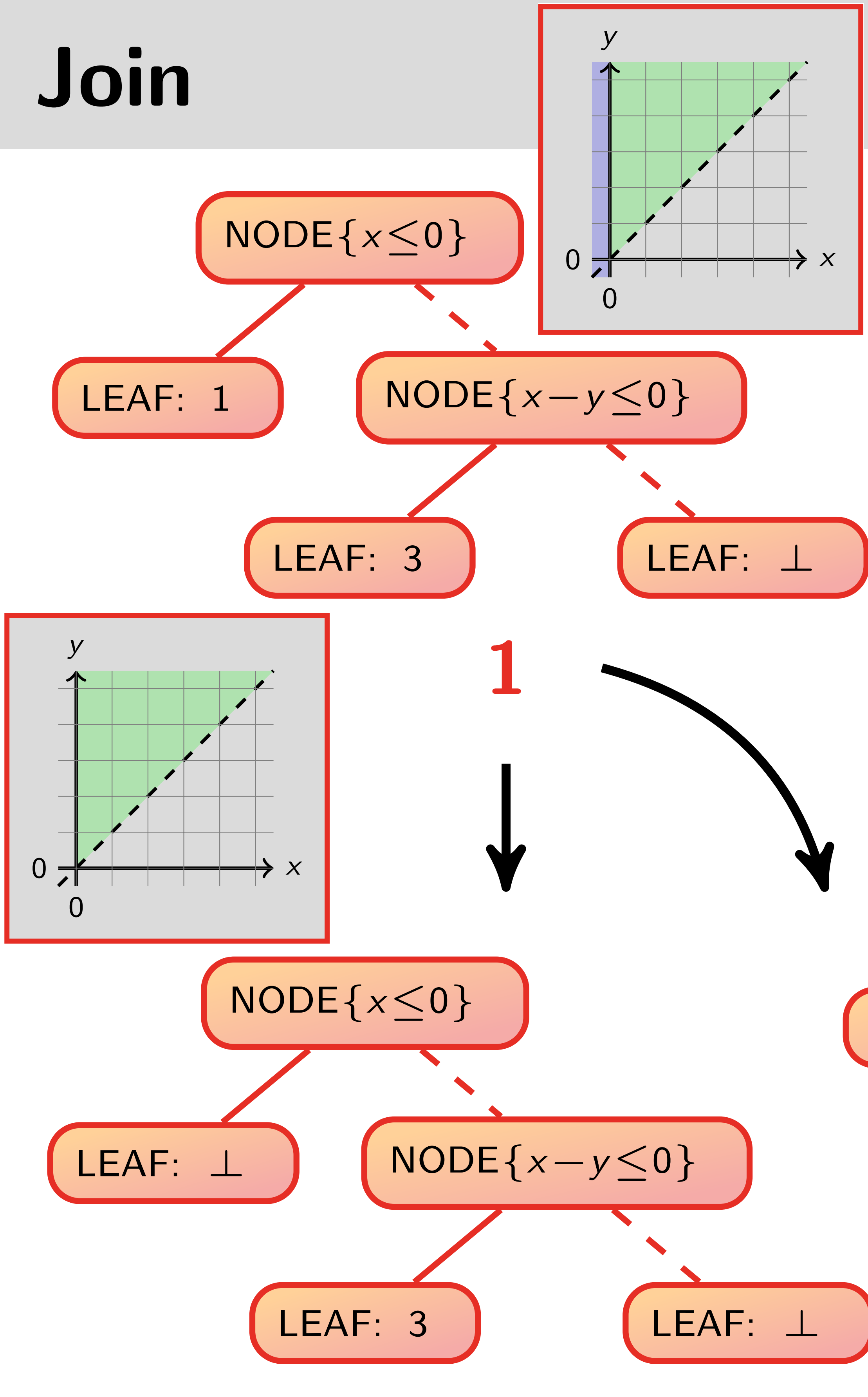


Algorithm 1 : Tree Unification

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

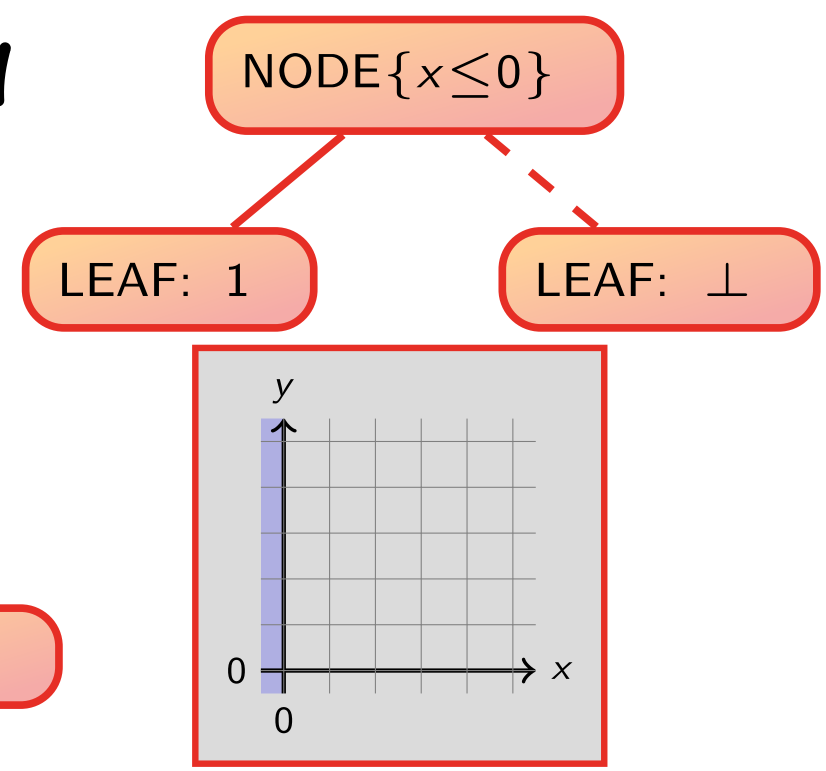


Join



Algorithm 1 : Tree Unification

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



Example

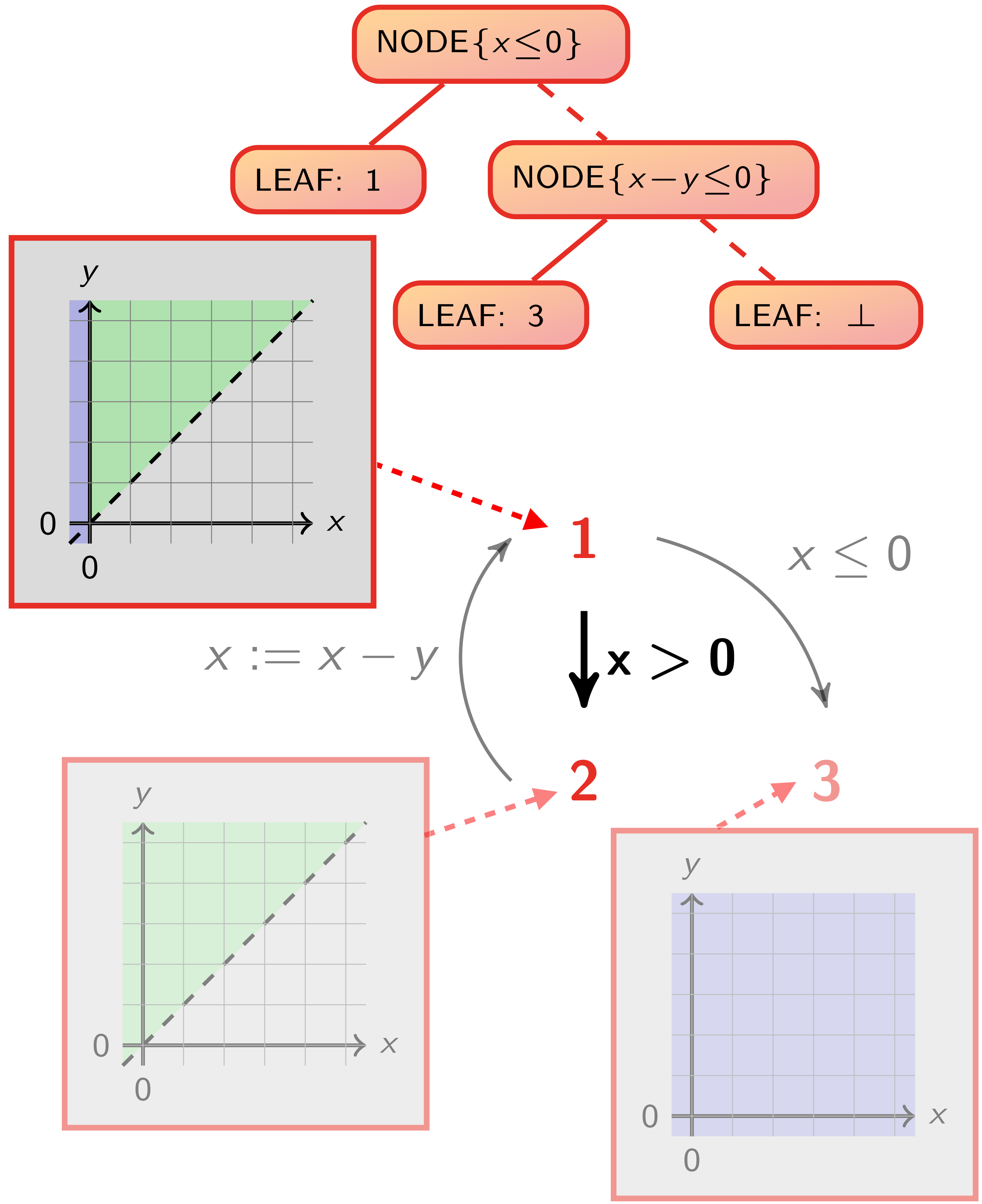
int : x, y

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

[2]$x := x - y$

od[3]

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



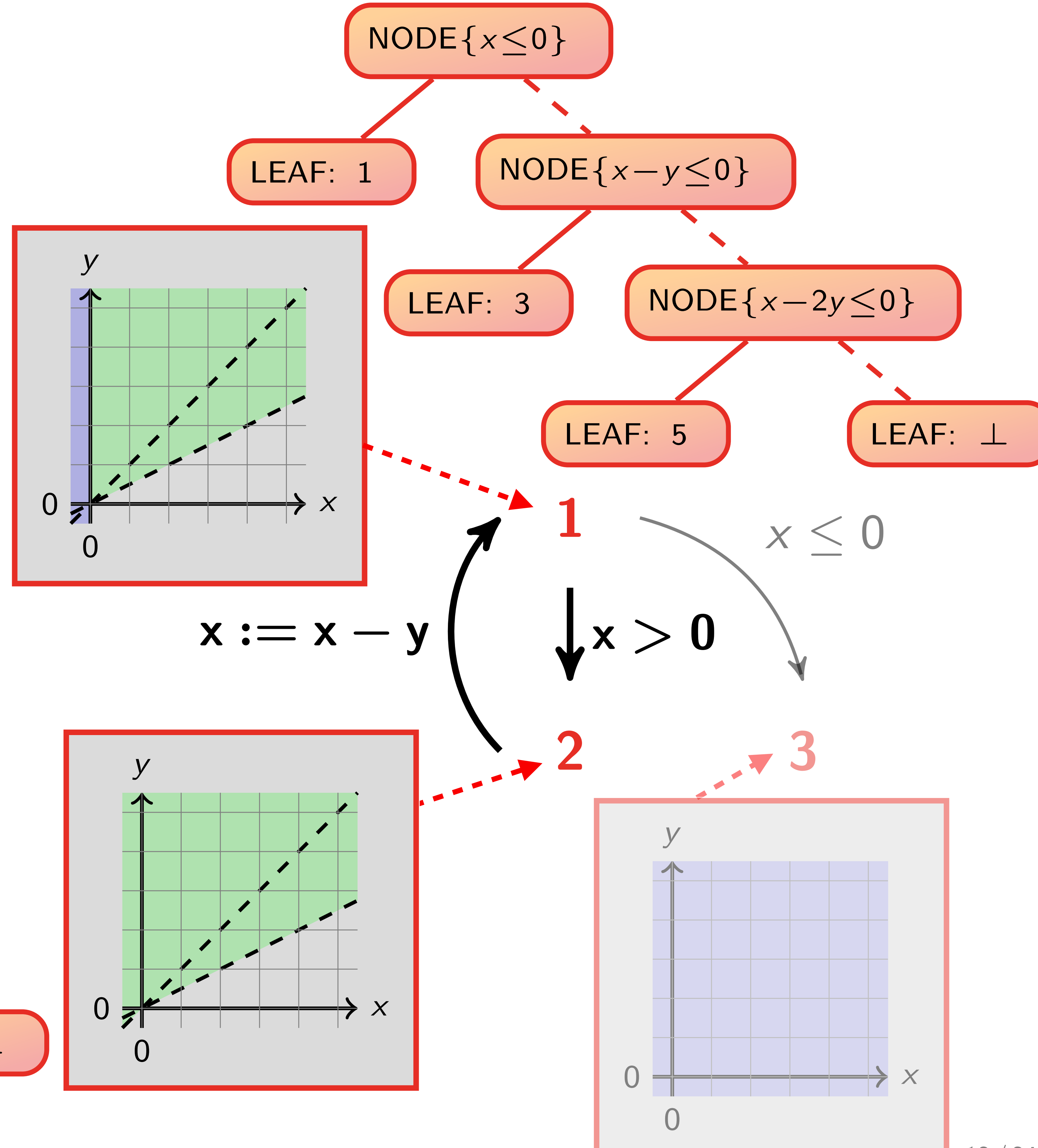
Example

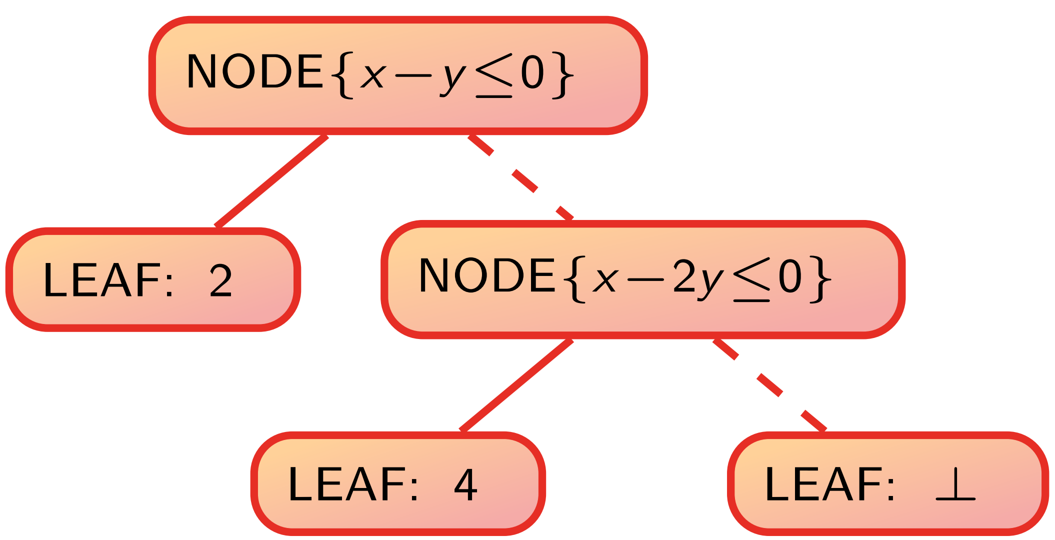
int : x, y

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

[2]$x := x - y$

od[3]

