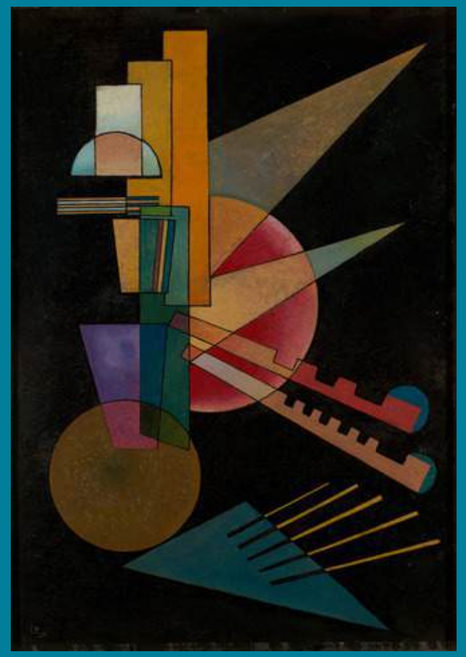


Kandinsky - Abstract Interpretation, 1925

Abstract Interpretation

and Applications in Security, Data Science, and Machine Learning

OPLSS 2025

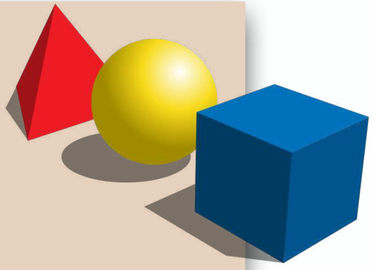
Caterina Urban
Inria & École Normale Supérieure | Université PSL

Termination Static Analysis

Abstract Program Termination Semantics

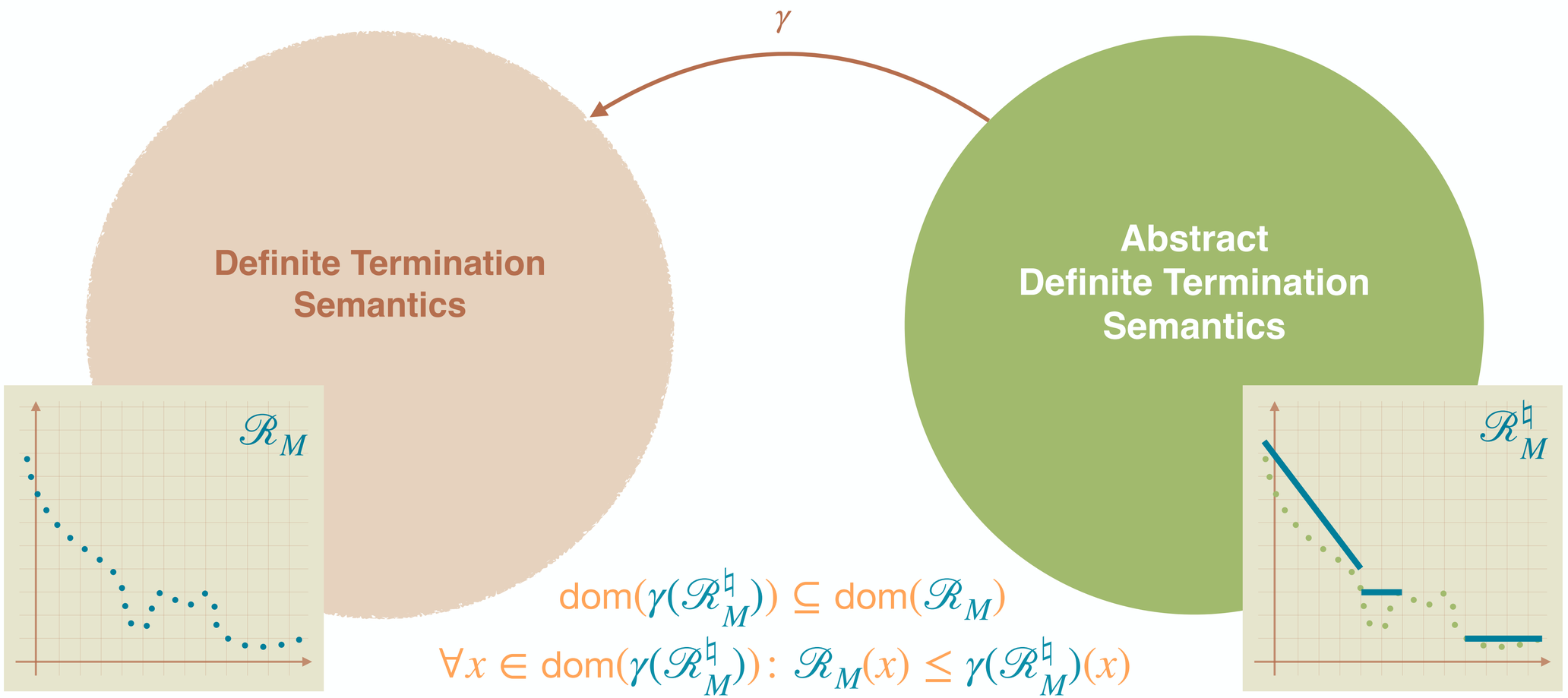
practical tools
targeting specific programs

abstract semantics, abstract domains
algorithmic approaches to decide program properties

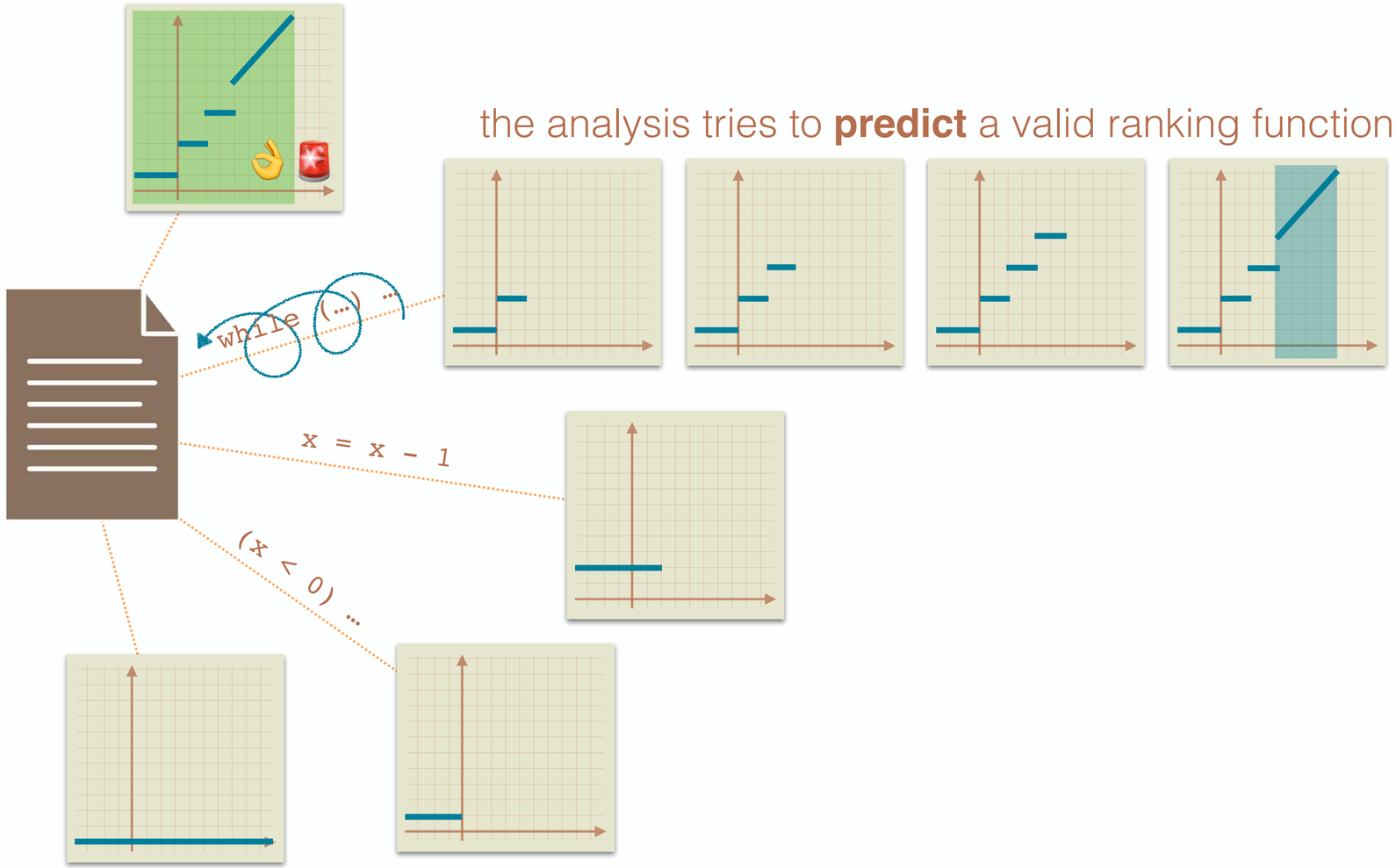


concrete semantics
mathematical models of the program behavior

Piecewise-Defined Ranking Functions



Termination Static Analysis

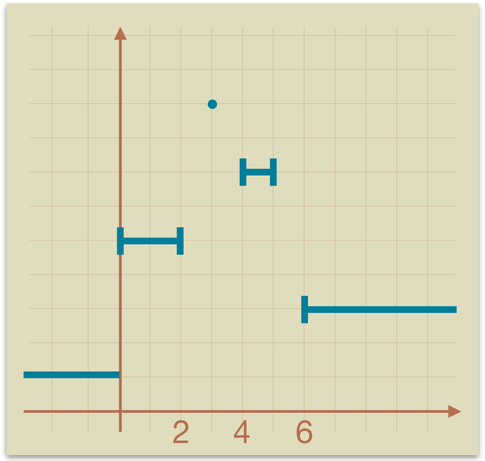


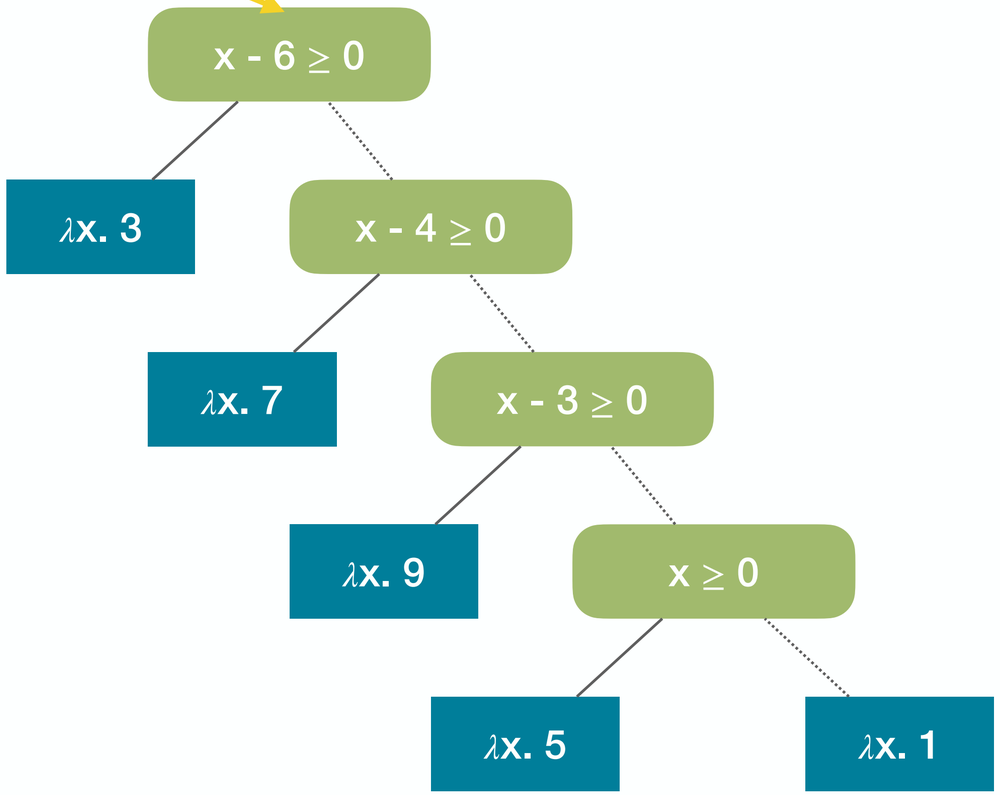
Piecewise-Defined Function Domain

$\langle \mathscr{A}, \preccurlyeq_A \rangle$

Example

[1]x ← [-∞, +∞]
while [2](x ≥ 0) **do**
    [3]x ← - 2 · x + 10
done[4]



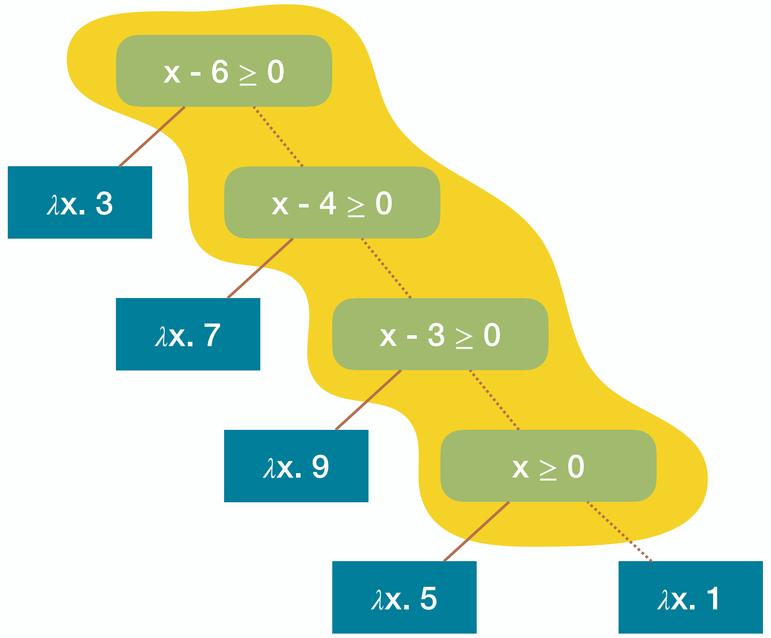


Piecewise-Defined Function Domain

Linear Constraints Auxiliary Abstract Domain

- Parameterized by an ***underlying numerical abstract domain*** $\langle \mathscr{D}, \sqsubseteq_D \rangle$ (e.g., intervals, polyhedra):

$$\langle \mathscr{P}(\mathscr{C}/\equiv_C), \sqsubseteq_D \rangle \quad \underset{\alpha_C}{\overset{\gamma_C}{\rightleftarrows}} \quad \langle \mathscr{D}, \sqsubseteq_D \rangle$$



Example:

$$X \to [-\infty, 3],\ Y \to [0, \infty] \quad \xrightarrow{\gamma_C} \quad \{3 - X \geq 0,\ Y \geq 0\}$$

- $\mathscr{C}$ is a set of linear constraints *in canonical form*, equipped with a total order $\leq_C$:

$$\mathscr{C} \overset{\text{def}}{=} \{c_1 \cdot X_1 + c_k \cdot X_k + c_{k+1} \geq 0 \mid X_1, \ldots, X_k \in \mathbb{V} \wedge c_1, \ldots, c_{k+1} \in \mathbb{Z} \wedge \gcd(|c_1|, \ldots, |c_{k+1}|) = 1\}$$

Natural-Valued Ranking Functions

Piecewise-Defined Function Domain

Functions Auxiliary Abstract Domain

- Parameterized by an ***underlying numerical abstract domain*** $\langle \mathscr{D}, \sqsubseteq_D \rangle$

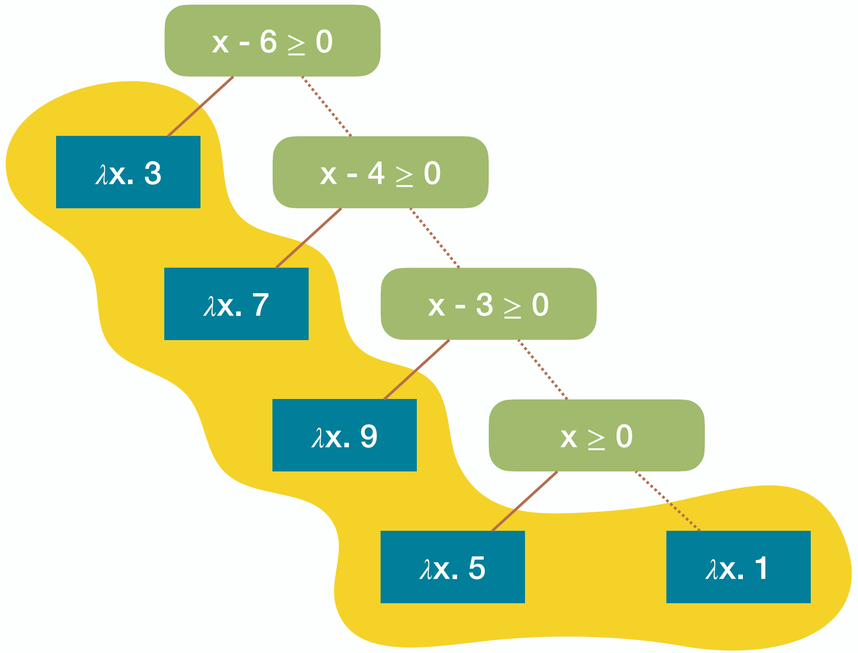
- $\mathscr{F} \stackrel{\text{def}}{=} \{ \bot_F \} \cup (\mathbb{Z}^{|\mathbb{V}|} \to \mathbb{N}) \cup \{ \top_F \}$

  We consider **affine functions**:

  $$\mathscr{F}_A \stackrel{\text{def}}{=} \{ \bot_F \} \cup \{ f \colon \mathbb{Z}^{|\mathbb{V}|} \to \mathbb{N} \mid$$

  $$f(X_1, \ldots, X_k) = \sum_{i=1}^{k} m_i \cdot X_i + q$$

  $$\} \cup \{ \top_F \}$$



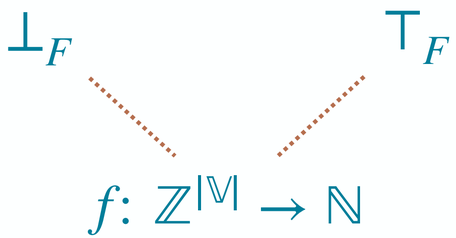
Piecewise-Defined Function Domain

Functions Auxiliary Abstract Domain (continue)

- **approximation order** $\preccurlyeq_F[D]$, where $D \in \mathscr{D}$:
  - between defined leaf nodes:

    $$f_1 \preccurlyeq_F[D] f_2 \overset{\text{def}}{=} \forall \rho \in \gamma_D(D) : f_1(\ldots, \rho(X_i), \ldots) \leq f_2(\ldots, \rho(X_i), \ldots)$$

  - otherwise (i.e., when one or both leaf nodes are undefined):



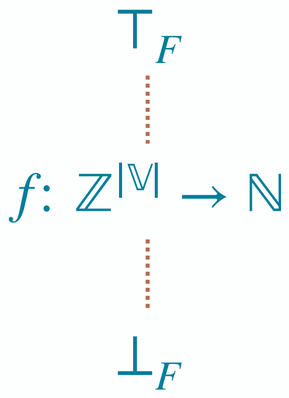
Piecewise-Defined Function Domain

Functions Auxiliary Abstract Domain (continue)

- **computational order** $\sqsubseteq_F[D]$, where $D \in \mathscr{D}$:
  - between defined leaf nodes:

    $$f_1 \preccurlyeq_F[D] f_2 \overset{\text{def}}{=} \forall \rho \in \gamma_D(D) : f_1(\ldots, \rho(X_i), \ldots) \leq f_2(\ldots, \rho(X_i), \ldots)$$

  - otherwise (i.e., when one or both leaf nodes are undefined):



Piecewise-Defined Functions Domain

$$\mathscr{A} \overset{\text{def}}{=} \{\text{LEAF}: f \mid f \in \mathscr{F}\} \cup \{\text{NODE}\{c\}: t_1; t_2 \mid c \in \mathscr{C} \wedge t_1, t_2 \in \mathscr{A}\}$$

- **concretization function** $\gamma_A : \mathscr{A} \to (\mathscr{E} \rightharpoonup \mathbb{O})$:

  $$\gamma_A(t) \overset{\text{def}}{=} \overline{\gamma}_A[\varnothing](t)$$

  where $\overline{\gamma}_A : \mathscr{P}(\mathscr{C} / \equiv_C) \to \mathscr{A} \to (\mathscr{E} \rightharpoonup \mathbb{O})$:

  $$\overline{\gamma}_A[C](\text{LEAF}: f) \overset{\text{def}}{=} \gamma_F[\alpha_C(C)](f)$$
  $$\overline{\gamma}_A[C](\text{NODE}\{c\}: t_1; t_2) \overset{\text{def}}{=} \overline{\gamma}_A[C \cup \{c\}](t_1) \dot{\cup} \overline{\gamma}_A[C \cup \{\neg c\}](t_2)$$

  and $\gamma_F : \mathscr{D} \to \mathscr{F} \to (\mathscr{E} \rightharpoonup \mathbb{O})$:

  $$\gamma_F[D](\bot_F) \overset{\text{def}}{=} \dot{\varnothing}$$
  $$\gamma_F[D](f) \overset{\text{def}}{=} \lambda\rho \in \gamma_D(D) : f(\ldots, \rho(X_i), \ldots)$$
  $$\gamma_F[D](\top_F) \overset{\text{def}}{=} \dot{\varnothing}$$

Piecewise-Defined Functions Domain

Abstract Domain Operators

- They manipulate elements in $\mathscr{A}_{\mathsf{NIL}} \stackrel{\text{def}}{=} \{\mathsf{NIL}\} \cup \mathscr{A}$

- The **binary operators** rely on a tree unification algorithm
  - approximation order $\preccurlyeq_A$ and computational order $\sqsubseteq_A$
  - approximation join $\curlyvee_A$ and computational join $\sqcup_A$
  - meet $\curlywedge_A$
  - widening $\triangledown_A$

- The **unary operators** rely on a tree pruning algorithm
  - assignment $\overleftarrow{\mathsf{ASSIGN}}_A[\![X \leftarrow e]\!]$
  - test $\mathsf{FILTER}_A[\![e]\!]$

Piecewise-Defined Functions Domain

Abstract Domain Operators

- They manipulate elements in $\mathscr{A}_{\mathsf{NIL}} \stackrel{\text{def}}{=} \{\mathsf{NIL}\} \cup \mathscr{A}$

- The **binary operators** rely on a **tree unification algorithm**
  - approximation order $\preccurlyeq_A$ and computational order $\sqsubseteq_A$
  - approximation join $\curlyvee_A$ and computational join $\sqcup_A$
  - meet $\curlywedge_A$
  - widening $\triangledown_A$

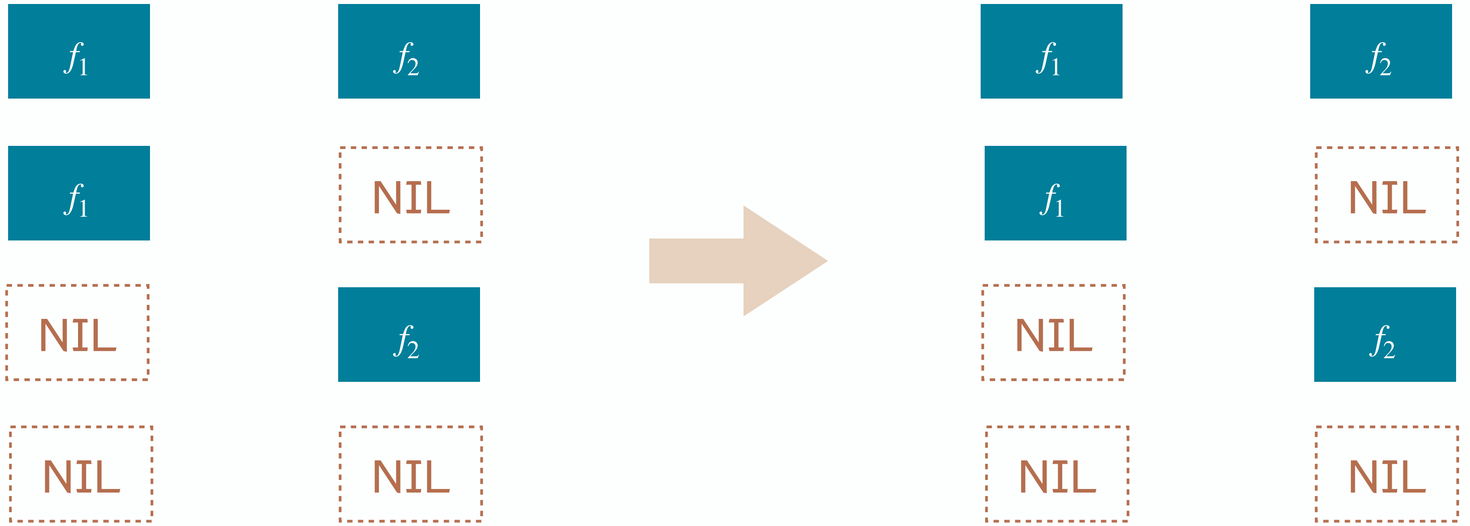
- The **unary operators** rely on a tree pruning algorithm
  - assignment $\overleftarrow{\mathsf{ASSIGN}}_A[\![X \leftarrow e]\!]$
  - test $\mathsf{FILTER}_A[\![e]\!]$

Piecewise-Defined Functions Domain

Tree Unification

Goal: find a **common refinement** for the given decision trees

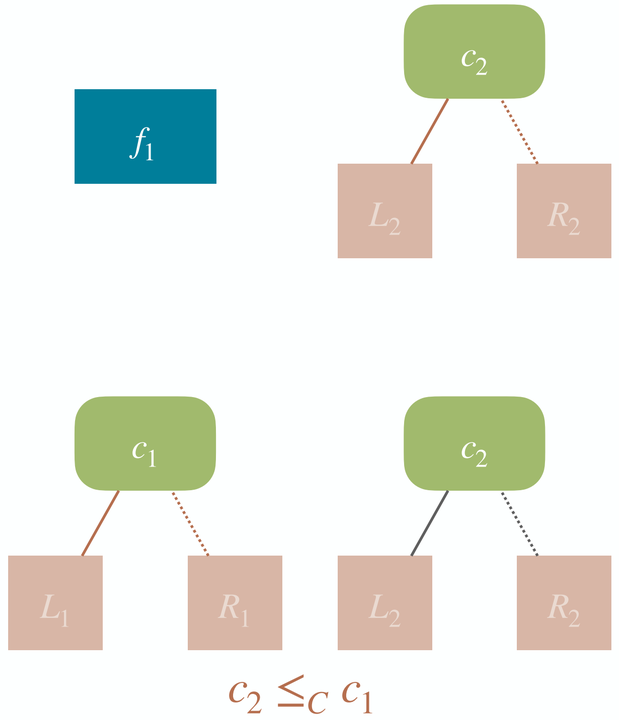
- Base cases:

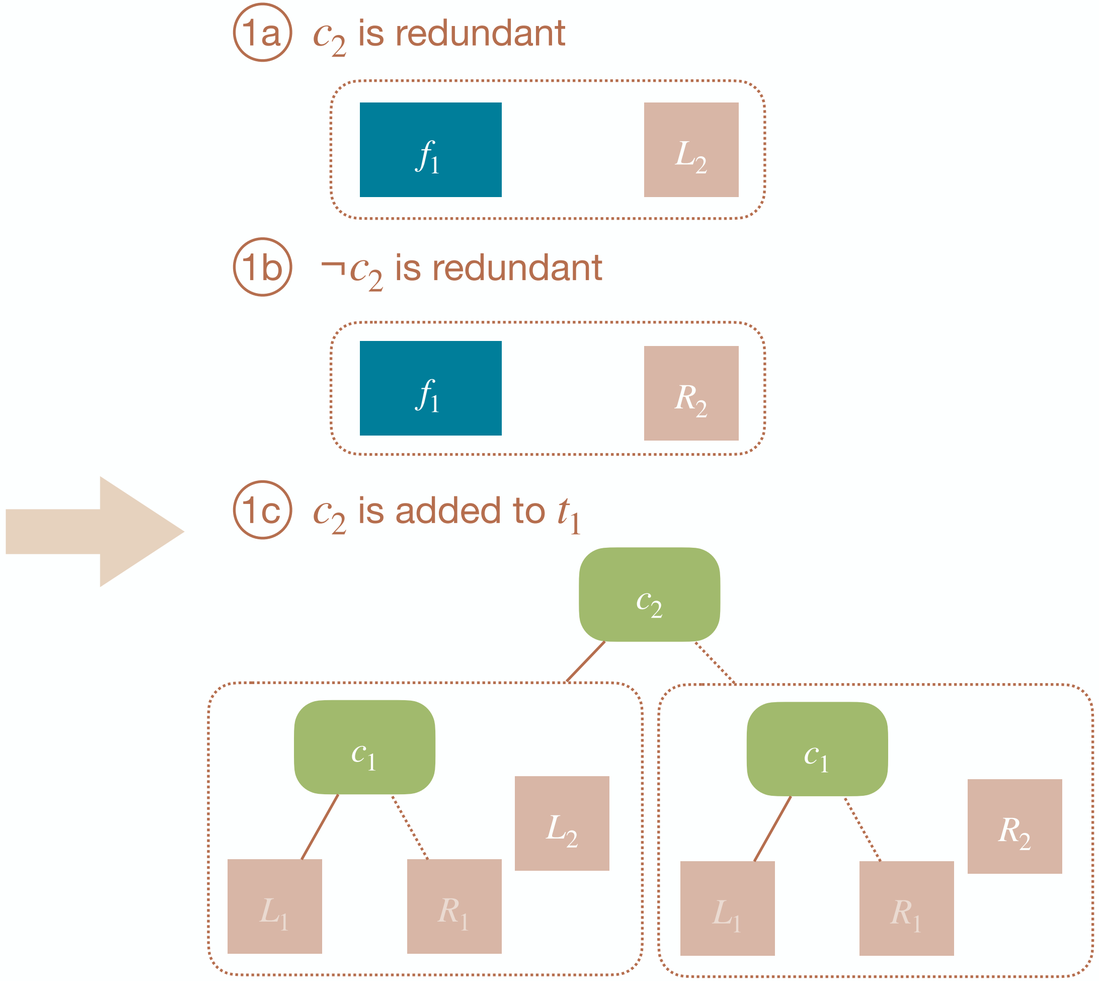


Piecewise-Defined Functions Domain

Tree Unification (continue)

- Case ①

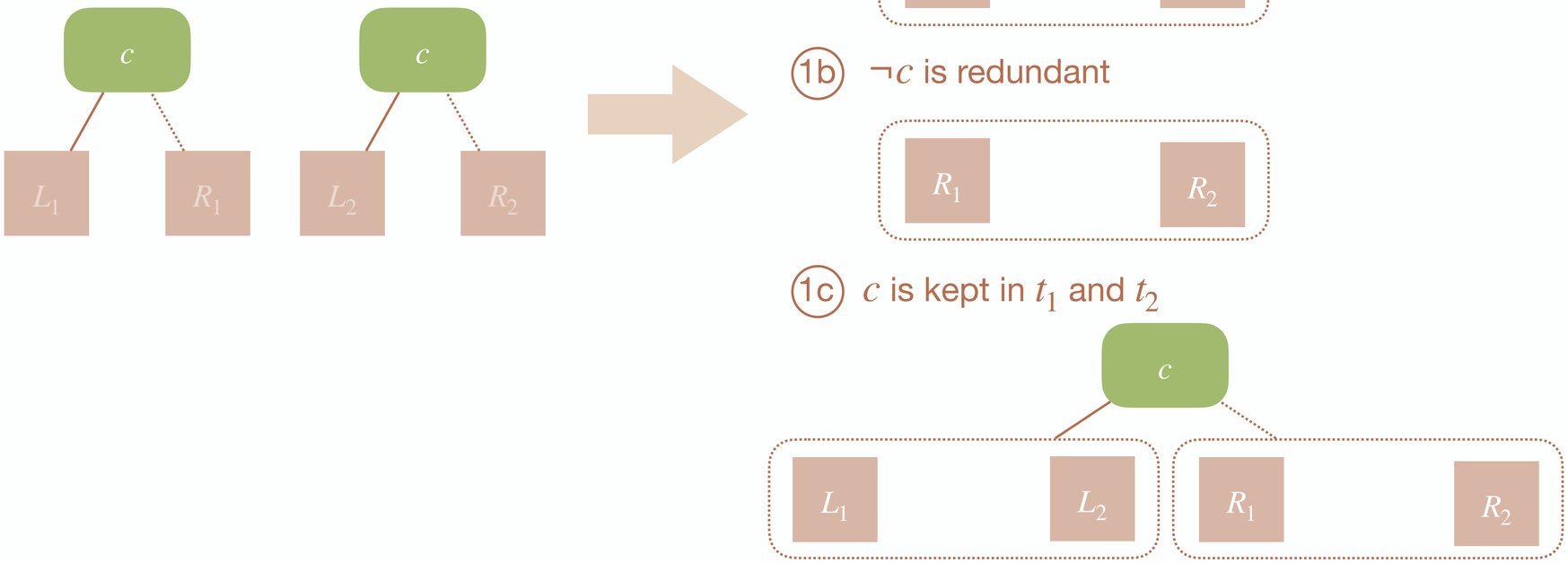




Piecewise-Defined Functions Domain

Tree Unification (continue)

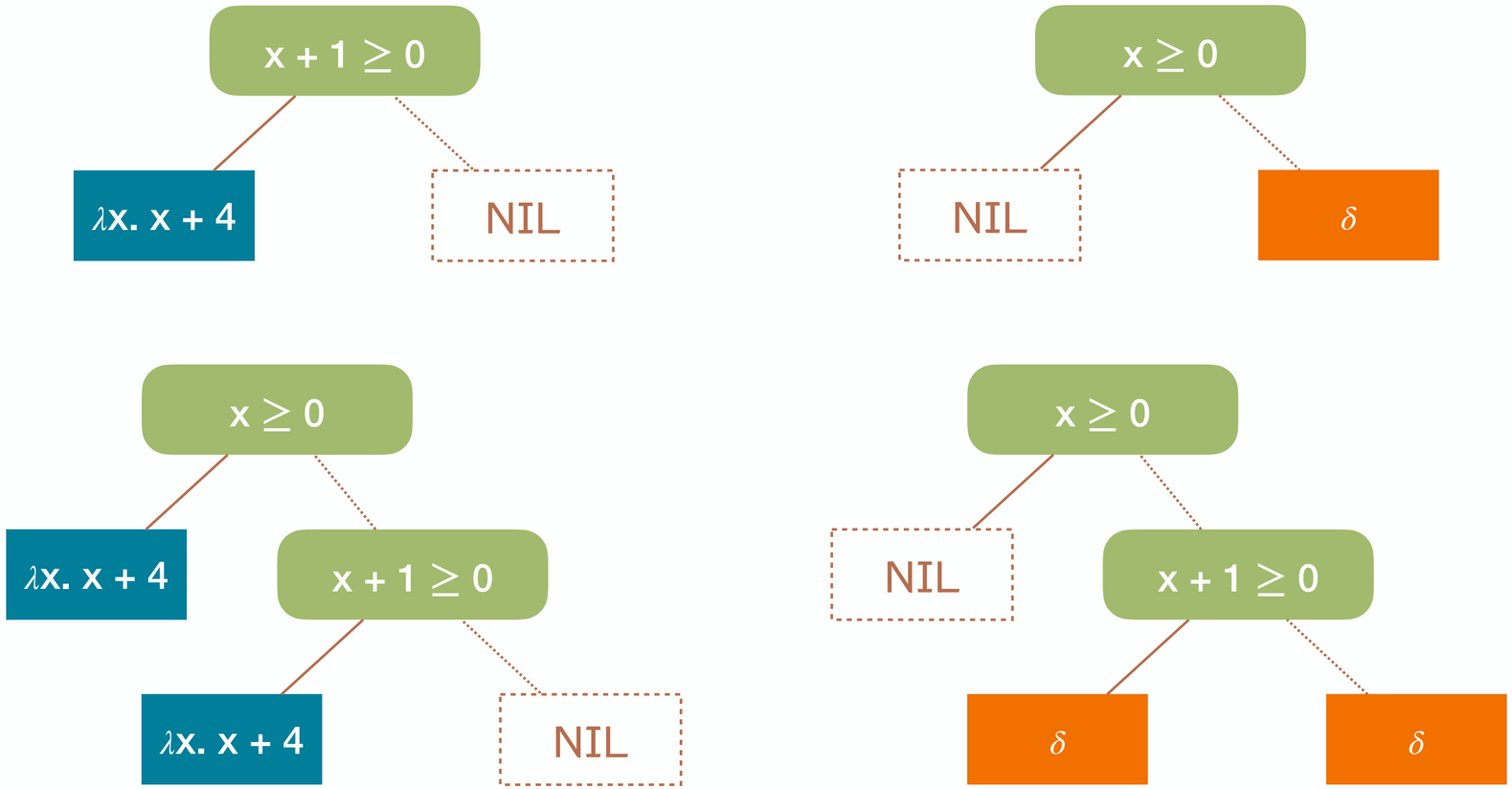
- Case ② (simmetric to ①)
- Case ③



Piecewise-Defined Functions Domain

Tree Unification (continue)

