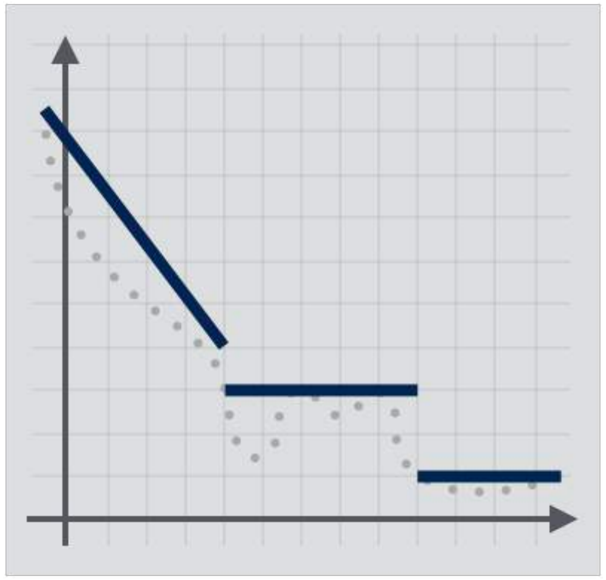
Termination Analysis

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

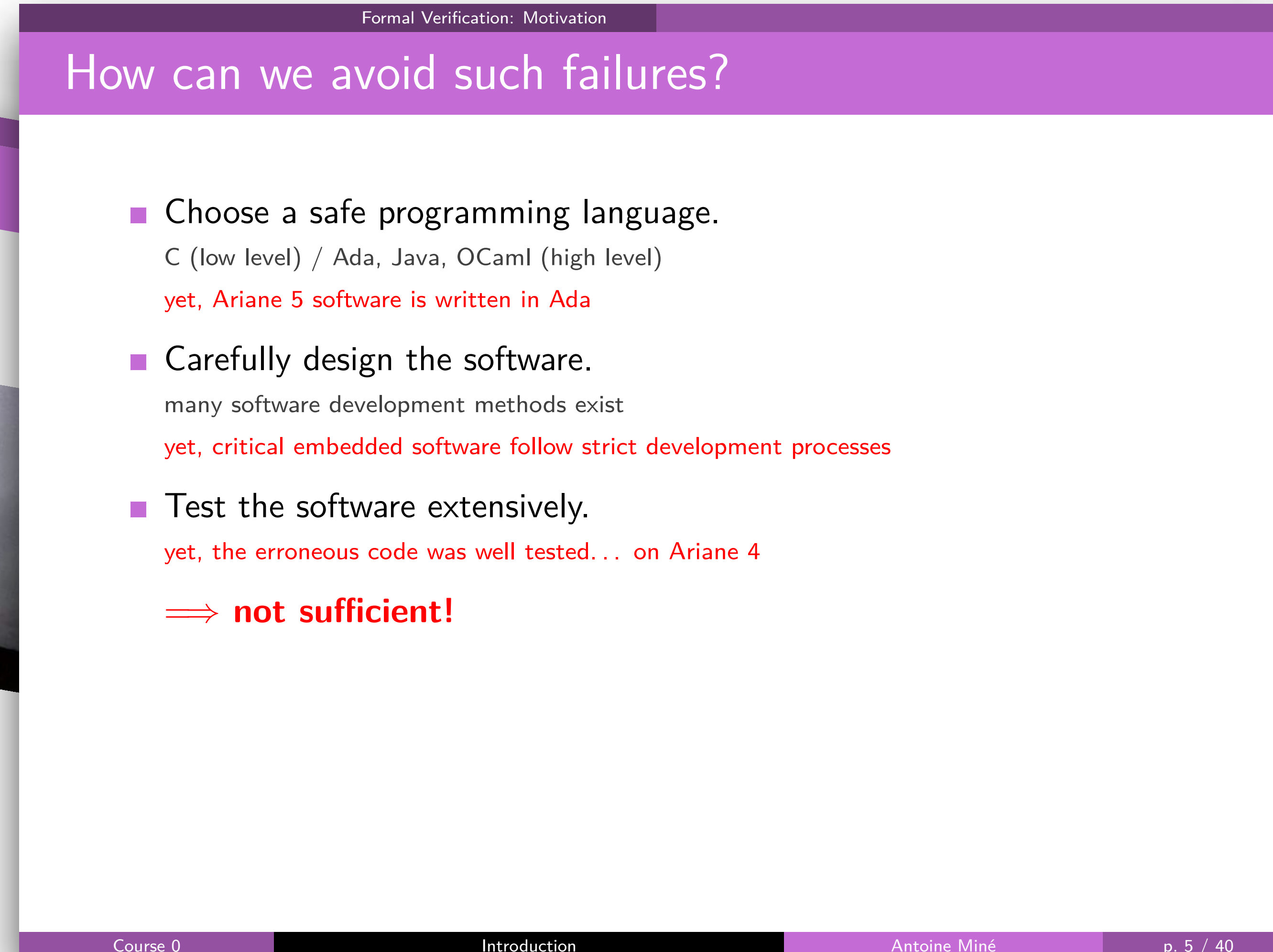


Caterina Urban

November 16th, 2023

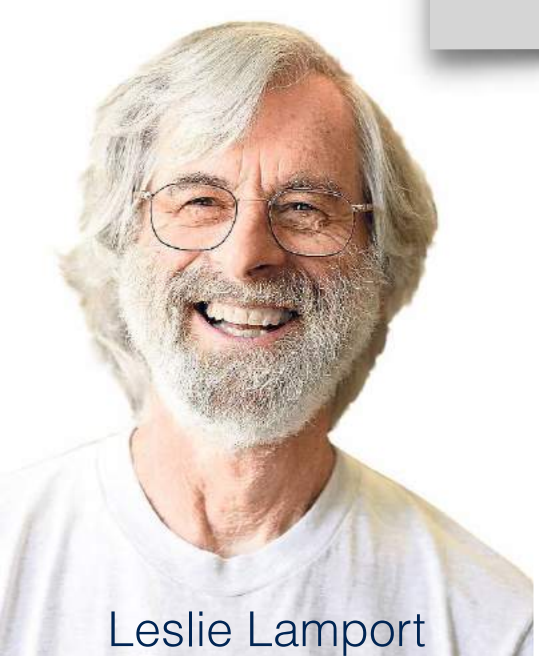
Year 2023-2024

So far, we have focused on **using static analysis to avoid** **software failures**



that is, for **proving** **Safety Properties**

Safety vs Liveness Properties


Leslie Lamport

Liveness Properties

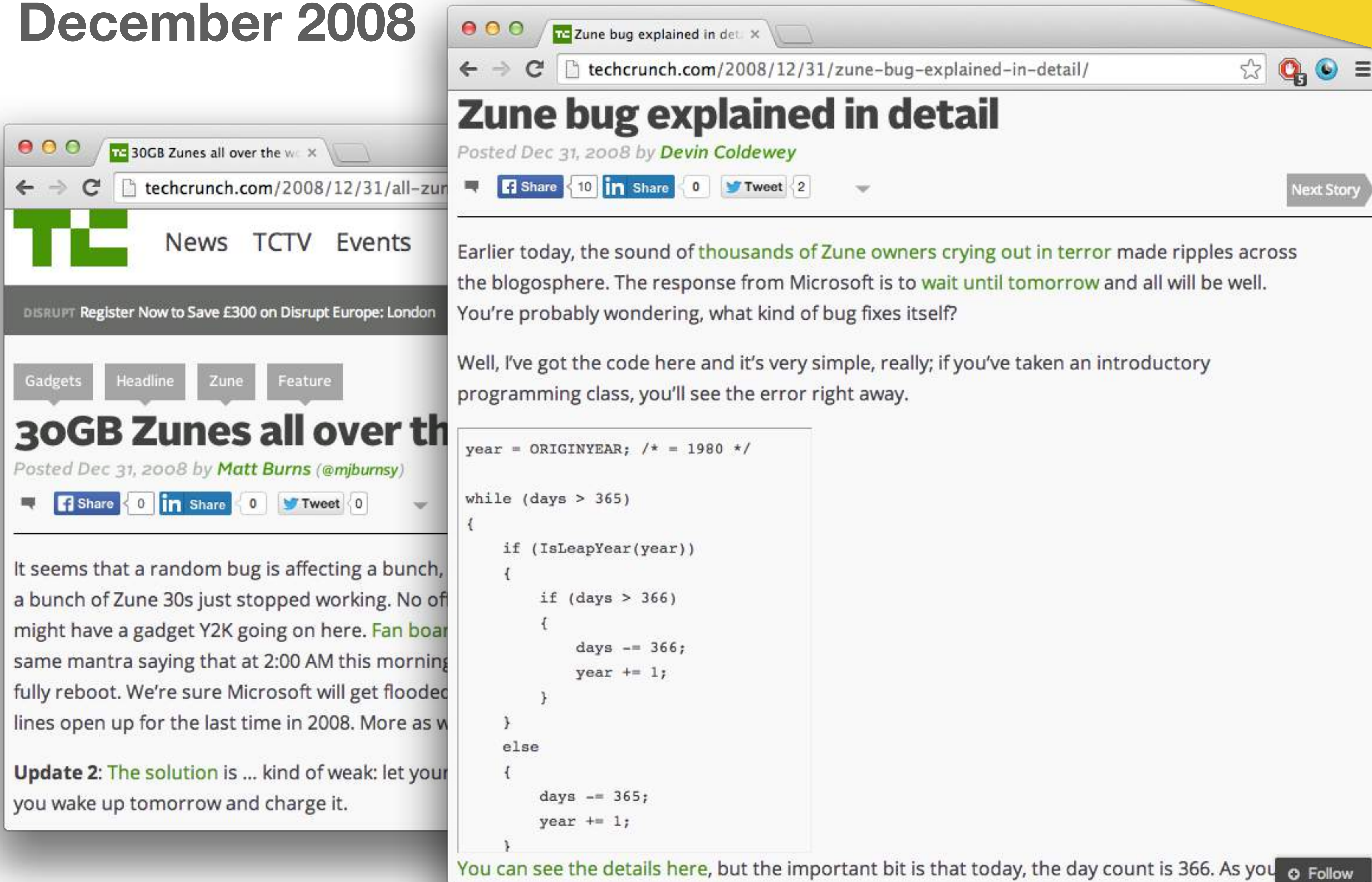
- **Guarantee Properties**
  "something good eventually happens <u>at least once</u>"
  - Example: Program Termination
- **Recurrence Properties**
  "something good eventually happens <u>infinitely often</u>"
  - Example: Starvation Freedom

Program Termination

The Zune Bug

31 December 2008

unresponsive systems



30GB Zunes all over the w...

techcrunch.com/2008/12/31/all-zun...

News TCTV Events

DISRUPT Register Now to Save £300 on Disrupt Europe: London

Gadgets Headline Zune Feature

30GB Zunes all over th...

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

Share 0 Share 0 Tweet 0

It seems that a random bug is affecting a bunch, ...
a bunch of Zune 30s just stopped working. No of...
might have a gadget Y2K going on here. Fan boar...
same mantra saying that at 2:00 AM this morning...
fully reboot. We're sure Microsoft will get flooded...
lines open up for the last time in 2008. More as w...

Update 2: The solution is ... kind of weak: let your...
you wake up tomorrow and charge it.

Zune bug explained in det...

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

Zune bug explained in detail

Posted Dec 31, 2008 by Devin Coldewey

Share 10 Share 0 Tweet 2 Next Story

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

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

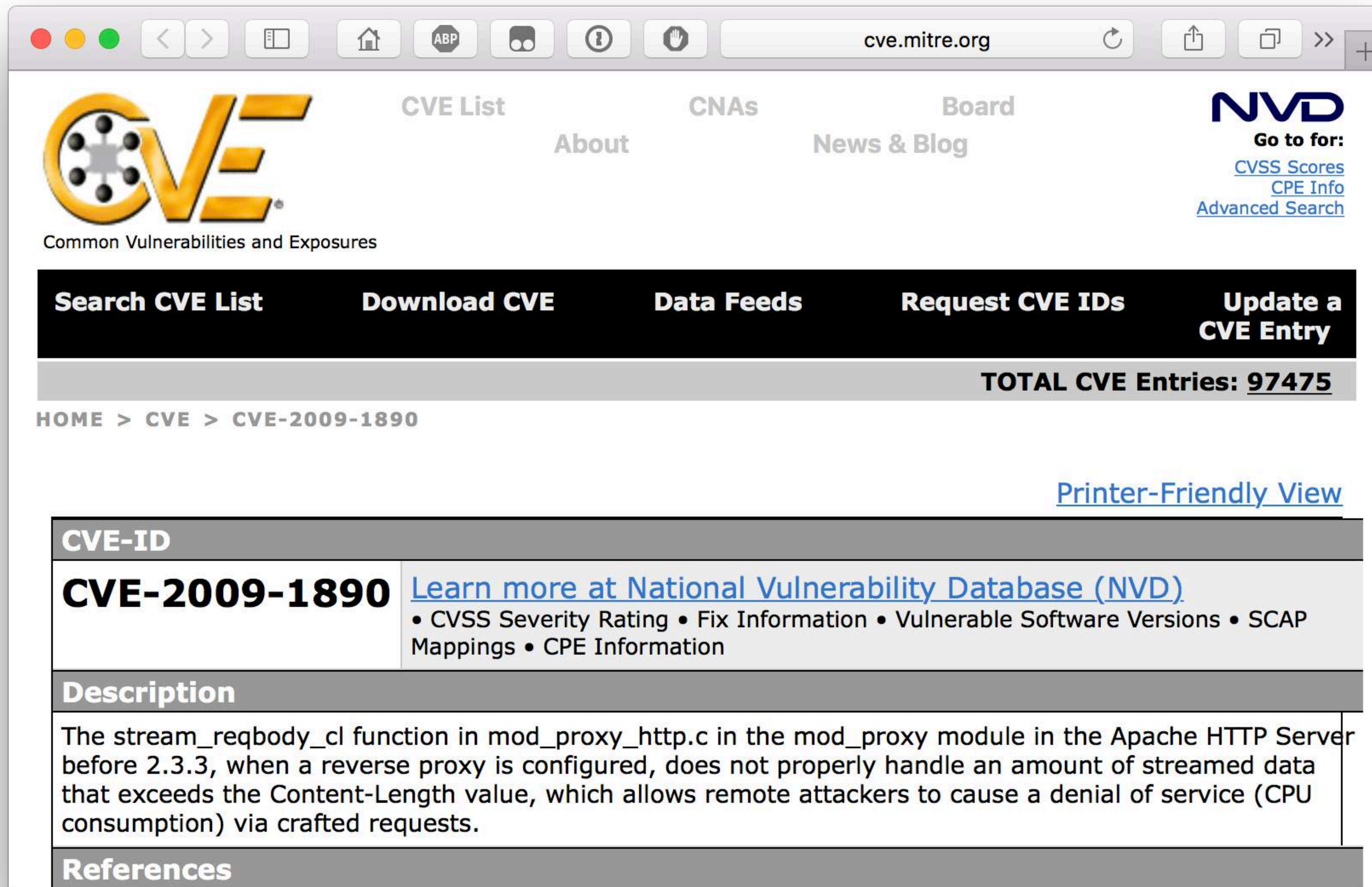
```
year = ORIGINYEAR; /* = 1980 */

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

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

Apache HTTP Server

Versions <2.3.3

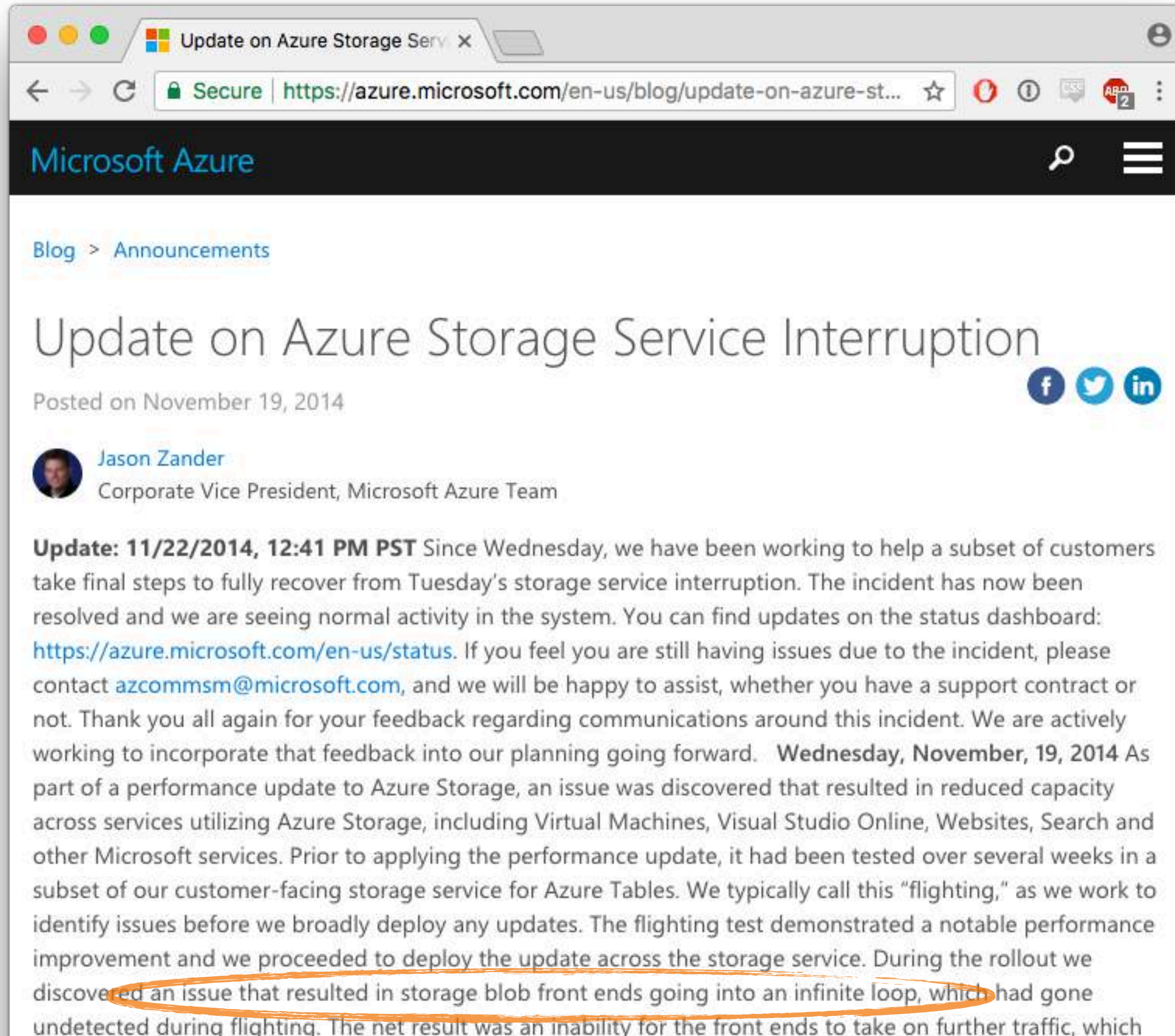
denial-of-service attacks



Azure Storage Service

19 November 2014

service interruptions



Update on Azure Storage Serv ×

Secure | https://azure.microsoft.com/en-us/blog/update-on-azure-st...

Microsoft Azure

Blog > Announcements

Update on Azure Storage Service Interruption

Posted on November 19, 2014

Jason Zander
Corporate Vice President, Microsoft Azure Team

Update: 11/22/2014, 12:41 PM PST Since Wednesday, we have been working to help a subset of customers take final steps to fully recover from Tuesday's storage service interruption. The incident has now been resolved and we are seeing normal activity in the system. You can find updates on the status dashboard: https://azure.microsoft.com/en-us/status. If you feel you are still having issues due to the incident, please contact azcommsm@microsoft.com, and we will be happy to assist, whether you have a support contract or not. Thank you all again for your feedback regarding communications around this incident. We are actively working to incorporate that feedback into our planning going forward. **Wednesday, November, 19, 2014** As part of a performance update to Azure Storage, an issue was discovered that resulted in reduced capacity across services utilizing Azure Storage, including Virtual Machines, Visual Studio Online, Websites, Search and other Microsoft services. Prior to applying the performance update, it had been tested over several weeks in a subset of our customer-facing storage service for Azure Tables. We typically call this "flighting," as we work to identify issues before we broadly deploy any updates. The flighting test demonstrated a notable performance improvement and we proceeded to deploy the update across the storage service. During the rollout we discovered an issue that resulted in storage blob front ends going into an infinite loop, which had gone undetected during flighting. The net result was an inability for the front ends to take on further traffic, which

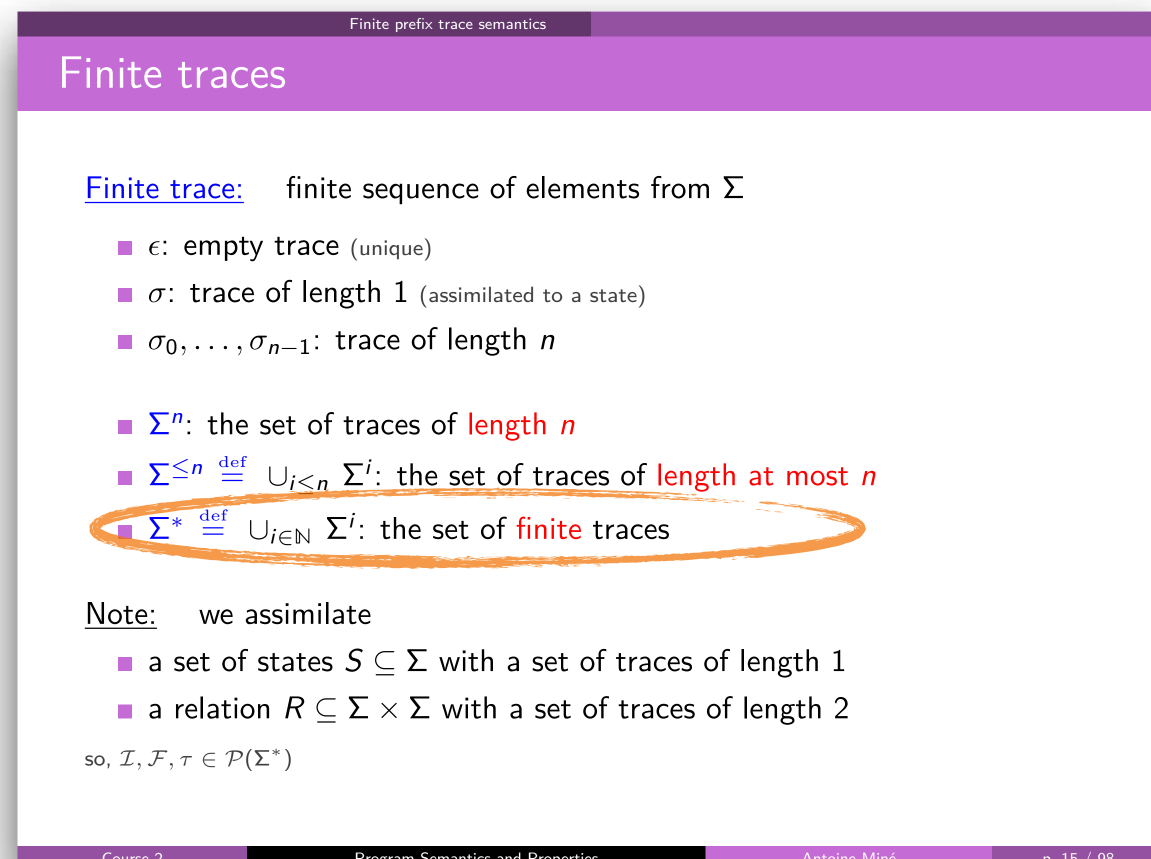
Potential and Definite Termination

Definition

A program with trace semantics $\mathscr{M} \in \mathscr{P}(\Sigma^\infty)$ **may terminate** if and only if $\mathscr{M} \cap \Sigma^* \neq \emptyset$

Definition

A program with trace semantics $\mathscr{M} \in \mathscr{P}(\Sigma^\infty)$ **must terminate** if and only if $\mathscr{M} \subseteq \Sigma^*$


Finite prefix trace semantics

Finite traces

Finite trace: finite sequence of elements from Σ

- ϵ: empty trace (unique)
- σ: trace of length 1 (assimilated to a state)
- $\sigma_0, \ldots, \sigma_{n-1}$: trace of length n

- Σ^n: the set of traces of length n
- $\Sigma^{\leq n} \stackrel{\text{def}}{=} \cup_{i<n} \Sigma^i$: the set of traces of length at most n
- $\Sigma^* \stackrel{\text{def}}{=} \cup_{i \in \mathbb{N}} \Sigma^i$: the set of finite traces

Note: we assimilate

- a set of states $S \subseteq \Sigma$ with a set of traces of length 1
- a relation $R \subseteq \Sigma \times \Sigma$ with a set of traces of length 2

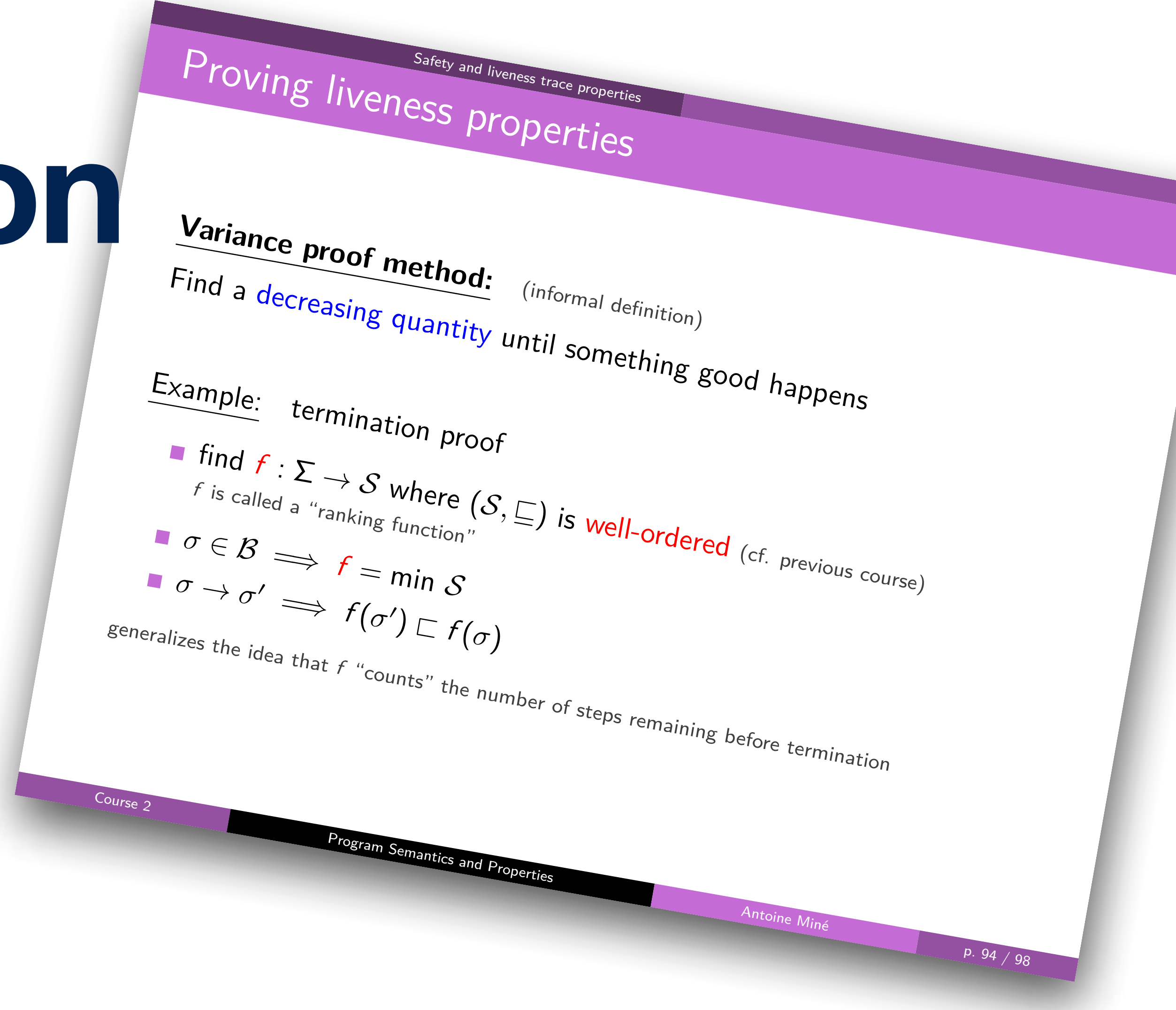
so, $\mathcal{I}, \mathcal{F}, \tau \in \mathcal{P}(\Sigma^*)$

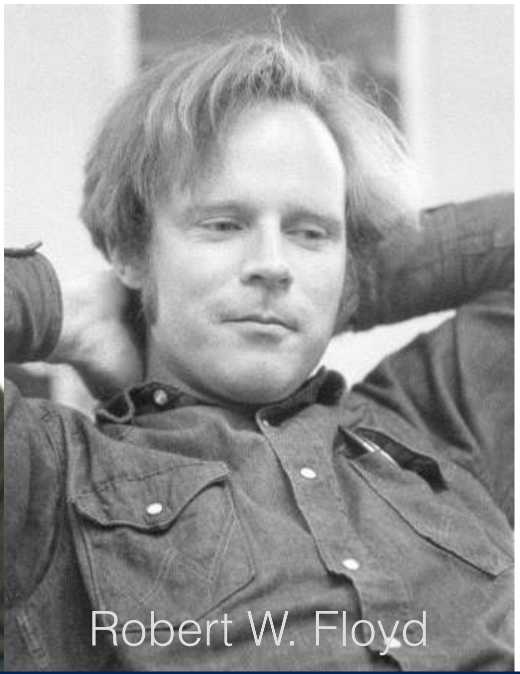
Course 2 | Program Semantics and Properties | Antoine Miné | p. 15 / 98

In *absence of non-determinism*, potential and definite termination coincide

Definite Termination

Ranking Functions





Definition

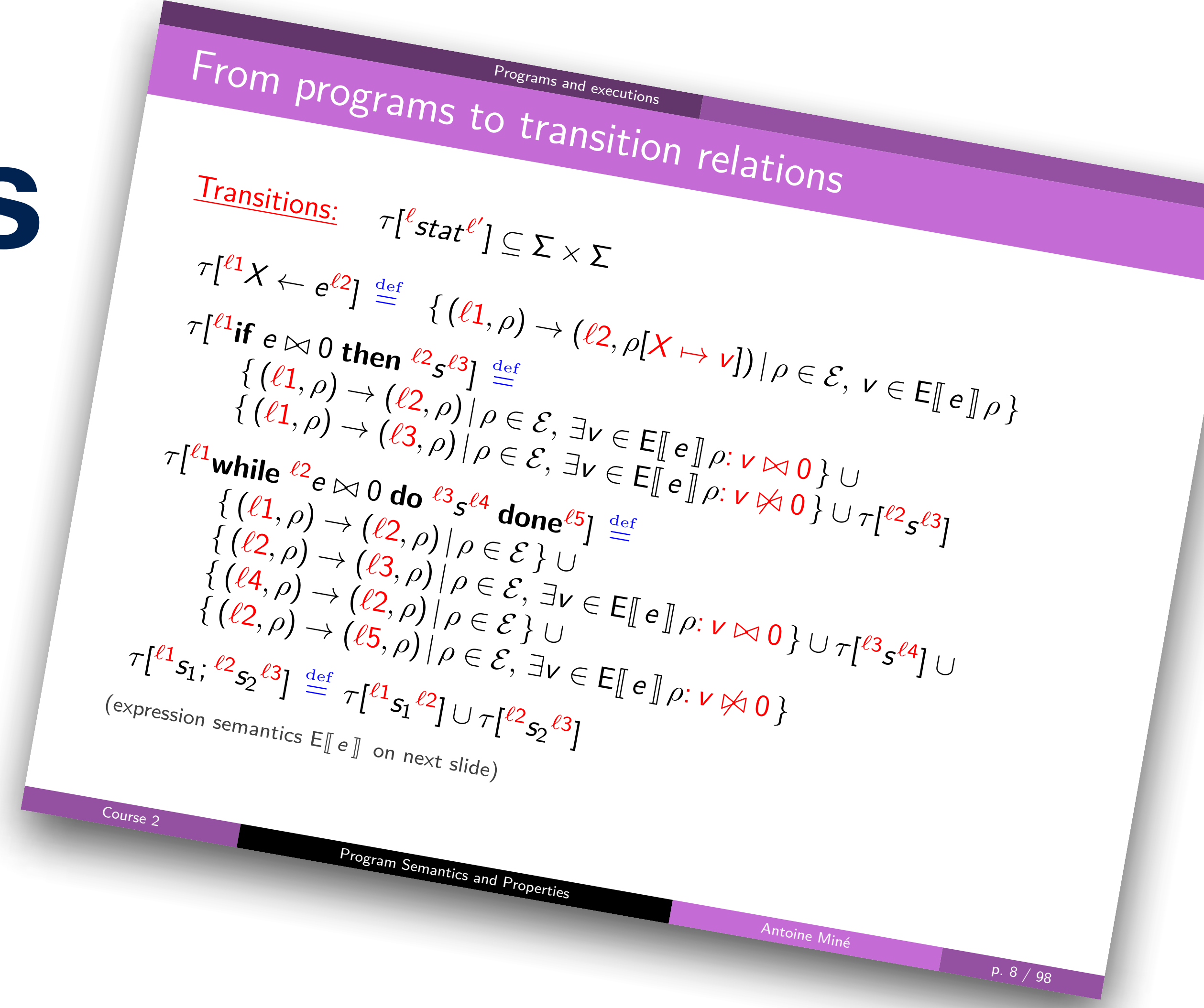
Given a transition system $\langle \Sigma, \tau \rangle$, a **ranking function** is a partial function $f \colon \Sigma \rightharpoonup \mathscr{W}$ from the set of program states Σ into a well-ordered set $\langle \mathscr{W}, \leq \rangle$ whose value *strictly decreases* through transitions between states, that is, $\forall \sigma, \sigma' \in \text{dom}(f) \colon (\sigma, \sigma') \in \tau \Rightarrow f(\sigma') < f(\sigma)$

The best known *well-ordered sets* are **naturals** $\langle \mathbb{N}, \leq \rangle$ and **ordinals** $\langle \mathbb{O}, \leq \rangle$

Ranking Functions

Example (continue)

^{1}x ← [-∞, +∞]
 while ^{2}(1 - x < 0) **do**
  ^{3}x ← x - 1
od^{4}



$$\Sigma \stackrel{\text{def}}{=} \{1,2,3,4\} \times \mathscr{E}$$

$$\tau \stackrel{\text{def}}{=} \{(1,\rho) \to (2, \rho[X \mapsto v]) \mid \rho \in \mathscr{E}, v \in \mathbb{Z}\}$$
$$\cup \{(2,\rho) \to (3,\rho) \mid \ \mid \rho \in \mathscr{E}, \exists v \in E[\![1 - x]\!]\rho : v < 0\}$$
$$\cup \{(3,\rho) \to (2, \rho[X \mapsto v]) \mid \rho \in \mathscr{E}, v \in E[\![x - 1]\!]\rho\}$$
$$\cup \{(2,\rho) \to (4,\rho) \mid \ \mid \rho \in \mathscr{E}, \exists v \in E[\![1 - x]\!]\rho : v \not< 0\}$$

Ranking Functions

Example (continue)