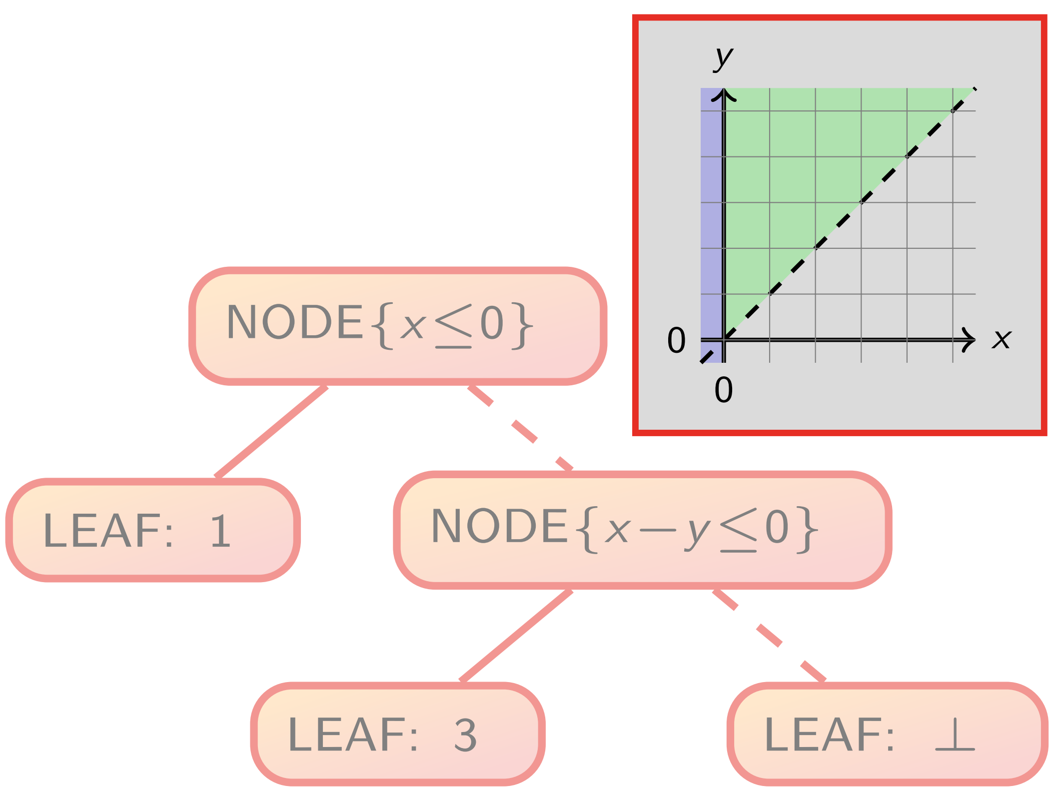


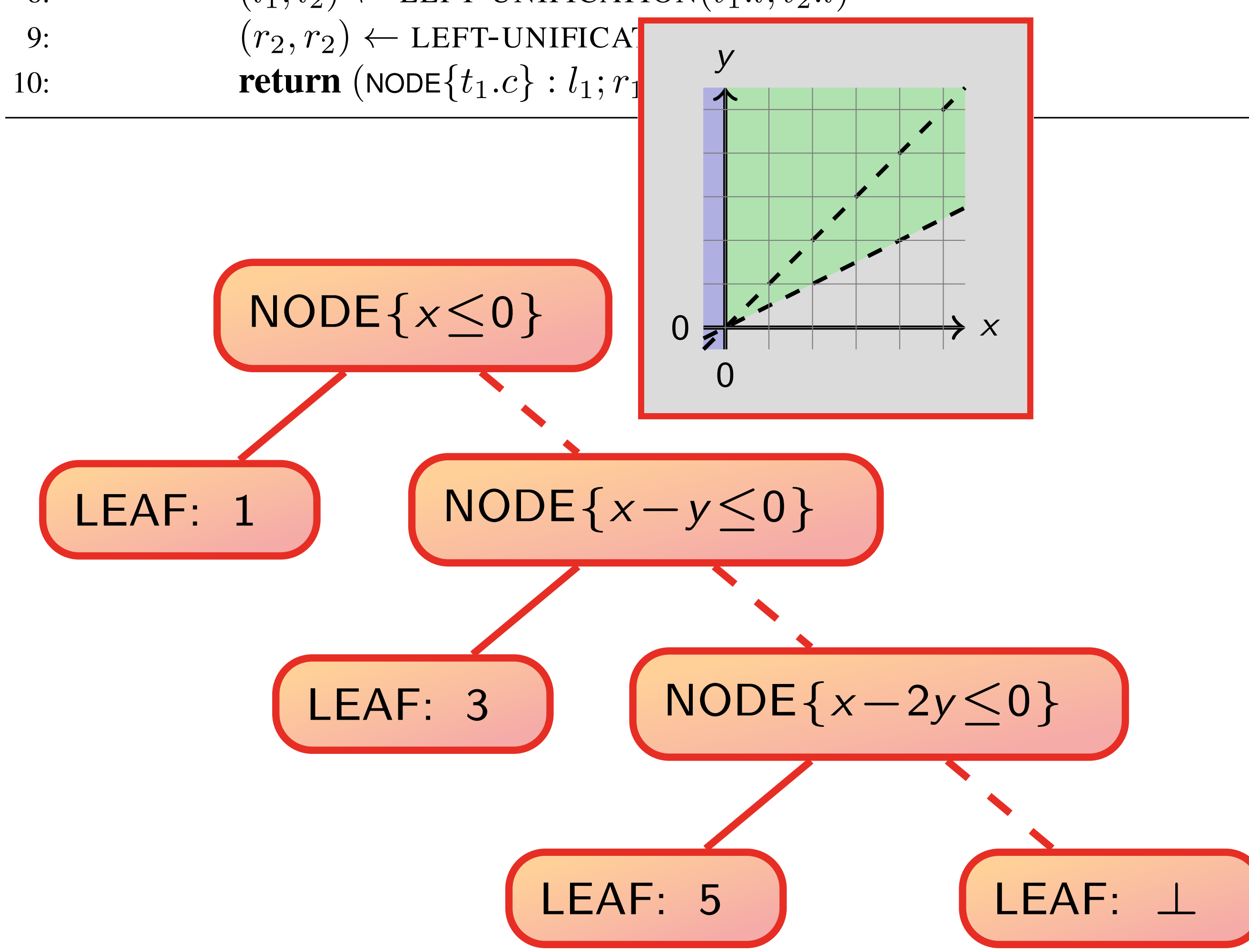


Widening: Left Unification

Algorithm 5 : Tree Left Unification

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

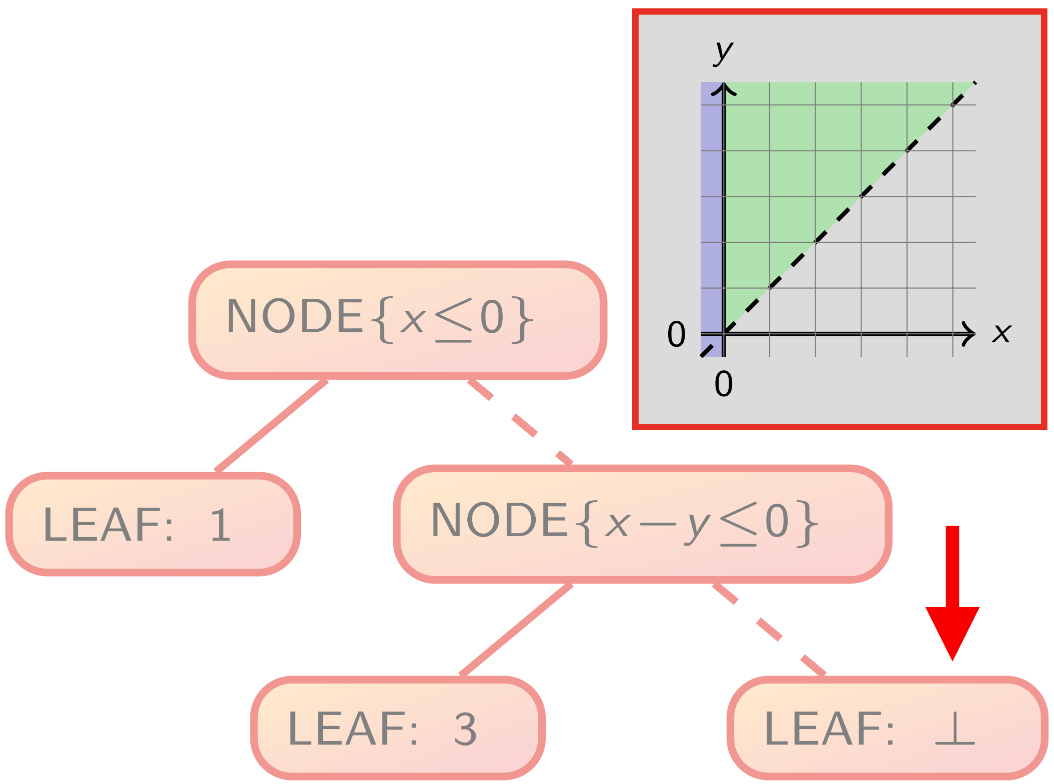


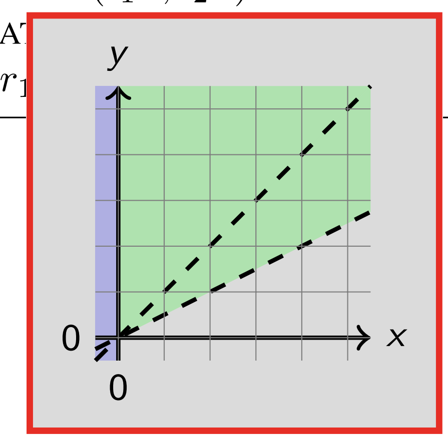


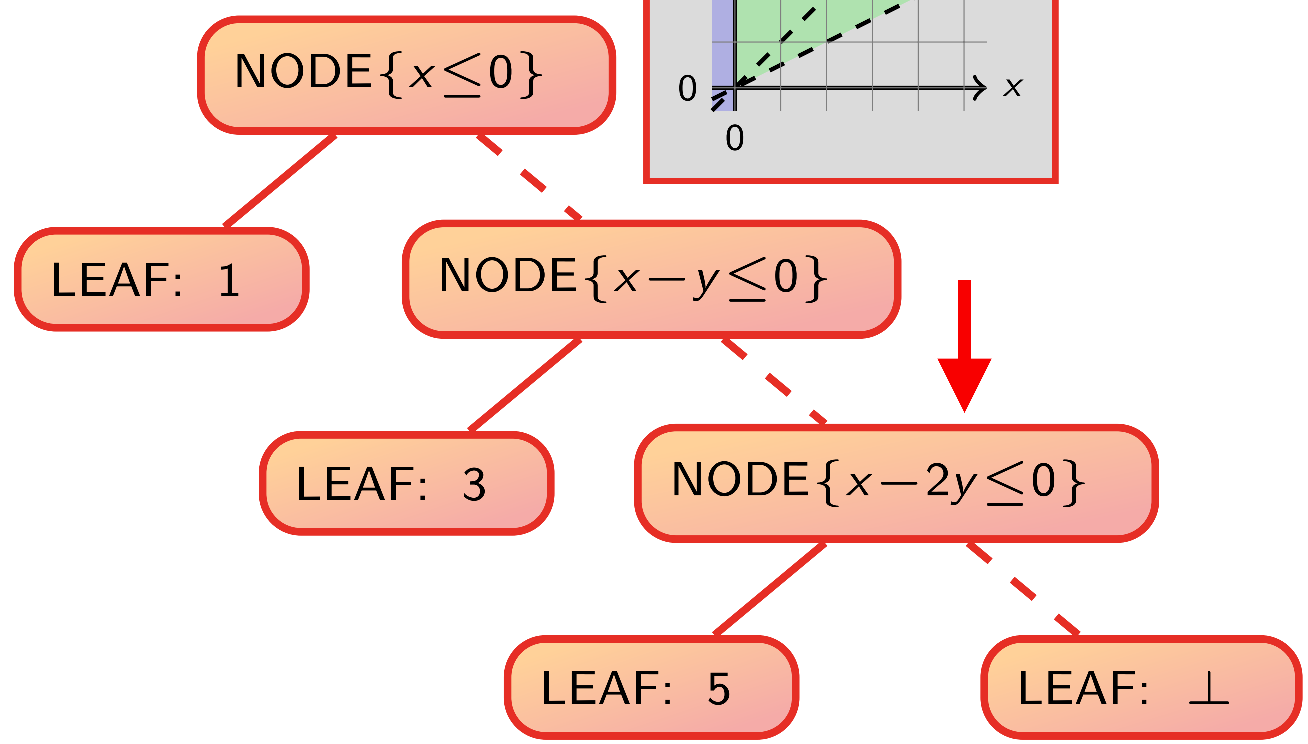
Widening: Left Unification

Algorithm 5 : Tree Left Unification

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



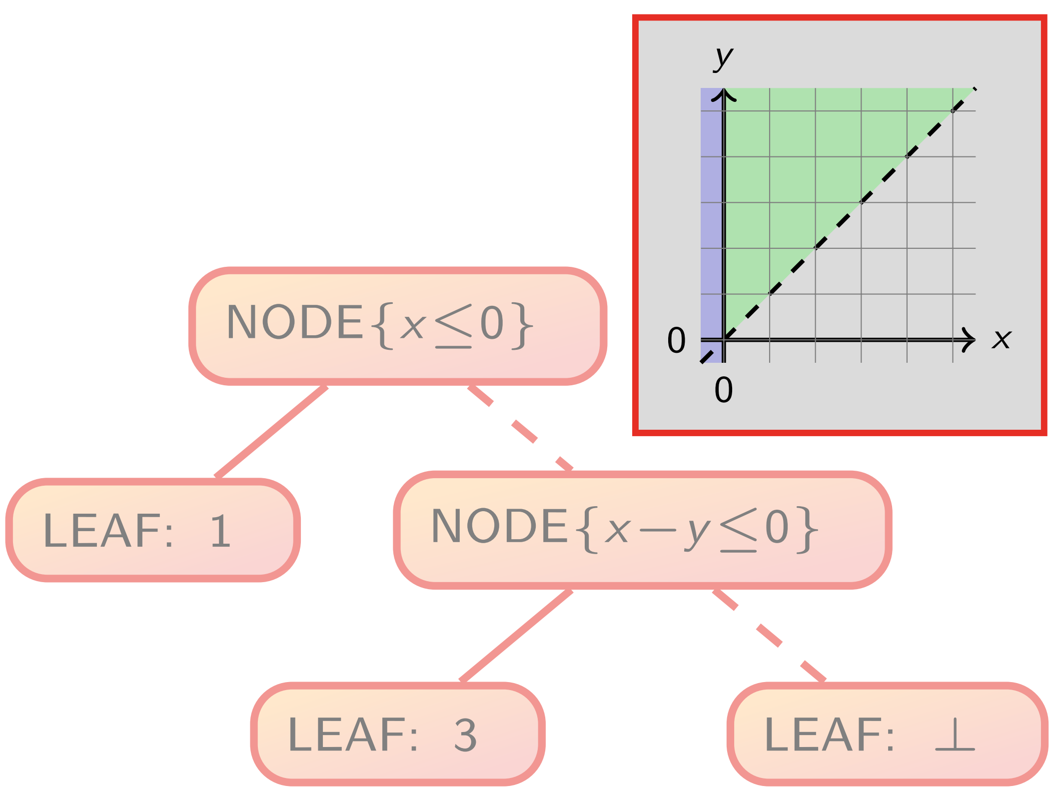


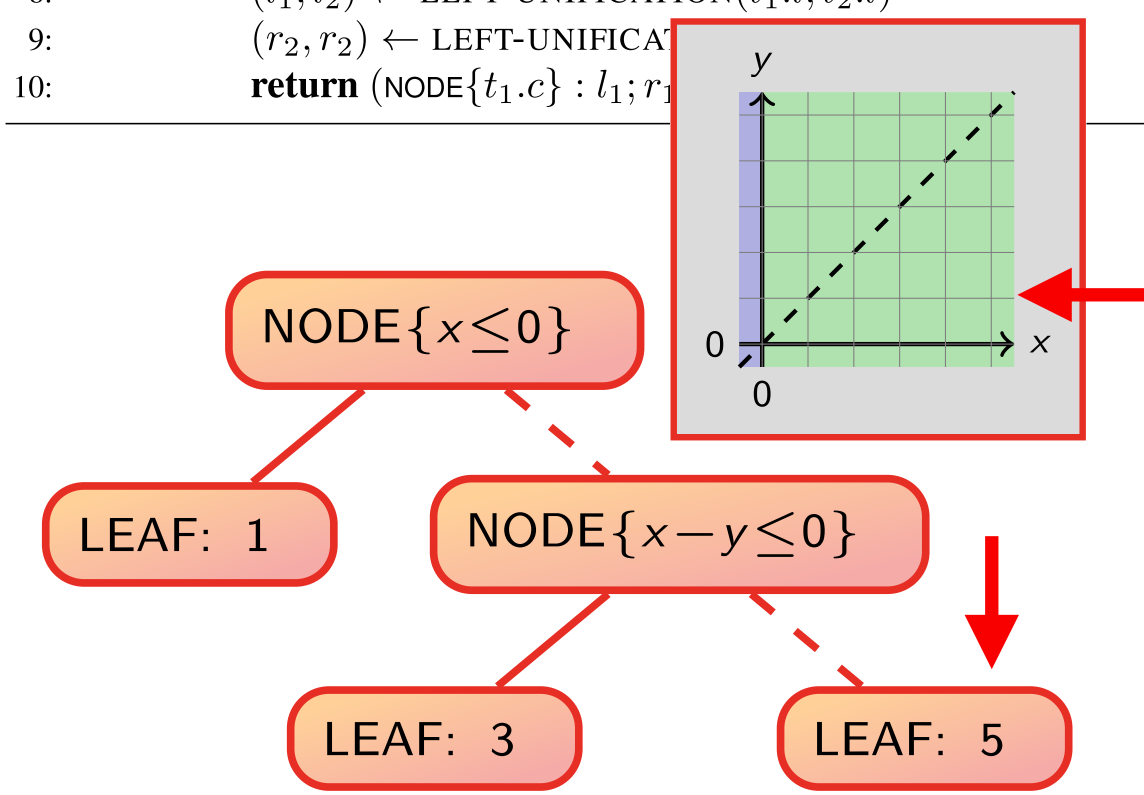


Widening: Left Unification

Algorithm 5 : Tree Left Unification

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





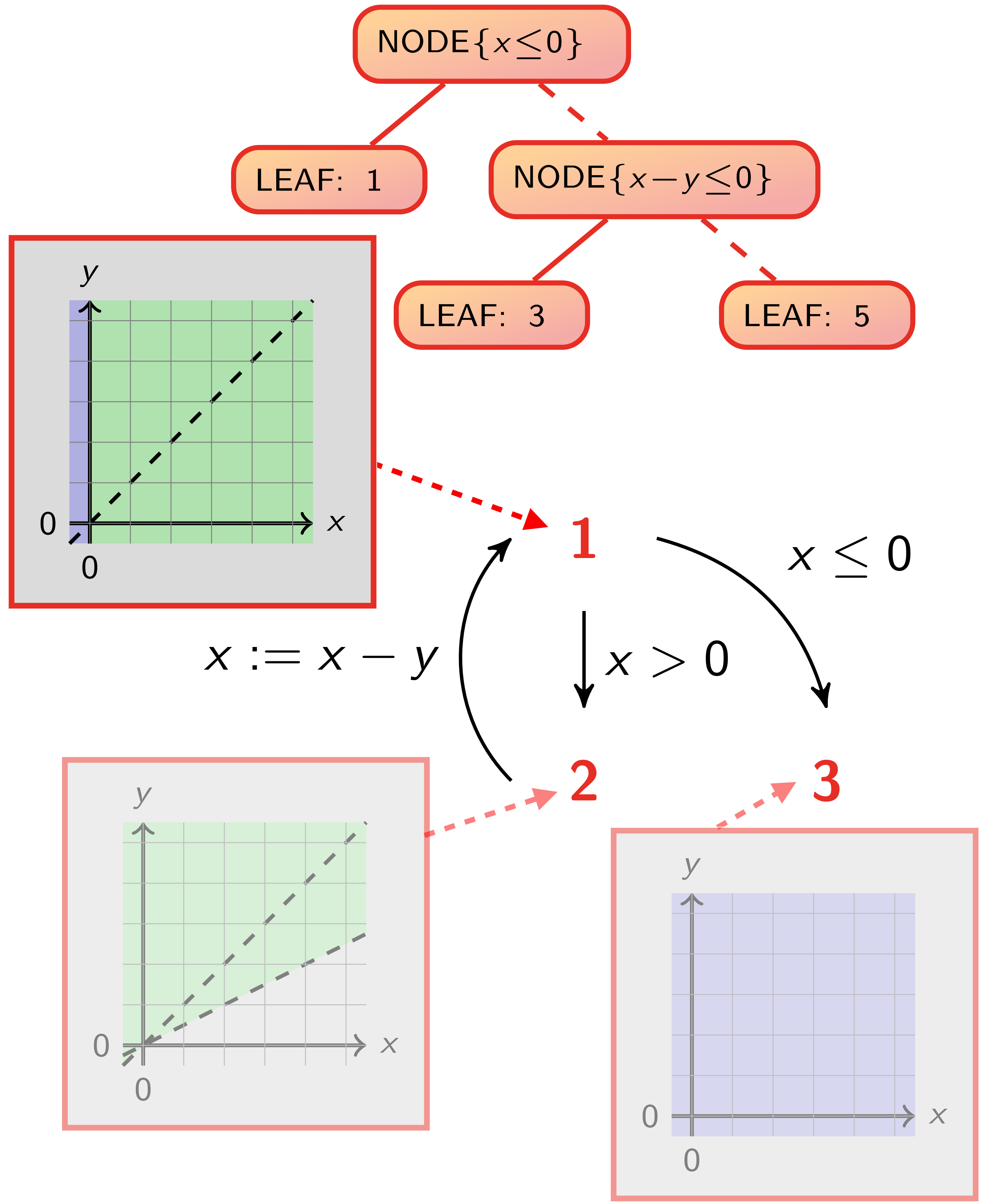
Example

$\mathrm{int} : x,\ y$

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

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

od^{3}



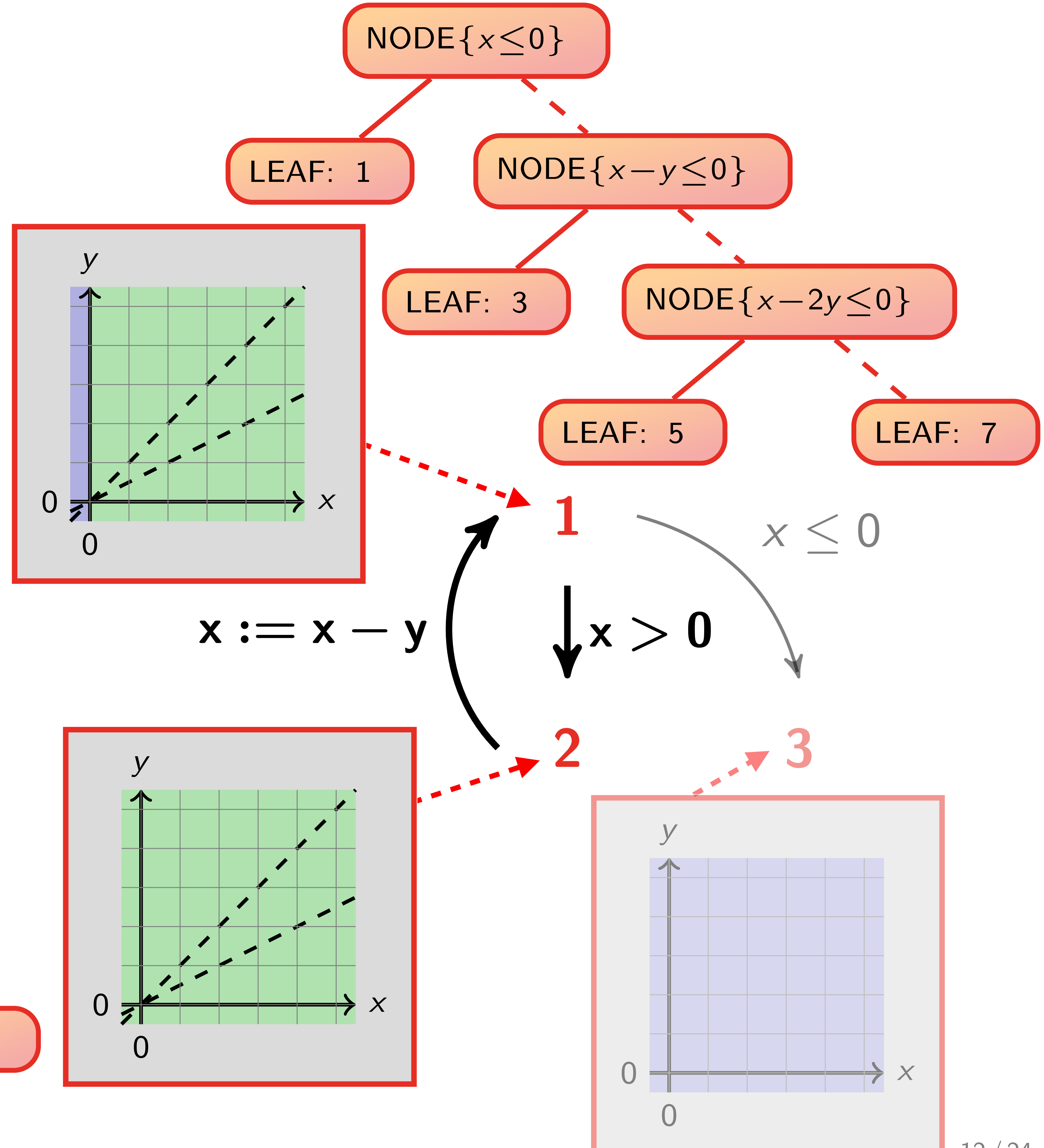
Example

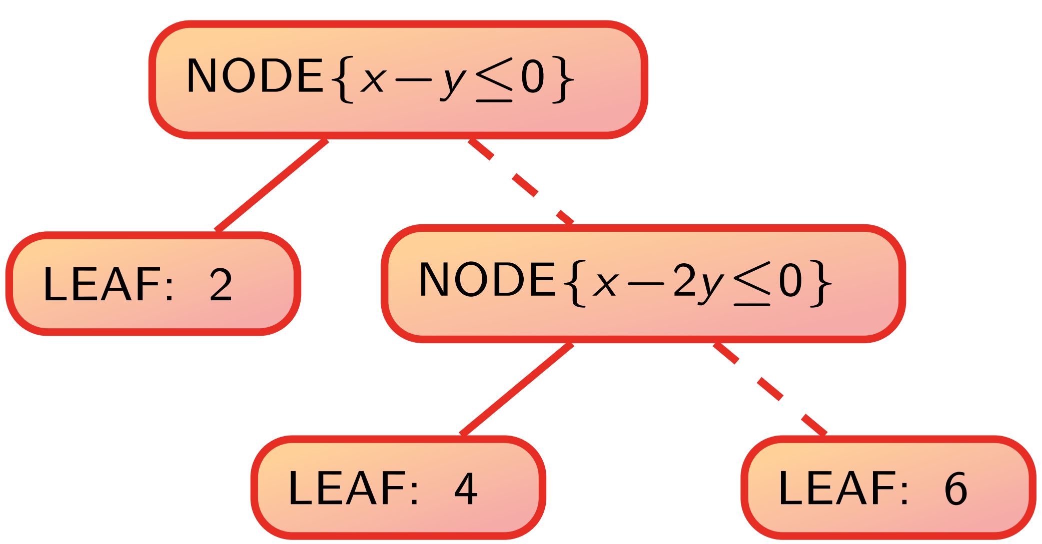
int : x, y

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

[2]$x := x - y$

od[3]