Example



Piecewise-Defined Functions Domain

Abstract Domain Operators

- They manipulate elements in $\mathscr{A}_{\mathsf{NIL}} \stackrel{\text{def}}{=} \{\mathsf{NIL}\} \cup \mathscr{A}$

- The **binary operators** rely on a tree unification algorithm

  - **approximation order $\preccurlyeq_A$ and computational order $\sqsubseteq_A$**
  - approximation join $\curlyvee_A$ and computational join $\sqcup_A$
  - meet $\curlywedge_A$
  - widening $\triangledown_A$

- The **unary operators** rely on a tree pruning algorithm

  - assignment $\overleftarrow{\mathsf{ASSIGN}}_A[\![X \leftarrow e]\!]$
  - test $\mathsf{FILTER}_A[\![e]\!]$

Piecewise-Defined Functions Domain

Order

1. Perform **tree unification**
2. Recursively descend the trees while *accumulating the linear constraints encountered along the paths* into a set of constraints C
3. Compare the leaf nodes using
   the **approximation order** $\preccurlyeq_F [\alpha_C(C)]$
   or the **computational order** $\sqsubseteq_F [\alpha_C(C)]$

The concretization function γ_A is monotonic with respect to $\preccurlyeq_A$:

Lemma

$$\forall t_1, t_2 \in \mathscr{A} : t_1 \preccurlyeq_A t_2 \Rightarrow \gamma_A(t_1) \preccurlyeq \gamma_A(t_2)$$

Piecewise-Defined Functions Domain

Abstract Domain Operators

- They manipulate elements in $\mathscr{A}_{\mathsf{NIL}} \stackrel{\text{def}}{=} \{\mathsf{NIL}\} \cup \mathscr{A}$

- The **binary operators** rely on a tree unification algorithm
  - approximation order $\preccurlyeq_A$ and computational order $\sqsubseteq_A$
  - **approximation join $\curlyvee_A$ and computational join $\sqcup_A$**
  - meet $\curlywedge_A$
  - widening $\triangledown_A$

- The **unary operators** rely on a tree pruning algorithm
  - assignment $\overleftarrow{\mathsf{ASSIGN}}_A[\![X \leftarrow e]\!]$
  - test $\mathsf{FILTER}_A[\![e]\!]$

Piecewise-Defined Functions Domain

Join

1. Perform **tree unification**

2. Recursively descend the trees while *accumulating the linear constraints encountered along the paths* into a set of constraints C

3. $\mathsf{NIL} \curlyvee_A t \stackrel{\text{def}}{=} t$
   $t \curlyvee_A \mathsf{NIL} \stackrel{\text{def}}{=} t$

4. Join the leaf nodes using
   the **approximation join** $\curlyvee_F [\alpha_C(C)]$
   or the **computational join** $\sqcup_F [\alpha_C(C)]$

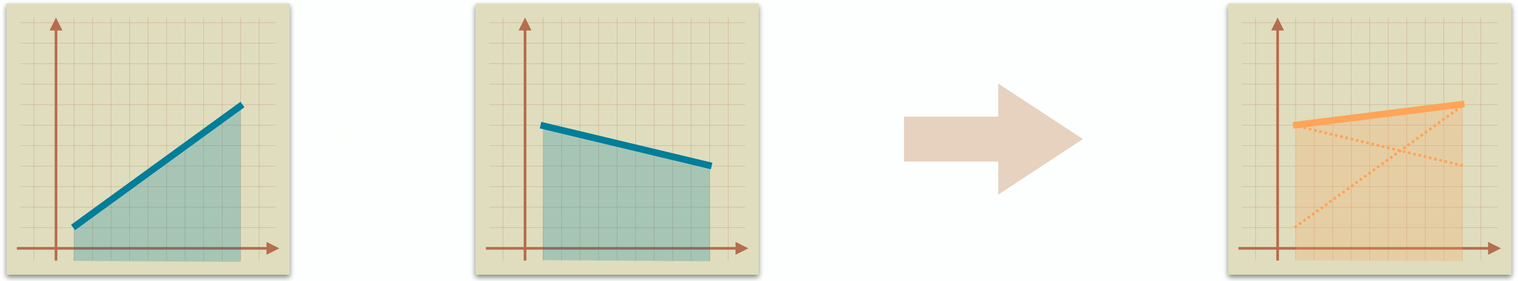
Piecewise-Defined Functions Domain

Join (continue)

- **approximation join** $\curlyvee_F [D]$, where $D \in \mathscr{D}$:
  - between <u>defined</u> leaf nodes:

$$f_1 \curlyvee_F [D] \; f_2 \overset{\text{def}}{=} \begin{cases} f = \lambda\rho \in \gamma_D(D) : \max(f_1(\ldots, \rho(X_i), \ldots), f_2(\ldots, \rho(X_i), \ldots)) & f \in \mathscr{F} \setminus \{ \bot_F, \top_F \} \\ \top_F & \text{otherwise} \end{cases}$$

Example:



Piecewise-Defined Functions Domain

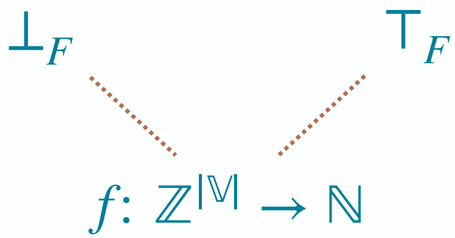
Join (continue)

- **approximation join** $\curlyvee_F [D]$, where $D \in \mathscr{D}$:
  - between <u>defined</u> leaf nodes:

$$f_1 \curlyvee_F [D]\ f_2 \overset{\text{def}}{=} \begin{cases} f = \lambda\rho \in \gamma_D(D) : \max(f_1(\ldots, \rho(X_i), \ldots), f_2(\ldots, \rho(X_i), \ldots)) & f \in \mathscr{F} \setminus \{\ \bot_F\ ,\ \top_F\ \} \\ \top_F & \text{otherwise} \end{cases}$$

  - otherwise (i.e., when one or both leaf nodes are <u>undefined</u>):

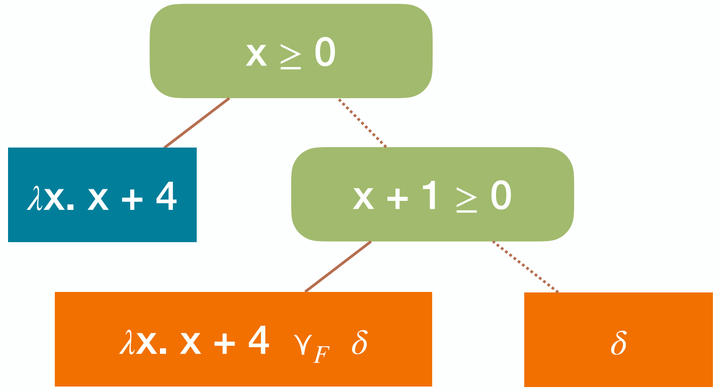
$$\begin{aligned} \bot_F \curlyvee_F [D]\ f &\overset{\text{def}}{=} \bot_F & f \in \mathscr{F} \setminus \{\ \top_F\ \} \\ f \curlyvee_F [D]\ \bot_F &\overset{\text{def}}{=} \bot_F & f \in \mathscr{F} \setminus \{\ \top_F\ \} \\ \top_F \curlyvee_F [D]\ f &\overset{\text{def}}{=} \top_F & f \in \mathscr{F} \setminus \{\ \bot_F\ \} \\ f \curlyvee_F [D]\ \top_F &\overset{\text{def}}{=} \top_F & f \in \mathscr{F} \setminus \{\ \bot_F\ \} \end{aligned}$$



Piecewise-Defined Functions Domain

Join (continue)

Example



Piecewise-Defined Functions Domain

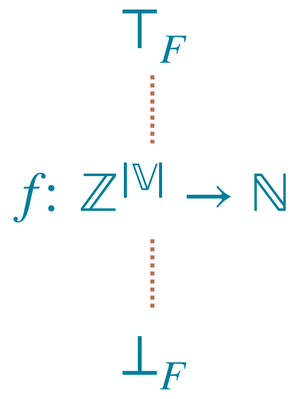
Join (continue)

- **computational join** $\sqcup_F [D]$, where $D \in \mathscr{D}$:
  - between defined leaf nodes:

$$f_1 \curlyvee_F [D]\ f_2 \overset{\text{def}}{=} \begin{cases} f = \lambda\rho \in \gamma_D(D) \colon \max(f_1(\ldots, \rho(X_i), \ldots), f_2(\ldots, \rho(X_i), \ldots)) & f \in \mathscr{F} \setminus \{\ \bot_F\ ,\ \top_F\ \} \\ \top_F & \text{otherwise} \end{cases}$$

  - otherwise (i.e., when one or both leaf nodes are undefined):

$$\begin{aligned} \bot_F \sqcup_F [D]\ f &\overset{\text{def}}{=} f && f \in \mathscr{F} \\ f \sqcup_F [D]\ \bot_F &\overset{\text{def}}{=} f && f \in \mathscr{F} \\ \top_F \sqcup_F [D]\ f &\overset{\text{def}}{=} \top_F && f \in \mathscr{F} \\ f \sqcup_F [D]\ \top_F &\overset{\text{def}}{=} \top_F && f \in \mathscr{F} \end{aligned}$$



Piecewise-Defined Functions Domain

Abstract Domain Operators

- They manipulate elements in $\mathscr{A}_{\mathsf{NIL}} \stackrel{\text{def}}{=} \{\mathsf{NIL}\} \cup \mathscr{A}$

- The **binary operators** rely on a tree unification algorithm
  - approximation order $\preccurlyeq_A$ and computational order $\sqsubseteq_A$
  - approximation join $\curlyvee_A$ and computational join $\sqcup_A$
  - **meet** $\curlywedge_A$
  - widening $\triangledown_A$

- The **unary operators** rely on a tree pruning algorithm
  - assignment $\overleftarrow{\mathsf{ASSIGN}}_A[\![X \leftarrow e]\!]$
  - test $\mathsf{FILTER}_A[\![e]\!]$

Piecewise-Defined Functions Domain

Meet

1. Perform **tree unification**

2. Recursively descend the trees while *accumulating the linear constraints encountered along the paths* into a set of constraints C

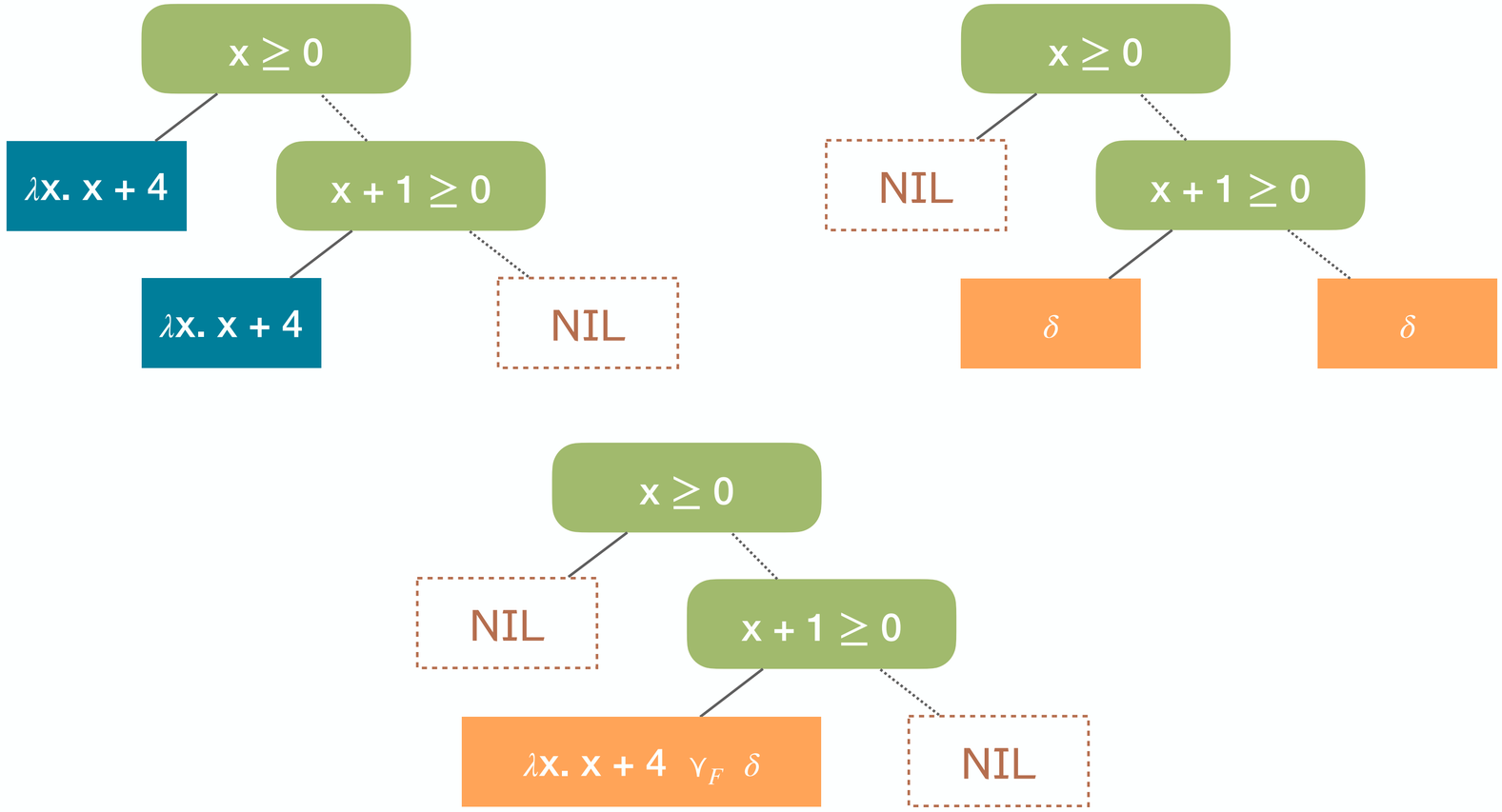
3. $\texttt{NIL} \curlyvee_A t \stackrel{\text{def}}{=} \texttt{NIL}$
   $t \curlyvee_A \texttt{NIL} \stackrel{\text{def}}{=} \texttt{NIL}$

4. Join the leaf nodes using the **approximation join** $\curlyvee_F [\alpha_C(C)]$

Piecewise-Defined Functions Domain

Meet (continue)

Example



Piecewise-Defined Functions Domain

Abstract Domain Operators

- They manipulate elements in $\mathscr{A}_{\mathsf{NIL}} \stackrel{\text{def}}{=} \{\mathsf{NIL}\} \cup \mathscr{A}$

- The **binary operators** rely on a tree unification algorithm
  - approximation order $\preccurlyeq_A$ and computational order $\sqsubseteq_A$
  - approximation join $\curlyvee_A$ and computational join $\sqcup_A$
  - meet $\curlywedge_A$
  - **widening** $\triangledown_A$

- The **unary operators** rely on a tree pruning algorithm
  - assignment $\overleftarrow{\mathsf{ASSIGN}}_A[\![X \leftarrow e]\!]$
  - test $\mathsf{FILTER}_A[\![e]\!]$

Piecewise-Defined Functions Domain

Widening

Goal: try to **predict** a valid ranking function

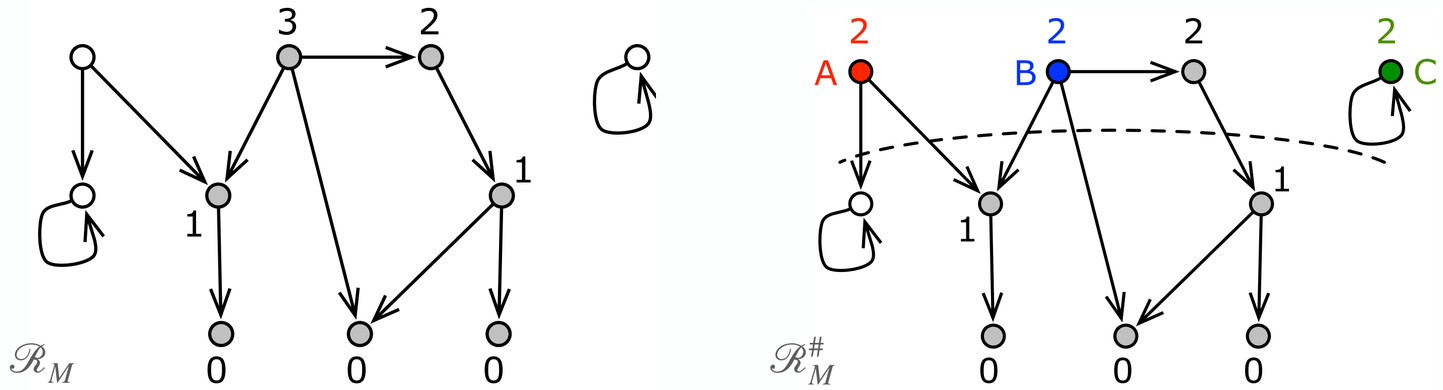
The prediction can (temporarily) be wrong!, i.e., it can

- *under-approximate* the value of $\mathscr{R}_M$

and/or

- *over-approximate* the domain dom($\mathscr{R}_M$) of $\mathscr{R}_M$

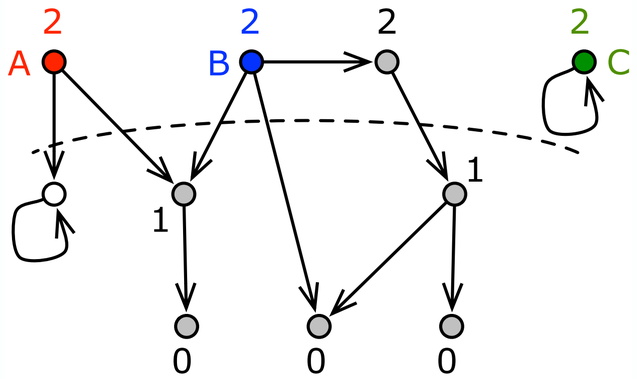
Example



Piecewise-Defined Functions Domain

Widening (continue)

1. Check for **case A** (i.e., wrong domain predictions)
2. Perform **domain widening**
3. Check for **case B or C** (i.e., wrong value predictions)
4. Perform **value widening**



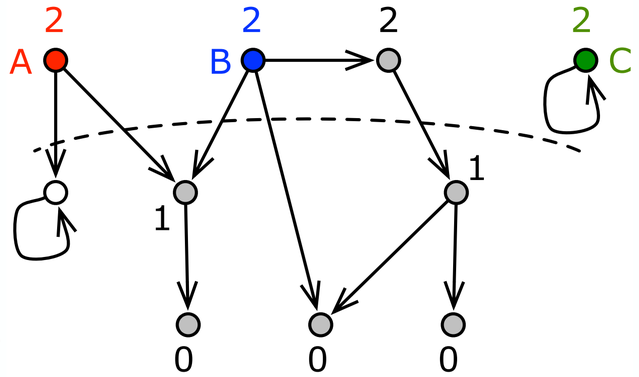
Piecewise-Defined Functions Domain

Widening (continue)

Check for Case A

Lemma

Let $\text{dom}(\gamma_A(\mathscr{R}_M^{\#n}(\ell)))\backslash\text{dom}(\mathscr{R}_M(\ell)) \neq \emptyset$. Then, in case A, we have $\text{dom}(\gamma_A(\mathscr{R}_M^{\#n+1}(\ell)))\backslash\text{dom}(\mathscr{R}_M(\ell)) \subset \text{dom}(\gamma_A(\mathscr{R}_M^{\#n}(\ell)))\backslash\text{dom}(\mathscr{R}_M(\ell))$.



1. Perform **tree unification**
2. Recursively descend the trees while *accumulating the linear constraints encountered along the paths* into a set of constraints C
3. f $\quad \bot_F \quad \Rightarrow \quad f \quad \top_F$

Piecewise-Defined Functions Domain

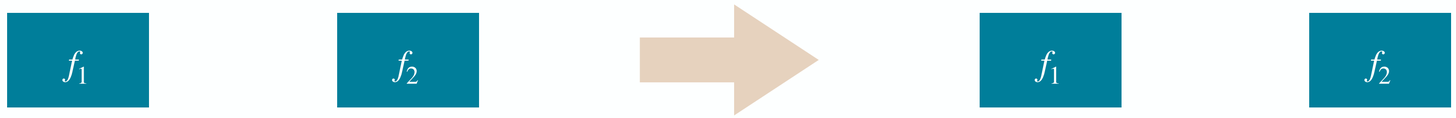
Widening (continue)

Domain Widening

Goal: **limit the size** of the decision trees

Left unification: variant of tree unification that forces the structure of t_1 on t_2

- Base case:

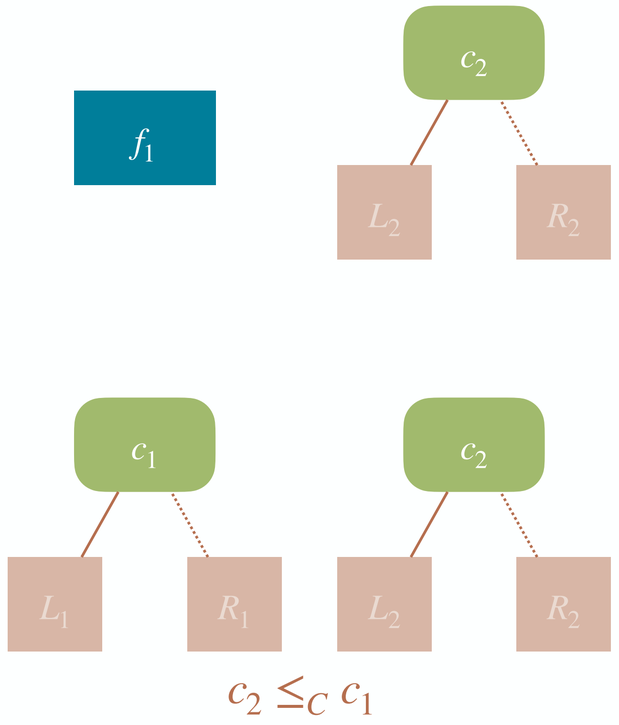


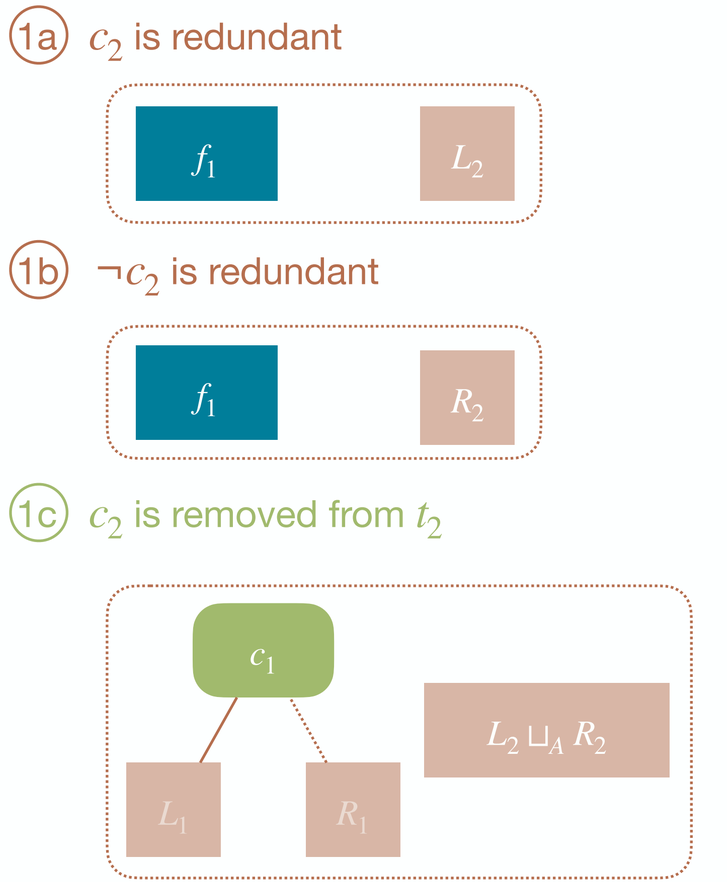
Piecewise-Defined Functions Domain

Widening (continue)

Domain Widening

- Case ①



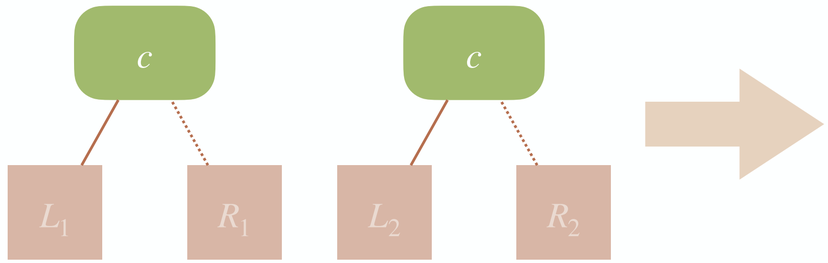


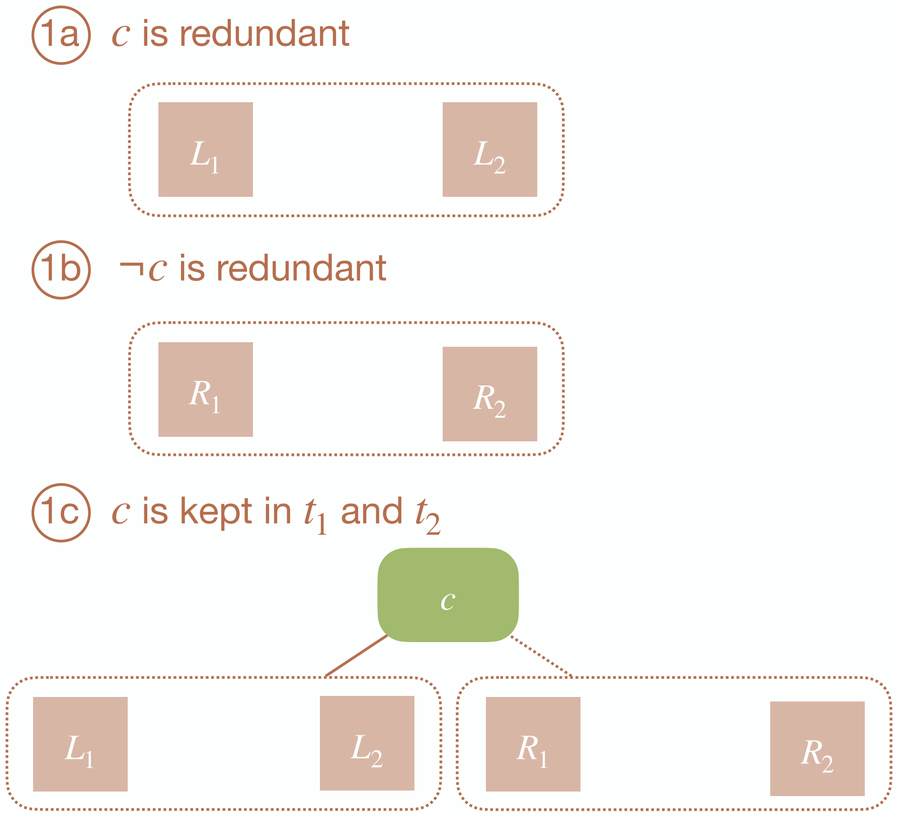
Piecewise-Defined Functions Domain

Widening (continue)

Domain Widening

- Case ② (as for tree unification)
- Case ③





Piecewise-Defined Functions Domain

Widening (continue)

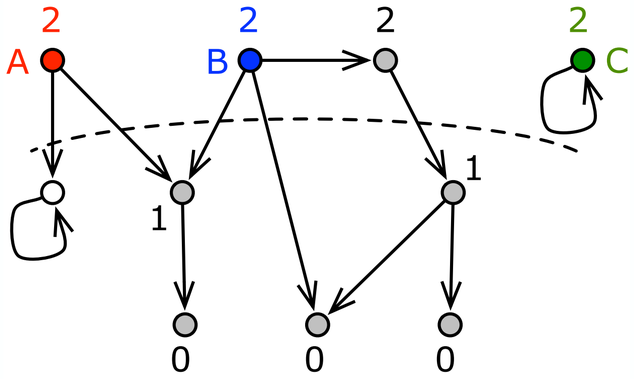
Check for Case B or C

Lemma

Let $\gamma_A(\mathscr{R}_M^{\#n}(\ell))(\overline{\rho}) < \mathscr{R}_M(\ell)(\overline{\rho})$ for some $\overline{\rho} \in \text{dom}(\mathscr{R}_M(\ell)) \cap \text{dom}(\gamma_A(\mathscr{R}_M^{\#n})(\ell))$ (case B). Then, there exists $\rho \in \text{dom}(\gamma_A(\mathscr{R}_M^{\#n+1}(\ell))) \cap \text{dom}(\mathscr{R}_M^{\#n}(\ell))$ such that $\gamma_A(\mathscr{R}_M^{\#n}(\ell))(\rho) < \gamma_A(\mathscr{R}_M^{\#n+1}(\ell))(\rho)$.

Lemma

Let $\text{dom}(\gamma_A(\mathscr{R}_M^{\#n}(\ell)))\backslash\text{dom}(\mathscr{R}_M(\ell)) \neq \emptyset$. Then, for all $\rho \in \text{dom}(\gamma_A(\mathscr{R}_M^{\#n}(\ell)))\backslash\text{dom}(\mathscr{R}_M(\ell))$ in case C, we have $\gamma_A(\mathscr{R}_M^{\#n}(\ell))(\rho) < \gamma_A(\mathscr{R}_M^{\#n+1}(\ell))(\rho)$.



