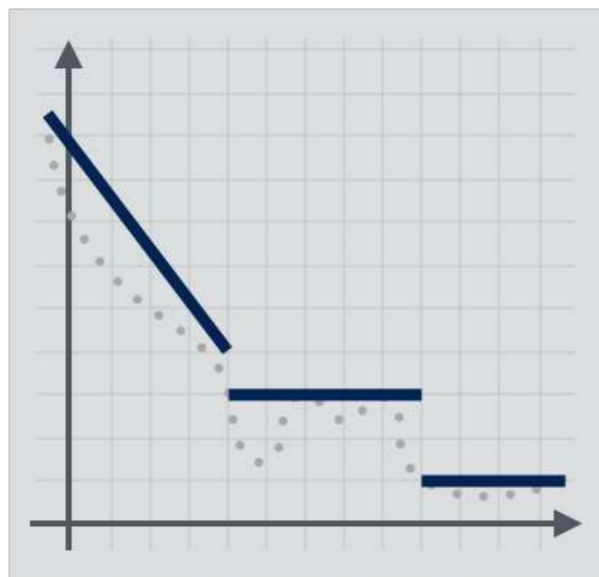


# Termination Analysis


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



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

Formal Verification: Motivation

## Historic example: Ariane 5, Flight 501



Maiden flight of the Ariane 5 Launcher, 4 June 1996.  
Cost of failure estimated at more than 370 000 000 US\$<sup>1</sup>

<sup>1</sup>M. Dowson. "The Ariane 5 Software Failure". Software Engineering Notes 22 (2): 84, March 1997.

Course 0 Introduction Antoine Miné p. 3 / 40

Formal Verification: Motivation

## How can we avoid such failures?

- Choose a safe programming language.  
C (low level) / Ada, Java, OCaml (high level)  
yet, Ariane 5 software is written in Ada
- Carefully design the software.  
many software development methods exist  
yet, critical embedded software follow strict development processes
- Test the software extensively.  
yet, the erroneous code was well tested... on Ariane 4  
**⇒ not sufficient!**

We should use **formal methods**.  
provide rigorous, mathematical insurance of correctness  
may not prove everything, but give a precise notion of what is proved

**This case triggered the first large scale static code analysis**  
PolySpace Verifier, using abstract interpretation

Course 0 Introduction Antoine Miné p. 5 / 40

that is, for **proving Safety Properties**

# Safety vs Liveness Properties

Safety and liveness trace properties

## Safety properties for traces

**Idea:** a safety property  $P$  models that “nothing bad will ever occur”

- $P$  is provable by exhaustive testing  
(observe the prefix trace semantics:  $\mathcal{T}_p(\mathcal{I}) \subseteq P$ )
- $P$  is disprovable by finding a single finite execution not in  $P$

Examples:

- any **state property**:
- **ordering**:  $P \stackrel{\text{def}}{=} \Sigma^{\omega}$   
no  $b$  can appear without  
but we can have only  $a$ ,  
(not a state property)
- but **termination**  
disproving requires ex

Course 2

## Liveness properties

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

Liveness properties model that “something good eventually occurs”

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

p. 92 / 98

## Safety Properties

“something bad  
never happens”

“something good  
eventually happens”

## Liveness Properties



Leslie Lamport

# 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

# Program Termination

# The Zune Bug

31 December 2008

unresponsive systems

30GB Zunes all over the world

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

News TCTV Events

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

Gadgets | **Headline** | Zune | Feature

## 30GB Zunes all over the world

Posted Dec 31, 2008 by [Matt Burns \(@mjburnsy\)](#)

Share 0 | in Share 0 | Tweet 0

It seems that a random bug is affecting a bunch, a bunch of Zune 30s just stopped working. No one might have a gadget Y2K going on here. [Fan boards](#) 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 detail

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

## Zune bug explained in detail

Posted Dec 31, 2008 by [Devin Coldewey](#)