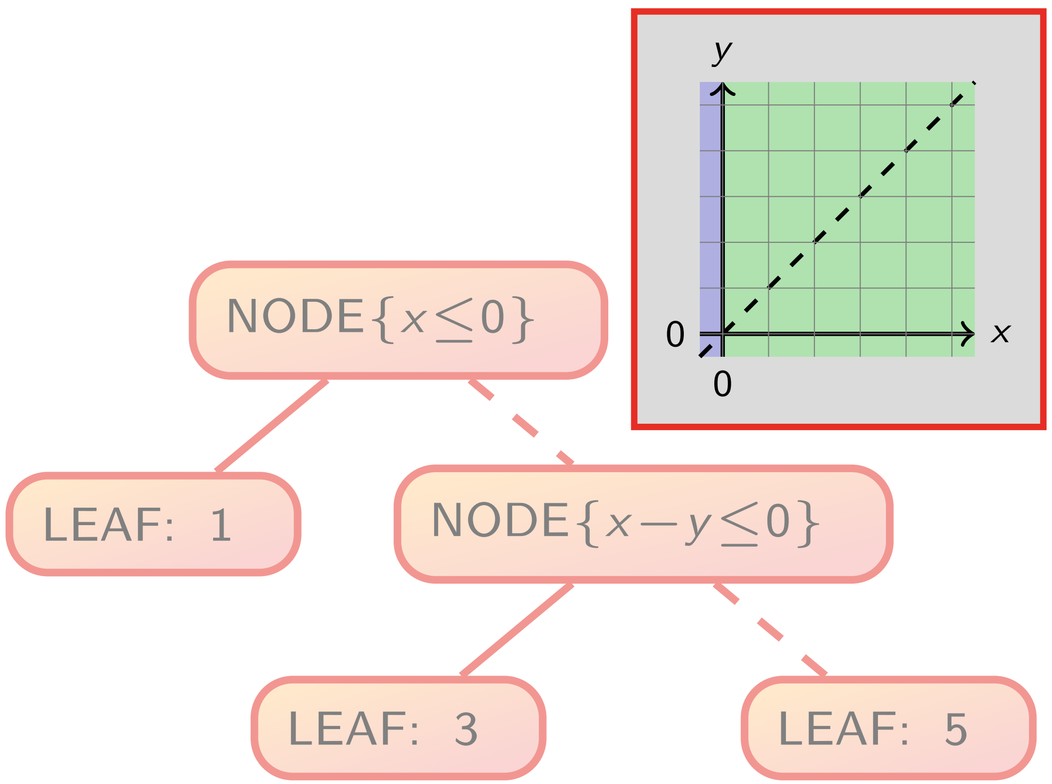


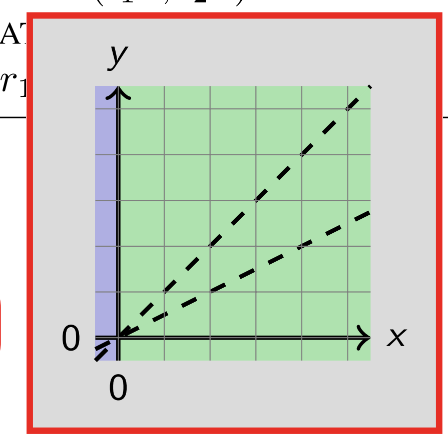


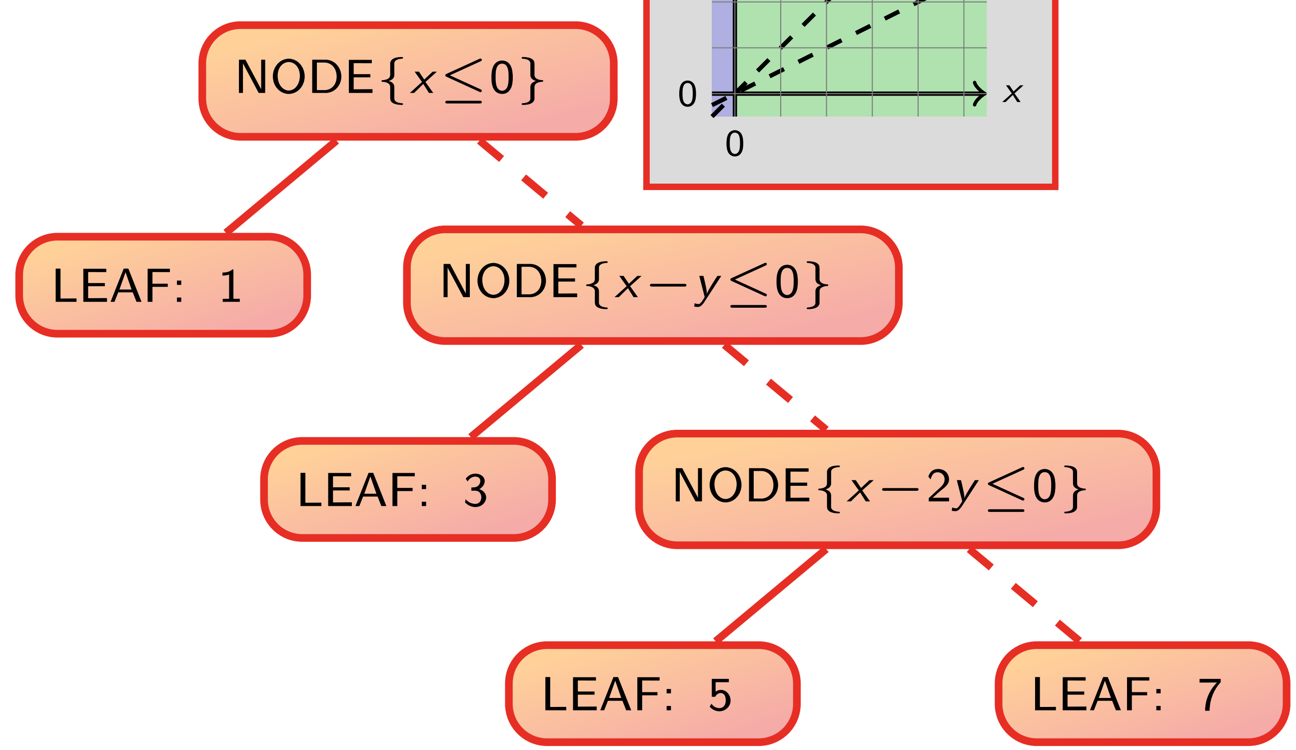
Widening: Domain Over-Approximation

Algorithm 5 : Tree Left Unification

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





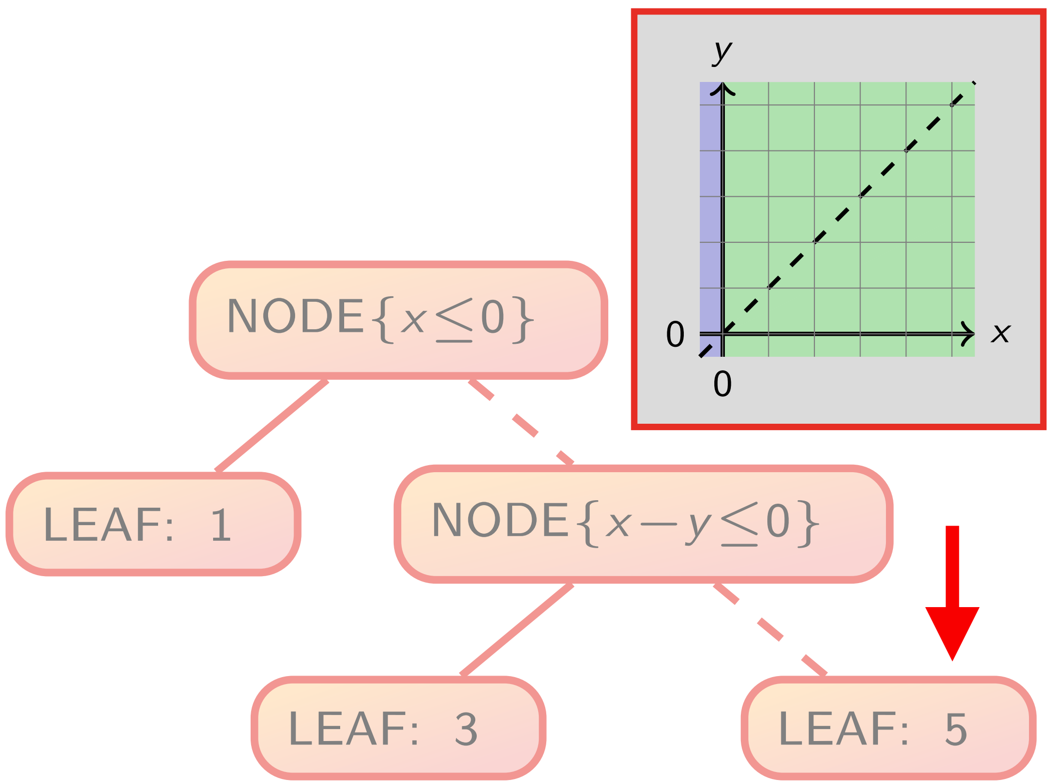


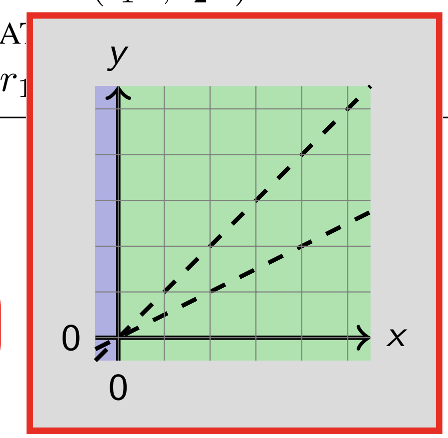
Widening: Domain Over-Approximation

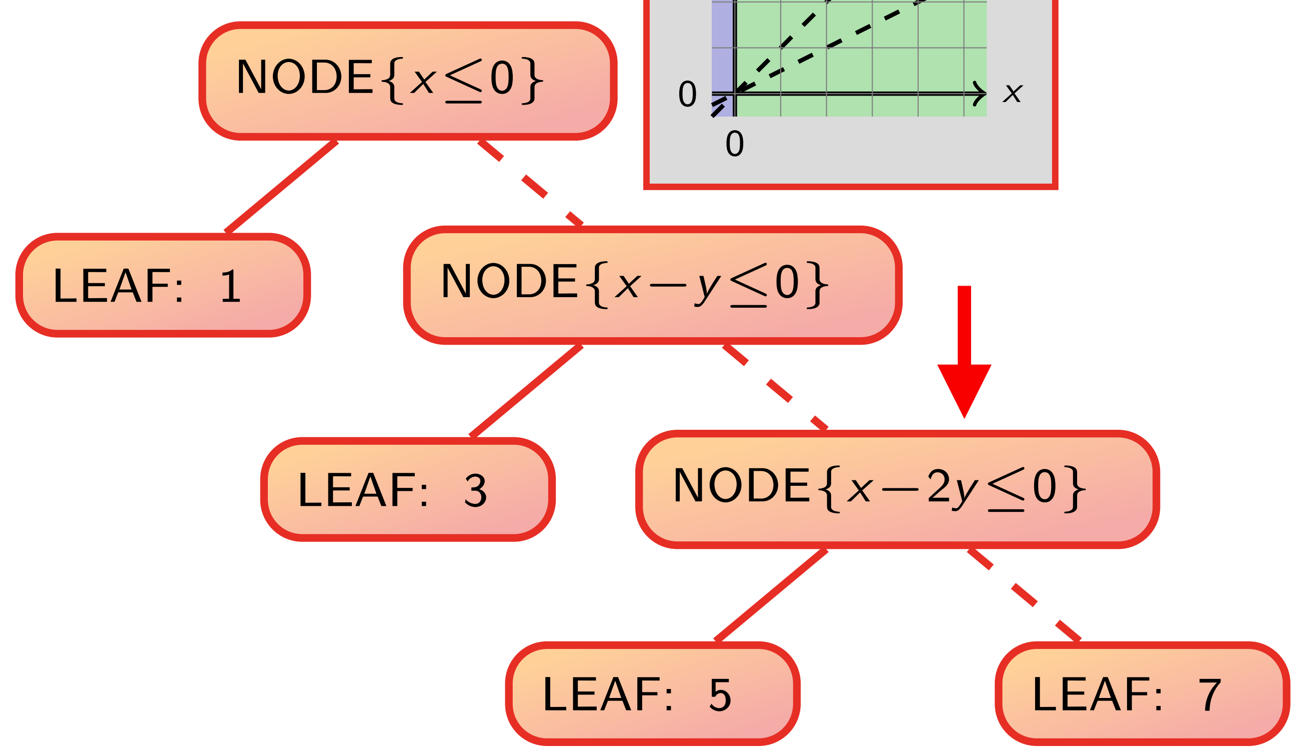
Algorithm 5 : Tree Left Unification

---

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



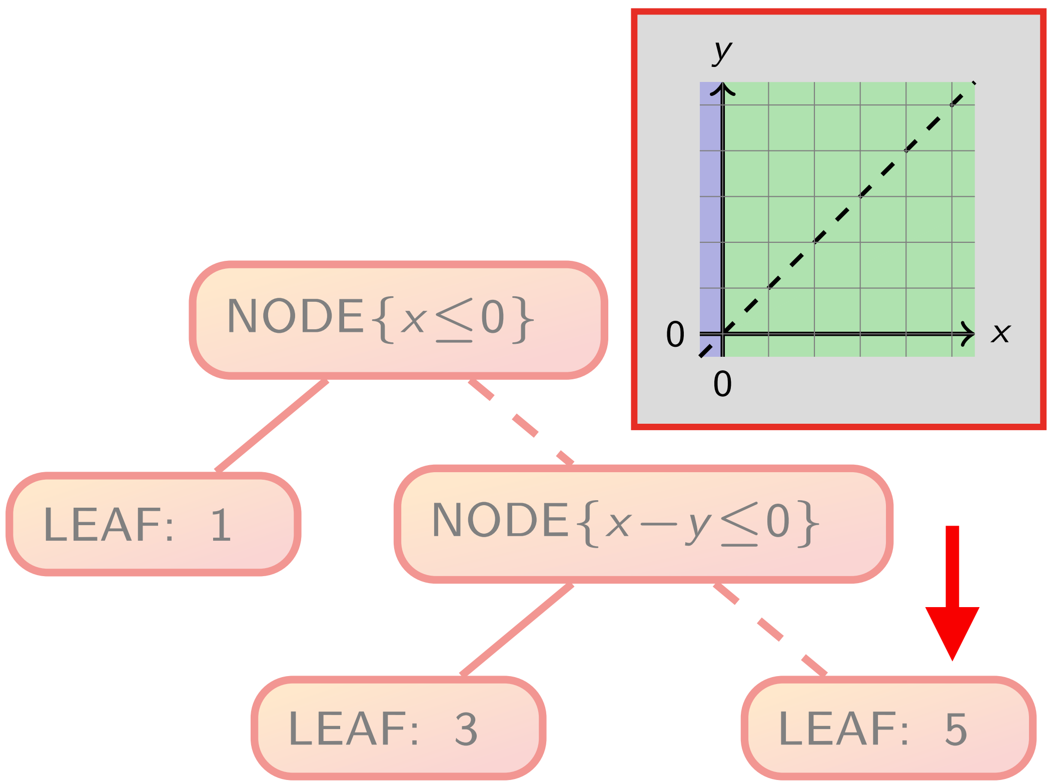


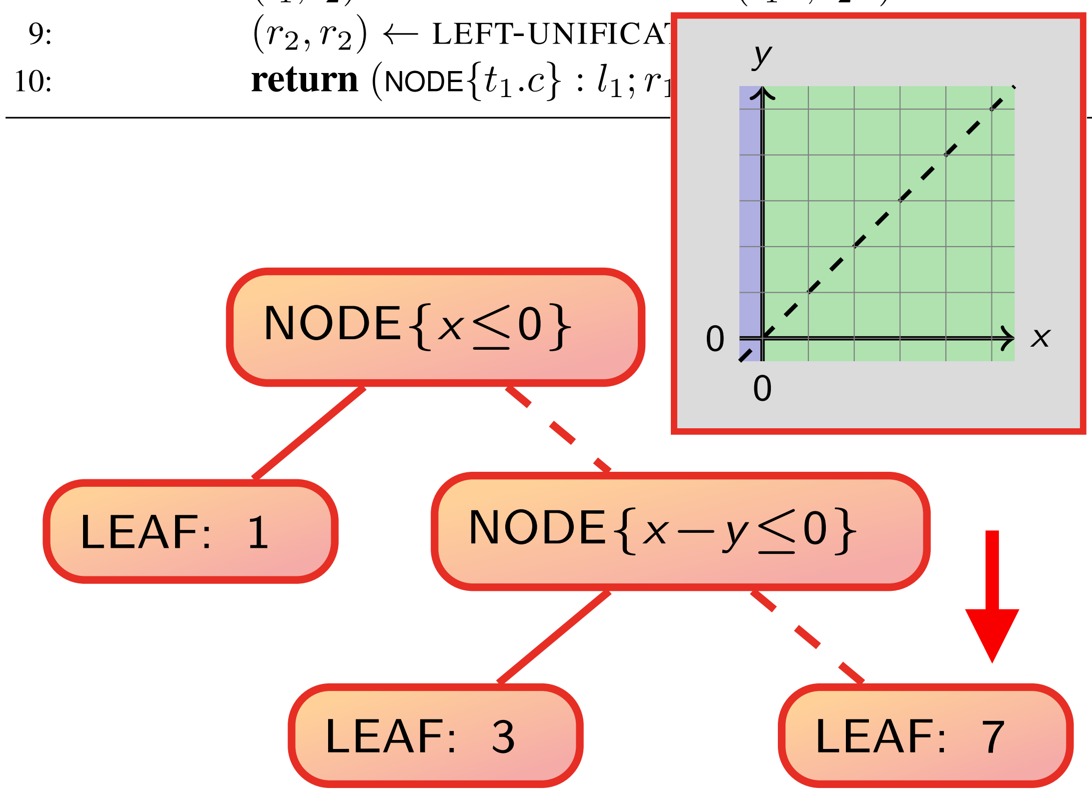


Widening: Domain Over-Approximation

Algorithm 5 : Tree Left Unification

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

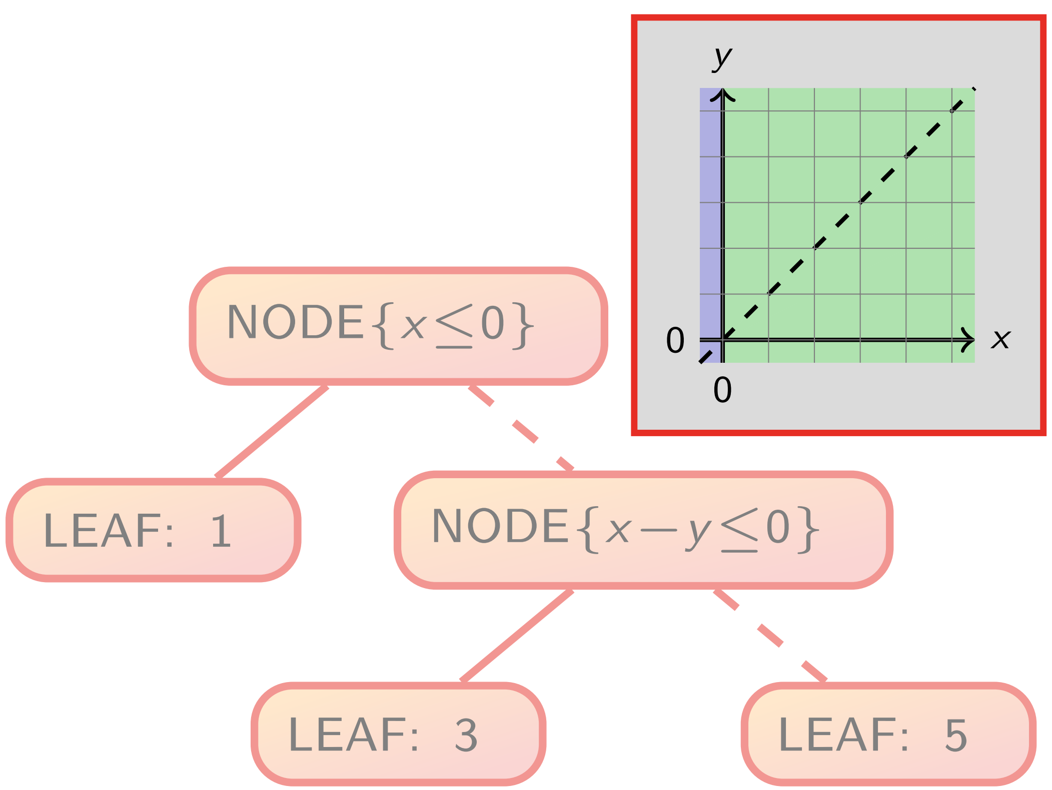


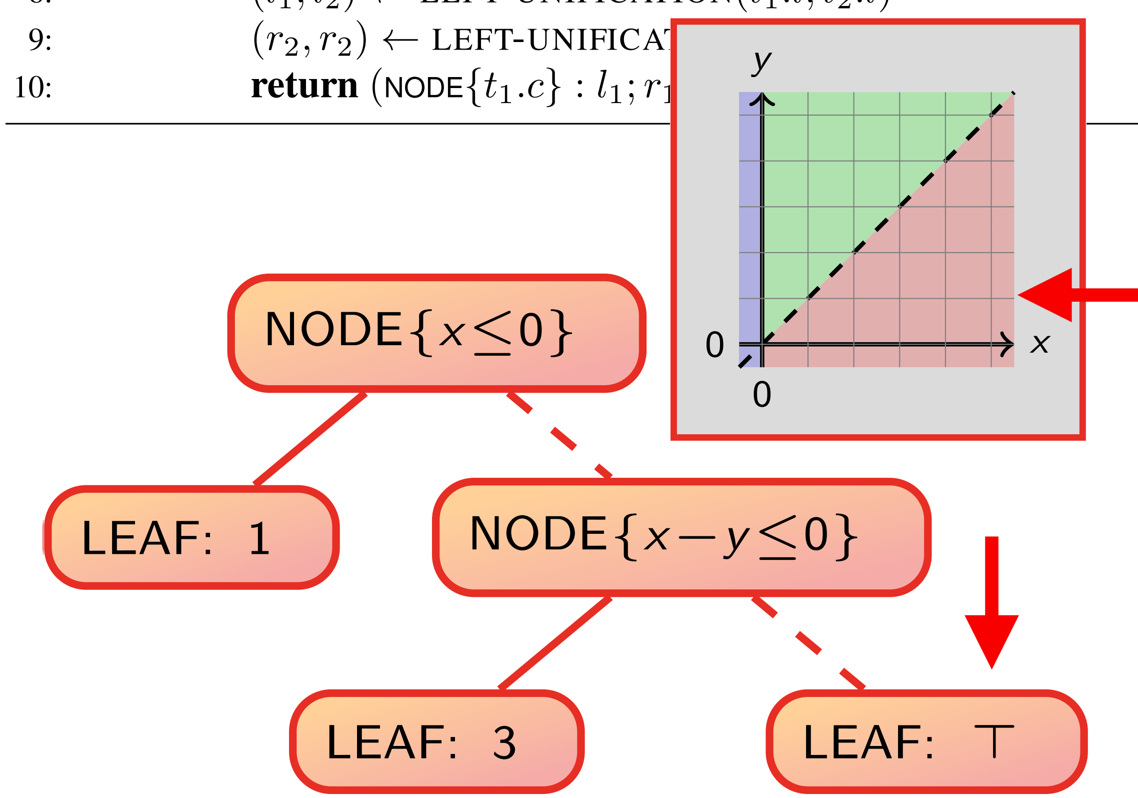


Widening: Domain Over-Approximation

Algorithm 5 : Tree Left Unification