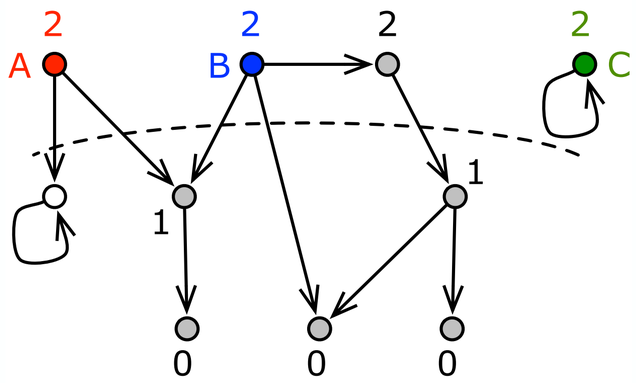
Piecewise-Defined Functions Domain

Widening (continue)

Check for Case B or C

1. Recursively descend the trees while *accumulating the linear constraints encountered along the paths* into a set of constraints C
2. f_1 f_2 → f_1 $\top_F$

$f_1 \in \mathscr{F} \backslash \{ \bot_F , \top_F \} \wedge f_2 \not\preccurlyeq_F [\alpha_c(C)] f_1$



Piecewise-Defined Functions Domain

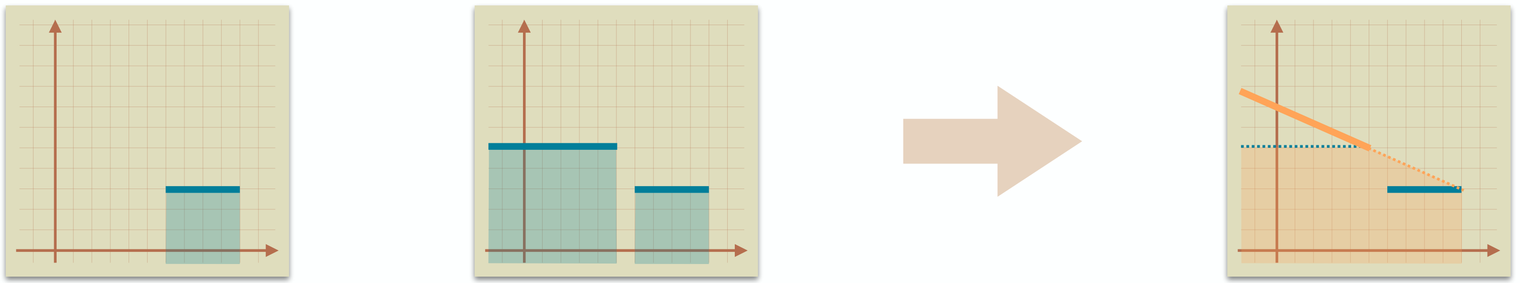
Widening (continue)

Value Widening

1. Recursively descend the trees while *accumulating the linear constraints encountered along the paths* into a set of constraints C

2. Widen each (defined) leaf node f with respect to each of their adjacent (defined) leaf node $\bar{f}$ using the **extrapolation operator** $\blacktriangledown_F [\alpha_C(\overline{C}), \alpha_C(C)]$, where $\overline{C}$ is the set of constraints along the path to $\bar{f}$

   Example:



Piecewise-Defined Functions Domain

Abstract Domain Operators

- They manipulate elements in $\mathscr{A}_{\mathsf{NIL}} \stackrel{\text{def}}{=} \{\mathsf{NIL}\} \cup \mathscr{A}$

- The **binary operators** rely on a tree unification algorithm

  - approximation order $\preccurlyeq_A$ and computational order $\sqsubseteq_A$
  - approximation join $\curlyvee_A$ and computational join $\sqcup_A$
  - meet $\curlywedge_A$
  - widening $\triangledown_A$

- The **unary operators** rely on a **tree pruning algorithm**

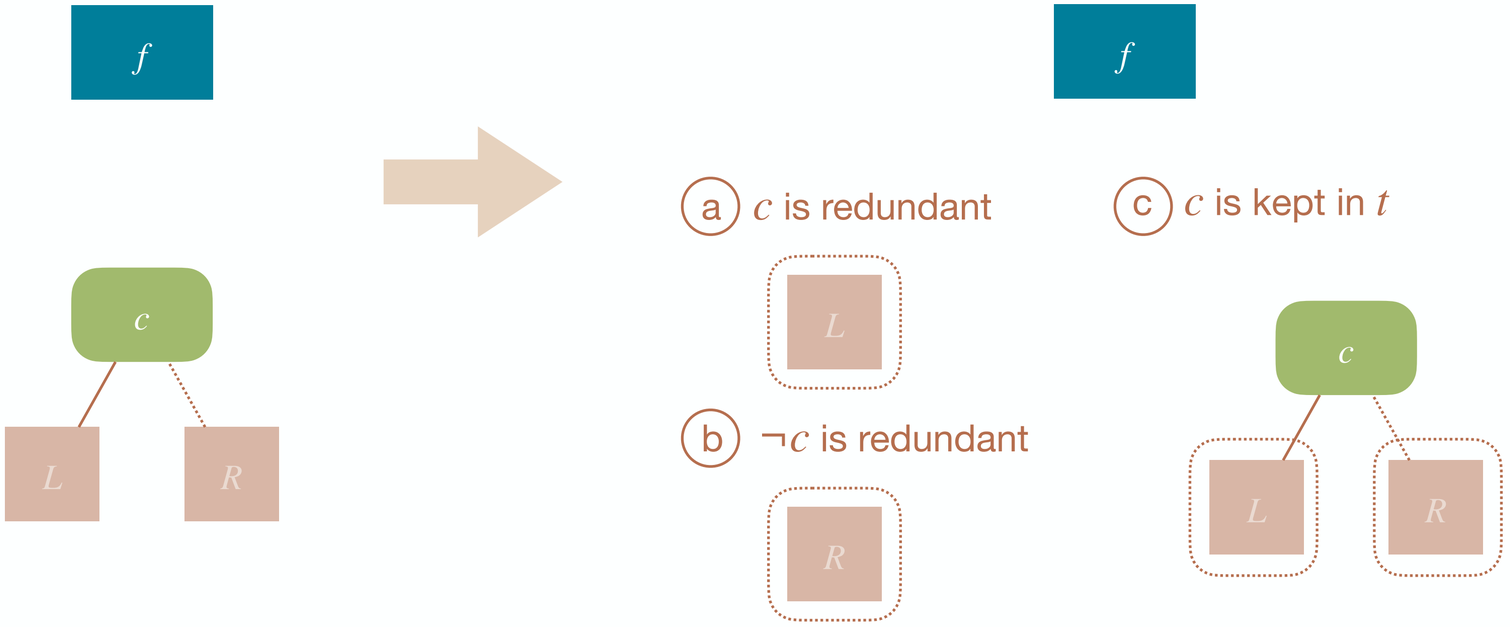
  - assignment $\overleftarrow{\mathsf{ASSIGN}}_A[\![X \leftarrow e]\!]$
  - test $\mathsf{FILTER}_A[\![e]\!]$

Piecewise-Defined Functions Domain

Tree Pruning

Goal: **add** a set J of **linear constraints** to the decision tree

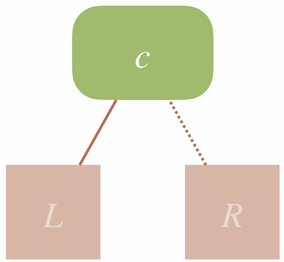
- Base case ($J = \emptyset$)

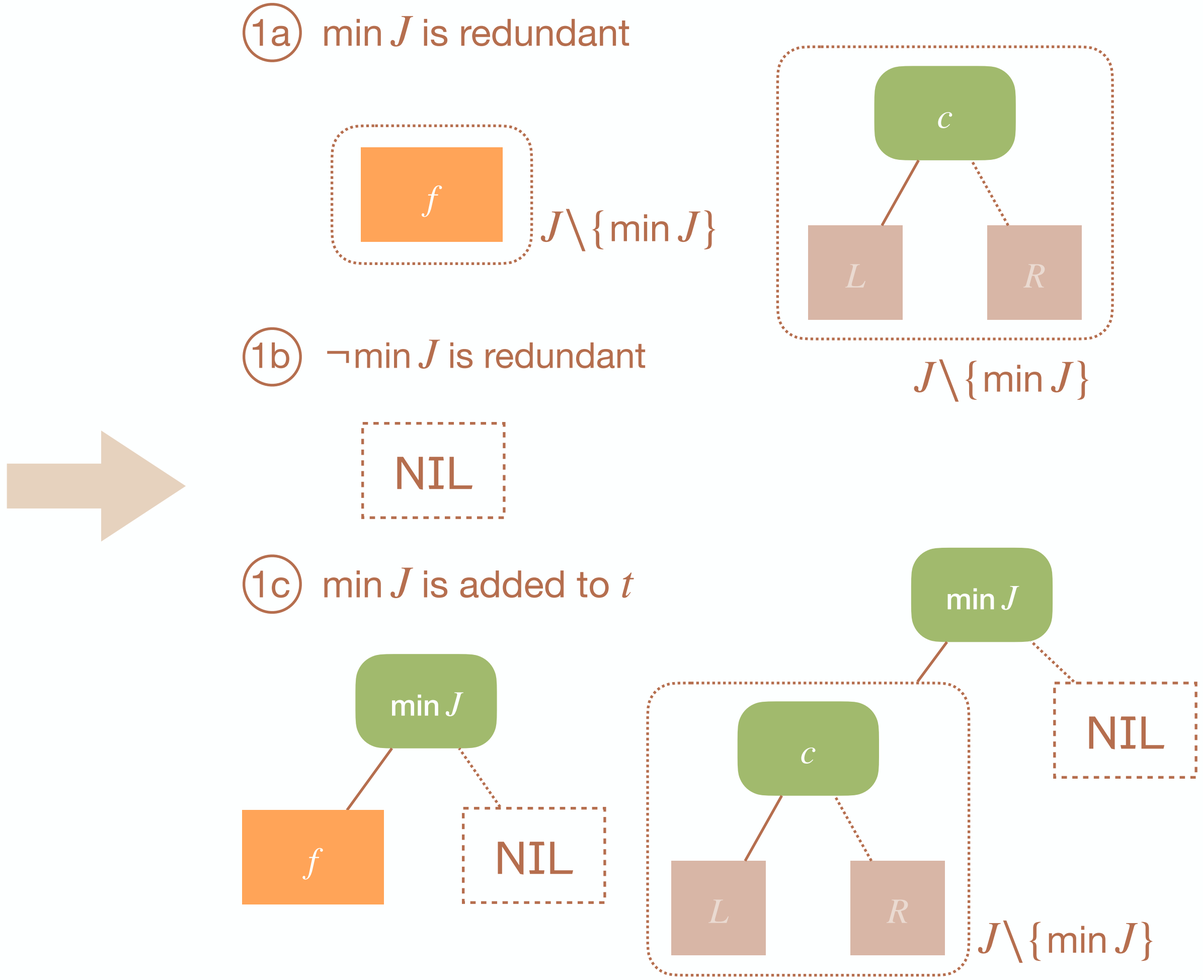


Piecewise-Defined Functions Domain

Tree Pruning (continue)

- Case ①

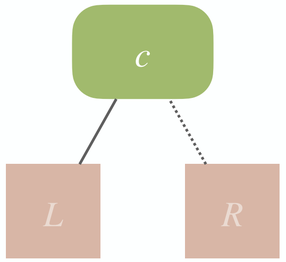




Piecewise-Defined Functions Domain

Tree Pruning (continue)

- Case ②

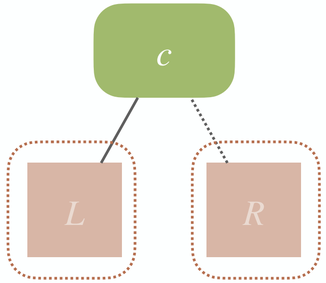


$c \leq_C \min J$

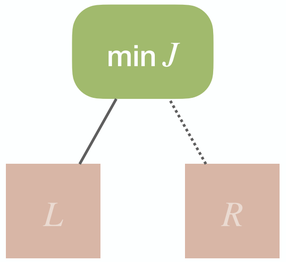
2a c is redundant

2b $\neg c$ is redundant

2c c is kept in t



- Case ③



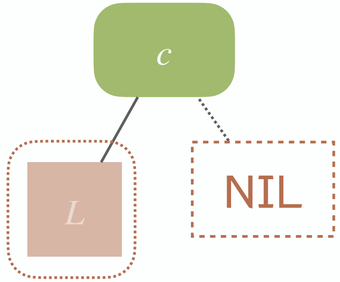
3a $\min J$ is redundant

$J \setminus \{\min J\}$

3b $\neg \min J$ is redundant

NIL

3c $\min J$ is kept in t

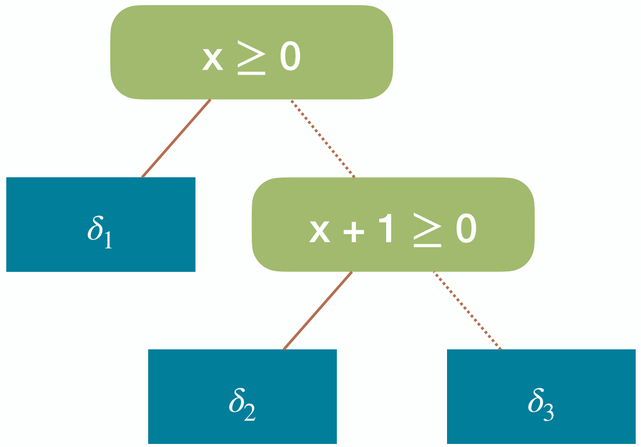


$J \setminus \{\min J\}$

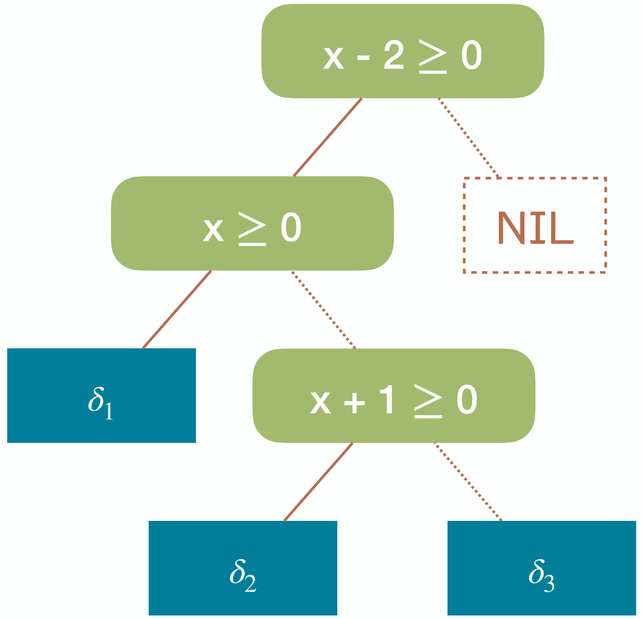
Piecewise-Defined Functions Domain

Tree Pruning (continue)

Example



$$J \stackrel{\text{def}}{=} \{x - 2 \geq 0\}$$



Piecewise-Defined Functions Domain

Abstract Domain Operators

- They manipulate elements in $\mathscr{A}_{\mathsf{NIL}} \stackrel{\text{def}}{=} \{\mathsf{NIL}\} \cup \mathscr{A}$

- The **binary operators** rely on a tree unification algorithm

  - approximation order $\preccurlyeq_A$ and computational order $\sqsubseteq_A$
  - approximation join $\curlyvee_A$ and computational join $\sqcup_A$
  - meet $\curlywedge_A$
  - widening $\triangledown_A$

- The **unary operators** rely on a tree pruning algorithm

  - **assignment** $\overleftarrow{\mathsf{ASSIGN}}_A[\![X \leftarrow e]\!]$
  - test $\mathsf{FILTER}_A[\![e]\!]$

Piecewise-Defined Functions Domain

Assignments

$\overleftarrow{\text{ASSIGN}}_A[\![X \leftarrow e]\!]$

- Base case (f)

  Apply $\overleftarrow{\text{ASSIGN}}_F[\![X \leftarrow e]\!][\alpha_C(C)]$ on the defined leaf nodes

$$\overleftarrow{\text{ASSIGN}}_F[\![X \leftarrow e]\!][D](f) \stackrel{\text{def}}{=} \begin{cases} \bar{f} & \bar{f} \in \mathcal{F} \setminus \{ \bot_F , \top_F \} \\ \top_F & \text{otherwise} \end{cases} \qquad f \in \mathcal{F} \setminus \{ \bot_F , \top_F \}$$

$$\text{where } \bar{f}(\ldots, X_i, X, \ldots) \stackrel{\text{def}}{=} \max\{ f(\ldots, \rho(X_i), v, \ldots) + 1 \mid \rho \in \gamma_D(\overleftarrow{\text{ASSIGN}}_D[\![X \leftarrow e]\!]D) \wedge v \in E[\![e]\!]\rho \}$$

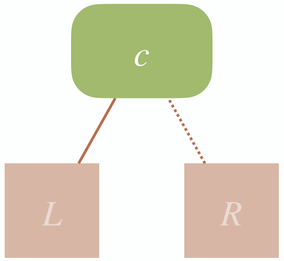
Example:

$\overleftarrow{\text{ASSIGN}}_F[\![x \leftarrow x + [1,2]]\!][\ \top_D\ ](\lambda x\,.\,x + 1) = \lambda x\,.\,x + 4$
(since $f(x + [1,2]) + 1 = x + [1,2] + 1 + 1 = x + [3,4]$ and $\max(3,4) = 4$

Piecewise-Defined Functions Domain

Assignments

$\overleftarrow{\text{ASSIGN}}_A[\![X \leftarrow e]\!]$

- 

Convert $\overleftarrow{\text{ASSIGN}}_D[\![X \leftarrow e]\!](\alpha_C(\{c\}))$ **and**
$\overleftarrow{\text{ASSIGN}}_D[\![X \leftarrow e]\!](\alpha_C(\{\neg c\}))$
into sets I **and** J **of linear constraints** *in canonical form*

case ① $I = J = \varnothing$

case ② $I = \varnothing \wedge \bot_C \in J$

case ③ $\bot_C \in I \wedge J = \varnothing$

case ④

1. perform **tree pruning** on  and 
2. join the results with $\curlyvee_A$

Piecewise-Defined Functions Domain

Abstract Domain Operators

- They manipulate elements in $\mathscr{A}_{\mathsf{NIL}} \stackrel{\text{def}}{=} \{\mathsf{NIL}\} \cup \mathscr{A}$

- The **binary operators** rely on a tree unification algorithm
  - approximation order $\preccurlyeq_A$ and computational order $\sqsubseteq_A$
  - approximation join $\curlyvee_A$ and computational join $\sqcup_A$
  - meet $\curlywedge_A$
  - widening $\triangledown_A$

- The **unary operators** rely on a tree pruning algorithm
  - assignment $\overleftarrow{\mathsf{ASSIGN}}_A[\![X \leftarrow e]\!]$
  - **test** $\mathsf{FILTER}_A[\![e]\!]$

Piecewise-Defined Functions Domain

Tests

$\mathsf{FILTER}_A[\![e]\!]$

1. Recursively descend the tree and apply STEP_F on the defined leaf nodes to account for one more execution step needed before termination:

   $$\mathsf{STEP}_F(f) \stackrel{\text{def}}{=} \lambda X_1, \ldots, X_k \,.\, f(X_1, \ldots, X_k) + 1 \qquad\qquad f \in \mathscr{F} \setminus \{\, \bot_F \,, \top_F \,\}$$

2. **Convert e into a set J of linear constraints** *in canonical form*

   Example: $\alpha_C(\mathsf{FILTER}_D[\![e]\!]\ \top_D\ )$
   where $\langle \mathscr{D}, \sqsubseteq_D \rangle$ is the underlying numerical domain

3. Perform **tree pruning** with J

Abstract Definite Termination Semantics

For each program instruction stmt, we define a transformer $\mathscr{R}^{\#}_{M}[\![\text{stmt}]\!] : \mathscr{A} \to \mathscr{A}$:

- $\mathscr{R}^{\#}_{M}[\![X \leftarrow e]\!]t \overset{\text{def}}{=} \overleftarrow{\text{ASSIGN}}_{A}[\![X \leftarrow e]\!]t$

Lemma (Soundness)

$$\mathscr{R}_{M}[\![X \leftarrow e]\!]\gamma_{A}(t) \preccurlyeq \gamma_{A}(\mathscr{R}^{\#}_{M}[\![X \leftarrow e]\!]t)$$

Abstract Definite Termination Semantics

For each program instruction stmt, we define a transformer $\mathscr{R}^{\#}_{M}[\![\text{stmt}]\!] : \mathscr{A} \to \mathscr{A}$:

- $\mathscr{R}^{\#}_{M}[\![X \leftarrow e]\!]t \overset{\text{def}}{=} \overleftarrow{\text{ASSIGN}}_{A}[\![X \leftarrow e]\!]t$
- $\mathscr{R}^{\#}_{M}[\![\textbf{if}\ \ e \bowtie 0\ \textbf{then}\ s\ \textbf{end}]\!]t \overset{\text{def}}{=} \text{FILTER}_{A}[\![e \bowtie 0]\!](\mathscr{R}^{\#}_{M}[\![s_1]\!]t) \curlyvee_{T} \text{FILTER}_{A}[\![e \not\bowtie 0]\!](t)$

Lemma (Soundness)

$$\mathscr{R}_{M}[\![\textbf{if}\ .\ e \bowtie 0\ \textbf{then}\ s\ \textbf{end}]\!]\gamma_{A}(t) \preccurlyeq \gamma_{A}(\mathscr{R}^{\#}_{M}[\![\textbf{if}\ .\ e \bowtie 0\ \textbf{then}\ s\ \textbf{end}]\!]t)$$

Abstract Definite Termination Semantics

For each program instruction stmt, we define a transformer $\mathscr{R}^{\#}_{M}[\![\text{stmt}]\!] : \mathscr{A} \to \mathscr{A}$:

- $\mathscr{R}^{\#}_{M}[\![X \leftarrow e]\!]t \overset{\text{def}}{=} \overleftarrow{\text{ASSIGN}}_{A}[\![X \leftarrow e]\!]t$
- $\mathscr{R}^{\#}_{M}[\![\textbf{if}\ \ e \bowtie 0\ \textbf{then}\ s\ \textbf{end}]\!]t \overset{\text{def}}{=} \text{FILTER}_{A}[\![e \bowtie 0]\!](\mathscr{R}^{\#}_{M}[\![s_1]\!]t) \curlyvee_{T} \text{FILTER}_{A}[\![e \bowtie 0]\!](t)$
- $\mathscr{R}^{\#}_{M}[\![\textbf{while}\ \ e \bowtie 0\ \textbf{do}\ s\ \textbf{done}]\!]t \overset{\text{def}}{=} \text{lfp}^{\#}\overline{F}^{\#}_{M}$

  where $\overline{F}^{\#}_{M}(x) \overset{\text{def}}{=} \text{FILTER}_{A}[\![e \bowtie 0]\!](\mathscr{R}^{\#}_{M}[\![s]\!]x) \curlyvee_{T} \text{FILTER}_{A}[\![e \bowtie 0]\!](t)$

Lemma (Soundness)

$\mathscr{R}_{M}[\![\textbf{while}\ \ e \bowtie 0\ \textbf{do}\ s\ \textbf{done}]\!]\gamma_{A}(t) \preccurlyeq \gamma_{A}(\mathscr{R}^{\#}_{M}[\![\textbf{while}\ \ e \bowtie 0\ \textbf{do}\ s\ \textbf{done}]\!]t)$

Abstract Definite Termination Semantics

For each program instruction stmt, we define a transformer $\mathscr{R}^{\#}_{M}[\![\text{stmt}]\!] : \mathscr{A} \to \mathscr{A}$:

- $\mathscr{R}^{\#}_{M}[\![X \leftarrow e]\!]t \overset{\text{def}}{=} \overleftarrow{\text{ASSIGN}}_{A}[\![X \leftarrow e]\!]t$
- $\mathscr{R}^{\#}_{M}[\![\textbf{if}\ \ e \bowtie 0\ \textbf{then}\ s\ \textbf{end}]\!]t \overset{\text{def}}{=} \text{FILTER}_{A}[\![e \bowtie 0]\!](\mathscr{R}^{\#}_{M}[\![s_1]\!]t) \curlyvee_{T} \text{FILTER}_{A}[\![e \bowtie 0]\!](t)$
- $\mathscr{R}^{\#}_{M}[\![\textbf{while}\ \ e \bowtie 0\ \textbf{do}\ s\ \textbf{done}]\!]t \overset{\text{def}}{=} \text{lfp}^{\#}\overline{F}^{\#}_{M}$

  where $\overline{F}^{\#}_{M}(x) \overset{\text{def}}{=} \text{FILTER}_{A}[\![e \bowtie 0]\!](\mathscr{R}^{\#}_{M}[\![s]\!]x) \curlyvee_{T} \text{FILTER}_{A}[\![e \bowtie 0]\!](t)$
- $\mathscr{R}^{\#}_{M}[\![s_1; s_2]\!]t \overset{\text{def}}{=} \mathscr{R}^{\#}_{M}[\![s_1]\!](\mathscr{R}^{\#}_{M}[\![s_2]\!]t)$

Lemma (Soundness)

$\mathscr{R}_{M}[\![s_1; s_2]\!]\gamma_{A}(t) \preccurlyeq \gamma_{A}(\mathscr{R}^{\#}_{M}[\![s_1; s_2]\!]t)$

Abstract Definite Termination Semantics

Definition

The **abstract definite termination semantics** $\mathscr{R}^{\#}_{M}[\![\mathsf{s}^{\ell}]\!] \in \mathscr{A}$ of a program s^{ℓ} is:

$$\mathscr{R}^{\#}_{M}[\![\mathsf{s}^{\ell}]\!] \stackrel{\text{def}}{=} \mathscr{R}^{\#}_{M}[\![\mathsf{s}]\!](\mathsf{LEAF}: \lambda X_1, \ldots, X_k.0)$$

where $\mathscr{R}^{\#}_{M}[\![\mathsf{s}]\!] : \mathscr{A} \to \mathscr{A}$ is the abstract definite termination semantics of each program instruction s

Theorem (Soundness)

$$\mathscr{R}_{M}[\![\mathsf{s}^{\ell}]\!] \preccurlyeq \gamma_A(\mathscr{R}^{\#}_{M}[\![\mathsf{s}^{\ell}]\!])$$

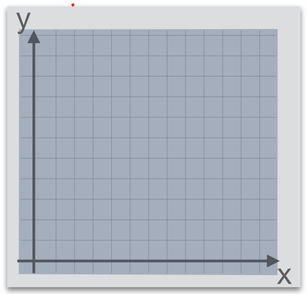
Corollary (Soundness)

A program s^{ℓ} **must terminate** starting from a set of initial states I if $I \subseteq \mathsf{dom}(\gamma_A(\mathscr{R}^{\#}_{M}[\![\mathsf{s}^{\ell}]\!]))$

Abstract Definite Termination Semantics

Example

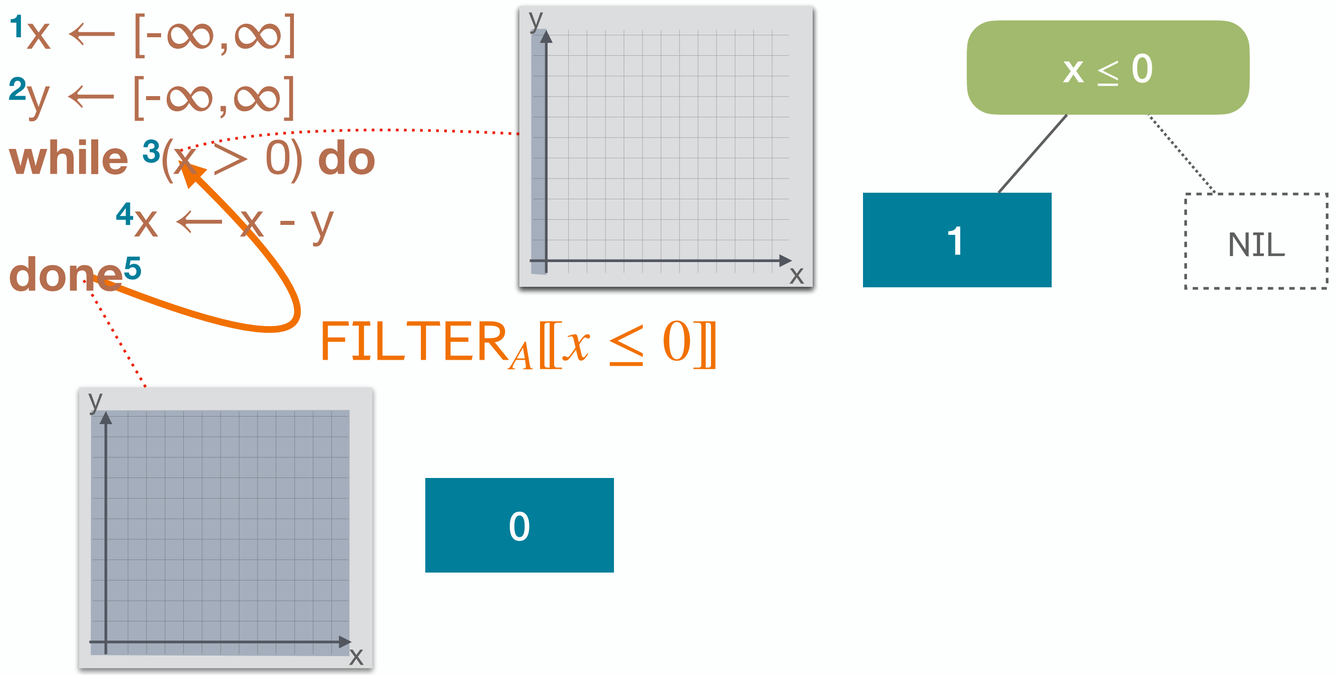
```
1x ← [-∞,∞]
2y ← [-∞,∞]
while 3(x > 0) do
        4x ← x - y
done5
```



0

Abstract Definite Termination Semantics

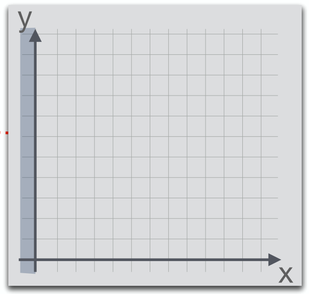
Example

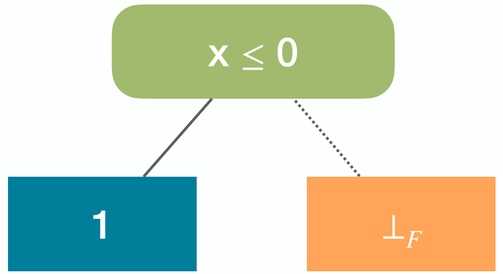


Abstract Definite Termination Semantics

Example

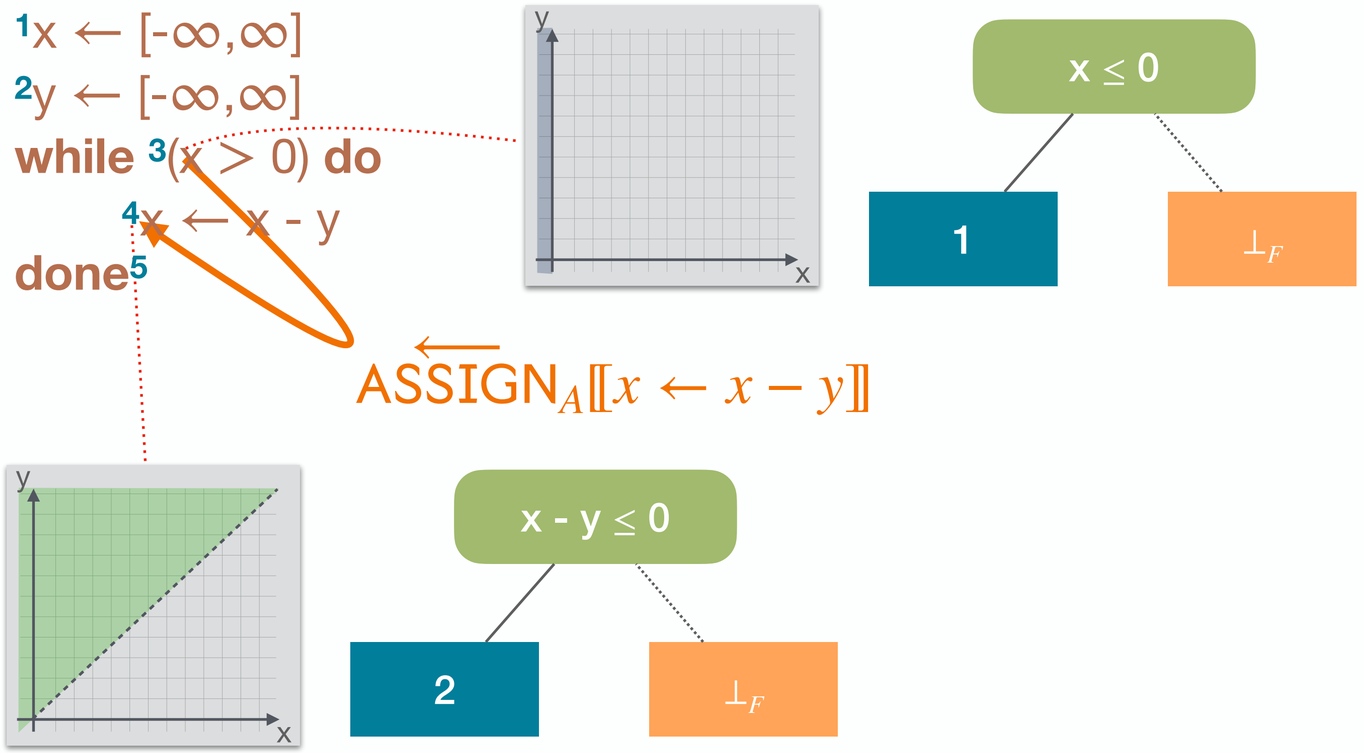
```
1x ← [-∞,∞]
2y ← [-∞,∞]
while 3(x > 0) do
      4x ← x - y
done5
```





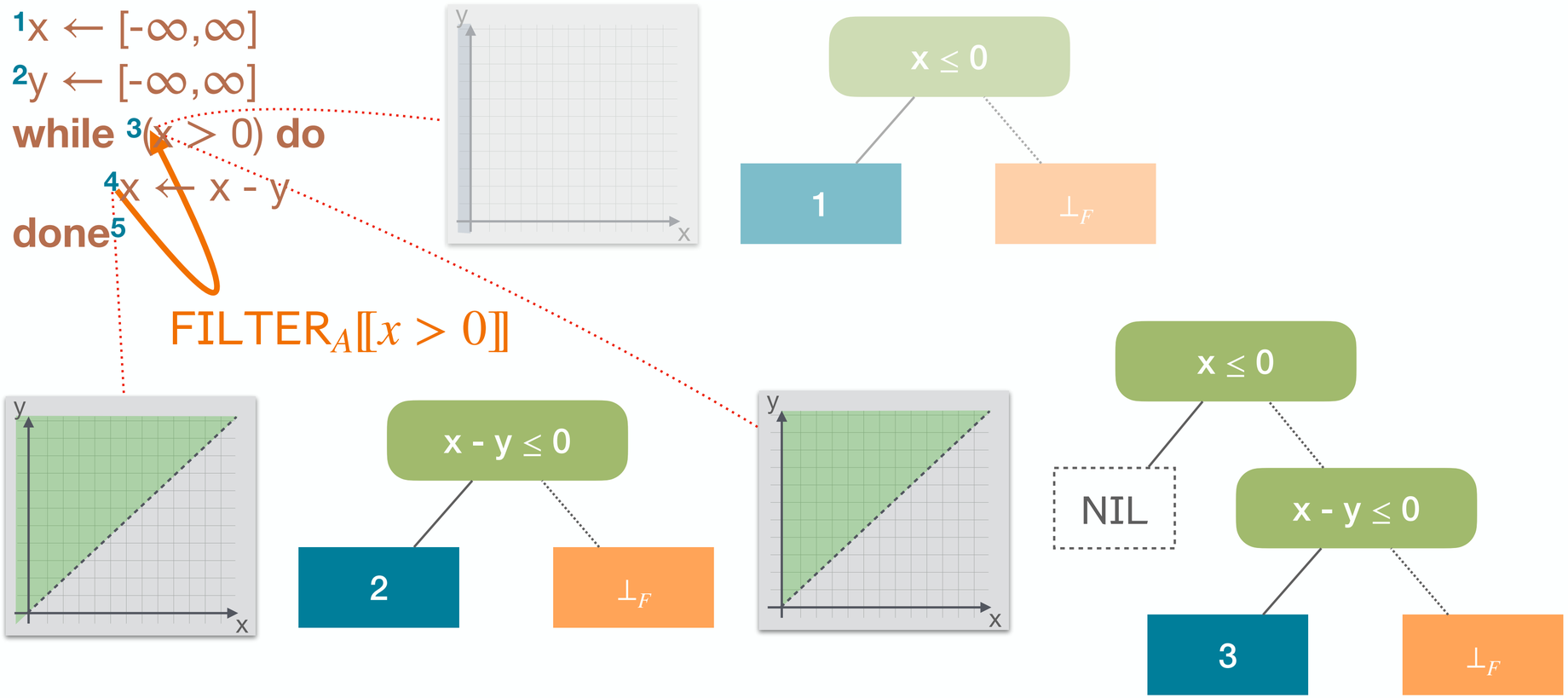
Abstract Definite Termination Semantics

Example



Abstract Definite Termination Semantics

Example

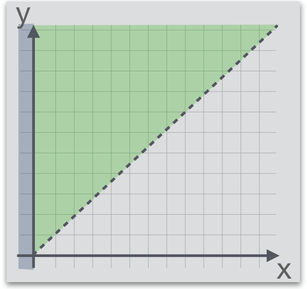


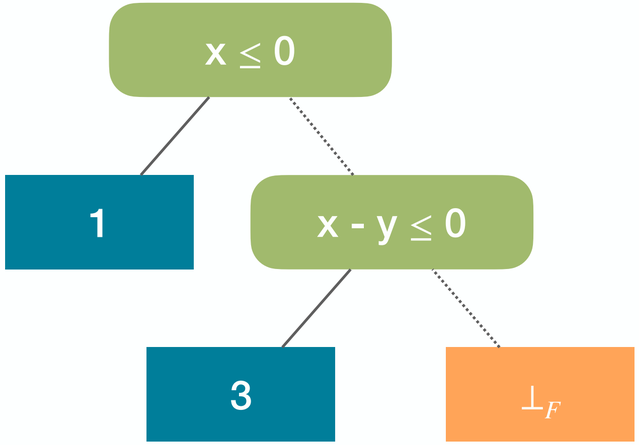
Abstract Definite Termination Semantics

Example

```
1x ← [-∞,∞]
2y ← [-∞,∞]
while 3(x > 0) do
      4x ← x - y
done5
```