Share 10 | in 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

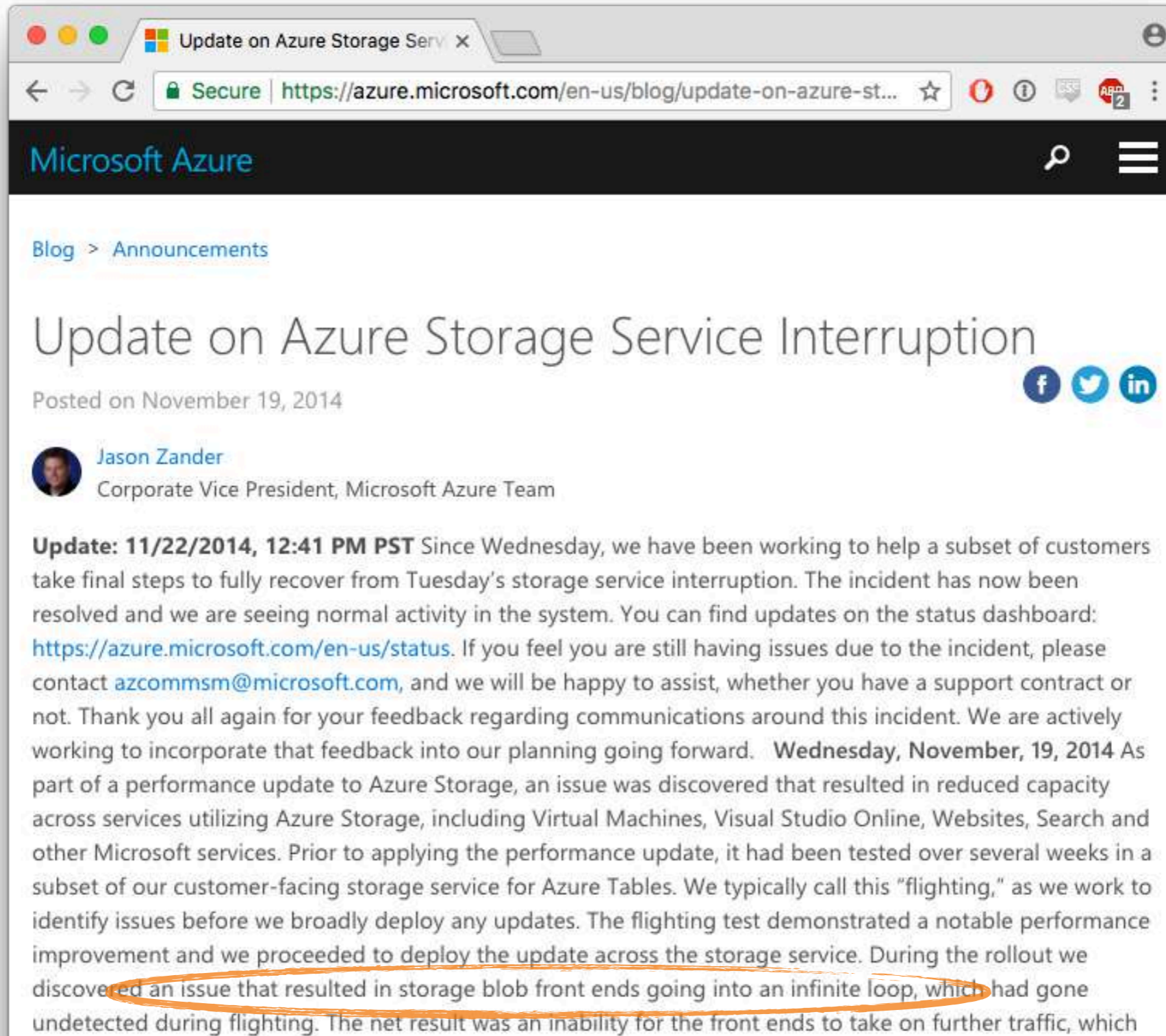
The screenshot shows a web browser window at [cve.mitre.org](http://cve.mitre.org). The page features the CVE logo and navigation links for 'CVE List', 'CNA's', 'Board', 'About', and 'News & Blog'. A 'NVD' (National Vulnerability Database) section is also present with links for 'CVSS Scores', 'CPE Info', and 'Advanced Search'. A navigation bar includes 'Search CVE List', 'Download CVE', 'Data Feeds', 'Request CVE IDs', and 'Update a CVE Entry'. A status bar indicates 'TOTAL CVE Entries: 97475'. The breadcrumb trail is 'HOME > CVE > CVE-2009-1890'. A 'Printer-Friendly View' link is available. The main content area is structured as follows:

CVE-ID	
<b>CVE-2009-1890</b>	<a href="#">Learn more at National Vulnerability Database (NVD)</a> <ul style="list-style-type: none"><li>• CVSS Severity Rating</li><li>• Fix Information</li><li>• Vulnerable Software Versions</li><li>• SCAP Mappings</li><li>• CPE Information</li></ul>
Description	
The stream_reqbody_cl function in mod_proxy_http.c in the mod_proxy module in the Apache HTTP Server before 2.3.3, when a reverse proxy is configured, does not properly handle an amount of streamed data that exceeds the Content-Length value, which allows remote attackers to cause a denial of service (CPU consumption) via crafted requests.	
References	

# Azure Storage Service

19 November 2014

Service interruptions



The screenshot shows a web browser window displaying a blog post from Microsoft Azure. The browser's address bar shows the URL <https://azure.microsoft.com/en-us/blog/update-on-azure-st...>. The page header includes the Microsoft Azure logo and navigation icons. The breadcrumb trail is 'Blog > Announcements'. The main heading is 'Update on Azure Storage Service Interruption', with social media sharing icons for Facebook, Twitter, and LinkedIn. The post is dated 'November 19, 2014' and is authored by 'Jason Zander', Corporate Vice President of the Microsoft Azure Team. The text of the post begins with an update: '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 [azcommasm@microsoft.com](mailto:azcommasm@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  $\mathcal{M} \in \mathcal{P}(\Sigma^\infty)$  **may terminate** if and only if  $\mathcal{M} \cap \Sigma^* \neq \emptyset$