```
¹x ← [-∞, +∞]
 while ²(1 - x < 0) do
       ³x ← x - 1
od⁴
```

Most obvious ranking function:
a mapping $f\colon \Sigma \rightharpoonup \mathbb{O}$
from each program state
to
(a well-chosen upper bound on)
the number of steps until termination

Ranking Functions

Example (continue)

[1]x ← [-∞, +∞]
while [2](1 - x < 0) **do**
  [3]x ← x - 1
od[4]

We define the ranking function $f\colon \Sigma \rightharpoonup \mathbb{O}$ by partitioning with respect to the program control points, i.e., $f\colon \mathscr{L} \to (\mathscr{E} \rightharpoonup \mathbb{O})$

$$f(\mathbf{4}) \stackrel{\text{def}}{=} \lambda\rho.0$$

$$f(\mathbf{2}) \stackrel{\text{def}}{=} \lambda\rho\,.\begin{cases} 1 & 1-\rho(x) \nless 0 \\ 2\rho(x)-1 & 1-\rho(x) < 0 \end{cases}$$

$$f(\mathbf{3}) \stackrel{\text{def}}{=} \lambda\rho\,.\begin{cases} 2 & 2-\rho(x) \nless 0 \\ 2\rho(x)-2 & 2-\rho(x) < 0 \end{cases}$$

$$f(\mathbf{1}) \stackrel{\text{def}}{=} \lambda\rho\,.\,\omega$$


Alan Turing


Robert W. Floyd

Potential Termination

Potential Ranking Functions

For proving potential termination, we use a *weaker* notion of ranking function, which *decreases along at least one transition* during program execution

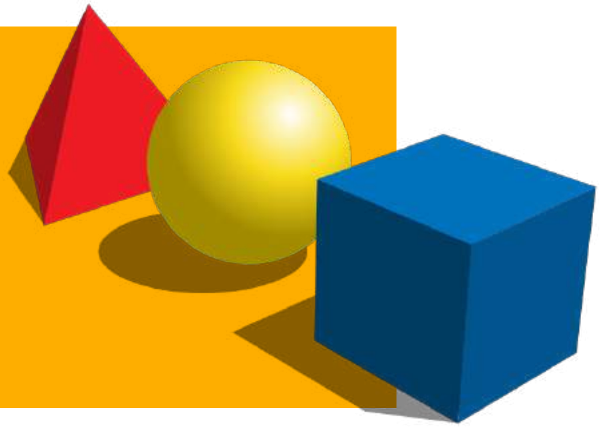
Definition

Given a transition system $\langle \Sigma, \tau \rangle$, a **potential ranking function** is a partial function $f \colon \Sigma \rightharpoonup \mathscr{W}$ from the set of states Σ into a well-ordered set $\langle \mathscr{W}, \leq \rangle$ whose value *strictly decreases* through at least one transitions from each state, that is, $\forall \sigma \in \mathrm{dom}(f) \colon (\exists \bar{\sigma} \in \mathrm{dom}(f) \colon (\sigma, \bar{\sigma}) \in \tau) \Rightarrow \exists \sigma' \in \mathrm{dom}(f) \colon (\sigma, \sigma') \in \tau \wedge f(\sigma') < f(\sigma)$

Abstract Interpretation Recipe

practical tools
targeting specific programs

algorithmic approaches
to decide program properties



mathematical models
of the program behavior

Abstract Interpretation Recipe

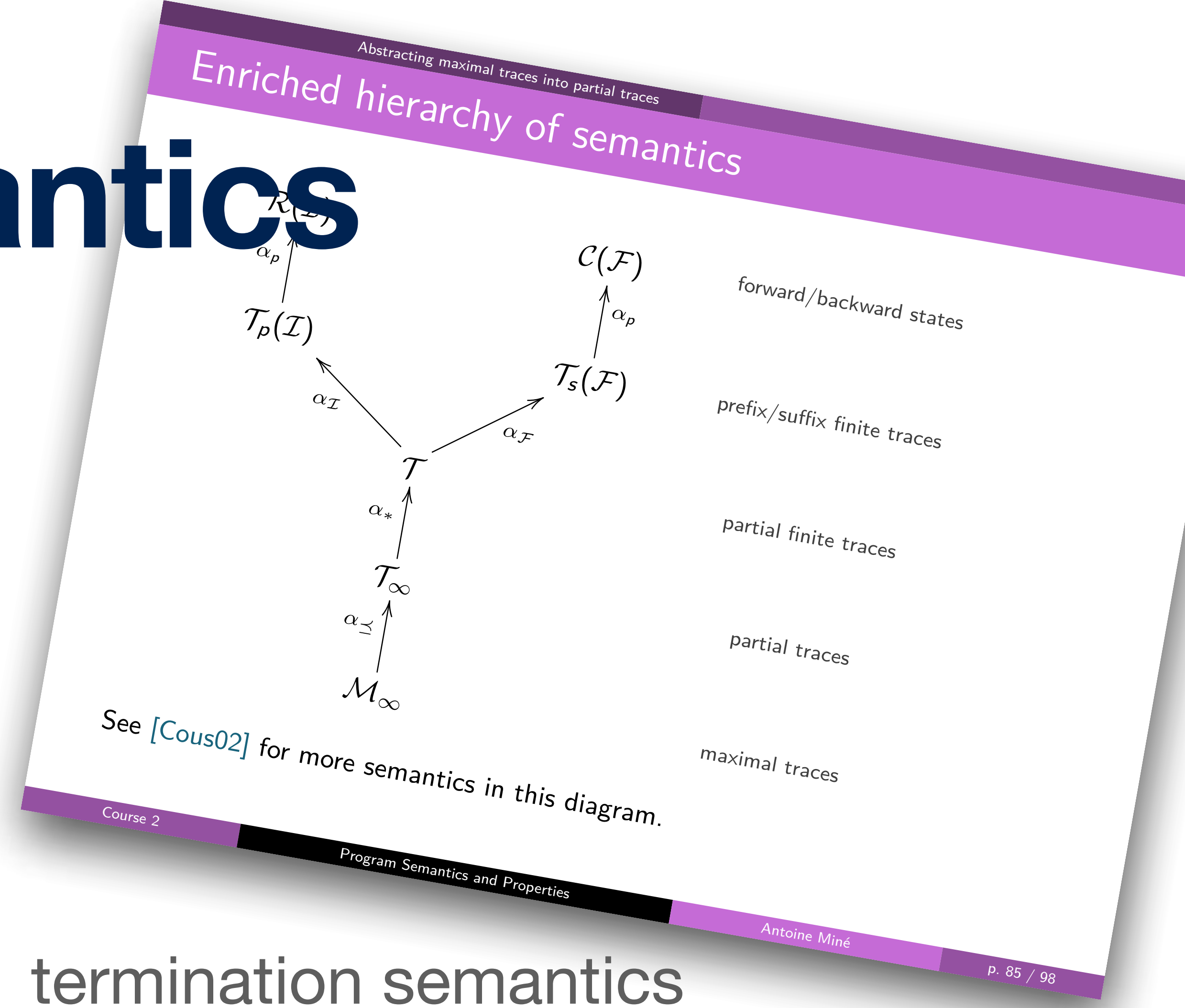
practical tools
targeting specific programs

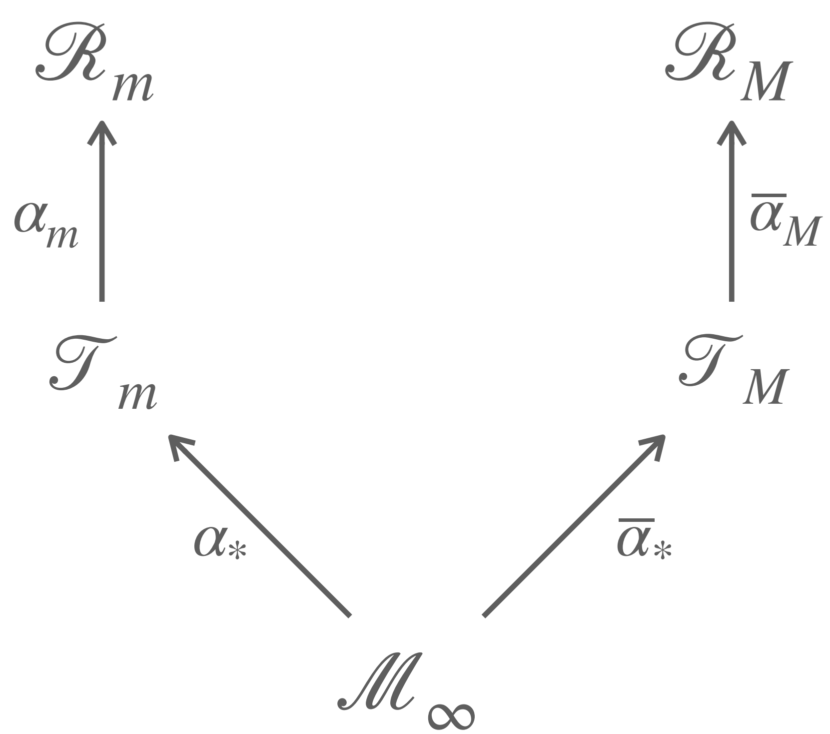
algorithmic approaches
to decide program properties

mathematical models
of the program behavior

Termination Semantics

Hierarchy of Semantics



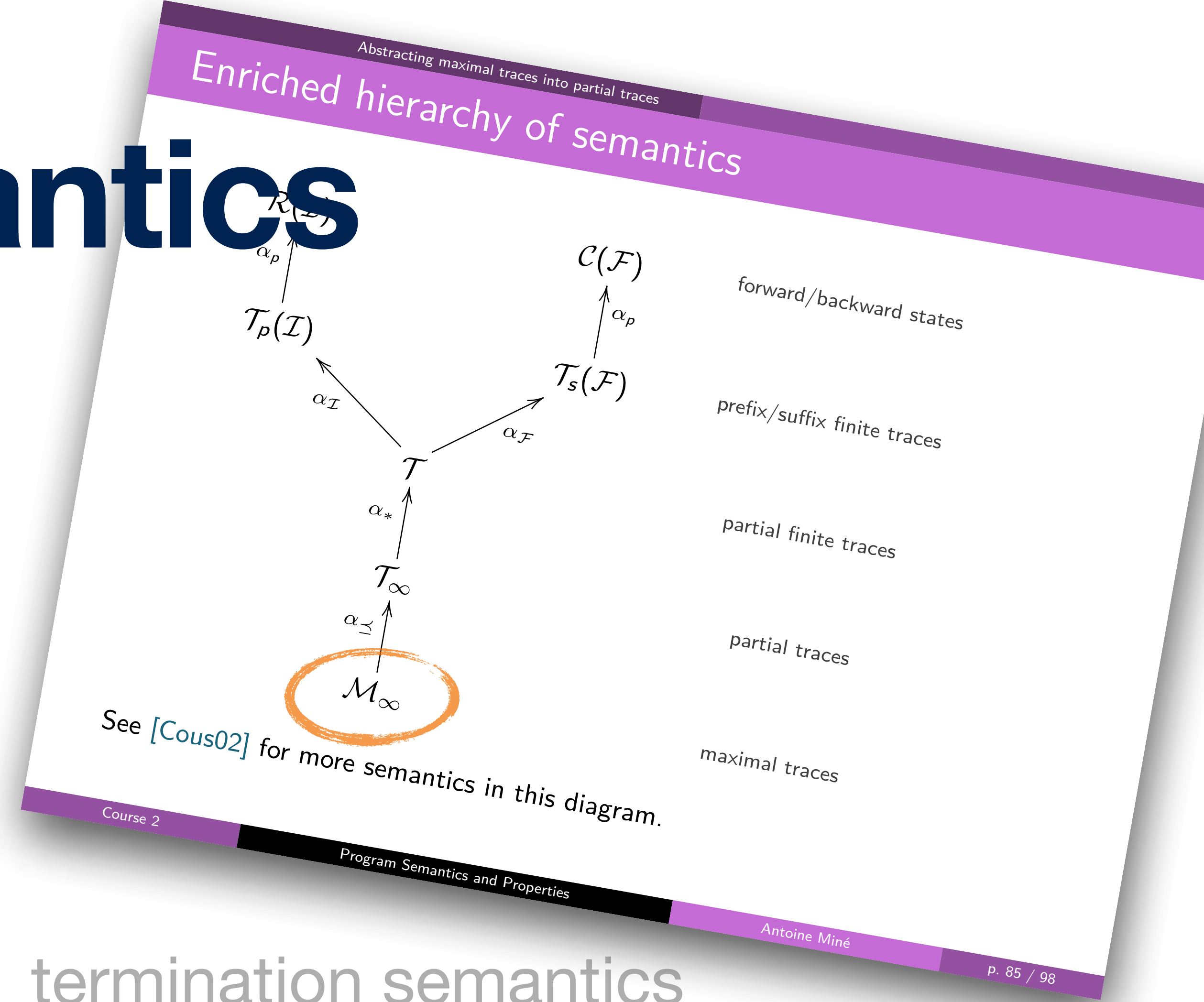


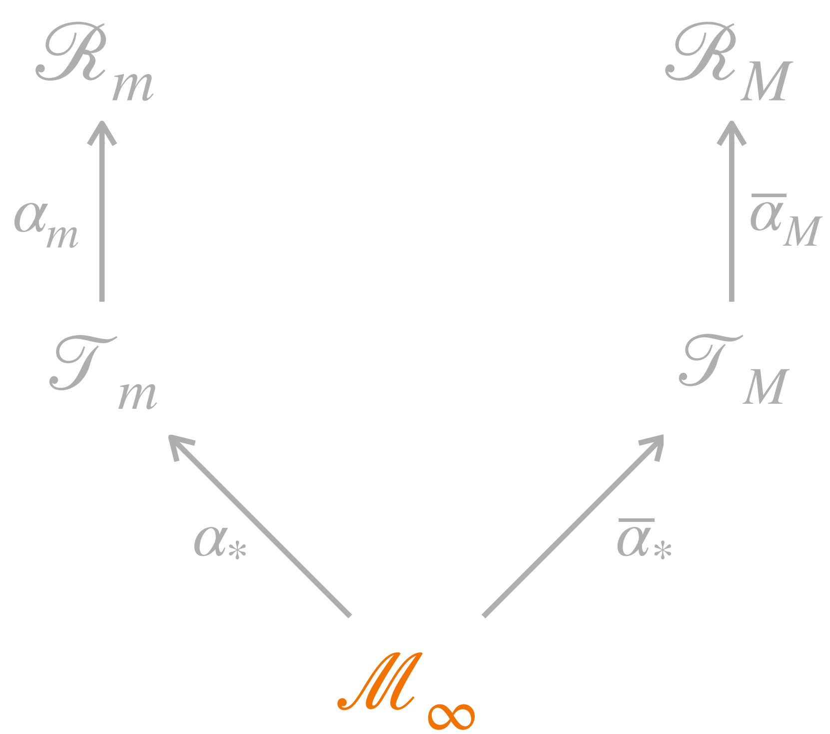
termination semantics

termination trace semantics

maximal trace semantics

Hierarchy of Semantics





termination semantics

termination trace semantics

maximal trace semantics

Maximal Trace Semantics

Example

while $^1([-\infty, +\infty] \neq 0)$ **do**

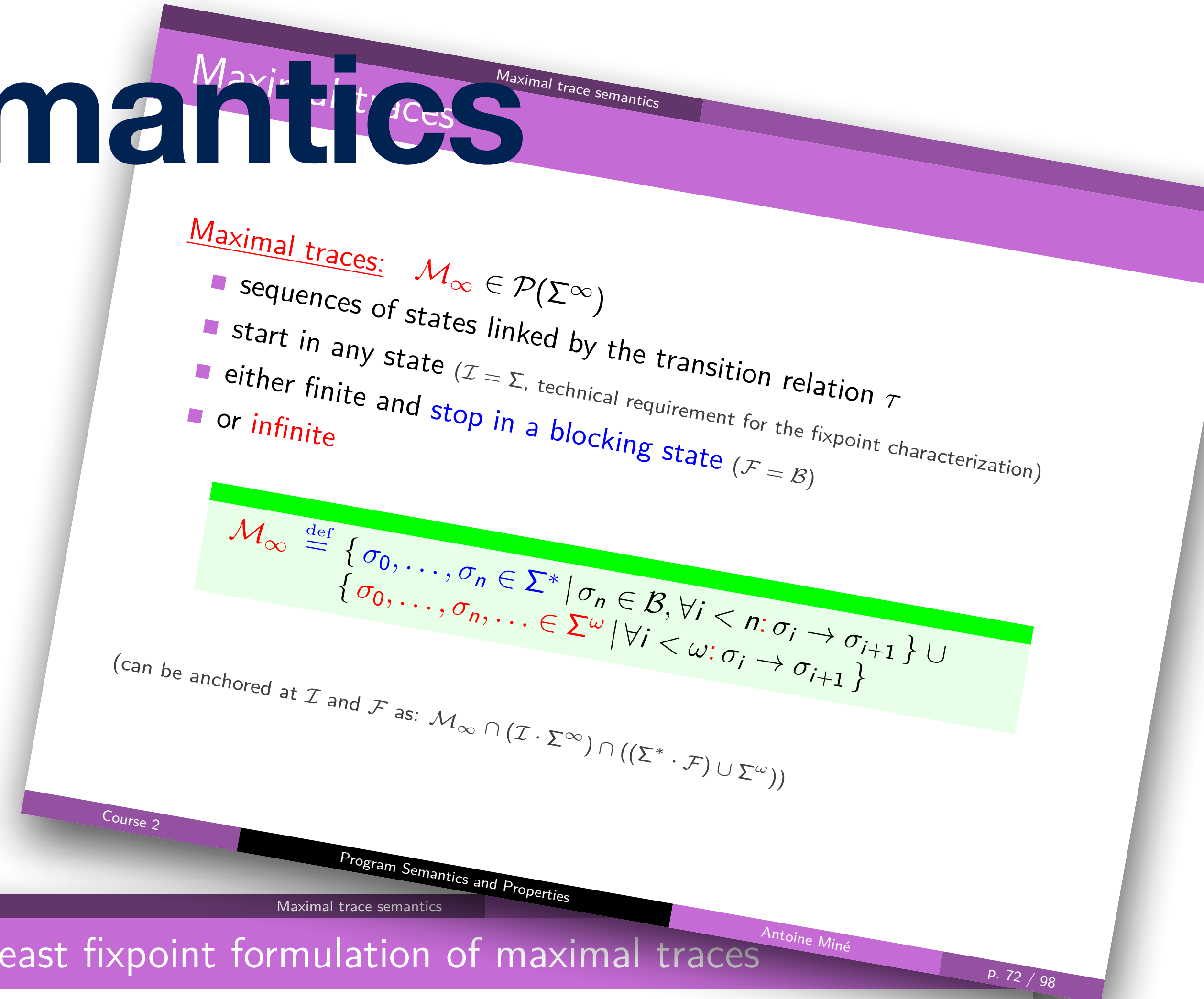
^{2}**skip**

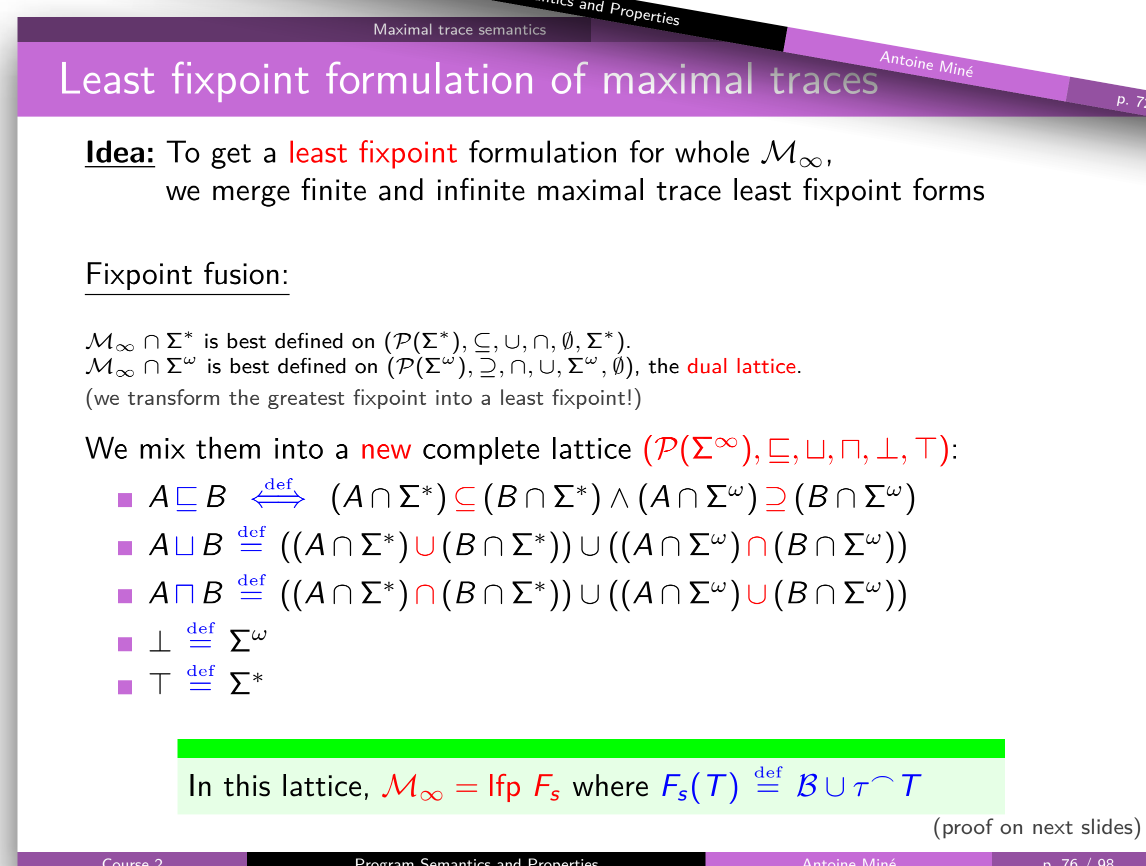
od^3

$$\Sigma \stackrel{\text{def}}{=} \{1,2,3\} \times \mathscr{E}$$

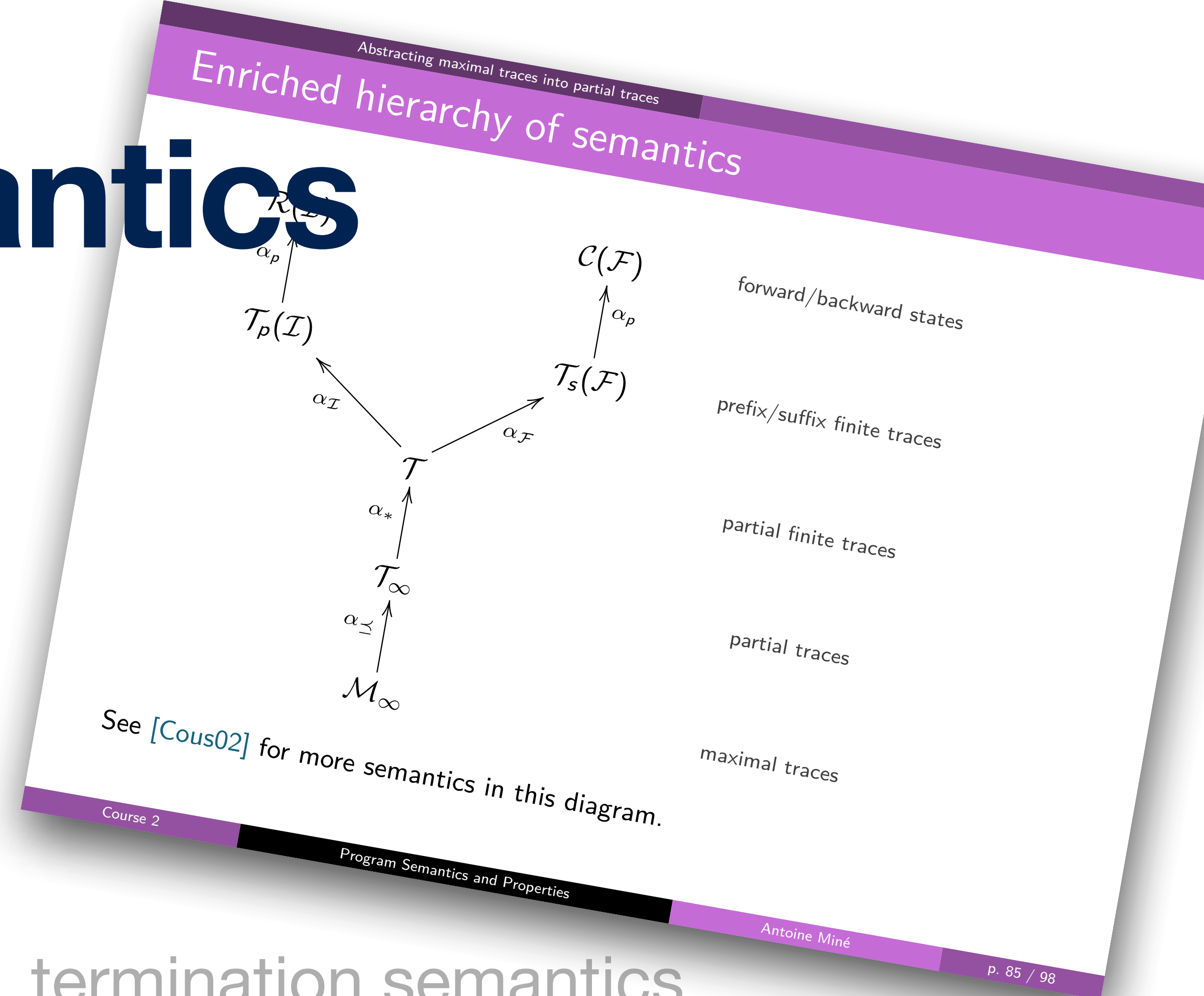
$$\begin{aligned}\tau \stackrel{\text{def}}{=} & \{(1,\rho) \rightarrow (2,\rho) \mid \rho \in \mathscr{E}\} \\ & \cup \{(2,\rho) \rightarrow (1,\rho) \mid \rho \in \mathscr{E}\} \\ & \cup \{(1,\rho) \rightarrow (3,\rho) \mid \rho \in \mathscr{E}\}\end{aligned}$$

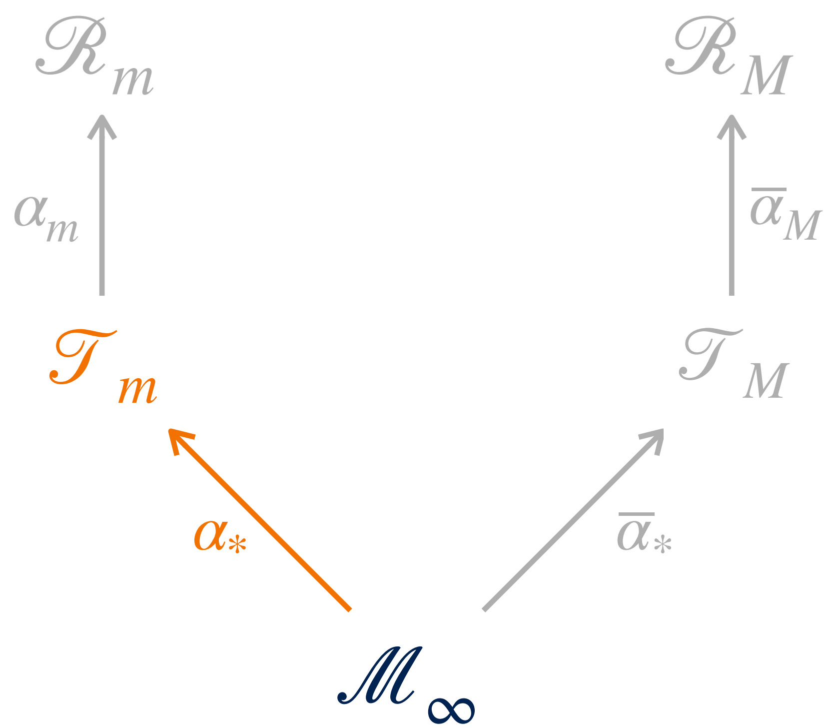
$$\begin{aligned}\mathscr{M}_\infty \stackrel{\text{def}}{=} & \{(1,\rho)(2,\rho)^*(3,\rho) \mid \rho \in \mathscr{E}\} \\ & \cup \{(1,\rho)(2,\rho)^\omega \mid \rho \in \mathscr{E}\}\end{aligned}$$





Hierarchy of Semantics





termination semantics

termination trace semantics

maximal trace semantics

Potential Termination Trace Semantics

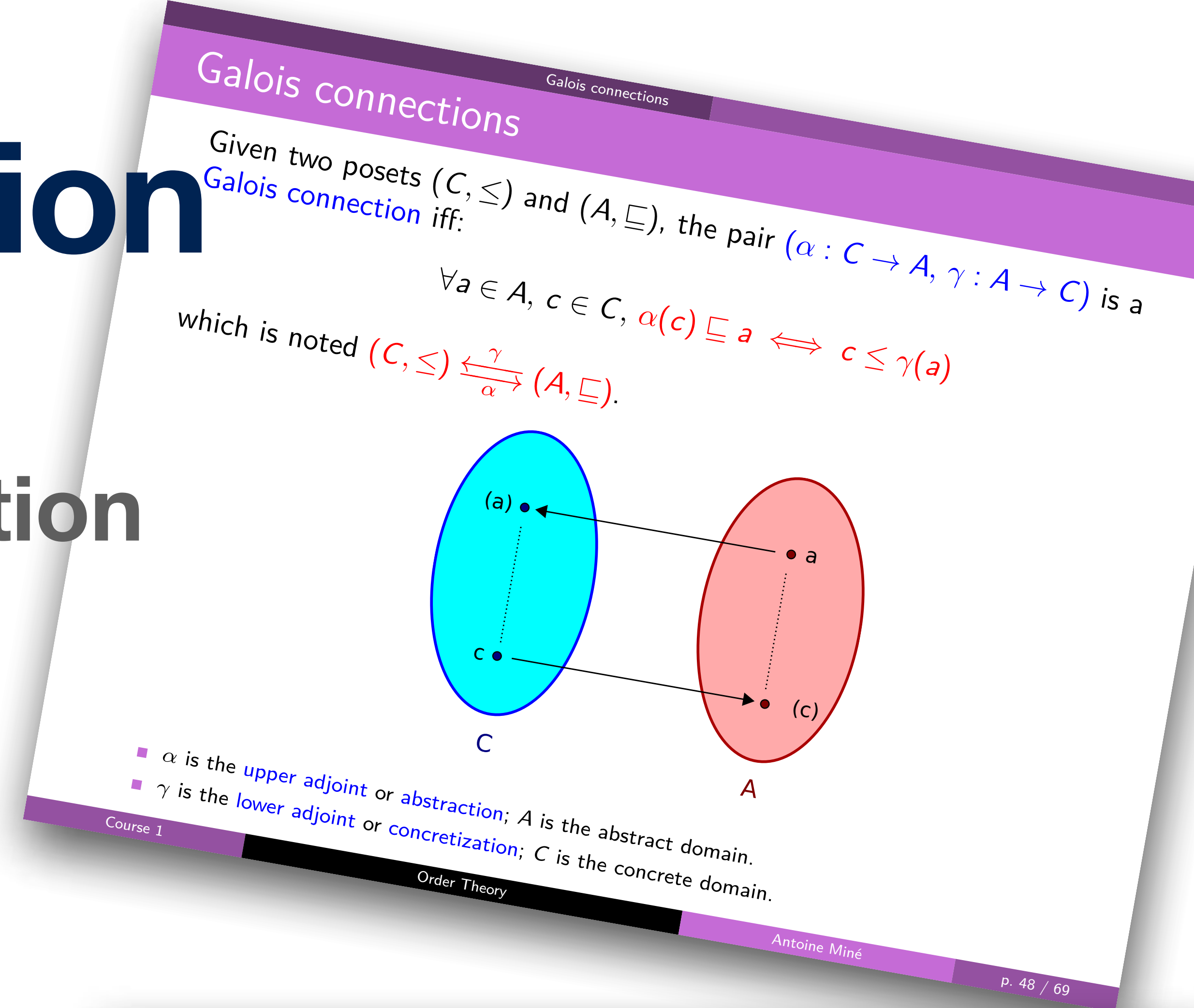
Potential Termination Abstraction

$$\langle \mathscr{P}(\Sigma^\infty), \sqsubseteq \rangle \underset{\alpha_*}{\overset{\gamma_*}{\rightleftarrows}} \langle \mathscr{P}(\Sigma^*), \subseteq \rangle$$

$$\alpha_*(T) \stackrel{\text{def}}{=} T \cap \Sigma^*$$

$$\gamma_*(T) \stackrel{\text{def}}{=} T$$

Example:
$\alpha_*(\{ab, aba, bb, ba^\omega\}) = \{ab, aba, bb\}$



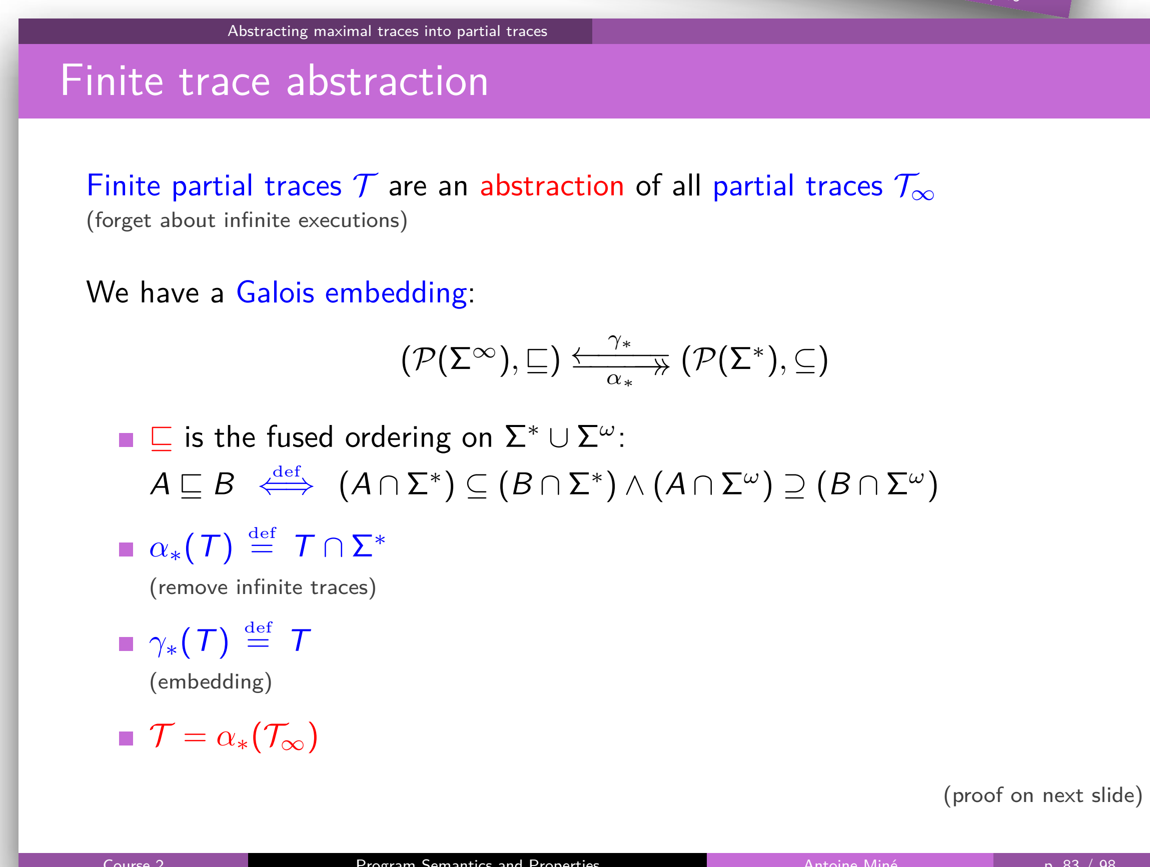
Galois connections

Given two posets $(C, \leq)$ and $(A, \sqsubseteq)$, the pair $(\alpha : C \to A, \gamma : A \to C)$ is a Galois connection iff:

$$\forall a \in A, c \in C,\ \alpha(c) \sqsubseteq a \iff c \leq \gamma(a)$$

which is noted $(C, \leq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \sqsubseteq)$.

- α is the upper adjoint or abstraction; A is the abstract domain.
- γ is the lower adjoint or concretization; C is the concrete domain.



Finite trace abstraction

Finite partial traces $\mathcal{T}$ are an abstraction of all partial traces $\mathcal{T}_\infty$
(forget about infinite executions)

We have a Galois embedding:

$$(\mathcal{P}(\Sigma^\infty), \sqsubseteq) \underset{\alpha_*}{\overset{\gamma_*}{\leftrightarrows}} (\mathcal{P}(\Sigma^*), \subseteq)$$

- $\sqsubseteq$ is the fused ordering on $\Sigma^* \cup \Sigma^\omega$:
  $A \sqsubseteq B \stackrel{\text{def}}{\iff} (A \cap \Sigma^*) \subseteq (B \cap \Sigma^*) \wedge (A \cap \Sigma^\omega) \supseteq (B \cap \Sigma^\omega)$
- $\alpha_*(T) \stackrel{\text{def}}{=} T \cap \Sigma^*$
  (remove infinite traces)
- $\gamma_*(T) \stackrel{\text{def}}{=} T$
  (embedding)
- $\mathcal{T} = \alpha_*(\mathcal{T}_\infty)$

(proof on next slide)

Potential Termination Trace Semantics

Kleenian Fixpoint Transfer

- $\langle \mathscr{P}(\Sigma^\infty), \sqsubseteq \rangle$

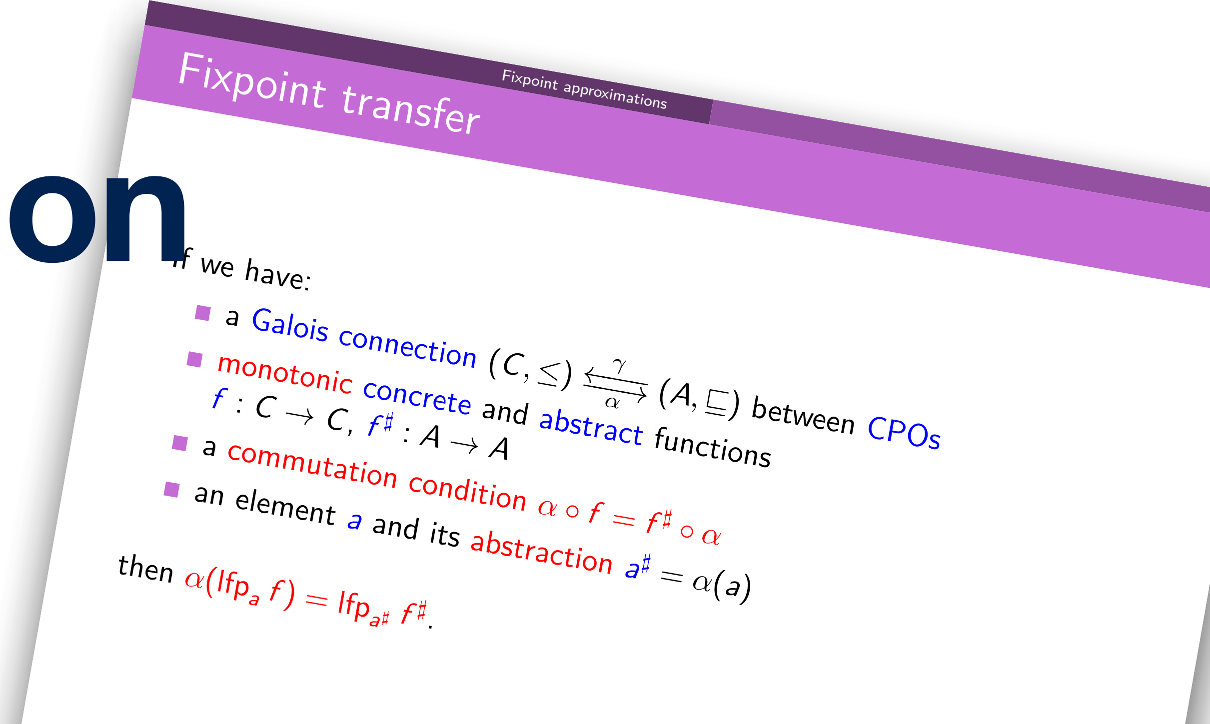
- $\mathscr{M}_\infty \overset{\text{def}}{=} \mathsf{lfp}^{\sqsubseteq} F_s$
  $F_s(T) \overset{\text{def}}{=} \mathscr{B} \cup \tau \frown T$

- $\langle \mathscr{P}(\Sigma^*), \subseteq \rangle$

- $\alpha_* : \mathscr{P}(\Sigma^\infty) \to \mathscr{P}(\Sigma^*)$
  $\alpha_*(T) \overset{\text{def}}{=} T \cap \Sigma^*$

$$\mathscr{T}_m \overset{\text{def}}{=} \alpha_*(\mathscr{M}_\infty) = \mathsf{lfp}^{\subseteq} F_*$$

$$F_*(T) \overset{\text{def}}{=} \mathscr{B} \cup \tau \frown T$$


Fixpoint approximations

Fixpoint transfer

If we have:

- a Galois connection $(C, \leq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \sqsubseteq)$ between CPOs
- monotonic concrete and abstract functions $f : C \to C$, $f^\sharp : A \to A$
- a commutation condition $\alpha \circ f = f^\sharp \circ \alpha$
- an element a and its abstraction $a^\sharp = \alpha(a)$

then $\alpha(\mathsf{lfp}_a f) = \mathsf{lfp}_{a^\sharp} f^\sharp$.

Theorem

Let $\langle C, \leq \rangle$ and $\langle A, \sqsubseteq \rangle$ be complete partial orders, let $f\colon C \to C$ and $f^\#\colon A \to A$ be monotonic functions, and let $\alpha\colon C \to A$ be <u>*a continous* abstraction function</u> such that $\alpha(a) = a^\#$, for $a \in C$ and $a^\# \in A$, and that satisfies the commutation condition $\alpha \circ f = f^\# \circ \alpha$. Then, we have the fixpoint abstraction $\alpha(\mathsf{lfp}^{\leq}_{a} f) = \mathsf{lfp}^{\sqsubseteq}_{a^\#} f^\#$.

Potential Termination Trace Semantics

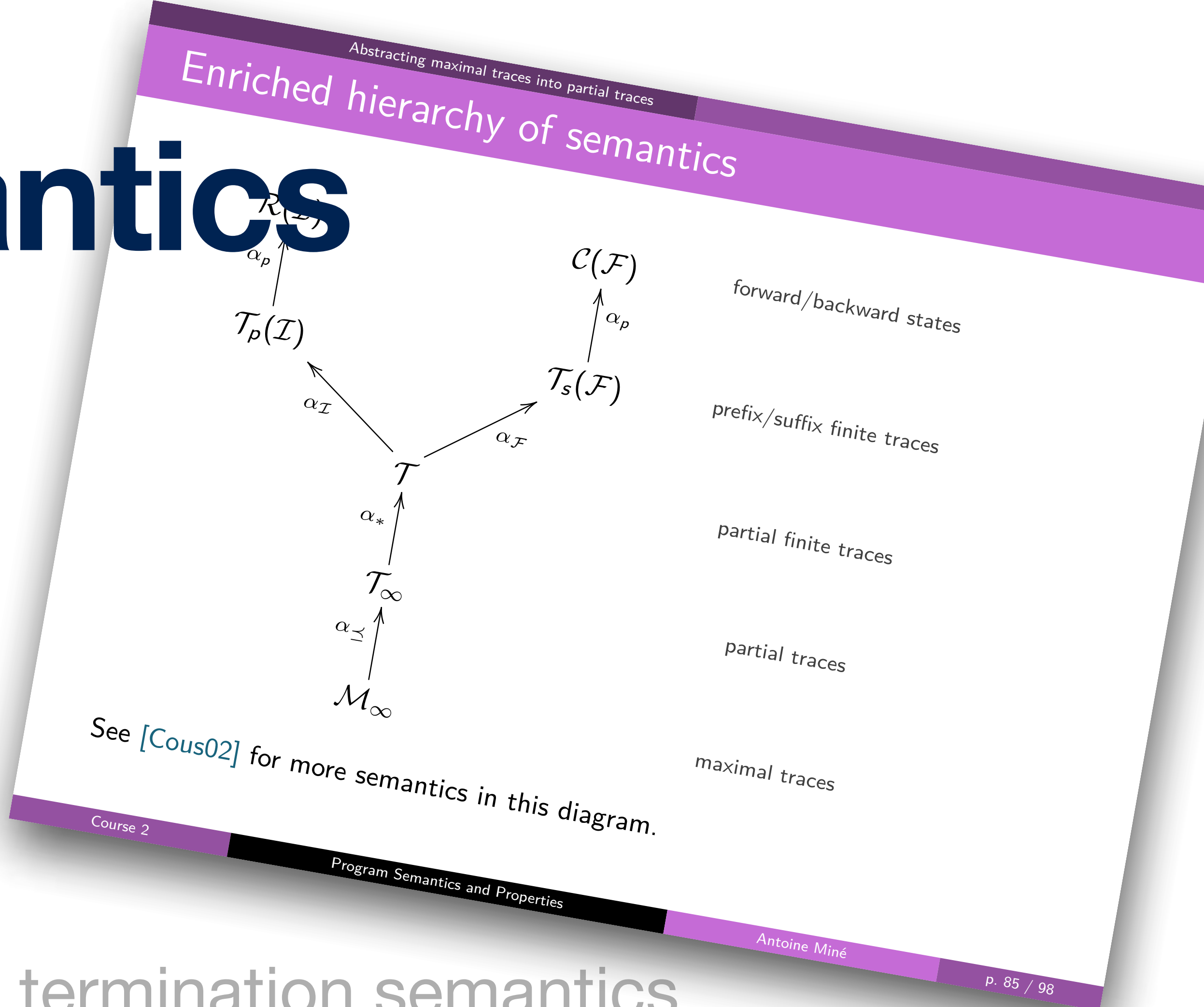
Example

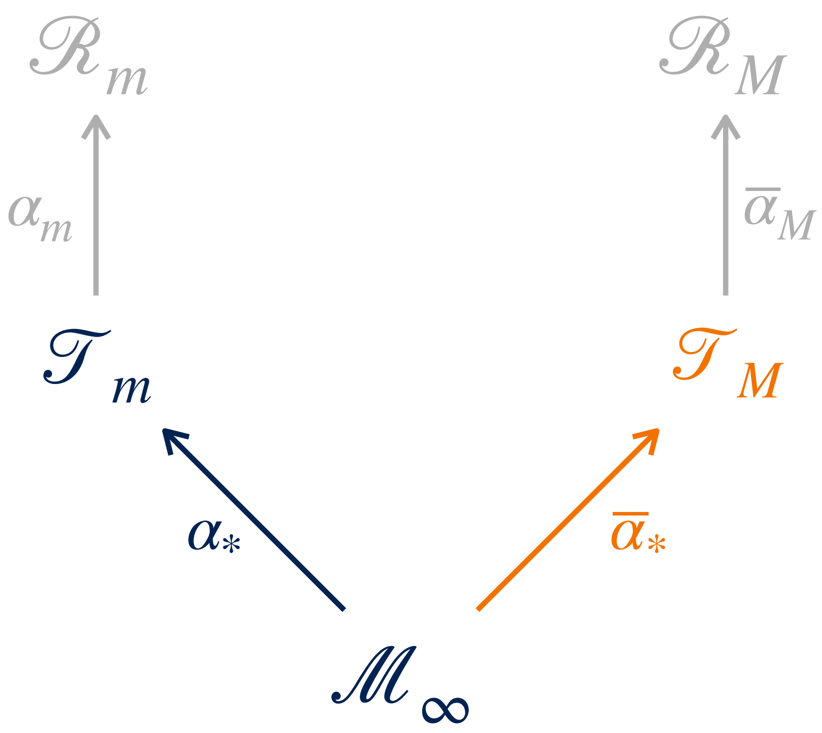
while [1]([-∞, +∞] ≠ 0) **do**
    [2]**skip**
od[3]

$$\mathscr{M}_\infty \stackrel{\text{def}}{=} \{(\mathbf{1}, \rho)(\mathbf{2}, \rho)^*(\mathbf{3}, \rho) \mid \rho \in \mathscr{E}\} \cup \{(\mathbf{1}, \rho)(\mathbf{2}, \rho)^\omega \mid \rho \in \mathscr{E}\}$$

$$\mathscr{T}_m \stackrel{\text{def}}{=} \{(\mathbf{1}, \rho)(\mathbf{2}, \rho)^*(\mathbf{3}, \rho) \mid \rho \in \mathscr{E}\}$$

Hierarchy of Semantics





termination semantics

termination trace semantics

maximal trace semantics

Definite Termination Trace Semantics

Definite Termination Abstraction

$$\langle \mathscr{P}(\Sigma^\infty), \sqsubseteq \rangle \xrightarrow{\overline{\alpha}_*} \langle \mathscr{P}(\Sigma^*), \subseteq \rangle$$

$$\overline{\alpha}_*(T) \stackrel{\text{def}}{=} \{t \in T \cap \Sigma^* \mid \text{nhdb}(t, T \cap \Sigma^\omega) = \varnothing\}$$

$$\text{nhdb}(t, T) \stackrel{\text{def}}{=} \{t' \in T \mid \text{pf}(t) \cap \text{pf}(t') \neq \varnothing\}$$

$$\text{pf}(t) \stackrel{\text{def}}{=} \{t' \in \Sigma^\infty \setminus \{\epsilon\} \mid \exists t'' \in \Sigma^\infty \colon t = t' \cdot t''\}$$

Example:
$\alpha_*(\{ab, aba, bb, ba^\omega\}) = \{ab, aba\}$ since $\text{pf}(bb) \cap \text{pf}(ba^\omega) = \{b\} \neq \varnothing$

Definite Termination Trace Semantics

Tarskian Fixpoint Transfer

- $\langle \mathscr{P}(\Sigma^\infty), \sqsubseteq, \sqcup, \sqcap, \Sigma^\omega, \Sigma^* \rangle$
- $\mathscr{M}_\infty \stackrel{\text{def}}{=} \text{lfp}^{\sqsubseteq} F_s$
  $F_s(T) \stackrel{\text{def}}{=} \mathscr{B} \cup \tau \frown T$
- $\langle \mathscr{P}(\Sigma^*), \subseteq, \cup, \cap, \emptyset, \Sigma^* \rangle$
- $\overline{\alpha}_*: \mathscr{P}(\Sigma^\infty) \to \mathscr{P}(\Sigma^*)$

$\mathscr{T}_M \stackrel{\text{def}}{=} \overline{\alpha}_*(\mathscr{M}_\infty) = \text{lfp}^{\subseteq} \overline{F}_*$

$\overline{F}_*(T) \stackrel{\text{def}}{=} \mathscr{B} \cup ((\tau \frown T) \cap (\Sigma^+ \backslash (\tau \frown (\Sigma^+ \backslash T))))$

Theorem

Let $\langle C, \leq, \vee, \wedge, \bot, \top \rangle$ and $\langle A, \sqsubseteq, \sqcup, \sqcap, \bot^\#, \top^\# \rangle$ be complete lattices, let $f: C \to C$ and $f^\#: A \to A$ be monotonic functions, and let $\alpha: C \to A$ be an abstraction function that is a complete $\wedge$-morphism ($\forall S \subseteq C: f(\wedge S) = \sqcap \{f(s) \mid s \in S\}$) and that satisfies $f^\# \circ \alpha \sqsubseteq \alpha \circ f$ and the post-fixpoint correspondence $\forall a^\# \in A: f^\#(a^\#) \sqsubseteq a^\# \Rightarrow \exists a \in C: f(a) \leq d \wedge \alpha(a) = a^\#$ (i.e., each abstract post-fixpoint of $f^\#$ is the abstraction by α of some concrete post-fixpoint of f). Then, we have the fixpoint abstraction $\alpha(\text{lfp}^{\leq} f) = \text{lfp}^{\sqsubseteq} f^\#$.

(see proof in [Cousot02])

Definite Termination Trace Semantics

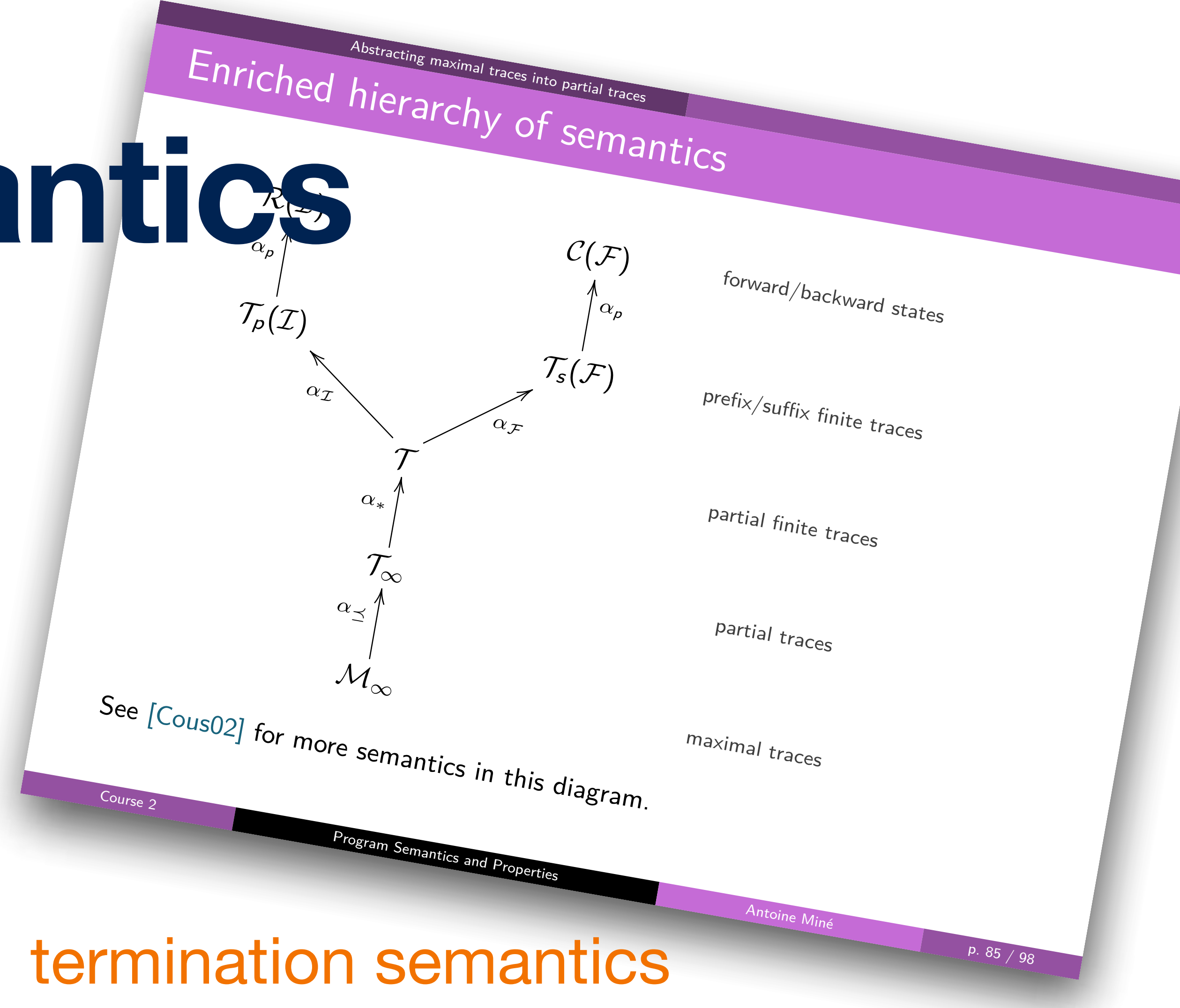
Example

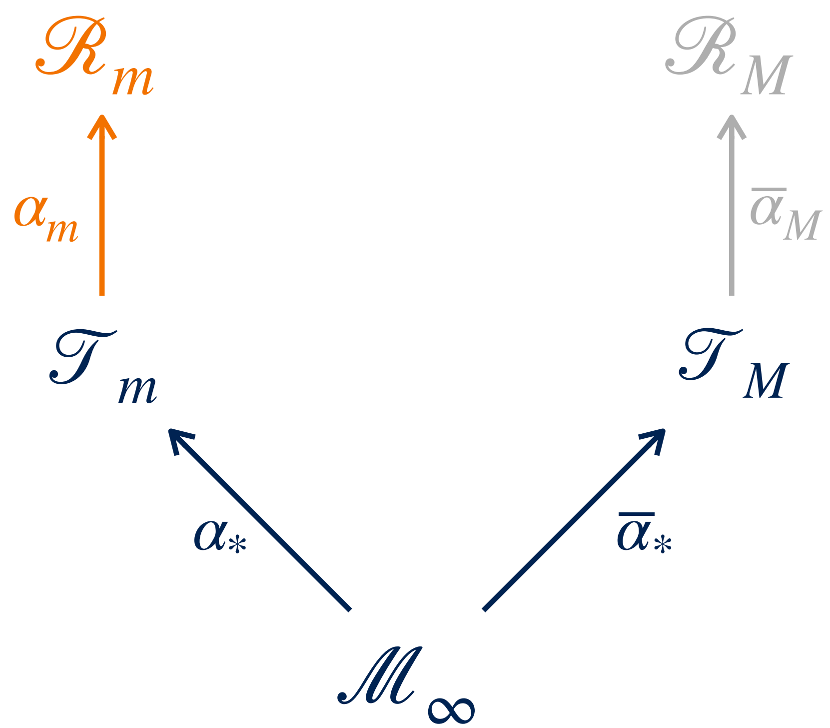
while [1]([-∞, +∞] ≠ 0) **do**
  [2]**skip**
od[3]

$$\mathscr{M}_\infty \stackrel{\text{def}}{=} \{(\mathbf{1}, \rho)(\mathbf{2}, \rho)^*(\mathbf{3}, \rho) \mid \rho \in \mathscr{E}\} \cup \{(\mathbf{1}, \rho)(\mathbf{2}, \rho)^\omega \mid \rho \in \mathscr{E}\}$$

$$\mathscr{T}_M \stackrel{\text{def}}{=} \varnothing$$

Hierarchy of Semantics





termination semantics

termination trace semantics

maximal trace semantics

Potential Termination Semantics

Potential Ranking Abstraction

count execution steps backwards

$\langle \mathscr{P}(\Sigma^*), \subseteq \rangle$ $\qquad$ $\langle \Sigma \rightharpoonup \mathbb{O}, \preceq \rangle$

α_m

$f_1 \preceq f_2 \overset{\text{def}}{=} \text{dom}(f_1) \subseteq \text{dom}(f_2) \land \forall x \in \text{dom}(f_1): f_1(x) \leq f_2(x)$

$$\alpha_m(T) \overset{\text{def}}{=} \alpha_v(\overrightarrow{\alpha}(T))$$

$$\alpha_v(\emptyset) \overset{\text{def}}{=} \dot{\emptyset}$$

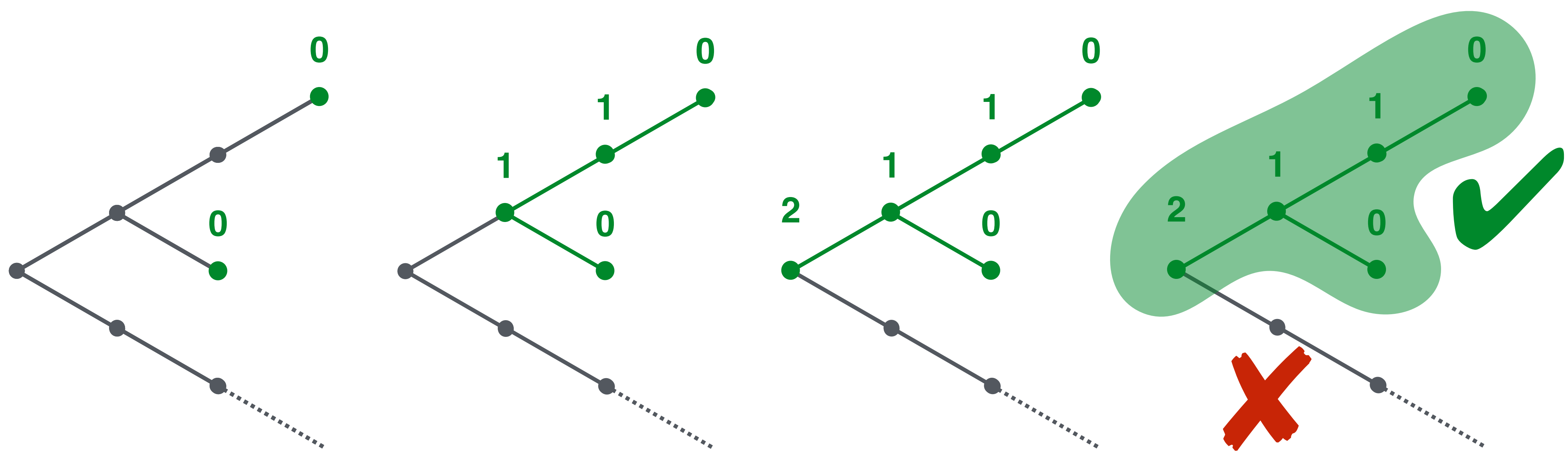
$$\alpha_v(r)\sigma \overset{\text{def}}{=} \begin{cases} 0 & \forall \sigma' \in \Sigma\colon (\sigma, \sigma') \notin r \\ \inf\{\alpha_v(r)\sigma' + 1 \mid \sigma' \in \text{dom}(\alpha_v(r)) \land (\sigma, \sigma') \in r\} & \text{otherwise} \end{cases}$$

$$\overrightarrow{\alpha}(T) \overset{\text{def}}{=} \{(\sigma, \sigma') \in \Sigma \times \Sigma \mid \exists t \in \Sigma^*, t' \in \Sigma^\infty\colon t\sigma\sigma't' \in T\}$$

Potential Termination Semantics

$$\mathcal{R}_m \stackrel{\text{def}}{=} \alpha_m(\mathcal{T}_m) = \text{lfp}^{\preceq} F_m$$

$$F_m(f)\sigma \stackrel{\text{def}}{=} \begin{cases} 0 & \sigma \in \mathcal{B} \\ \inf\{f(\sigma') + 1 \mid (\sigma, \sigma') \in \tau\} & \sigma \in \text{pre}_\tau(\text{dom}(f)) \\ \text{undefined} & \text{otherwise} \end{cases}$$



Theorem

A program **may terminate** for traces starting from a set of initial state $\mathcal{I}$ if and only if $\mathcal{I} \subseteq \text{dom}(\mathcal{R}_m)$

Potential Termination Semantics

Exercise

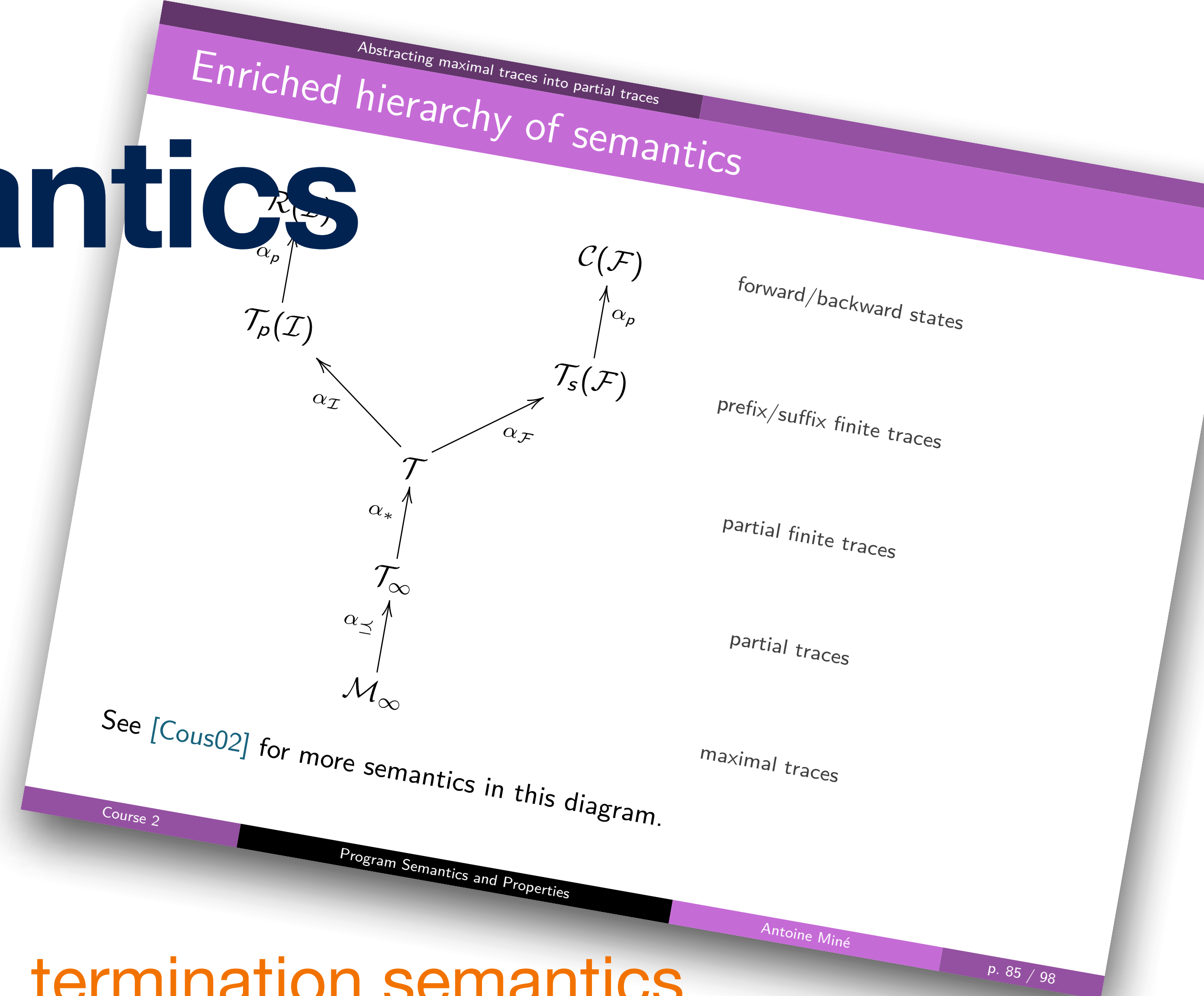
Show that the following fixpoint definition of the potential termination semantics **does not guarantee the existence of a least fixpoint**:

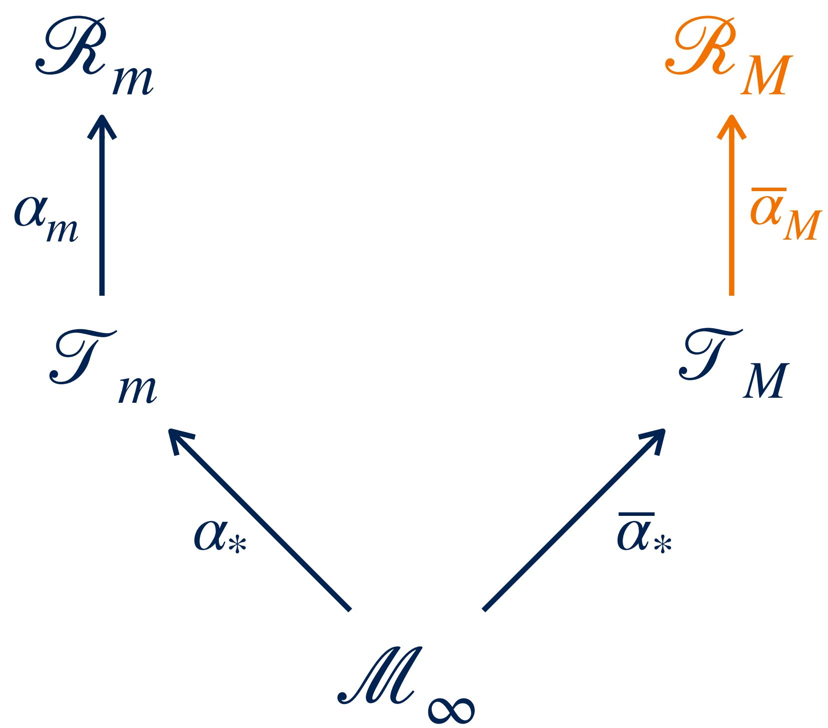
$$\mathcal{R}_m \stackrel{\text{def}}{=} \alpha_m(\mathcal{T}_m) = \text{lfp}^{\preceq} F_m$$

$$F_m(f)\sigma \stackrel{\text{def}}{=} \begin{cases} 0 & \sigma \in \mathcal{B} \\ \text{sup}\{f(\sigma') + 1 \mid (\sigma, \sigma') \in \tau\} & \sigma \in \text{pre}_\tau(\text{dom}(f)) \\ \text{undefined} & \text{otherwise} \end{cases}$$

Hint: find a program for which the values of the iterates of the potential termination semantics are always increasing

Hierarchy of Semantics





termination semantics

termination trace semantics

maximal trace semantics

Definite Termination Semantics

Ranking Abstraction

count execution steps backwards

$\langle \mathscr{P}(\Sigma^*), \subseteq \rangle$ $\xrightarrow{\alpha_m}$ $\langle \Sigma \rightharpoonup \mathbb{O}, \preceq \rangle$

$$\overline{\alpha}_M(T) \stackrel{\text{def}}{=} \overline{\alpha}_V(\overrightarrow{\alpha}(T))$$

$$\overline{\alpha}_V(\emptyset) \stackrel{\text{def}}{=} \dot{\emptyset}$$

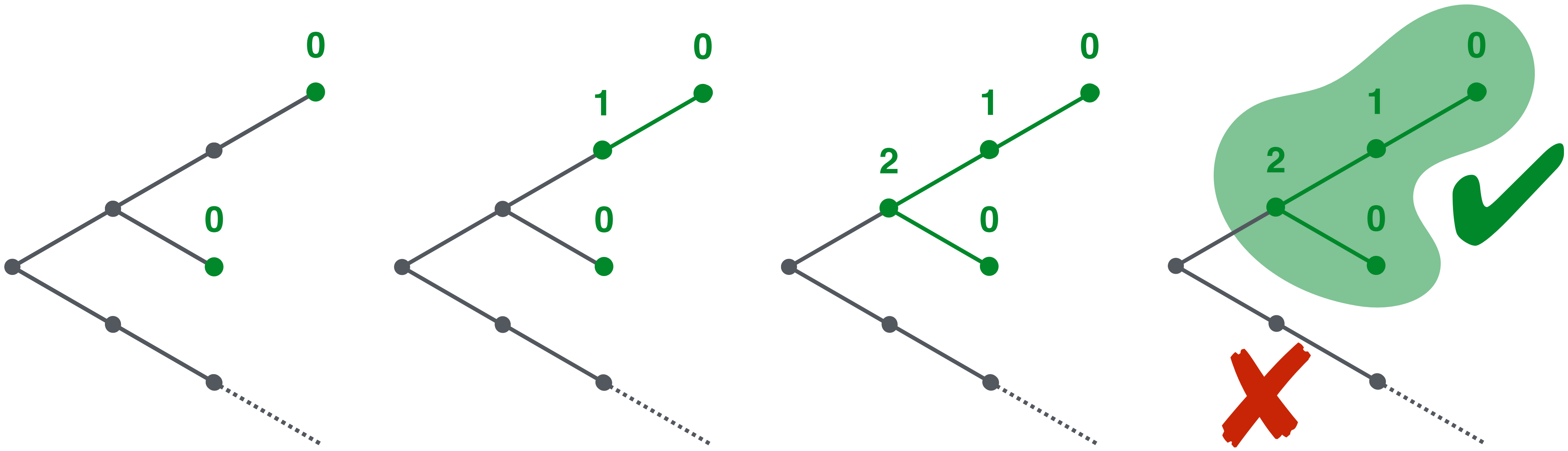
$$\overline{\alpha}_V(r)\sigma \stackrel{\text{def}}{=} \begin{cases} 0 & \forall \sigma' \in \Sigma\colon (\sigma, \sigma') \notin r \\ \text{sup}\{\overline{\alpha}_V(r)\sigma' + 1 \mid \sigma' \in \text{dom}(\overline{\alpha}_V(r)) \wedge (\sigma, \sigma') \in r\} & \text{otherwise} \end{cases}$$

$$\overrightarrow{\alpha}(T) \stackrel{\text{def}}{=} \{(\sigma, \sigma') \in \Sigma \times \Sigma \mid \exists t \in \Sigma^*, t' \in \Sigma^\infty\colon t\sigma\sigma't' \in T\}$$

Definite Termination Semantics

$$\mathcal{R}_M \overset{\text{def}}{=} \overline{\alpha}_M(\mathcal{T}_M) = \text{lfp}^{\preceq}\, \overline{F}_M$$

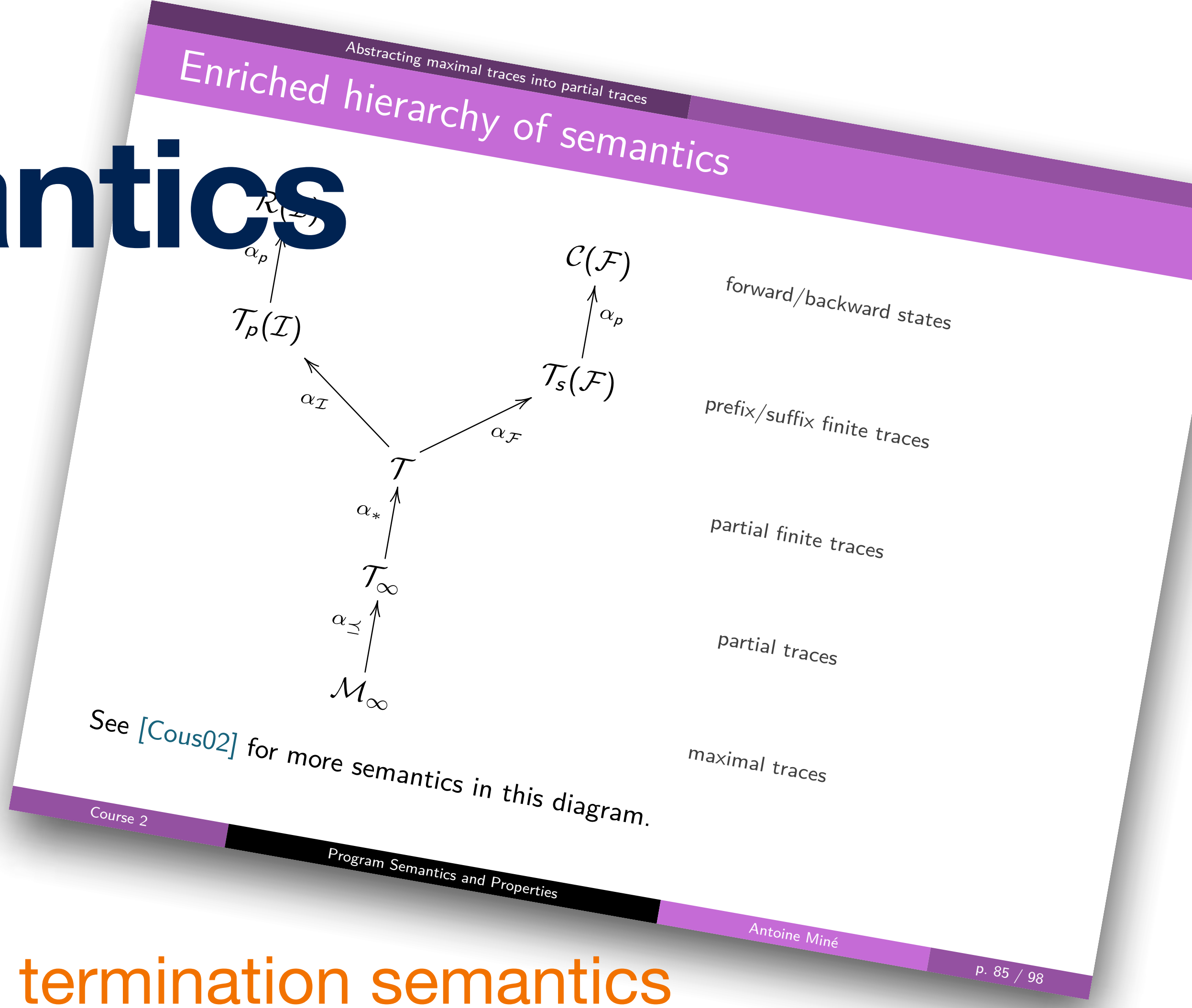
$$\overline{F}_M(f)\sigma \overset{\text{def}}{=} \begin{cases} 0 & \sigma \in \mathcal{B} \\ \sup\{f(\sigma') + 1 \mid (\sigma, \sigma') \in \tau\} & \sigma \in \widetilde{\text{pre}}_\tau(\text{dom}(f)) \\ \text{undefined} & \text{otherwise} \end{cases}$$

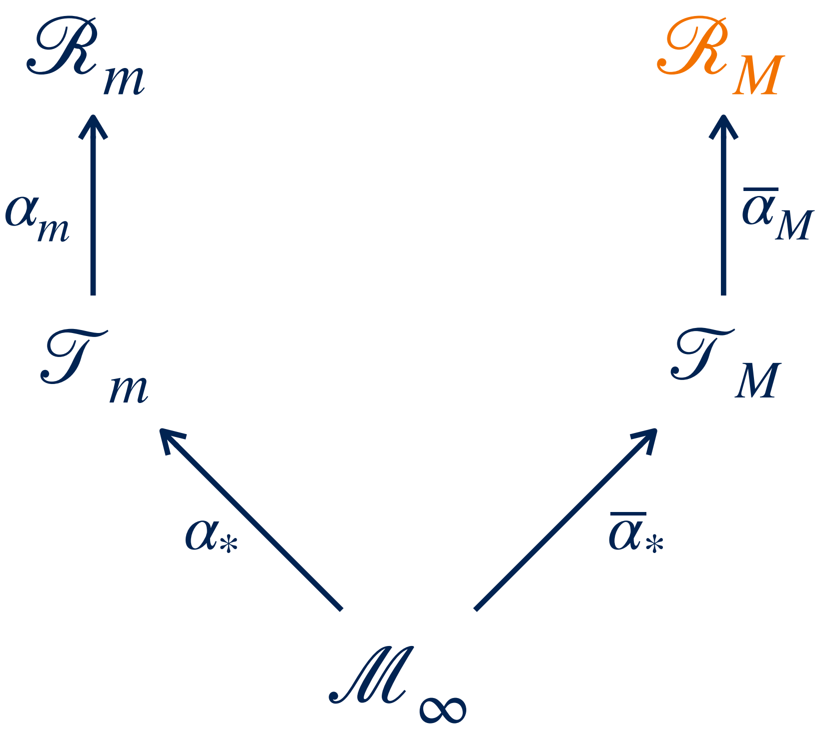


Theorem

A program **must terminate** for traces starting from a set of initial states $\mathcal{I}$ if and only if $\mathcal{I} \subseteq \text{dom}(\mathcal{R}_M)$

Hierarchy of Semantics





termination semantics

termination trace semantics

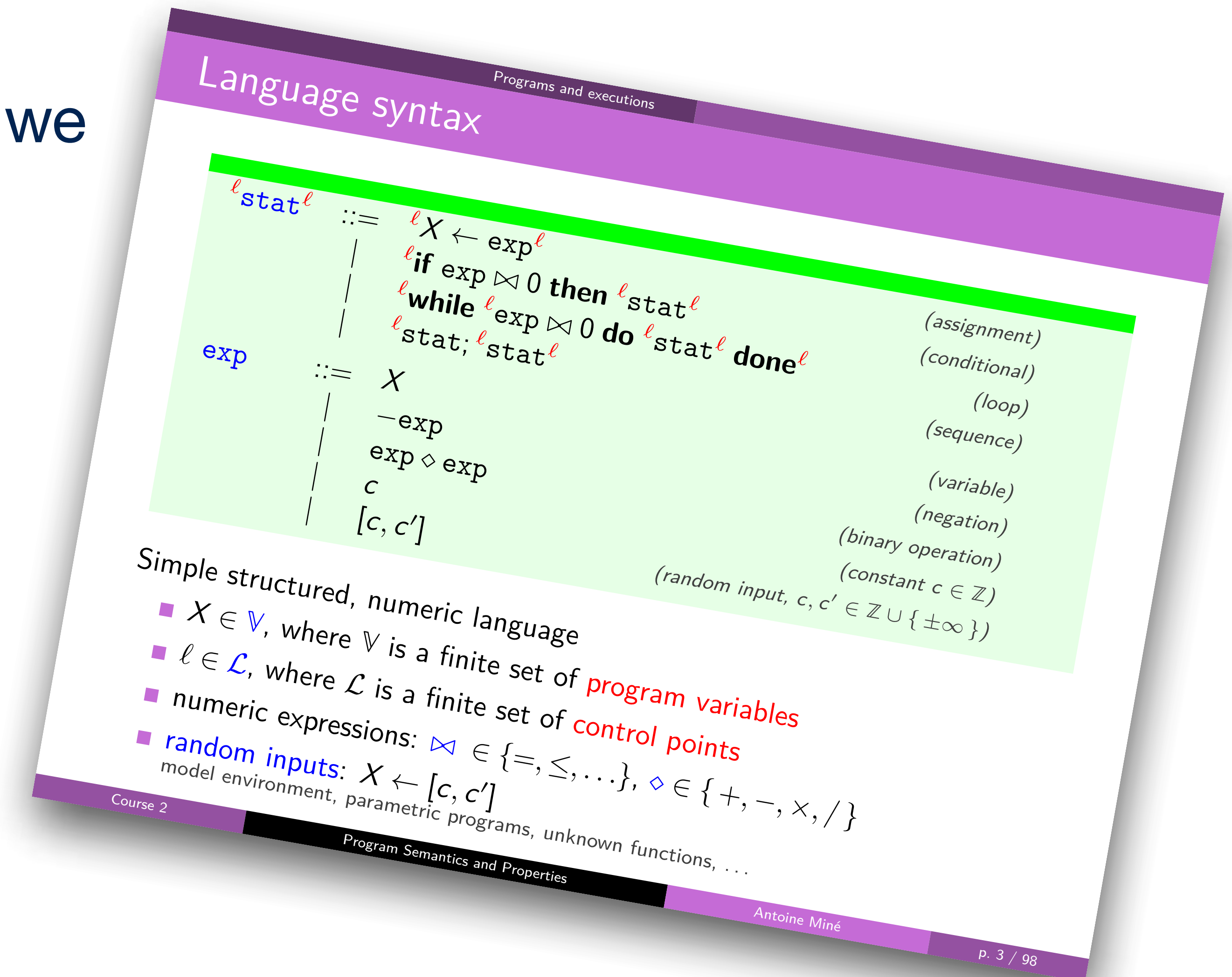
maximal trace semantics

Denotational Definite Termination Semantics

We define the definite termination semantics $\mathcal{R}_M \colon \Sigma \rightharpoonup \mathbb{O}$ by partitioning with respect to the program control points, i.e., $\mathcal{R}_M \colon \mathcal{L} \to (\mathcal{E} \rightharpoonup \mathbb{O})$.

Thus, for each program instruction stat, we define a transformer $\mathcal{R}_M[\![\text{stat}]\!] \colon (\mathcal{E} \rightharpoonup \mathbb{O}) \to (\mathcal{E} \rightharpoonup \mathbb{O})$:

- $\mathcal{R}_M[\![{}^{\ell}X \leftarrow e]\!]$
- $\mathcal{R}_M[\![\textbf{if } {}^{\ell} e \bowtie 0 \textbf{ then } s]\!]$
- $\mathcal{R}_M[\![\textbf{while } {}^{\ell} e \bowtie 0 \textbf{ do } s \textbf{ done}]\!]$
- $\mathcal{R}_M[\![s_1; s_2]\!]$



Denotational Definite Termination Semantics

$\mathcal{R}_M[\![{}^{\ell}X \leftarrow e]\!]$

$$\mathcal{R}_M[\![{}^{\ell}X \leftarrow e]\!]f \overset{\text{def}}{=} \lambda\rho\,.\begin{cases} \sup\{f(\rho[X \mapsto v])+1 \mid v \in E[\![e]\!]\rho\} & \exists \bar{v} \in E[\![e]\!]\rho \land \\ & \forall v \in E[\![e]\!]\rho : \rho[X \mapsto v] \in \text{dom}(f) \\ \text{undefined} & \text{otherwise} \end{cases}$$

Example:

Let $\mathbb{V} = \{x\}$ and $f : \mathcal{E} \rightharpoonup \mathbb{O}$ defined as follows:

$$f(\rho) \overset{\text{def}}{=} \begin{cases} 2 & \rho(x) = 1 \\ 3 & \rho(x) = 2 \\ \text{undefined} & \text{otherwise} \end{cases}$$

We have

$$\mathcal{R}_M[\![x \leftarrow x + [1,2]]\!]f \overset{\text{def}}{=} \lambda\rho\,.\begin{cases} 4 & \rho(x) = 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

Denotational Definite Termination Semantics

$\mathcal{R}_M[\![\textbf{if}\ {}^{\ell}\ e \bowtie 0\ \textbf{then}\ s]\!]$

$$\mathcal{R}_M[\![\textbf{if}\ {}^{\ell}\ e \bowtie 0\ \textbf{then}\ s]\!]f \stackrel{\text{def}}{=} \lambda\rho\,.\begin{cases} ① \\ ② \\ ③ \\ \text{undefined} & \text{otherwise} \end{cases}$$

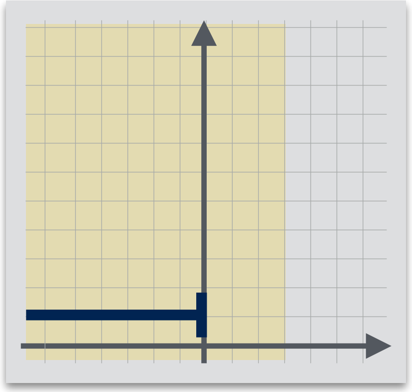
① $\sup\{\mathcal{R}_M[\![s]\!]f(\rho)+1,\ f(\rho)+1\}$ $\quad \rho \in \text{dom}(\mathcal{R}_M[\![s]\!]f) \cap \text{dom}(f) \wedge \exists v_1, v_2 \in E[\![e]\!]\rho :\ v_1 \bowtie 0 \wedge v_2 \not\bowtie 0$

② $\mathcal{R}_M[\![s]\!]f(\rho)+1$ $\quad \rho \in \text{dom}(\mathcal{R}_M[\![s]\!]f) \wedge \forall v \in E[\![e]\!]\rho :\ v \bowtie 0$

③ $f(\rho)+1$ $\quad \rho \in \text{dom}(f) \wedge \forall v \in E[\![e]\!]\rho :\ v \not\bowtie 0$

Denotational Definite Termination Semantics

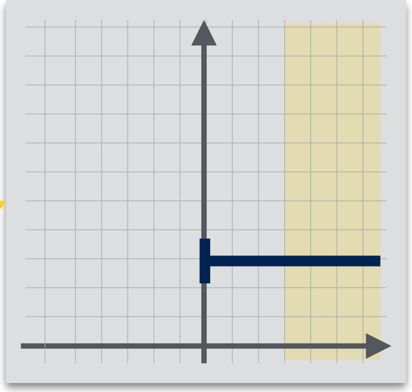
$\mathscr{R}_M[\![\textbf{if } {}^{\ell} e \bowtie 0 \textbf{ then } s]\!]$ **(continue)**



Example:

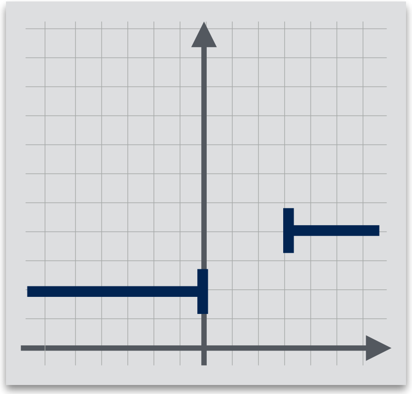
Let $\mathbb{V} = \{x\}$ and $f\colon \mathscr{E} \rightharpoonup \mathbb{O}$, and $\mathscr{R}_M[\![s]\!]f$ defined as follows:

$$f \stackrel{\text{def}}{=} \lambda\rho\,.\begin{cases} 1 & \rho(x) \leq 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$



$$\mathscr{R}_M[\![s]\!]f \stackrel{\text{def}}{=} \lambda\rho\,.\begin{cases} 3 & 0 \leq \rho(x) \\ \text{undefined} & \text{otherwise} \end{cases}$$

We have



$$\mathscr{R}_M[\![\textbf{if } 3 - x < 0 \textbf{ then } s]\!]f \stackrel{\text{def}}{=} \lambda\rho\,.\begin{cases} 2 & \rho(x) \leq 0 \\ 4 & 3 < \rho(x) \\ \text{undefined} & \text{otherwise} \end{cases}$$

and $\mathscr{R}_M[\![\textbf{if } [-\infty, +\infty] \neq 0 \textbf{ then } s]\!]f \stackrel{\text{def}}{=} \lambda\rho\,.\begin{cases} 4 & \rho(x) = 0 \\ \text{undefined} & \text{otherwise} \end{cases}$

Denotational Definite Termination Semantics

$\mathcal{R}_M[\![\textbf{while}\ {}^{\ell}\ e \bowtie 0\ \textbf{do}\ s\ \textbf{done}]\!]$

$$\mathcal{R}_M[\![\textbf{while}\ {}^{\ell}\ e \bowtie 0\ \textbf{do}\ s\ \textbf{done}]\!]f \stackrel{\text{def}}{=} \text{lfp}^{\preceq}_{\dot{\emptyset}}\ \overline{F}_M$$

$$F_M(x) \stackrel{\text{def}}{=} \lambda\rho\,.\begin{cases} ① \\ ② \\ ③ \\ \text{undefined} & \text{otherwise} \end{cases}$$

① $\text{sup}\{\mathcal{R}_M[\![s]\!]x(\rho) + 1,\ f(\rho) + 1\}$ $\quad \rho \in \text{dom}(\mathcal{R}_M[\![s]\!]x) \cap \text{dom}(f) \wedge$
$\exists v_1, v_2 \in E[\![e]\!]\rho :\ v_1 \bowtie 0 \wedge v_2 \not\bowtie 0$

② $\mathcal{R}_M[\![s]\!]x(\rho) + 1$ $\quad \rho \in \text{dom}(\mathcal{R}_M[\![s]\!]x) \wedge$
$\forall v \in E[\![e]\!]\rho :\ v \bowtie 0$

③ $f(\rho) + 1$ $\quad \rho \in \text{dom}(f) \wedge \forall v \in E[\![e]\!]\rho :\ v \not\bowtie 0$

Denotational Definite Termination Semantics

$\mathcal{R}_M[\![s_1; s_2]\!]$

$$\mathcal{R}_M[\![s_1; s_2]\!]f \stackrel{\text{def}}{=} \mathcal{R}_M[\![s_1]\!](\mathcal{R}_M[\![s_2]\!]f)$$

Denotational Definite Termination Semantics

Definition

The **definite termination semantics** $\mathcal{R}_M[\![\text{stat}^{\ell}]\!] : \mathcal{E} \rightharpoonup \mathbb{O}$ of a program stat^{ℓ} is:

$$\mathcal{R}_M[\![\text{stat}^{\ell}]\!] \stackrel{\text{def}}{=} \mathcal{R}_M[\![\text{stat}]\!](\lambda\rho.0)$$

where $\mathcal{R}_M[\![\text{stat}]\!] : (\mathcal{E} \rightharpoonup \mathbb{O}) \to (\mathcal{E} \rightharpoonup \mathbb{O})$ is the definite termination semantics of each program instruction stat

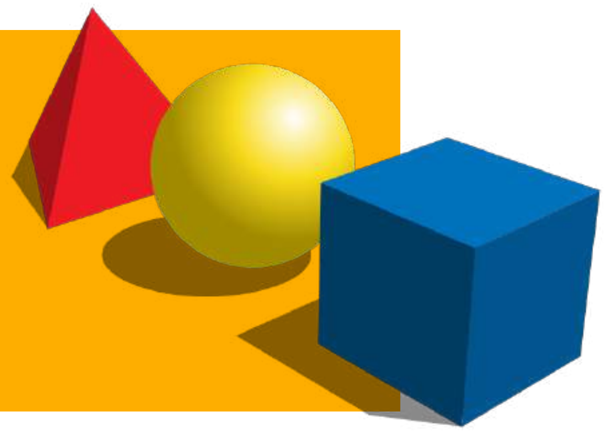