γ_A

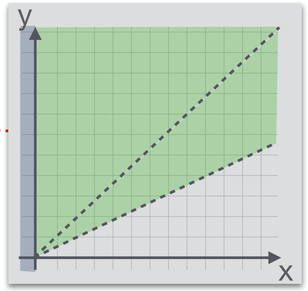


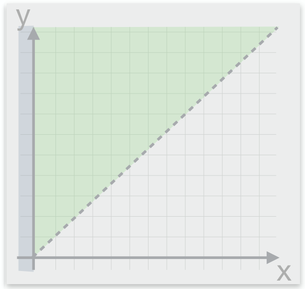


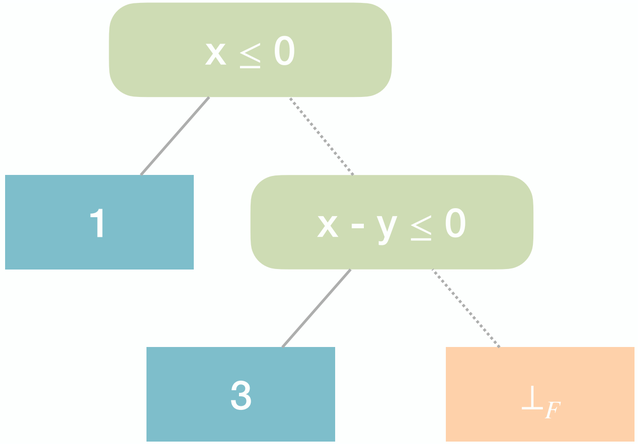
Abstract Definite Termination Semantics

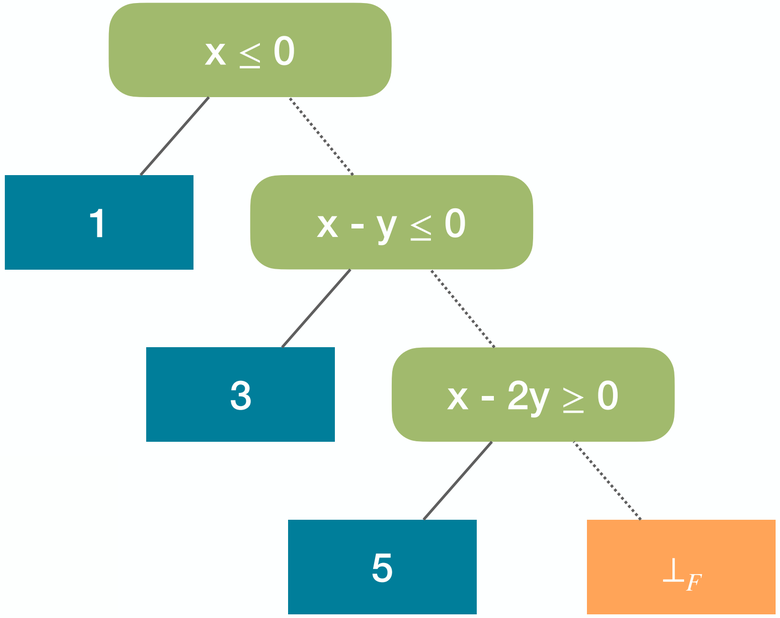
Example

$^1x \leftarrow [-\infty,\infty]$
$^2y \leftarrow [-\infty,\infty]$
while $^3(x > 0)$ **do**
$^4x \leftarrow x - y$
done^5



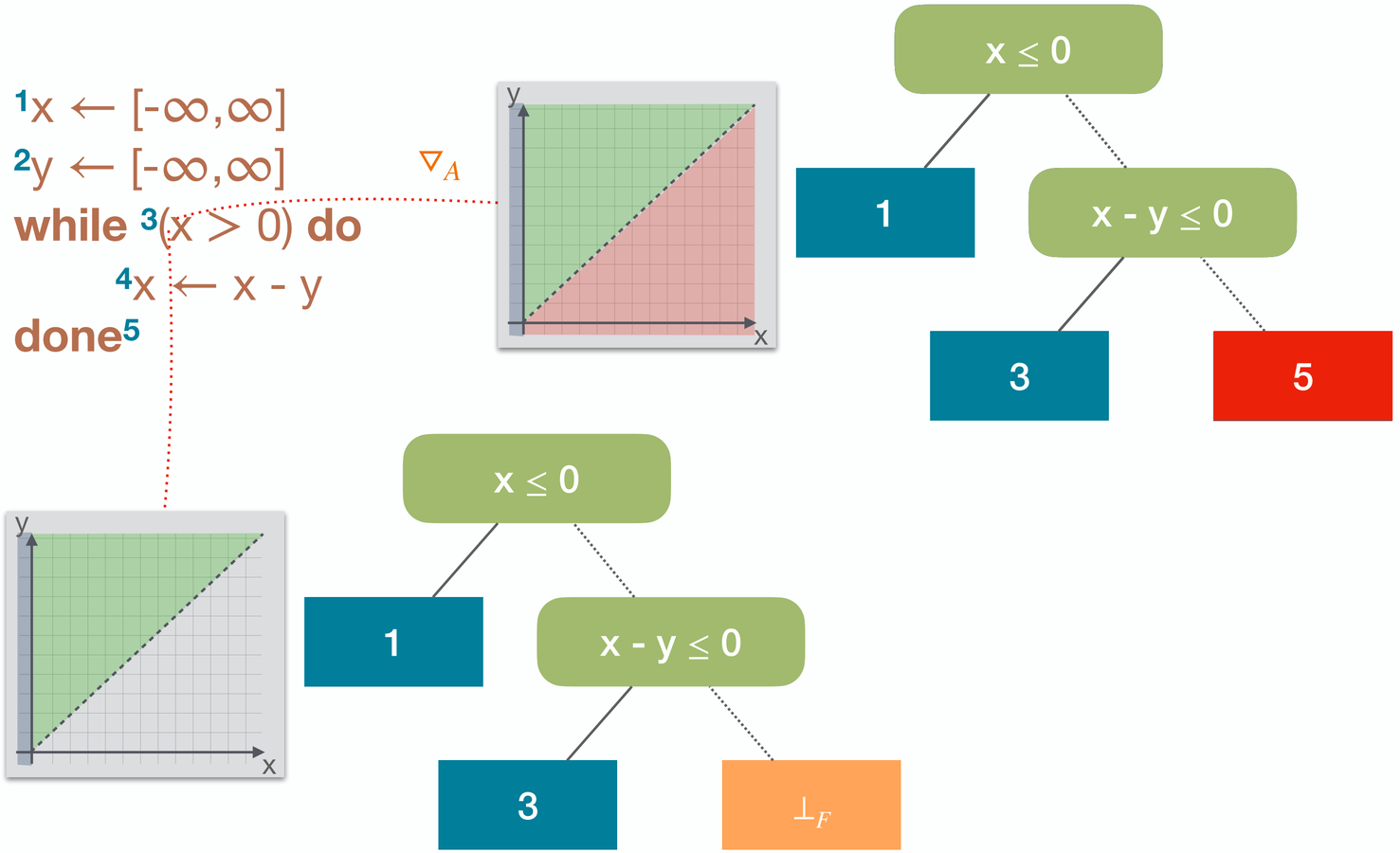






Abstract Definite Termination Semantics

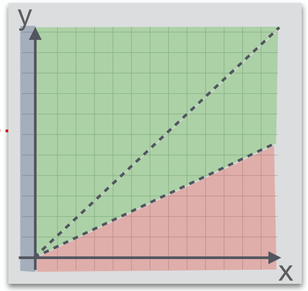
Example

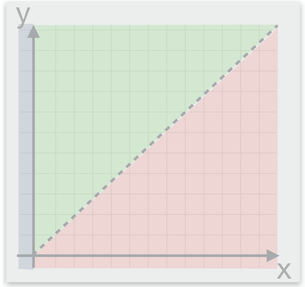


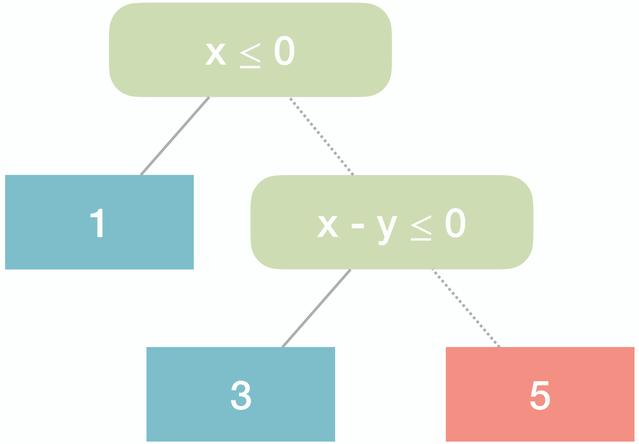
Abstract Definite Termination Semantics

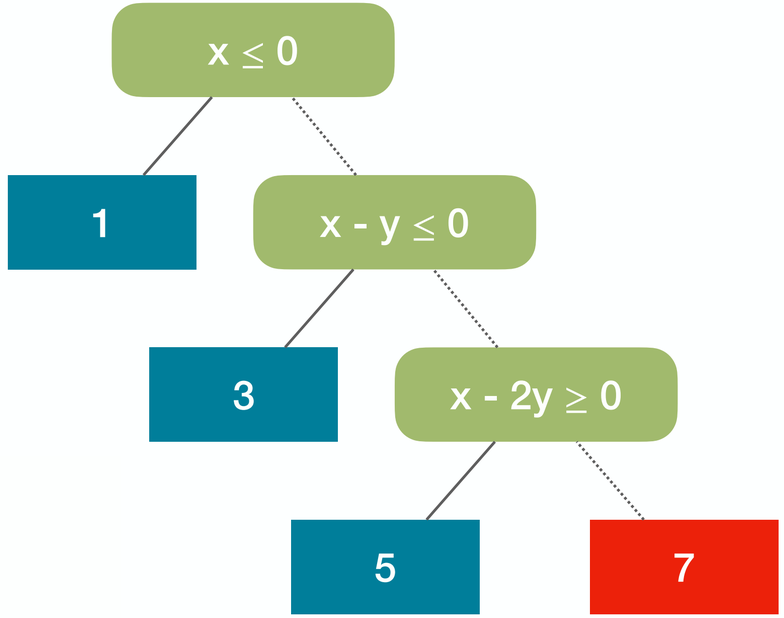
Example

$^1x \leftarrow [-\infty,\infty]$
$^2y \leftarrow [-\infty,\infty]$
while $^3(x > 0)$ **do**
    $^4x \leftarrow x - y$
done^5



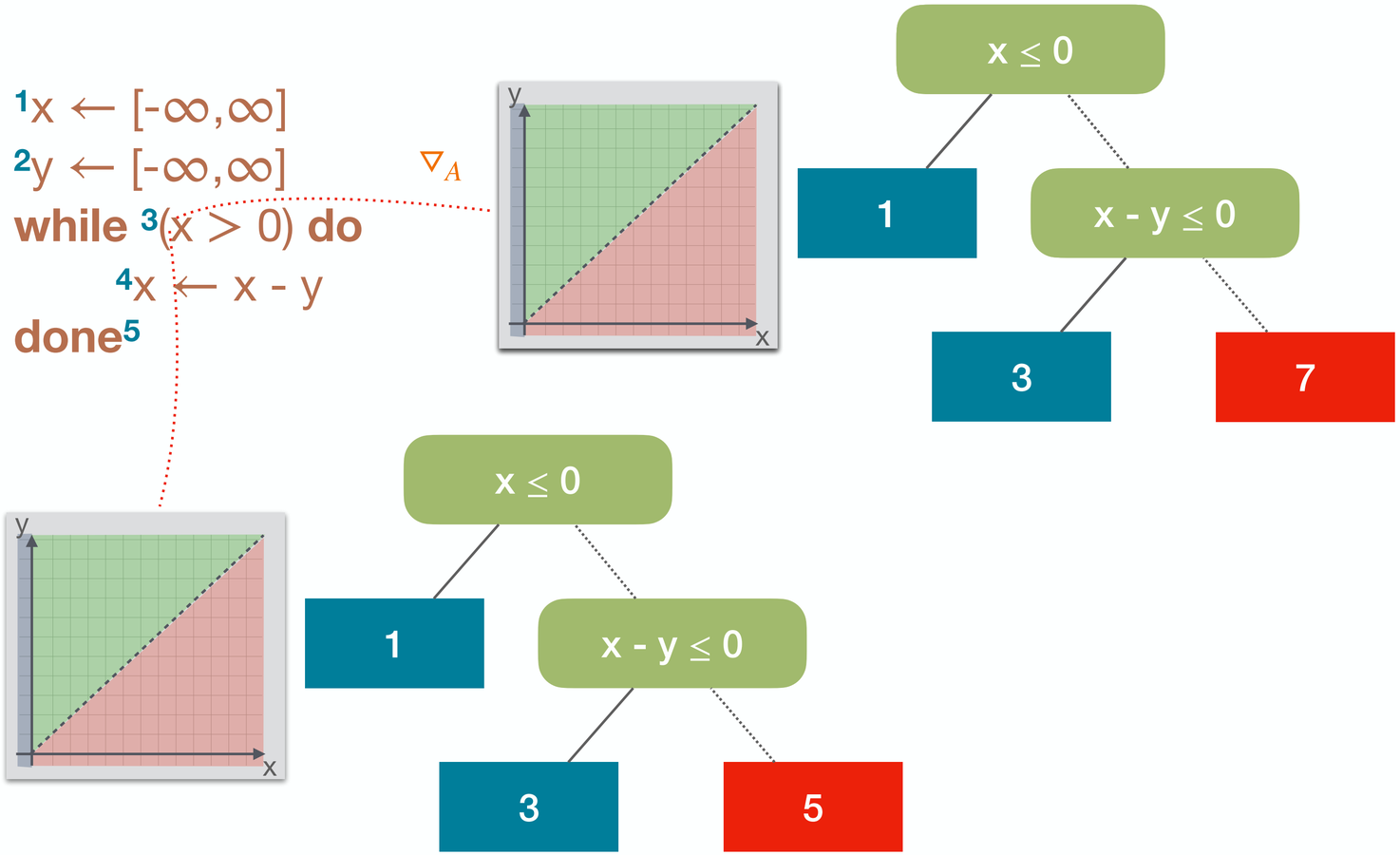






Abstract Definite Termination Semantics

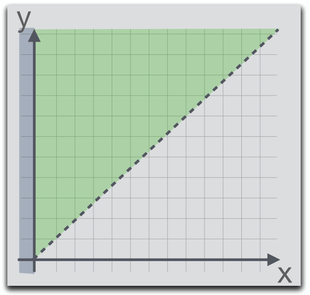
Example

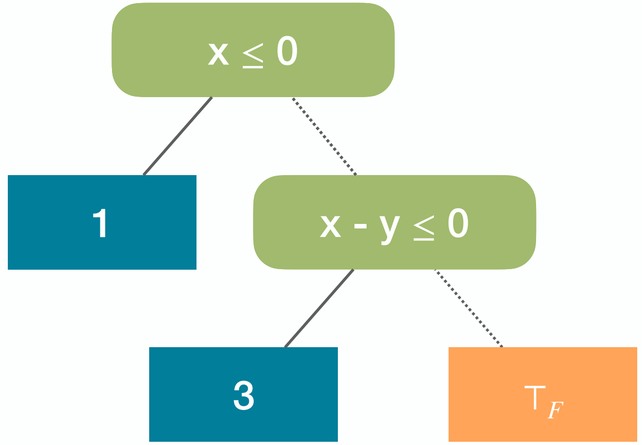


Abstract Definite Termination Semantics

Example

```
1x ← [-∞,∞]
2y ← [-∞,∞]
while 3(x > 0) do
      4x ← x - y
done5
```

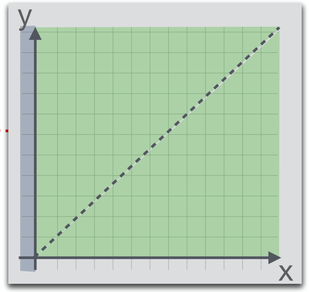




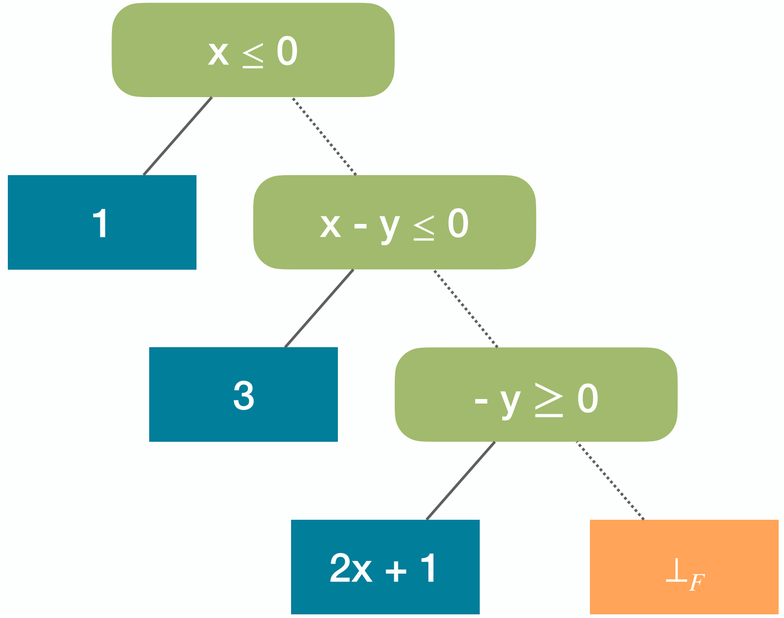
Abstract Definite Termination Semantics

Example

$^{1}x \leftarrow [-\infty,\infty]$
$^{2}y \leftarrow [-\infty,\infty]$
while $^{3}(x > 0)$ **do**
    $^{4}x \leftarrow x - y$
done^{5}



Better Widening

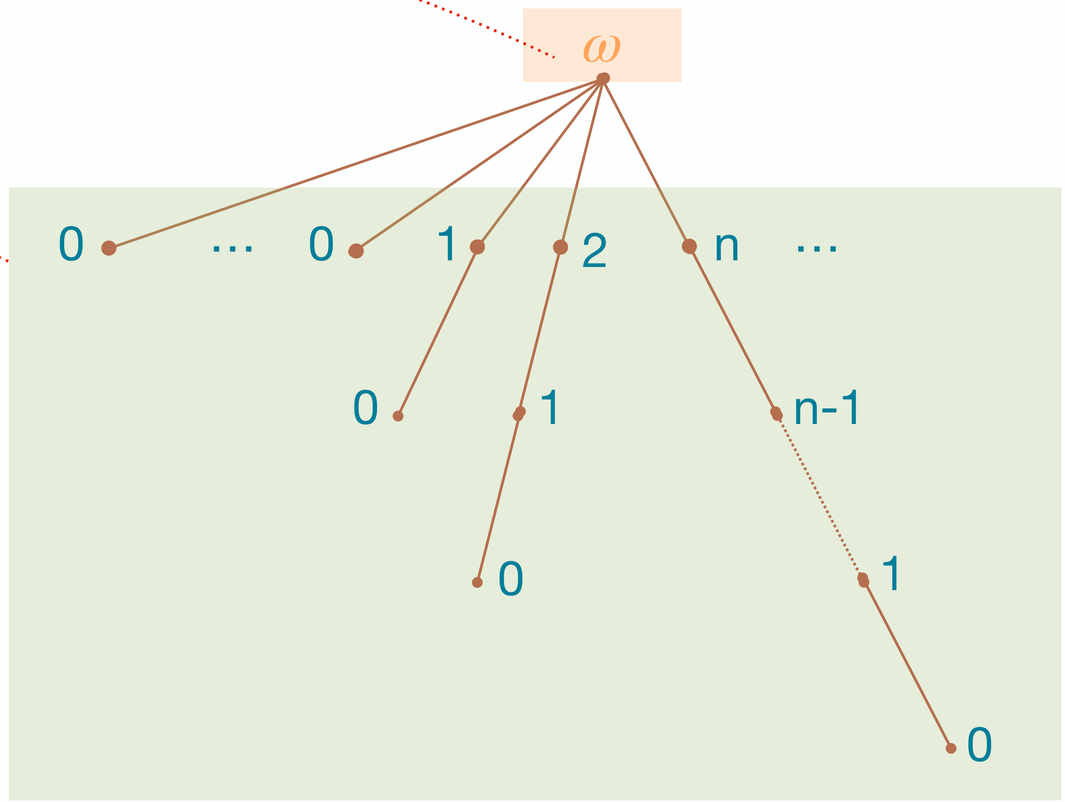


Ordinal-Valued Ranking Functions

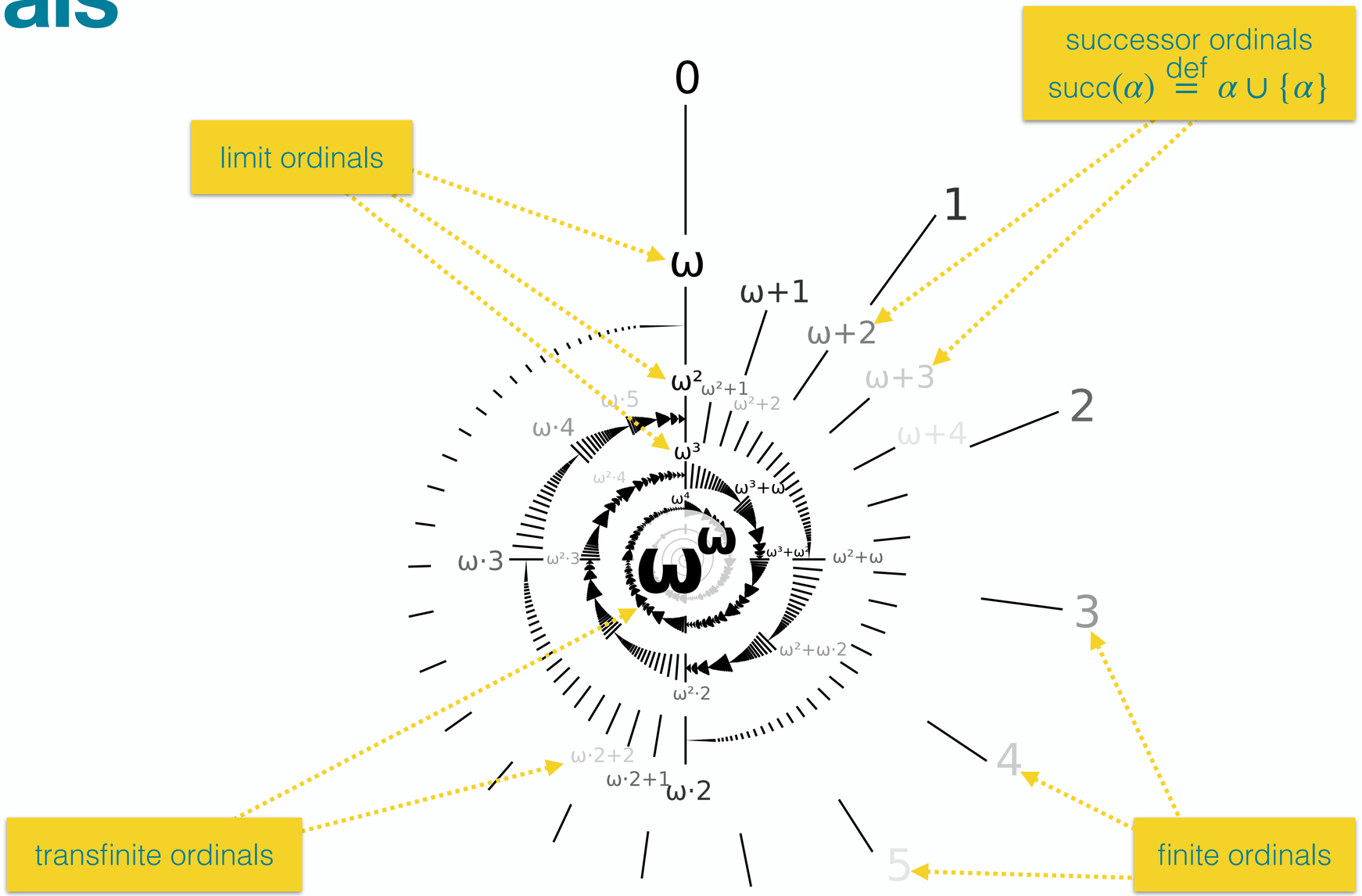
Need for Ordinals

Example

```
1x ← [-∞, +∞]
while 2(x > 0) do
      3x ← x - 1
done4
```



Ordinals


successor ordinals
succ(α) def= α ∪ {α}
limit ordinals
transfinite ordinals
finite ordinals
0
1
2
3
4
5
ω
ω+1
ω+2
ω+3
ω+4
ω·2
ω·2+1
ω·2+2
ω·3
ω·4
ω·5
ω²
ω²+1
ω²+2
ω²+ω
ω²+ω·2
ω²·2
ω²·3
ω²·4
ω³
ω³+ω
ω³+ω²
ω⁴
ωω

Ordinal Arithmetic

Addition

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

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

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

Properties

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

Ordinal Arithmetic

Multiplication

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

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

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

Properties

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

Piecewise-Defined Function Domain

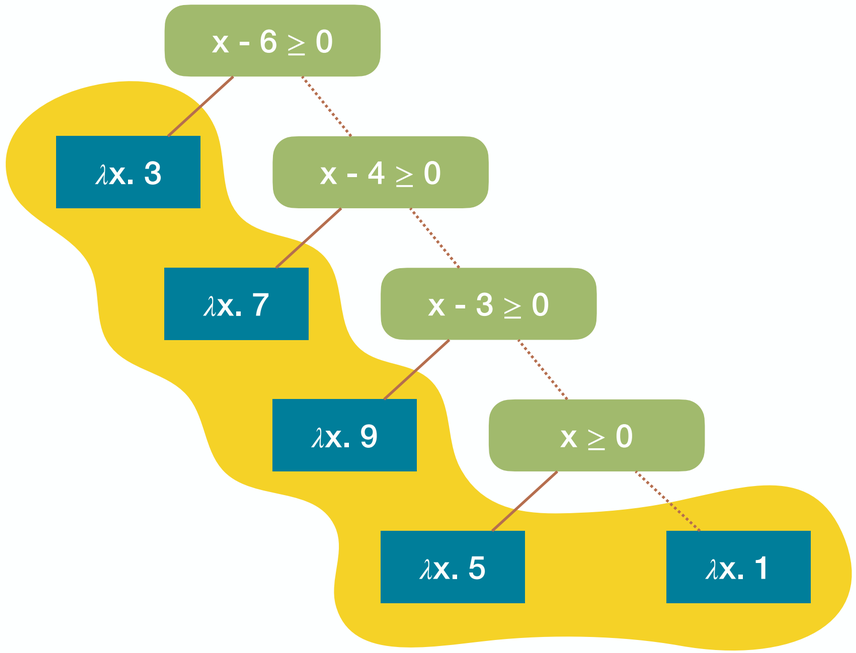
Functions Auxiliary Abstract Domain

- Parameterized by an ***underlying numerical abstract domain*** $\langle \mathscr{D}, \sqsubseteq_D \rangle$

- $\mathscr{W} \stackrel{\text{def}}{=} \{ \bot_W \} \cup \{ \sum_i \omega^i \cdot f_i \mid f_i \in \mathscr{F} \backslash \{ \bot_F, \top_F \} \} \cup \{ \top_W \}$

Cantor Normal Form
$\omega^{\beta_1} \cdot n_1 + \ldots + \omega^{\beta_k} \cdot n_k$

- $\mathscr{F} \stackrel{\text{def}}{=} \{ \bot_F \} \cup (\mathbb{Z}^{\mathbb{V}} \to \mathbb{N}) \cup \{ \top_F \}$



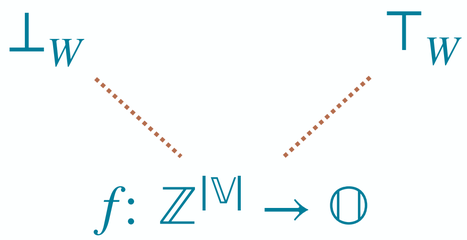
Piecewise-Defined Function Domain

Functions Auxiliary Abstract Domain (continue)

- **approximation order** $\preccurlyeq_F [D]$, where $D \in \mathscr{D}$:
  - between <u>defined</u> leaf nodes:

$$\sum_i \omega^i \cdot f_{i_1} \preccurlyeq_W [D] \sum_i \omega^i \cdot f_{i_2} \stackrel{\text{def}}{=} \forall \rho \in \gamma_D(D) : \sum_i \omega^i \cdot f_{i_1}(\,..\rho(X_i)\,..\,) \leq \sum_i \omega^i \cdot f_{i_2}(\,..\rho(X_i)\,..\,)$$

  - otherwise (i.e., when one or both leaf nodes are <u>undefined</u>):



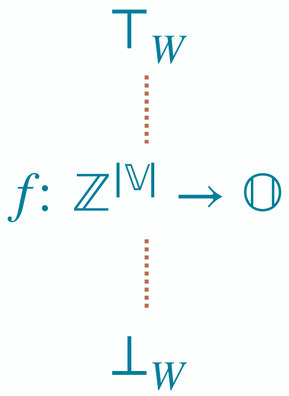
Piecewise-Defined Function Domain

Functions Auxiliary Abstract Domain (continue)

- **computational order** $\sqsubseteq_F [D]$, where $D \in \mathscr{D}$:
  - between defined leaf nodes:

$$\sum_i \omega^i \cdot f_{i_1} \sqsubseteq_W [D] \sum_i \omega^i \cdot f_{i_2} \overset{\text{def}}{=} \forall \rho \in \gamma_D(D) : \sum_i \omega^i \cdot f_{i_1}(\,..\rho(X_i)\,..\,) \leq \sum_i \omega^i \cdot f_{i_2}(\,..\rho(X_i)\,..\,)$$

  - otherwise (i.e., when one or both leaf nodes are undefined):



Piecewise-Defined Functions Domain

$$\mathscr{A} \overset{\text{def}}{=} \{\text{LEAF}: f \mid f \in \mathscr{F}\} \cup \{\text{NODE}\{c\}: t_1; t_2 \mid c \in \mathscr{C} \wedge t_1, t_2 \in \mathscr{A}\}$$

- **concretization function** $\gamma_A : \mathscr{A} \to (\mathscr{E} \rightharpoonup \mathbb{O})$:

  $$\gamma_A(t) \overset{\text{def}}{=} \overline{\gamma}_A[\varnothing](t)$$

  where $\overline{\gamma}_A : \mathscr{P}(\mathscr{C} / \equiv_C) \to \mathscr{A} \to (\mathscr{E} \rightharpoonup \mathbb{O})$:

  $$\overline{\gamma}_A[C](\text{LEAF}: f) \overset{\text{def}}{=} \gamma_F[\alpha_C(C)](f)$$
  $$\overline{\gamma}_A[C](\text{NODE}\{c\}: t_1; t_2) \overset{\text{def}}{=} \overline{\gamma}_A[C \cup \{c\}](t_1) \dot{\cup} \overline{\gamma}_A[C \cup \{\neg c\}](t_2)$$

  and $\gamma_F : \mathscr{D} \to \mathscr{F} \to (\mathscr{E} \rightharpoonup \mathbb{O})$:

  $$\gamma_F[D](\bot_F) \overset{\text{def}}{=} \dot{\varnothing}$$
  $$\gamma_F[D]\left(\sum_i \omega^i \cdot f_i\right) \overset{\text{def}}{=} \lambda\rho \in \gamma_D(D): \sum_i \omega^i \cdot f_i(\ldots, \rho(X_i), \ldots)$$
  $$\gamma_F[D](\top_F) \overset{\text{def}}{=} \dot{\varnothing}$$

Piecewise-Defined Functions Domain

Abstract Domain Operators

- They manipulate elements in $\mathscr{A}_{\mathsf{NIL}} \stackrel{\text{def}}{=} \{\mathsf{NIL}\} \cup \mathscr{A}$

- The **binary operators** rely on a tree unification algorithm
  - approximation order $\preccurlyeq_A$ and computational order $\sqsubseteq_A$
  - approximation join $\curlyvee_A$ and computational join $\sqcup_A$
  - meet $\curlywedge_A$
  - widening $\triangledown_A$

- The **unary operators** rely on a tree pruning algorithm
  - assignment $\overleftarrow{\mathsf{ASSIGN}}_A[\![X \leftarrow e]\!]$
  - test $\mathsf{FILTER}_A[\![e]\!]$

Piecewise-Defined Functions Domain

Abstract Domain Operators

- They manipulate elements in $\mathscr{A}_{\mathsf{NIL}} \stackrel{\text{def}}{=} \{\mathsf{NIL}\} \cup \mathscr{A}$

- The **binary operators** rely on a tree unification algorithm
  - approximation order $\preccurlyeq_A$ and computational order $\sqsubseteq_A$
  - **approximation join $\curlyvee_A$ and computational join $\sqcup_A$**
  - meet $\curlywedge_A$
  - widening $\triangledown_A$

- The **unary operators** rely on a tree pruning algorithm
  - assignment $\overleftarrow{\mathsf{ASSIGN}}_A[\![X \leftarrow e]\!]$
  - test $\mathsf{FILTER}_A[\![e]\!]$

Piecewise-Defined Functions Domain

Approximation Join

- **approximation join** $\curlyvee_F [D]$, where $D \in \mathscr{D}$:
  - between <u>defined</u> leaf nodes:

    approximation join $\curlyvee_F [D]$ in ascending powers of ω

    Example:

$$
\begin{array}{rcccccc}
f_1 & \equiv & \omega^2 \cdot x_1 & + & \omega \cdot x_2 & + & 3 \\
f_2 & \equiv & \omega^2 \cdot x_1 & + & \omega \cdot (-x_2) & + & 4 \\
f_1 \curlyvee_W [\ \top_D\ ] f_2 & \equiv & \omega^2 \cdot (x_1 + 1) & + & \omega \cdot 0 & + & 4
\end{array}
$$

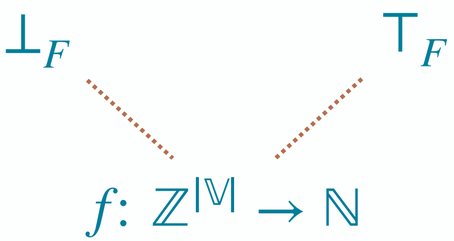
Piecewise-Defined Functions Domain

Approximation Join

- **approximation join** $\curlyvee_F [D]$, where $D \in \mathscr{D}$:
  - between <u>defined</u> leaf nodes:

    approximation join $\curlyvee_F [D]$ in ascending powers of ω
  - otherwise (i.e., when one or both leaf nodes are <u>undefined</u>):

$$\begin{aligned}
\bot_W \curlyvee_W [D]\ f &\stackrel{\text{def}}{=} \bot_W && f \in \mathscr{W} \setminus \{\ \top_W\ \} \\
f \curlyvee_W [D]\ \bot_W &\stackrel{\text{def}}{=} \bot_W && f \in \mathscr{W} \setminus \{\ \top_W\ \} \\
\top_W \curlyvee_W [D]\ f &\stackrel{\text{def}}{=} \top_W && f \in \mathscr{W} \setminus \{\ \bot_W\ \} \\
f \curlyvee_W [D]\ \top_W &\stackrel{\text{def}}{=} \top_W && f \in \mathscr{W} \setminus \{\ \bot_W\ \}
\end{aligned}$$



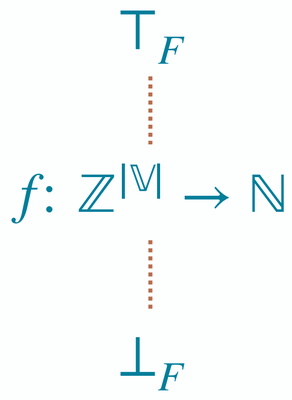
Piecewise-Defined Functions Domain

Computational Join

- **computational join** $\sqcup_F [D]$, where $D \in \mathscr{D}$:
  - between <u>defined</u> leaf nodes:

    computational join $\sqcup_W [D]$ in ascending powers of ω
  - otherwise (i.e., when one or both leaf nodes are <u>undefined</u>):

$$\begin{aligned}
\bot_W \sqcup_W [D]\ f &\overset{\text{def}}{=} f && f \in \mathscr{W} \\
f \sqcup_W [D]\ \bot_W &\overset{\text{def}}{=} f && f \in \mathscr{W} \\
\top_W \sqcup_W [D]\ f &\overset{\text{def}}{=} \top_W && f \in \mathscr{W} \\
f \sqcup_W [D]\ \top_W &\overset{\text{def}}{=} \top_W && f \in \mathscr{W}
\end{aligned}$$