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





Example

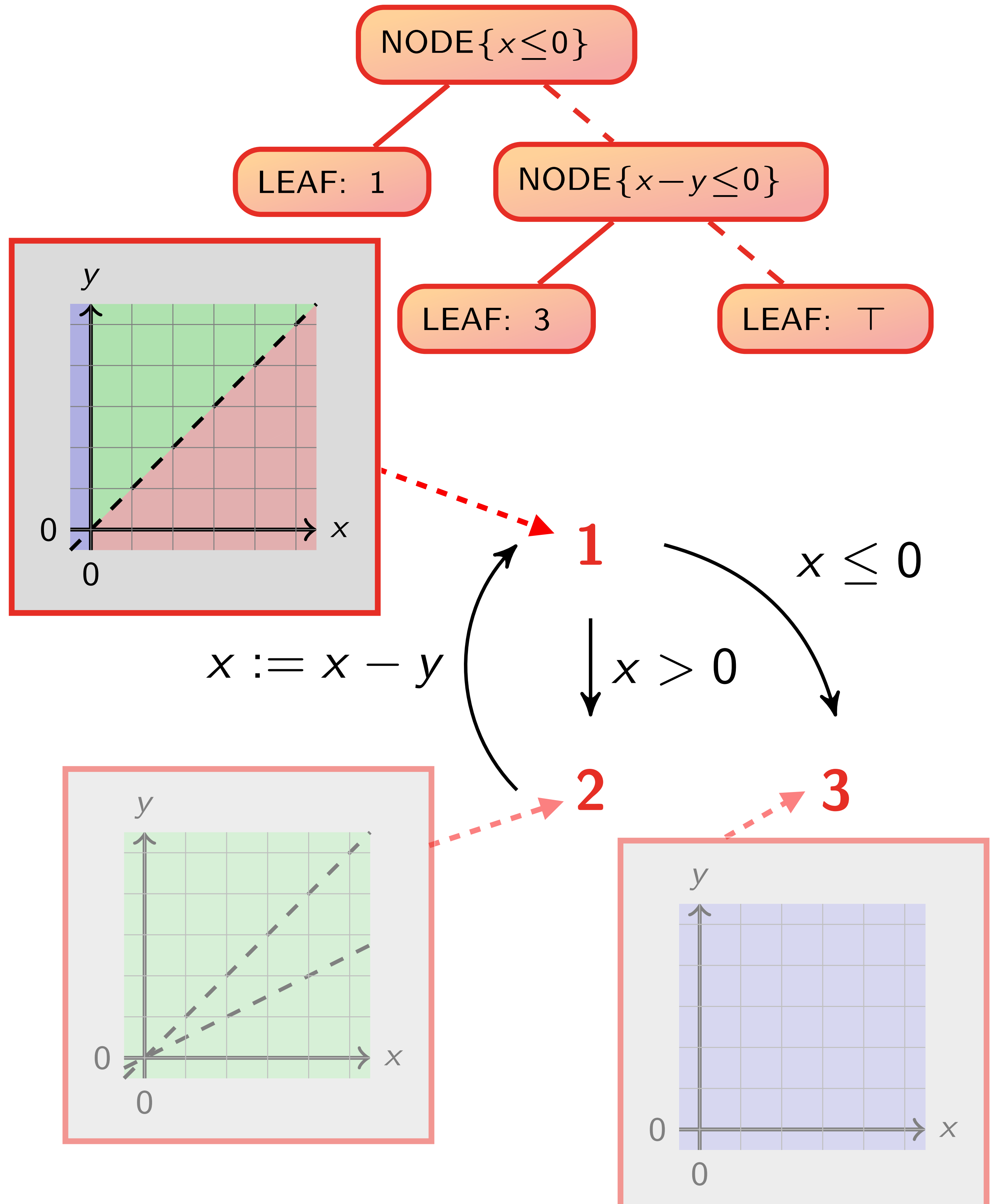
int : x, y

while $^1(x > 0)$ do

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

od^3

→ the widening is sound!



Example

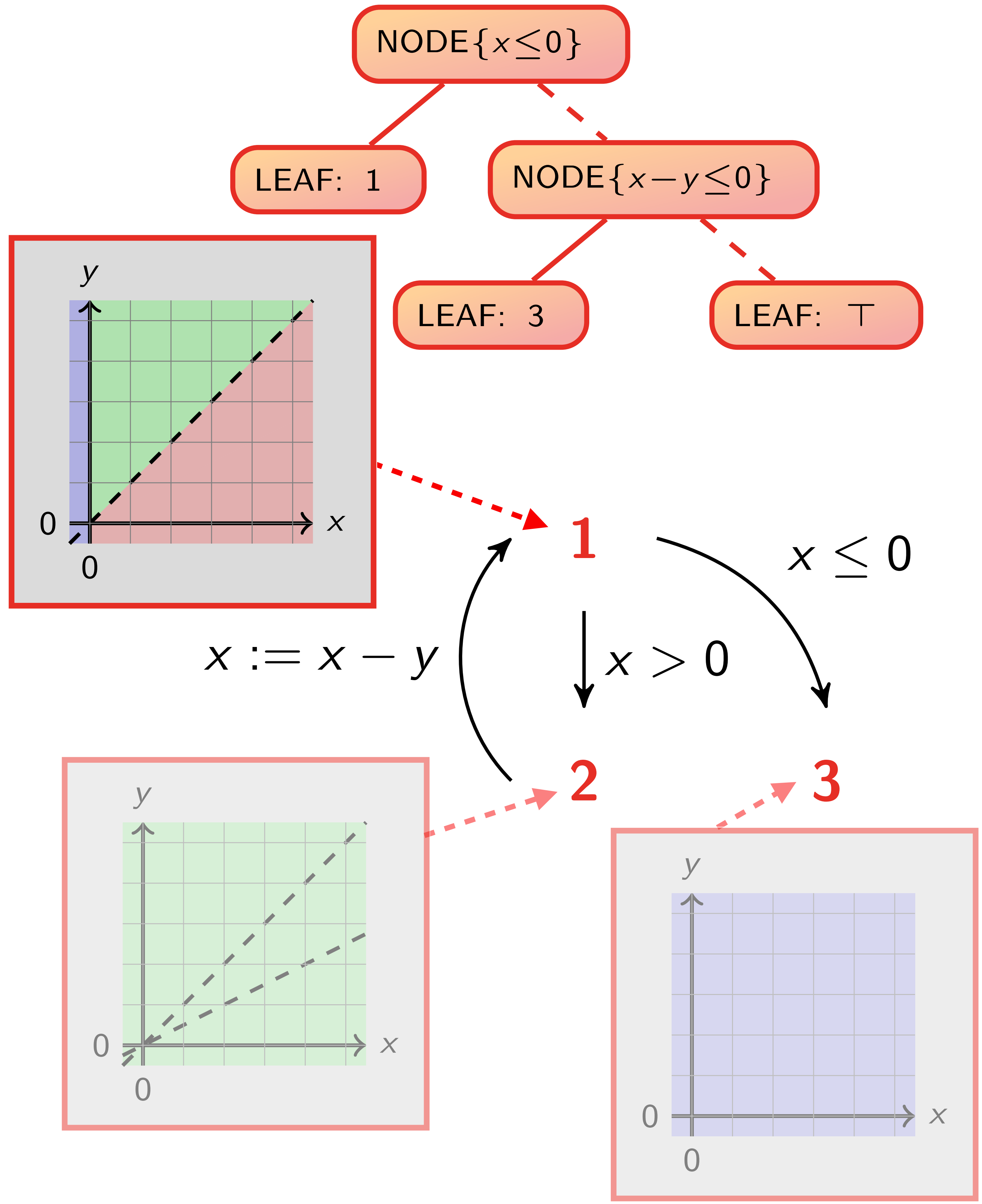
int : x, y

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

  [2]$x := x - y$

od[3]

the widening is **sound**!



Example

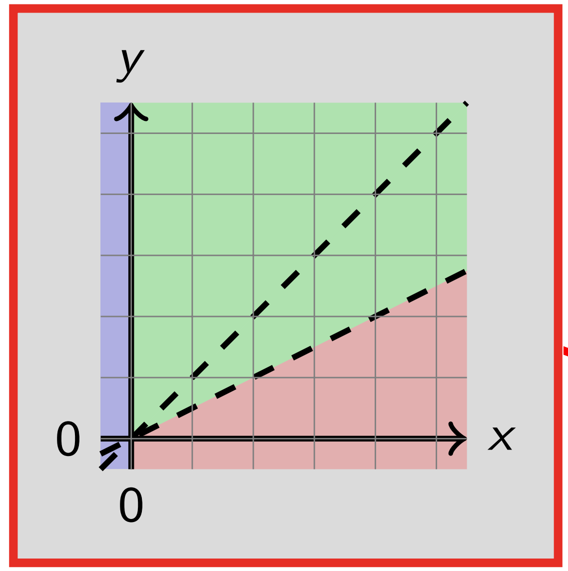
int : x, y

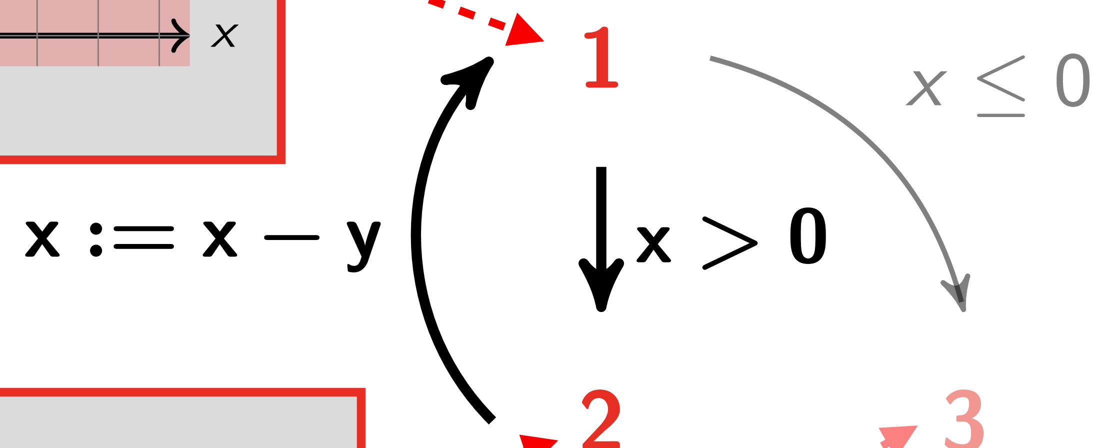
while [1] $(x > 0)$ do

[2] $x := x - y$

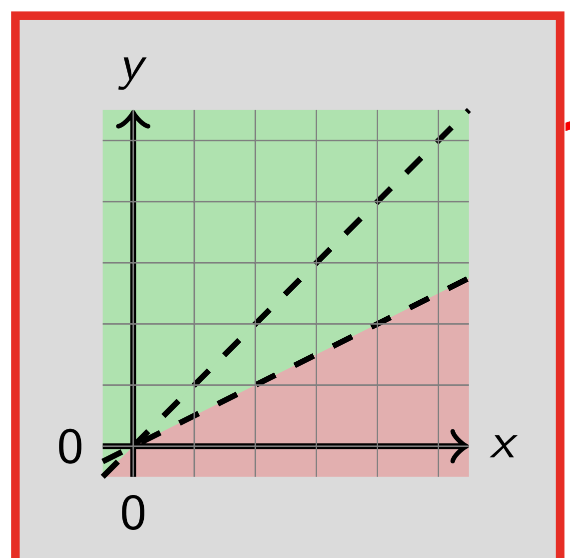
od [3]

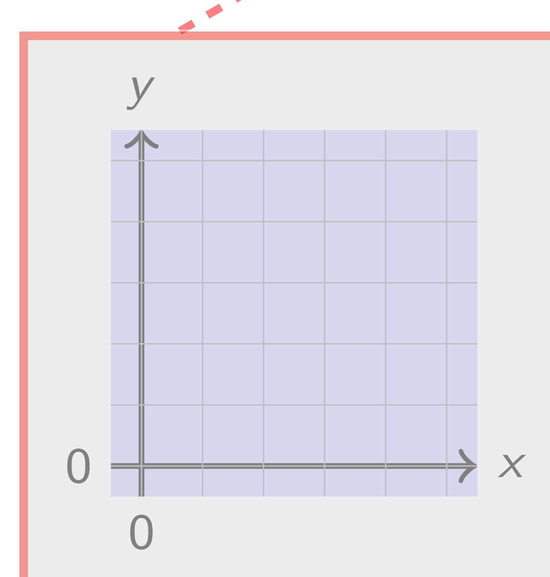
- NODE$\{x \leq 0\}$
  - LEAF: 1
  - NODE$\{x - y \leq 0\}$
    - LEAF: 3
    - NODE$\{x - 2y \leq 0\}$
      - LEAF: 5
      - LEAF: $\top$

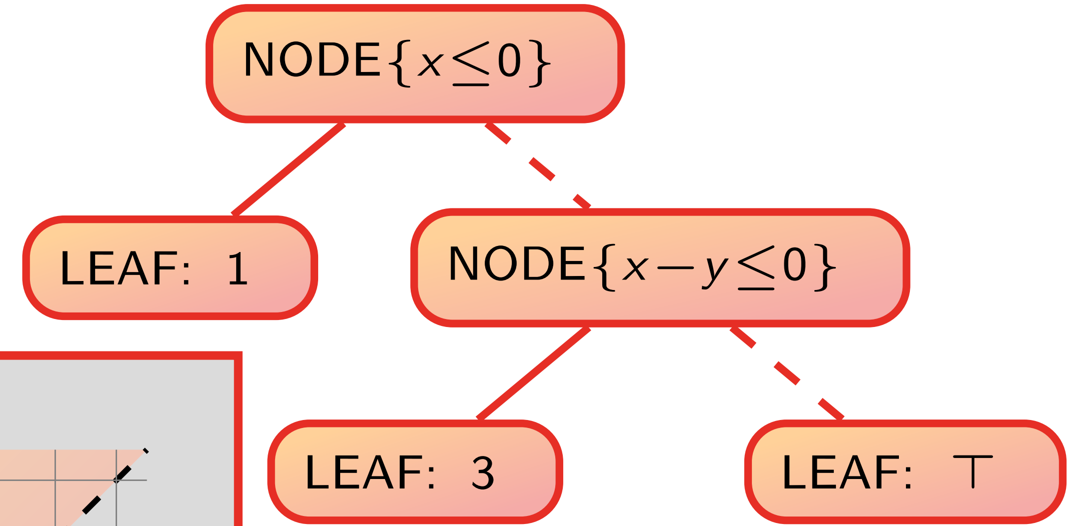




- NODE$\{x - y \leq 0\}$
  - LEAF: 2
  - NODE$\{x - 2y \leq 0\}$
    - LEAF: 4
    - LEAF: $\top$