Theorem

A program stat^{ℓ} **must terminate** for traces starting from a set of initial states $\mathcal{I}$ if and only if $\mathcal{I} \subseteq \text{dom}(\mathcal{R}_m[\![\text{stat}^{\ell}]\!])$

Abstract Interpretation Recipe

practical tools
targeting specific programs

algorithmic approaches
to decide program properties



mathematical models
of the program behavior

Piecewise-Defined Ranking Functions Abstract Domain

Concretization-Based Piecewise Abstraction

γ_A

$\mathcal{R}^{\#}_M[\![\mathsf{stat}^{\ell}]\!] \in \mathcal{A}$

$\langle \mathcal{E} \rightharpoonup \mathbb{O}, \preccurlyeq \rangle$ $\quad$ $\langle \mathcal{A}, \preccurlyeq_A \rangle$

$\mathcal{R}_M[\![\mathsf{stat}^{\ell}]\!] : \mathcal{E} \rightharpoonup \mathbb{O}$

$f_1 \preccurlyeq f_2 \overset{\text{def}}{=} \mathrm{dom}(f_1) \supseteq \mathrm{dom}(f_2) \wedge \forall x \in \mathrm{dom}(f_1) : f_1(x) \leq f_2(x)$

approximation order

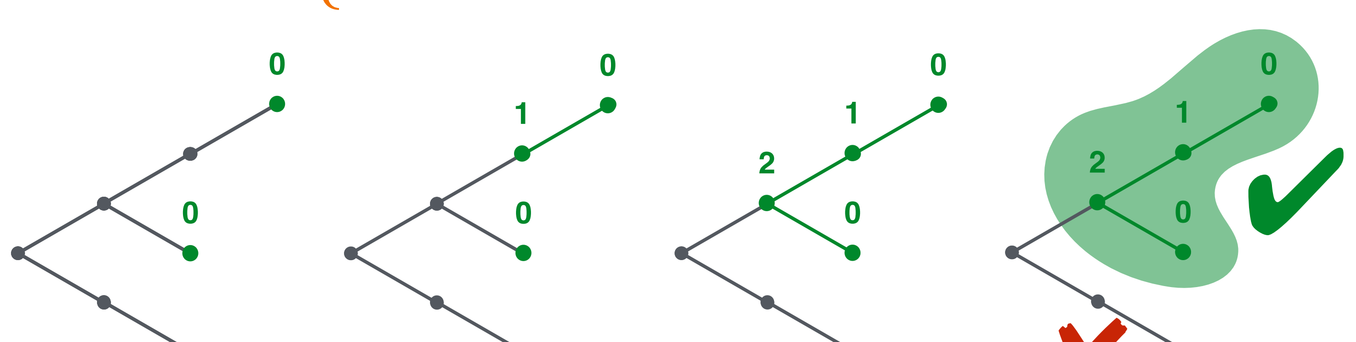
Definite Termination Semantics

$\mathcal{R}_M \overset{\text{def}}{=} \overline{\alpha}_M(\mathcal{T}_M) = \mathsf{lfp}^{\preceq}\, \overline{F}_M$

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

computational order

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



Concretization-Based Piecewise Abstraction

γ_A

$\mathcal{R}^{\#}_M[\![\text{stat}^{\ell}]\!] \in \mathcal{A}$

$\langle \mathcal{E} \rightharpoonup \mathbb{O}, \preccurlyeq \rangle$ $\langle \mathcal{A}, \preccurlyeq_A \rangle$

$\mathcal{R}_M[\![\text{stat}^{\ell}]\!] : \mathcal{E} \rightharpoonup \mathbb{O}$

$f_1 \preccurlyeq f_2 \overset{\text{def}}{=} \text{dom}(f_1) \supseteq \text{dom}(f_2) \wedge \forall x \in \text{dom}(f_1) : f_1(x) \leq f_2(x)$

approximation order

By *pointwise lifiting* we obtain an abstraction $\mathcal{R}^{\#}_M$ of $\mathcal{R}_M$:

$\dot{\gamma}_A$

$\langle \mathcal{L} \rightarrow (\mathcal{E} \rightharpoonup \mathbb{O}), \dot{\preccurlyeq} \rangle$ $\langle \mathcal{L} \rightarrow \mathcal{A}, \dot{\preccurlyeq}_A \rangle$

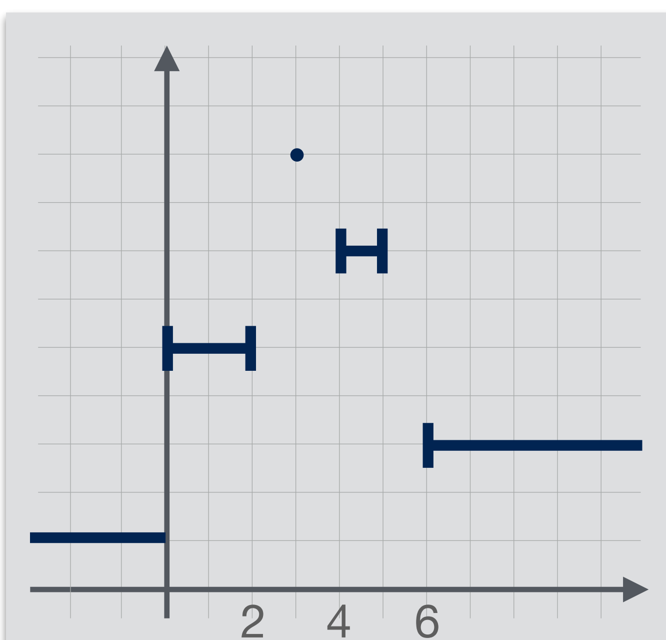
$\mathcal{R}_M : \mathcal{L} \rightarrow (\mathcal{E} \rightharpoonup \mathbb{O})$ $\mathcal{R}^{\#}_M : \mathcal{L} \rightarrow \mathcal{A}$

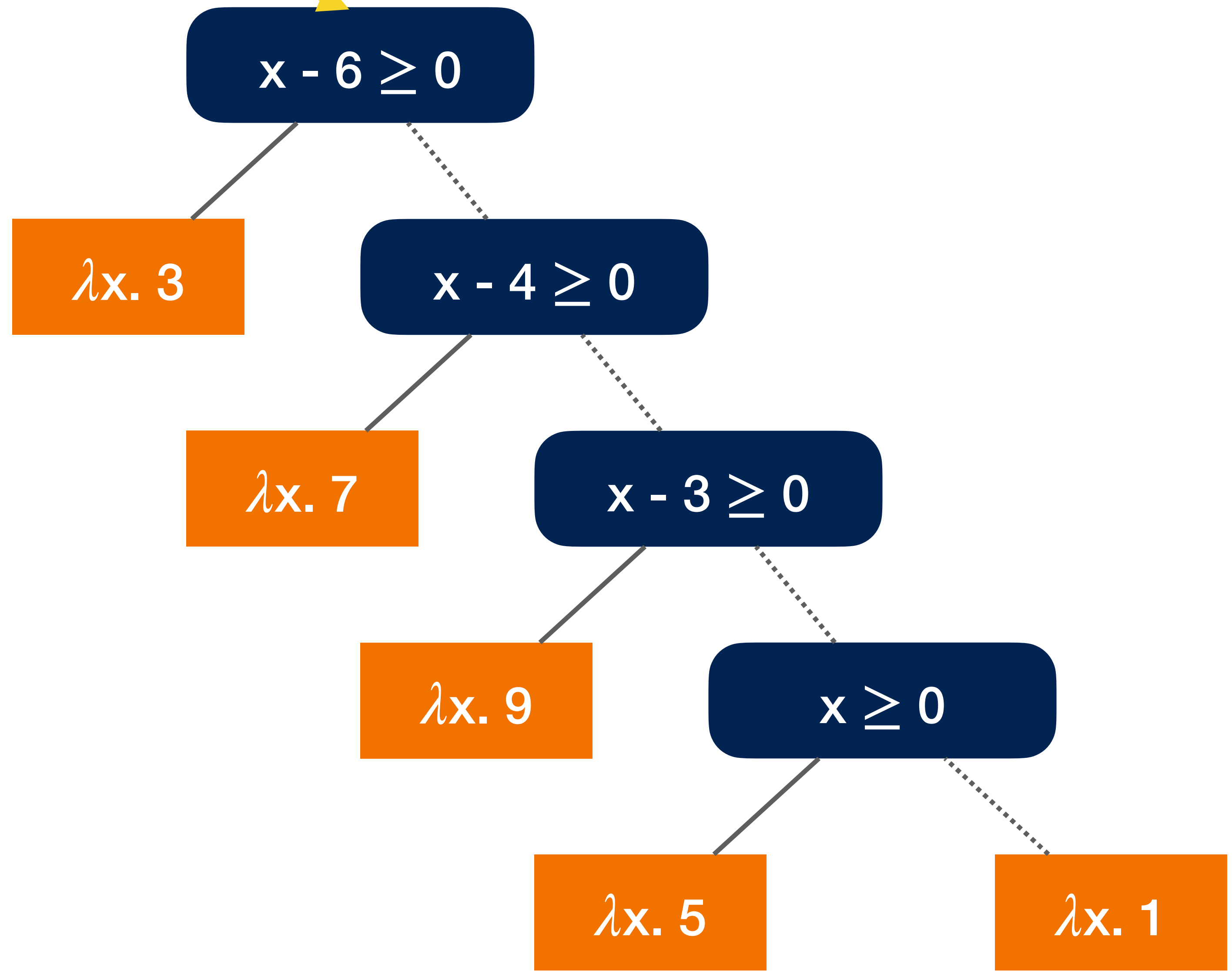
Piecewise-Defined Ranking Functions Abstract Domain

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

Example

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

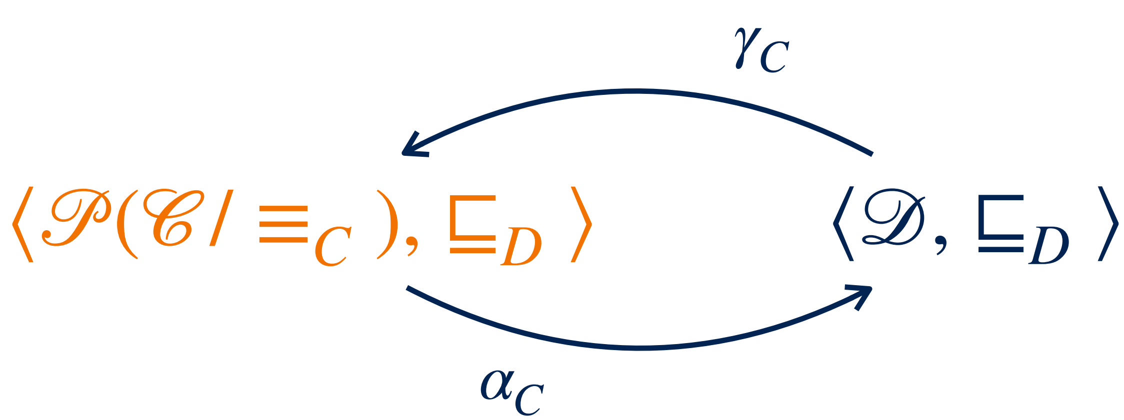


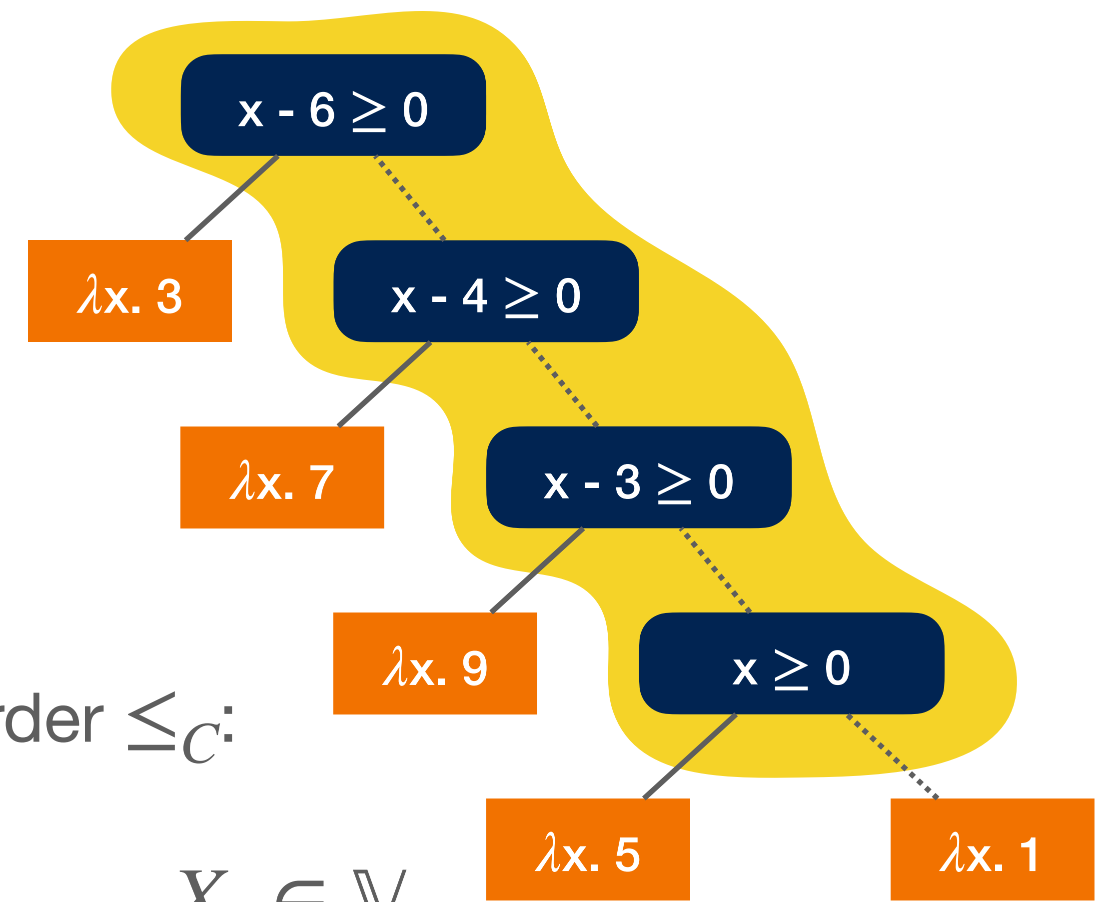


Piecewise-Defined Ranking Functions Abstract Domain

Linear Constraints Auxiliary Abstract Domain

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





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

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

Piecewise-Defined Ranking Functions Abstract Domain

Functions Auxiliary Abstract Domain

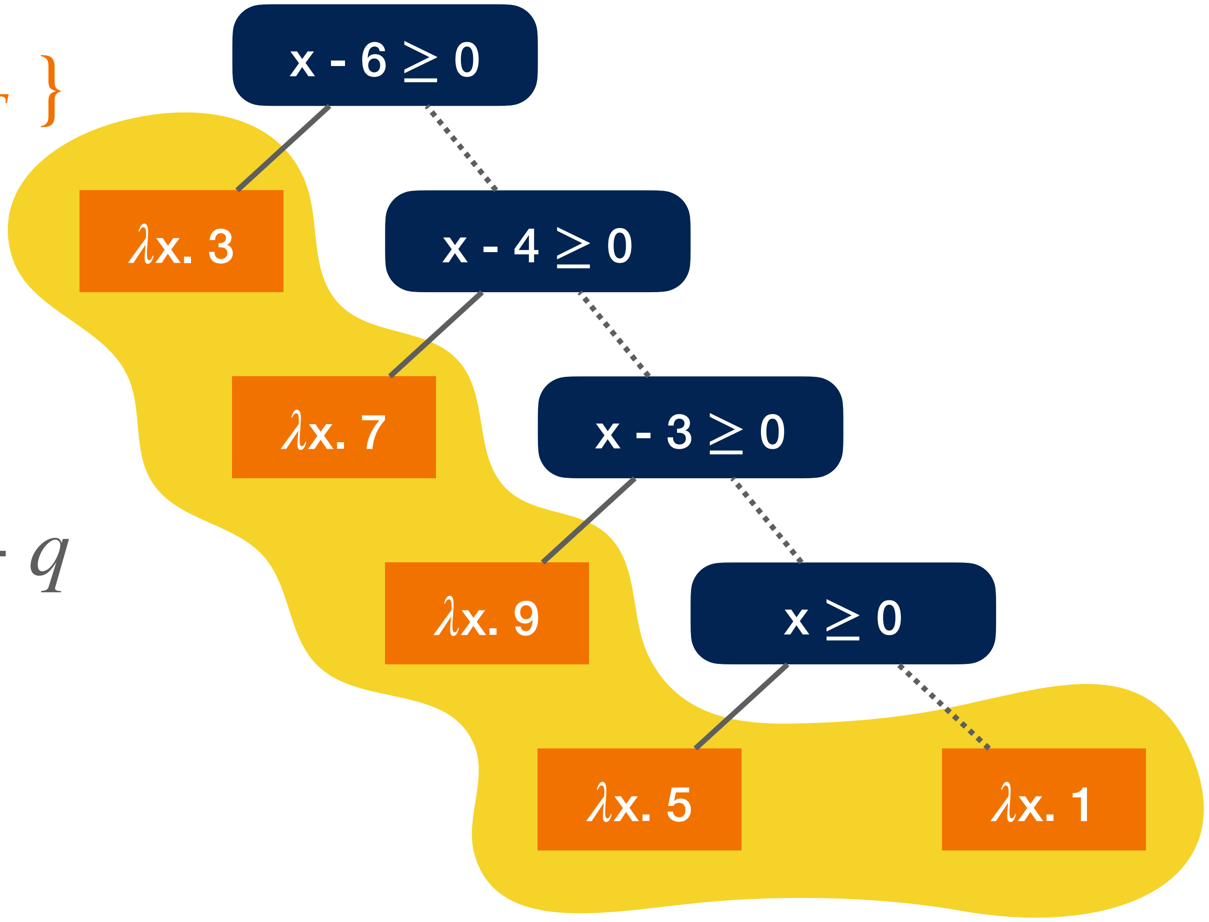
- Parameterized by an ***underlying numerical abstract domain*** $\langle \mathcal{D}, \sqsubseteq_D \rangle$
- $\mathcal{F} \stackrel{\text{def}}{=} \{ \perp_F \} \cup (\mathbb{Z}^{|\mathbb{V}|} \rightarrow \mathbb{N}) \cup \{ \top_F \}$

  We consider **affine functions**:

  $\mathcal{F}_A \stackrel{\text{def}}{=} \{ \perp_F \} \cup \{ f \colon \mathbb{Z}^{|\mathbb{V}|} \rightarrow \mathbb{N} \mid$

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

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



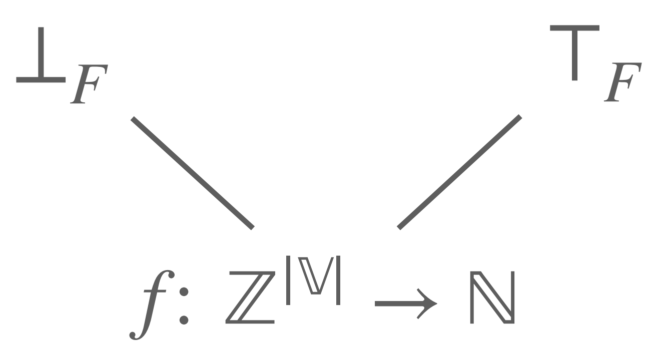
Piecewise-Defined Ranking Functions Abstract Domain

Functions Auxiliary Abstract Domain (continue)

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

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

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



Piecewise-Defined Ranking Functions Abstract Domain

Functions Auxiliary Abstract Domain (continue)

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

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

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

$$\top_F$$

$$|$$

$$f : \mathbb{Z}^{|\mathbb{V}|} \rightarrow \mathbb{N}$$

$$|$$

$$\bot_F$$

Piecewise-Defined Ranking Functions Abstract Domain

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

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

$$\gamma_A(t) \stackrel{\text{def}}{=} \bar{\gamma}_A[\emptyset](t)$$

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

$$\bar{\gamma}_A[C](\text{LEAF}: f) \stackrel{\text{def}}{=} \gamma_F[\alpha_C(C)](f)$$

$$\bar{\gamma}_A[C](\text{NODE}\{c\}: t_1; t_2) \stackrel{\text{def}}{=} \bar{\gamma}_A[C \cup \{c\}](t_1) \,\dot{\cup}\, \bar{\gamma}_A[C \cup \{\neg c\}](t_2)$$

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

$$\gamma_F[D](\perp_F) \stackrel{\text{def}}{=} \dot{\emptyset}$$

$$\gamma_F[D](f) \stackrel{\text{def}}{=} \lambda\rho \in \gamma_D(D) : f(\ldots, \rho(X_i), \ldots)$$

$$\gamma_F[D](\top_F) \stackrel{\text{def}}{=} \dot{\emptyset}$$

Piecewise-Defined Ranking Functions Abstract Domain

Abstract Domain Operators

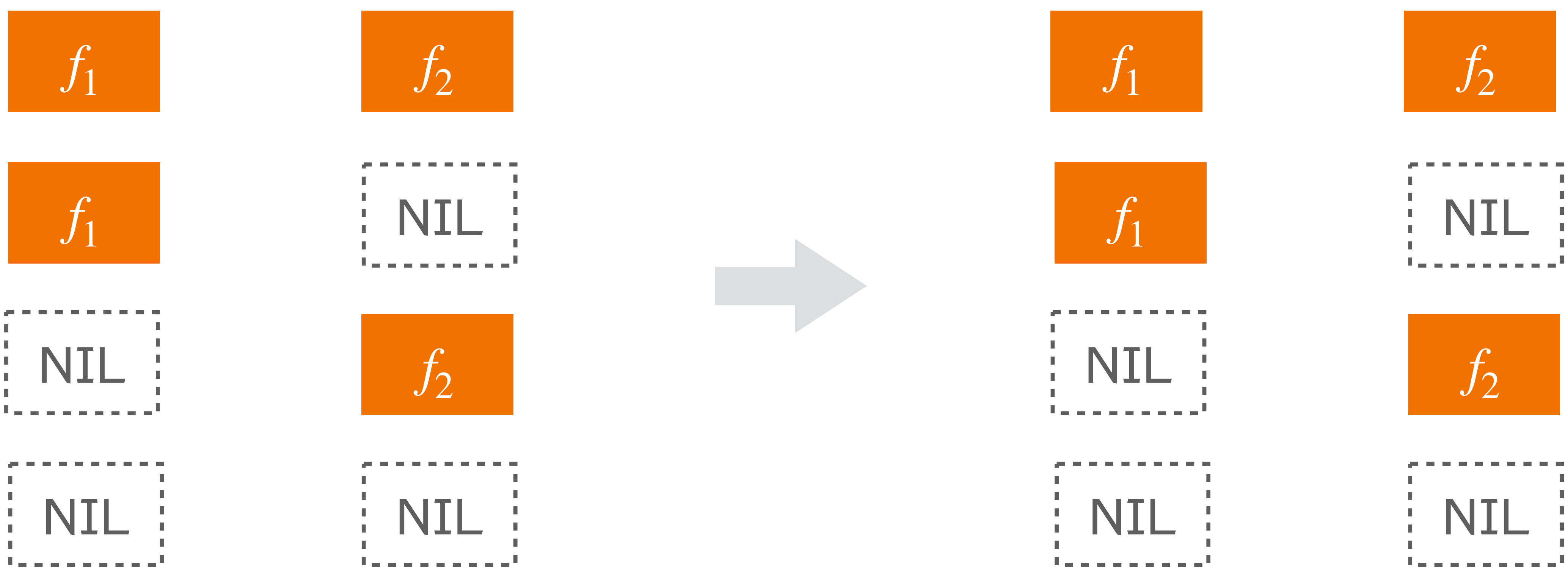
- They manipulate elements in $\mathscr{A}_{\mathsf{NIL}} \stackrel{\text{def}}{=} \{\mathsf{NIL}\} \cup \mathscr{A}$
- The **binary operators** rely on a tree unification algorithm
  - approximation order $\preccurlyeq_A$ and computational order $\sqsubseteq_A$
  - approximation join $\curlyvee_A$ and 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 Ranking Functions Abstract Domain

Tree Unification

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

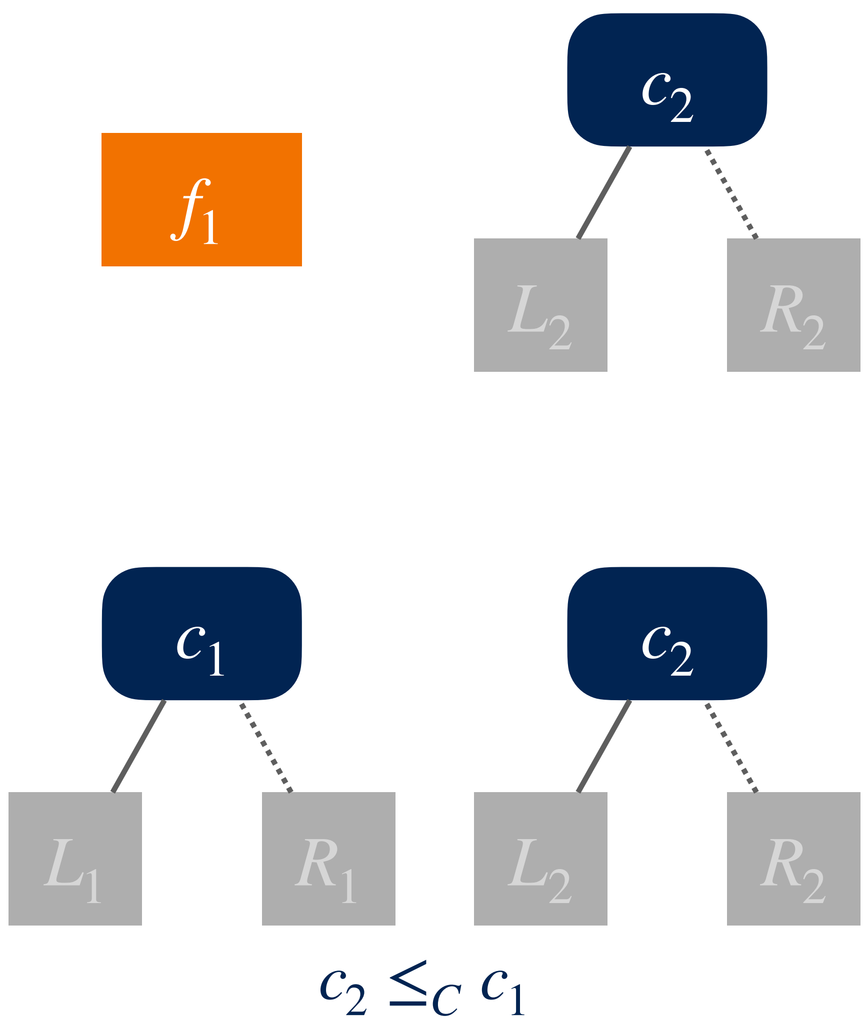
- Base cases:



Piecewise-Defined Ranking Functions Abstract Domain

Tree Unification (continue)

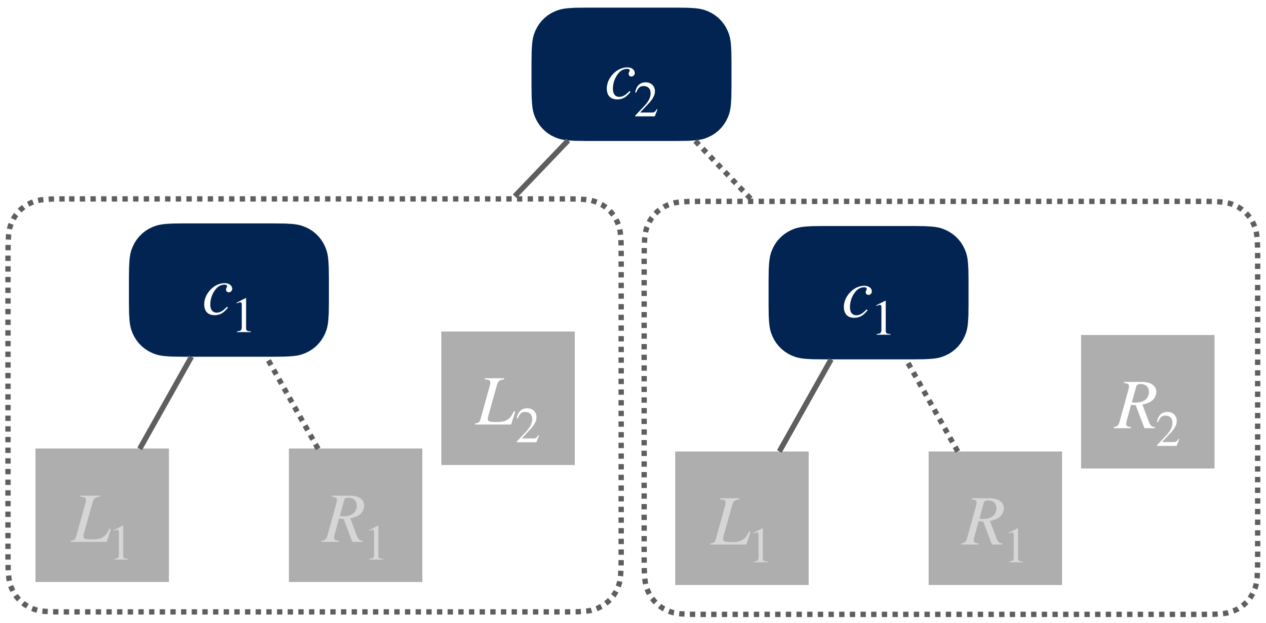
- Case ①



(1a) c_2 is redundant

(1b) $\neg c_2$ is redundant

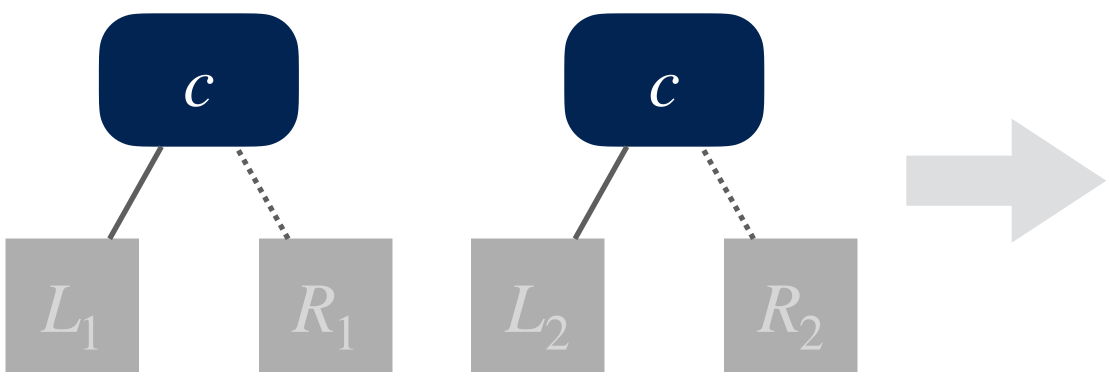
(1c) c_2 is added to t_1



Piecewise-Defined Ranking Functions Abstract Domain

Tree Unification (continue)

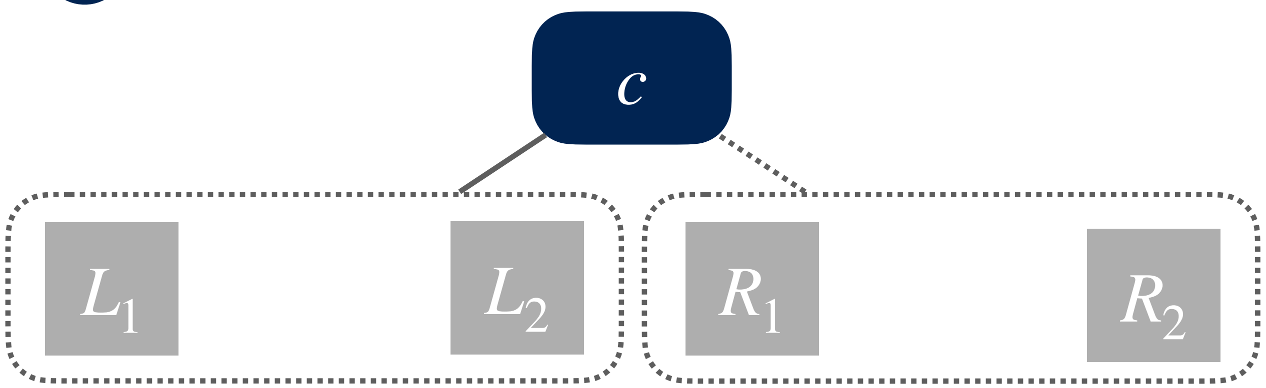
- Case ② (simmetric to ①)
- Case ③



(1a) c is redundant

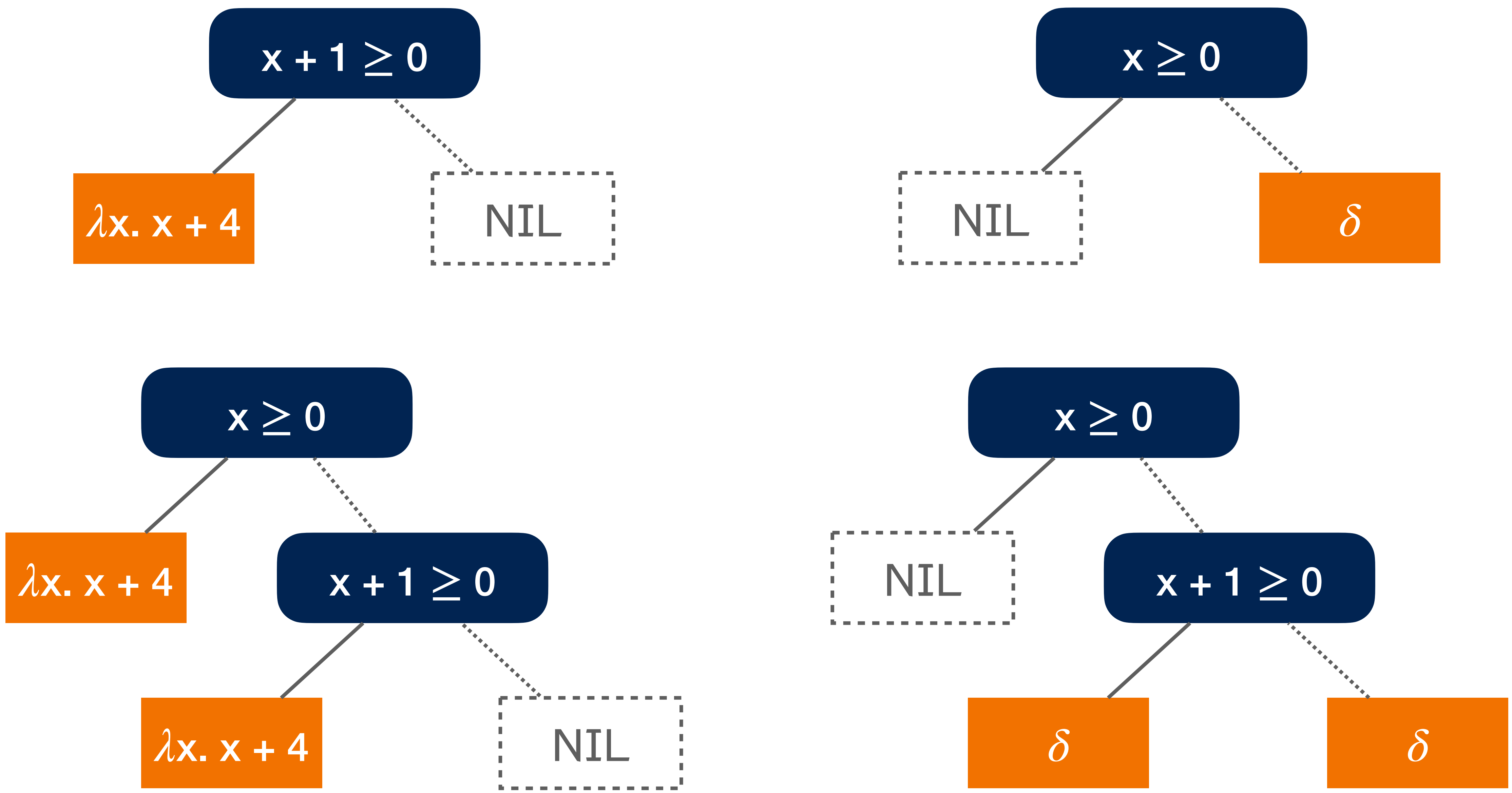
(1b) $\neg c$ is redundant

(1c) c is kept in t_1 and t_2



Piecewise-Defined Ranking Functions Abstract Domain

Tree Unification (continue) Example



Piecewise-Defined Ranking Functions Abstract 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 Ranking Functions Abstract 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. $\text{NIL} \curlyvee_A t \stackrel{\text{def}}{=} t$
   $t \curlyvee_A \text{NIL} \stackrel{\text{def}}{=} t$

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 Ranking Functions Abstract Domain

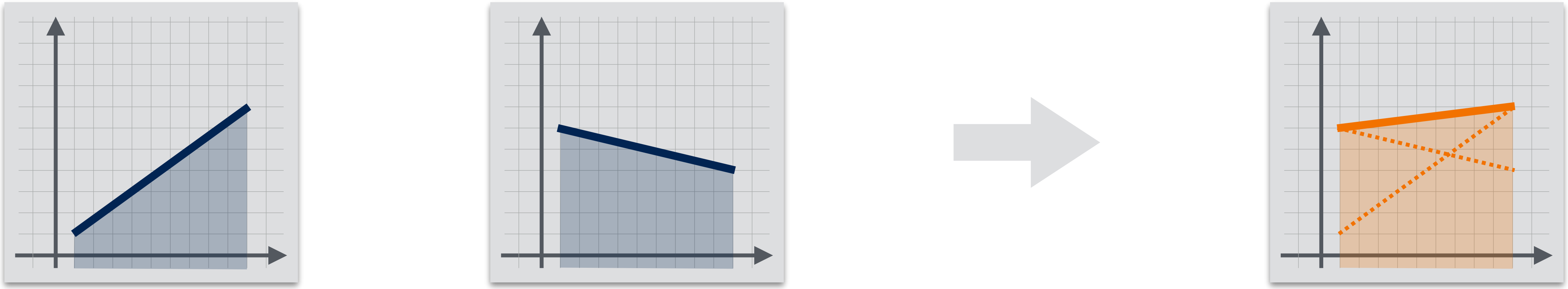
Join (continue)

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

$$f_1 \curlyvee_F [D]\ f_2 \stackrel{\text{def}}{=} \begin{cases} f & f \in \mathscr{F} \setminus \{ \bot_F , \top_F \} \\ \top_F & \text{otherwise} \end{cases}$$

where $f \stackrel{\text{def}}{=} \lambda\rho \in \gamma_D(D):\ \max(f_1(\ldots, \rho(X_i), \ldots), f_2(\ldots, \rho(X_i), \ldots))$

Example:



Piecewise-Defined Ranking Functions Abstract Domain

Join (continue)

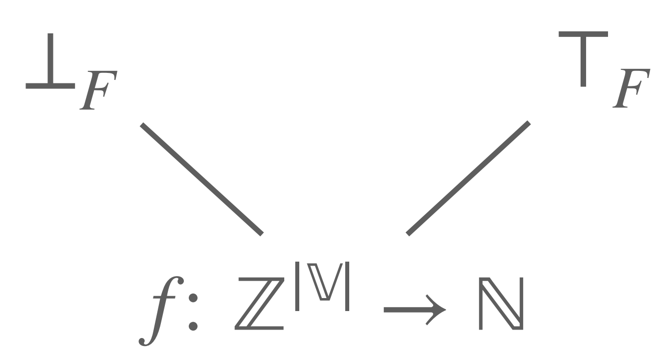
- **approximation join** $\curlyvee_F [D]$, where $D \in \mathscr{D}$:
  - between <u>defined</u> leaf nodes:

$$f_1 \curlyvee_F [D]\, f_2 \stackrel{\text{def}}{=} \begin{cases} f & f \in \mathscr{F} \setminus \{\, \bot_F \,, \top_F \,\} \\ \top_F & \text{otherwise} \end{cases}$$

$$\text{where } f \stackrel{\text{def}}{=} \lambda \rho \in \gamma_D(D) : \ \max(f_1(\ldots, \rho(X_i), \ldots), f_2(\ldots, \rho(X_i), \ldots))$$

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

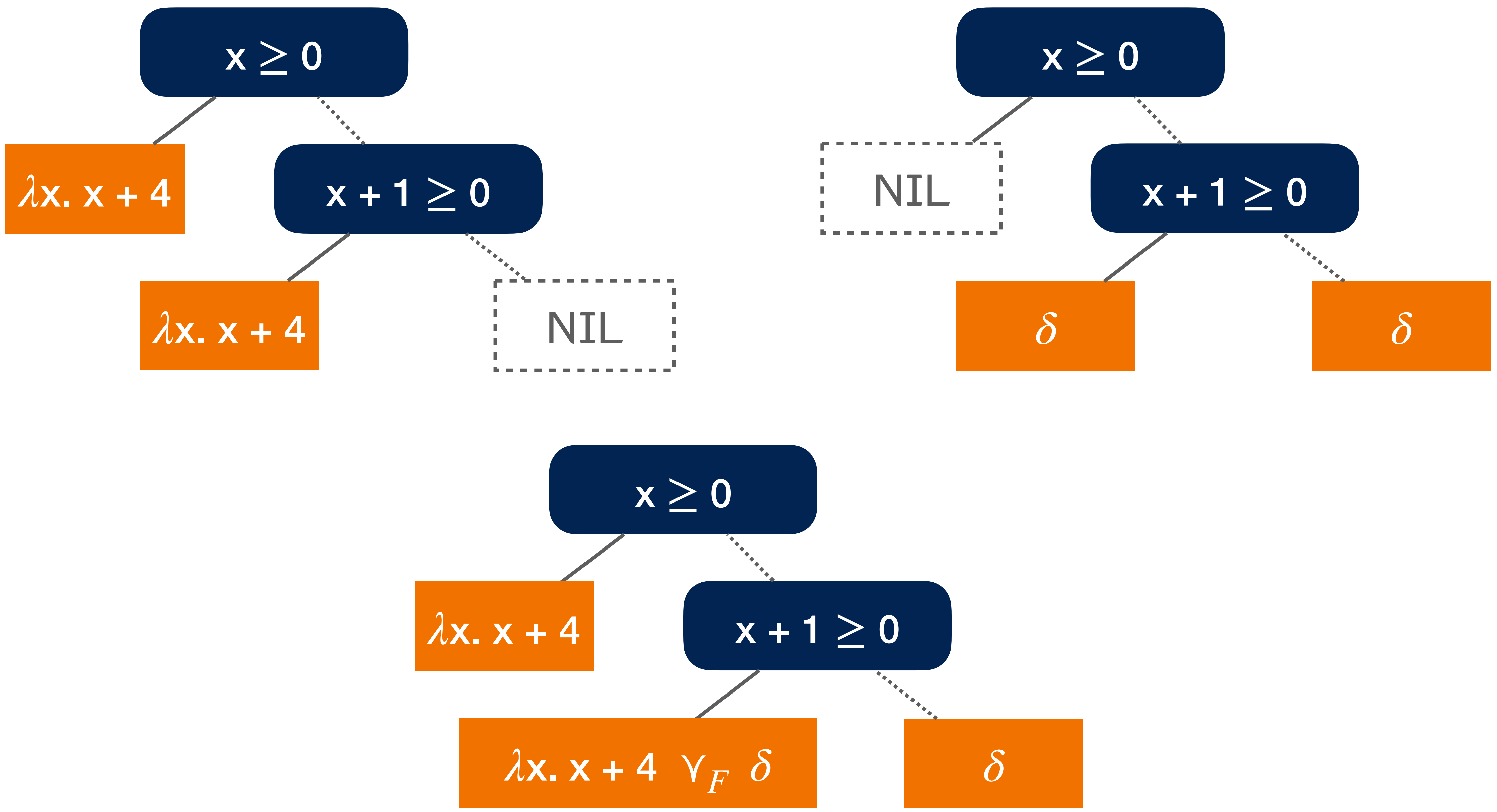
$$\begin{aligned} \bot_F \curlyvee_F [D]\, f &\stackrel{\text{def}}{=} \bot_F && f \in \mathscr{F} \setminus \{\, \top_F \,\} \\ f \curlyvee_F [D]\, \bot_F &\stackrel{\text{def}}{=} \bot_F && f \in \mathscr{F} \setminus \{\, \top_F \,\} \\ \top_F \curlyvee_F [D]\, f &\stackrel{\text{def}}{=} \top_F &&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



Piecewise-Defined Ranking Functions Abstract Domain

Join (continue)

Example



Piecewise-Defined Ranking Functions Abstract Domain

Join (continue)

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

$$f_1 \curlyvee_F [D]\, f_2 \overset{\text{def}}{=} \begin{cases} f & f \in \mathscr{F} \setminus \{ \bot_F, \top_F \} \\ \top_F & \text{otherwise} \end{cases}$$

where $f \overset{\text{def}}{=} \lambda \rho \in \gamma_D(D) : \max(f_1(\ldots, \rho(X_i), \ldots), f_2(\ldots, \rho(X_i), \ldots))$

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

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

$$\begin{array}{c} \top_F \\ | \\ f : \mathbb{Z}^{|\mathbb{V}|} \to \mathbb{N} \\ | \\ \bot_F \end{array}$$

Piecewise-Defined Ranking Functions Abstract 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

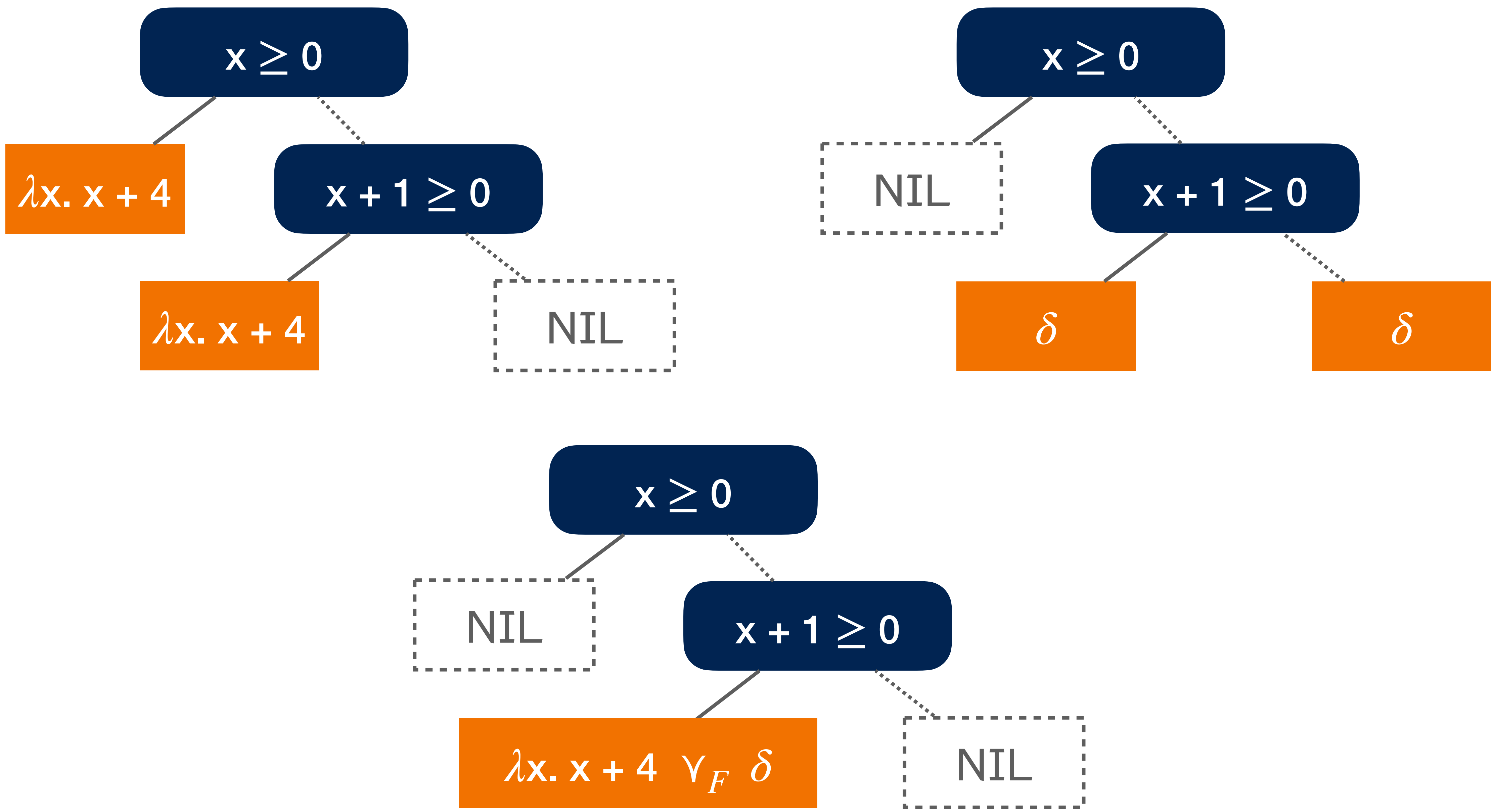
3. $\text{NIL} \curlyvee_A t \stackrel{\text{def}}{=} \text{NIL}$
   $t \curlyvee_A \text{NIL} \stackrel{\text{def}}{=} \text{NIL}$

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

Piecewise-Defined Ranking Functions Abstract Domain

Meet (continue)

Example



Piecewise-Defined Ranking Functions Abstract Domain

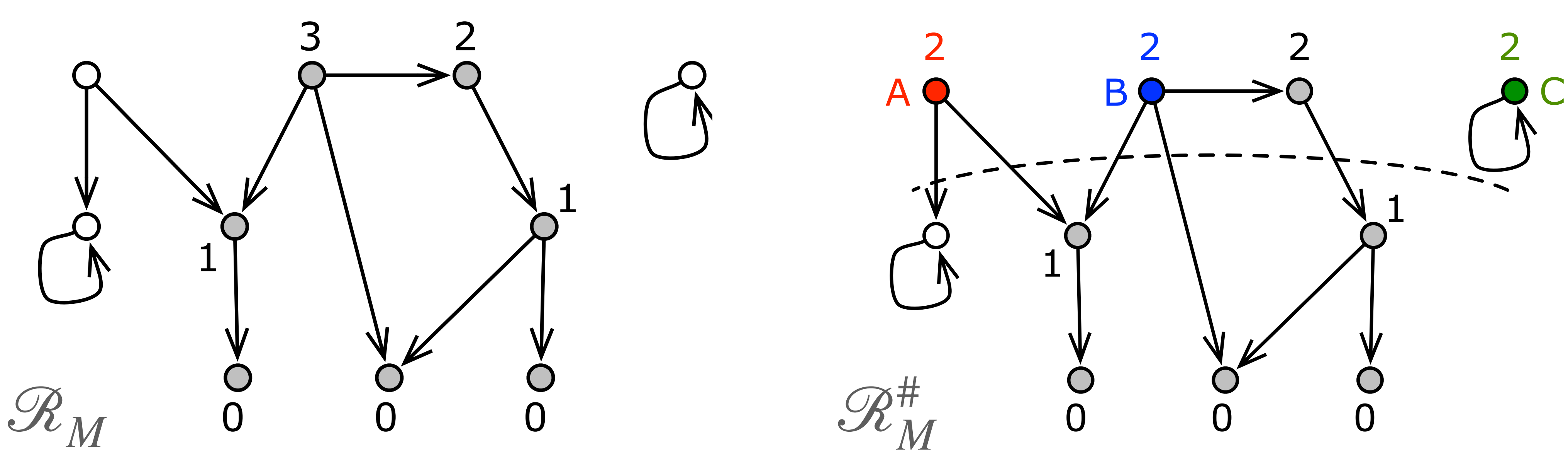
Widening

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

The prediction can (temporarily) be wrong!, i.e.,
- *under-approximates* the value of $\mathscr{R}_M$

and/or
- *over-approximates* the domain dom($\mathscr{R}_M$) of $\mathscr{R}_M$

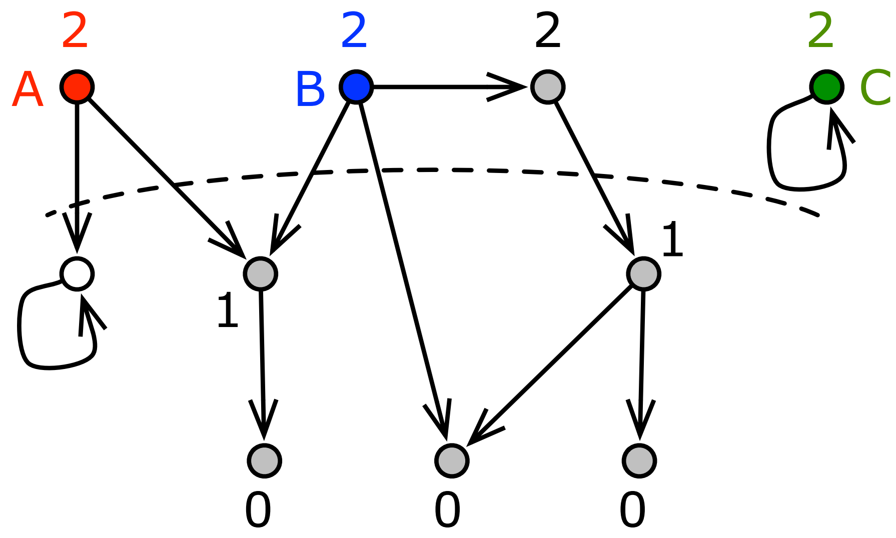
Example



Piecewise-Defined Ranking Functions Abstract 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 Ranking Functions Abstract Domain

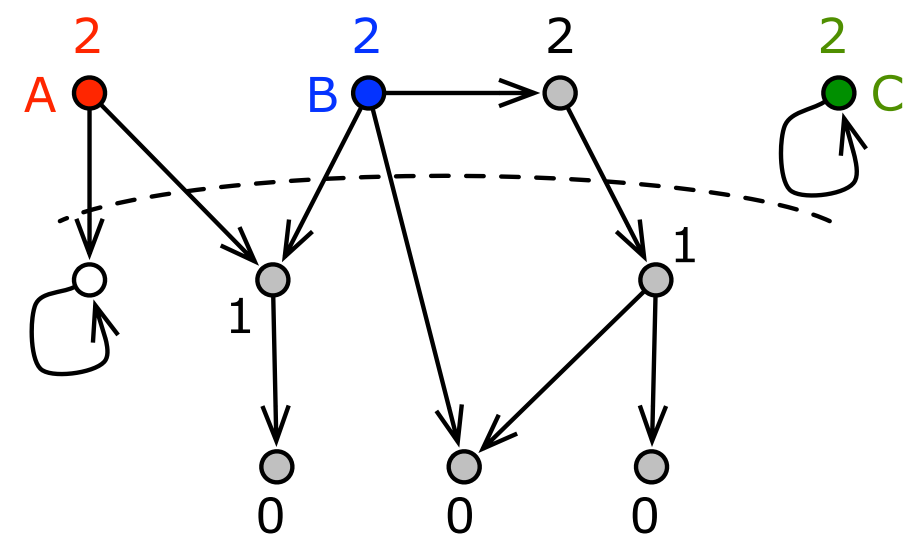
Widening (continue)

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 $\text{dom}(\gamma_A(\mathscr{R}_M^{\#n+1}(\ell)))\backslash\text{dom}(\mathscr{R}_M(\ell)) \subset \text{dom}(\gamma_A(\mathscr{R}_M^{\#n}(\ell)))\backslash\text{dom}(\mathscr{R}_M(\ell))$.

(see proof in [Urban15])



Piecewise-Defined Ranking Functions Abstract Domain

Widening (continue) — Check for Case A

Lemma

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

(see proof in [Urban15])

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 $\quad \bot_F \quad \rightarrow \quad f \quad \top_F$

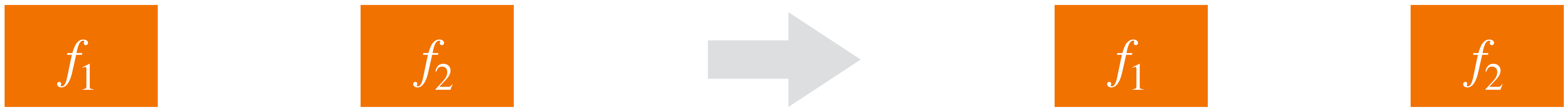
Piecewise-Defined Ranking Functions Abstract Domain

Widening (continue)

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:

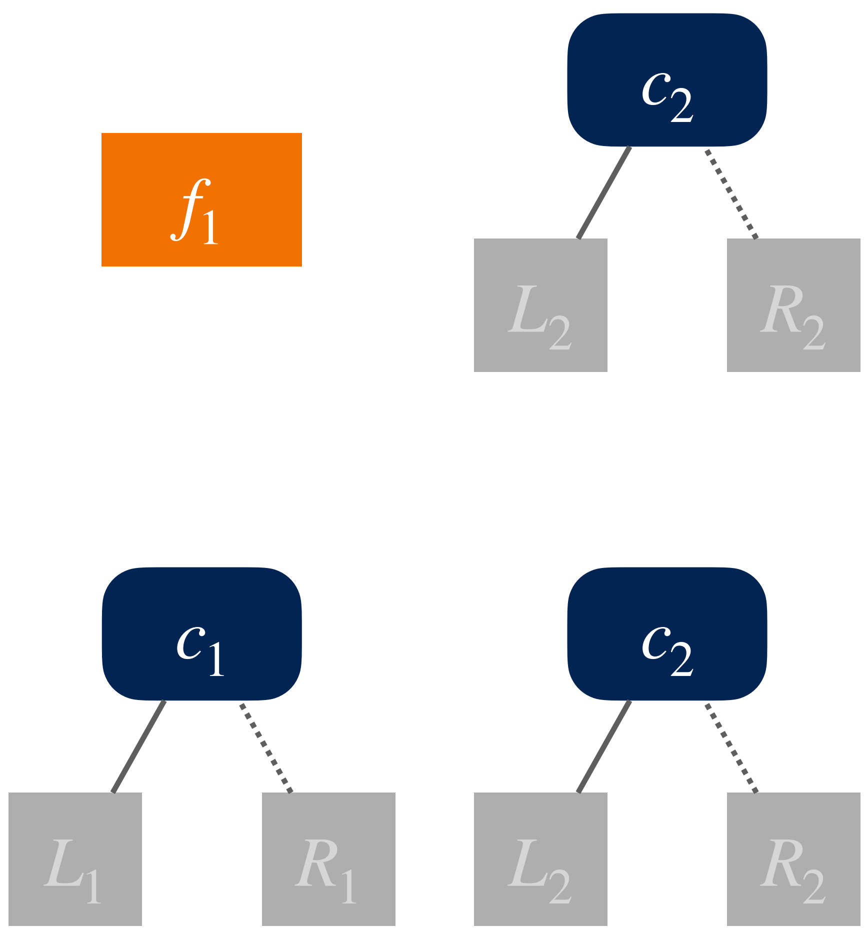


Piecewise-Defined Ranking Functions Abstract Domain

Widening (continue)

Domain Widening

- Case ①

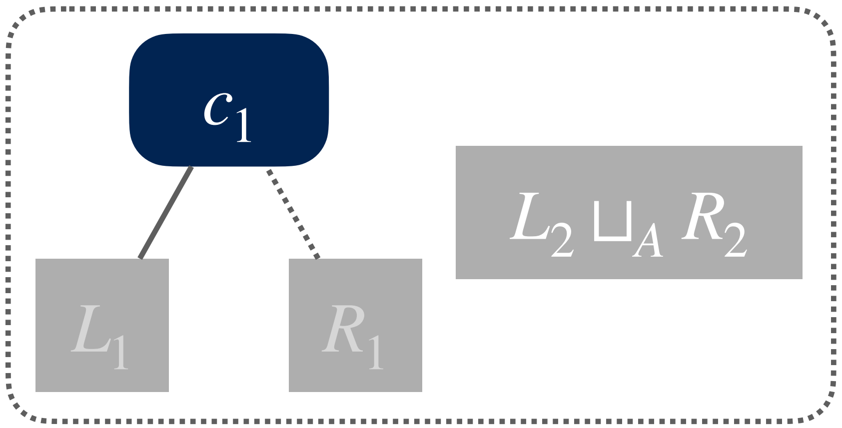


$c_2 \leq_C c_1$

(1a) c_2 is redundant

(1b) $\neg c_2$ is redundant

(1c) c_2 is removed from t_2

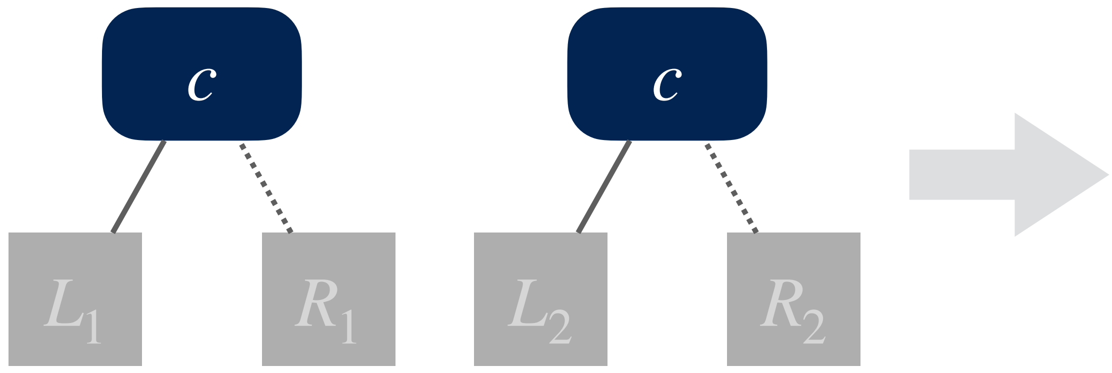


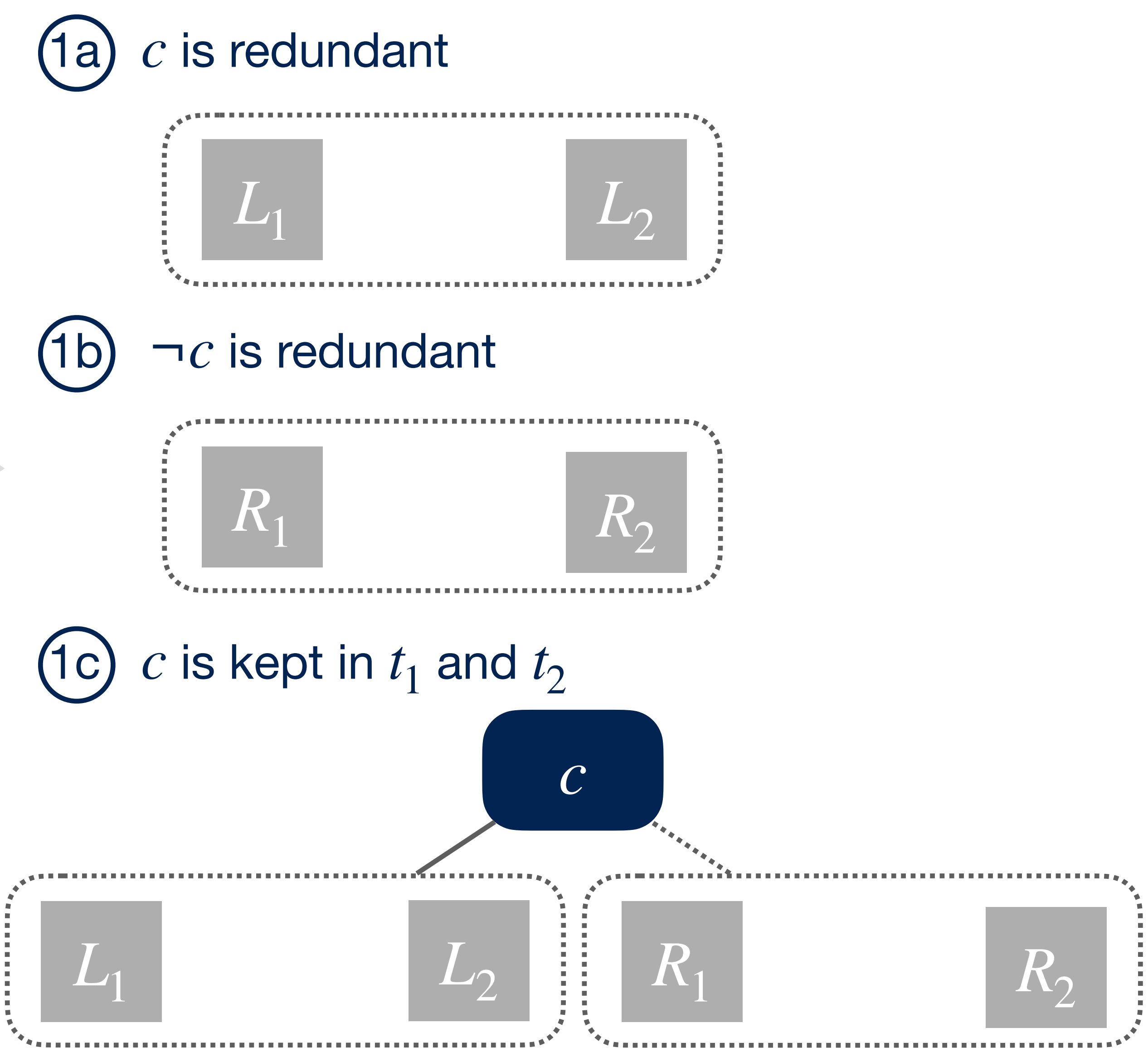
Piecewise-Defined Ranking Functions Abstract Domain

Widening (continue)

Domain Widening

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



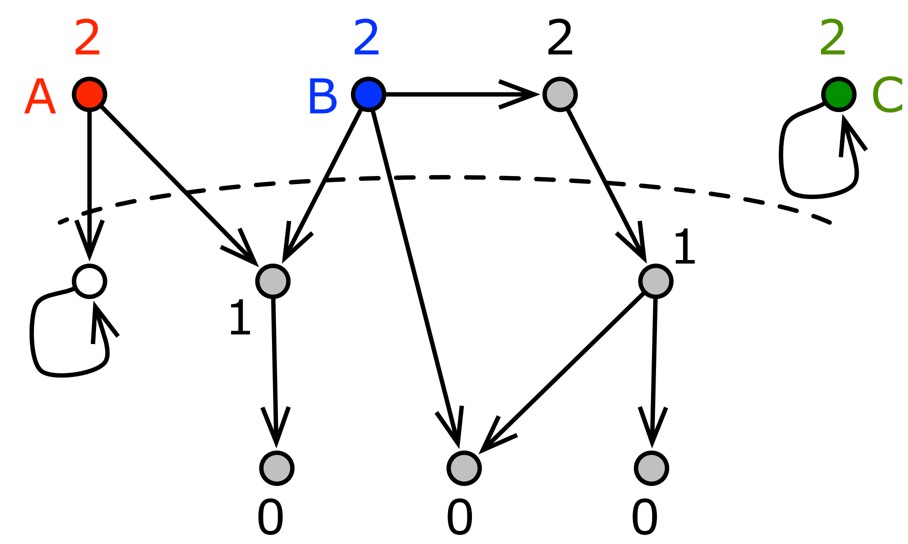


Piecewise-Defined Ranking Functions Abstract Domain

Widening (continue) Check for Case B or C

Lemma

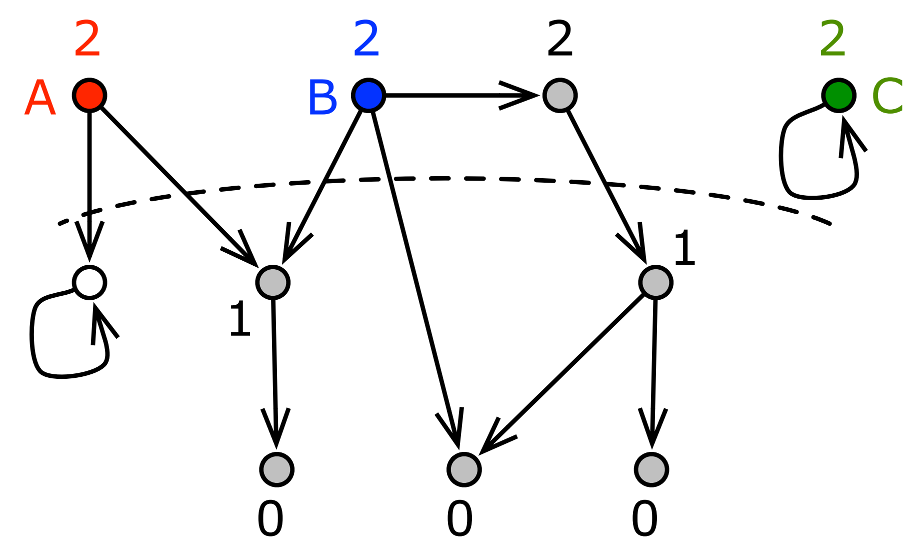
Let $\gamma_A(\mathscr{R}_M^{\#n}(\ell))(\overline{\rho}) < \mathscr{R}_M(\ell)(\overline{\rho})$ for some $\overline{\rho} \in \text{dom}(\mathscr{R}_M(\ell)) \cap \text{dom}(\gamma_A(\mathscr{R}_M^{\#n})(\ell))$ (case B). Then, there exists $\rho \in \text{dom}(\gamma_A(\mathscr{R}_M^{\#n+1}(\ell)))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



Piecewise-Defined Ranking Functions Abstract Domain

Widening (continue)

Check for Case B or C



Lemma

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

(see proof in [Urban15])

Piecewise-Defined Ranking Functions Abstract 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 $\top_F$

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

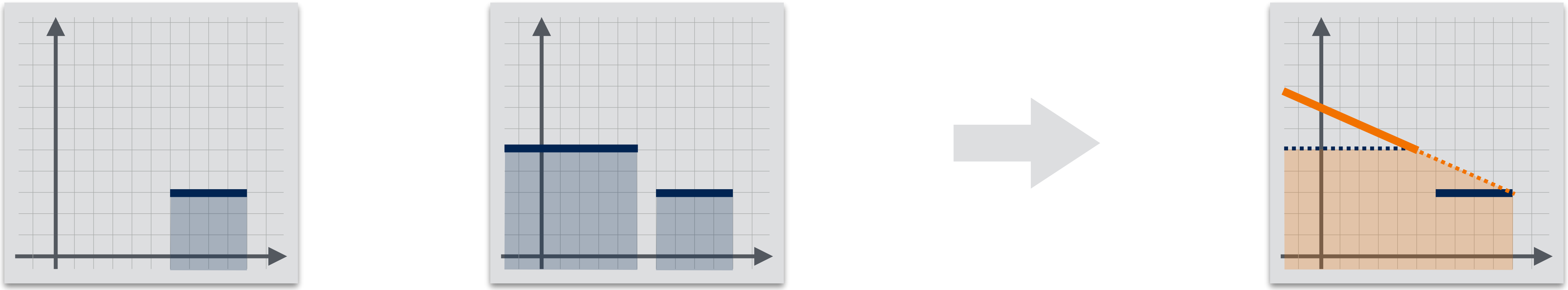
Piecewise-Defined Ranking Functions Abstract Domain

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:

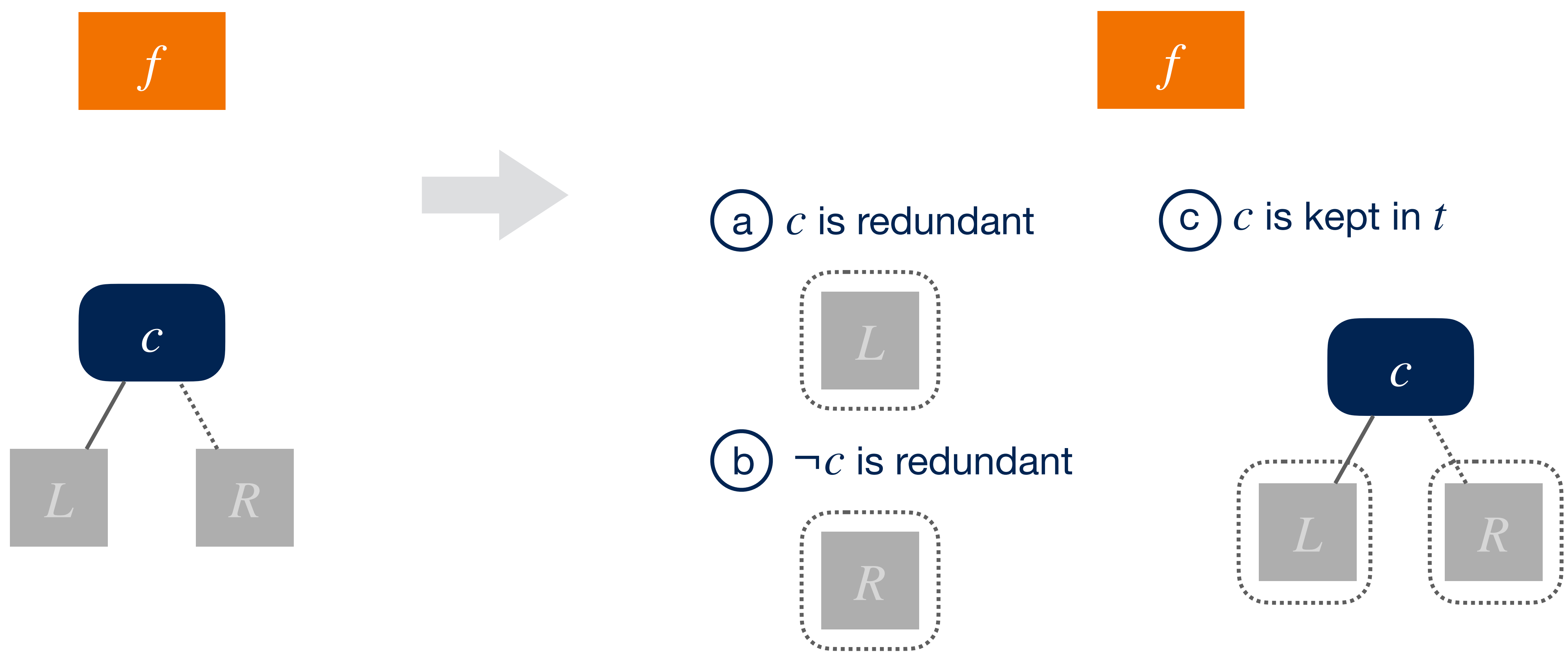


Piecewise-Defined Ranking Functions Abstract Domain

Tree Pruning

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

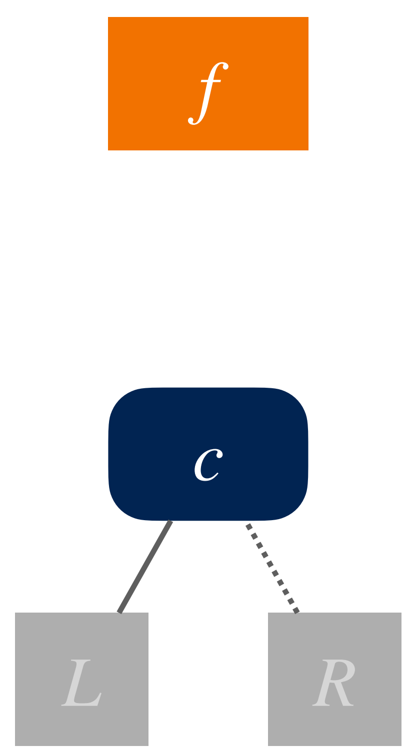
- Base case ($J = \emptyset$)



Piecewise-Defined Ranking Functions Abstract Domain

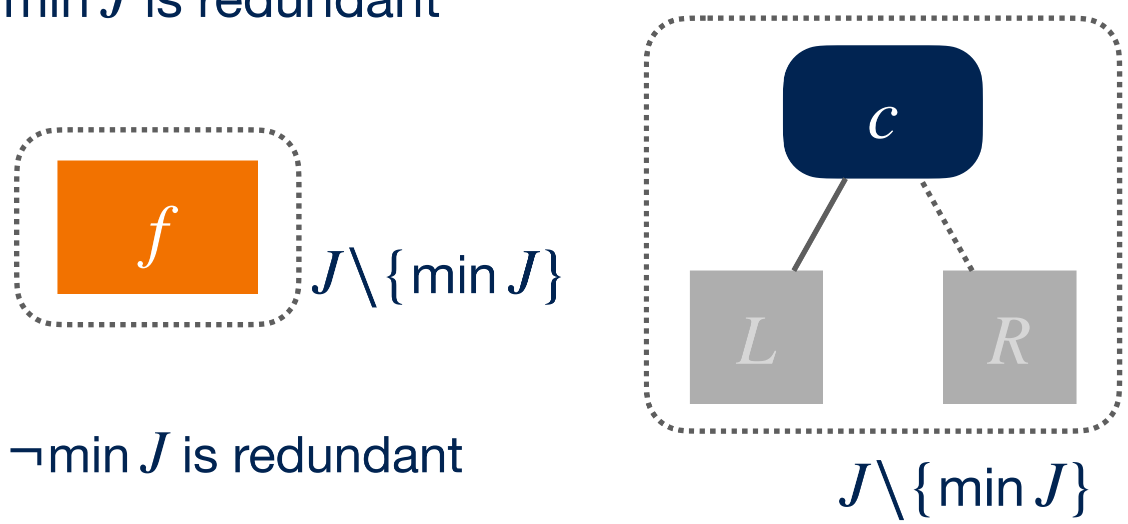
Tree Pruning (continue)

- Case ①



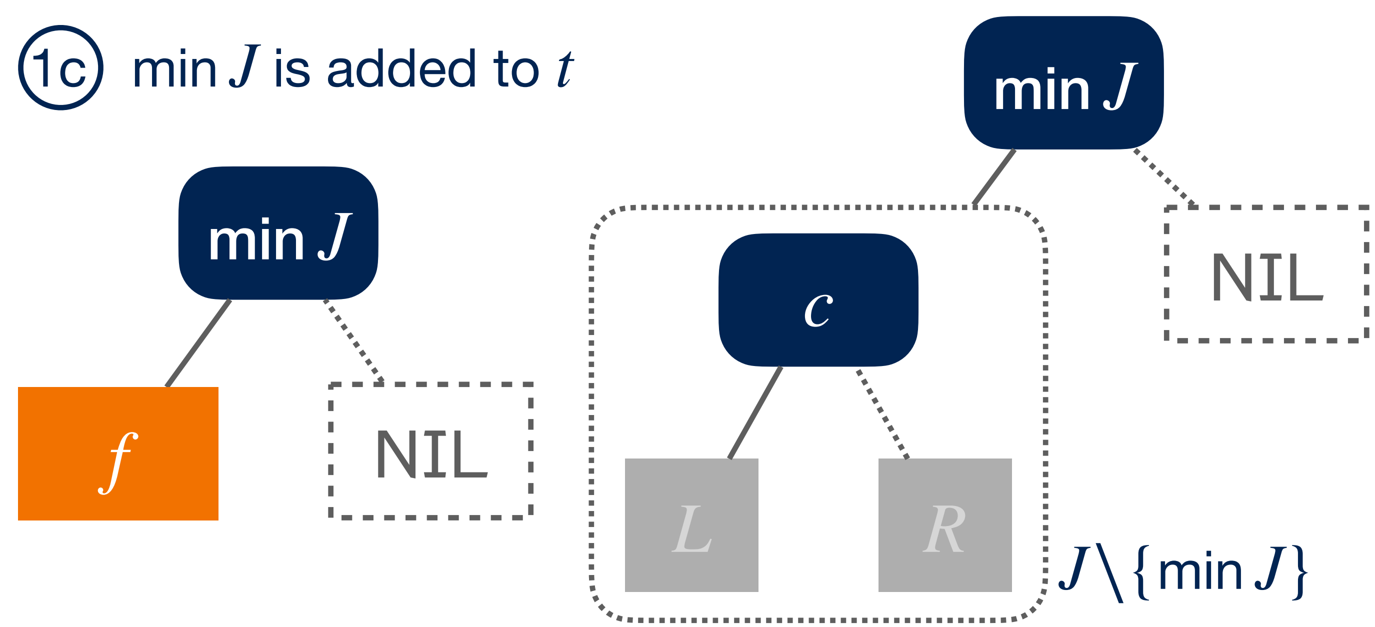
$\min J \leq_C c$

(1a) $\min J$ is redundant



(1b) $\neg\min J$ is redundant

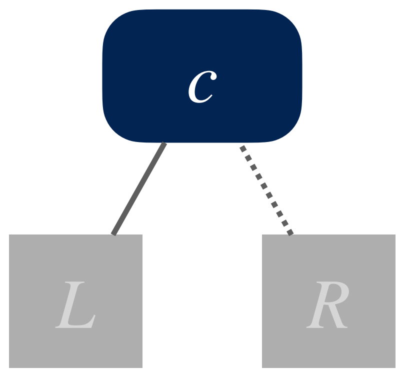
(1c) $\min J$ is added to t



Piecewise-Defined Ranking Functions Abstract Domain

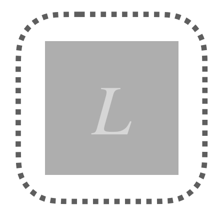
Tree Pruning (continue)

- Case ②



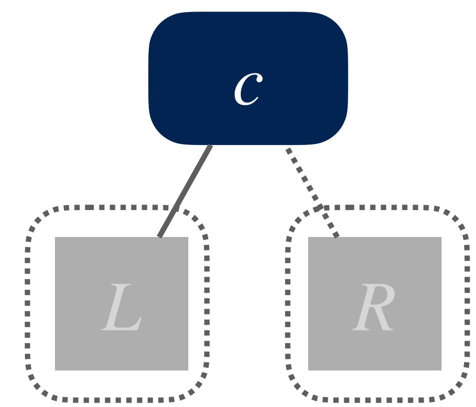
$c \leq_C \min J$

② a c is redundant

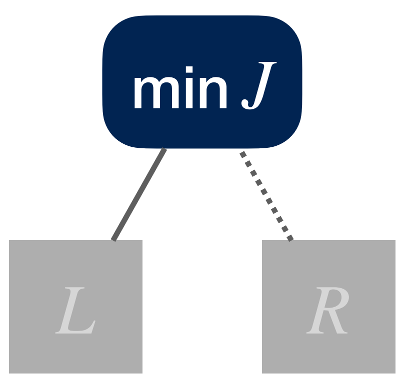


② b $\neg c$ is redundant

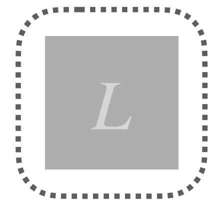
② c c is kept in t



- Case ③



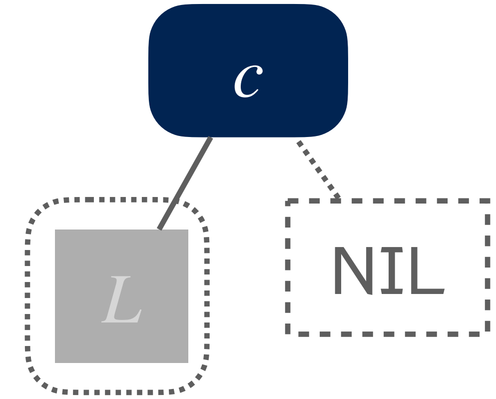
③ a $\min J$ is redundant



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

③ b $\neg \min J$ is redundant

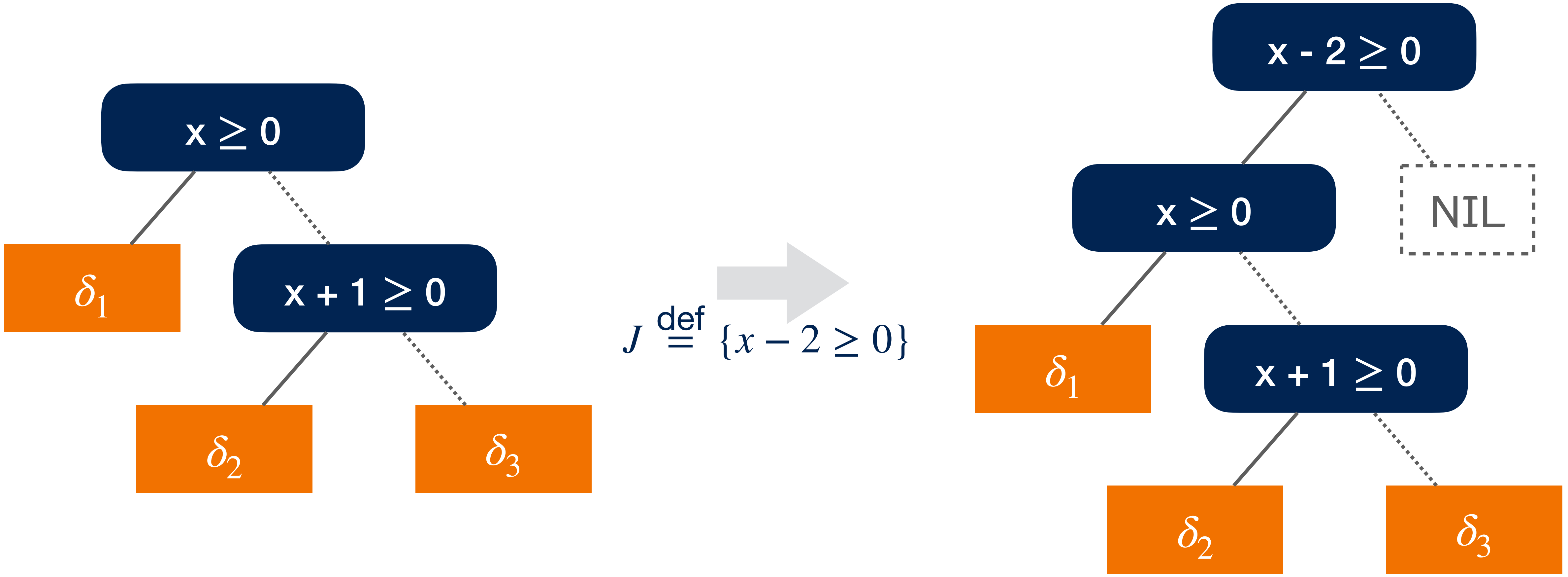
③ c $\min J$ is kept in t



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

Piecewise-Defined Ranking Functions Abstract Domain

Tree Pruning (continue) Example



Piecewise-Defined Ranking Functions Abstract 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 <u>defined</u> leaf nodes

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

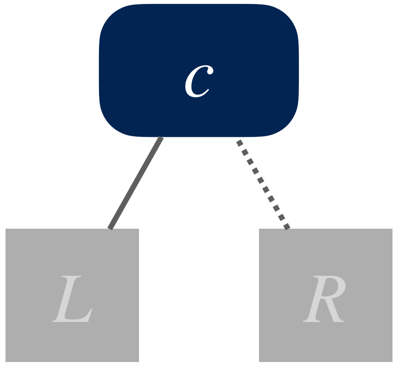
where $\bar{f}(\ldots, X_i, X, \ldots) \stackrel{\text{def}}{=} \max\{f(\ldots, \rho(X_i), v, \ldots) + 1 \mid \rho \in \gamma_D(R) \wedge v \in E[\![e]\!]\rho\}$
and $R \stackrel{\text{def}}{=} \overleftarrow{\text{ASSIGN}}_D[\![X \leftarrow e]\!]D$

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 Ranking Functions Abstract Domain

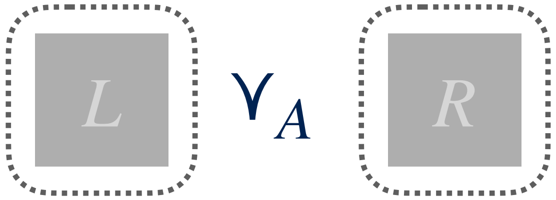
Assignments

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

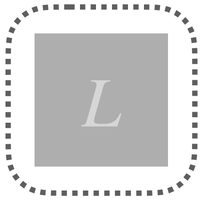
- 

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 $I = J = \emptyset$

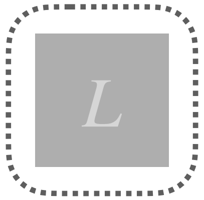


case ② $I = \emptyset \wedge \bot_C \in J$



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

case ④

1. perform **tree pruning** on 

and 

2. join the results with $\curlyvee_A$

Piecewise-Defined Ranking Functions Abstract Domain

Tests

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

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

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

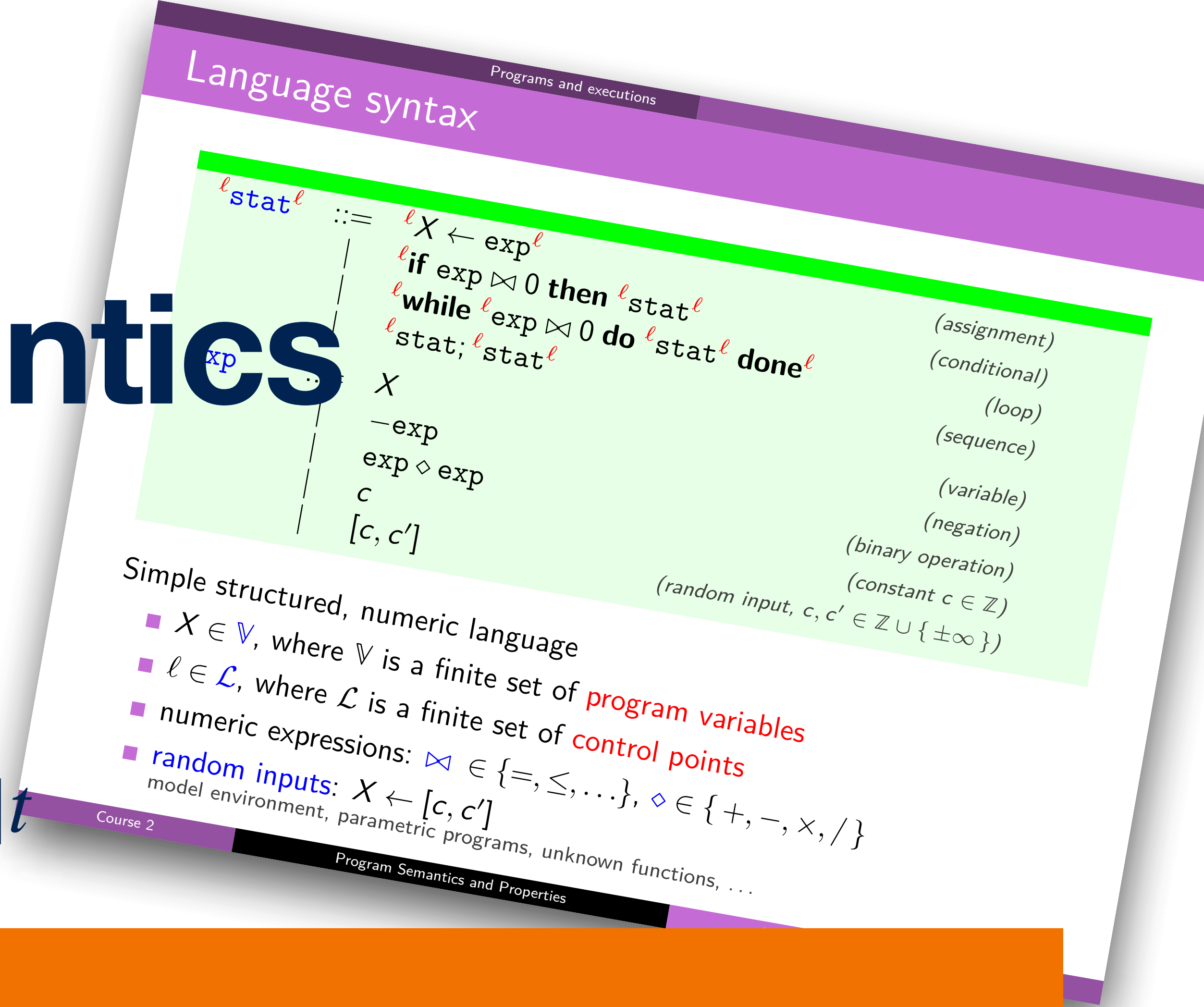
2. **Convert e into a set J of linear constraints** *in canonical form*

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

3. Perform **tree pruning** with J

Abstract Definite Termination Semantics

For each program instruction stat, we define a transformer $\mathcal{R}^{\#}_M[\![\text{stat}]\!] : \mathcal{A} \rightarrow \mathcal{A}$:



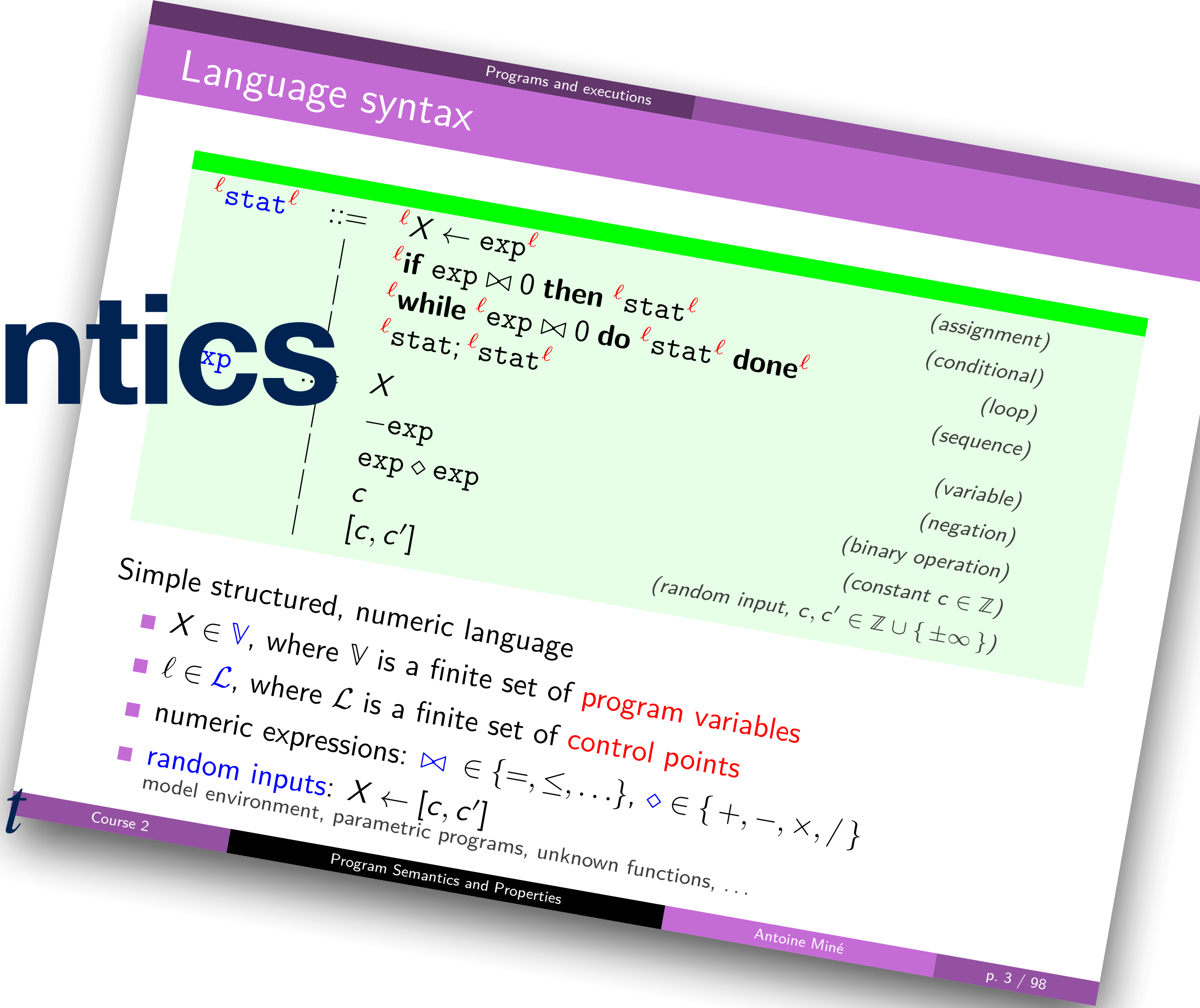
- $\mathcal{R}^{\#}_M[\![{}^{\ell}X \leftarrow e]\!]t \stackrel{\text{def}}{=} \overleftarrow{\text{ASSIGN}}_A[\![X \leftarrow e]\!]t$

Lemma (Soundness)

$$\mathcal{R}_M[\![{}^{\ell}X \leftarrow e]\!]\gamma_A(t) \preccurlyeq \gamma_A(\mathcal{R}^{\#}_M[\![{}^{\ell}X \leftarrow e]\!]t)$$

(see proof in [Urban15])

Abstract Definite Termination Semantics



For each program instruction stat, we define a transformer $\mathcal{R}^\#_M[\![\text{stat}]\!] : \mathcal{A} \rightarrow \mathcal{A}$:

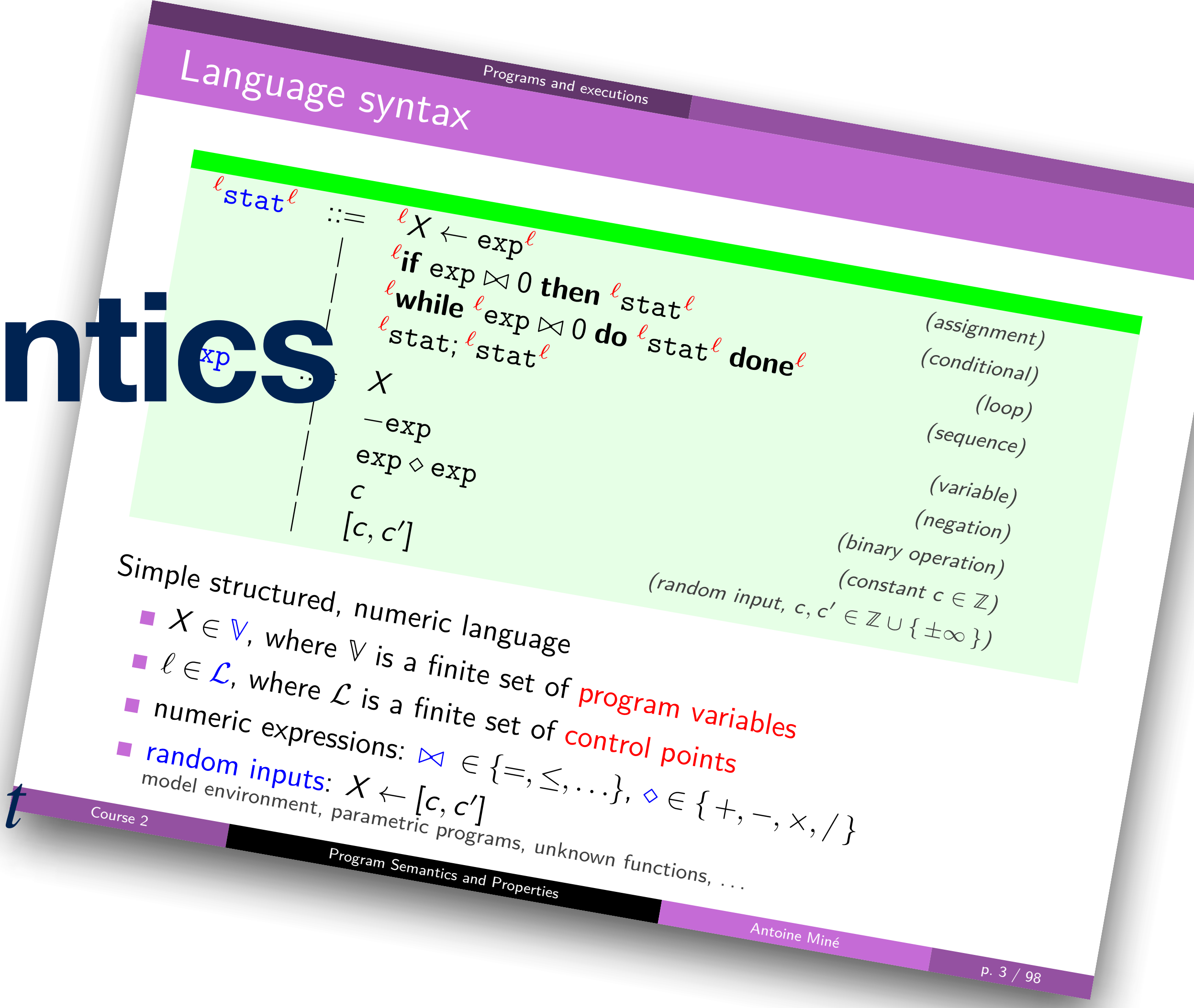
- $\mathcal{R}^\#_M[\![^{\ell} X \leftarrow e]\!]t \stackrel{\text{def}}{=} \overleftarrow{\text{ASSIGN}}_A[\![X \leftarrow e]\!]t$
- $\mathcal{R}^\#_M[\![\textbf{if}\ ^{\ell}\ e \bowtie 0\ \textbf{then}\ s]\!]t \stackrel{\text{def}}{=}$ $\text{FILTER}_A[\![e \bowtie 0]\!](\mathcal{R}^\#_M[\![s]\!]t) \curlyvee_T \text{FILTER}_A[\![e \not\bowtie 0]\!]t$

Lemma (Soundness)

$$\mathcal{R}_M[\![\textbf{if}\ ^{\ell}\ e \bowtie 0\ \textbf{then}\ s]\!]\gamma_A(t) \preccurlyeq \gamma_A(\mathcal{R}^\#_M[\![\textbf{if}\ ^{\ell}\ e \bowtie 0\ \textbf{then}\ s]\!]t)$$

(see proof in [Urban15])

Abstract Definite Termination Semantics



For each program instruction stat, we define a transformer $\mathcal{R}_M^{\#}[\![\text{stat}]\!] : \mathcal{A} \to \mathcal{A}$:

- $\mathcal{R}_M^{\#}[\![^{\ell}X \leftarrow e]\!]t \stackrel{\text{def}}{=} \text{ASSIGN}_A[\![X \leftarrow e]\!]t$

- $\mathcal{R}_M^{\#}[\![\textbf{if } ^{\ell}\ e \bowtie 0 \textbf{ then } s]\!]t \stackrel{\text{def}}{=}$
  $\text{FILTER}_A[\![e \bowtie 0]\!](\mathcal{R}_M^{\#}[\![s]\!]t) \curlyvee_T \text{FILTER}_A[\![e \not\bowtie 0]\!]t$

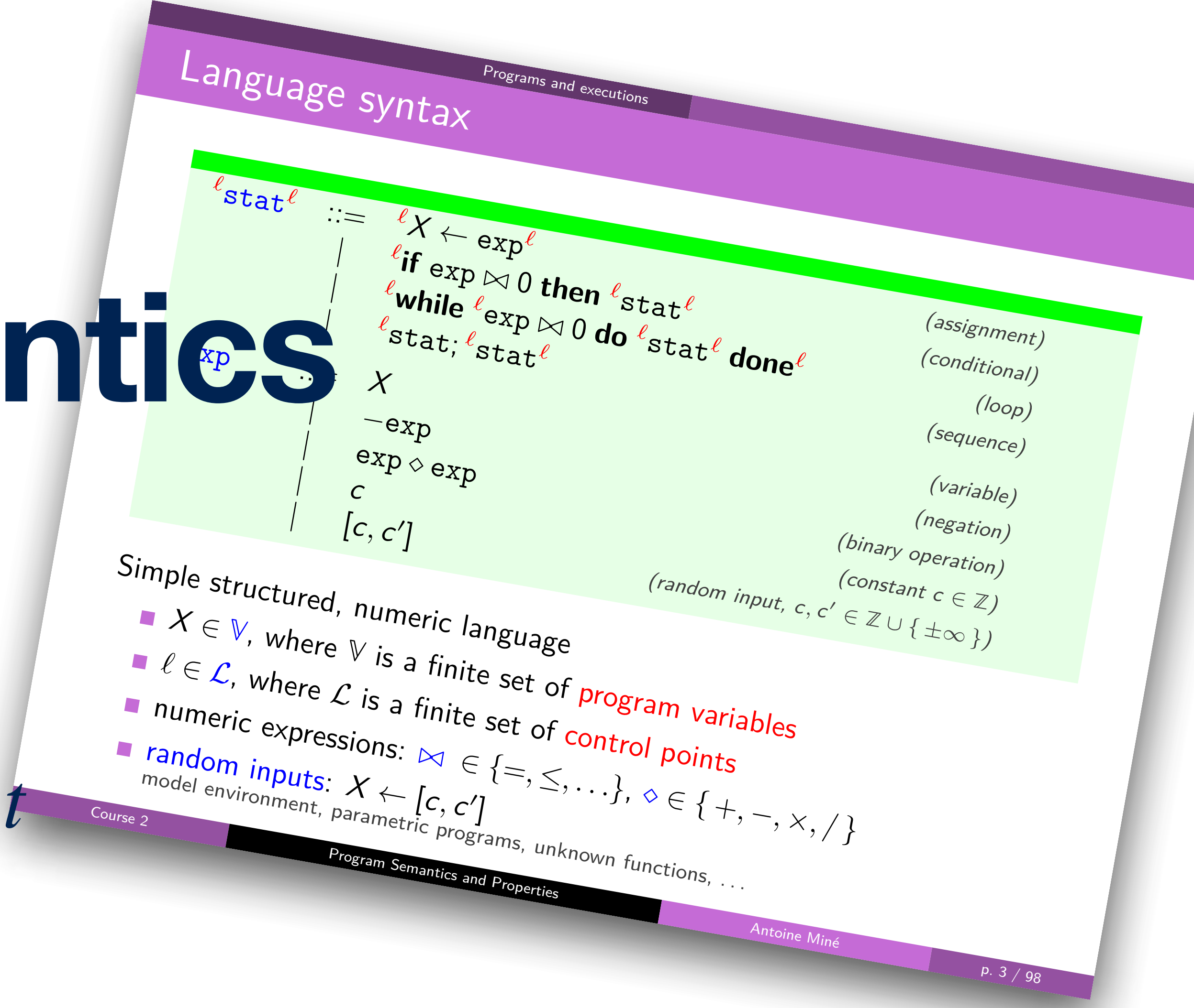
- $\mathcal{R}_M^{\#}[\![\textbf{while } ^{\ell}\ e \bowtie 0 \textbf{ do } s \textbf{ done}]\!]t \stackrel{\text{def}}{=} \text{lfp}^{\#}\overline{F}_M^{\#}$
  where $\overline{F}_M^{\#}(x) \stackrel{\text{def}}{=} \text{FILTER}_A[\![e \bowtie 0]\!](\mathcal{R}_M^{\#}[\![s]\!]x) \curlyvee_T \text{FILTER}_A[\![e \not\bowtie 0]\!](t)$

Lemma (Soundness)

$\mathcal{R}_M[\![\textbf{while } ^{\ell}\ e \bowtie 0 \textbf{ do } s \textbf{ done}]\!]\gamma_A(t) \preccurlyeq \gamma_A(\mathcal{R}_M^{\#}[\![\textbf{while } ^{\ell}\ e \bowtie 0 \textbf{ do } s \textbf{ done}]\!]t)$

(see proof in [Urban15])

Abstract Definite Termination Semantics



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

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

- $\mathcal{R}^{\#}_{M}[\![\textbf{if } {}^{\ell}\, e \bowtie 0 \textbf{ then } s]\!]t \stackrel{\text{def}}{=}$ $\text{FILTER}_{A}[\![e \bowtie 0]\!](\mathcal{R}^{\#}_{M}[\![s]\!]t) \curlyvee_{T} \text{FILTER}_{A}[\![e \not\bowtie 0]\!]t$

- $\mathcal{R}^{\#}_{M}[\![\textbf{while } {}^{\ell}\, e \bowtie 0 \textbf{ do } s \textbf{ done}]\!]t \stackrel{\text{def}}{=} \text{lfp}^{\#}\overline{F}^{\#}_{M}$
  where $\overline{F}^{\#}_{M}(x) \stackrel{\text{def}}{=} \text{FILTER}_{A}[\![e \bowtie 0]\!](\mathcal{R}^{\#}_{M}[\![s]\!]x) \curlyvee_{T} \text{FILTER}_{A}[\![e \not\bowtie 0]\!](t)$

- $\mathcal{R}^{\#}_{M}[\![s_1; s_2]\!]t \stackrel{\text{def}}{=} \mathcal{R}^{\#}_{M}[\![s_1]\!](\mathcal{R}^{\#}_{M}[\![s_2]\!]t)$

Abstract Definite Termination Semantics

Definition

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

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

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

Theorem (Soundness)

$$\mathcal{R}_{M}[\![\text{stat}^{\ell}]\!] \preccurlyeq \gamma_A(\mathcal{R}^{\#}_{M}[\![\text{stat}^{\ell}]\!])$$

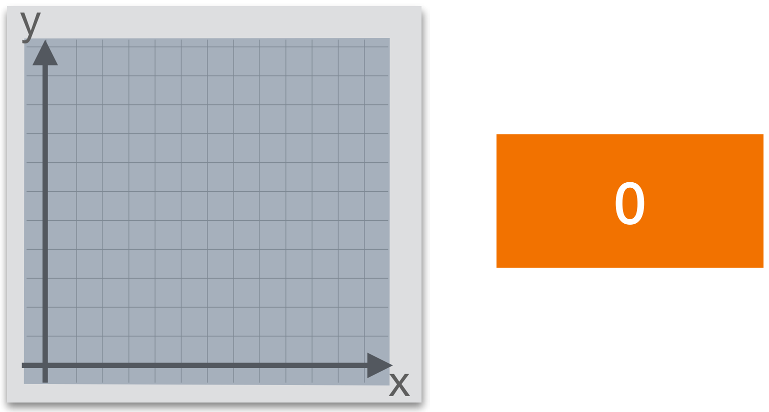
Corollary (Soundness)

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

Abstract Definite Termination Semantics

Example

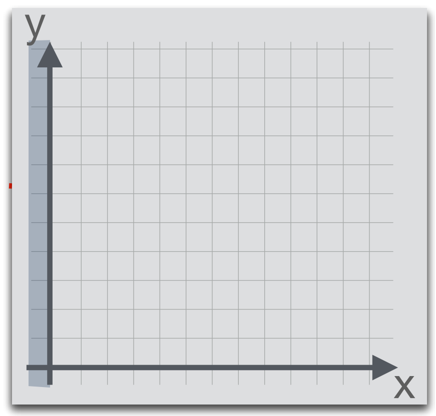
[1]x ← [-∞, +∞]
[2]y ← [-∞, +∞]
while [3](x > 0) **do**
[4]x ← x - y
od[5]

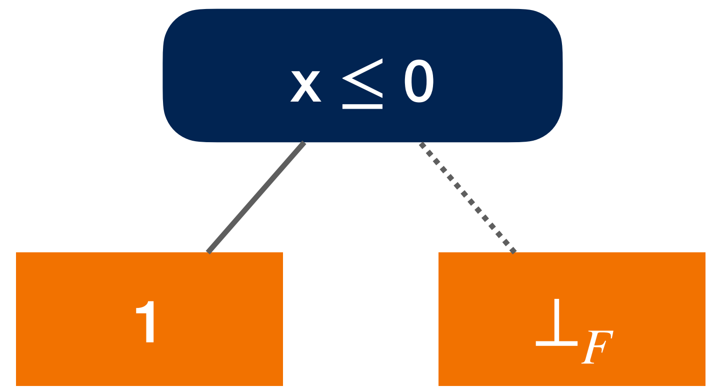


Abstract Definite Termination Semantics

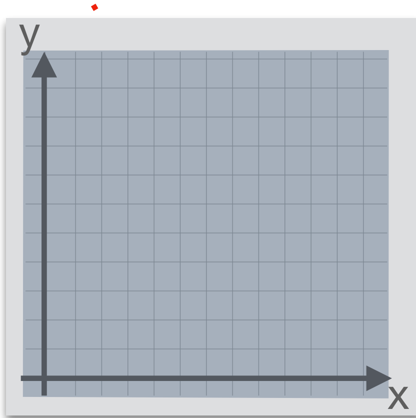
Example

[1]x ← [-∞, +∞]
[2]y ← [-∞, +∞]
while [3](x > 0) **do**
[4]x ← x - y
od[5]





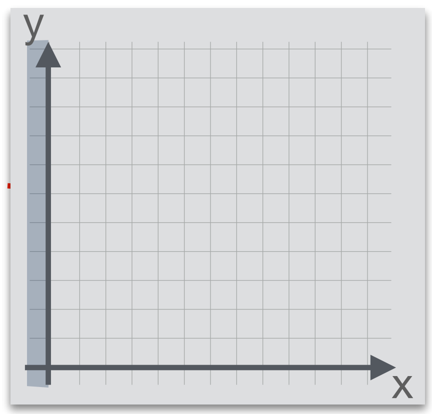
$\text{FILTER}_A[\![x \leq 0]\!]$

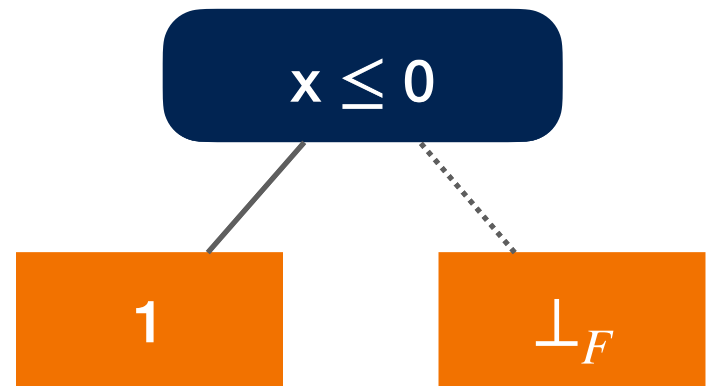


Abstract Definite Termination Semantics

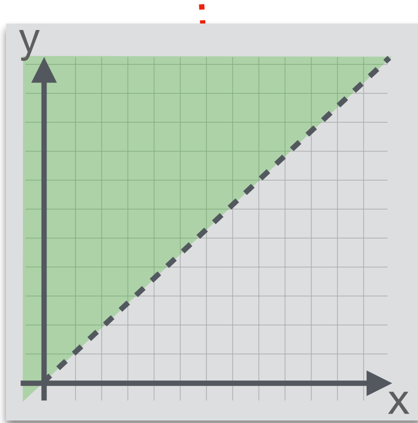
Example

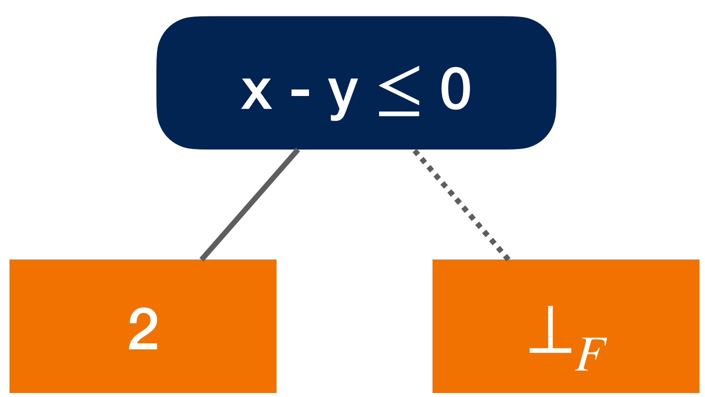
[1]x ← [-∞, +∞]
[2]y ← [-∞, +∞]
while [3](x > 0) **do**
  [4]x ← x - y
od[5]





$\overleftarrow{\text{ASSIGN}}_A[\![x \leftarrow x - y]\!]$

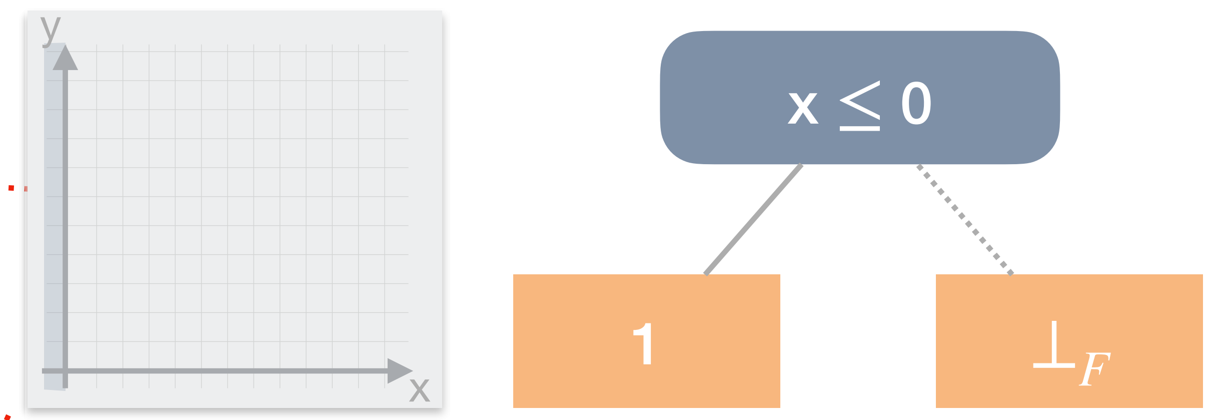


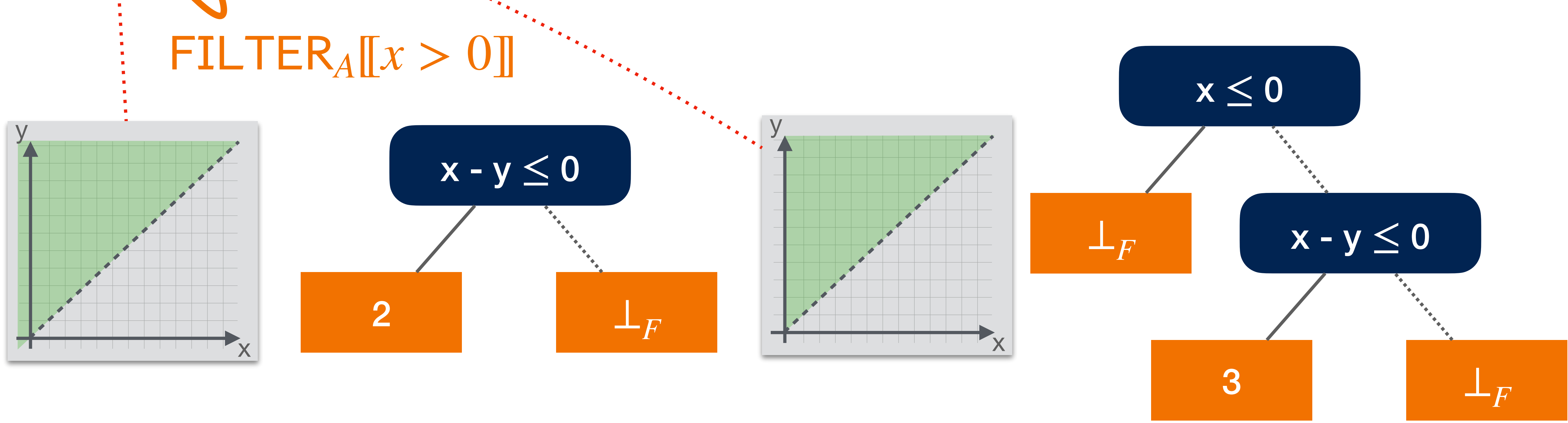


Abstract Definite Termination Semantics

Example