Piecewise-Defined Functions Domain

Abstract Domain Operators

- They manipulate elements in $\mathscr{A}_{\mathsf{NIL}} \stackrel{\text{def}}{=} \{\mathsf{NIL}\} \cup \mathscr{A}$

- The **binary operators** rely on a tree unification algorithm
  - approximation order $\preccurlyeq_A$ and computational order $\sqsubseteq_A$
  - approximation join $\curlyvee_A$ and computational join $\sqcup_A$
  - meet $\curlywedge_A$
  - **widening** ∇_A

- The **unary operators** rely on a tree pruning algorithm
  - assignment $\overleftarrow{\mathsf{ASSIGN}}_A[\![X \leftarrow e]\!]$
  - test $\mathsf{FILTER}_A[\![e]\!]$

Piecewise-Defined Functions Domain

Widening

Value Widening

1. Recursively descend the trees while *accumulating the linear constraints encountered along the paths* into a set of constraints C

2. Widen each (defined) leaf node f with respect to each of their adjacent (defined) leaf node $\bar{f}$ using the **extrapolation operator** $\blacktriangledown_F\,[\alpha_C(\overline{C}), \alpha_C(C)]$, where $\overline{C}$ is the set of constraints along the path to $\bar{f}$
   in ascending powers of ω

yield T_W when the extrapolation of natural-valued functions yields T_F

Piecewise-Defined Functions Domain

Abstract Domain Operators

- They manipulate elements in $\mathscr{A}_{\mathsf{NIL}} \stackrel{\text{def}}{=} \{\mathsf{NIL}\} \cup \mathscr{A}$

- The **binary operators** rely on a tree unification algorithm
  - approximation order $\preccurlyeq_A$ and computational order $\sqsubseteq_A$
  - approximation join $\curlyvee_A$ and computational join $\sqcup_A$
  - meet $\curlywedge_A$
  - widening $\triangledown_A$

- The **unary operators** rely on a tree pruning algorithm
  - **assignment** $\overleftarrow{\mathsf{ASSIGN}}_A[\![X \leftarrow e]\!]$
  - test $\mathsf{FILTER}_A[\![e]\!]$

Piecewise-Defined Functions Domain

Assignments

$\overleftarrow{\text{ASSIGN}}_A[\![X \leftarrow e]\!]$

- Base case (f)

  Apply $\overleftarrow{\text{ASSIGN}}_F[\![X \leftarrow e]\!][\alpha_C(C)]$ on the defined leaf nodes
  in ascending powers of ω

  Example:

$$f \equiv \omega \cdot x_1 + x_2$$

$$\overleftarrow{\text{ASSIGN}}_W[\![x_1 \leftarrow [-\infty, +\infty]]\!][\top_D] \equiv \omega^2 \cdot 1 + \omega \cdot 0 + x2 + 1$$

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

Abstract Definite Termination Semantics

Example

```
1x1 ← [-∞,∞]
2x2 ← [-∞, ∞]
 while 3(x1 > 0 ∧ x2 > 0) do
       4b ← [-∞, ∞]
       if 5(b ≥ 0) then
             6x1 ← x1 - 1
             7x2 ← [-∞, ∞]
       else
             8x2 ← x2 - 1
done9
```

$$f_3 \stackrel{\text{def}}{=} \begin{cases} 1 & x_1 \leq 0 \vee x_2 \leq 0 \\ \omega \cdot (x_1 - 7) + 7x_1 + 3x_2 - 5 & x_1 > 0 \wedge x_2 > 0 \end{cases}$$

Static Termination Analysis

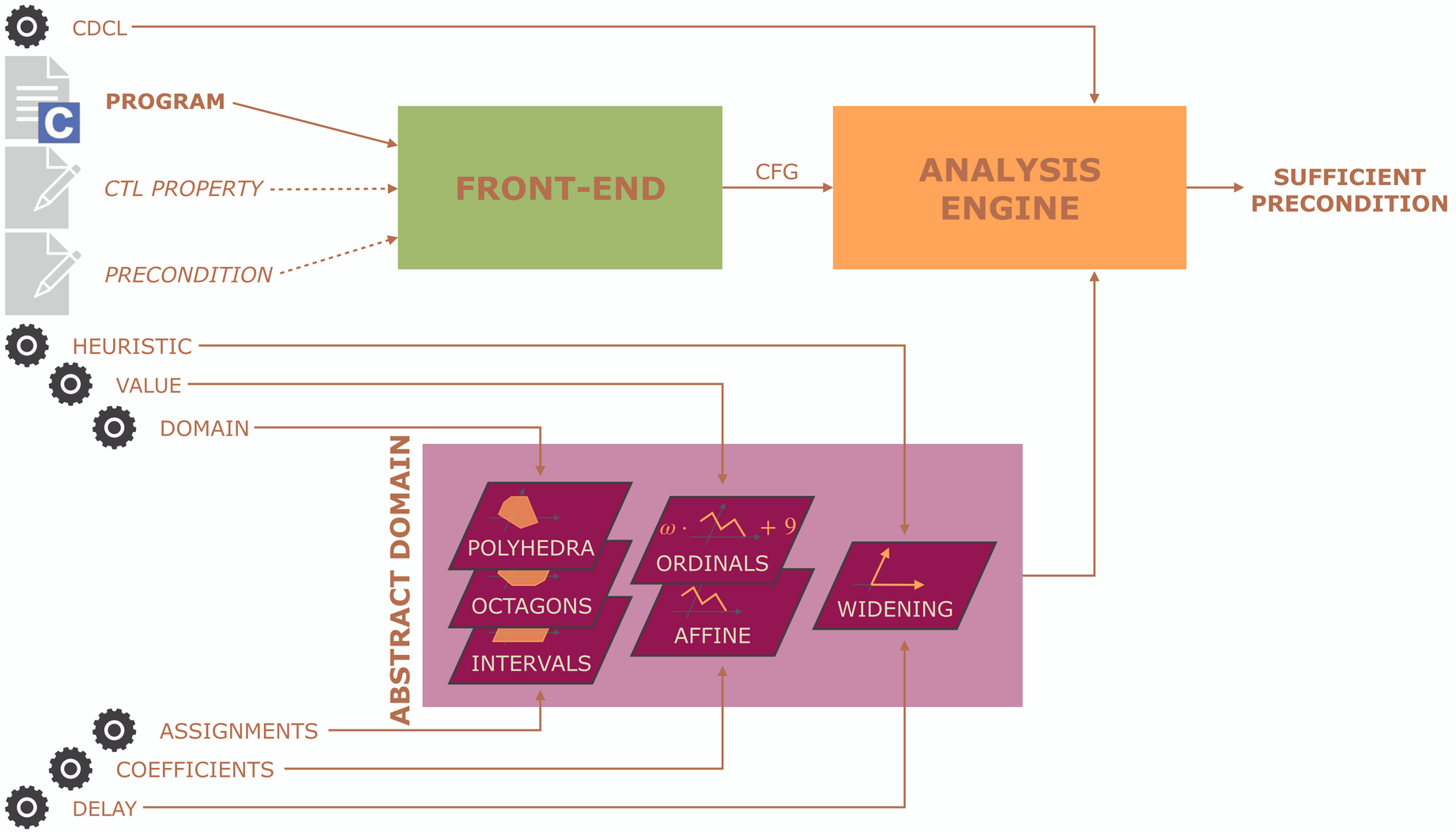
FuncTion

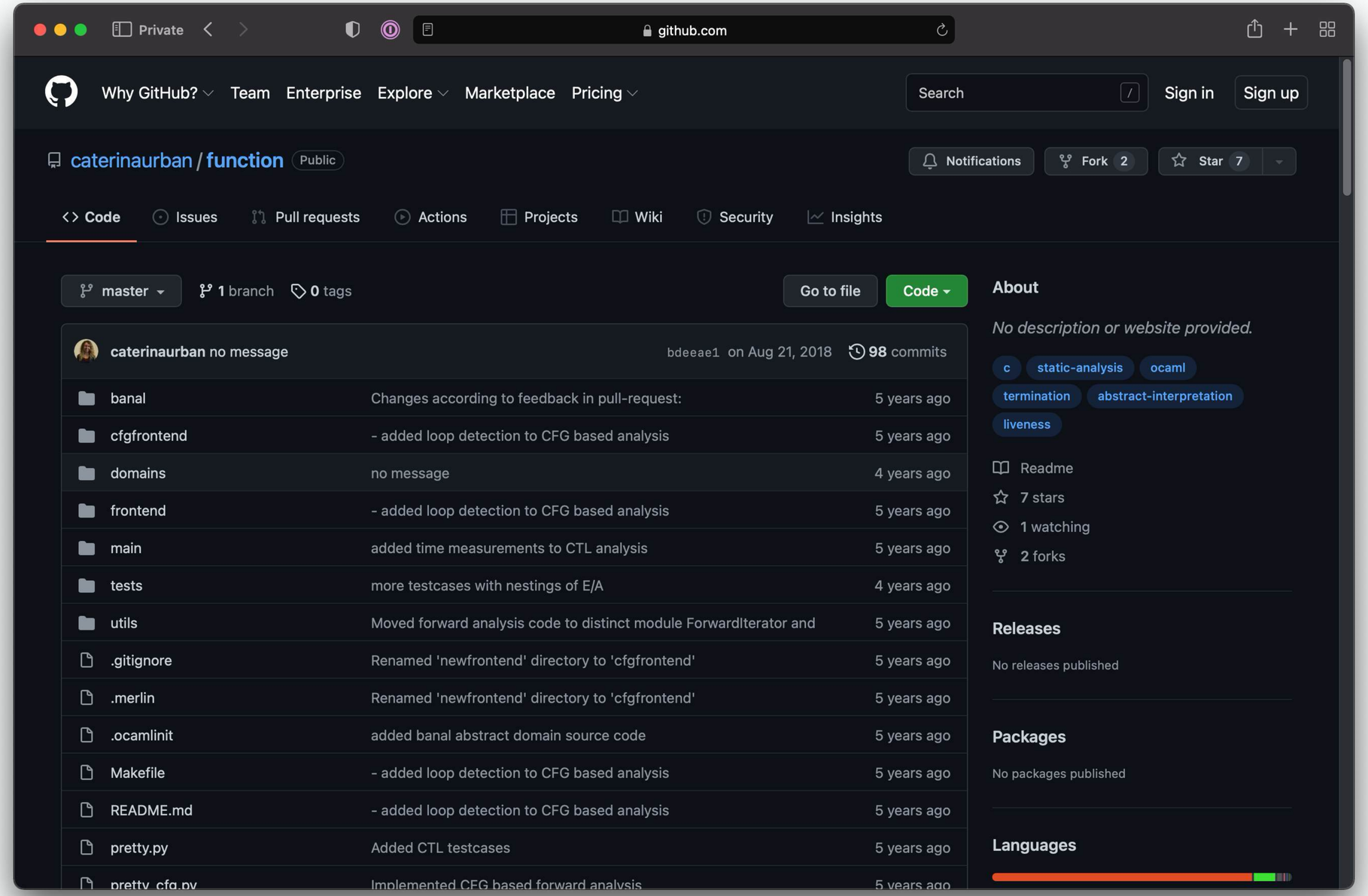
practical tools
targeting specific programs

abstract semantics, abstract domains
algorithmic approaches to decide program properties

concrete semantics
mathematical models of the program behavior

FuncTion

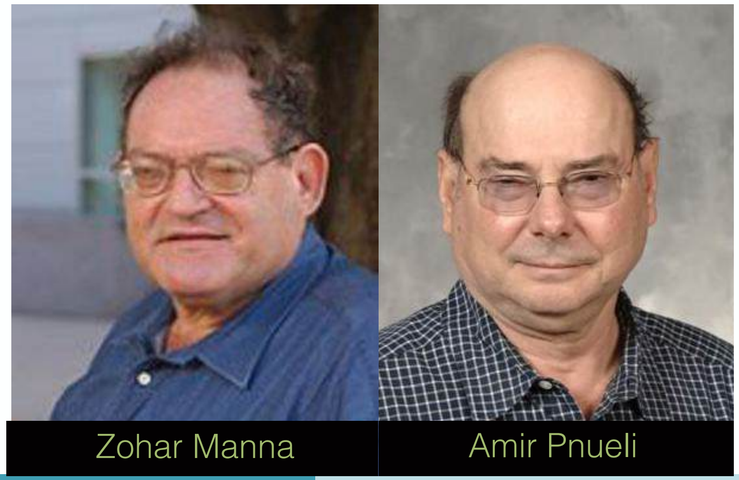

CDCL
PROGRAM
CTL PROPERTY
PRECONDITION
FRONT-END
CFG
ANALYSIS ENGINE
SUFFICIENT PRECONDITION
HEURISTIC
VALUE
DOMAIN
ABSTRACT DOMAIN
POLYHEDRA
OCTAGONS
INTERVALS
ORDINALS
AFFINE
WIDENING
ASSIGNMENTS
COEFFICIENTS
DELAY


Private
github.com
Why GitHub?
Team
Enterprise
Explore
Marketplace
Pricing
Search
Sign in
Sign up
caterinaurban / function
Public
Notifications
Fork 2
Star 7
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
master
1 branch
0 tags
Go to file
Code
caterinaurban no message
bdeeae1 on Aug 21, 2018
98 commits
banal Changes according to feedback in pull-request: 5 years ago
cfgfrontend - added loop detection to CFG based analysis 5 years ago
domains no message 4 years ago
frontend - added loop detection to CFG based analysis 5 years ago
main added time measurements to CTL analysis 5 years ago
tests more testcases with nestings of E/A 4 years ago
utils Moved forward analysis code to distinct module ForwardIterator and 5 years ago
.gitignore Renamed 'newfrontend' directory to 'cfgfrontend' 5 years ago
.merlin Renamed 'newfrontend' directory to 'cfgfrontend' 5 years ago
.ocamlinit added banal abstract domain source code 5 years ago
Makefile - added loop detection to CFG based analysis 5 years ago
README.md - added loop detection to CFG based analysis 5 years ago
pretty.py Added CTL testcases 5 years ago
pretty_cfg.py Implemented CFG based forward analysis 5 years ago
About
No description or website provided.
c
static-analysis
ocaml
termination
abstract-interpretation
liveness
Readme
7 stars
1 watching
2 forks
Releases
No releases published
Packages
No packages published
Languages

Liveness Properties

Liveness Properties

- **Guarantee Properties**
  “something good eventually happens at least once”
  - Example: Program Termination
- **Recurrence Properties**
  “something good eventually happens infinitely often”
  - Example: Starvation Freedom


Zohar Manna

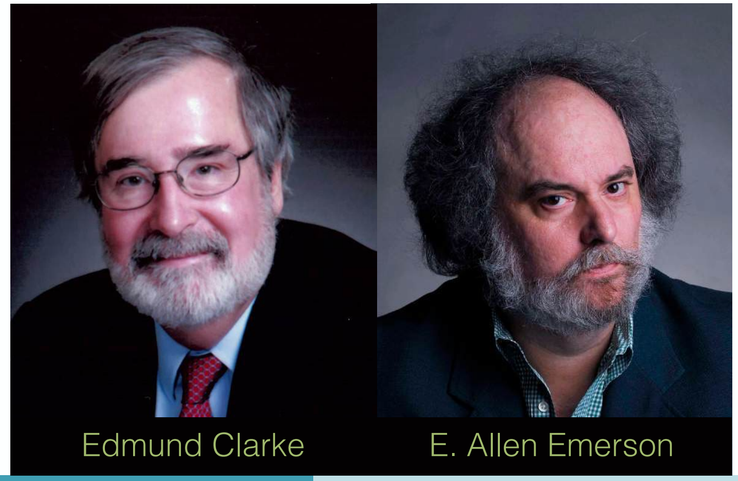
Amir Pnueli

Computation Tree Logic (CTL)

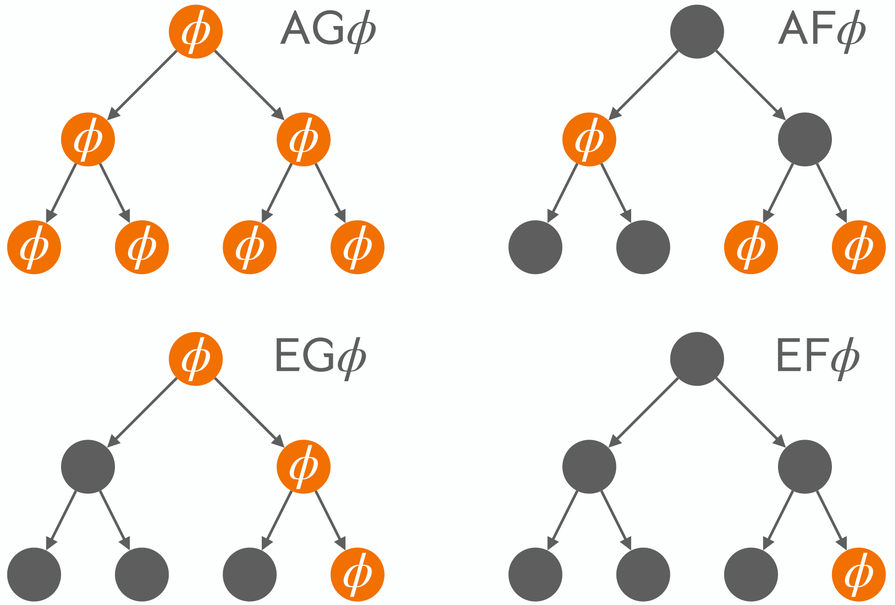
Branching Temporal Logic

$$\phi ::= a \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \mathrm{AX}\phi \mid \mathrm{AG}\phi \mid \mathrm{A}(\phi \mathrm{U} \phi) \mid \mathrm{EX}\phi \mid \mathrm{EG}\phi \mid \mathrm{E}(\phi \mathrm{U} \phi)$$

$$\mathrm{AF}\phi \equiv \mathrm{A}(\textbf{true}\ \mathrm{U}\ \phi) \qquad \mathrm{EF}\phi \equiv \mathrm{E}(\textbf{true}\ \mathrm{U}\ \phi)$$



Edmund Clarke E. Allen Emerson



Guarantee Properties

Guarantee Properties

“something good eventually happens at least once”

$$\mathsf{AF}\,\phi$$

$\phi ::= e \bowtie 0 \mid \ell : e \bowtie 0 \mid \phi \wedge \phi \mid \phi \vee \phi$ $\qquad\qquad$ $\ell \in \mathscr{L}$

Example:

```
1x ← [-∞, +∞]
while 2(x ≥ 0) do
        3x ← x + 1
done4
while 5(0 ≥ 0) do
        if 6(x ≤ 10) do
                7x ← x + 1
        else
                8x ← -x
done9
```

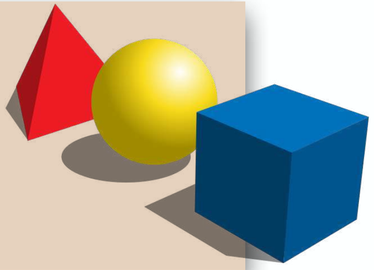
AF $(x = 3)$ is satisfied for $I \stackrel{\text{def}}{=} \{(1,\rho) \in \Sigma \mid \rho(x) \leq 3\}$

Static Guarantee Analysis

3-Step Recipe

practical tools
targeting specific programs

abstract semantics, abstract domains
algorithmic approaches to decide program properties



concrete semantics
mathematical models of the program behavior

Static Guarantee Analysis

Program Guarantee Semantics

practical tools
targeting specific programs

abstract semantics, abstract domains
algorithmic approaches to decide program properties

concrete semantics
mathematical models of the program behavior

Guarantee Program Semantics

Definite Termination Program Semantics

$$\mathscr{R}_M = \mathsf{lfp}^{\leq} \overline{F}_M$$

$$\overline{F}_M(f)\sigma \stackrel{\mathsf{def}}{=} \begin{cases} 0 & \sigma \in \mathscr{B} \\ \sup\{f(\sigma') + 1 \mid (\sigma, \sigma') \in \tau\} & \sigma \in \widetilde{\mathsf{pre}}_\tau(\mathsf{dom}(f)) \\ \text{undefined} & \text{otherwise} \end{cases}$$

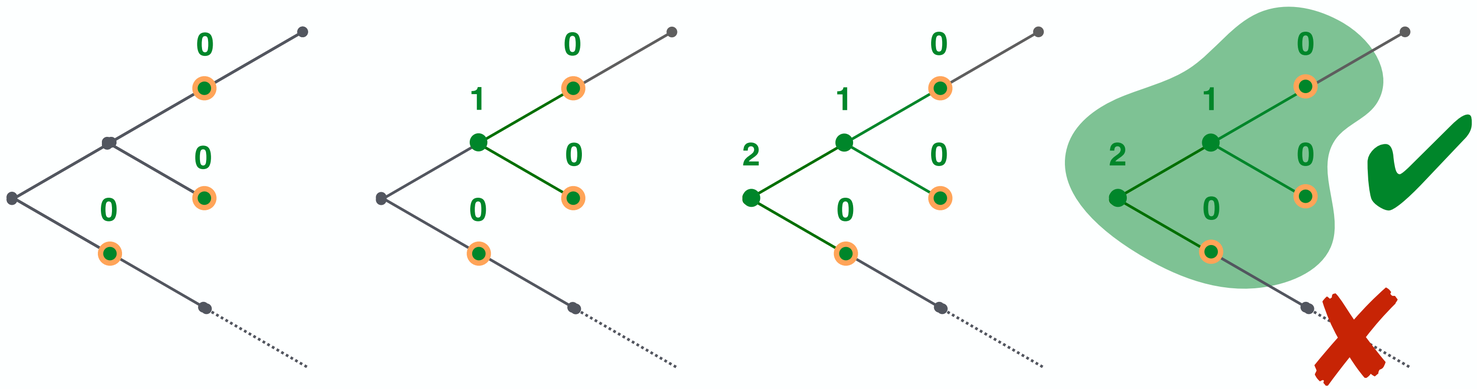
$$\mathscr{R}^{\varphi}_G \stackrel{\mathsf{def}}{=} \mathsf{lfp}^{\leq} \overline{F}_G \left[\{\sigma \in \Sigma \mid \sigma \vDash \varphi\}\right]$$

$$\overline{F}_G[S]f \stackrel{\mathsf{def}}{=} \lambda\sigma\,. \begin{cases} 0 & \sigma \in S \\ \sup\{f(\sigma') + 1 \mid (\sigma, \sigma') \in \tau\} & \sigma \notin S \wedge \sigma \in \widetilde{\mathsf{pre}}_\tau(\mathsf{dom}(f)) \\ \text{undefined} & \text{otherwise} \end{cases}$$

Guarantee Program Semantics

$$\mathscr{R}^{\varphi}_{G} \stackrel{\text{def}}{=} \text{lfp}^{\preceq}\, \overline{F}_{G}\left[\{\sigma \in \Sigma \mid \sigma \vDash \varphi\}\right]$$

$$\overline{F}_{G}[S]f \stackrel{\text{def}}{=} \lambda\sigma.\begin{cases} 0 & \sigma \in S \\ \sup\{f(\sigma') + 1 \mid (\sigma,\sigma') \in \tau\} & \sigma \notin S \wedge \sigma \in \widetilde{\text{pre}}_{\tau}(\text{dom}(f)) \\ \text{undefined} & \text{otherwise} \end{cases}$$



Theorem (Soudness and Completeness)

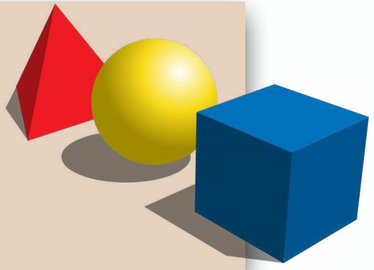
A program satisfies a **guarantee property** AF φ starting from a set of initial states I if and only if $I \subseteq \text{dom}(\mathscr{R}^{\varphi}_{G})$

Static Guarantee Analysis

Abstract Program Guarantee Semantics

practical tools
targeting specific programs

abstract semantics, abstract domains
algorithmic approaches to decide program properties



concrete semantics
mathematical models of the program behavior

Abstract Program Guarantee Semantics

Piecewise-Defined Ranking Functions Abstract Domain

For each program instruction stmt, we define a transformer $\mathscr{R}^{\varphi\#}_{G}[\![\text{stmt}]\!] : \mathscr{A} \to \mathscr{A}$:

- $\mathscr{R}^{\varphi\#}_{G}[\![X \leftarrow e]\!]t \overset{\text{def}}{=} \text{RESET}^{G}_{A}[\![\varphi]\!](\overleftarrow{\text{ASSIGN}}_{A}[\![X \leftarrow e]\!]t)$
- $\mathscr{R}^{\varphi\#}_{G}[\![\textbf{if}\ \ e \bowtie 0\ \textbf{then}\ s\ \textbf{end}]\!]t \overset{\text{def}}{=} \text{RESET}^{G}_{A}[\![\varphi]\!](\text{FILTER}_{A}[\![e \bowtie 0]\!](\mathscr{R}^{\varphi\#}_{G}[\![s]\!]t) \curlyvee_{T} \text{FILTER}_{A}[\![e \not\bowtie 0]\!]t))$
- $\mathscr{R}^{\varphi\#}_{G}[\![\textbf{while}\ \ e \bowtie 0\ \textbf{do}\ s\ \textbf{done}]\!]t \overset{\text{def}}{=} \text{lfp}^{\#}\overline{F}^{\varphi\#}_{G}$

  where $\overline{F}^{\varphi\#}_{G}(x) \overset{\text{def}}{=} \text{RESET}^{G}_{A}[\![\varphi]\!](\text{FILTER}_{A}[\![e \bowtie 0]\!](\mathscr{R}^{\varphi\#}_{G}[\![s]\!]x) \curlyvee_{T} \text{FILTER}_{A}[\![e \not\bowtie 0]\!](t)))$
- $\mathscr{R}^{\varphi\#}_{G}[\![s_1; s_2]\!]t \overset{\text{def}}{=} \mathscr{R}^{\varphi\#}_{G}[\![s_1]\!](\mathscr{R}^{\varphi\#}_{G}[\![s_2]\!]t)$

Abstract Program Guarantee Semantics

Piecewise-Defined Ranking Functions Abstract Domain

Definition

The **abstract guaranteee semantics** $\mathscr{R}^{\varphi\#}_{G}[\![\mathsf{s}^{\ell}]\!] \in \mathscr{A}$ of a program s^{ℓ} is:

$$\mathscr{R}^{\varphi\#}_{G}[\![\mathsf{s}^{\ell}]\!] \stackrel{\text{def}}{=} \mathscr{R}^{\varphi\#}_{G}[\![\mathsf{s}]\!](\mathsf{RESET}^{G}_{A}[\![\varphi]\!](\mathsf{LEAF}: \bot_{F}))$$

where $\mathscr{R}^{\varphi\#}_{G}[\![\mathsf{s}]\!] : \mathscr{A} \to \mathscr{A}$ is the abstract guarantee semantics of each program instruction s

Theorem (Soundness)

$$\mathscr{R}_{G}[\![\mathsf{s}^{\ell}]\!] \preccurlyeq \gamma_{A}(\mathscr{R}^{\#}_{G}[\![\mathsf{s}^{\ell}]\!])$$

Corollary (Soundness)

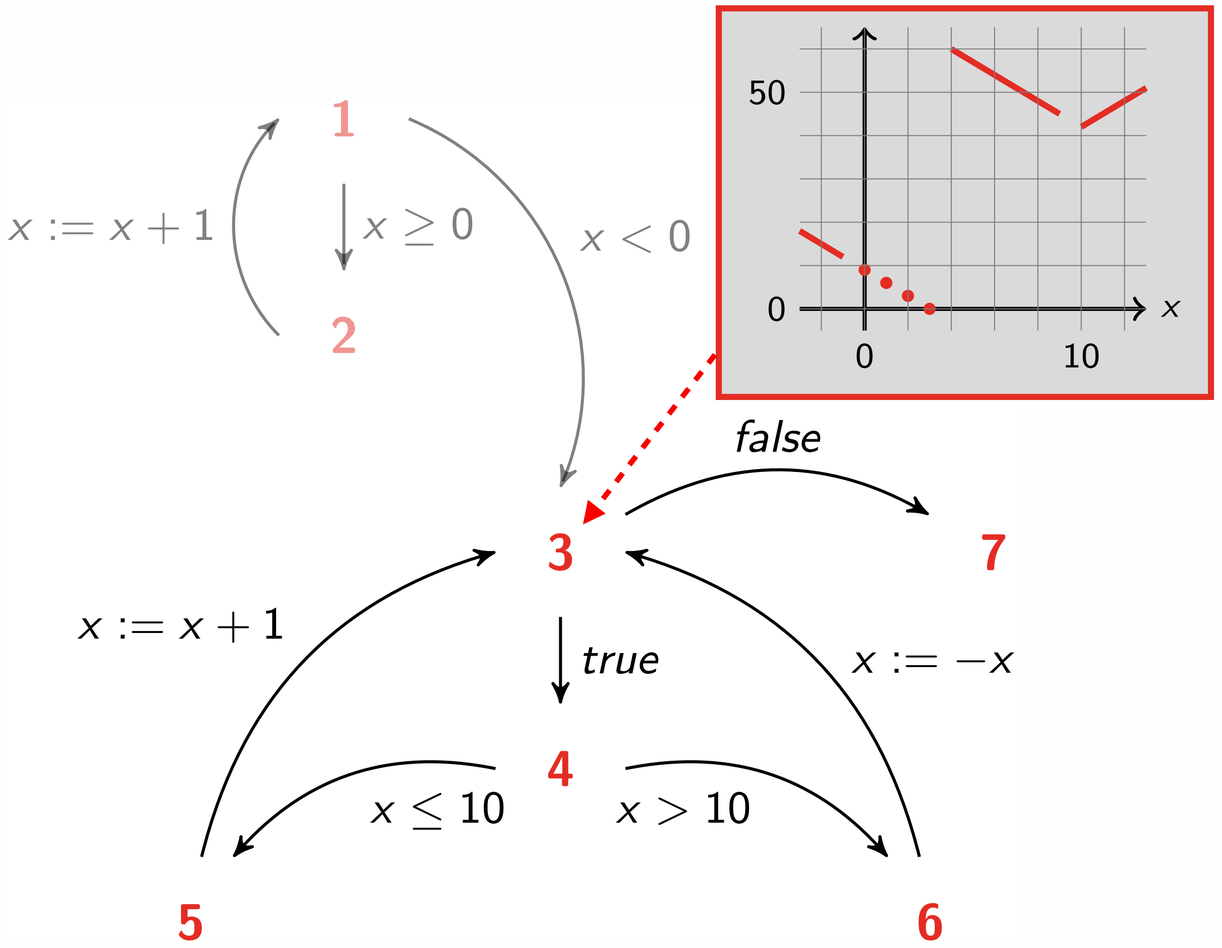
A program s^{ℓ} satisfies a **guarantee property** $\mathsf{AF}\,\varphi$ starting from a set of initial states I if $I \subseteq \mathrm{dom}(\gamma_{A}(\mathscr{R}^{\varphi\#}_{G}[\![\mathsf{s}^{\ell}]\!]))$

Abstract Program Guarantee Semantics

Example

```
while [1](x ≥ 0) do
      [2]x ← x + 1
done
while [3](0 ≥ 0) do
      if [4](x ≤ 10) do
            [5]x ← x + 1
      else
            [6]x ← -x
done[7]
```

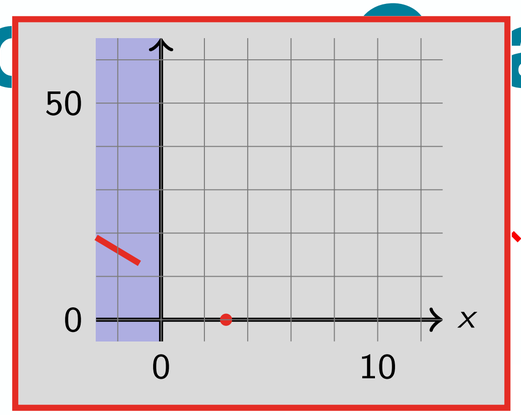
Property
AF $(x = 3)$

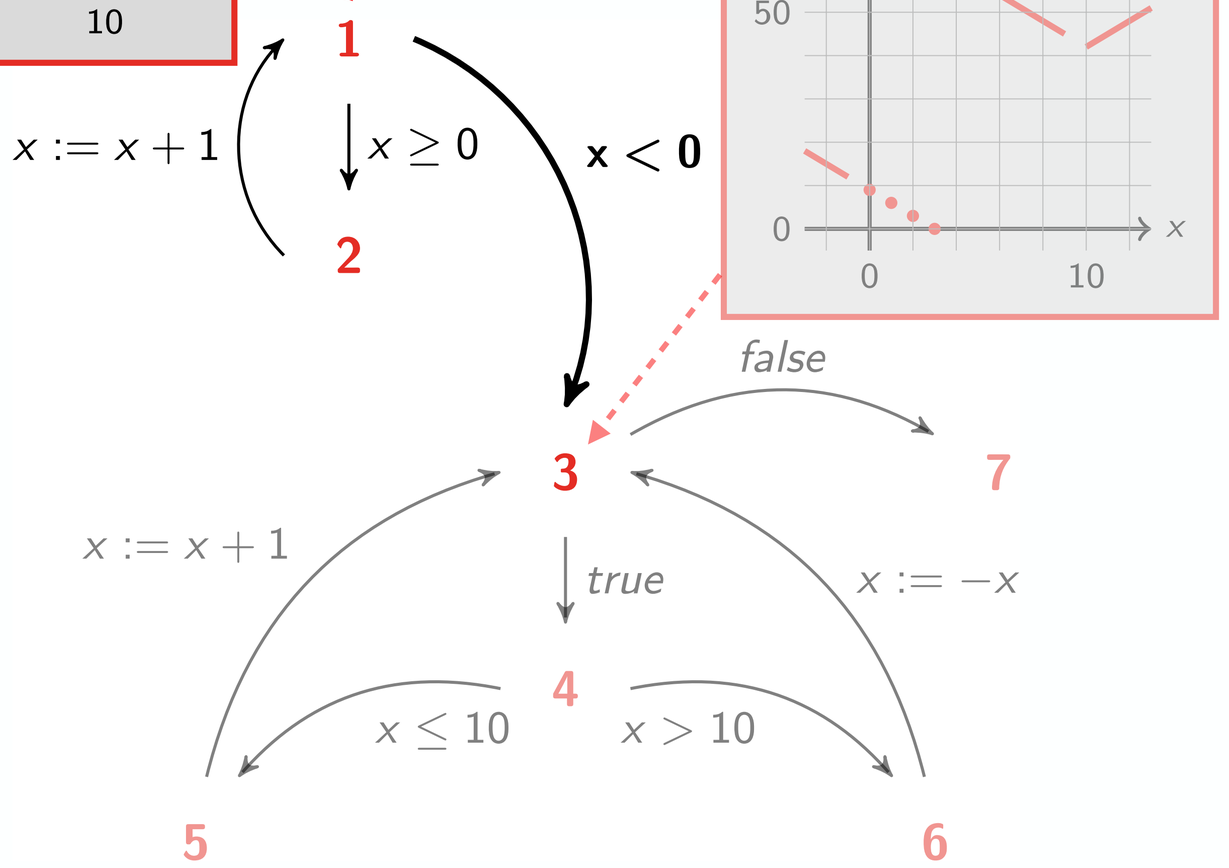


Abstract Pro arantee Semantics

Example

```
while [1](x ≥ 0) do
    [2]x ← x + 1
done
while [3](0 ≥ 0) do
    if [4](x ≤ 10) do
        [5]x ← x + 1
    else
        [6]x ← -x
done[7]
```