Example

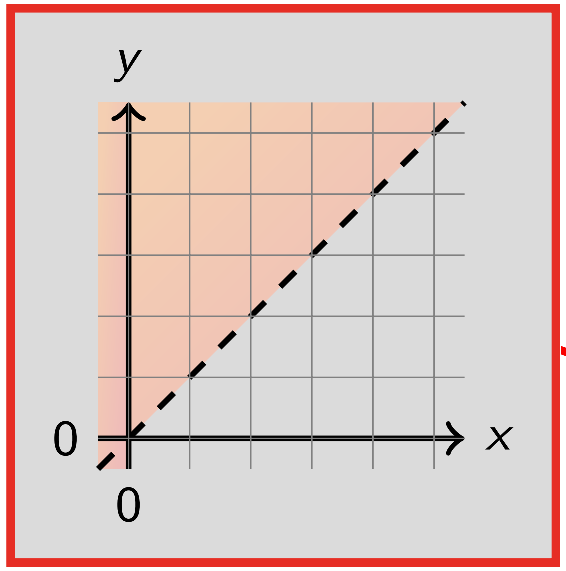
int : x, y

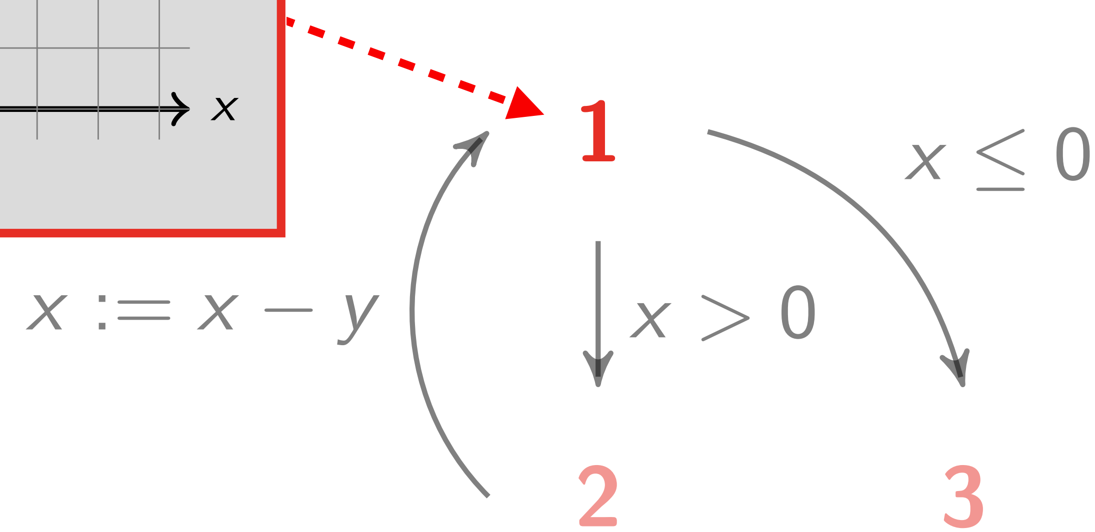
while [1]$(x > 0)$ do

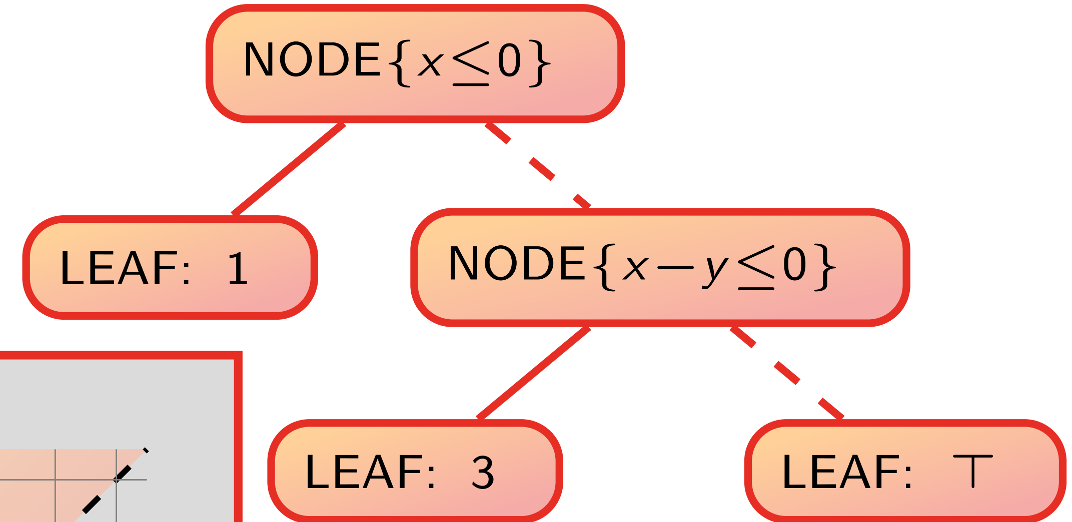
  [2]$x := x - y$

od[3]

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







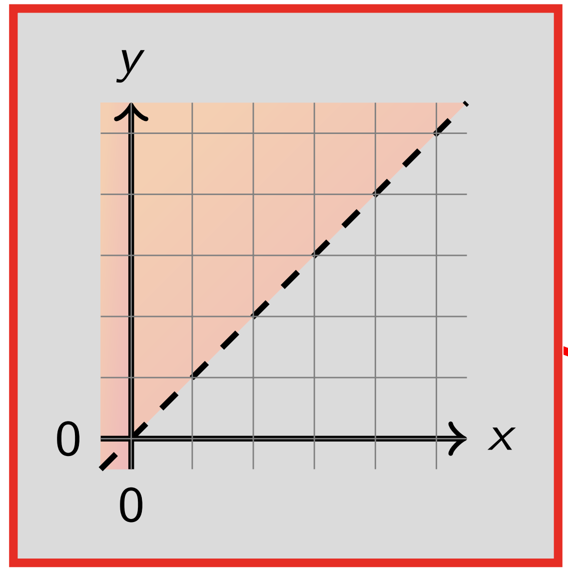
Example

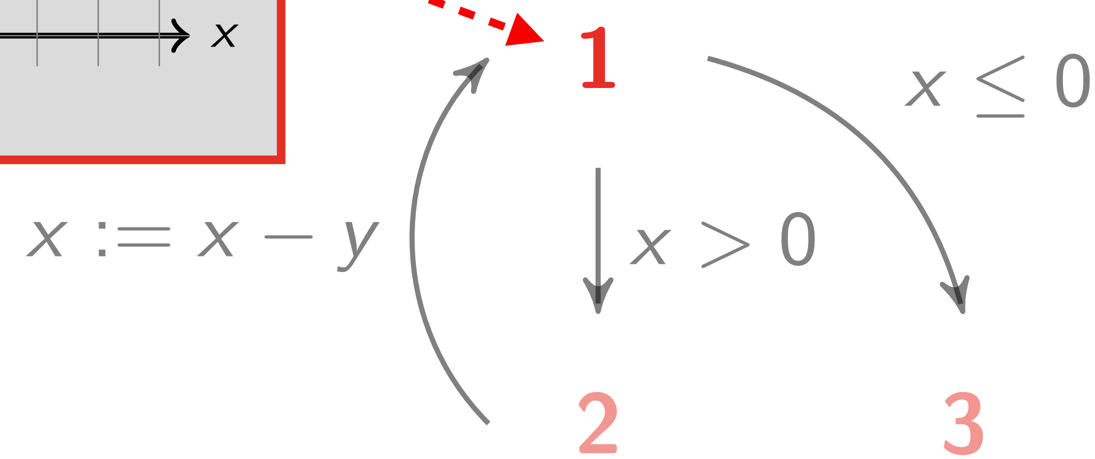
int : x, y

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

  [2]$x := x - y$

od[3]

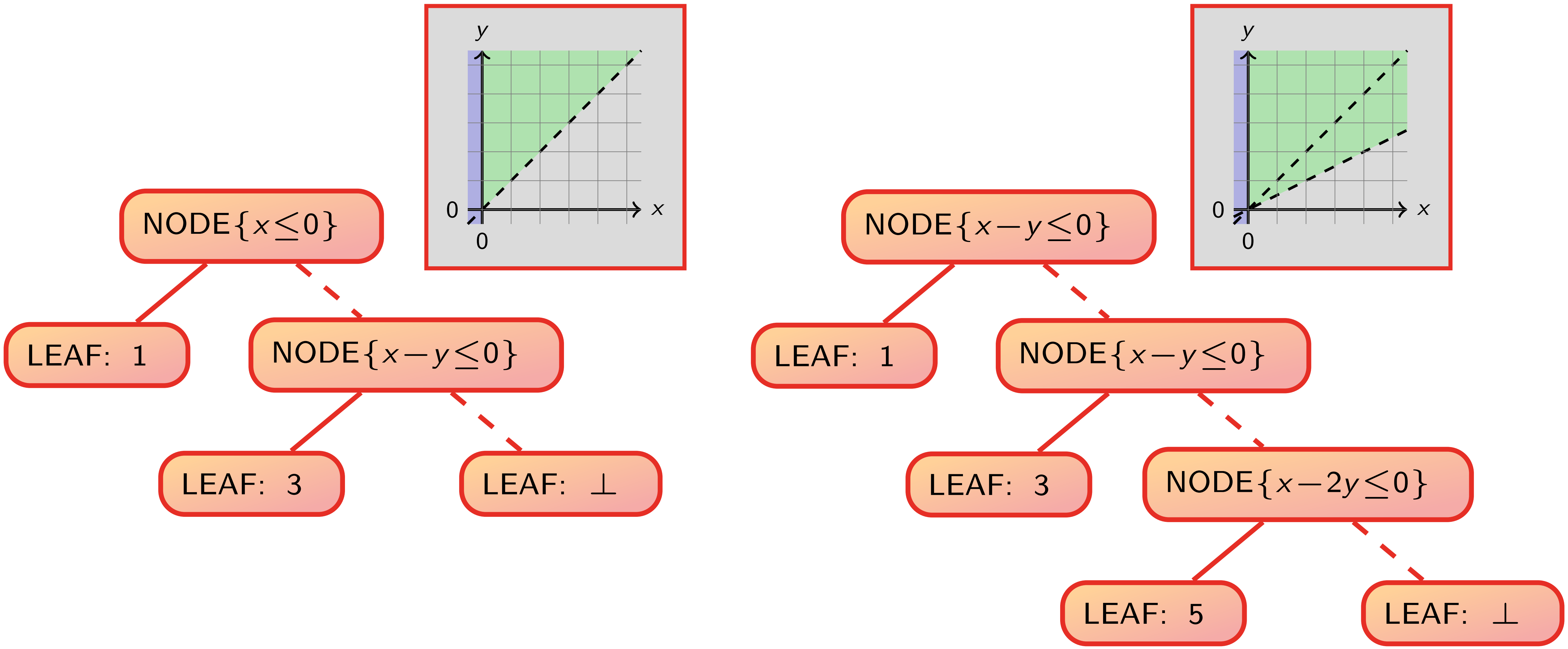




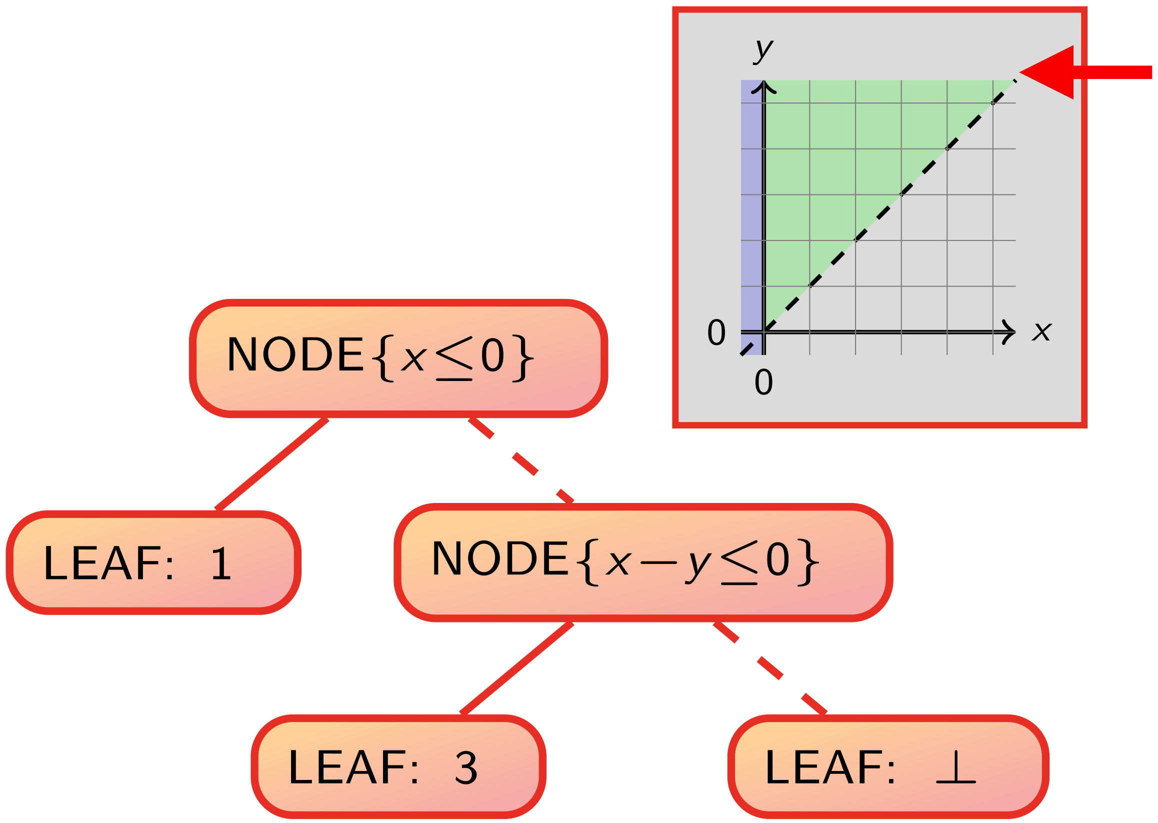
the analysis gives $x \leq 0 \vee x \leq y$ as **sufficient precondition**

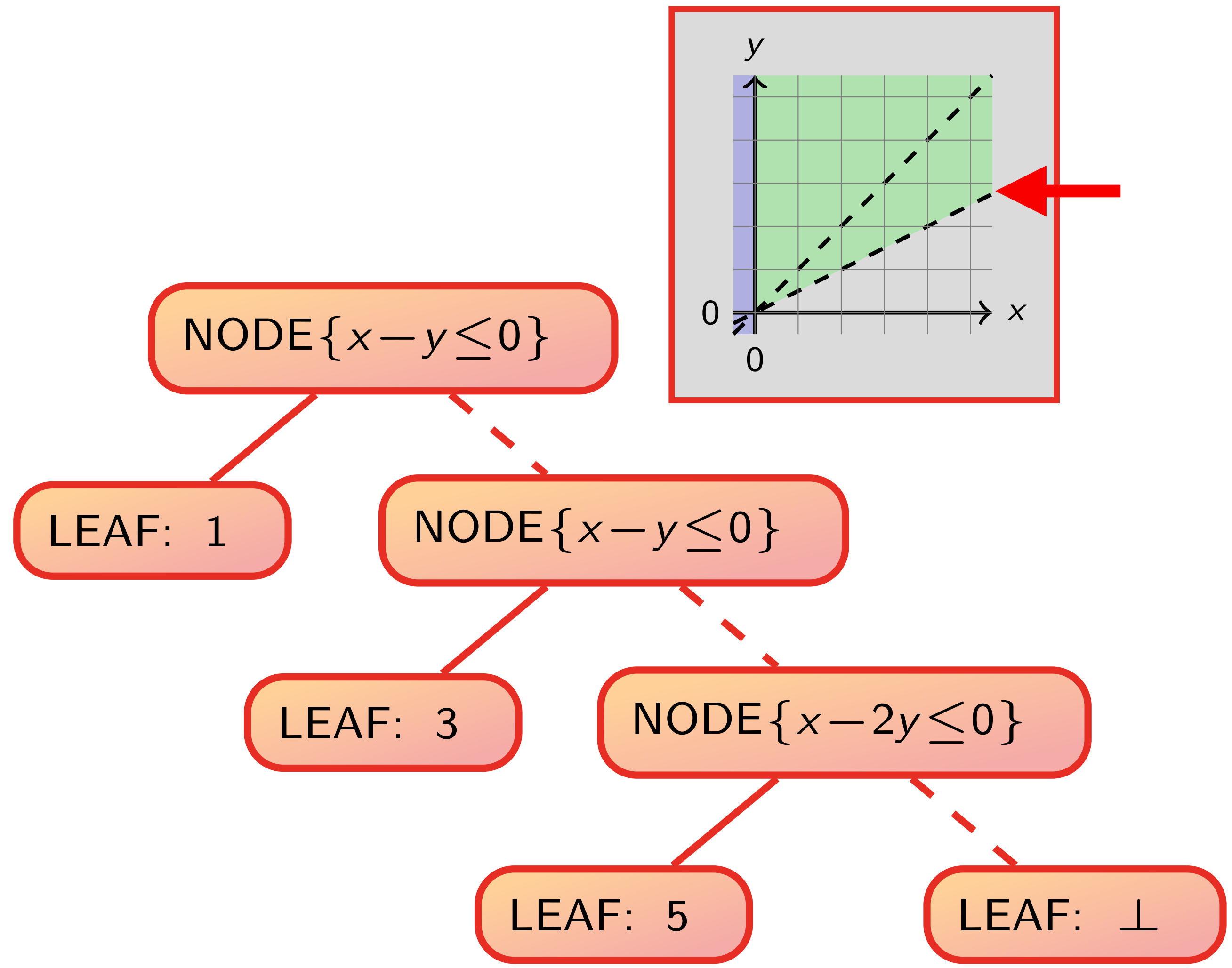
the **weakest precondition** is $\mathbf{x \leq 0 \vee y > 0}$