```
1x ← [-∞, +∞]
2y ← [-∞, +∞]
 while 3(x > 0) do
        4x ← x - y
od5
```



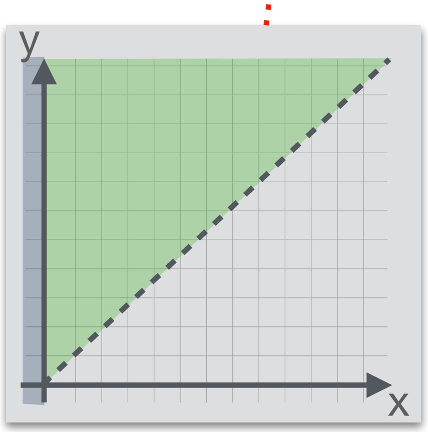


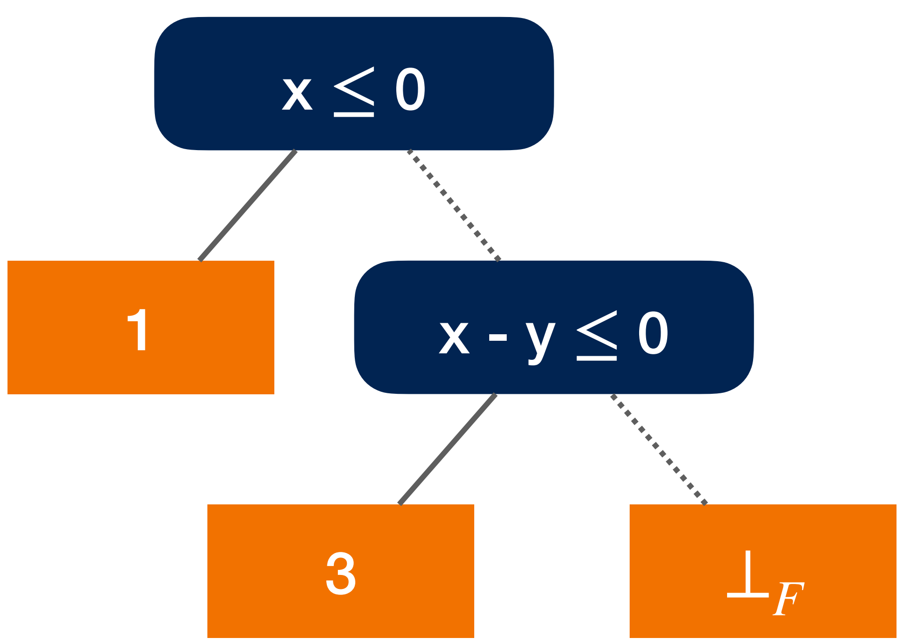
Abstract Definite Termination Semantics

Example