## Definition

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

Finite prefix trace semantics

### Finite traces

Finite trace: finite sequence of elements from  $\Sigma$

- $\epsilon$ : empty trace (unique)
- $\sigma$ : trace of length 1 (assimilated to a state)
- $\sigma_0, \dots, \sigma_{n-1}$ : trace of length  $n$
- $\Sigma^n$ : the set of traces of length  $n$
- $\Sigma^{\leq n} \stackrel{\text{def}}{=} \bigcup_{i < n} \Sigma^i$ : the set of traces of length at most  $n$
- $\Sigma^* \stackrel{\text{def}}{=} \bigcup_{i \in \mathbb{N}} \Sigma^i$ : 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

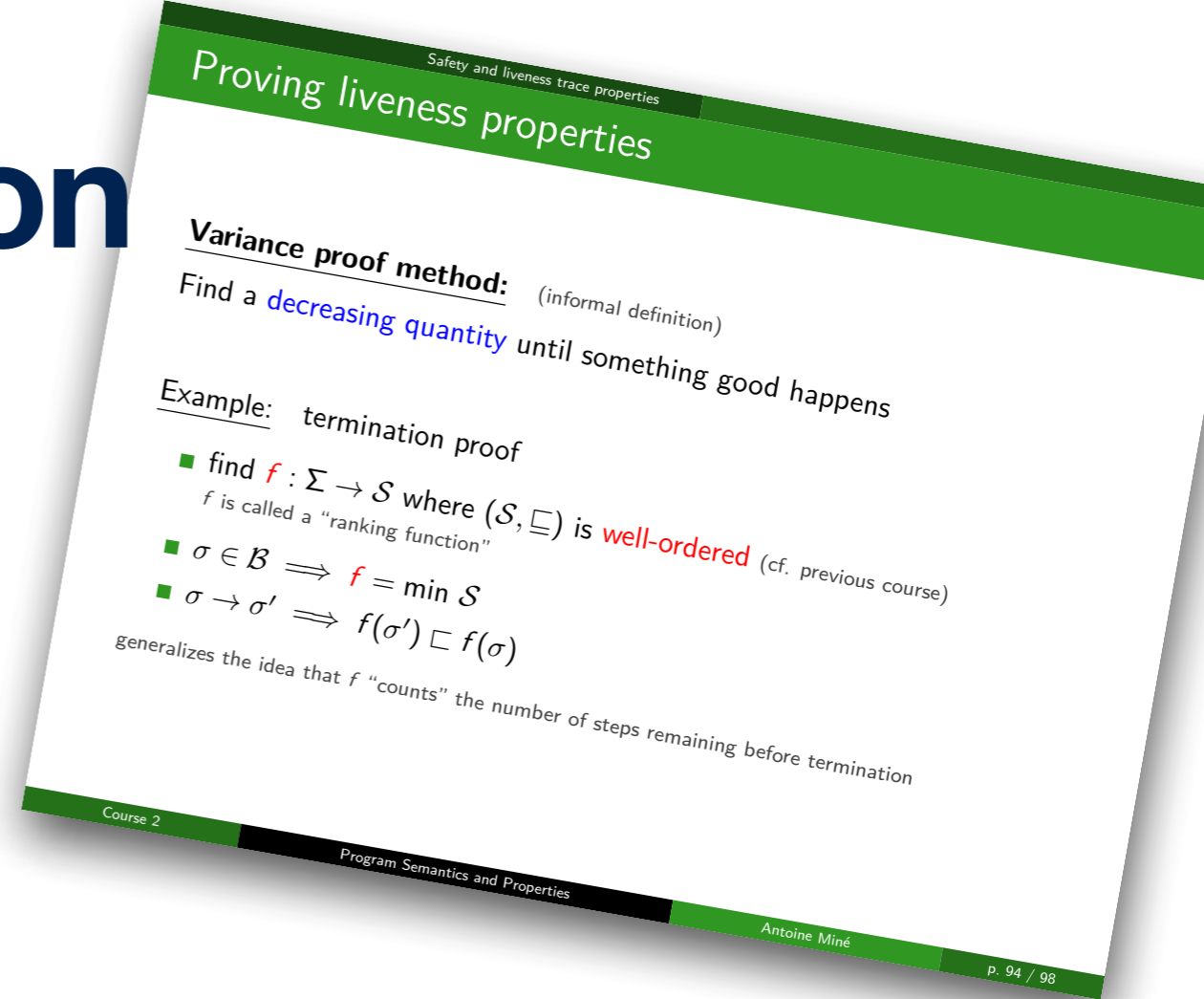