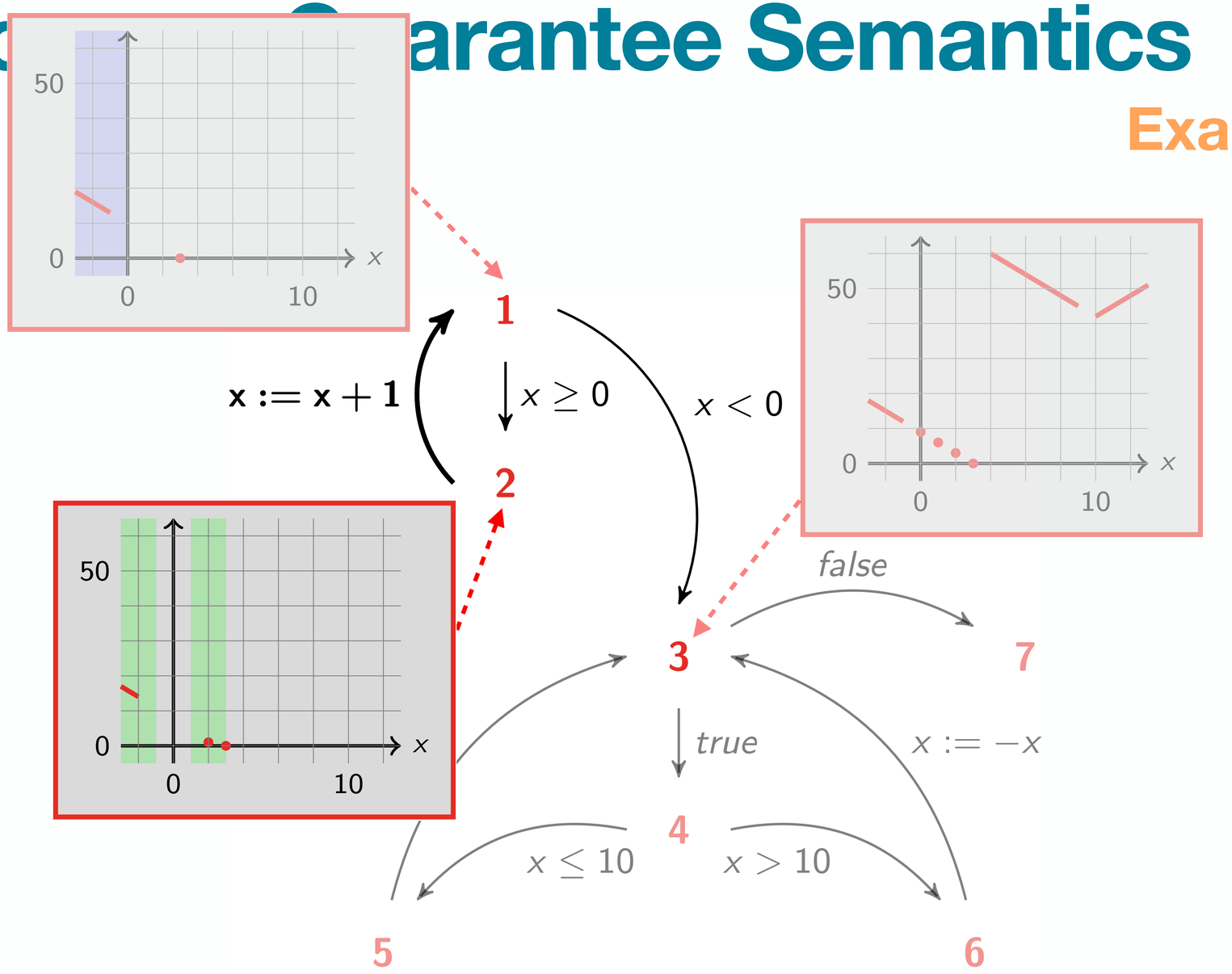
Property

$\mathrm{AF}\,(x = 3)$

Abstract Pro arantee Semantics

Example

```
while [1](x ≥ 0) do
    [2]x ← x + 1
done
while [3](0 ≥ 0) do
    if [4](x ≤ 10) do
        [5]x ← x + 1
    else
        [6]x ← -x
done[7]
```

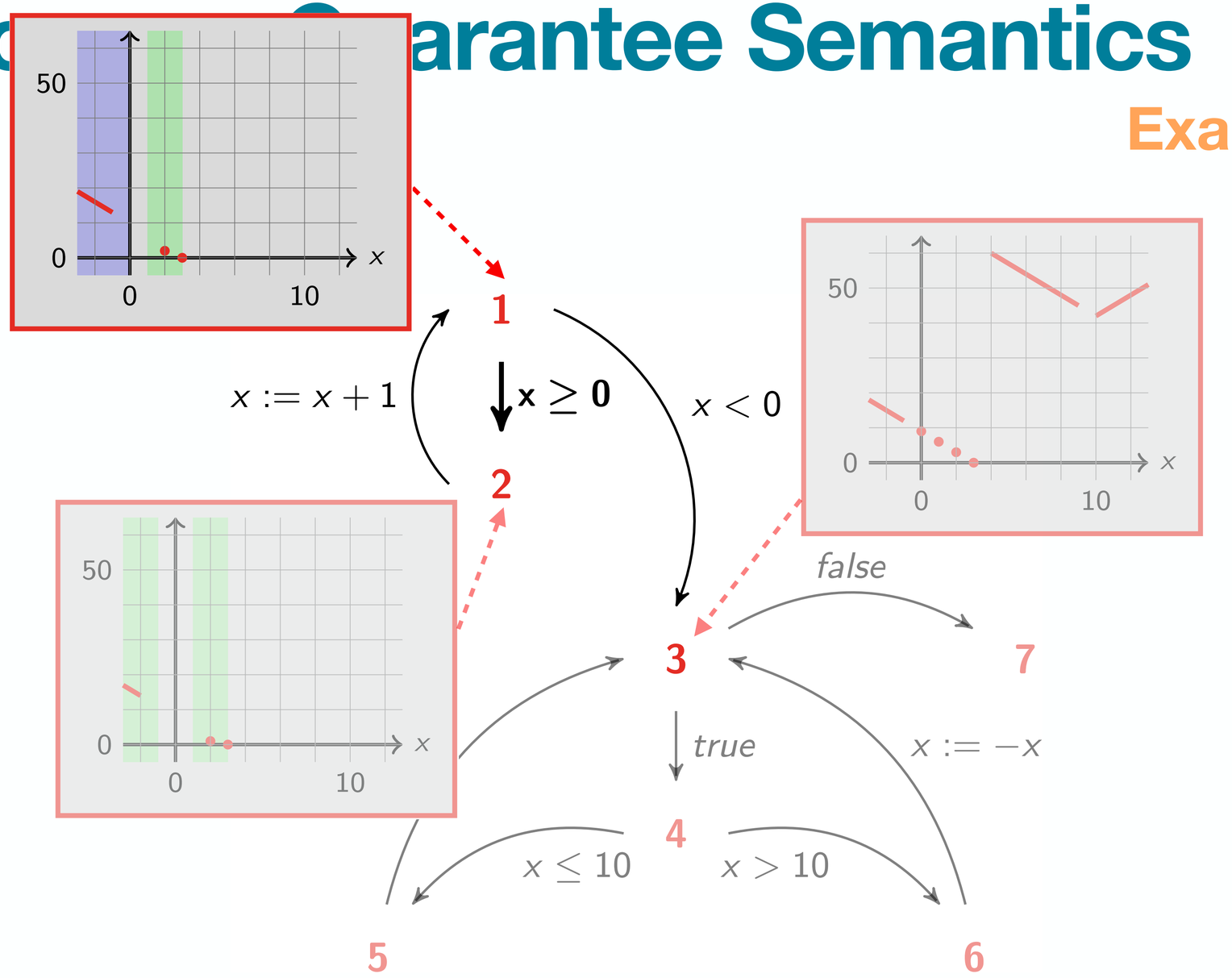


Property
AF $(x = 3)$

Abstract Pro arantee Semantics

Example

```
while [1](x ≥ 0) do
    [2]x ← x + 1
done
while [3](0 ≥ 0) do
    if [4](x ≤ 10) do
        [5]x ← x + 1
    else
        [6]x ← -x
done[7]
```

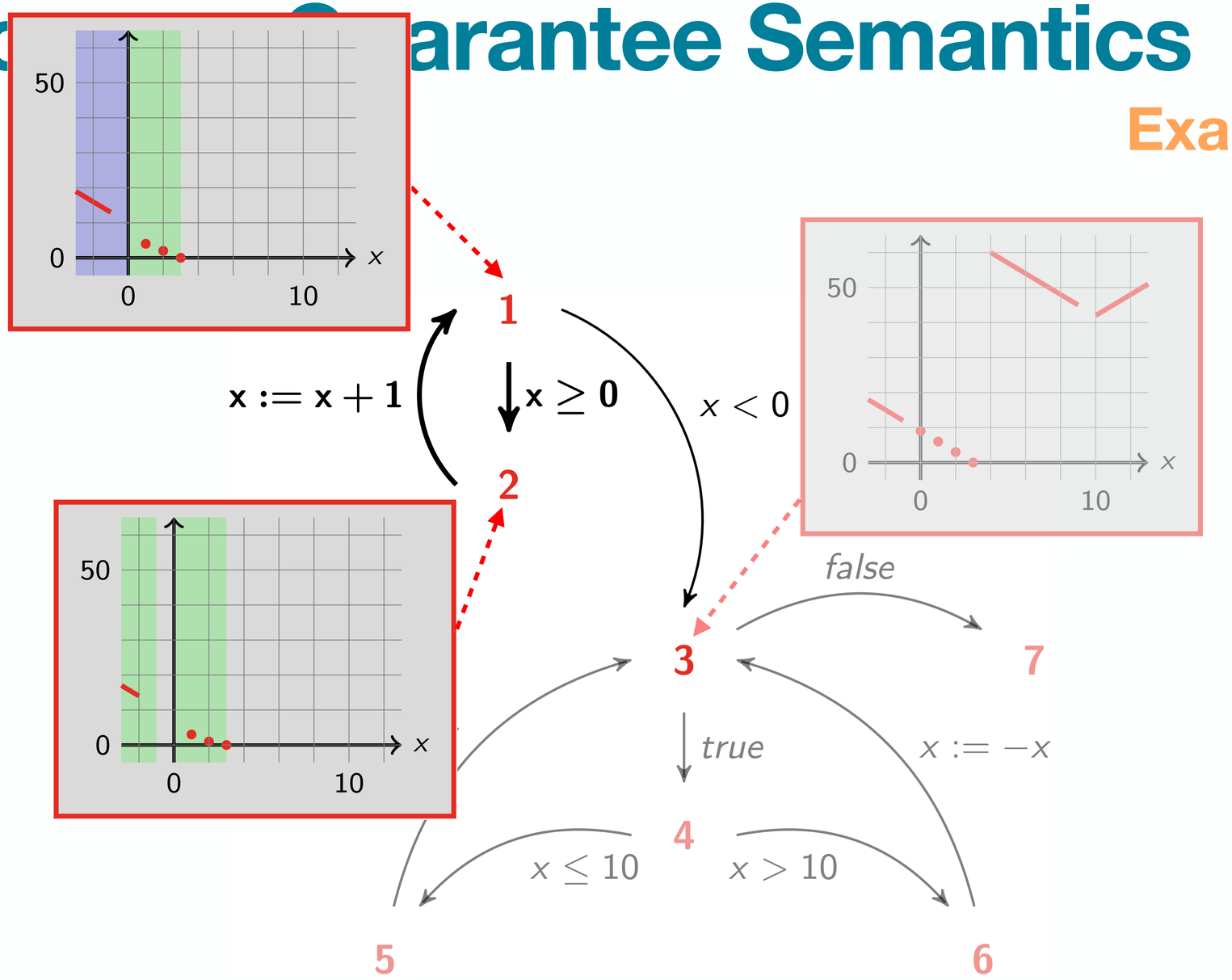


Property
AF $(x = 3)$

Abstract Pro arantee Semantics

Example

```
while [1](x ≥ 0) do
      [2]x ← x + 1
done
while [3](0 ≥ 0) do
      if [4](x ≤ 10) do
            [5]x ← x + 1
      else
            [6]x ← -x
done[7]
```

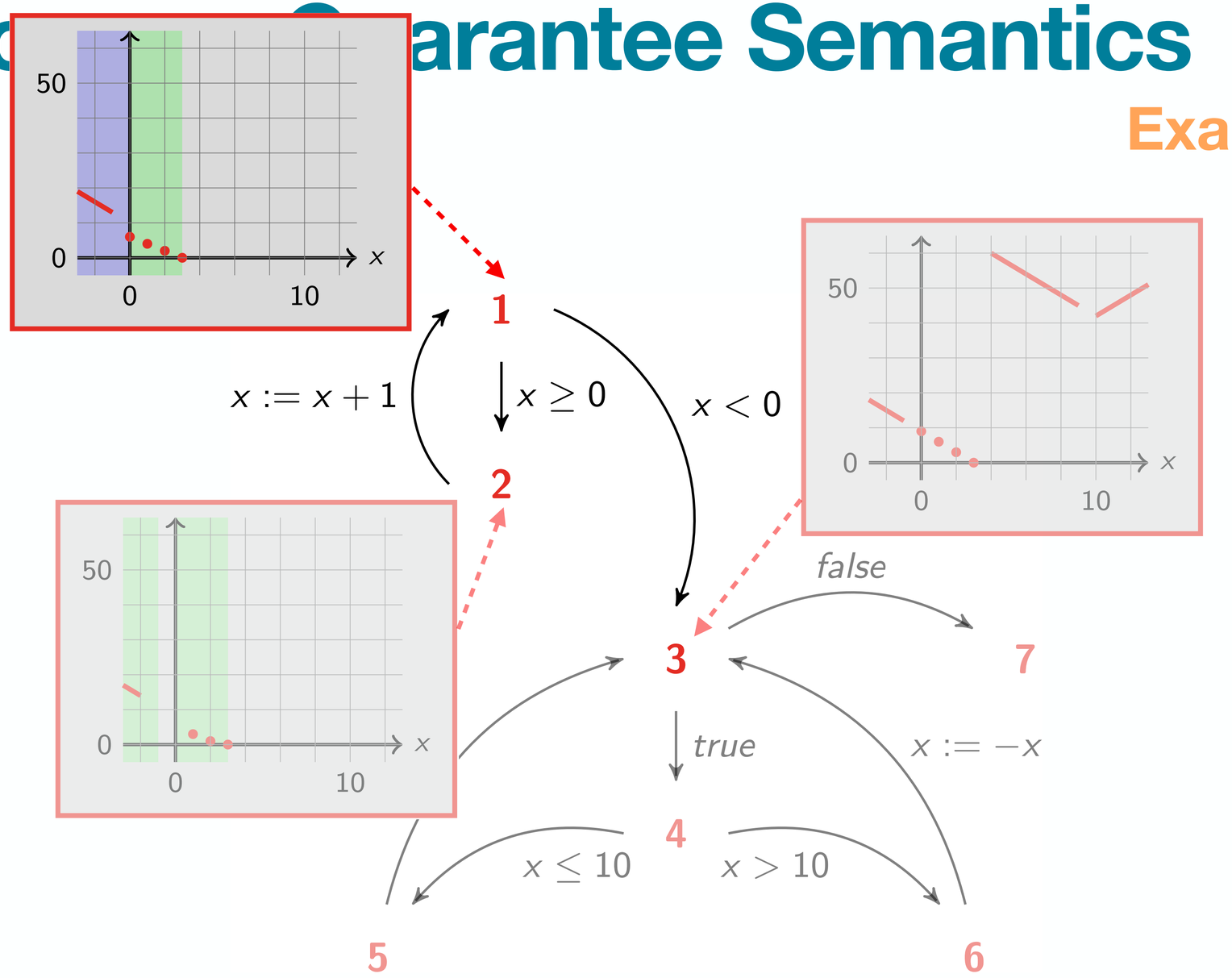


Property
AF $(x = 3)$

Abstract Pro arantee Semantics

Example

```
while [1](x ≥ 0) do
    [2]x ← x + 1
done
while [3](0 ≥ 0) do
    if [4](x ≤ 10) do
        [5]x ← x + 1
    else
        [6]x ← -x
done[7]
```

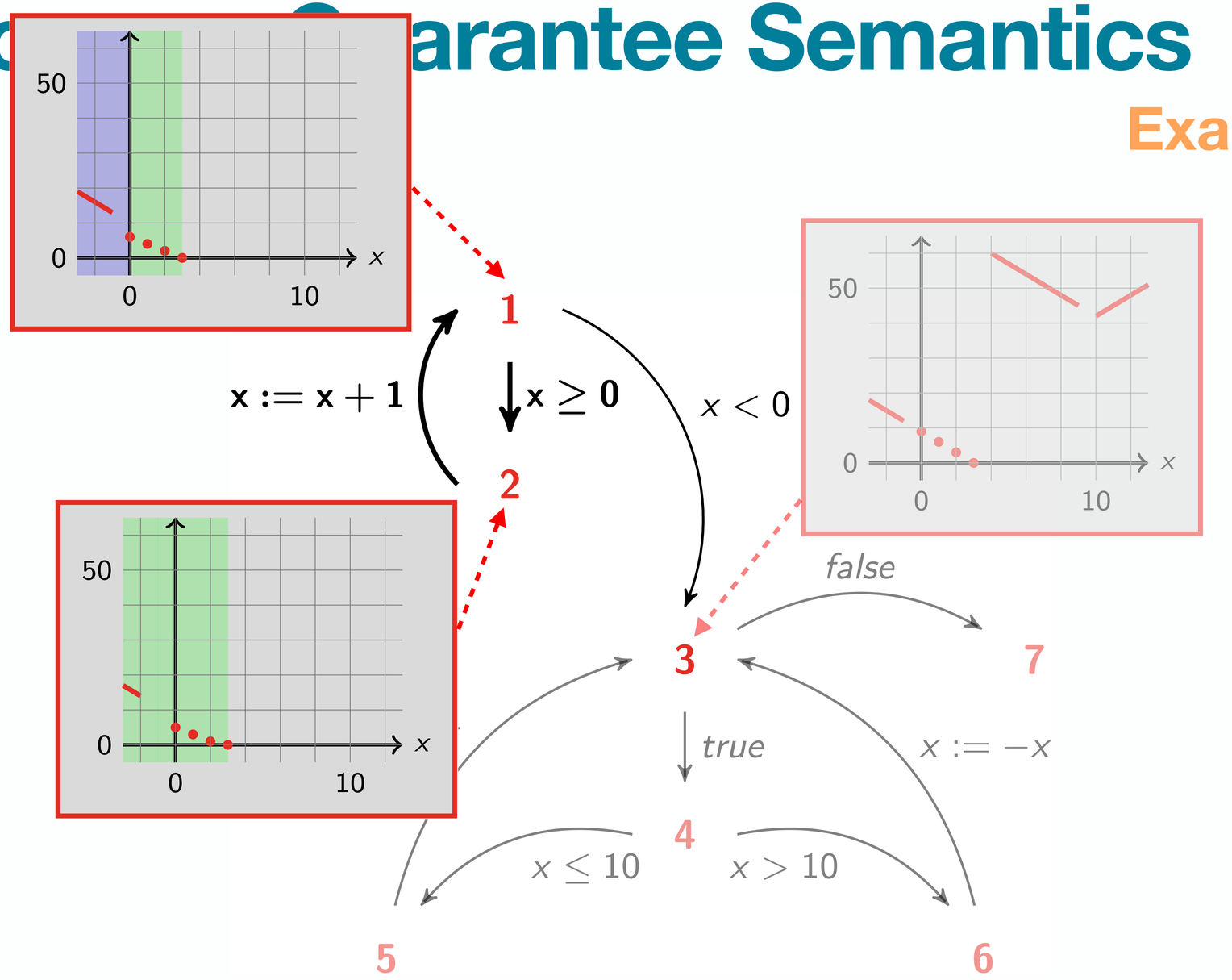


Property
AF $(x = 3)$

Abstract Pro arantee Semantics

Example

```
while [1](x ≥ 0) do
    [2]x ← x + 1
done
while [3](0 ≥ 0) do
    if [4](x ≤ 10) do
        [5]x ← x + 1
    else
        [6]x ← -x
done[7]
```

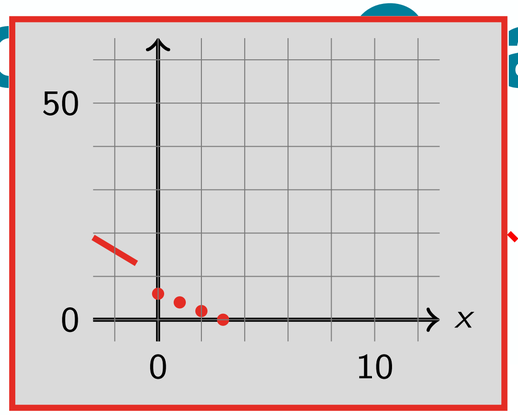


Property
AF $(x = 3)$

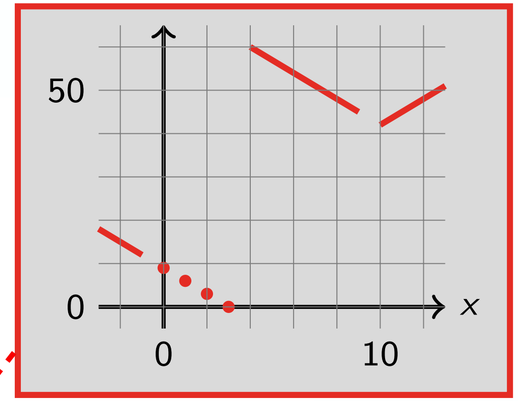
Abstract Pro arantee Semantics

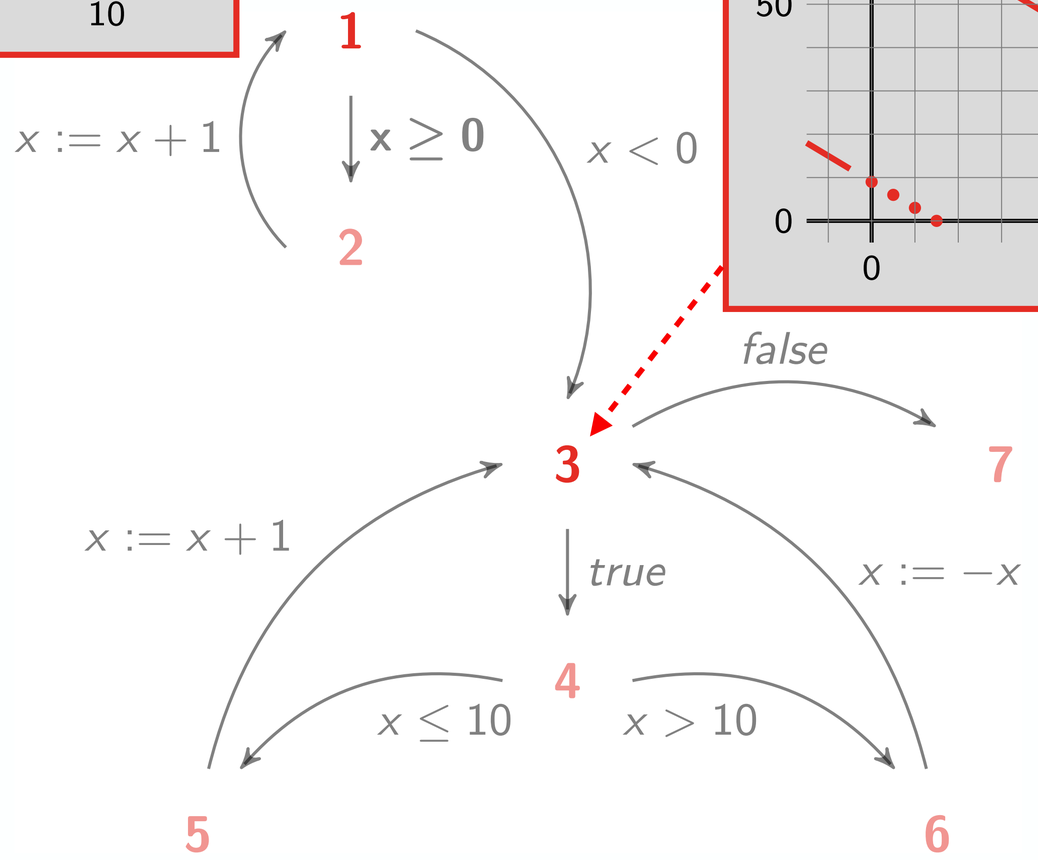
Example

```
while [1](x ≥ 0) do
      [2]x ← x + 1
done
while [3](0 ≥ 0) do
      if [4](x ≤ 10) do
            [5]x ← x + 1
      else
            [6]x ← -x
done[7]
```



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





Property
AF $(x = 3)$

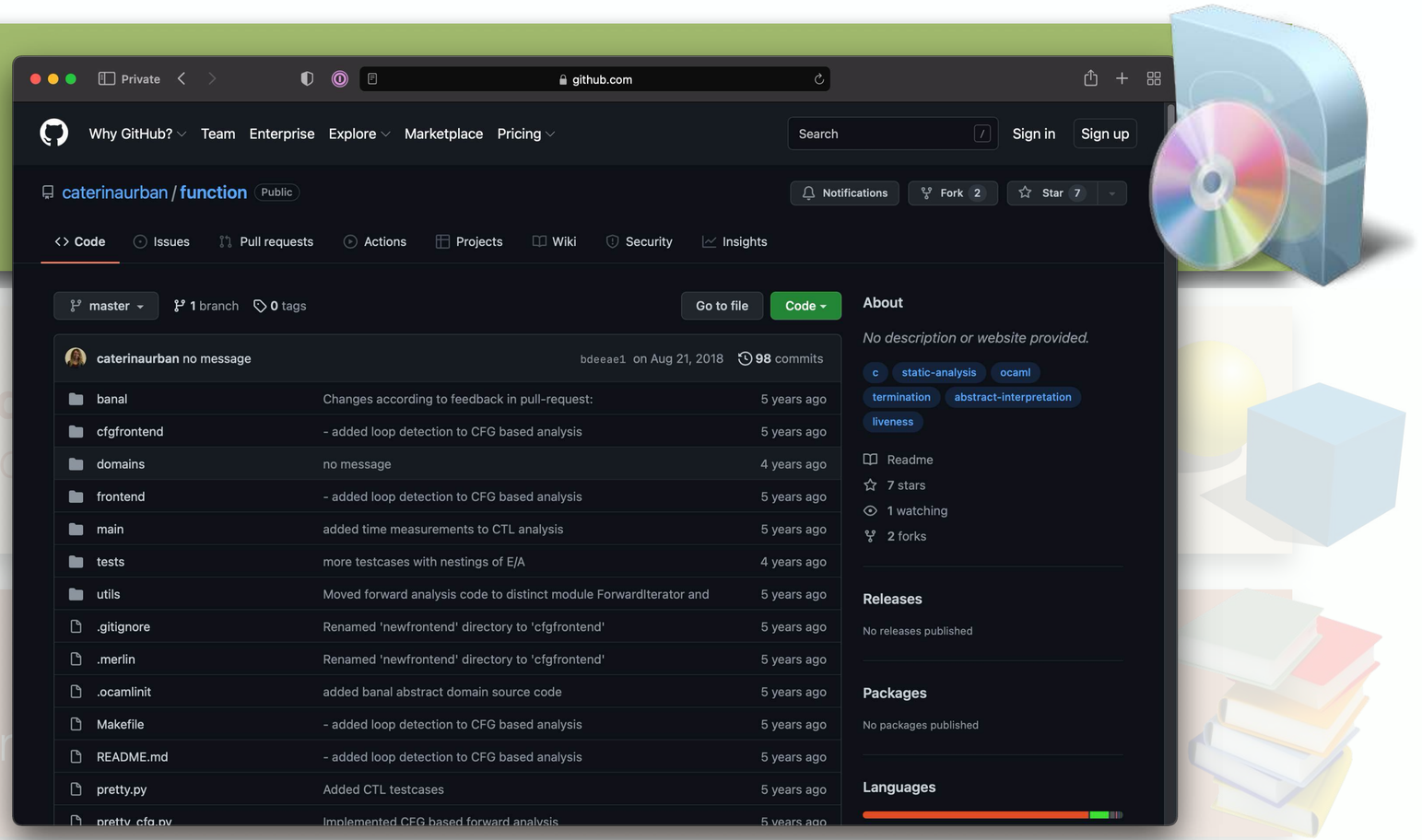
Static Guarantee Analysis

FuncTion

practical tools
targeting specific programs

abstract semantics, abstract
algorithmic approaches to de

concrete semantics
mathematical models of the pr



Recurrence Properties

Recurrence Properties

“something good eventually happens infinitely often”

$$\text{AG AF } \phi$$

$$\phi ::= e \bowtie 0 \mid \ell : e \bowtie 0 \mid \phi \wedge \phi \mid \phi \vee \phi \qquad \ell \in \mathscr{L}$$

Example:

```
1x ← [-∞, +∞]
while 2(x ≥ 0) do
        3x ← x + 1
done4
while 5(0 ≥ 0) do
        if 6(x ≤ 10) do
                7x ← x + 1
        else
                8x ← -x
done9
```

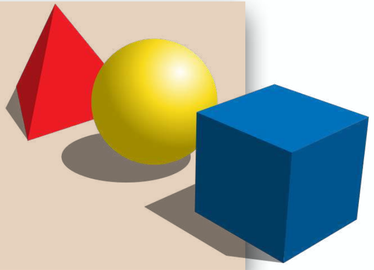
AG AF $(x = 3)$ is satisfied for $I \stackrel{\text{def}}{=} \{(1,\rho) \in \Sigma \mid \rho(x) < 0\}$

Static Recurrence Analysis

3-Step Recipe

practical tools
targeting specific programs

abstract semantics, abstract domains
algorithmic approaches to decide program properties



concrete semantics
mathematical models of the program behavior

Static Recurrence Analysis

Program Recurrence Semantics

practical tools
targeting specific programs

abstract semantics, abstract domains
algorithmic approaches to decide program properties

concrete semantics
mathematical models of the program behavior

Recurrence Program Semantics

Guarantee Semantics

$$\mathscr{R}^{\varphi}_{G} \overset{\text{def}}{=} \mathsf{lfp}^{\leq} \overline{F}_{G}\left[\{\sigma \in \Sigma \mid \sigma \vDash \varphi\}\right]$$

$$\overline{F}_{G}[S]f \overset{\text{def}}{=} \lambda\sigma\,.\begin{cases} 0 & \sigma \in S \\ \sup\{f(\sigma') + 1 \mid (\sigma,\sigma') \in \tau\} & \sigma \notin S \wedge \sigma \in \widetilde{\mathsf{pre}}_{\tau}(\mathsf{dom}(f)) \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\mathscr{R}^{\varphi}_{R} \overset{\text{def}}{=} \mathsf{gfp}_{\mathscr{R}^{\varphi}_{G}}^{\preceq} \overline{F}_{R}$$

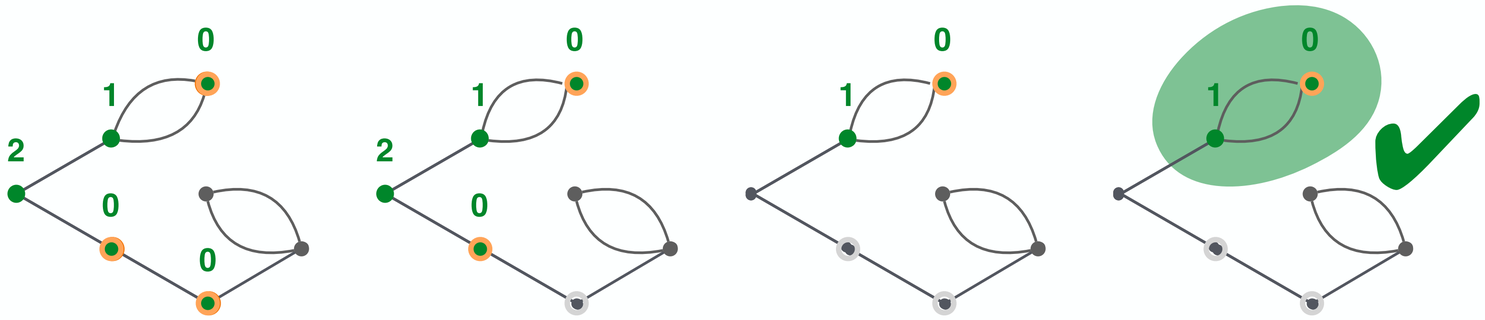
build upon the semantics of sub-formulas

$$\overline{F}_{R}(f)\sigma \overset{\text{def}}{=} \begin{cases} f(\sigma) & \sigma \in \mathsf{dom}(f) \cap \widetilde{\mathsf{pre}}_{\tau}(\mathsf{dom}(f)) \\ \text{undefined} & \text{otherwise} \end{cases}$$

Recurrence Program Semantics

$$\mathscr{R}^{\varphi}_{R} \overset{\text{def}}{=} \text{gfp}_{\mathscr{R}^{\varphi}_{G}}^{\preceq} \overline{F}_{R}$$

$$\overline{F}_{R}(f)\sigma \overset{\text{def}}{=} \begin{cases} f(\sigma) & \sigma \in \text{dom}(f) \cap \widetilde{\text{pre}}_{\tau}(\text{dom}(f)) \\ \text{undefined} & \text{otherwise} \end{cases}$$



Theorem (Soudness and Completeness)

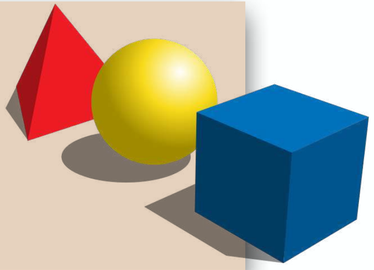
A program satisfies a **recurrence property** AG AF φ starting from a set of initial states I if and only if $I \subseteq \text{dom}(\mathscr{R}^{\varphi}_{F})$

Static Recurrence Analysis

Abstract Program Recurrence Semantics

practical tools
targeting specific programs

abstract semantics, abstract domains
algorithmic approaches to decide program properties



concrete semantics
mathematical models of the program behavior

Abstract Program Recurrence Semantics

Piecewise-Defined Ranking Functions Abstract Domain

For each program instruction stmt, we define a transformer $\mathscr{R}^{\varphi\#}_R[\![\text{stmt}]\!] : \mathscr{A} \to \mathscr{A}$:

- $\mathscr{R}^{\varphi\#}_R[\![X \leftarrow e]\!]t \stackrel{\text{def}}{=} \text{RESET}^R_A[\![\varphi]\!](\overleftarrow{\text{ASSIGN}}_A[\![X \leftarrow e]\!]t)$
- $\mathscr{R}^{\varphi\#}_R[\![\textbf{if}\ \ e \bowtie 0\ \textbf{then}\ s\ \textbf{end}]\!]t \stackrel{\text{def}}{=} \text{RESET}^R_A[\![\varphi]\!](\text{FILTER}_A[\![e \bowtie 0]\!](\mathscr{R}^{\varphi\#}_G[\![s]\!]t) \curlyvee_T \text{FILTER}_A[\![e \not\bowtie 0]\!]t))$
- $\mathscr{R}^{\varphi\#}_G[\![\textbf{while}\ \ e \bowtie 0\ \textbf{do}\ s\ \textbf{done}]\!]t \stackrel{\text{def}}{=} \text{gpf}^{\#}_{G(t)}\overline{F}^{\varphi\#}_R$

  where $G \stackrel{\text{def}}{=} \mathscr{R}^{\varphi\#}_G[\![\textbf{while}\ \ e \bowtie 0\ \textbf{do}\ s\ \textbf{done}]\!]$