```
¹x ← [-∞, +∞]
²y ← [-∞, +∞]
 while ³(x > 0) do
       ⁴x ← x - y
od⁵
```

γ_A





Abstract Definite Termination Semantics

Example

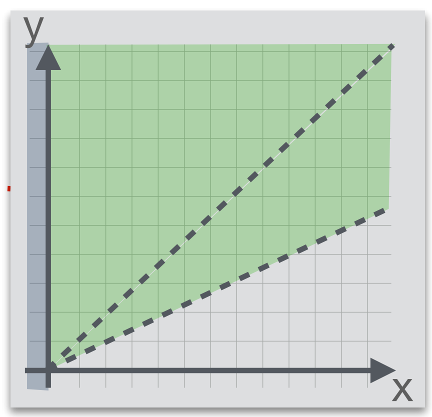
$^{1}x \leftarrow [-\infty, +\infty]$

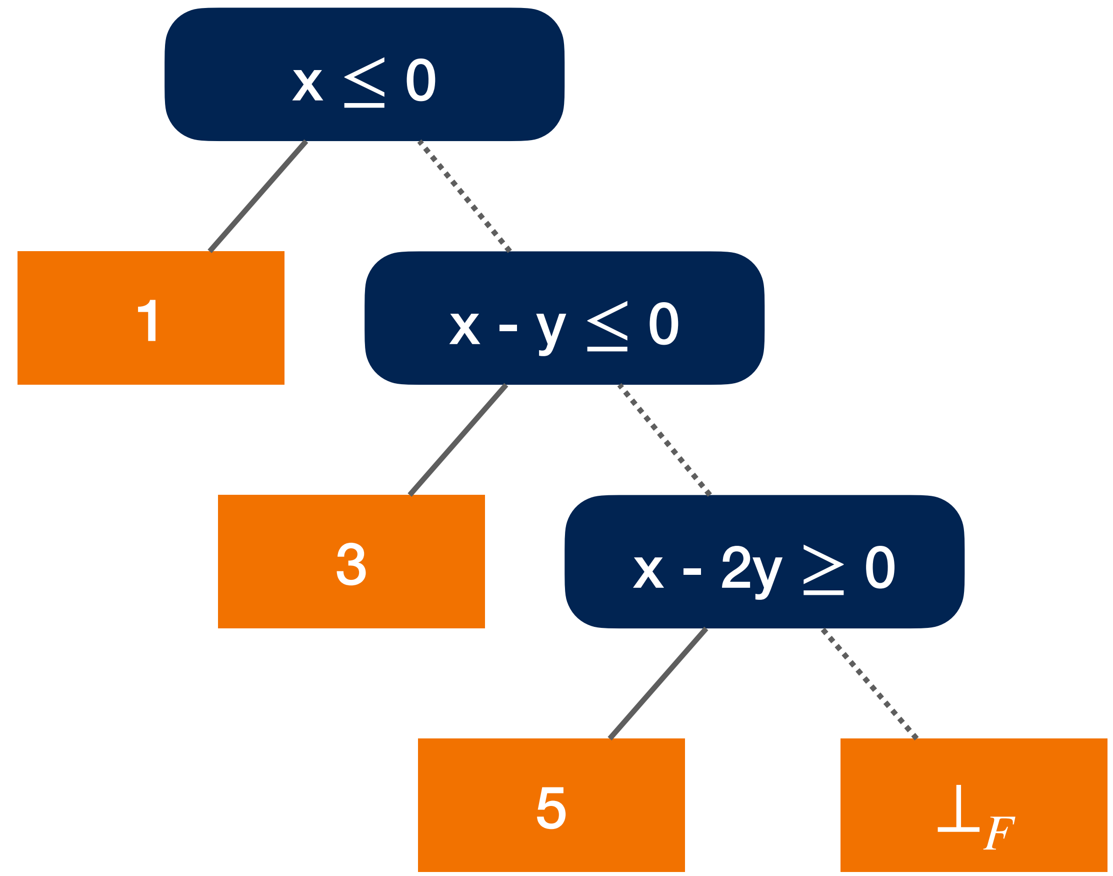
$^{2}y \leftarrow [-\infty, +\infty]$

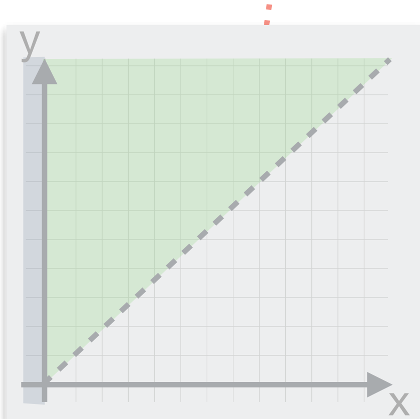
while $^{3}(x > 0)$ **do**

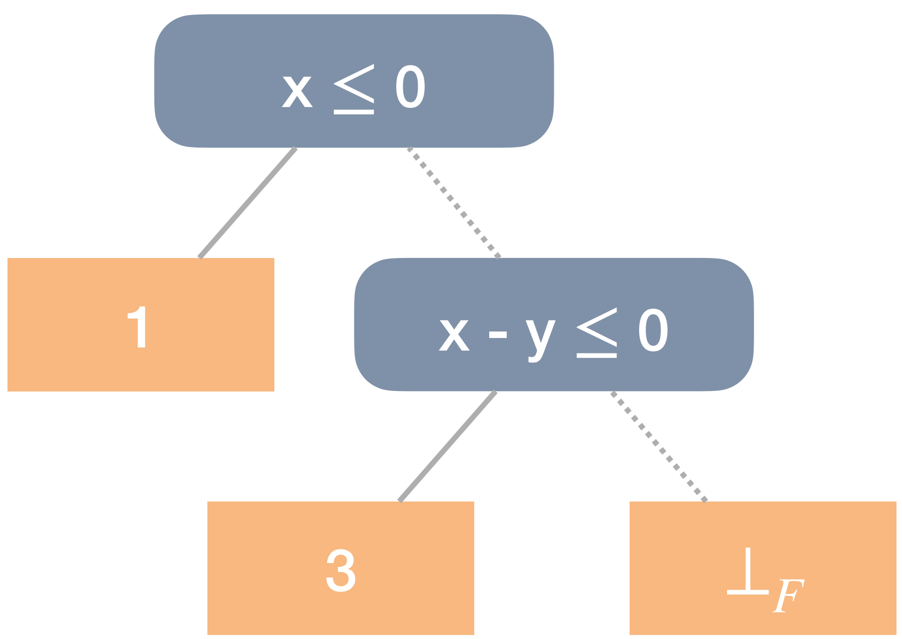
$\quad ^{4}x \leftarrow x - y$

od^{5}





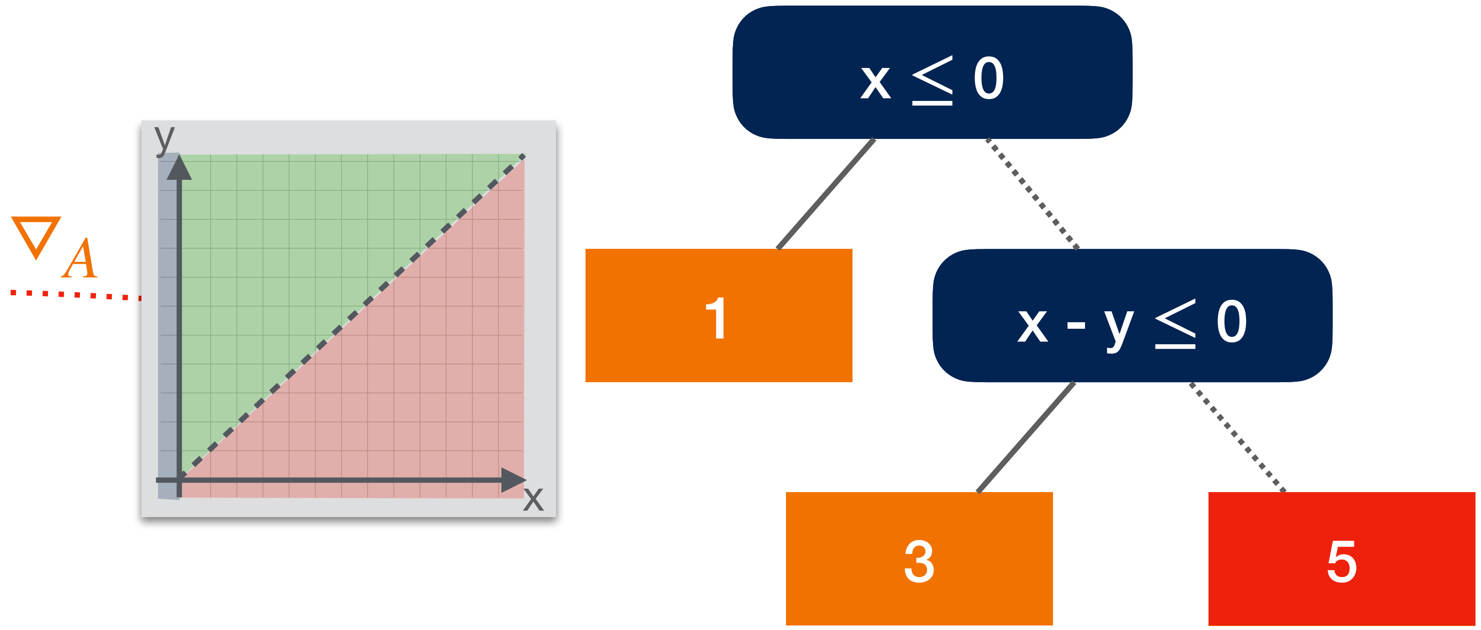


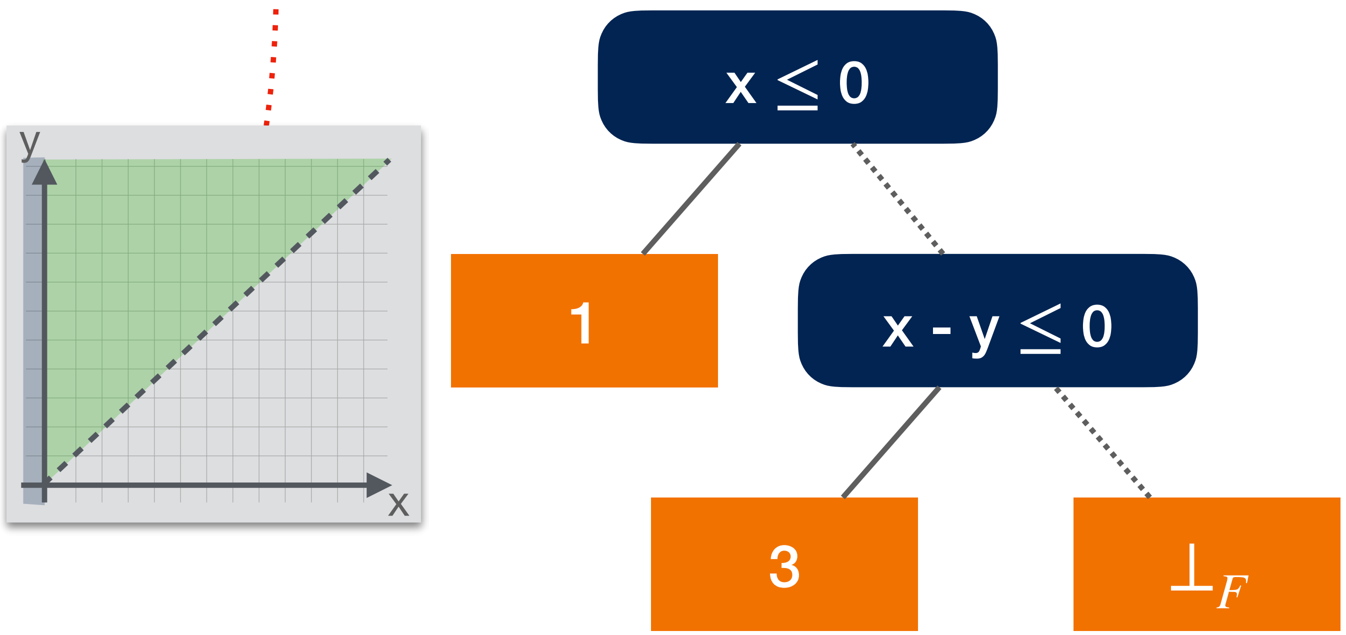


Abstract Definite Termination Semantics

Example