## Ranking Functions



Alan Turing



Robert W. Floyd



### Definition

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

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

# Ranking Functions

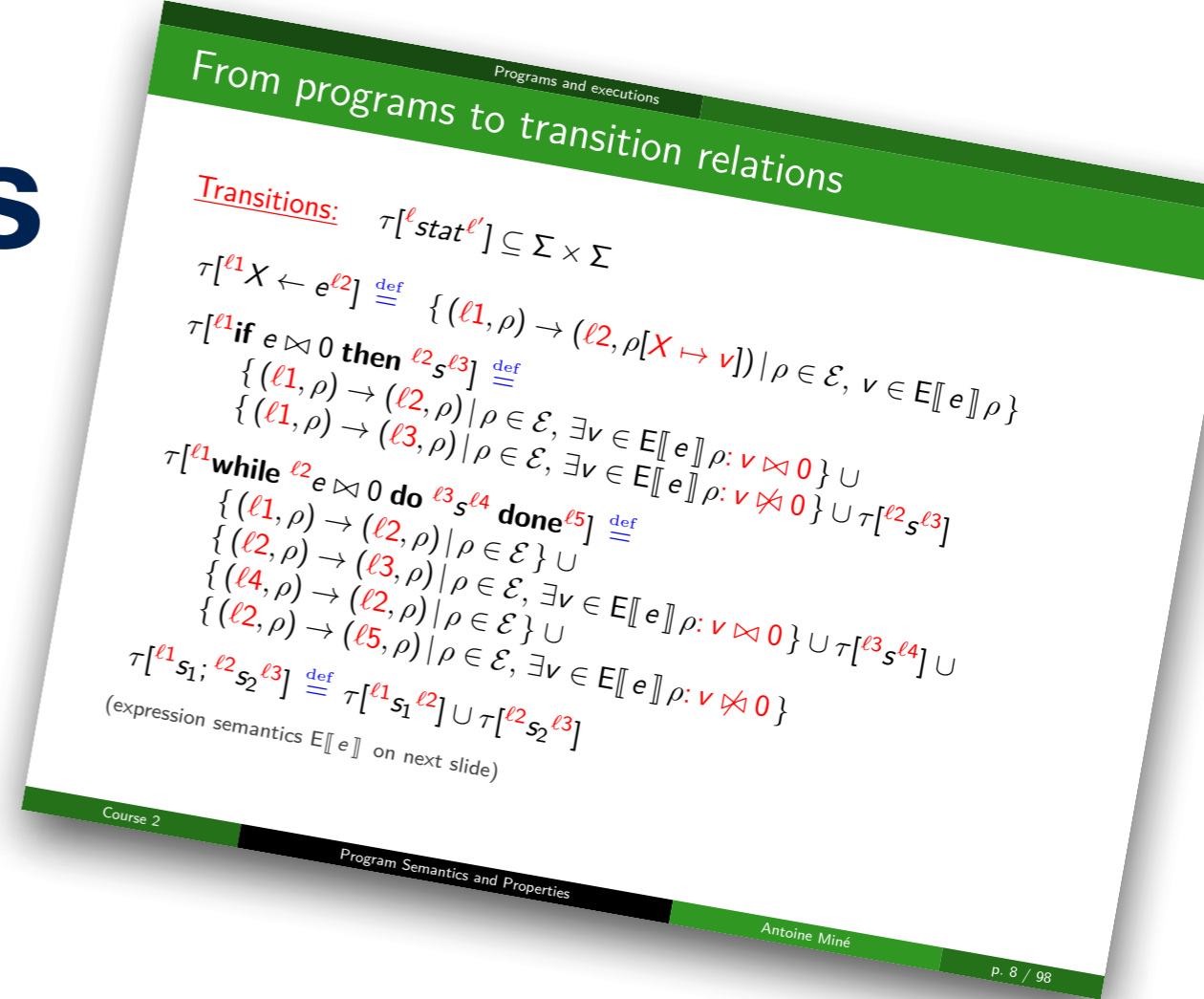
## Example (continue)

```

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

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

$$\tau \stackrel{\text{def}}{=} \{ (\mathbf{1}, \rho) \rightarrow (\mathbf{2}, \rho[X \mapsto v]) \mid \rho \in \mathcal{E}, v \in \mathbb{Z} \} \\ \cup \{ (\mathbf{2}, \rho) \rightarrow (\mathbf{3}, \rho) \mid \rho \in \mathcal{E}, \exists v \in E[[1 - x]]\rho : v < 0 \} \\ \cup \{ (\mathbf{3}, \rho) \rightarrow (\mathbf{2}, \rho[X \mapsto v]) \mid \rho \in \mathcal{E}, v \in E[[x - 1]]\rho \} \\ \cup \{ (\mathbf{2}, \rho) \rightarrow (\mathbf{4}, \rho) \mid \rho \in \mathcal{E}, \exists v \in E[[1 - x]]\rho : v \neq 0 \}$$



# Ranking Functions

## Example (continue)

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

*Most obvious ranking function:*

a mapping  $f: \Sigma \rightarrow \mathbb{Q}$

**from each program state**

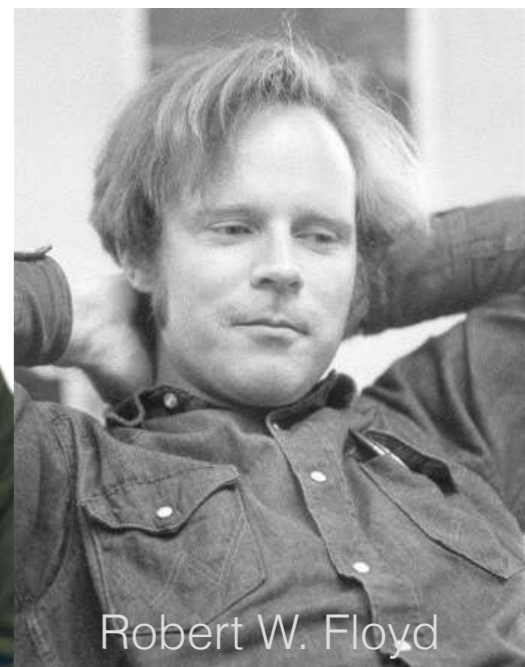
**to**

(a well-chosen upper bound on)

**the number of steps until termination**



Alan Turing



Robert W. Floyd

# Ranking Functions

## Example (continue)

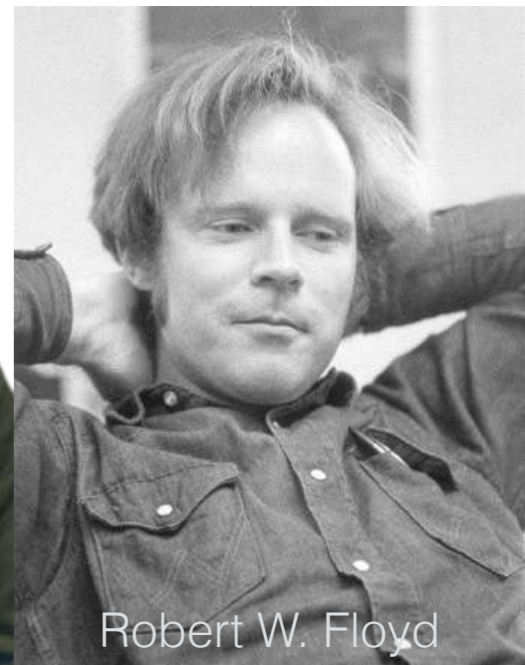