Widening

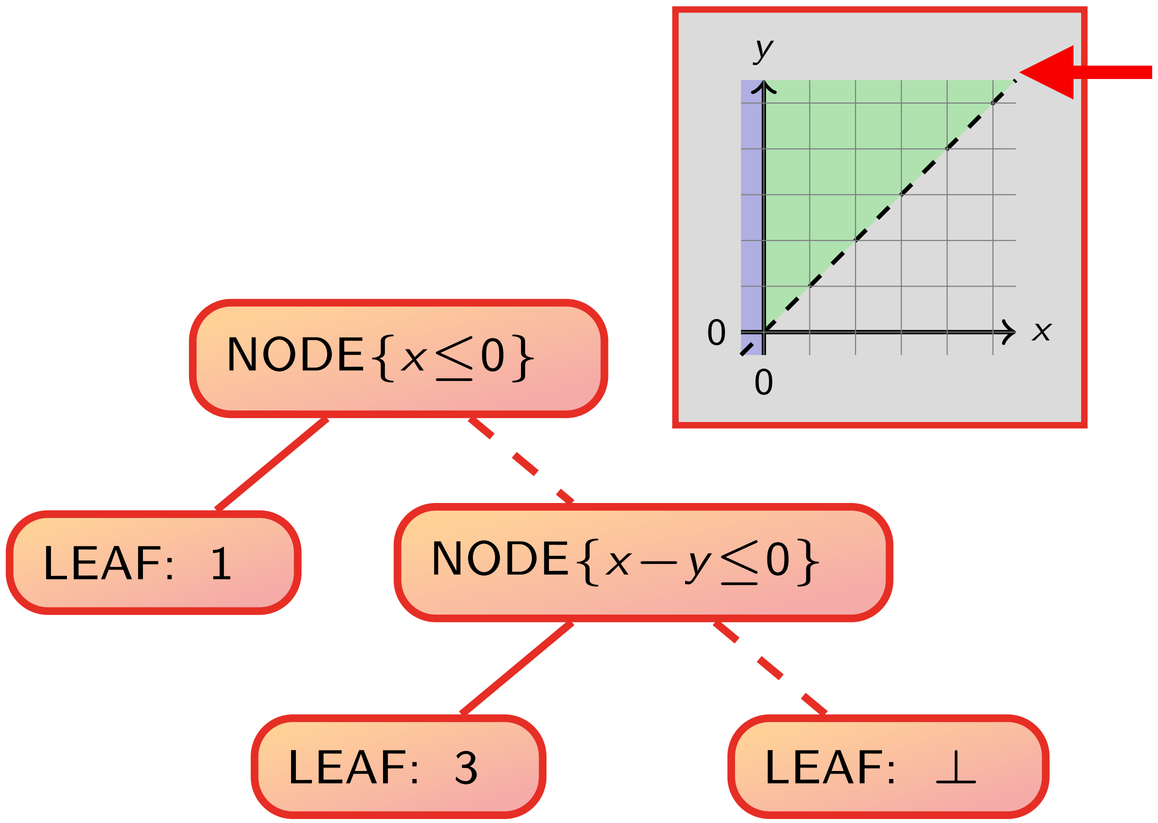

NODE{x≤0}
LEAF: 1
NODE{x−y≤0}
LEAF: 3
LEAF: ⊥
y
x
0
0
NODE{x−y≤0}
LEAF: 1
NODE{x−y≤0}
LEAF: 3
NODE{x−2y≤0}
LEAF: 5
LEAF: ⊥
y
x
0
0

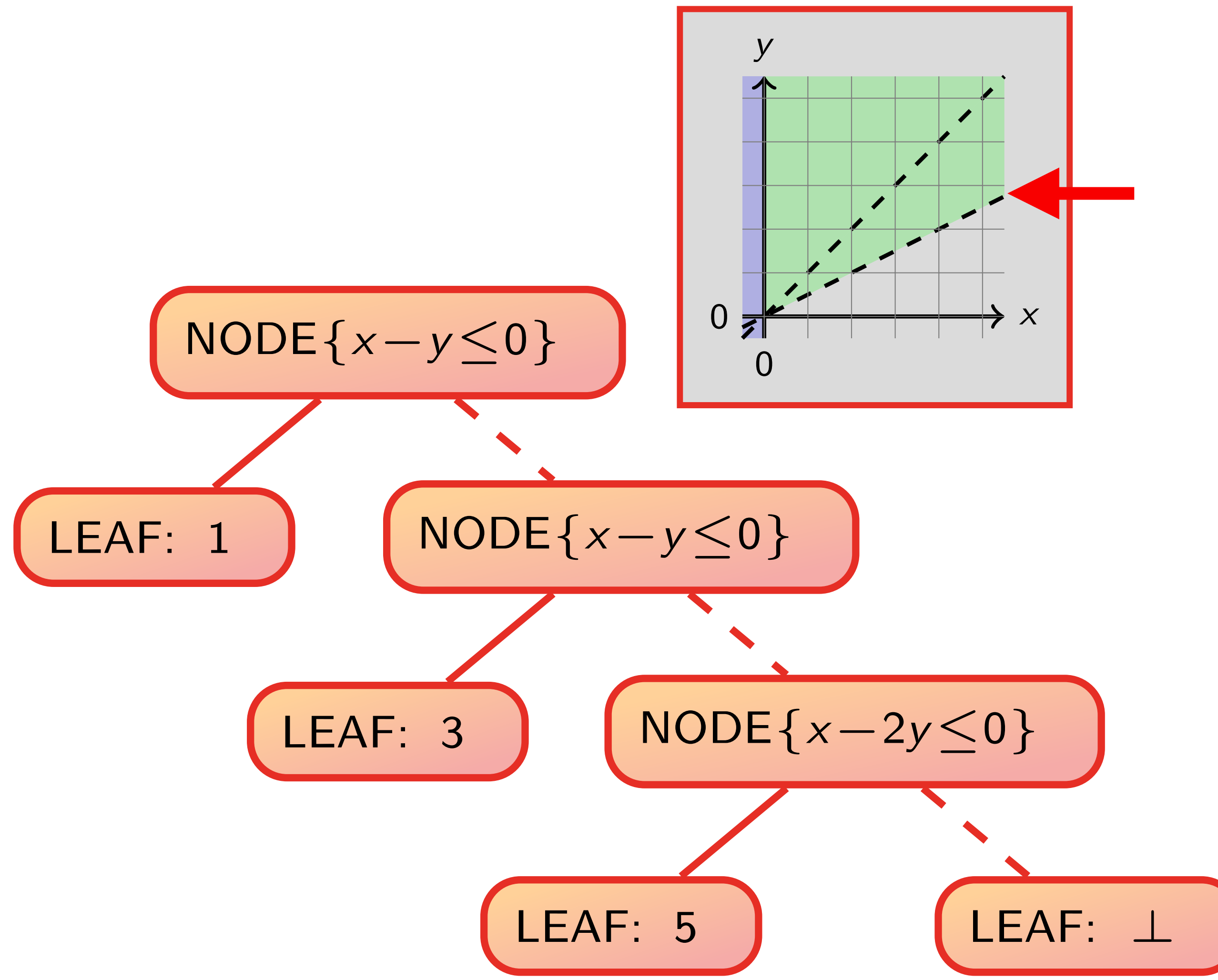
Widening

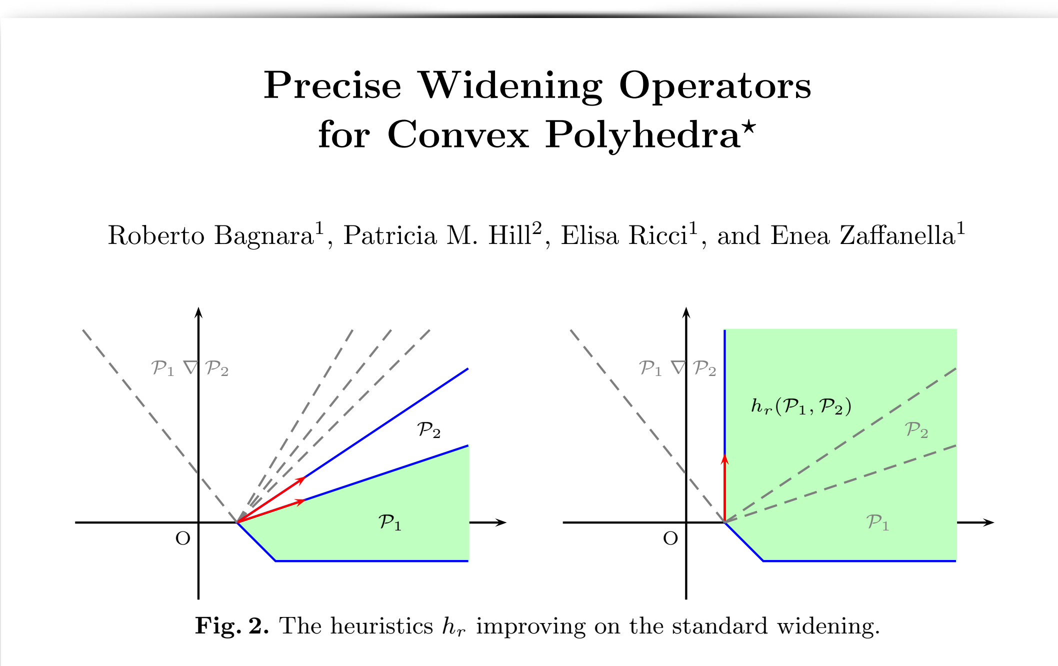

y
x
0
0
NODE{$x \leq 0$}
LEAF: 1
NODE{$x - y \leq 0$}
LEAF: 3
LEAF: $\bot$


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

Widening





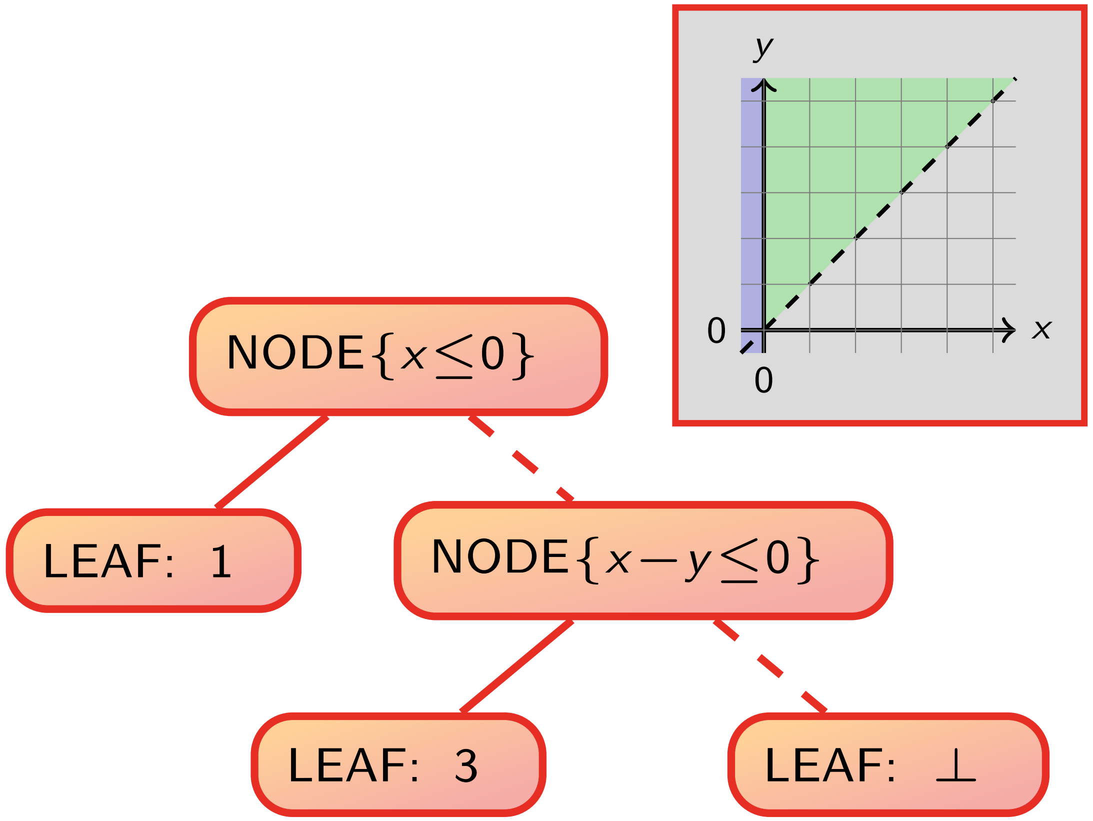


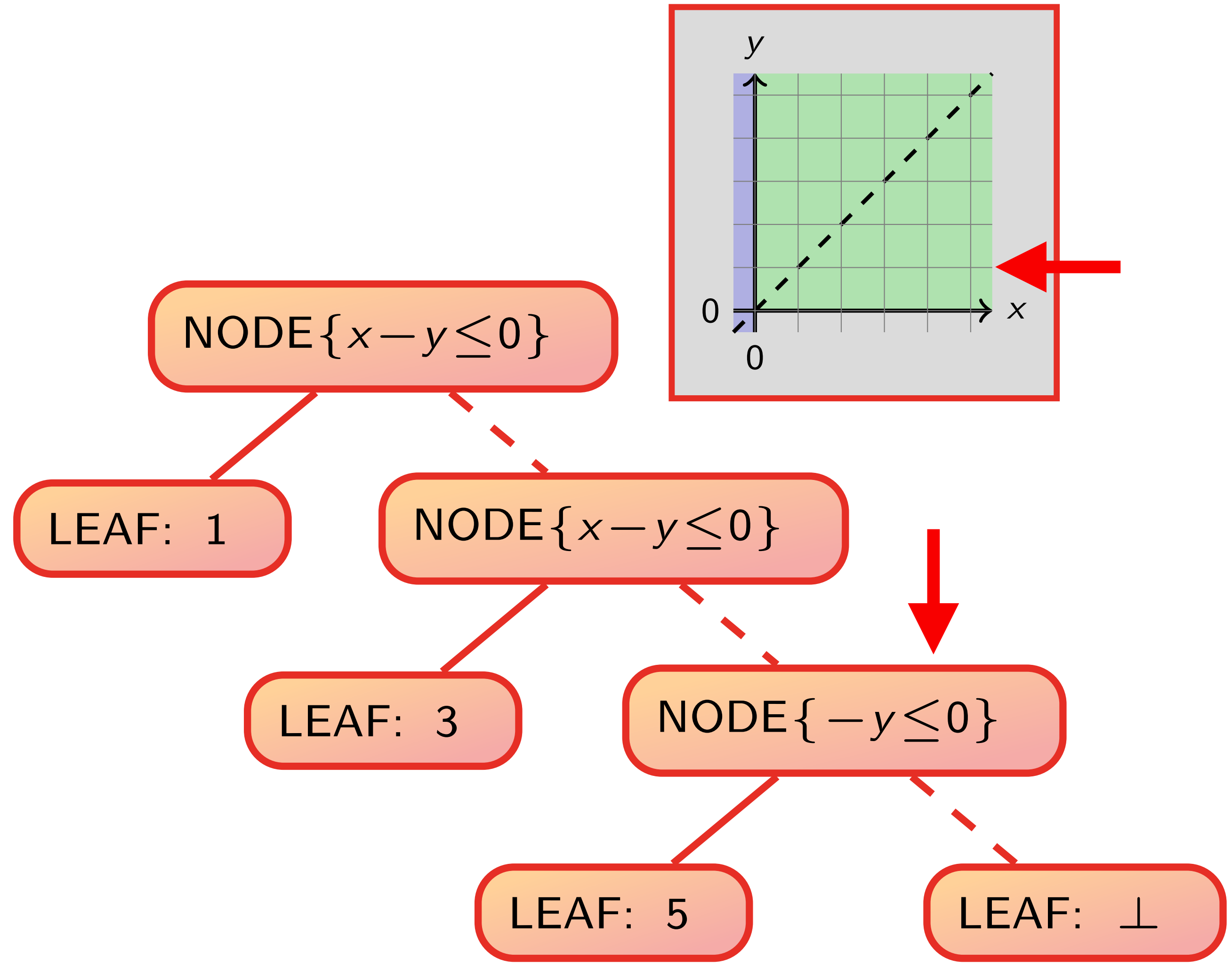
Precise Widening Operators for Convex Polyhedra⋆

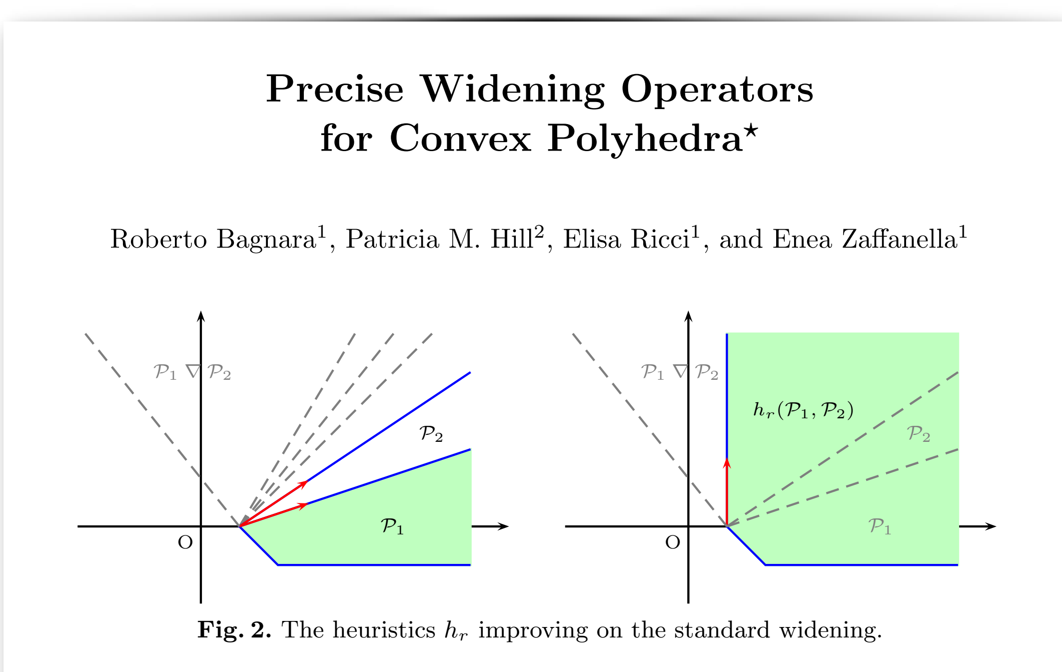
Roberto Bagnara[1], Patricia M. Hill[2], Elisa Ricci[1], and Enea Zaffanella[1]