  $\overline{F}^{\varphi\#}_R(x) \stackrel{\text{def}}{=} \text{RESET}^R_A[\![\varphi]\!](\text{FILTER}_A[\![e \bowtie 0]\!](\mathscr{R}^{\varphi\#}_G[\![s]\!]x) \curlyvee_T \text{FILTER}_A[\![e \not\bowtie 0]\!](t)))$
- $\mathscr{R}^{\varphi\#}_G[\![s_1; s_2]\!]t \stackrel{\text{def}}{=} \mathscr{R}^{\varphi\#}_G[\![s_1]\!](\mathscr{R}^{\varphi\#}_G[\![s_2]\!]t)$

Dual Widening

Definition

Let $\langle \mathscr{D}, \sqsubseteq \rangle$ be a poset. A **dual widening** $\overline{\nabla} \colon \mathscr{D} \times \mathscr{D} \to \mathscr{D}$ is such that:

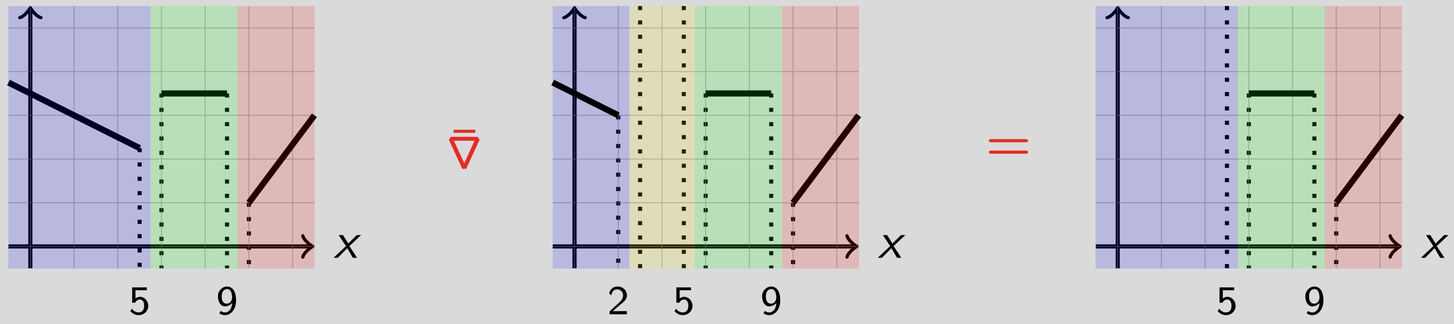
(i) for all elements $x, y \in \mathscr{D}$ we have $x \sqsupseteq x \mathbin{\overline{\nabla}} y$ and $y \sqsupseteq x \mathbin{\overline{\nabla}} y$

(ii) for all decreasing chains $x_0 \sqsupseteq x_1 \sqsupseteq \ldots \sqsupseteq x_n \sqsupseteq \ldots$ the chain

$$y_0 \stackrel{\text{def}}{=} x_0 \qquad y_{n+1} \stackrel{\text{def}}{=} y_n \mathbin{\overline{\nabla}} x_{n+1}$$

is ultimately stationary

Example



Abstract Program Recurrence Semantics

Piecewise-Defined Ranking Functions Abstract Domain

Definition

The **abstract recurrence semantics** $\mathscr{R}_R^{\varphi\#}[\![\mathsf{s}^\ell]\!] \in \mathscr{A}$ of a program s^ℓ is:

$$\mathscr{R}_R^{\varphi\#}[\![\mathsf{s}^\ell]\!] \stackrel{\text{def}}{=} \mathscr{R}_R^{\varphi\#}[\![\mathsf{s}]\!](\mathsf{LEAF}: \bot_F)$$

where $\mathscr{R}_R^{\varphi\#}[\![\mathsf{s}]\!] : \mathscr{A} \to \mathscr{A}$ is the abstract recurrence semantics of each program instruction s

Theorem (Soundness)

$$\mathscr{R}_R[\![\mathsf{s}^\ell]\!] \preccurlyeq \gamma_A(\mathscr{R}_R^{\#}[\![\mathsf{s}^\ell]\!])$$

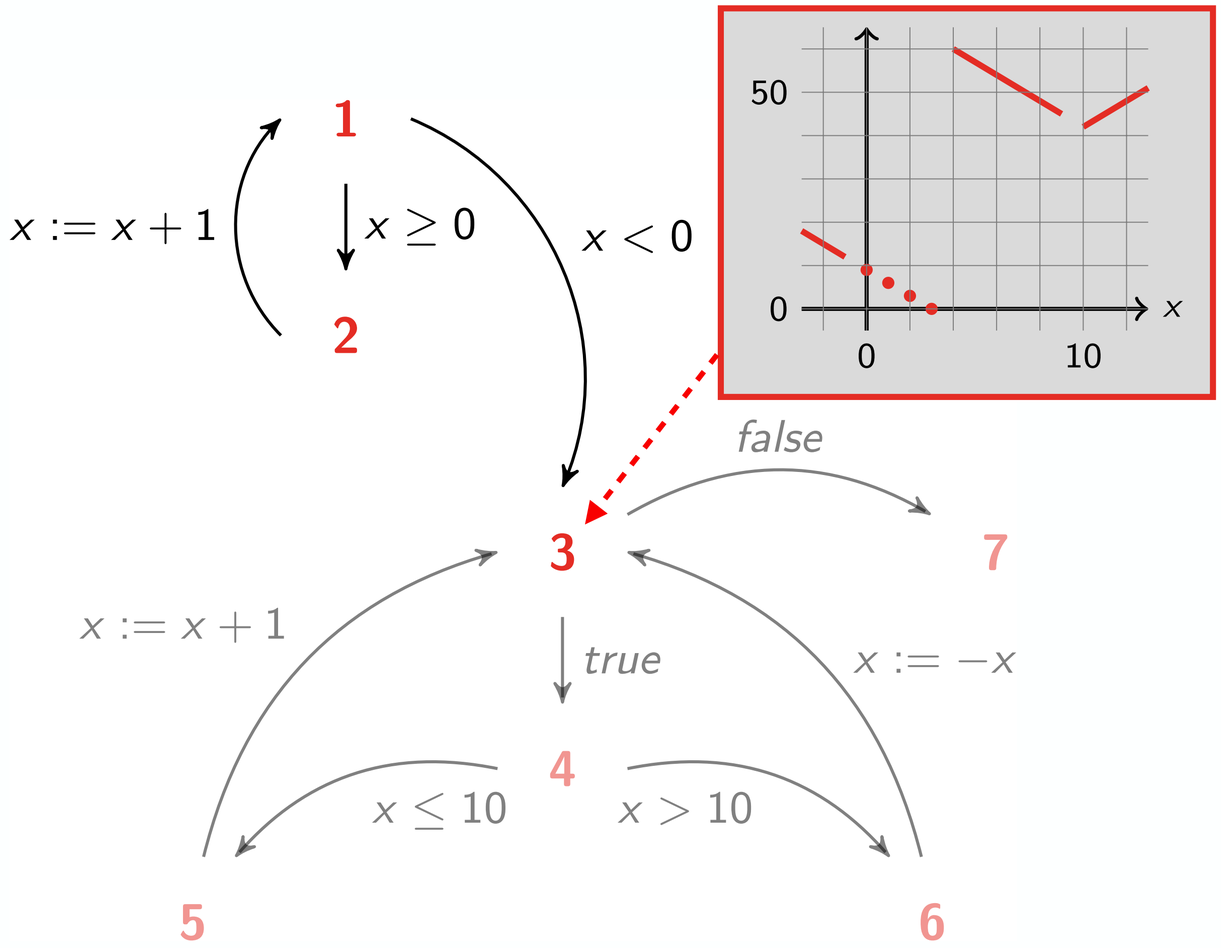
Corollary (Soundness)

A program s^ℓ satisfies a **recurrence property** AG AF φ starting from a set of initial states I if $I \subseteq \mathrm{dom}(\gamma_A(\mathscr{R}_R^{\varphi\#}[\![\mathsf{s}^\ell]\!]))$

Abstract Program Recurrence Semantics

Example

```
while [1](x ≥ 0) do
      [2]x ← x + 1
done
while [3](0 ≥ 0) do
      if [4](x ≤ 10) do
            [5]x ← x + 1
      else
            [6]x ← -x
done[7]
```

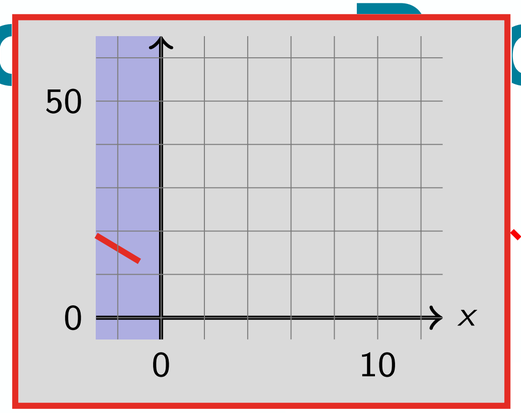


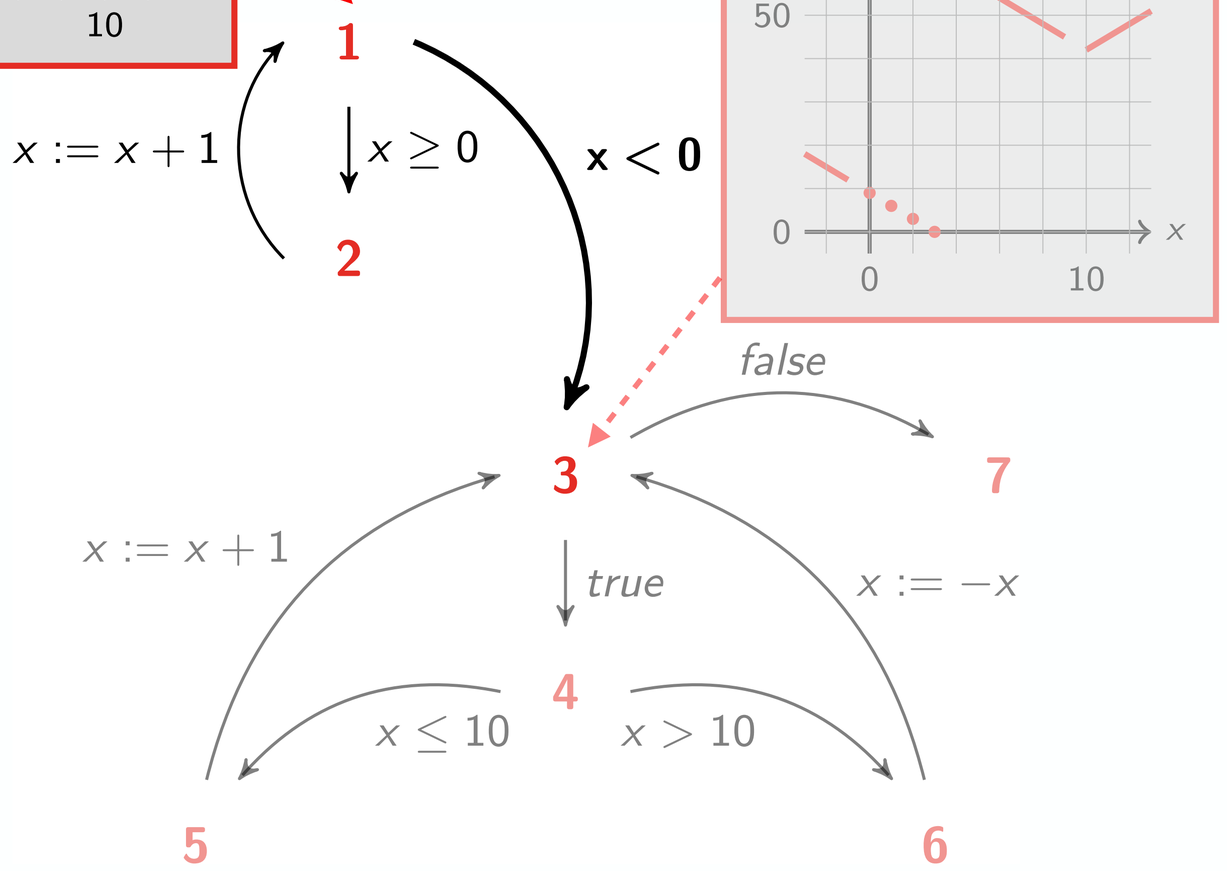
Property
AG AF $(x = 3)$

Abstract Pro currence Semantics

Example

```
while [1](x ≥ 0) do
    [2]x ← x + 1
done
while [3](0 ≥ 0) do
    if [4](x ≤ 10) do
        [5]x ← x + 1
    else
        [6]x ← -x
done[7]
```





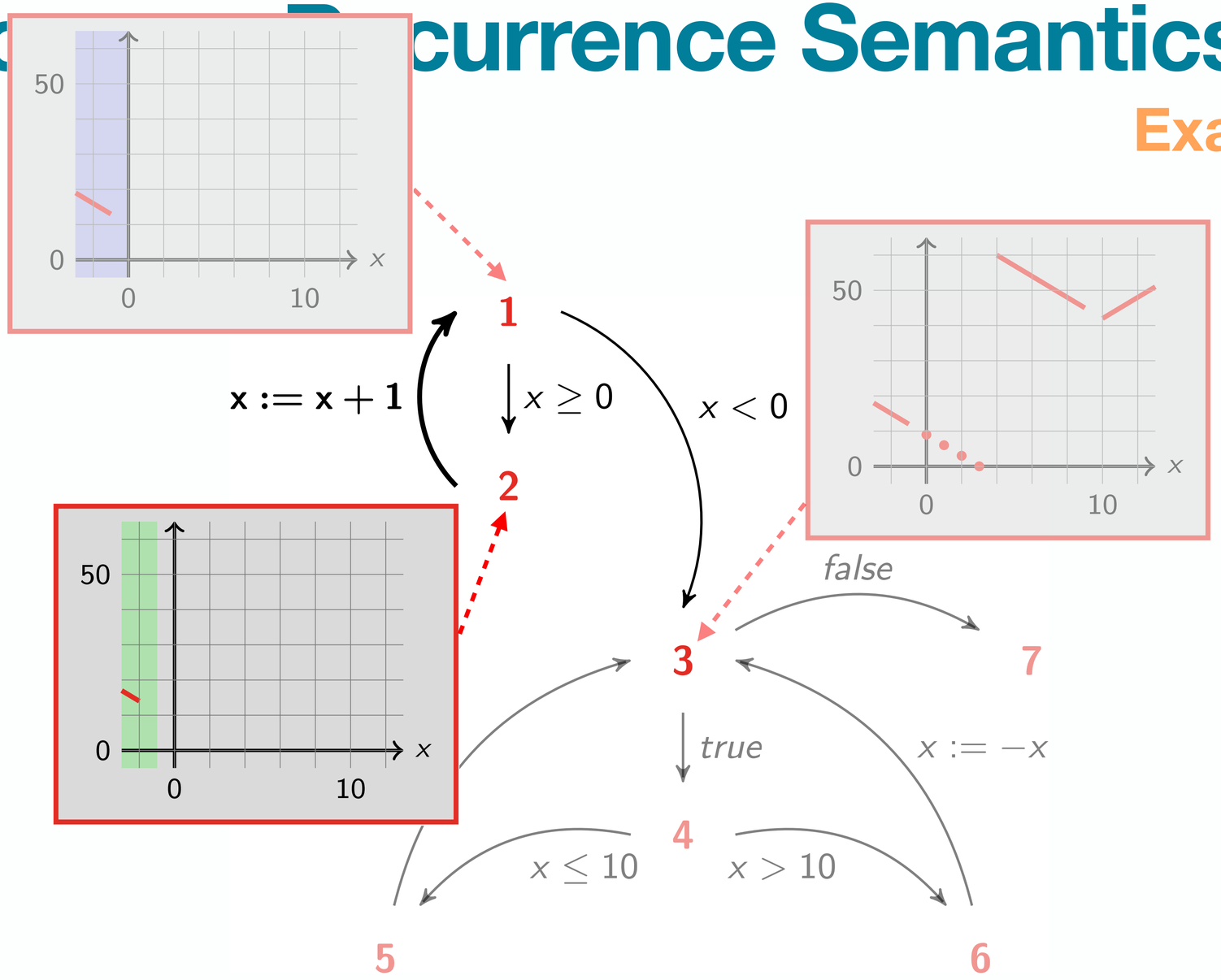
Property

$$\mathrm{AG\,AF}\,(x = 3)$$

Abstract Pro currence Semantics

Example

```
while [1](x ≥ 0) do
    [2]x ← x + 1
done
while [3](0 ≥ 0) do
    if [4](x ≤ 10) do
        [5]x ← x + 1
    else
        [6]x ← -x
done[7]
```

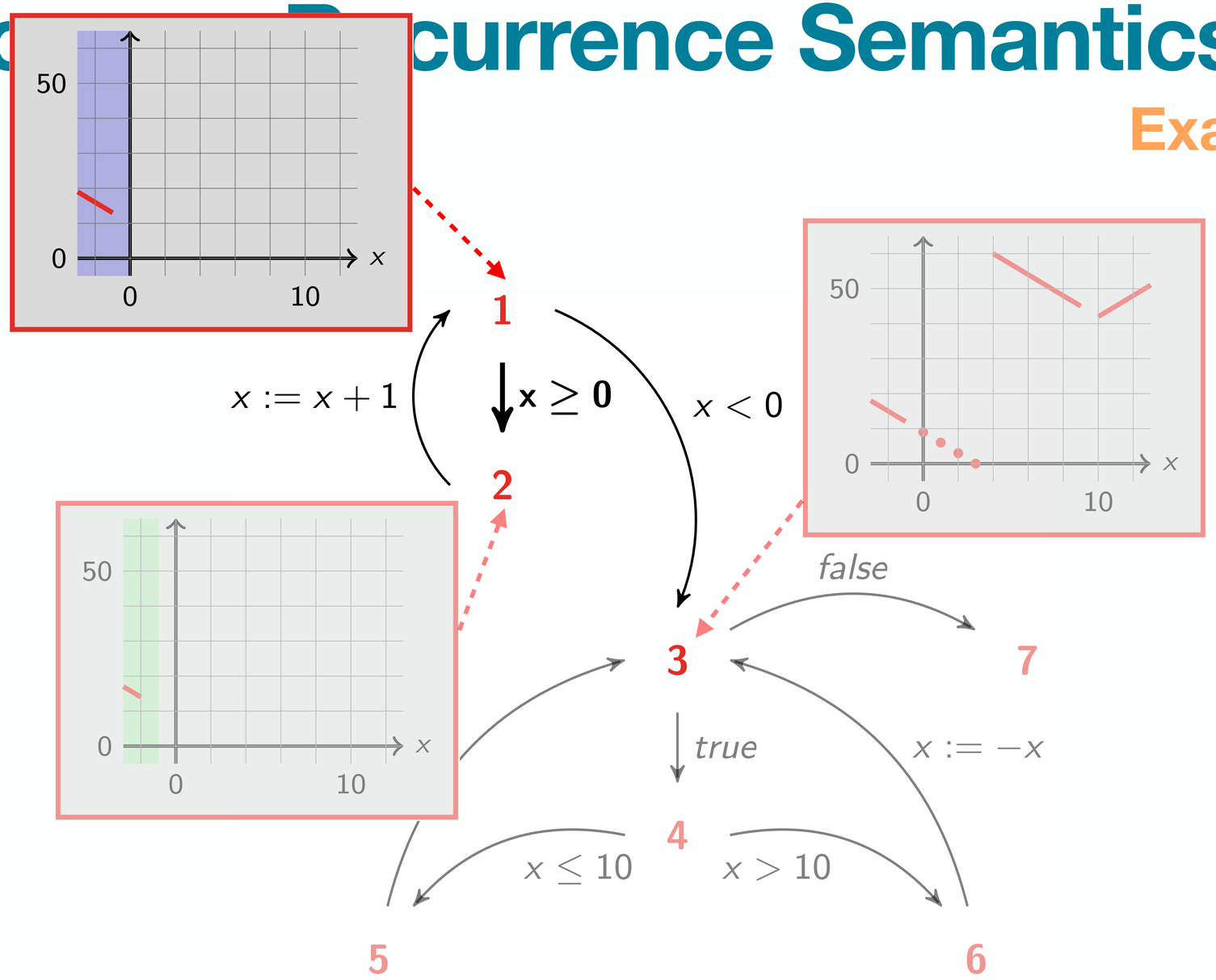


Property
AG AF $(x = 3)$

Abstract Pro currence Semantics

Example

```
while [1](x ≥ 0) do
      [2]x ← x + 1
done
while [3](0 ≥ 0) do
      if [4](x ≤ 10) do
            [5]x ← x + 1
      else
            [6]x ← -x
done[7]
```



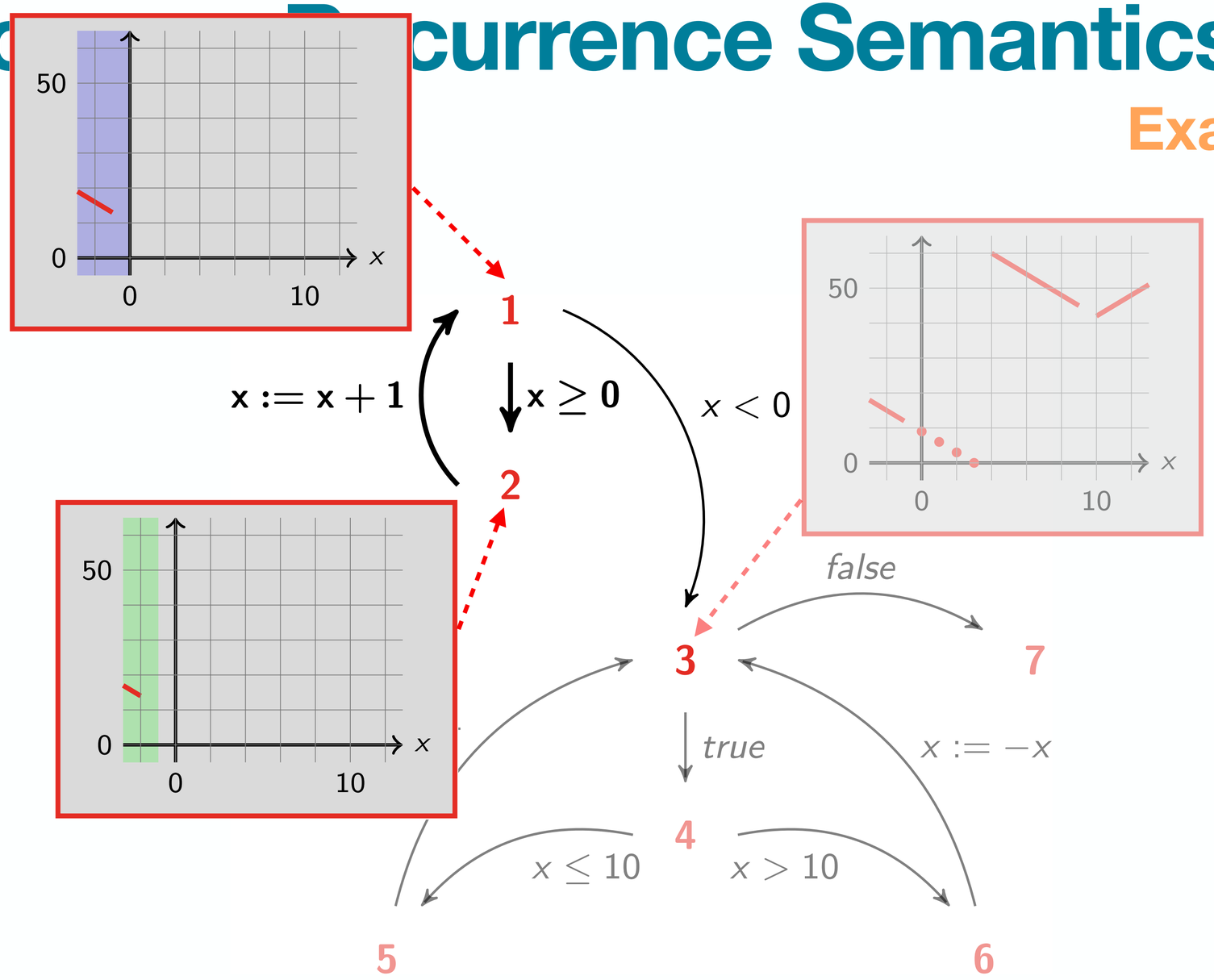
Property

$AG\,AF\,(x = 3)$

Abstract Pro currence Semantics

Example

```
while [1](x ≥ 0) do
    [2]x ← x + 1
done
while [3](0 ≥ 0) do
    if [4](x ≤ 10) do
        [5]x ← x + 1
    else
        [6]x ← -x
done[7]
```

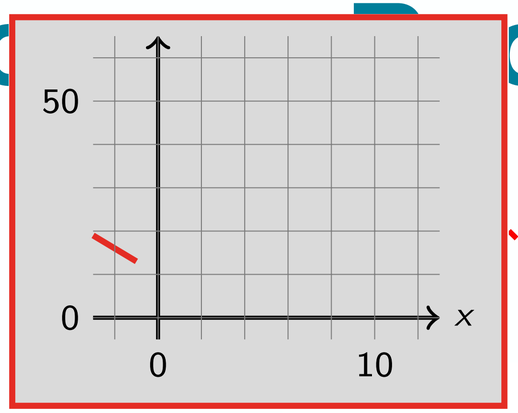


Property
AG AF $(x = 3)$

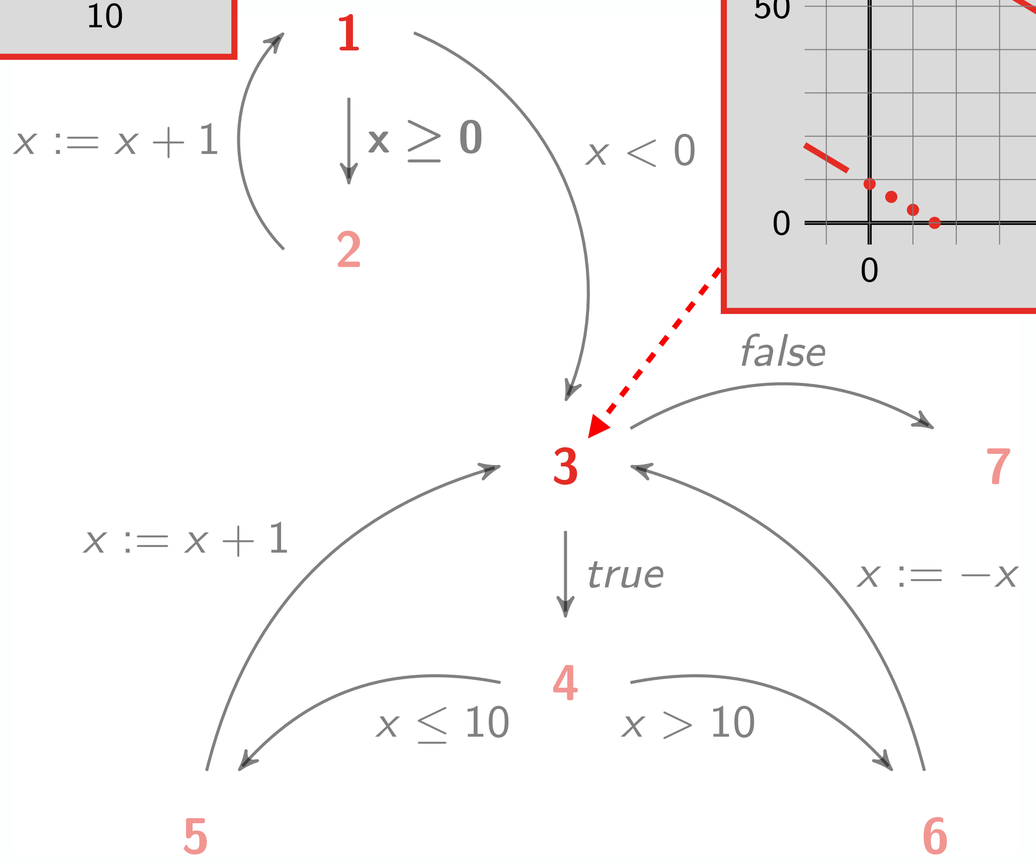
Abstract Pro currence Semantics

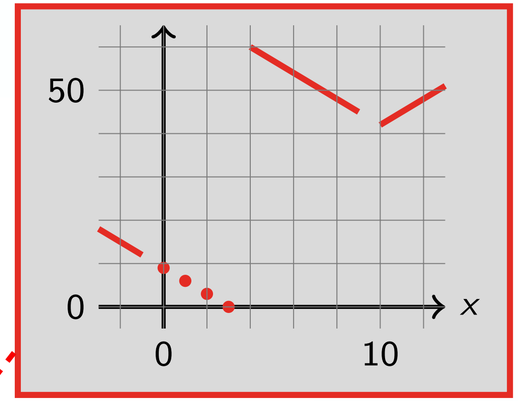
Example

```
while [1](x ≥ 0) do
    [2]x ← x + 1
done
while [3](0 ≥ 0) do
    if [4](x ≤ 10) do
        [5]x ← x + 1
    else
        [6]x ← -x
done[7]
```



the analysis gives $x < 0$ as **sufficient precondition**





Property
AG AF $(x = 3)$

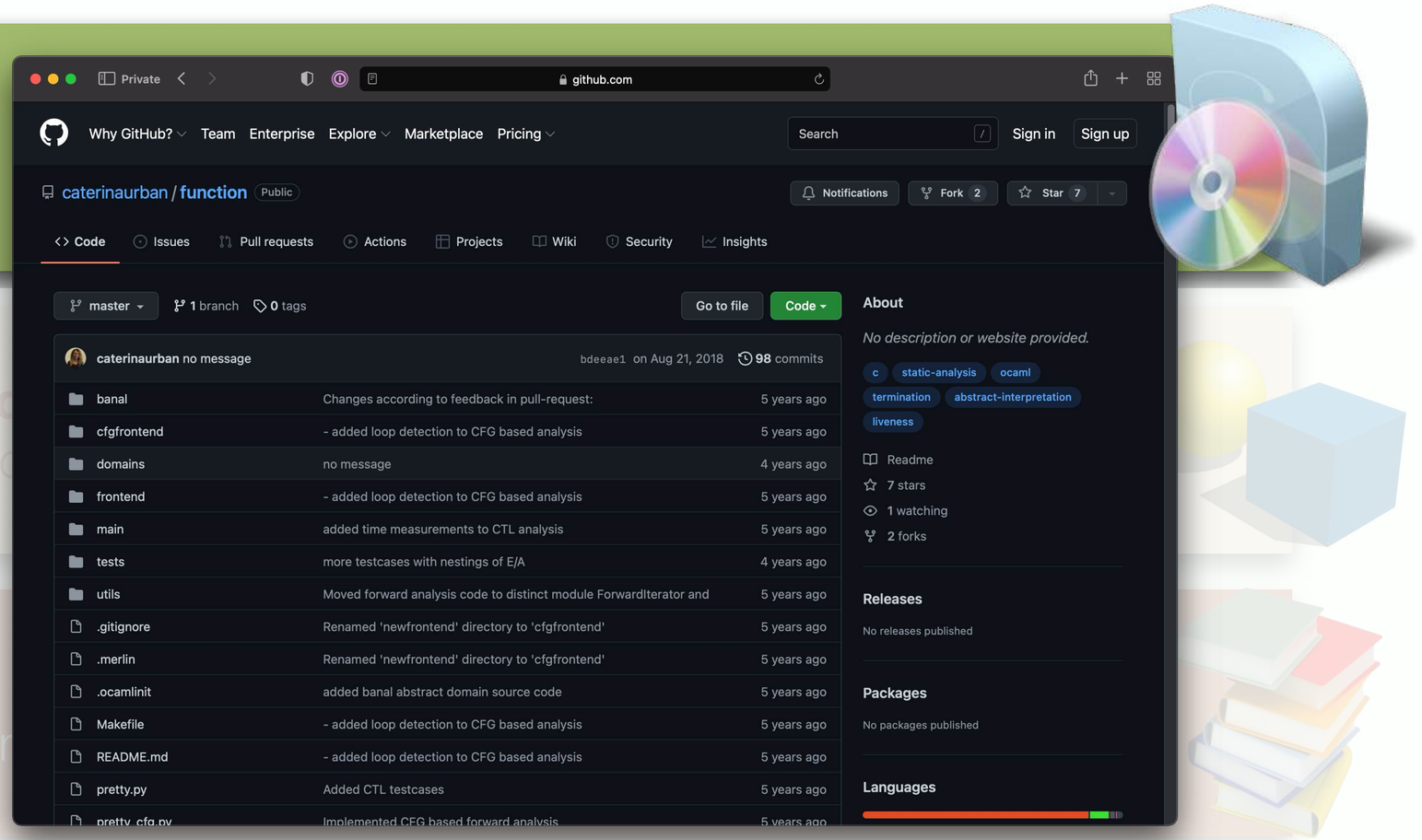
Static Recurrence Analysis

FuncTion

practical tools
targeting specific programs

abstract semantics, abstract
algorithmic approaches to de

concrete semantics
mathematical models of the pr



CTL Properties

Reading Suggestion

Abstract Interpretation of CTL Properties

Caterina Urban, Samuel Ueltschi, and Peter Müller

Department of Computer Science
ETH Zurich, Switzerland

Abstract. CTL is a temporal logic commonly used to express program properties. Most of the existing approaches for proving CTL properties only support certain classes of programs, limit their scope to a subset of CTL, or do not directly support certain existential CTL formulas. This paper presents an abstract interpretation framework for proving CTL properties that does not suffer from these limitations. Our approach automatically infers sufficient preconditions, and thus provides useful information even when a program satisfies a property only for some inputs.
We systematically derive a program semantics that precisely captures CTL properties by abstraction of the operational trace semantics of a program. We then leverage existing abstract domains based on piecewise-defined functions to derive decidable abstractions that are suitable for static program analysis. To handle existential CTL properties, we augment these abstract domains with under-approximating operators.
We implemented our approach in a prototype static analyzer. Our experimental evaluation demonstrates that the analysis is effective, even for CTL formulas with non-trivial nesting of universal and existential path quantifiers, and performs well on a wide variety of benchmarks.

Static Analysis of HyperLiveness Properties

The Art of Losing Precision

No Surprises, Please

What Could
Possibly Go Right?

It’s Complicated

Program Properties

$P \in \mathscr{P}(\mathscr{P}(\Sigma^\infty))$

```
1
a ← [0, +∞]
2
b ← [0, +∞]
3
q ← 0
4
r ← a
5
while 6(r ≥ b) do
    7
    r ← r − b
    8
    q ← q + 1
    9
done
10
```

Example

- Determinism: $P \stackrel{\text{def}}{=} \{\{\sigma\} \mid \sigma \in \Sigma^\infty\}$

Program Property Verification

$\mathscr{M} \in P \Leftrightarrow \underbrace{\{\mathscr{M}\}}_{\text{Collecting Semantics}} \subseteq P$

Which Non-Termination Alarm is Worse?

```
function f(x) {

  1...
  2z ← 10
  3if ( ... ) then
      while 4(z ≥ 0) do
        5z ← z - x
      done6
  else
      while 7(z ≥ x) do
        8c ←[-2, 1]
        9z ←z + c
      done10
  end

}11
```

 ⟵ diverges when x = 0

 ⟵ diverges when c ≥ 0

Which Non-Termination Alarm is Worse?