```
1x ← [-∞, +∞]
while 2(1 - x < 0) do
    3x ← x - 1
od4
```

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

$$\begin{aligned} f(\mathbf{4}) &\stackrel{\text{def}}{=} \lambda\rho.0 \\ f(\mathbf{2}) &\stackrel{\text{def}}{=} \lambda\rho. \begin{cases} 1 & 1 - \rho(x) \not< 0 \\ 2\rho(x) - 1 & 1 - \rho(x) < 0 \end{cases} \\ f(\mathbf{3}) &\stackrel{\text{def}}{=} \lambda\rho. \begin{cases} 2 & 2 - \rho(x) \not< 0 \\ 2\rho(x) - 2 & 2 - \rho(x) < 0 \end{cases} \\ f(\mathbf{1}) &\stackrel{\text{def}}{=} \lambda\rho.\omega \end{aligned}$$



Alan Turing



Robert W. Floyd

# Potential Termination

## Potential Ranking Functions

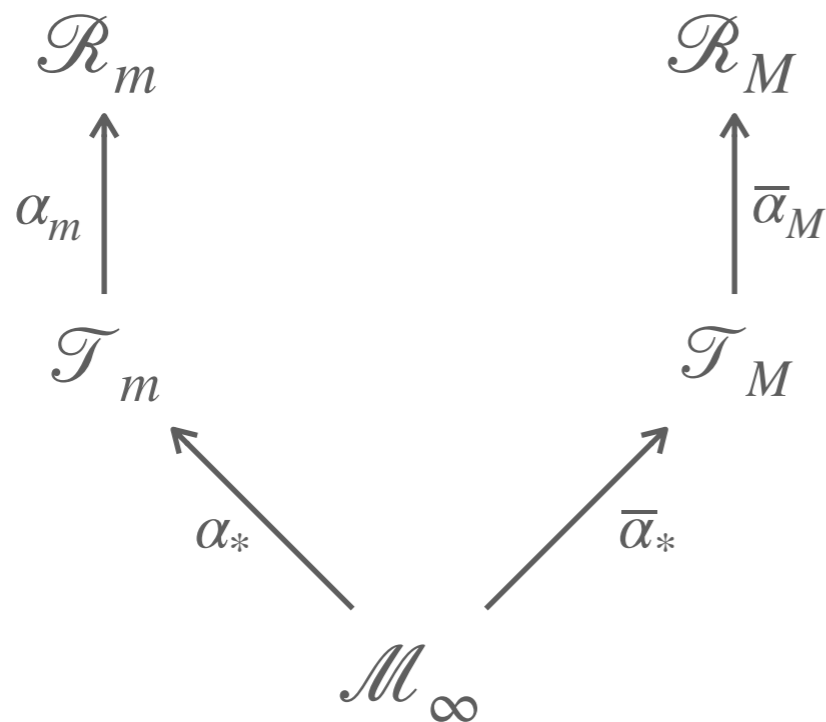