```
1x ← [-∞, +∞]
2y ← [-∞, +∞]
 while 3(x > 0) do
       4x ← x - y
od5
```

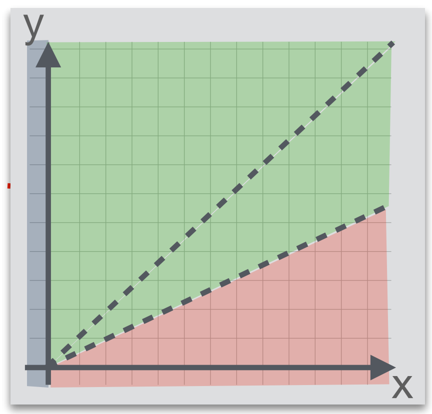


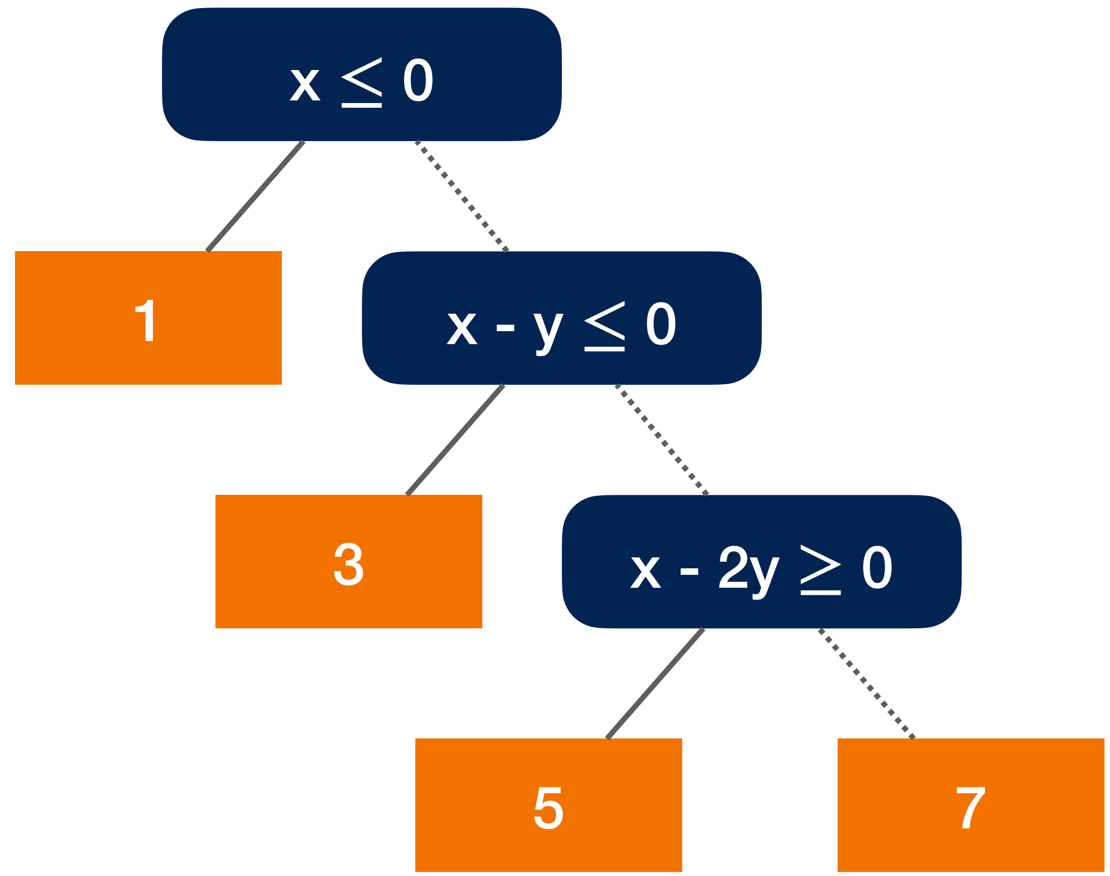


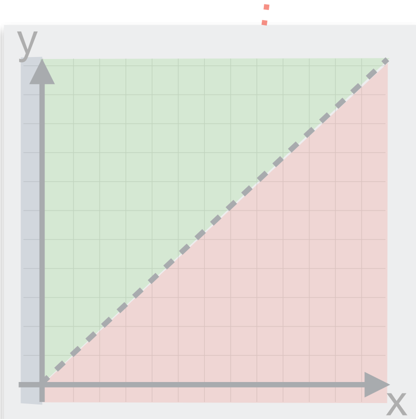
Abstract Definite Termination Semantics

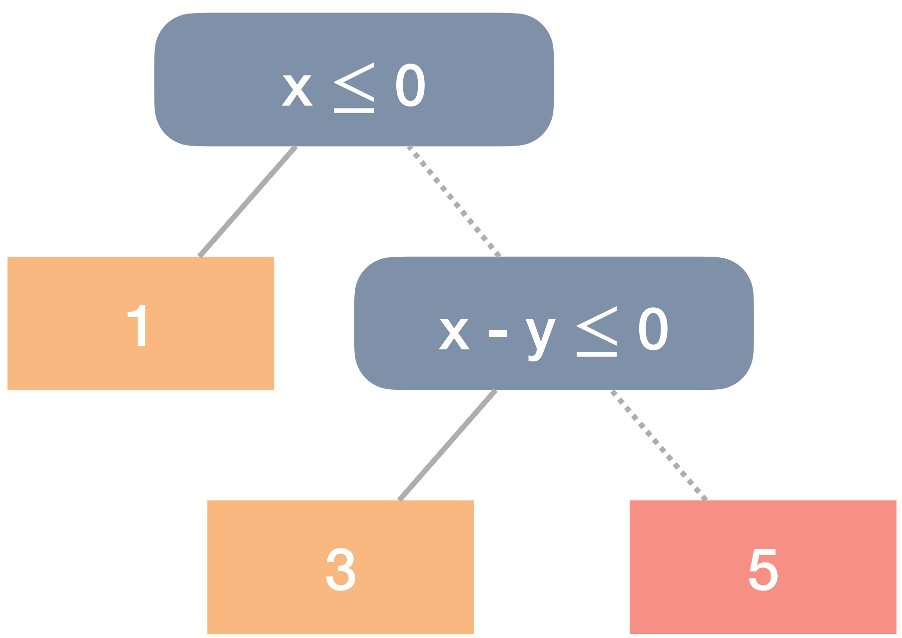
Example

[1]x ← [-∞, +∞]
[2]y ← [-∞, +∞]
while [3](x > 0) **do**
[4]x ← x - y
od[5]







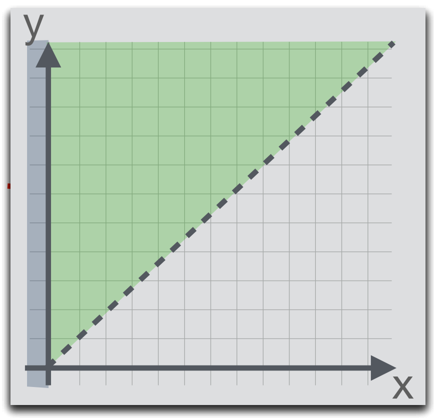


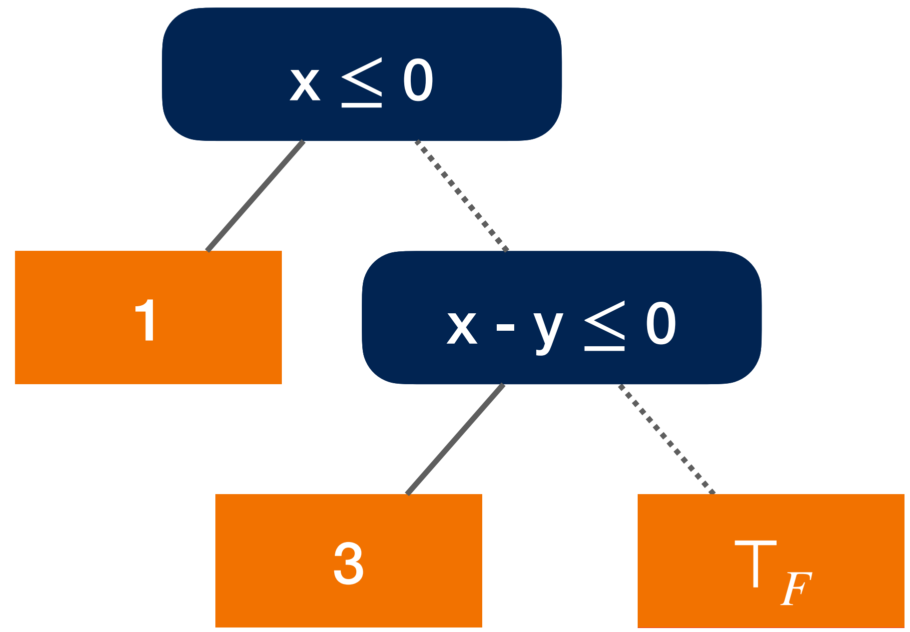
Abstract Definite Termination Semantics

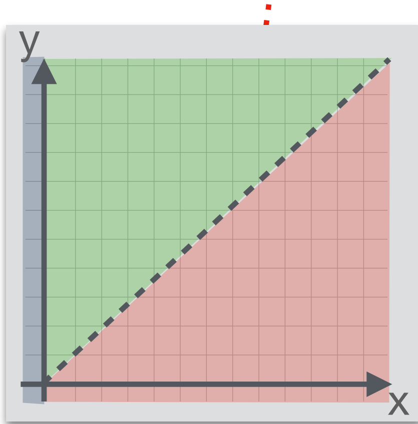
Example

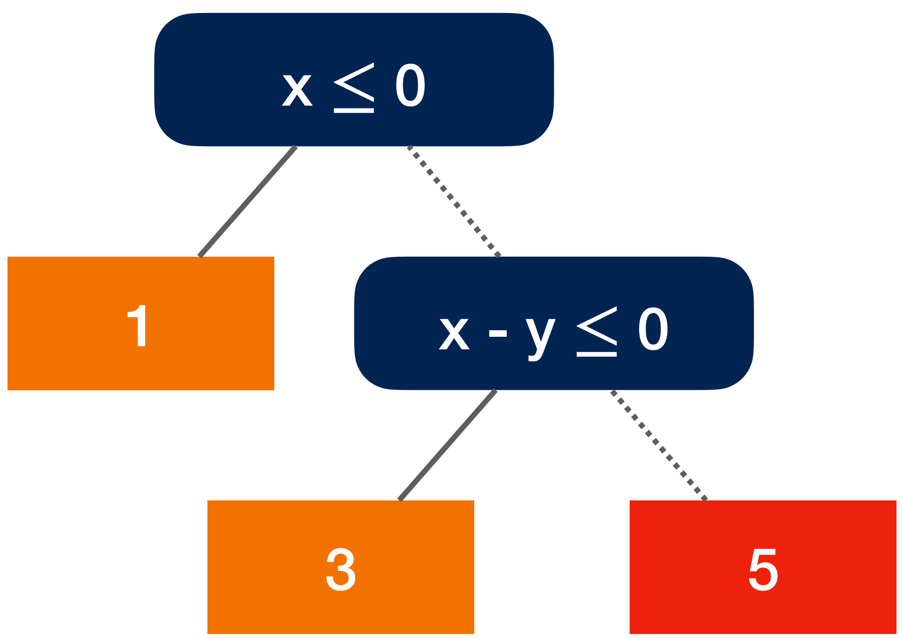
```
¹x ← [-∞, +∞]
²y ← [-∞, +∞]
 while ³(x > 0) do
       ⁴x ← x - y
od⁵
```

∇_A





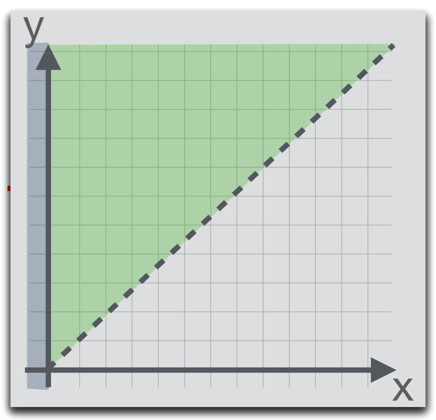


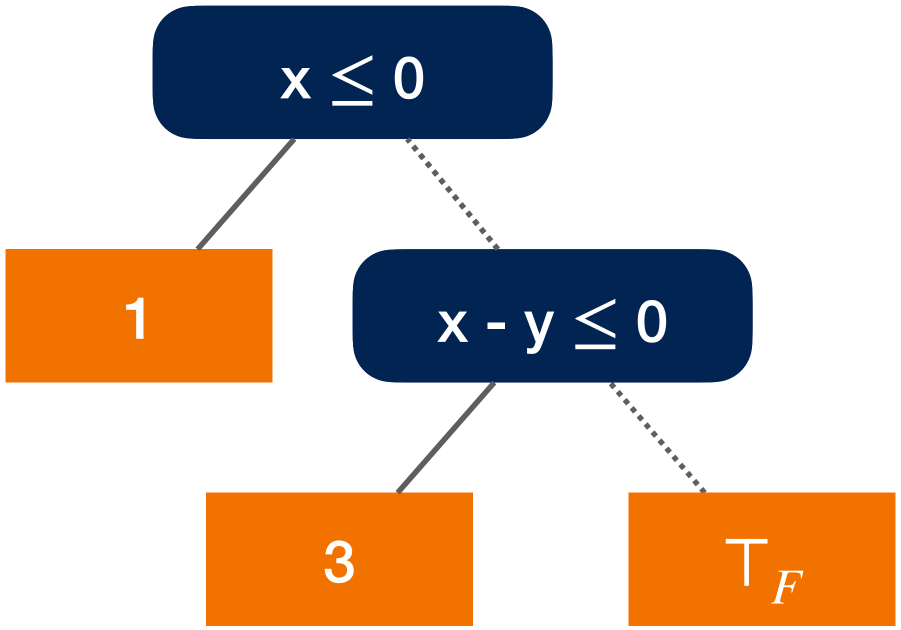


Abstract Definite Termination Semantics

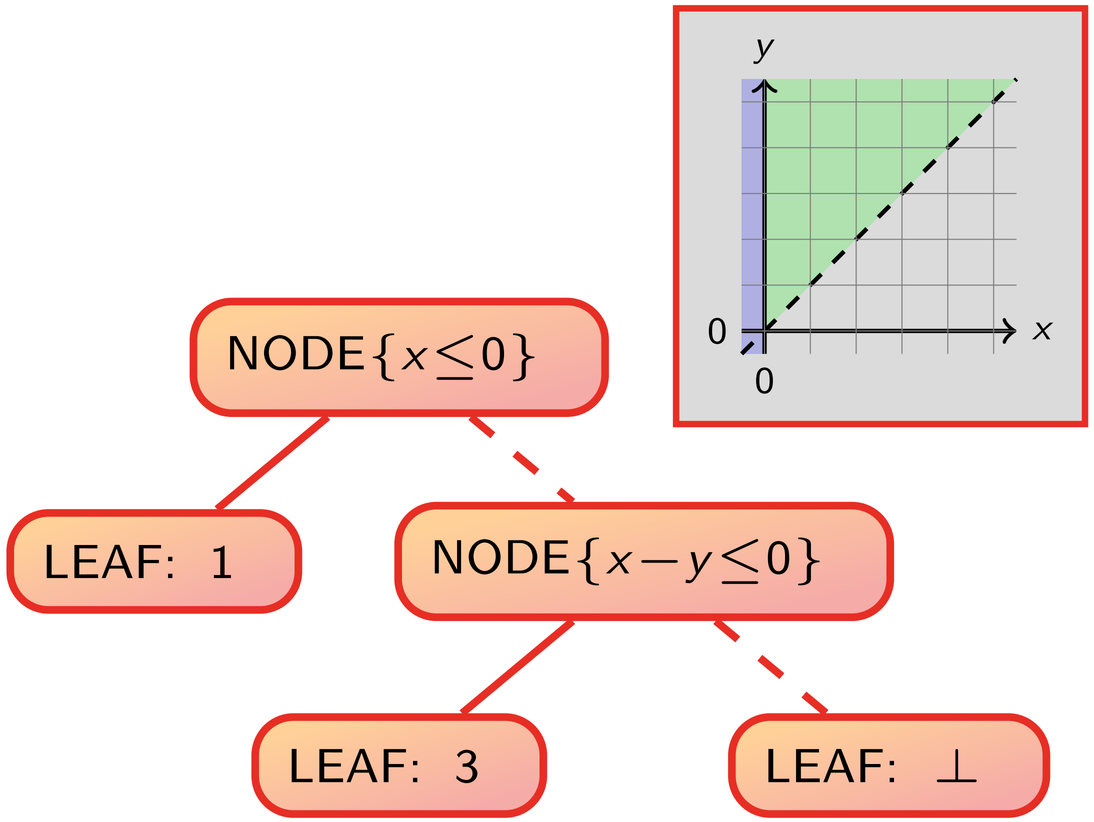
Example

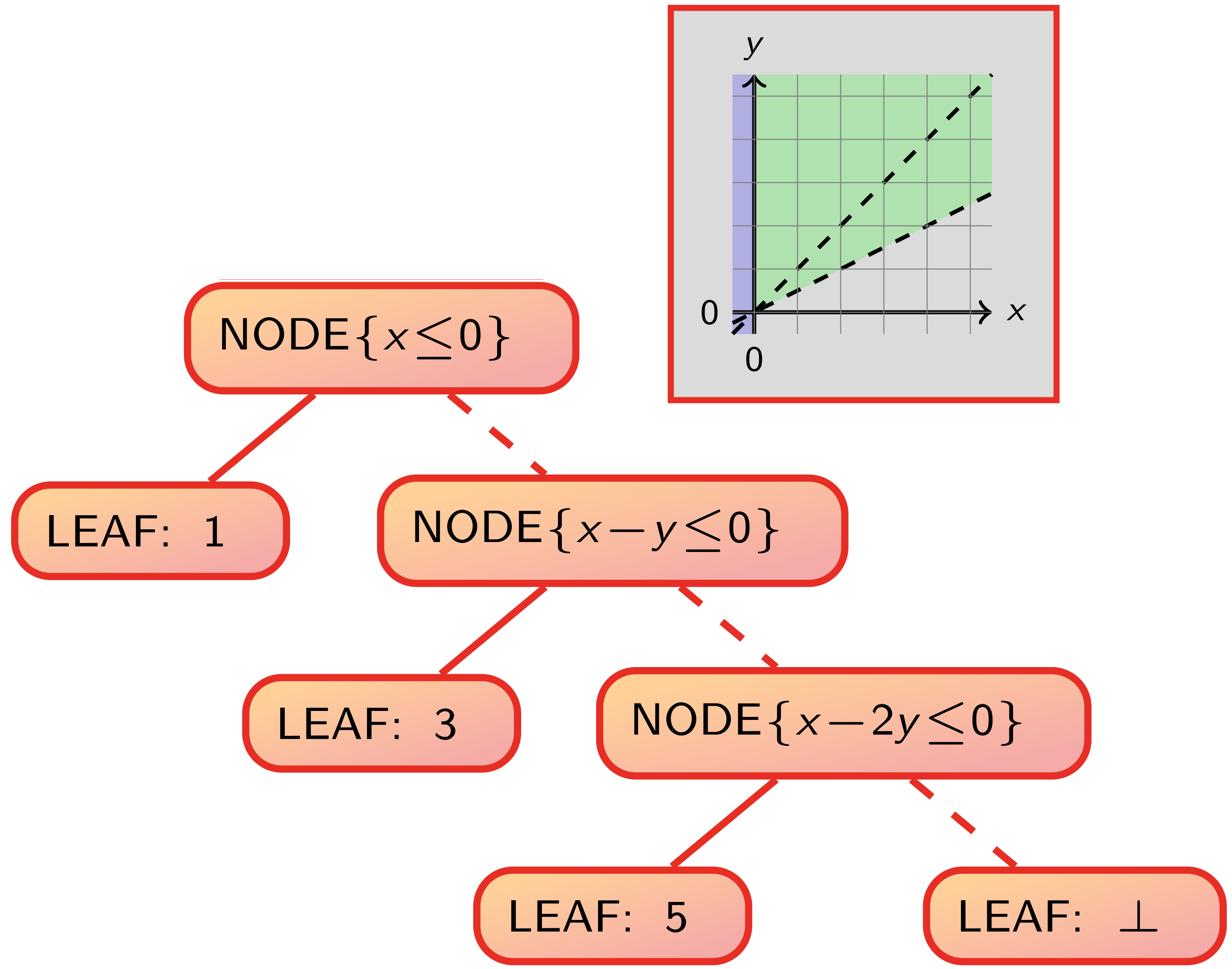
^{1}x ← [-∞, +∞]
^{2}y ← [-∞, +∞]
while ^{3}(x > 0) **do**
^{4}x ← x - y
od^5

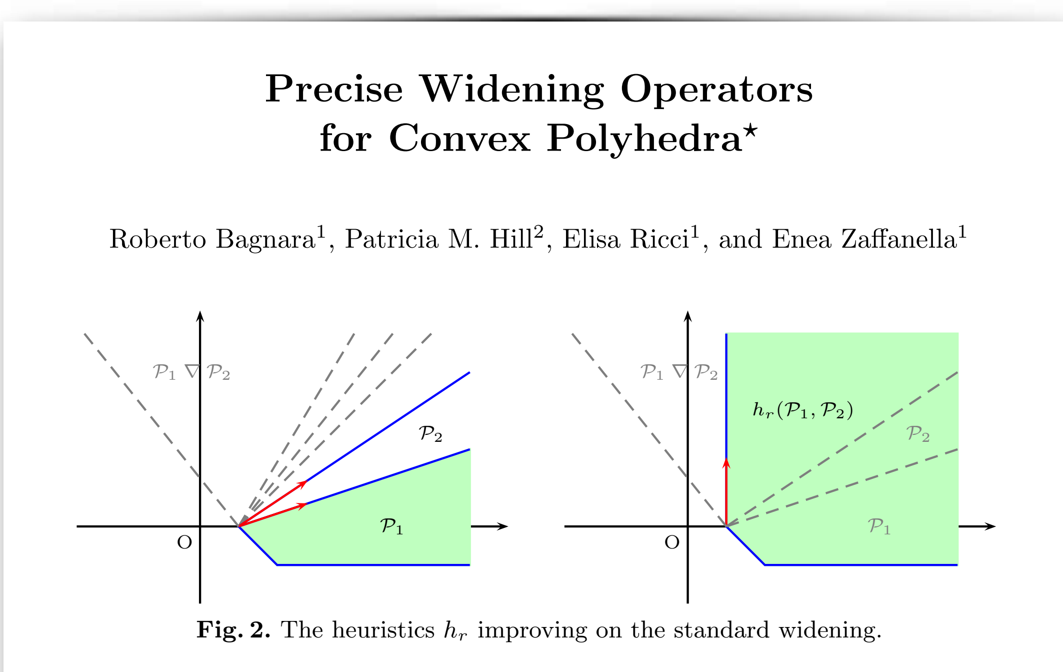




Better Widening



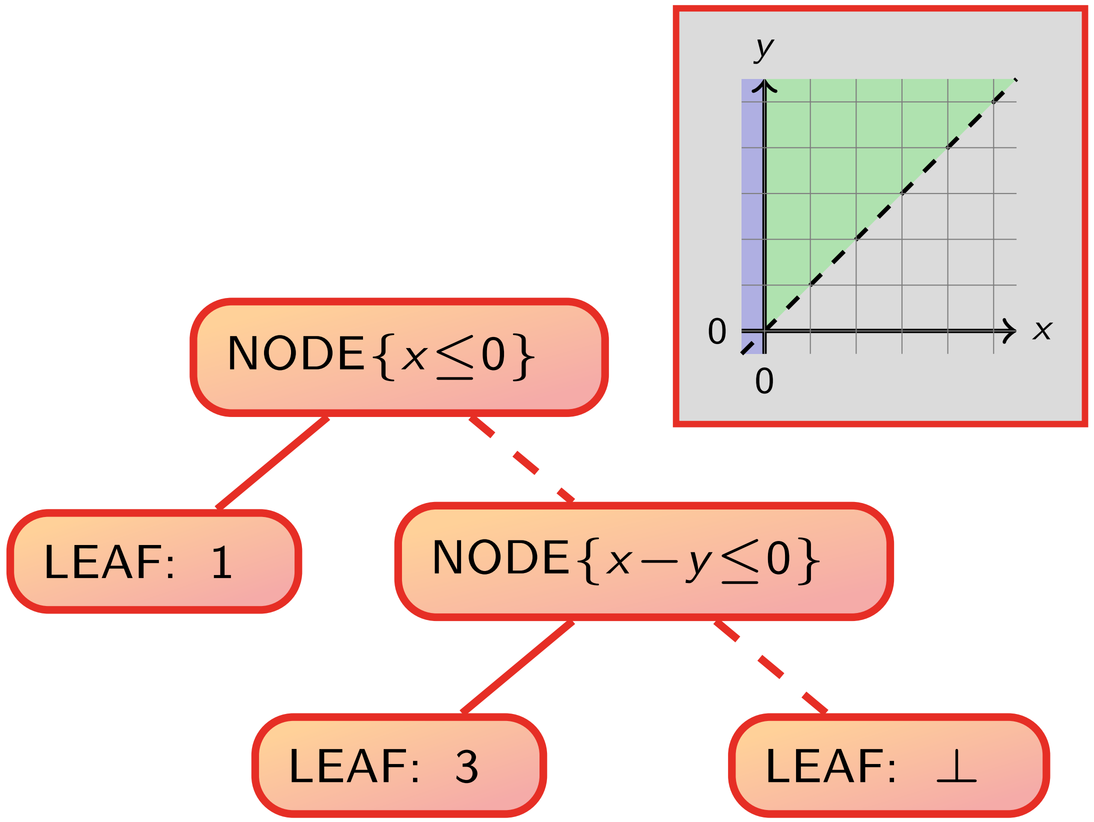


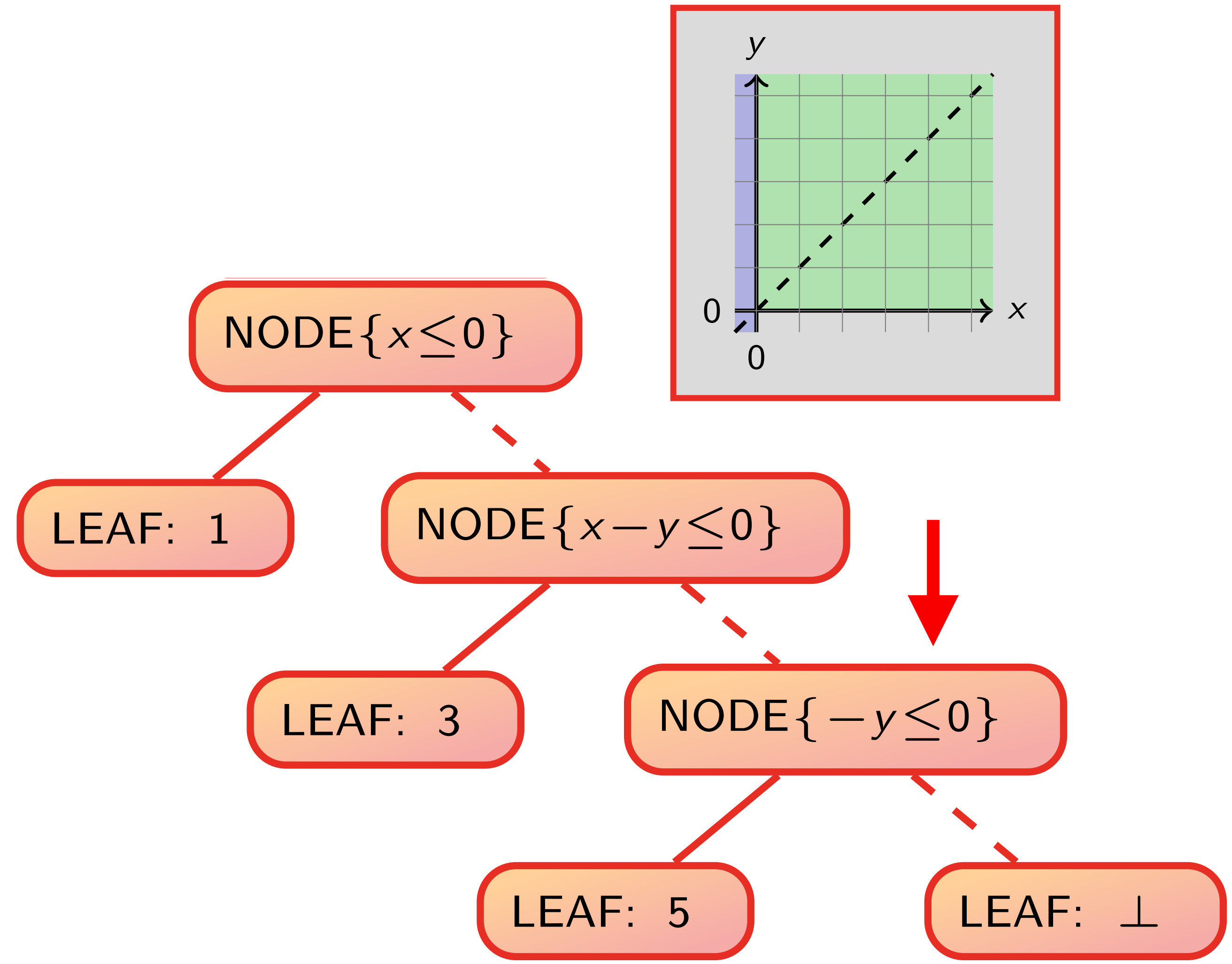

Precise Widening Operators
for Convex Polyhedra*

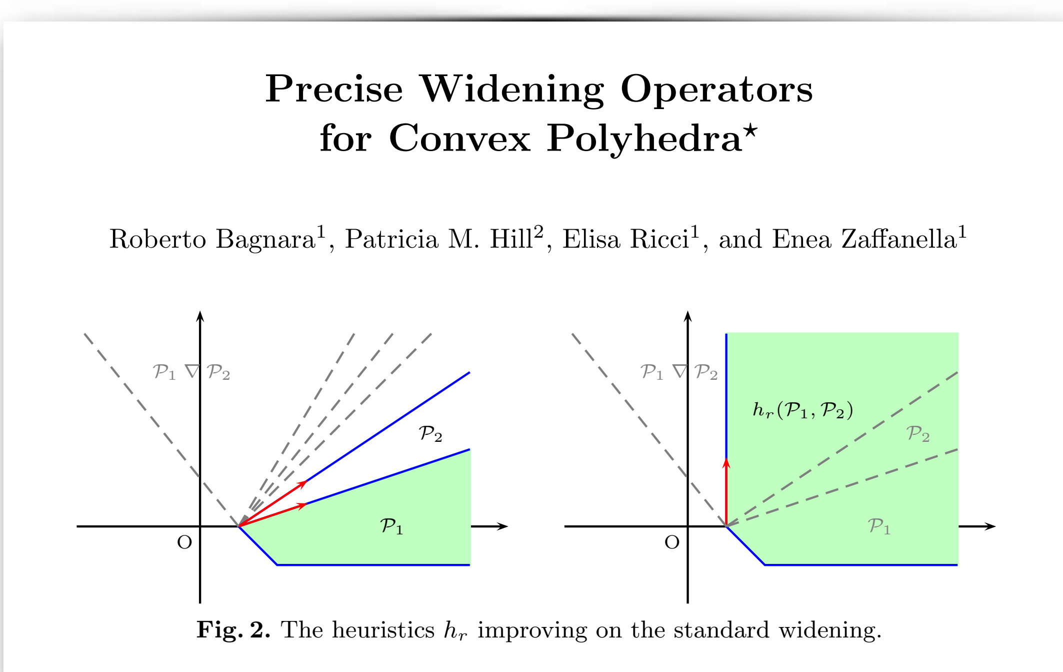
Roberto Bagnara[1], Patricia M. Hill[2], Elisa Ricci[1], and Enea Zaffanella[1]

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

Better Widening



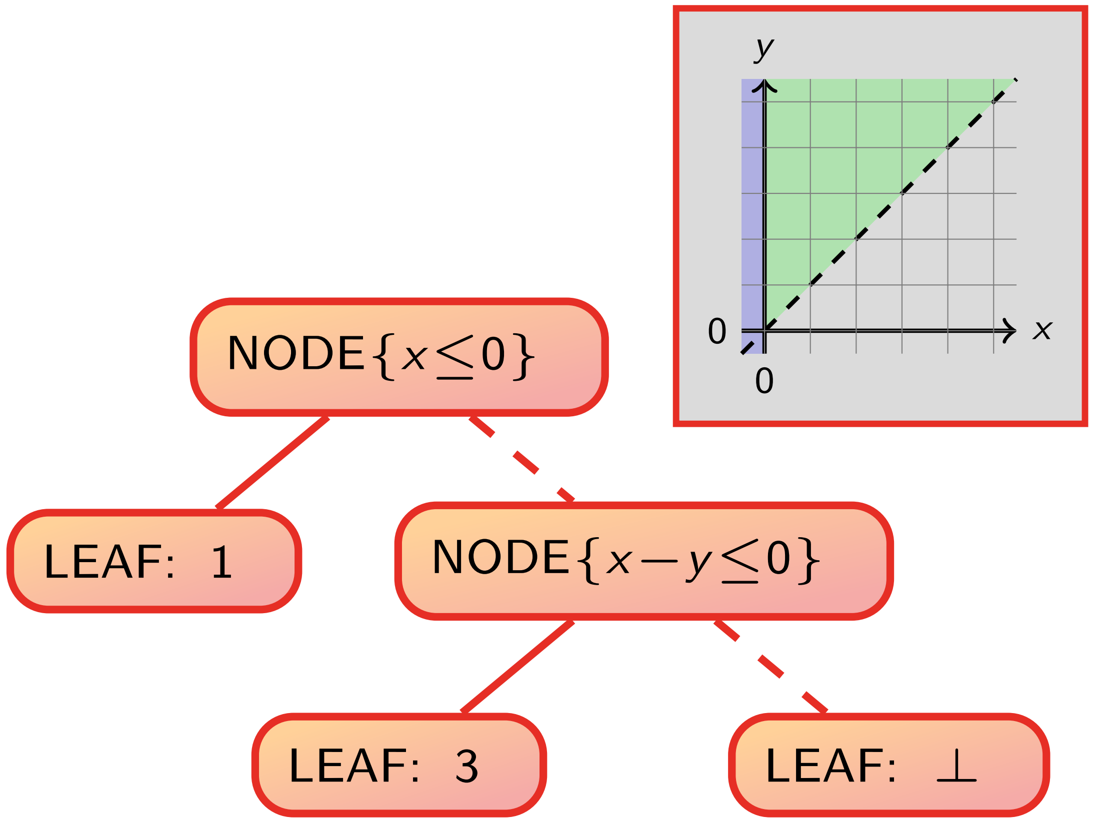


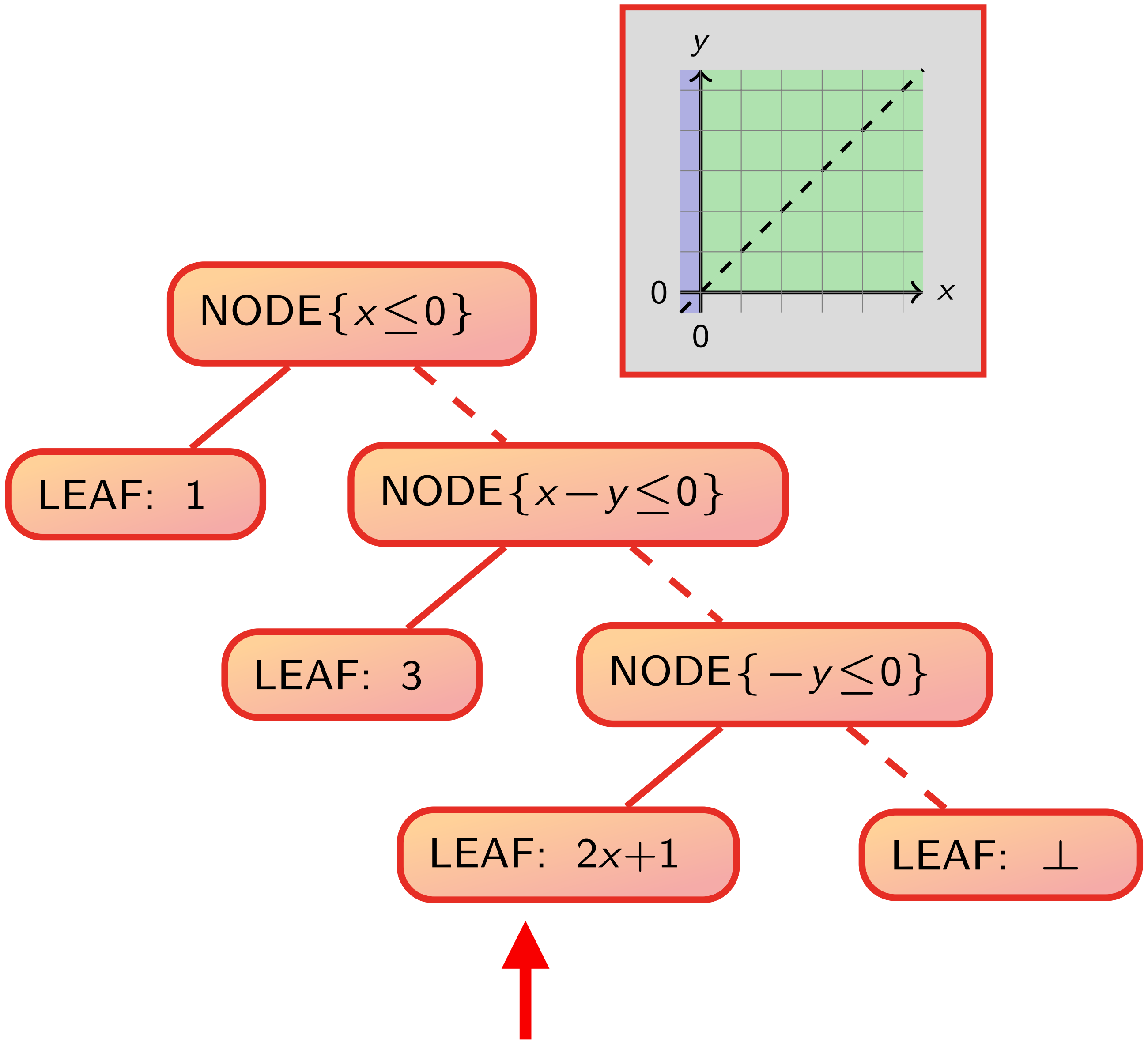

Precise Widening Operators
for Convex Polyhedra*

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

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

Better Widening





Example

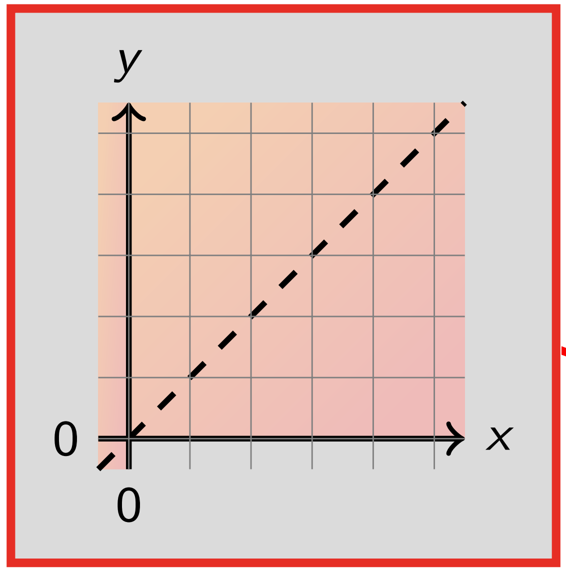
int : x, y

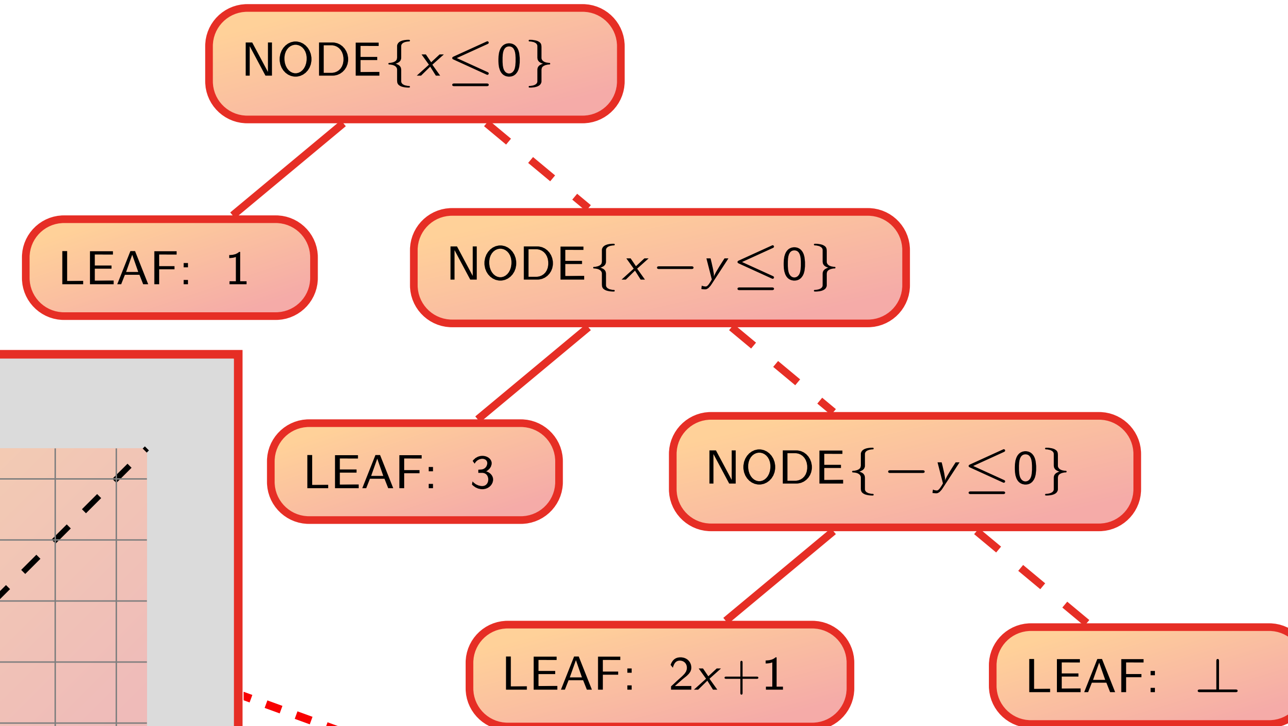
while [1]($x > 0$) do

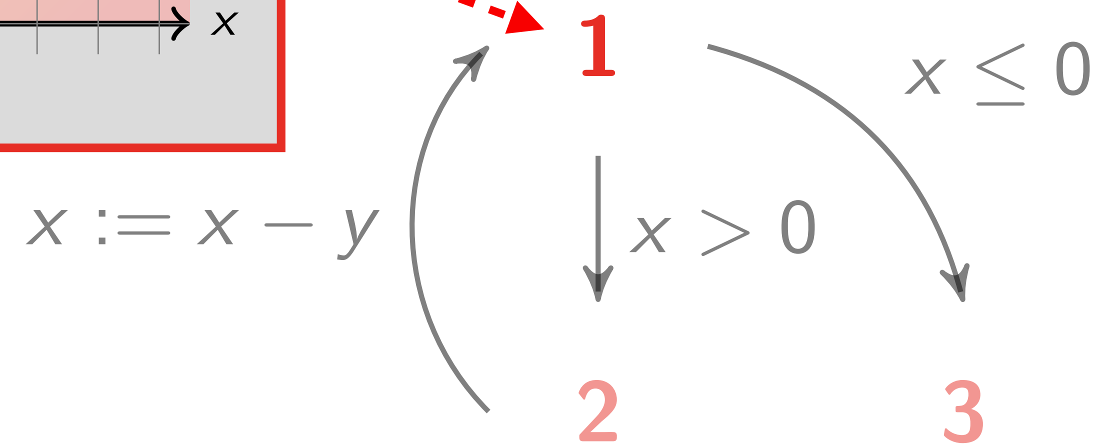
[2]$x := x - y$

od[3]

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





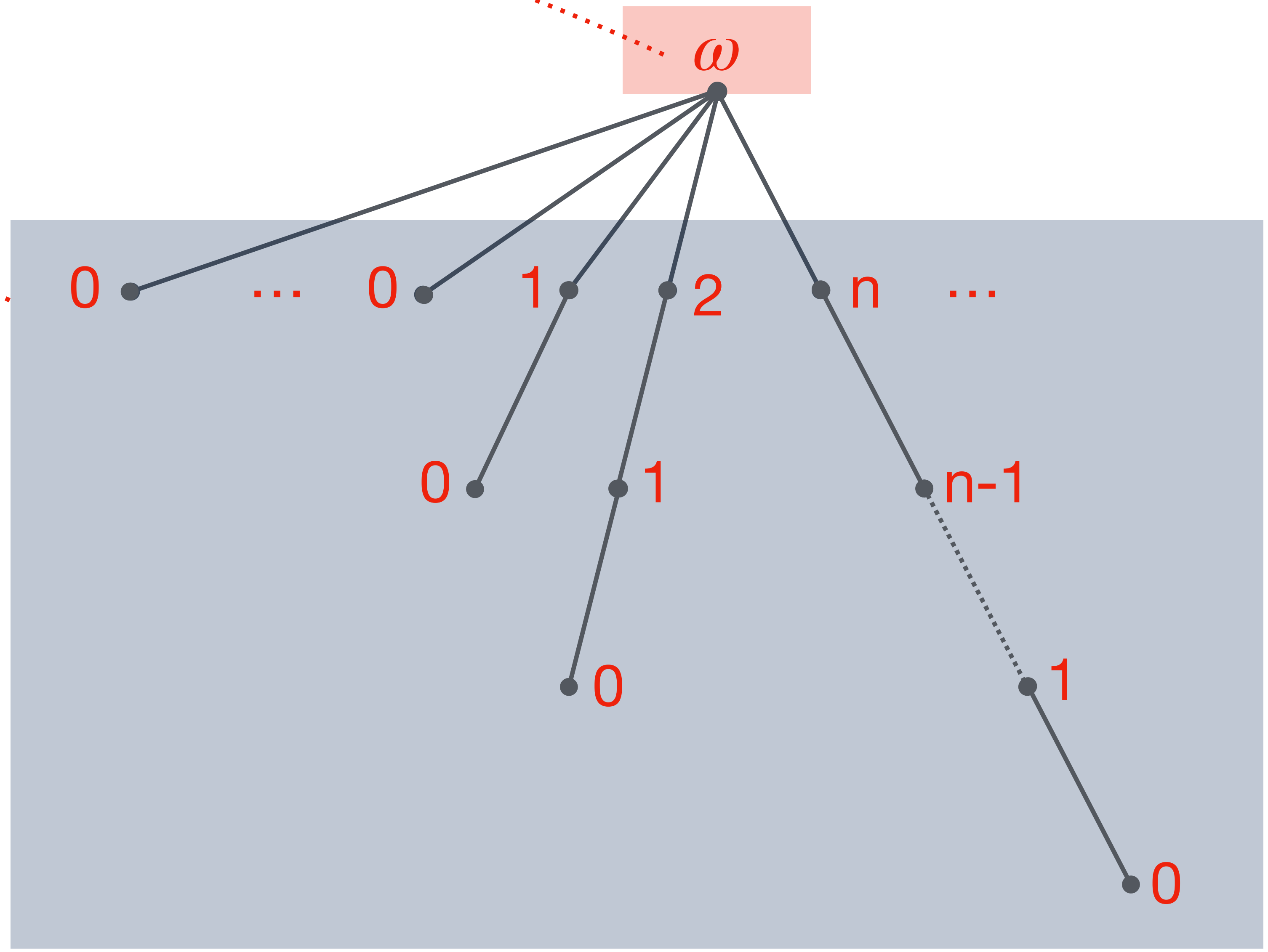


Ordinal-Valued Raking Functions

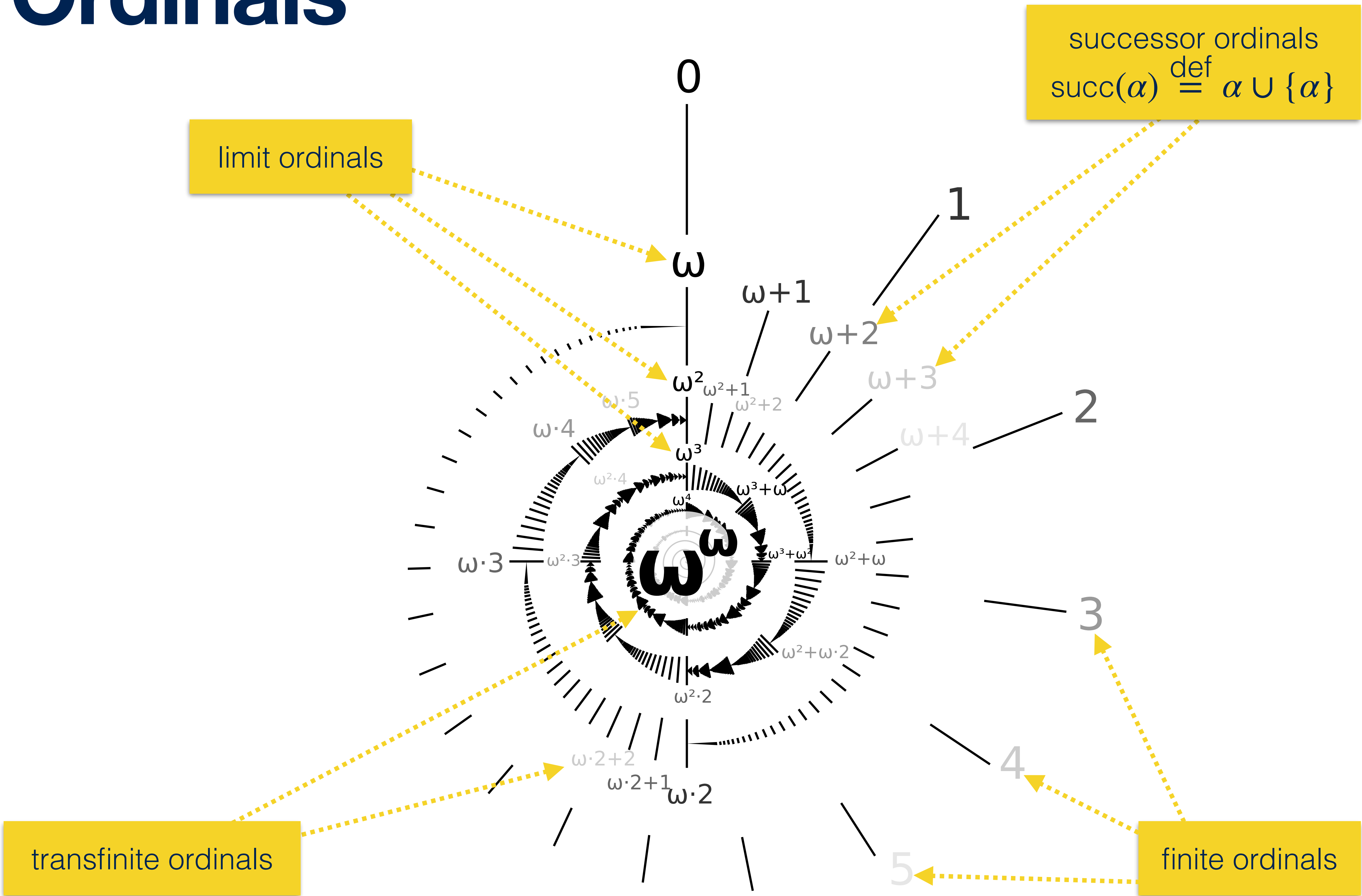
Need for Ordinals

Example