Robust Non-Termination

```
function f(x) {

  1...
  2z ← 10
  3if ( ... ) then
      while 4(z ≥ 0) do
        5z ← z - x
      done6
  else
      while 7(z ≥ x) do
        8c ←[-2, 1]
        9z ←z + c
      done10
  end

}11
```

🚨 ⟵ diverges when x ≤ 0

🚨 ⟵ diverges when c ≥ 0

Robust Non-Termination

∃ Input ∀ Non-Deterministic Choices : Program Diverges

```
function f(x){                                  demonic non-determinism

  1...
  2z ← 10
  3if ( ... ) then
      while 4(z ≥ 0) do                  🚨 ←— diverges when x ≤ 0
        5z ← z - x
      done6
  else
      while 7(z ≥ x) do
        8c ←[-2, 1]
        9z ←z + c
      done10
  end

}11
```

Termination Resilience

∀ Inputs ∃ Non-Deterministic Choice : Program Terminates

```
function f(x) {

  1...
  2z ← 10
  3if ( ... ) then
      while 4(z ≥ 0) do
        5z ← z - x
      done6
  else
      while 7(z ≥ x) do
        8c ←[-2, 1]
        9z ←z + c
      done10
  end

}11
```

👌 ⟵ terminates when **c < 0**, **independently of the value of x**

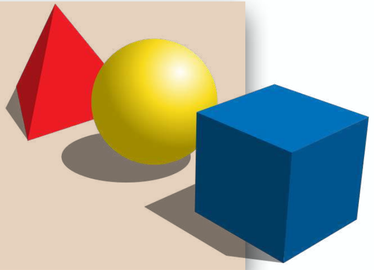
angelic non-determinism

Static Termination Resilience Analysis

3-Step Recipe

practical tools
targeting specific programs

abstract semantics, abstract domains
algorithmic approaches to decide program properties



concrete semantics
mathematical models of the program behavior

Static Termination Resilience Analysis

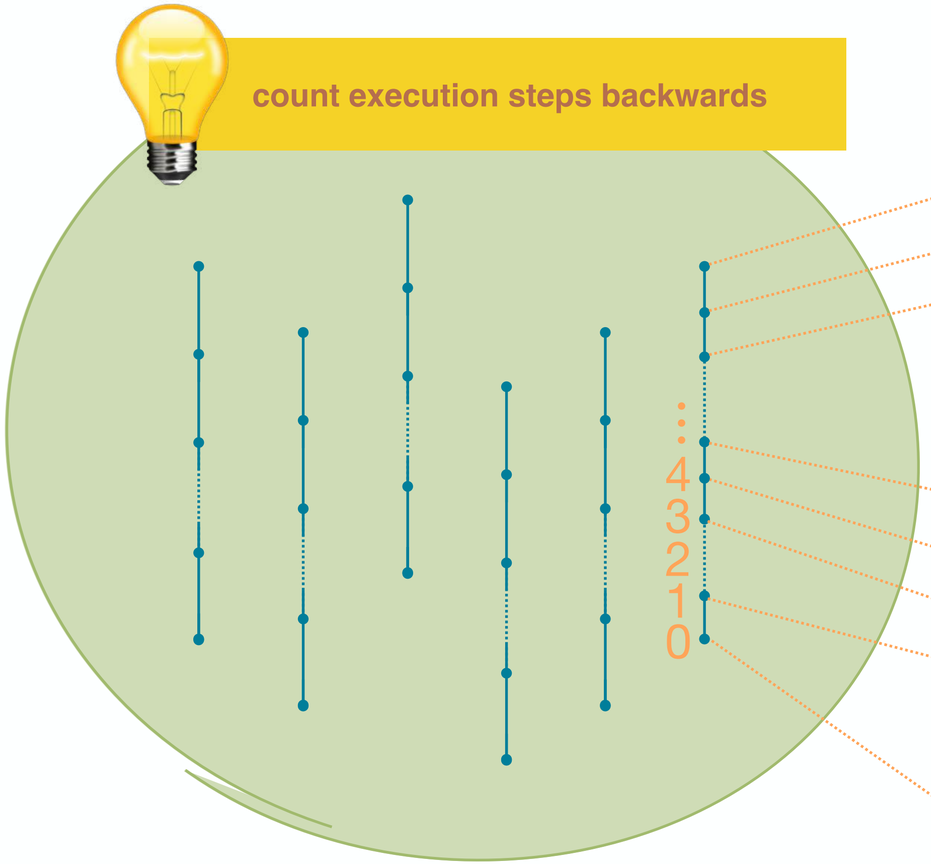
3-Step Recipe

practical tools
targeting specific programs

abstract semantics, abstract domains
algorithmic approaches to decide program properties

concrete semantics
mathematical models of the program behavior

Termination Resilience Semantics




```
function f(x) {

  1a ← [-∞, +∞]
  2z ← 10
  3if (a*a ≥ 0) then
      while 4(z ≥ 0) do
        5z ← z - x
      done6
  else
      while 7(z ≥ x) do
        8c ←[-2, 1]
        9z ←z + c
      done10
  end

}11
```

Termination Resilience Semantics

$$f_1 \preceq f_2 \stackrel{\text{def}}{=} \mathrm{dom}(f_1) \subseteq \mathrm{dom}(f_2) \wedge \forall x \in \mathrm{dom}(f_1) : f_1(x) \leq f_2(x)$$

$$\Theta \stackrel{\text{def}}{=} \mathrm{lfp}^{\preceq}_{\emptyset} \; \lambda f \lambda s \,.\, \begin{cases} 0 & s \in \Omega_\tau \\ \sup\{f(s') + 1 \mid \langle s, s' \rangle \in \tau\} & s \in \widetilde{\mathrm{pre}}_{\tau^{\mathsf{i}}}(\mathrm{dom}(f)) \\ \inf\{f(s') + 1 \mid \langle s, s' \rangle \in \tau\} & s \in \mathrm{pre}_{\tau^{\mathsf{r}}}(\mathrm{dom}(f)) \\ \text{undefined} & \text{otherwise} \end{cases}$$

final states

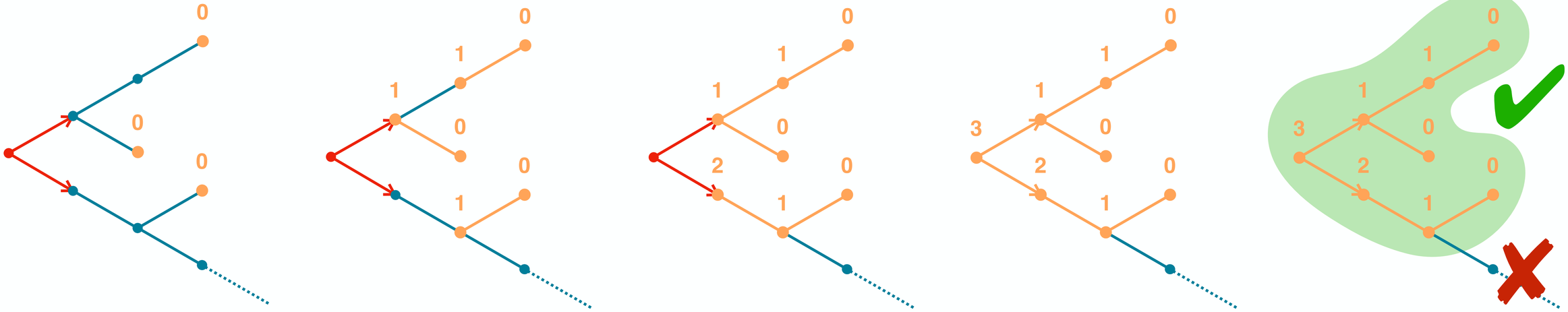
totally undefined function

$$\widetilde{\mathrm{pre}}_{\tau^{\mathsf{i}}}(X) \stackrel{\text{def}}{=} \{s \mid \forall s' \colon \langle s, s' \rangle \in \tau^{\mathsf{i}} \Rightarrow s' \in X\}$$

input transitions

$$\mathrm{pre}_{\tau^{\mathsf{r}}}(X) \stackrel{\text{def}}{=} \{s \mid \exists s' \in X \colon \langle s, s' \rangle \in \tau^{\mathsf{r}}\}$$

regular transitions

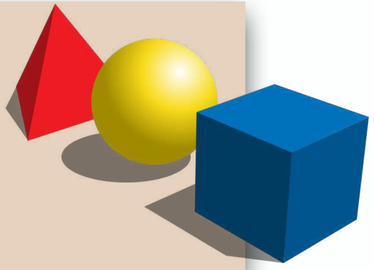


Static Termination Resilience Analysis

3-Step Recipe

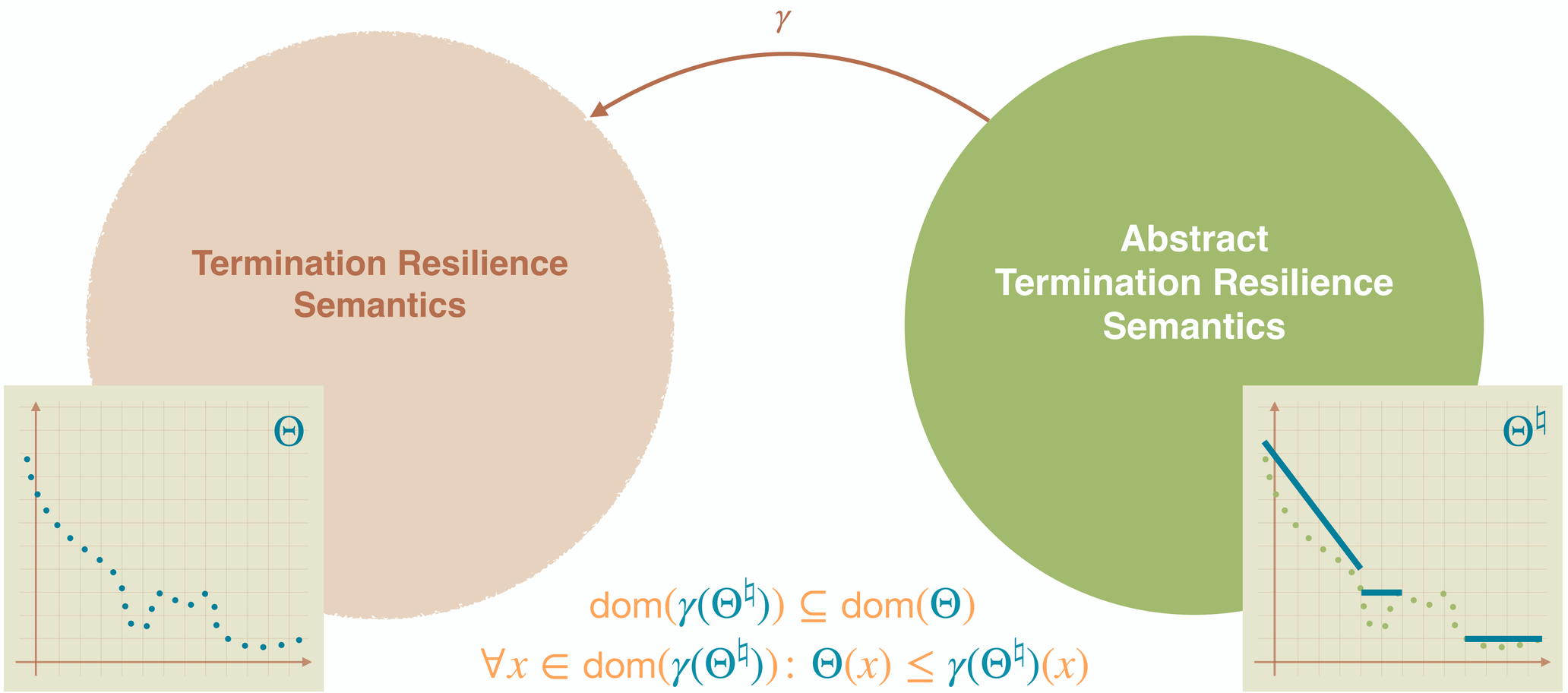
practical tools
targeting specific programs

abstract semantics, abstract domains
algorithmic approaches to decide program properties

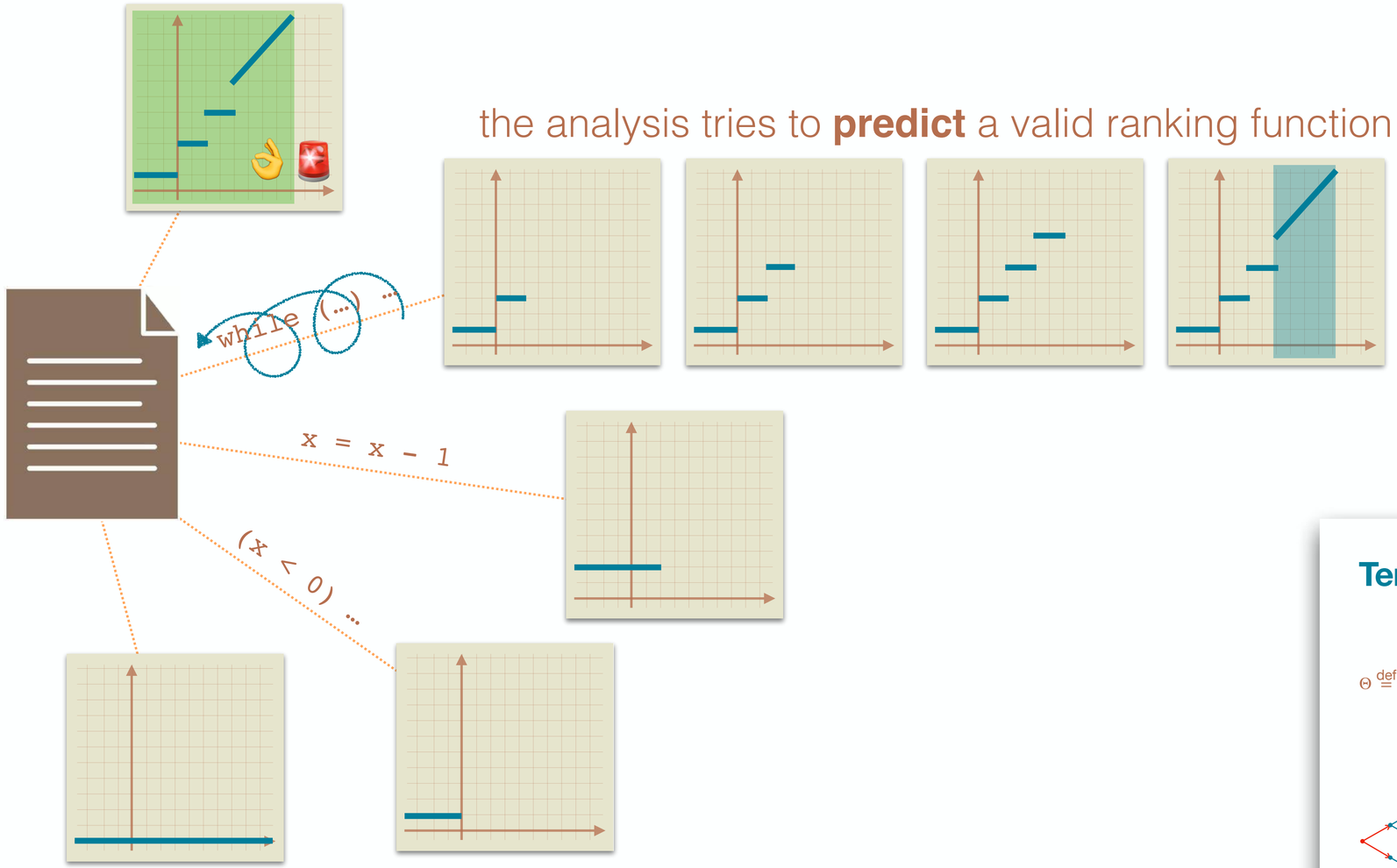


concrete semantics
mathematical models of the program behavior

Piecewise-Defined Ranking Functions



Static Termination Resilience Analysis



Termination Resilience Semantics

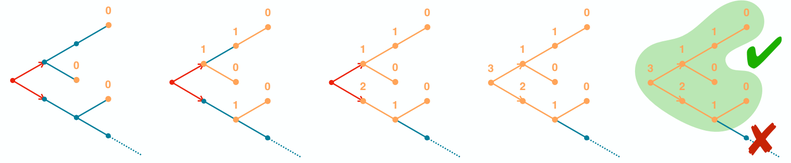
$$f_1 \sqsubseteq f_2 \overset{\text{def}}{=} \mathrm{dom}(f_1) \subseteq \mathrm{dom}(f_2) \wedge \forall x \in \mathrm{dom}(f_1) \colon f_1(x) \leq f_2(x)$$

$$\Theta \overset{\text{def}}{=} \mathrm{lfp}^{\sqsubseteq}_{\emptyset}\ \lambda f \lambda s \,.\, \begin{cases} 0 & \text{final states} \quad s \in \Omega_\tau \\ \sup\{f(s') + 1 \mid \langle s, s' \rangle \in \tau\} & s \in \widetilde{\mathrm{pre}}_\tau(\mathrm{dom}(f)) \\ \inf\{f(s') + 1 \mid \langle s, s' \rangle \in \tau\} & s \in \mathrm{pre}_\tau(\mathrm{dom}(f)) \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\widetilde{\mathrm{pre}}_\tau(X) \overset{\text{def}}{=} \{s \mid \forall s' \colon \langle s, s' \rangle \in \tau^{\mathrm{i}} \Rightarrow s' \in X\} \quad \text{input transitions}$$

$$\mathrm{pre}_\tau(X) \overset{\text{def}}{=} \{s \mid \exists s' \in X \colon \langle s, s' \rangle \in \tau^{\mathrm{r}}\} \quad \text{regular transitions}$$

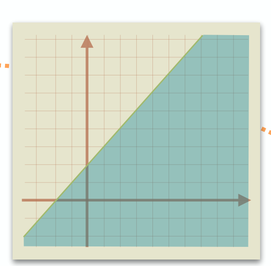
totally undefined function

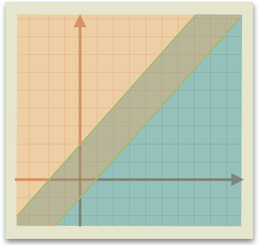


Static Termination Resilience Analysis

Non-Deterministic Variable Assignments

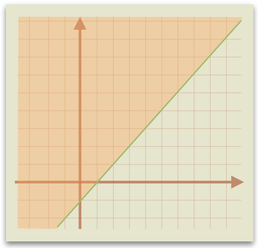
```
function f(x) {

  1a ← [-∞, +∞]
  2z ← 10
  3if (a*a ≥ 0) then
      while 4(z ≥ 0) do
        5z ← z - x
      done6
  else
      while 7(z ≥ x) do
        8c ←[-2, 1]
        9z ←z + c
      done10
  end

}11
```





z + [-2,1] - x ≥ 0
z - x ≥ [-1,2]
z - x ≥ -1

z + [-2,1] - x < 0
z - x < [-1,2]
z - x < 2



Termination Resilience Semantics

$$f_1 \sqsubseteq f_2 \stackrel{\text{def}}{=} \text{dom}(f_1) \subseteq \text{dom}(f_2) \wedge \forall x \in \text{dom}(f_1) : f_1(x) \leq f_2(x)$$

$$\Theta \stackrel{\text{def}}{=} \text{lfp}^{\sqsubseteq}_{\varnothing}\ \lambda f \lambda s.\ \begin{cases} 0 & s \in \Omega_\tau \quad \text{(final states)} \\ \inf\{f(s') + 1 \mid \langle s, s' \rangle \in \tau\} & s \in \text{pre}_\tau(\text{dom}(f)) \end{cases}$$

(regular transitions)

$$\text{pre}_\tau(X) \stackrel{\text{def}}{=} \{s \mid \exists s' \in X : \langle s, s' \rangle \in \tau\}$$

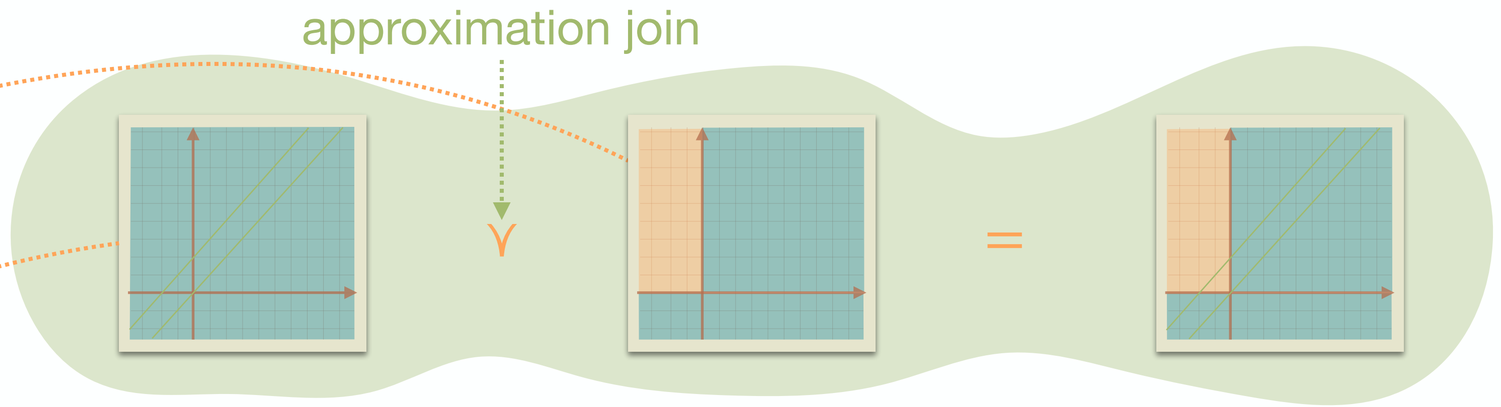
totally undefined function

13

Static Termination Resilience Analysis

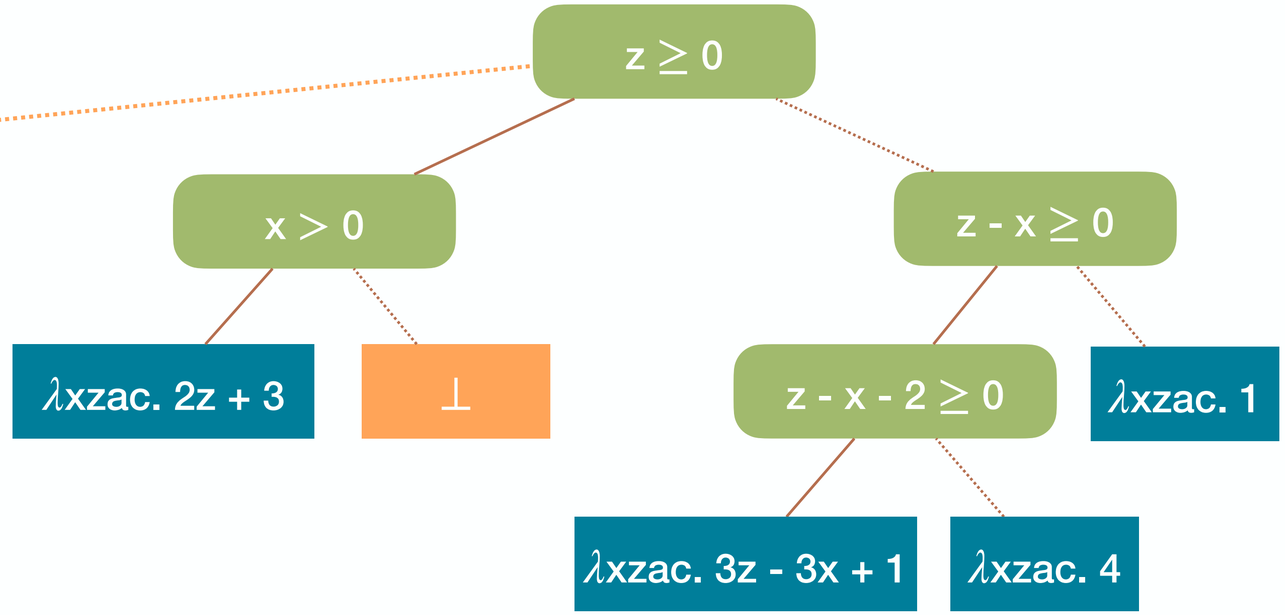
Approximation Join or Resilience Join?

```
function f(x) {
  1a ← [-∞, +∞]
  2z ← 10
  3if (a*a ≥ 0) then
     while 4(z ≥ 0) do
       5z ← z - x
     done6
  else
     while 7(z ≥ x) do
       8c ←[-2, 1]
       9z ←z + c
     done10
  fi
}11
```



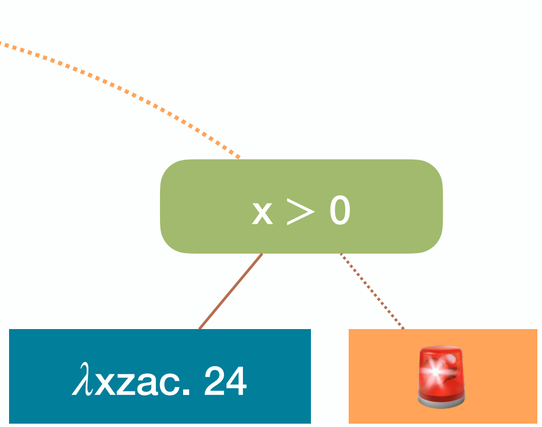
Static Termination Resilience Analysis

```
function f(x) {
  [1]a ← [-∞, +∞]
  [2]z ← 10
  [3]if (a*a ≥ 0) then
      while [4](z ≥ 0) do
        [5]z ← z - x
      done[6]
  else
      while [7](z ≥ x) do
        [8]c ←[-2, 1]
        [9]z ←z + c
      done[10]
  end
}[11]
```



Static Termination Resilience Analysis

```
function f(x) {
  1a ← [-∞, +∞]
  2z ← 10
  3if (a*a ≥ 0) then
      while 4(z ≥ 0) do
        5z ← z - x
      od6
  else
      while 7(z ≥ x) do
        8c ←[-2, 1]
        9z ←z + c
      od10
  fi

}11
```

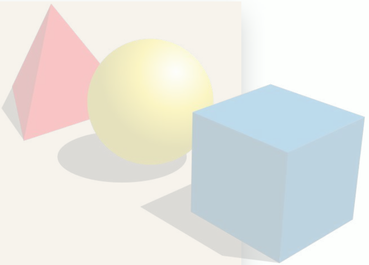


Static Termination Resilience Analysis

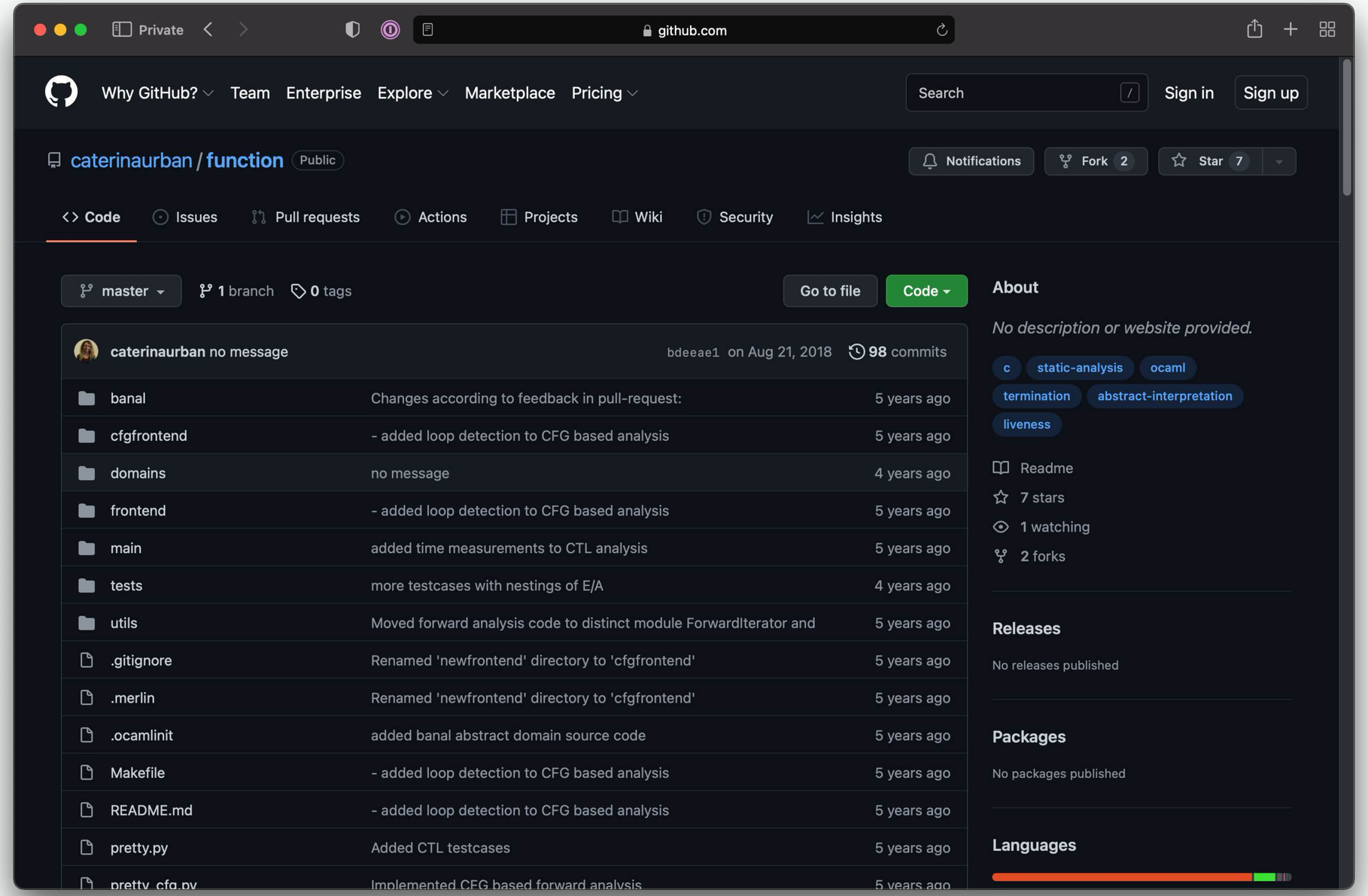
3-Step Recipe

practical tools
targeting specific programs

abstract semantics, abstract domains
algorithmic approaches to decide program properties



concrete semantics
mathematical models of the program behavior

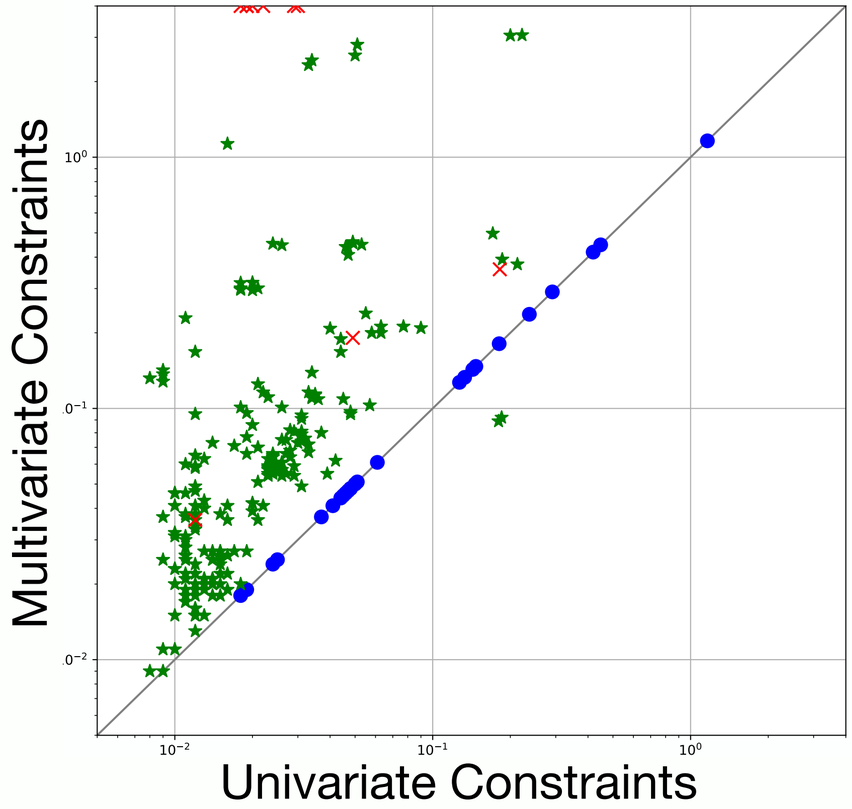

Private
github.com
Why GitHub?
Team
Enterprise
Explore
Marketplace
Pricing
Search
Sign in
Sign up
caterinaurban / function
Public
Notifications
Fork 2
Star 7
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
master
1 branch
0 tags
Go to file
Code
caterinaurban no message
bdeeae1 on Aug 21, 2018
98 commits
banal Changes according to feedback in pull-request: 5 years ago
cfgfrontend - added loop detection to CFG based analysis 5 years ago
domains no message 4 years ago
frontend - added loop detection to CFG based analysis 5 years ago
main added time measurements to CTL analysis 5 years ago
tests more testcases with nestings of E/A 4 years ago
utils Moved forward analysis code to distinct module ForwardIterator and 5 years ago
.gitignore Renamed 'newfrontend' directory to 'cfgfrontend' 5 years ago
.merlin Renamed 'newfrontend' directory to 'cfgfrontend' 5 years ago
.ocamlinit added banal abstract domain source code 5 years ago
Makefile - added loop detection to CFG based analysis 5 years ago
README.md - added loop detection to CFG based analysis 5 years ago
pretty.py Added CTL testcases 5 years ago
pretty_cfg.py Implemented CFG based forward analysis 5 years ago
About
No description or website provided.
c
static-analysis
ocaml
termination
abstract-interpretation
liveness
Readme
7 stars
1 watching
2 forks
Releases
No releases published
Packages
No packages published
Languages

Experimental Evaluation

	Benchmark	Property	Verified	Alarms	TO	Time
Univariate Constraints	SV-COMP 2024	Termination	0	119	0	3.5s
		Termination Resilience	61	58	0	3.6s
	Raad et al @ OOPSLA 2024	Termination	0	36	0	0.5s
		Termination Resilience	16	20	0	0.5s
	Shi et al. @ FSE 2022	Termination	0	85	0	2.0s
		Termination Resilience	57	28	0	2.2s
Multivariate Constraints	Benchmark	Property	Verified	Alarms	TO	Time
	SV-COMP 2024	Termination	0	119	0	7.2s
		Termination Resilience	76	43	0	16.9s
	Raad et al @ OOPSLA 2024	Termination	0	36	0	7.2s
		Termination Resilience	16	20	0	16.9s
	Shi et al. @ FSE 2022	Termination	0	85	0	69s
		Termination Resilience	49	28	8	500s

Experimental Evaluation

Univariate vs Multivariate Constraints



- ★ equal precision
- ✕ multivariate constraints are more precise
- ● univariate constraints are more precise (!)

Static Analysis by Abstract Interpretation

SOUNDNESS

€ 10 +
€ 40 +
€ 30 +
€ 10

€ 90

PROPERTY OF INTEREST

COMPLETENESS

€ 9.95 +
€ 35.85 +
€ 24.95 +
€ 4.85

€ 75.60

FALSE ALARM

THANKS!