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

Widening



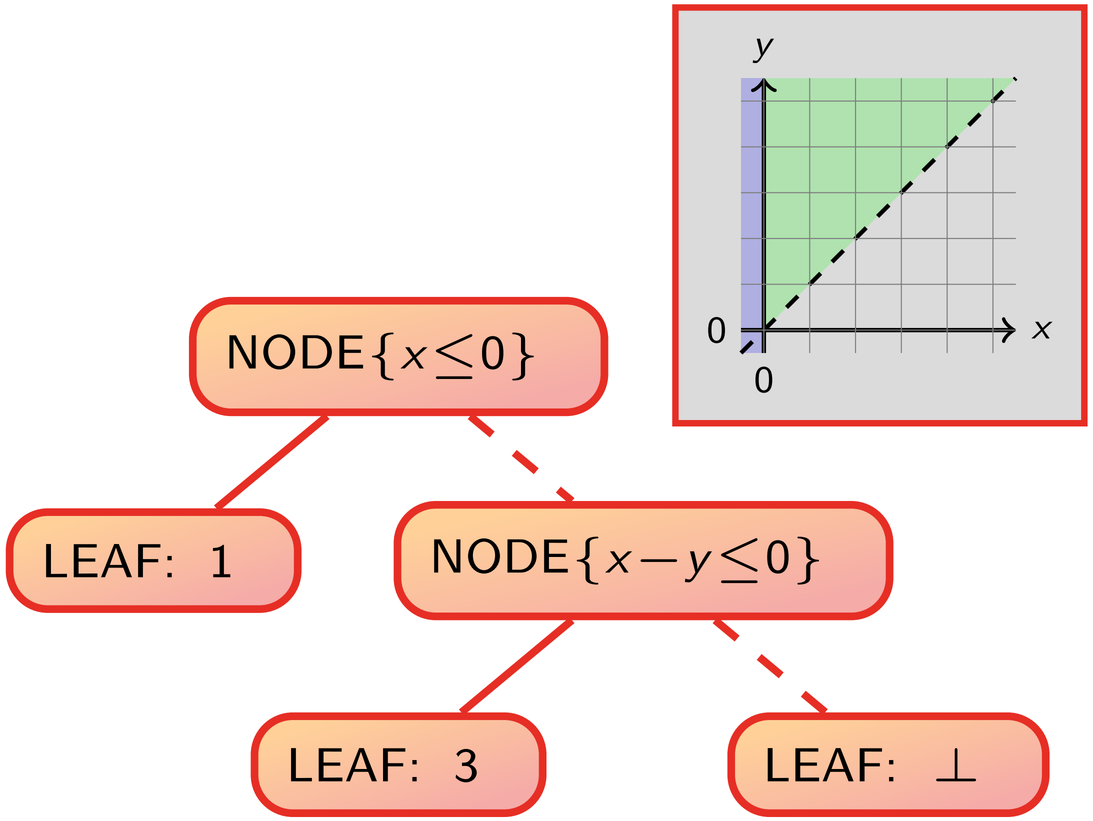


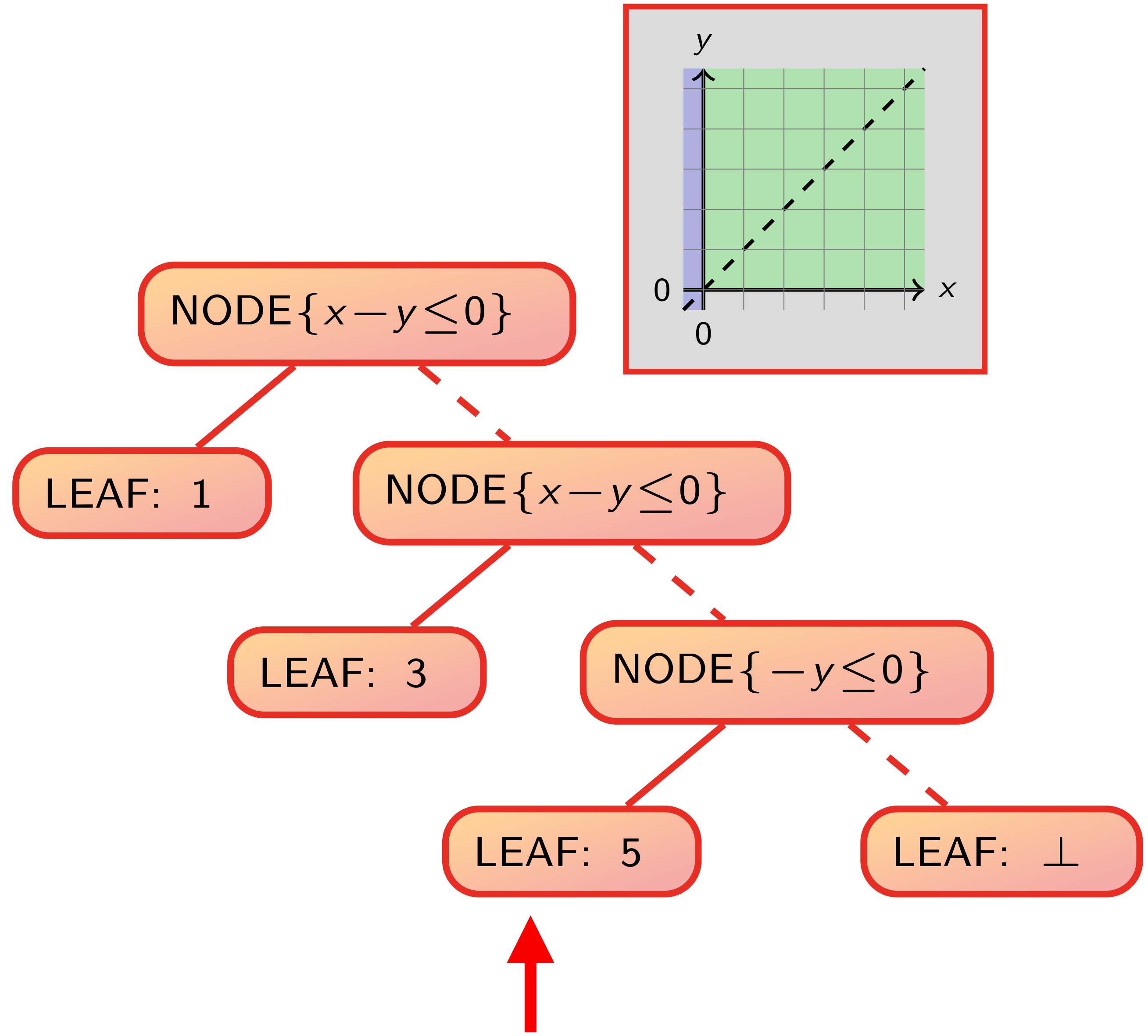

Precise Widening Operators for Convex Polyhedra*

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

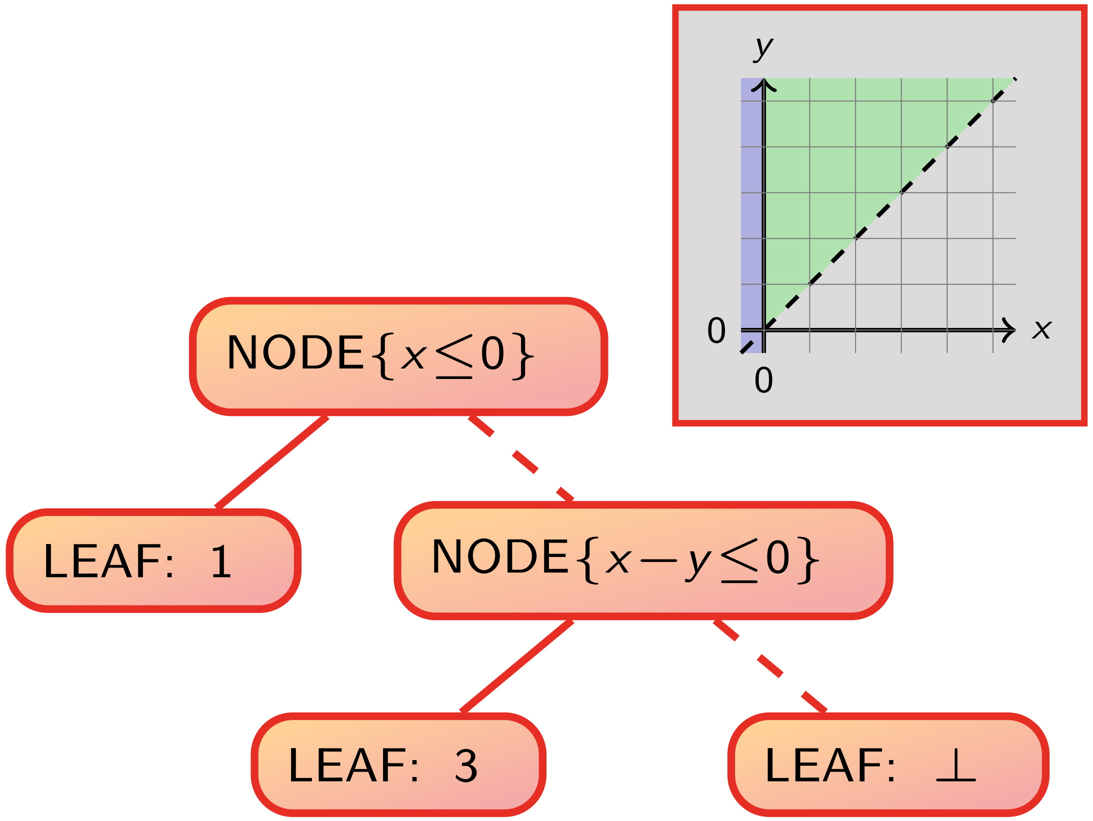
Fig. 2. The heuristics h_r improving on the standard widening.

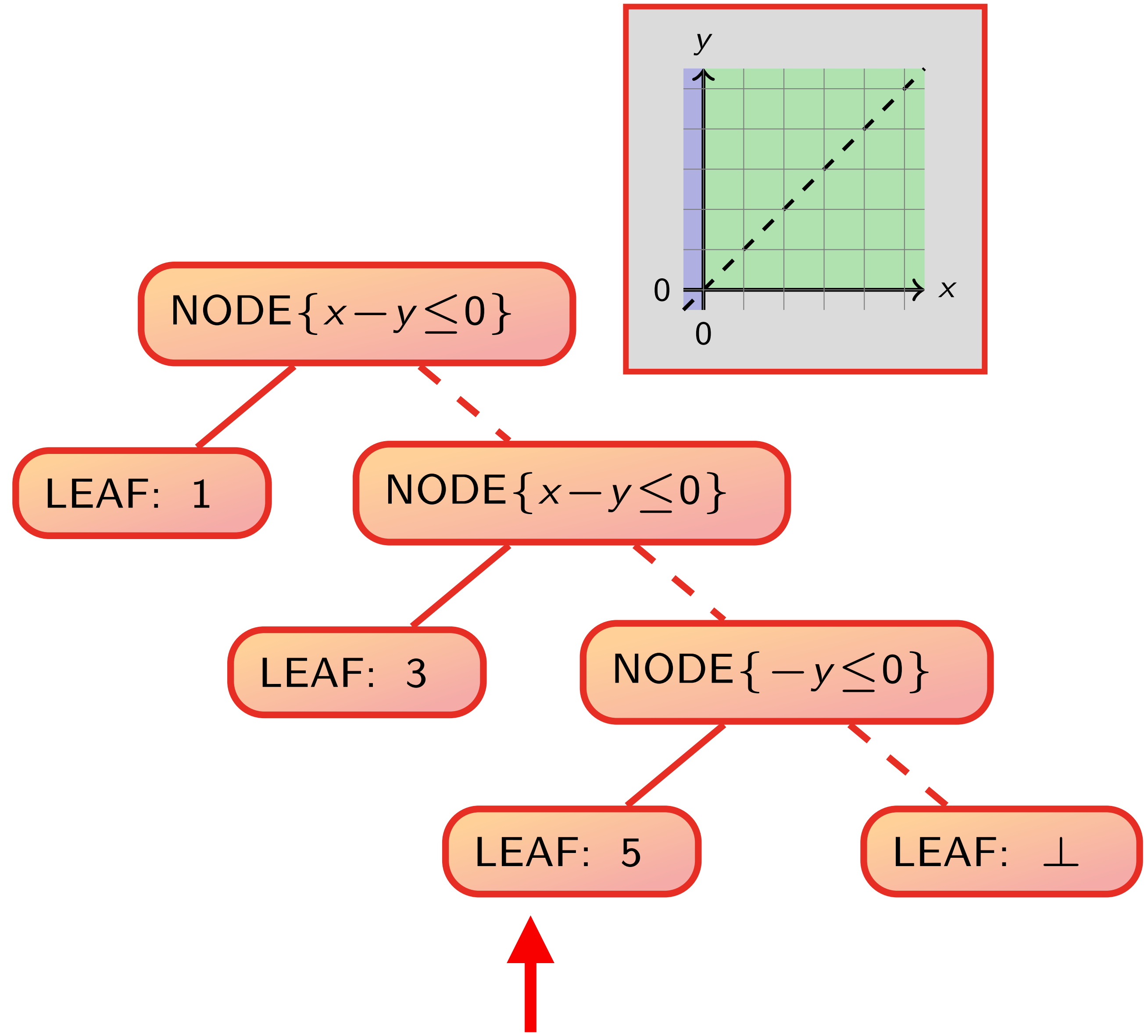
Widening

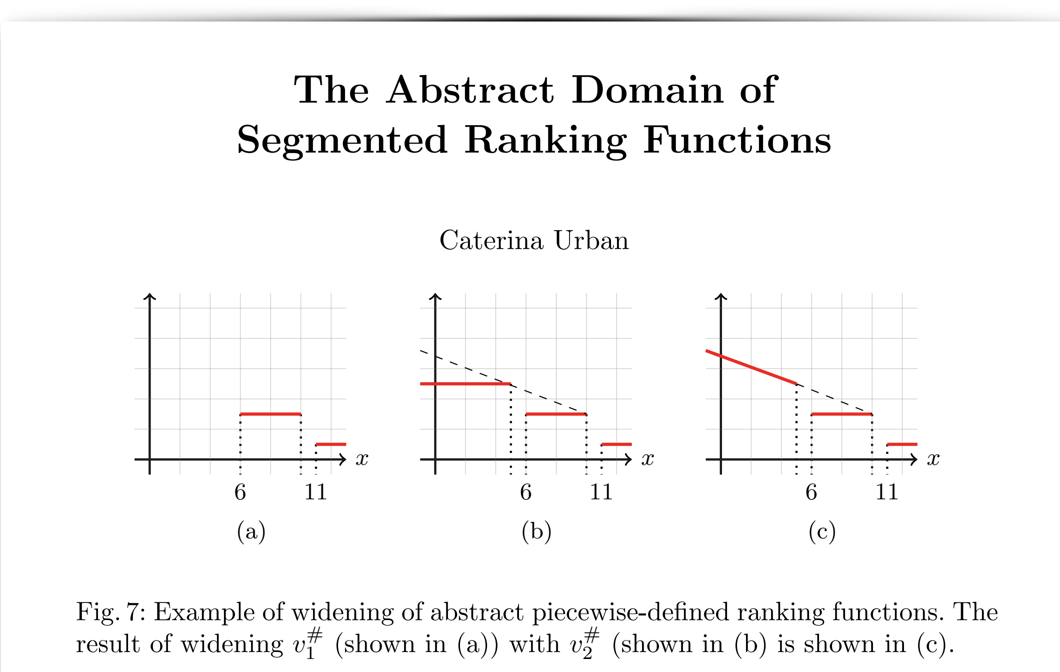

y
0
0
x
NODE{x≤0}
LEAF: 1
NODE{x−y≤0}
LEAF: 3
LEAF: ⊥


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

Widening





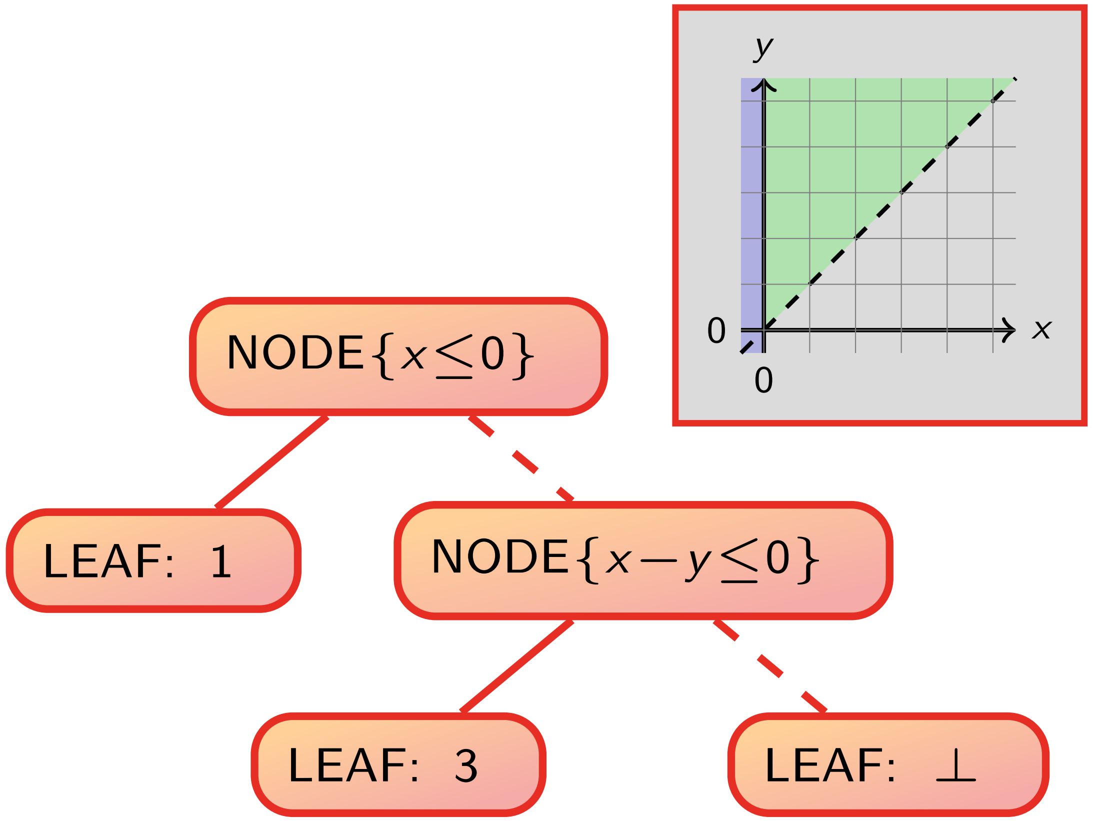


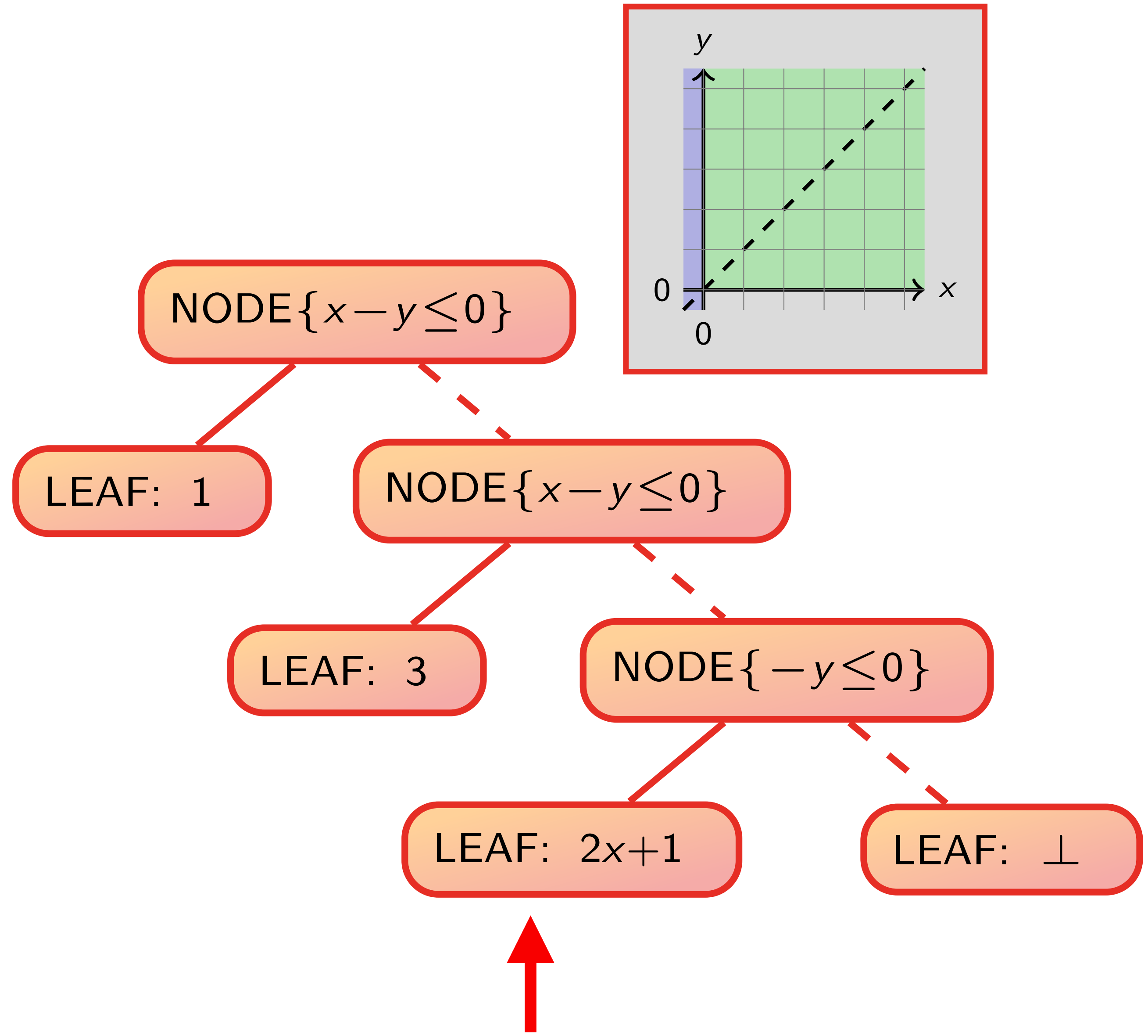
The Abstract Domain of
Segmented Ranking Functions