```
¹x ← [-∞, +∞]
 while ²(x > 0) do
       ³x ← x - 1
od⁴
```



Ordinals


successor ordinals
$\mathrm{succ}(\alpha) \stackrel{\text{def}}{=} \alpha \cup \{\alpha\}$
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
ω⁴
ω^{ω}

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

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

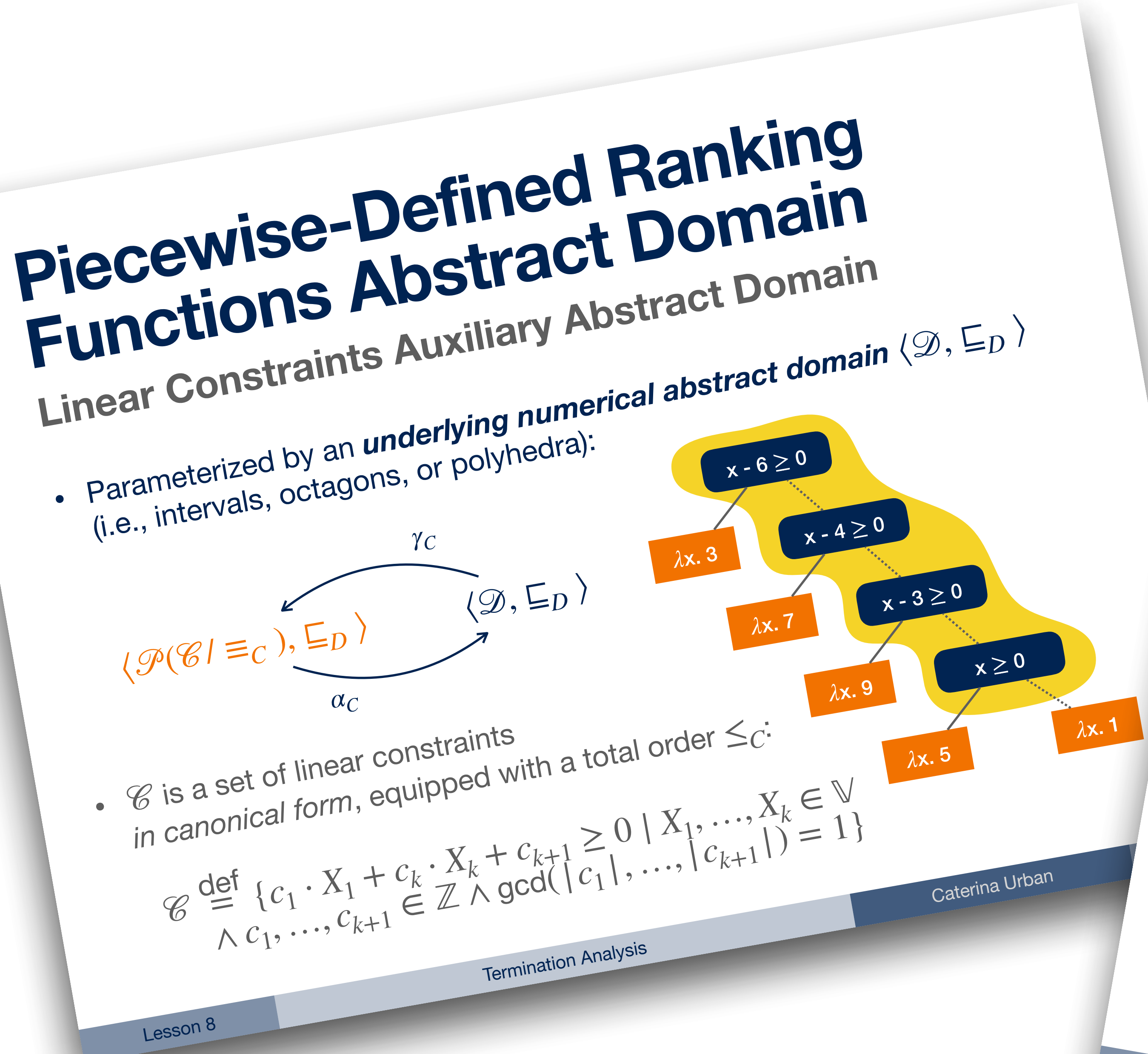
$$\alpha \cdot \text{succ}(\beta) = (\alpha \cdot \beta) + \alpha \qquad \text{(successor case)}$$

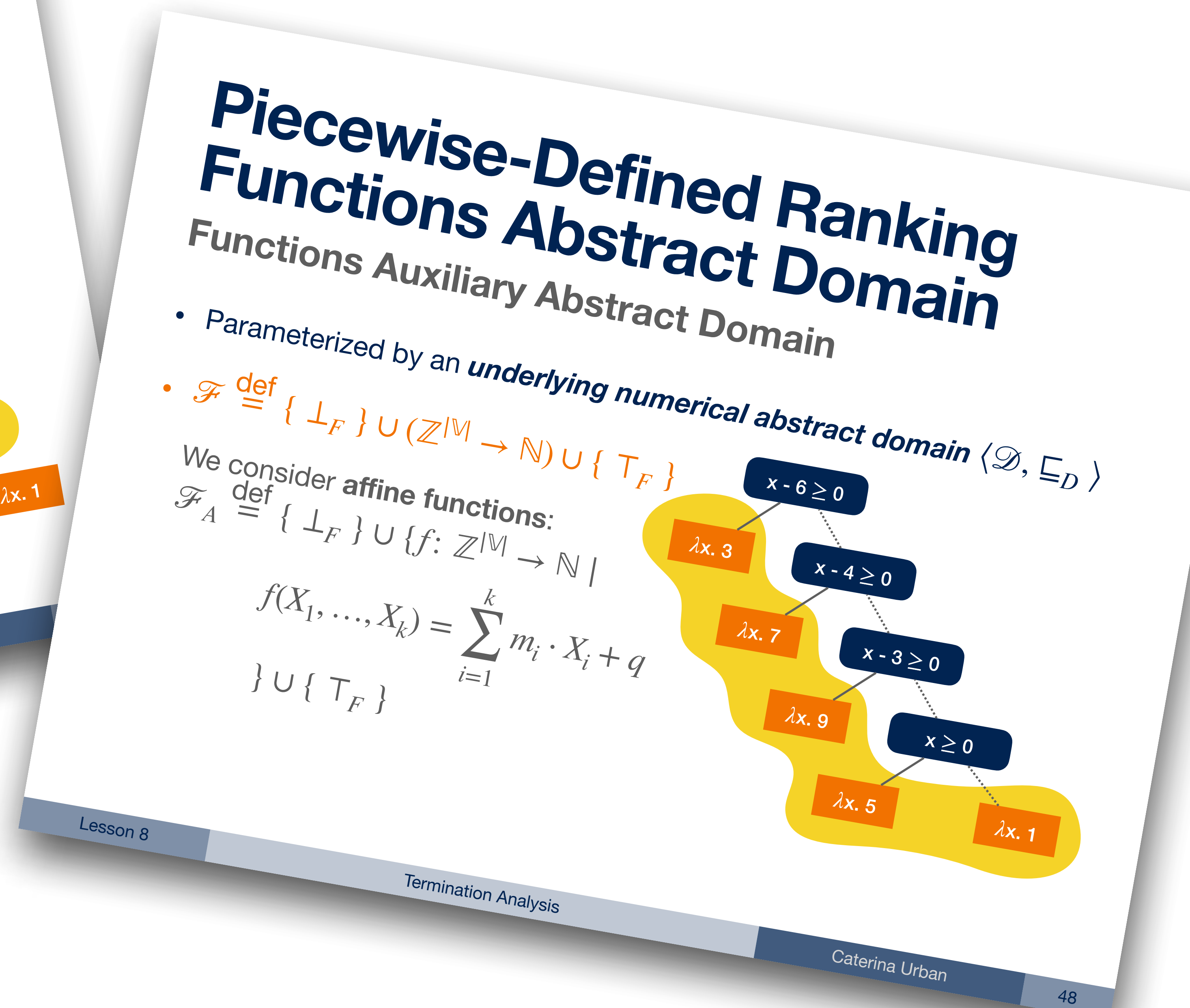
$$\alpha \cdot \beta = \bigcup_{\gamma < \beta} (\alpha \cdot \gamma) \qquad \text{(limit case)}$$

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 Ranking Functions Abstract Domain





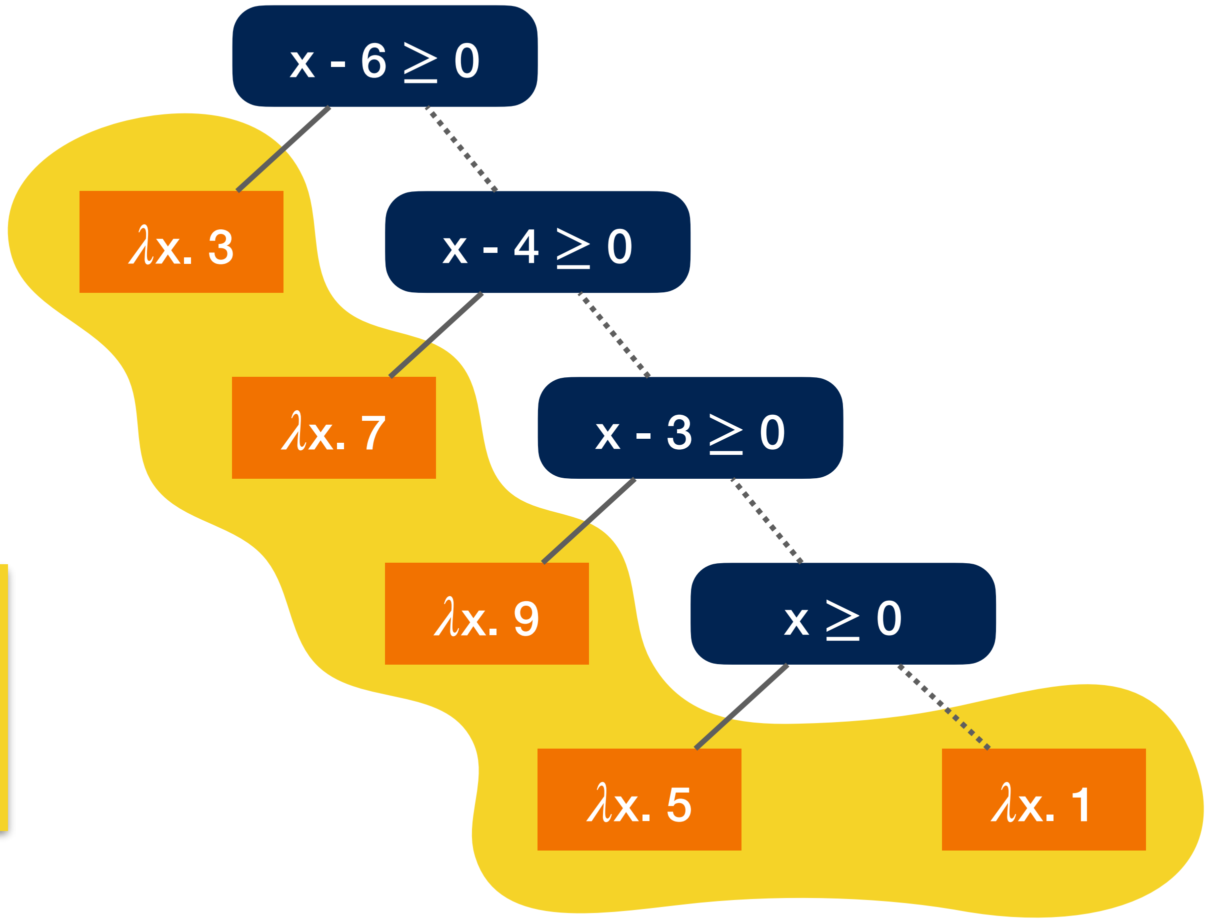
Piecewise-Defined Ranking Functions Abstract Domain

Ordinal-Valued Functions Auxiliary Domain

- Parameterized by the ***underlying functions auxiliary domain*** $\langle \mathcal{F}, \sqsubseteq_F \rangle$
- $\mathcal{W} \stackrel{\text{def}}{=} \{ \bot_W \} \cup \{ \sum_i \omega^i \cdot f_i \mid f_i \in \mathcal{F} \setminus \{ \bot_F, \top_F \} \} \cup \{ \top_W \}$

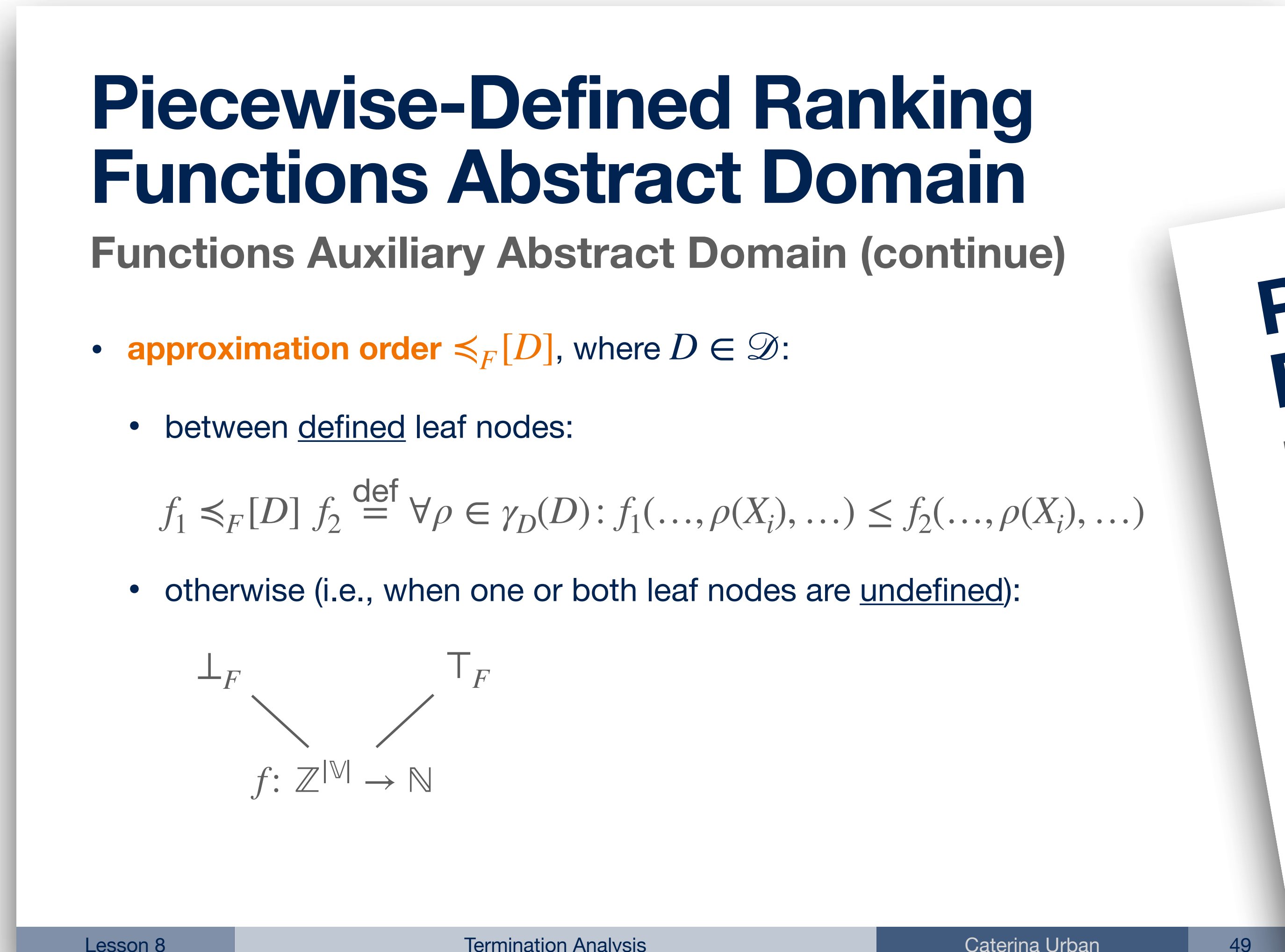
Cantor Normal Form

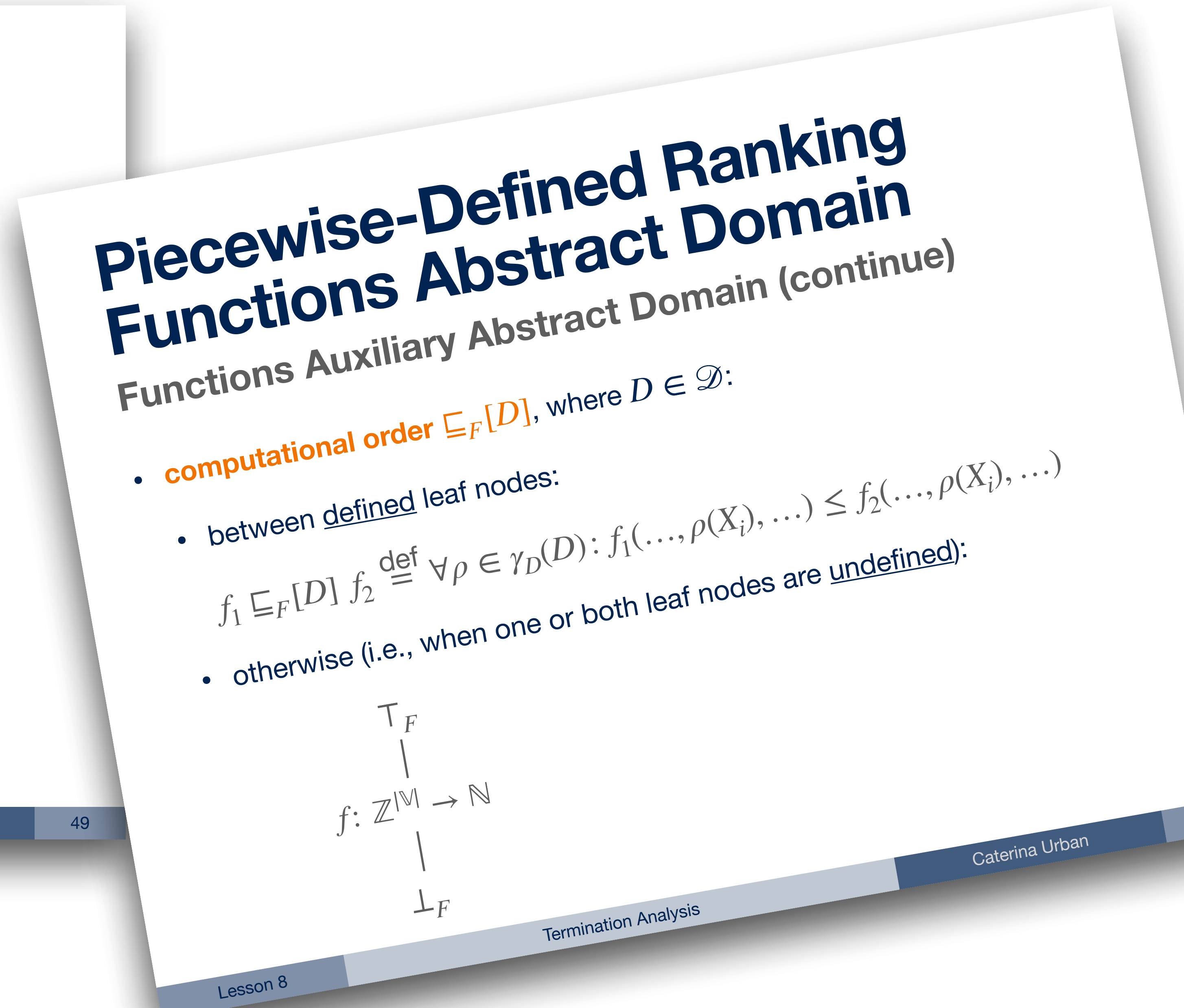
$\omega^{\beta_1} \cdot n_1 + \ldots + \omega^{\beta_k} \cdot n_k$



Piecewise-Defined Ranking Functions Abstract Domain

Ordinal-Valued Functions Auxiliary Domain (continue)





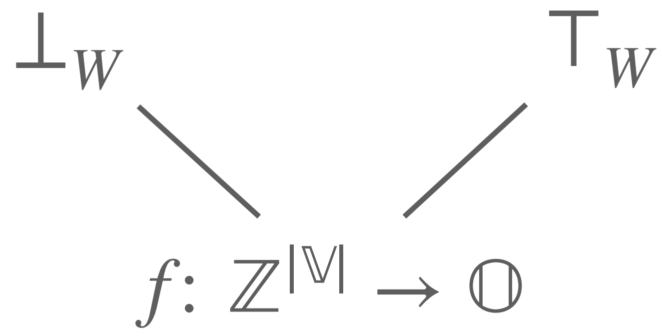
Piecewise-Defined Ranking Functions Abstract Domain

Ordinal-Valued Functions Auxiliary Domain (continue)

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

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

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



Piecewise-Defined Ranking Functions Abstract Domain

Ordinal-Valued Functions Auxiliary Domain (continue)

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

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

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

$$\top_W$$

$$|$$

$$f \colon \mathbb{Z}^{|\mathbb{V}|} \to \mathbb{O}$$

$$|$$

$$\bot_W$$

Piecewise-Defined Functions Abstrac

- $\mathscr{A} \stackrel{\text{def}}{=} \{\text{LEAF}: f \mid f \in \mathscr{W}\} \cup \{\text{NOD}$

- **concretization function** $\gamma_A: \mathscr{A} \to (\mathscr{E}$

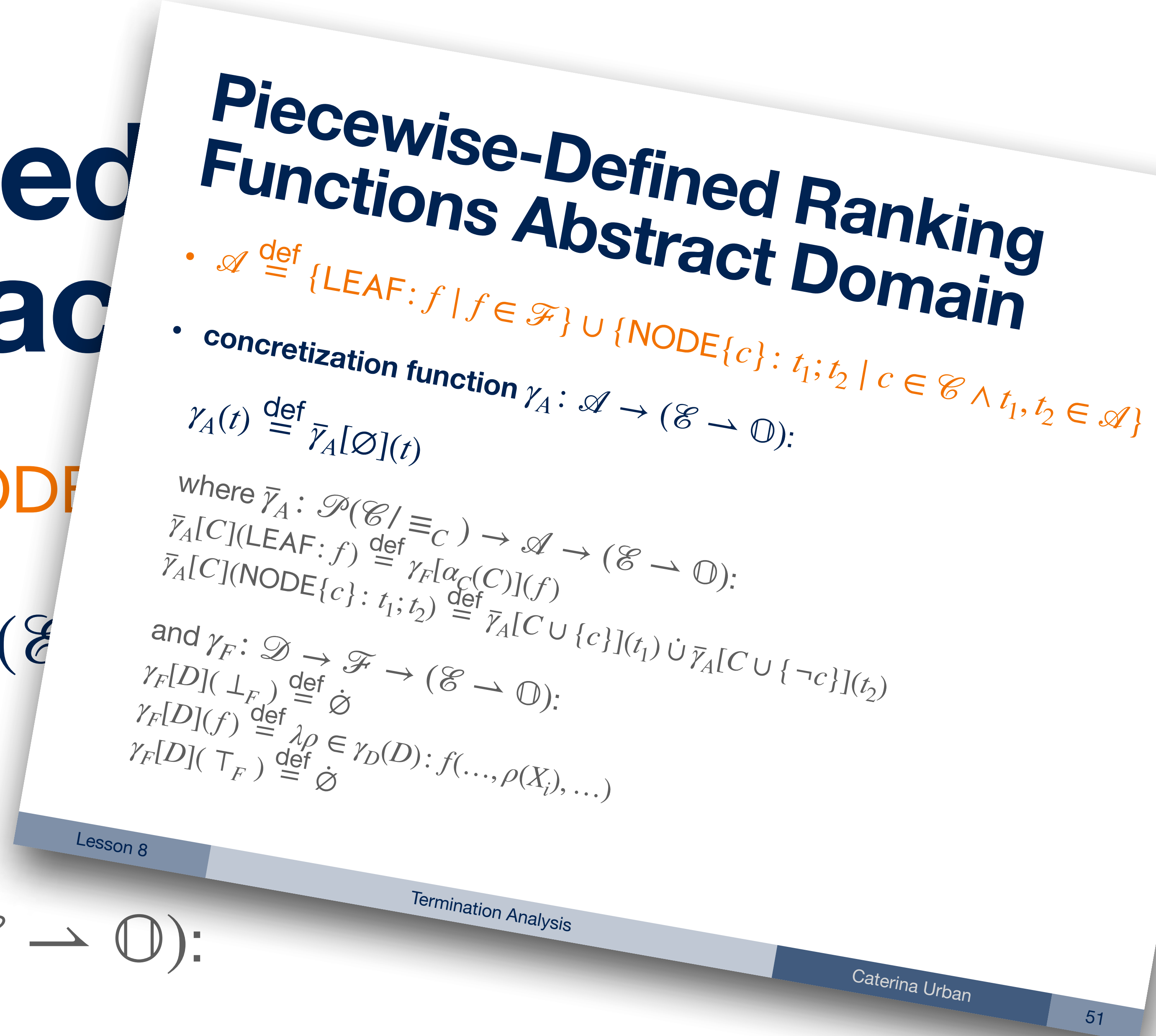
$$\gamma_A(t) \stackrel{\text{def}}{=} \bar{\gamma}_A[\varnothing](t)$$

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

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

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

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



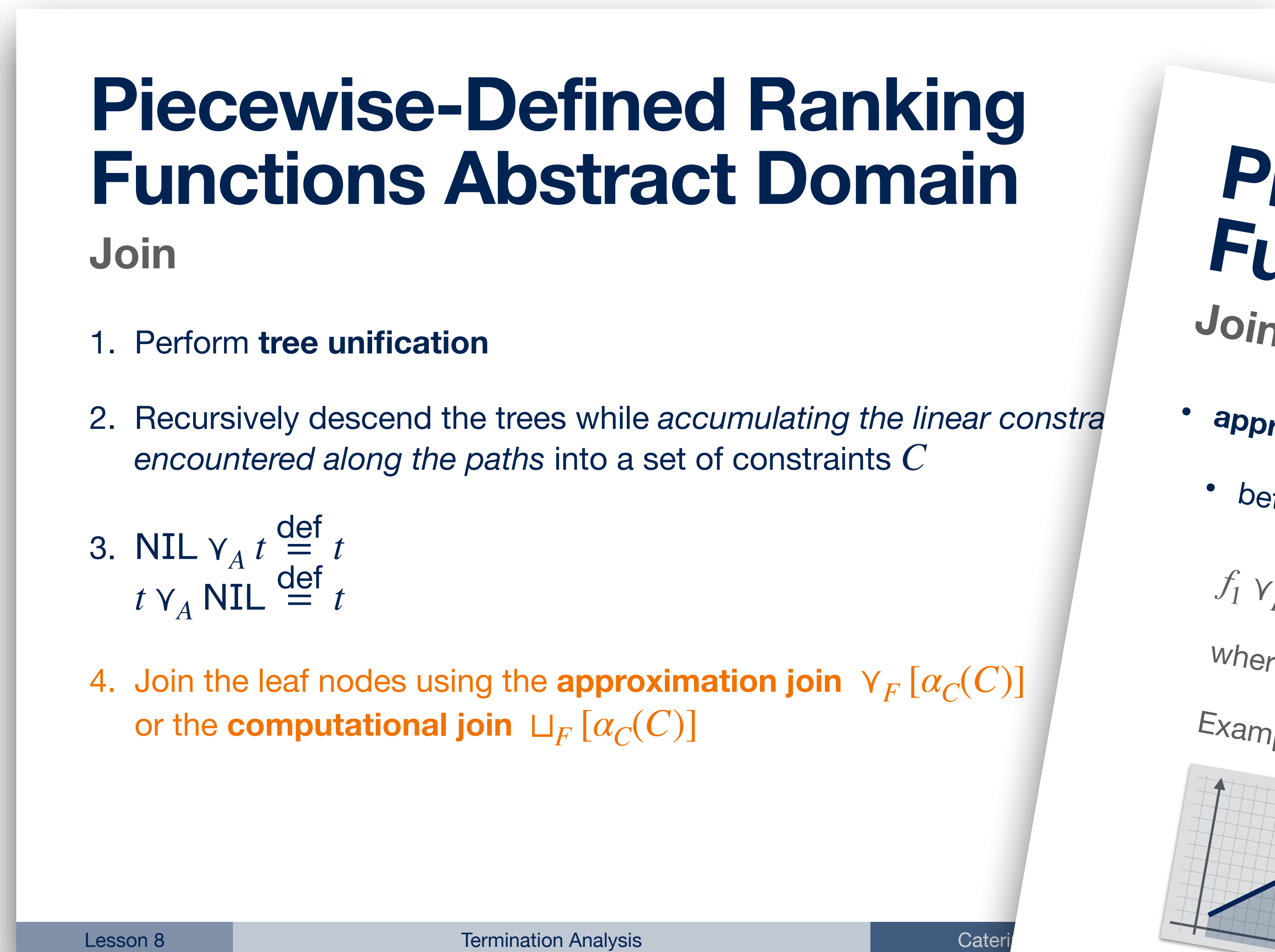
Piecewise-Defined Ranking Functions Abstract Domain

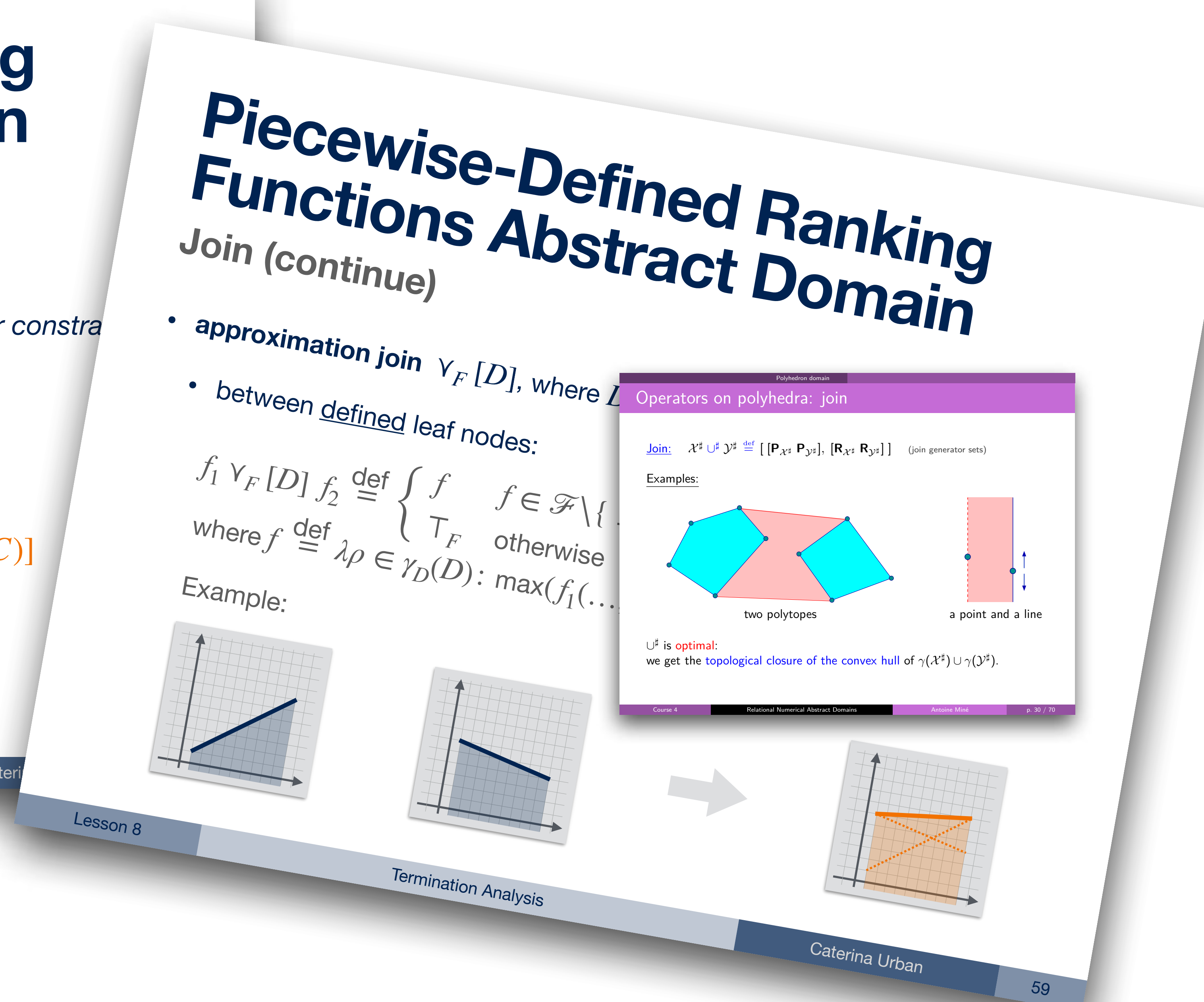
Abstract Domain Operators

- They manipulate elements in $\mathscr{A}_{\text{NIL}} \stackrel{\text{def}}{=} \{\text{NIL}\} \cup \mathscr{A}$
- 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 $\overleftarrow{\text{ASSIGN}}_A[\![X \leftarrow e]\!]$
  - test $\text{FILTER}_A[\![e]\!]$

Piecewise-Defined Ranking Functions Abstract Domain

Join





Piecewise-Defined Ranking Functions Abstract Domain

Join (continue)

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

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

    Example:

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

Piecewise-Defined Ranking Functions Abstract Domain

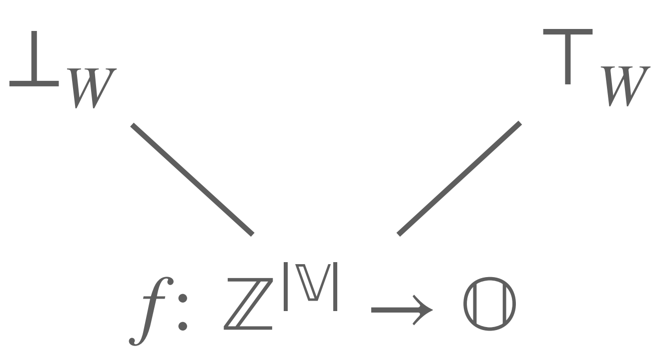
Join (continue)

- **approximation join** $\curlyvee_W [D]$, 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>):

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



Piecewise-Defined Ranking Functions Abstract Domain

Join (continue)

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

    computational join $\sqcup_W [D]$ in ascending powers of ω

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

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

$\top_W$

|

$f\colon \mathbb{Z}^{|\mathbb{V}|} \to \mathbb{O}$

|

$\bot_W$

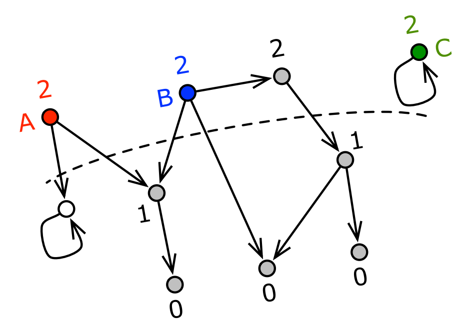
Piecewise-Defined Ranking Functions Abstract Domain

Widening

Piecewise-Defined Ranking Functions Abstract 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**

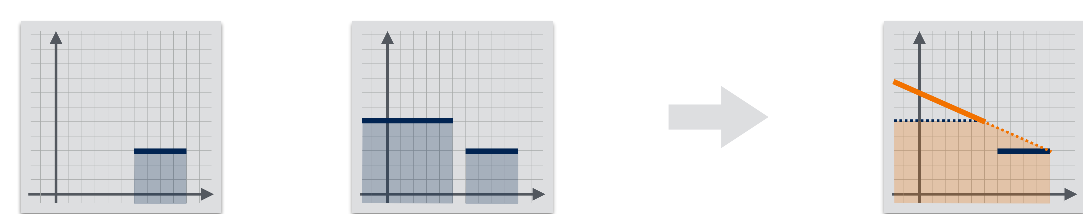


Piecewise-Defined Ranking Functions Abstract Domain

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 Ranking Functions Abstract 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

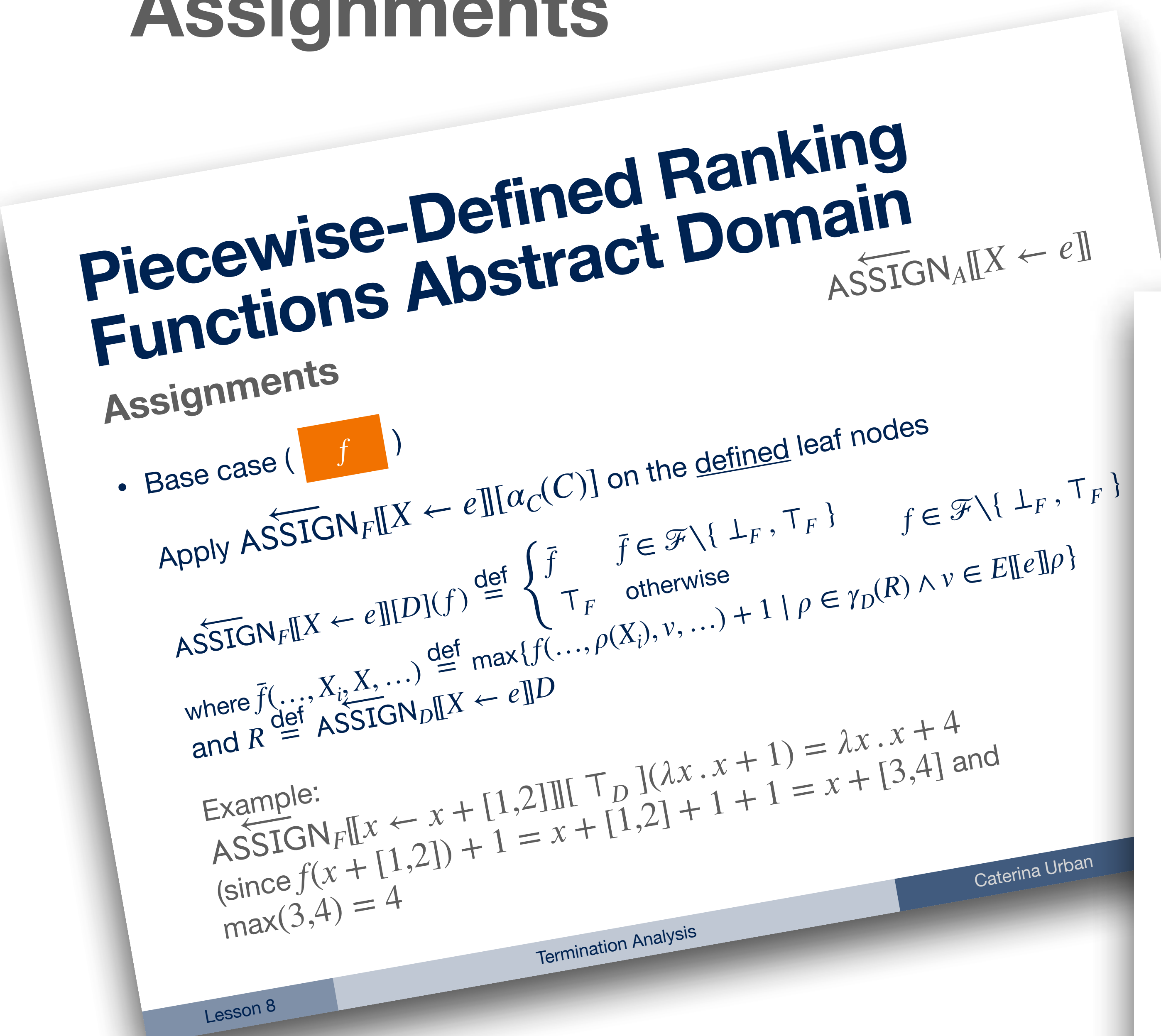
2. Widen each (defined) leaf node f with respect to each of their adjacent (defined) leaf node $\bar{f}$ using the **extrapolation operator** $\blacktriangledown_F [\alpha_C(\overline{C}), \alpha_C(C)]$, where $\overline{C}$ is the set of constraints along the path to $\bar{f}$, **in ascending powers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

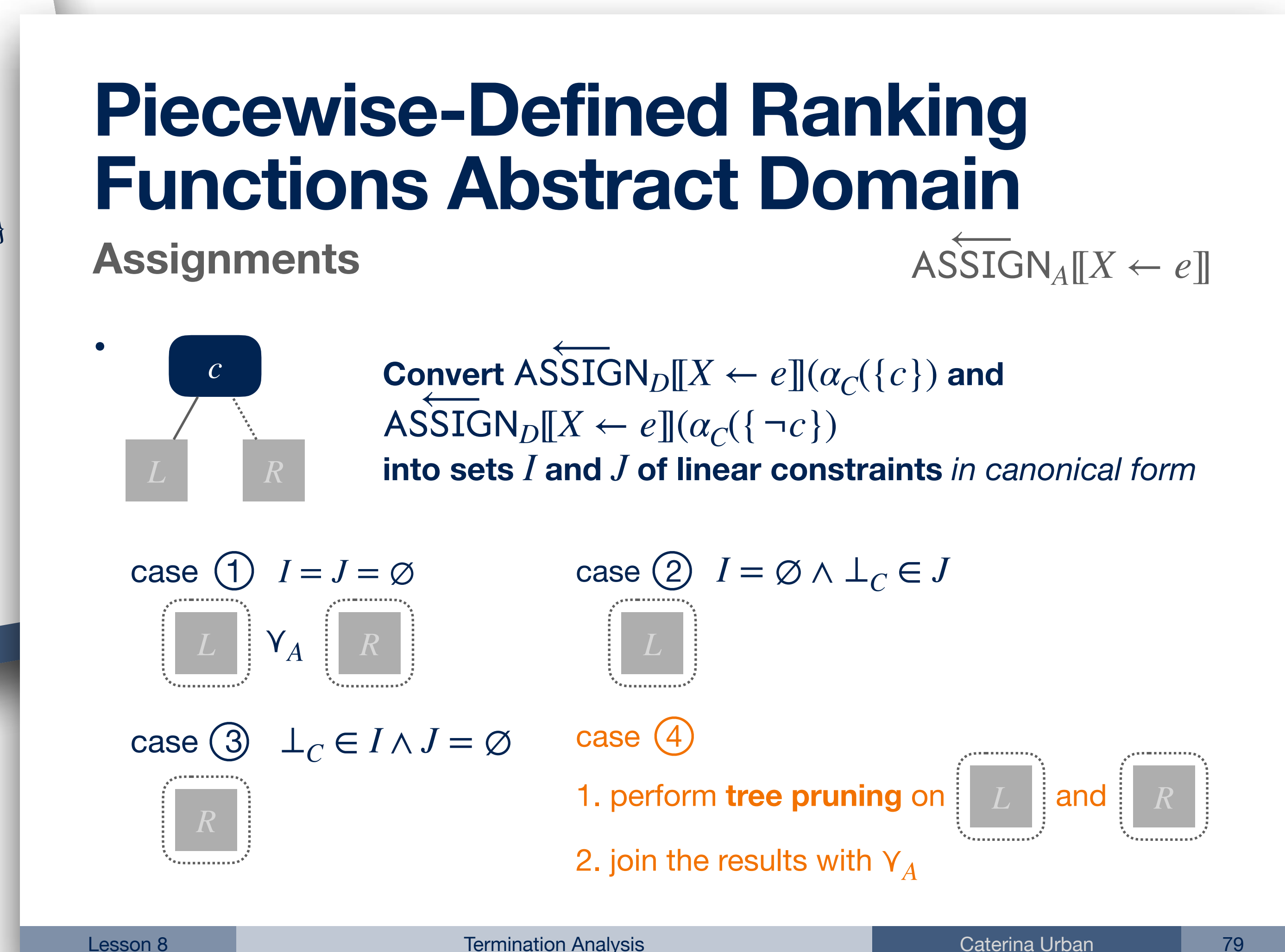
yield $\top_W$ when the extrapolation of natural-valued functions yields $\top_F$

Piecewise-Defined Ranking Functions Abstract Domain

Assignments

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





Piecewise-Defined Ranking Functions Abstract Domain

Assignments (continue)

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

- Base case (f)

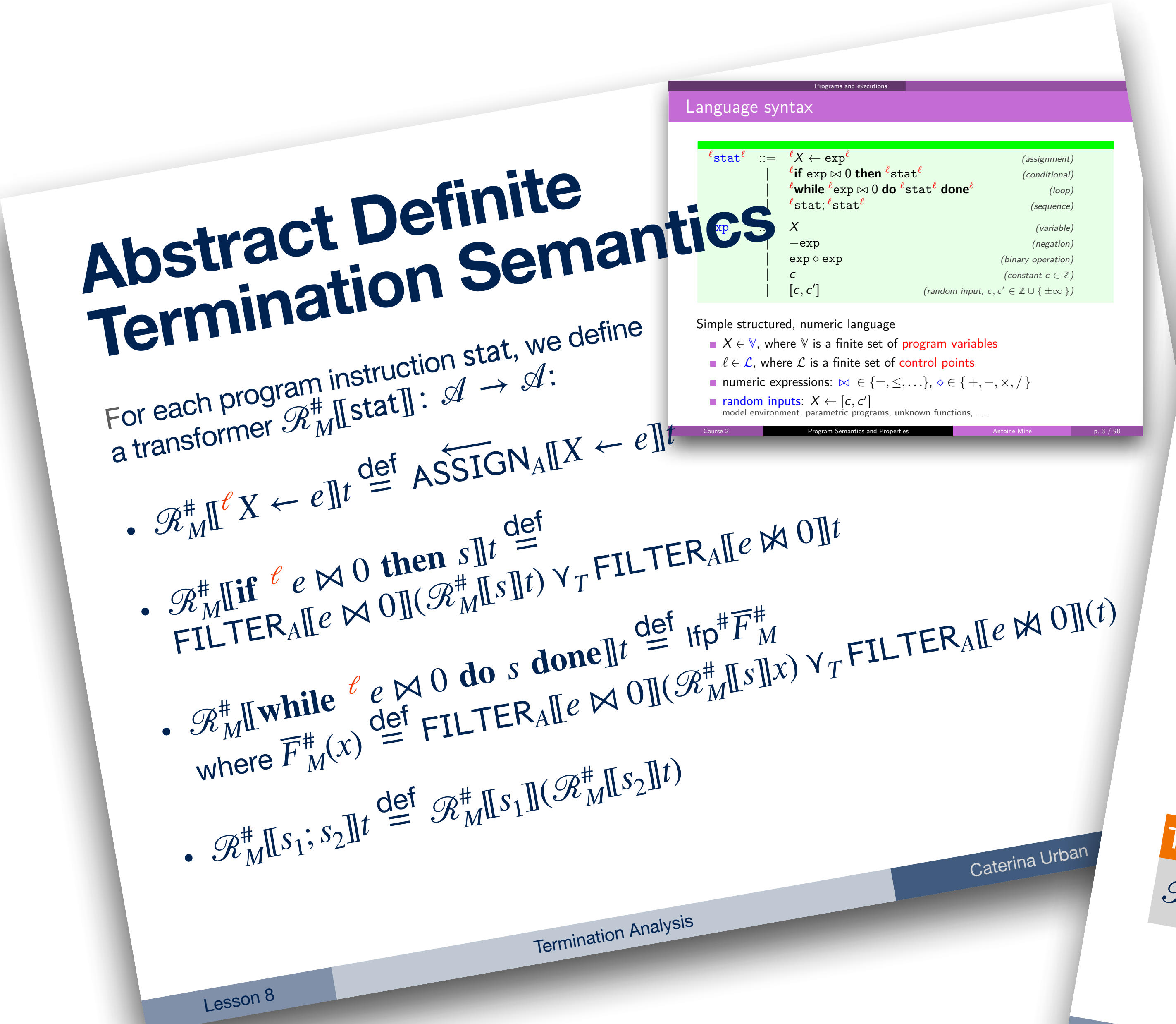
  Apply $\overleftarrow{\mathsf{ASSIGN}}_F[\![X \leftarrow e]\!][\alpha_C(C)]$ on the defined leaf nodes **in ascending powers of ω**

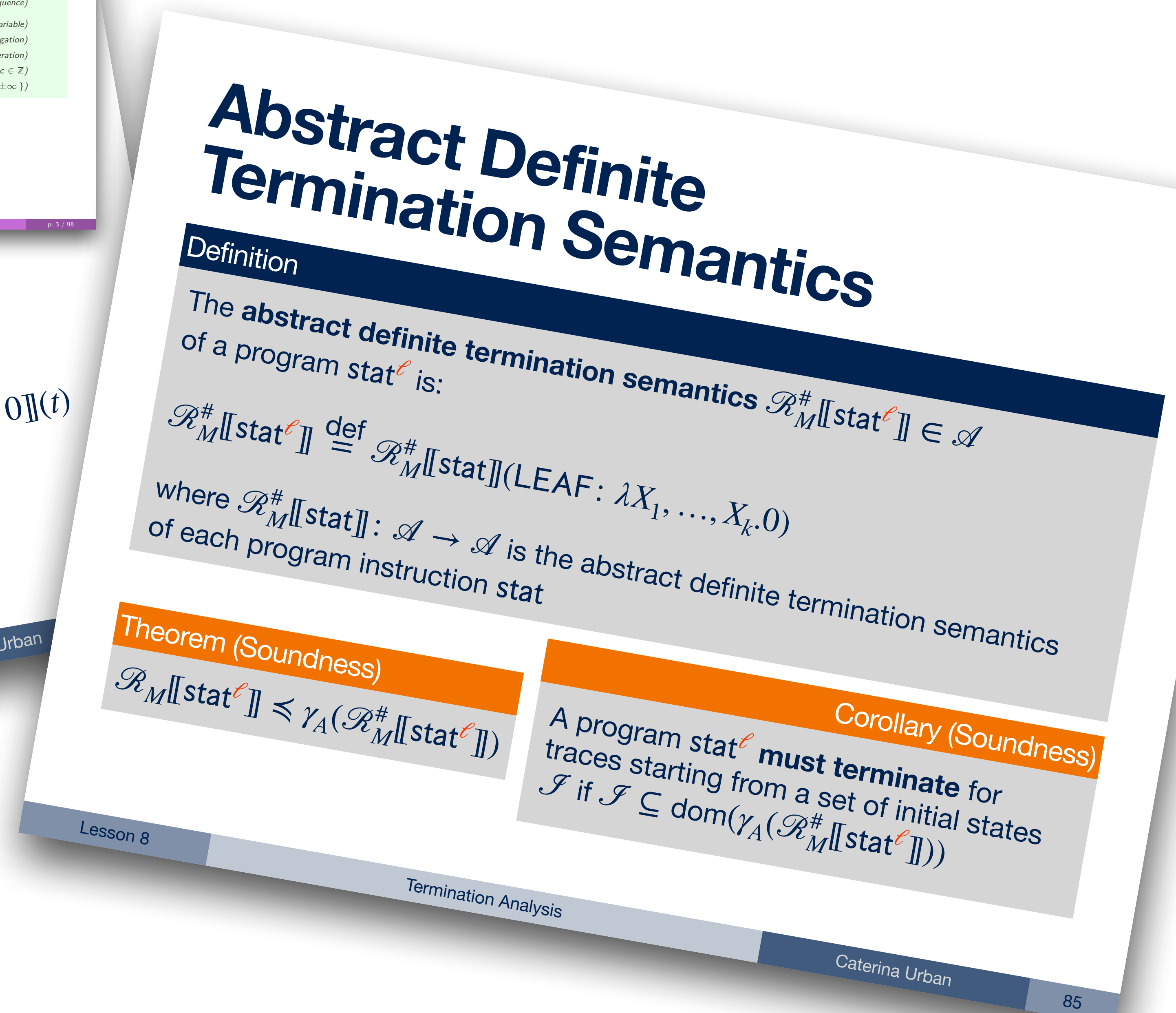
  Example:

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

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

Abstract Definite Termination Semantics





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
od⁹
```