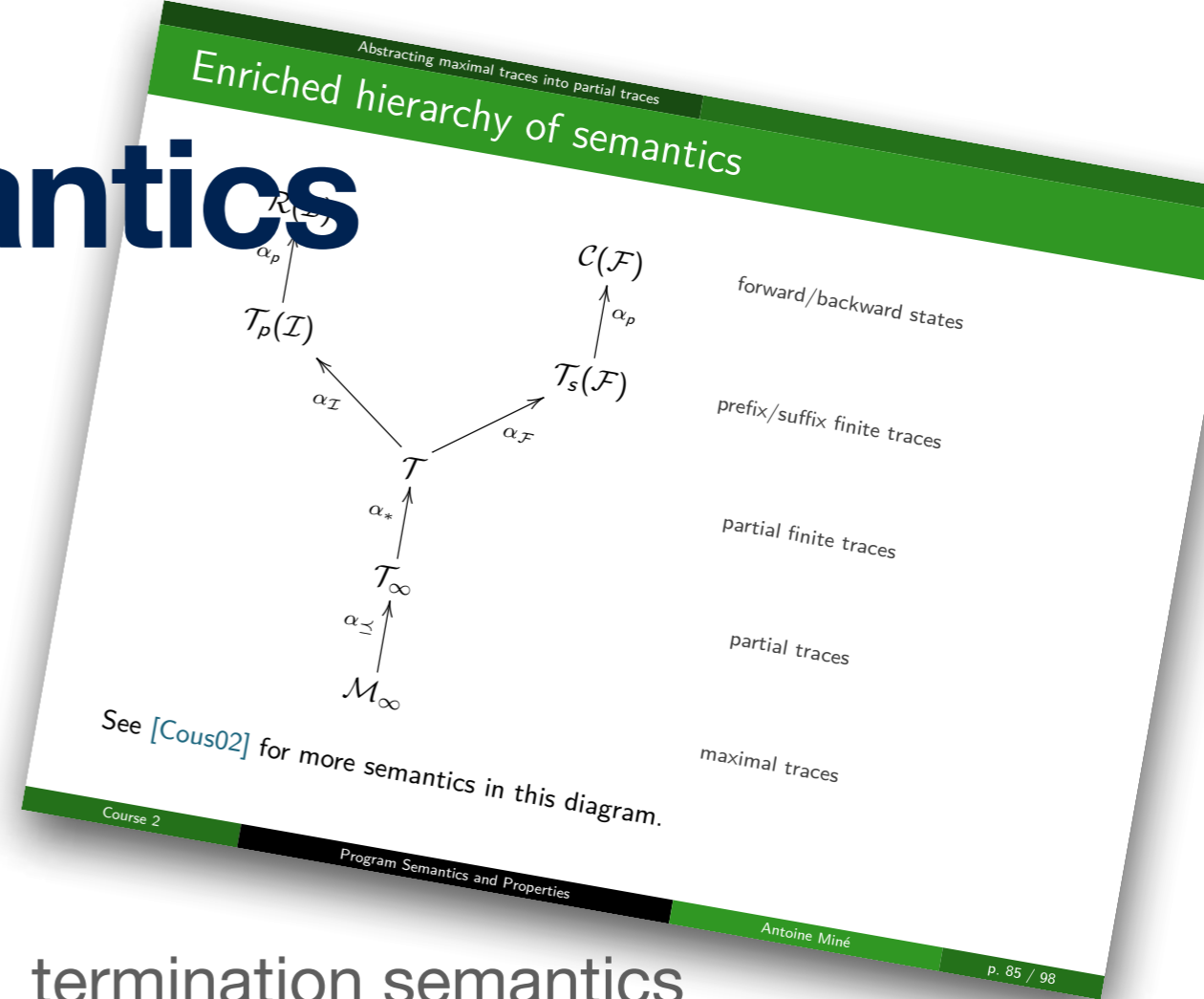
For proving potential termination, we use a *weaker* notion of ranking function, which *decreases along at least one transition* during program execution

### Definition

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

# Termination Semantics

# Hierarchy of Semantics



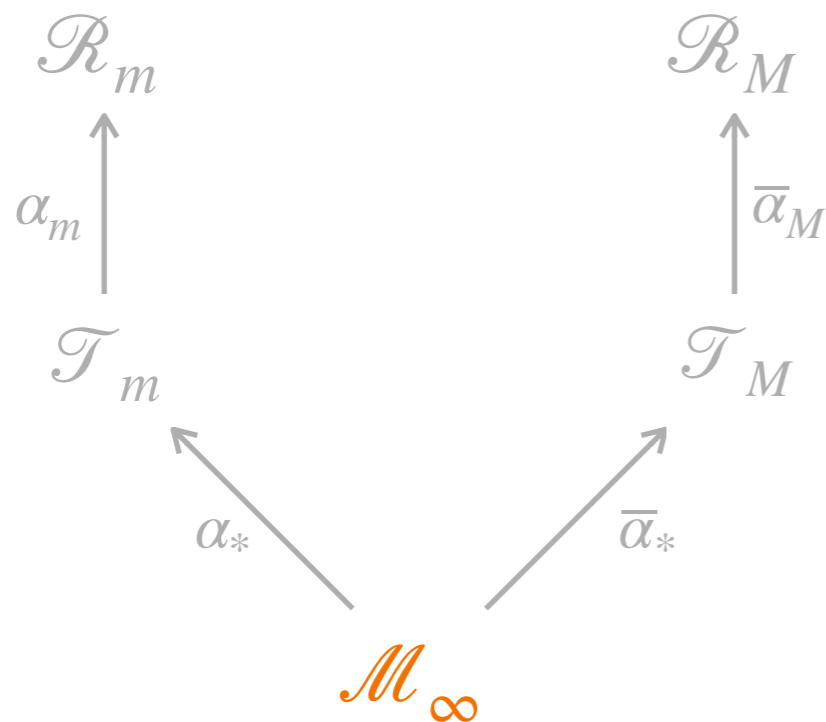
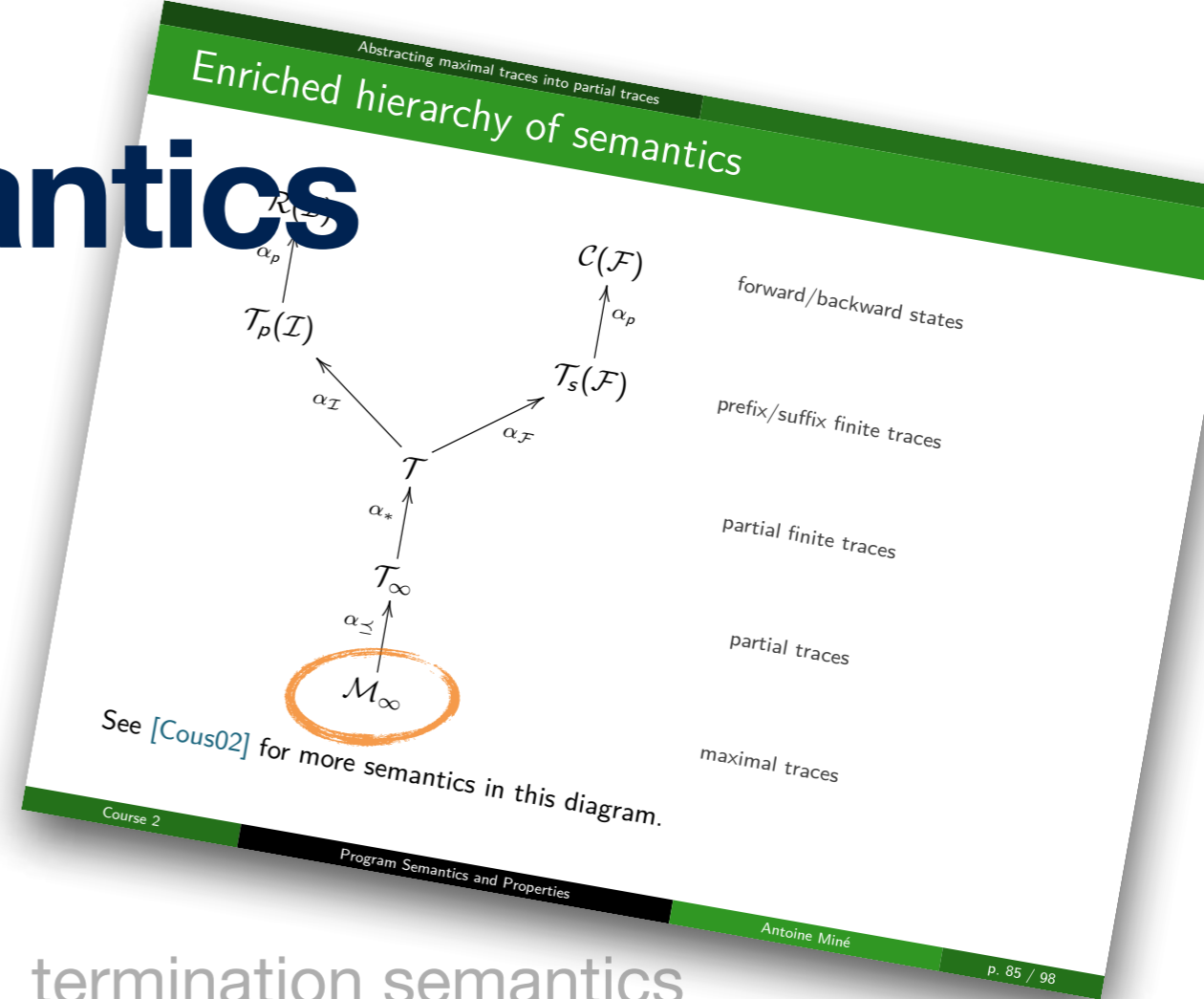
termination semantics

termination trace semantics

maximal trace semantics



# Hierarchy of Semantics



termination semantics

termination trace semantics

maximal trace semantics

# Maximal Trace Semantics

## Example