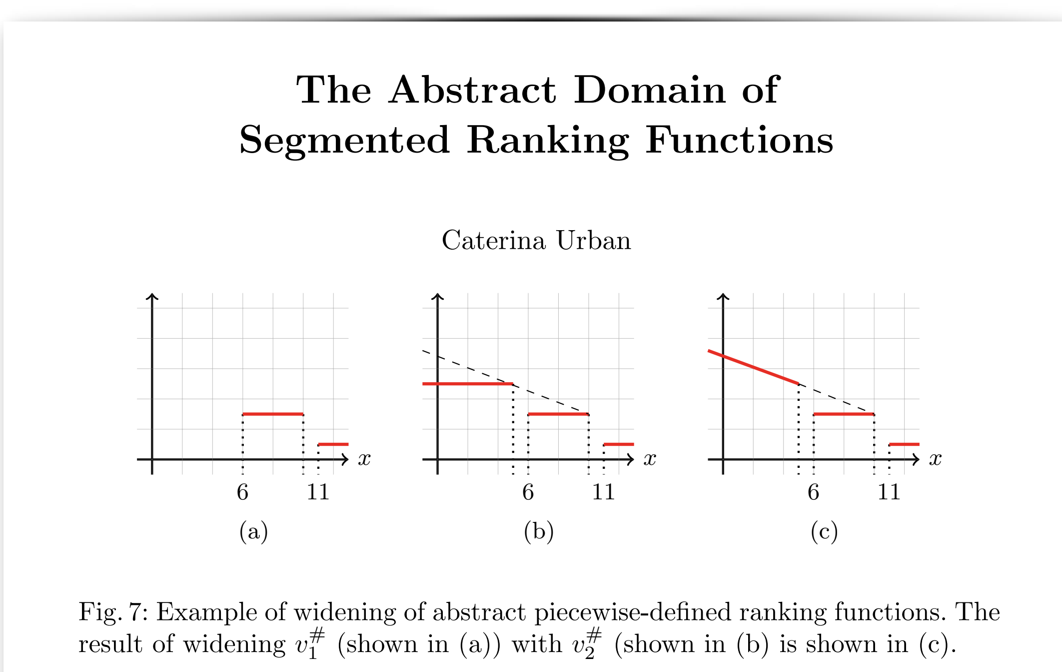
Caterina Urban

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

Widening



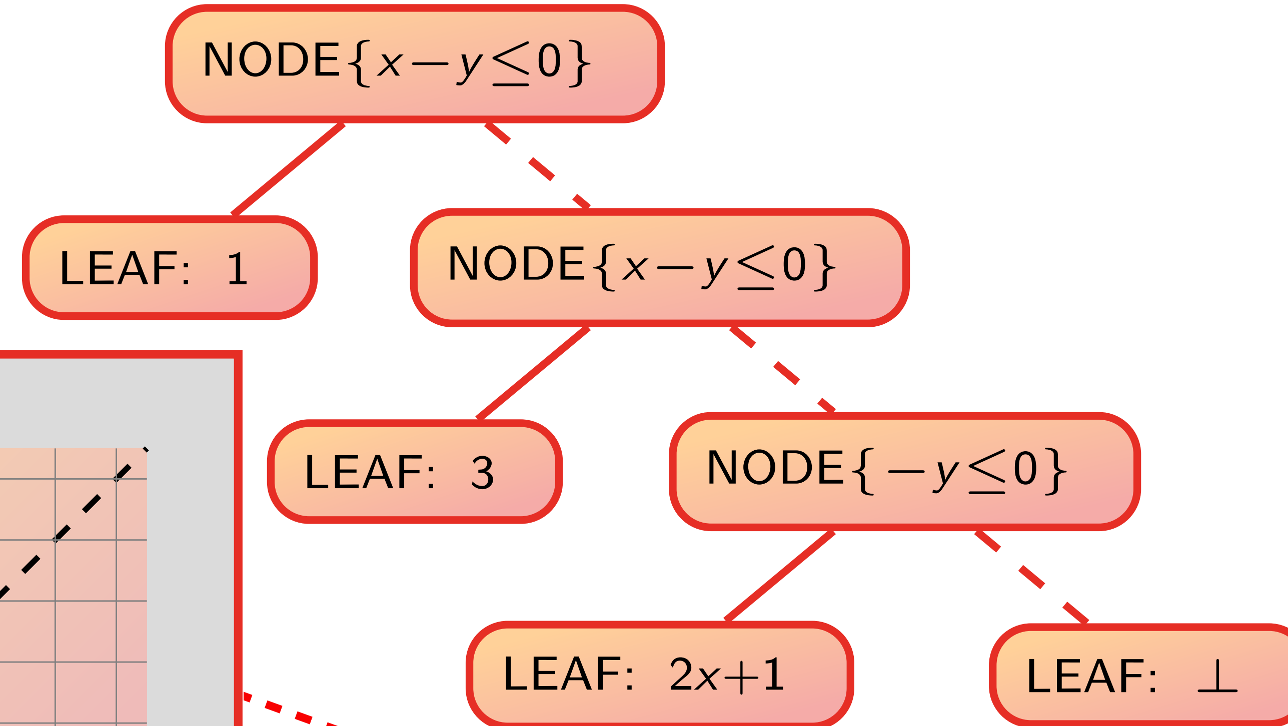




The Abstract Domain of
Segmented Ranking Functions

Caterina Urban

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



Example

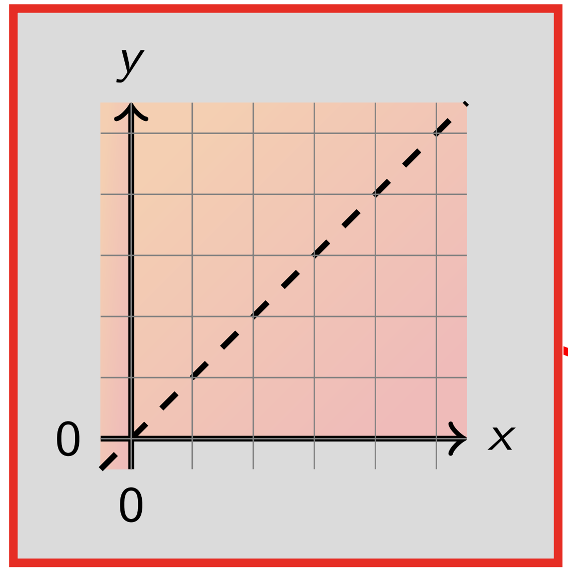
int : x, y

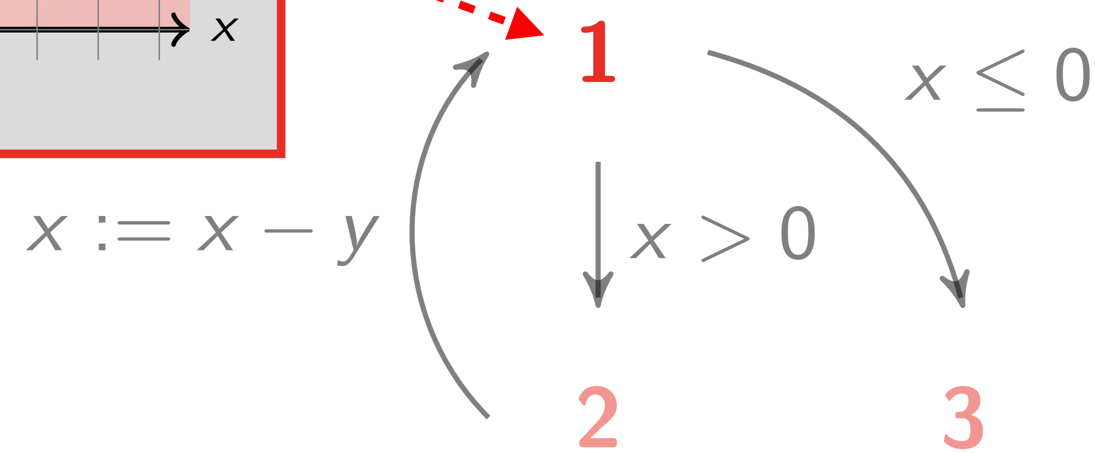
while [1]$(x > 0)$ do

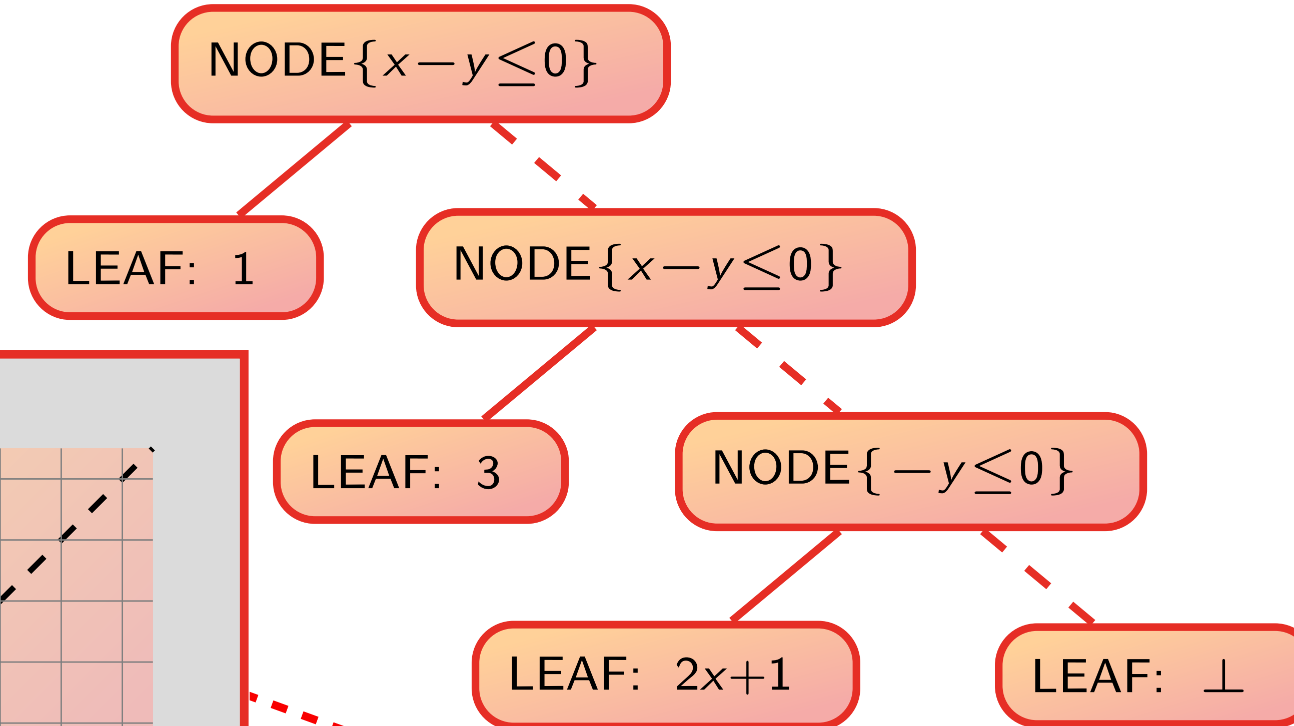
  [2]$x := x - y$

od[3]

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







Example

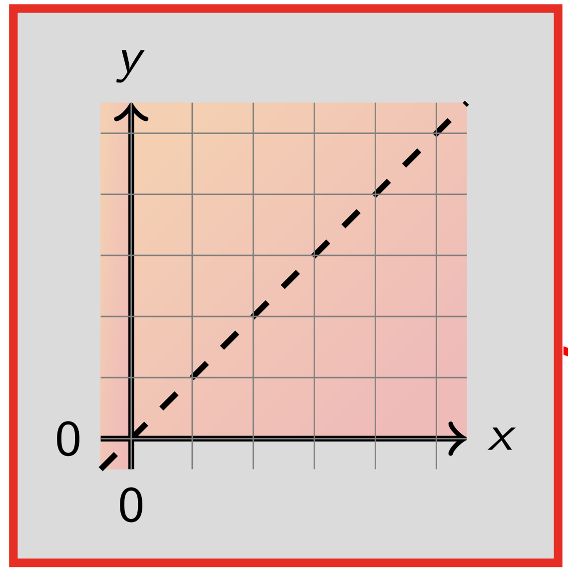
int : x, y

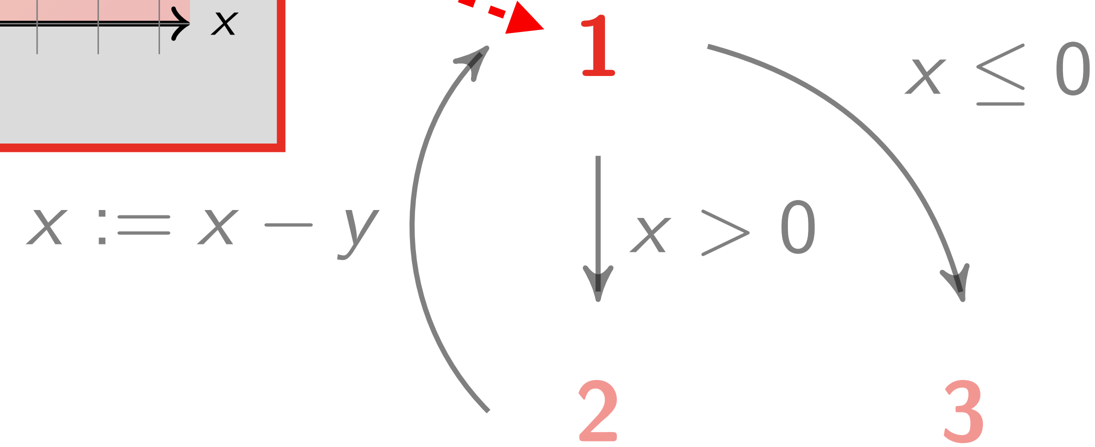
while [1]$(x > 0)$ do

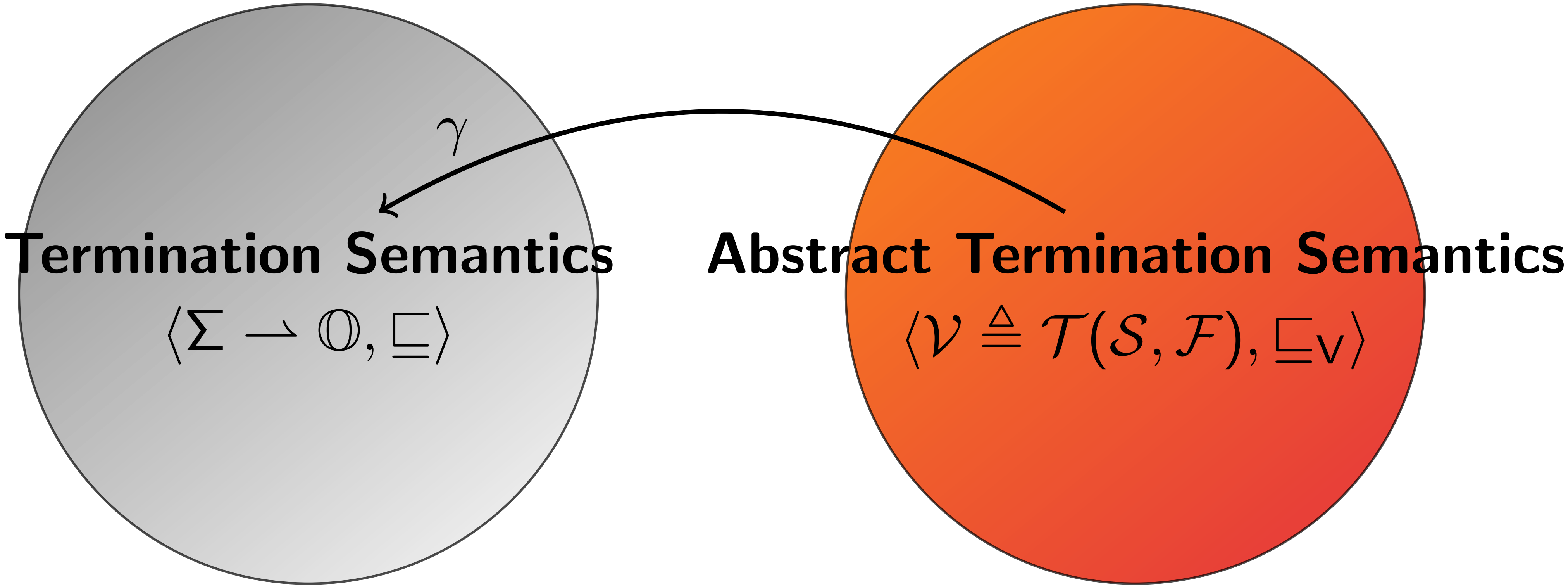
  [2]$x := x - y$

od[3]

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

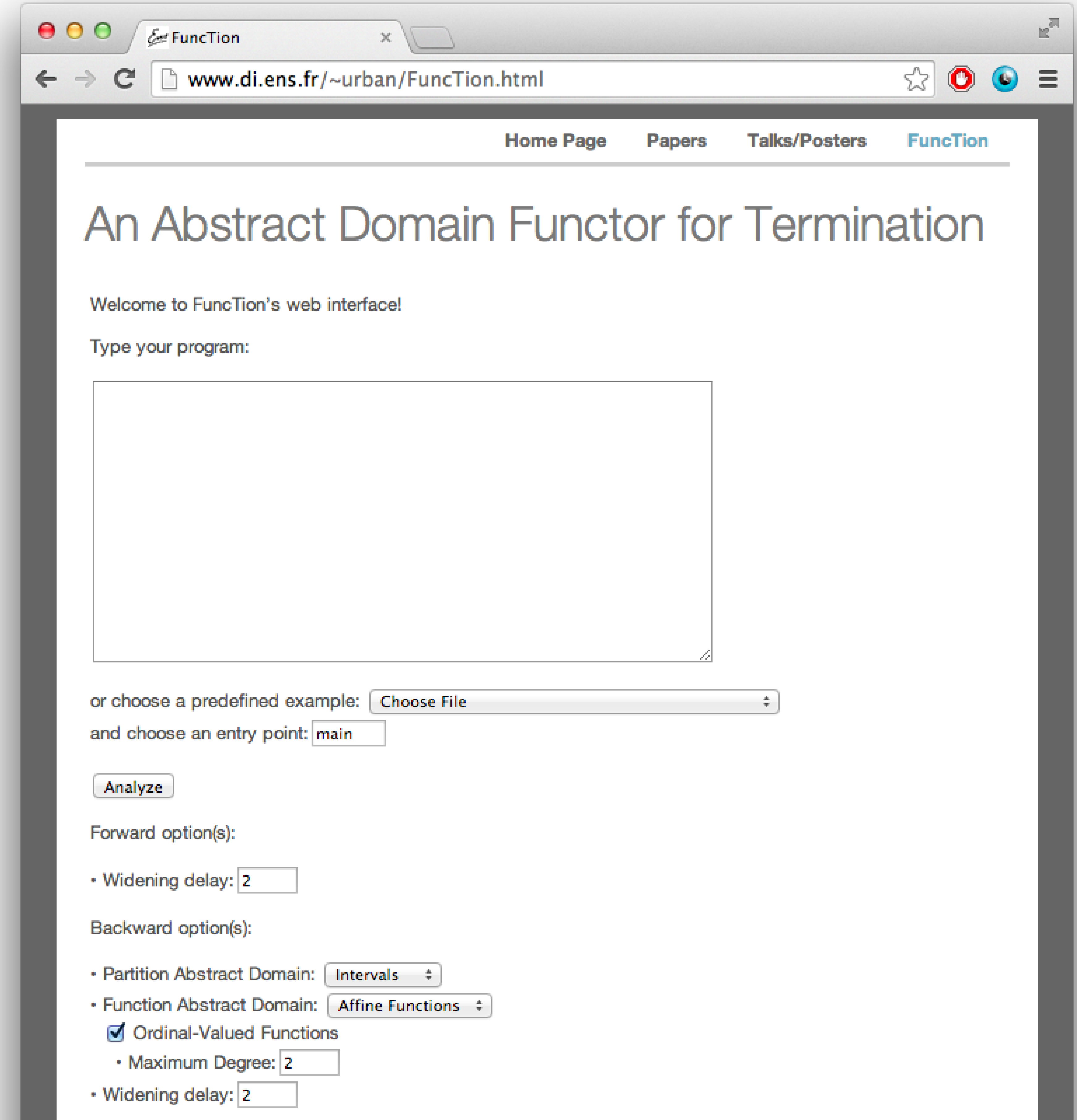






Theorem (Soundness)

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


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

Experiments

Benchmark: 87 terminating C programs collected from the literature

Tools:

- FuncTion
- AProVE
- T2
- Ultimate Büchi Automizer

Result:

	Tot	FuncTion	AProVE	T2	Ultimate	Time	Timeouts
FuncTion	51	–	8	8	3	6s	5
AProVE	60	17	–	7	2	35s	19
T2	73	30	20	–	3	2s	0
Ultimate	79	31	21	9	–	9s	1

Conclusions

- family of **abstract domains** for program termination
  - piecewise-defined ranking functions
  - backward analysis
  - sufficient preconditions for termination
- instances based on **decision trees**

Future Work

- **more abstract domains**
  - non-linear ranking functions
  - better widening
- **fair termination**
- other **liveness** properties

Conclusions

- family of **abstract domains** for program termination
  - piecewise-defined ranking functions
  - backward analysis
  - sufficient preconditions for termination
- instances based on **decision trees**

Future Work

- **more abstract domains**
  - non-linear ranking functions
  - better widening
- **fair termination**
- other **liveness** properties

Thank You!