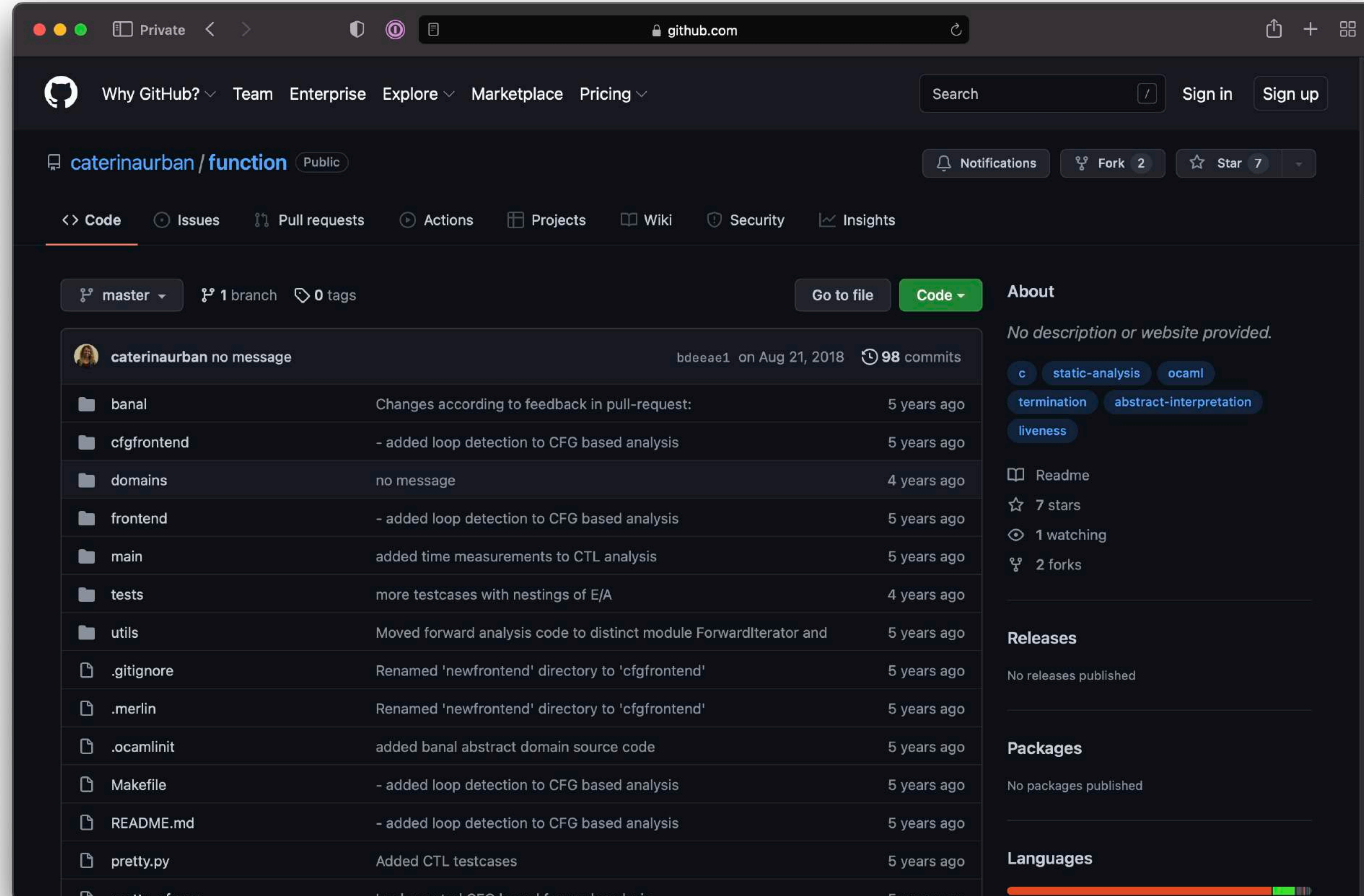
$$f_3 \stackrel{\text{def}}{=} \begin{cases} 1 & x_1 \leq 0 \lor x_2 \leq 0 \\ \omega \cdot (x_1 - 7) + 7x_1 + 3x_2 - 5 & x_1 > 0 \land x_2 > 0 \end{cases}$$

Abstract Interpretation Recipe

practical tools
targeting specific programs

algorithmic approaches
to decide program properties

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
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

Abstract Interpretation Recipe

practical tools
targeting specific programs

algorithmic approaches
to decide program properties

mathematical models
of the program behavior

Bibliography

[Cousot02] **Patrick Cousot**. Constructive Design of a Hierarchy of Semantics of a Transition System by Abstract Interpretation. In Theoretical Computer Science 277(1-2):47–103, 2002.

[Cousot12] **Patrick Cousot and Radhia Cousot**. An Abstract Interpretation Framework for Termination. In POPL, pages 245–258, 2012.

[Urban15] **Caterina Urban**. Static Analysis by Abstract Interpretation of Functional Temporal Properties of Programs. PhD Thesis, École Normale Supérieure, 2015.

[Urban17] **Nathanaël Courant and Caterina Urban**. Precise Widening Operators for Proving Termination by Abstract Interpretation. In TACAS, 2017.

extensions with **other widening heuristics**