```
while 1([-∞, +∞] ≠ 0) do
  2skip
od3
```

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

$$\tau \stackrel{\text{def}}{=} \left\{ \begin{array}{l} (\mathbf{1}, \rho) \rightarrow (\mathbf{2}, \rho) \mid \rho \in \mathcal{E} \\ (\mathbf{2}, \rho) \rightarrow (\mathbf{1}, \rho) \mid \rho \in \mathcal{E} \\ (\mathbf{1}, \rho) \rightarrow (\mathbf{3}, \rho) \mid \rho \in \mathcal{E} \end{array} \right\}$$

$$\mathcal{M}_\infty \stackrel{\text{def}}{=} \left\{ \begin{array}{l} (\mathbf{1}, \rho)(\mathbf{2}, \rho)^*(\mathbf{3}, \rho) \mid \rho \in \mathcal{E} \\ (\mathbf{1}, \rho)(\mathbf{2}, \rho)^\omega \mid \rho \in \mathcal{E} \end{array} \right\}$$

Maximal trace semantics

**Maximal traces:**  $\mathcal{M}_\infty \in \mathcal{P}(\Sigma^\infty)$

- sequences of states linked by the transition relation  $\tau$
- start in any state ( $\mathcal{I} = \Sigma$ , technical requirement for the fixpoint characterization)
- either finite and **stop in a blocking state** ( $\mathcal{F} = \mathcal{B}$ )
- or **infinite**

$$\mathcal{M}_\infty \stackrel{\text{def}}{=} \left\{ \begin{array}{l} \sigma_0, \dots, \sigma_n \in \Sigma^* \mid \sigma_n \in \mathcal{B}, \forall i < n: \sigma_i \rightarrow \sigma_{i+1} \\ \sigma_0, \dots, \sigma_n, \dots \in \Sigma^\omega \mid \forall i < \omega: \sigma_i \rightarrow \sigma_{i+1} \end{array} \right\}$$

(can be anchored at  $\mathcal{I}$  and  $\mathcal{F}$  as:  $\mathcal{M}_\infty \cap (\mathcal{I} \cdot \Sigma^\infty) \cap ((\Sigma^* \cdot \mathcal{F}) \cup \Sigma^\omega)$ )

Course 2 Program Semantics and Properties Antoine Miné p. 72 / 98

Maximal trace semantics

## Least fixpoint formulation of maximal traces

Antoine Miné p. 72 / 98

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

**Fixpoint fusion:**

$\mathcal{M}_\infty \cap \Sigma^*$  is best defined on  $(\mathcal{P}(\Sigma^*), \subseteq, \cup, \cap, \emptyset, \Sigma^*)$ .  
 $\mathcal{M}_\infty \cap \Sigma^\omega$  is best defined on  $(\mathcal{P}(\Sigma^\omega), \supseteq, \cap, \cup, \Sigma^\omega, \emptyset)$ , the **dual lattice**.  
 (we transform the greatest fixpoint into a least fixpoint!)

We mix them into a **new** complete lattice  $(\mathcal{P}(\Sigma^\infty), \sqsubseteq, \sqcup, \sqcap, \perp, \top)$ :

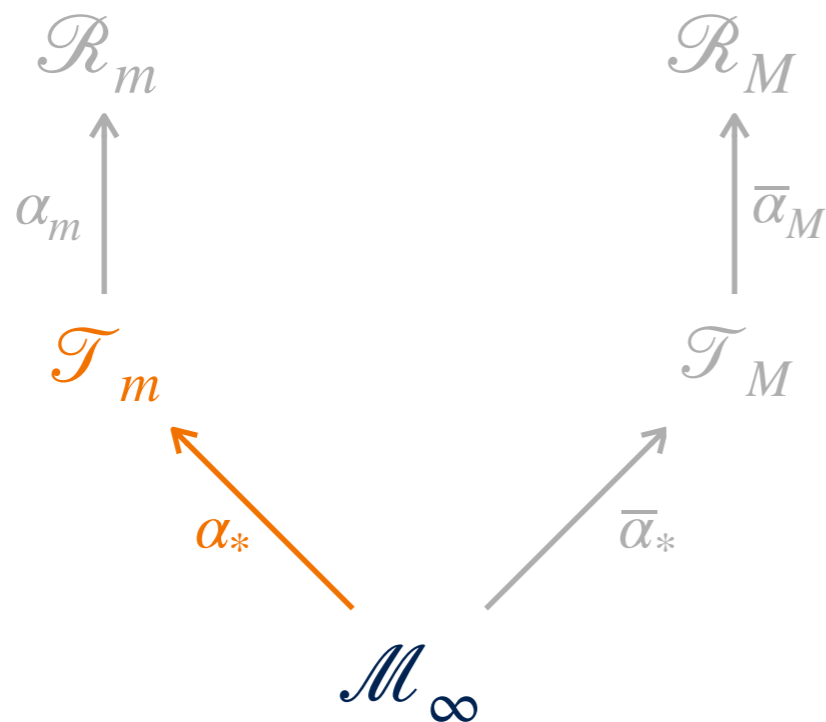
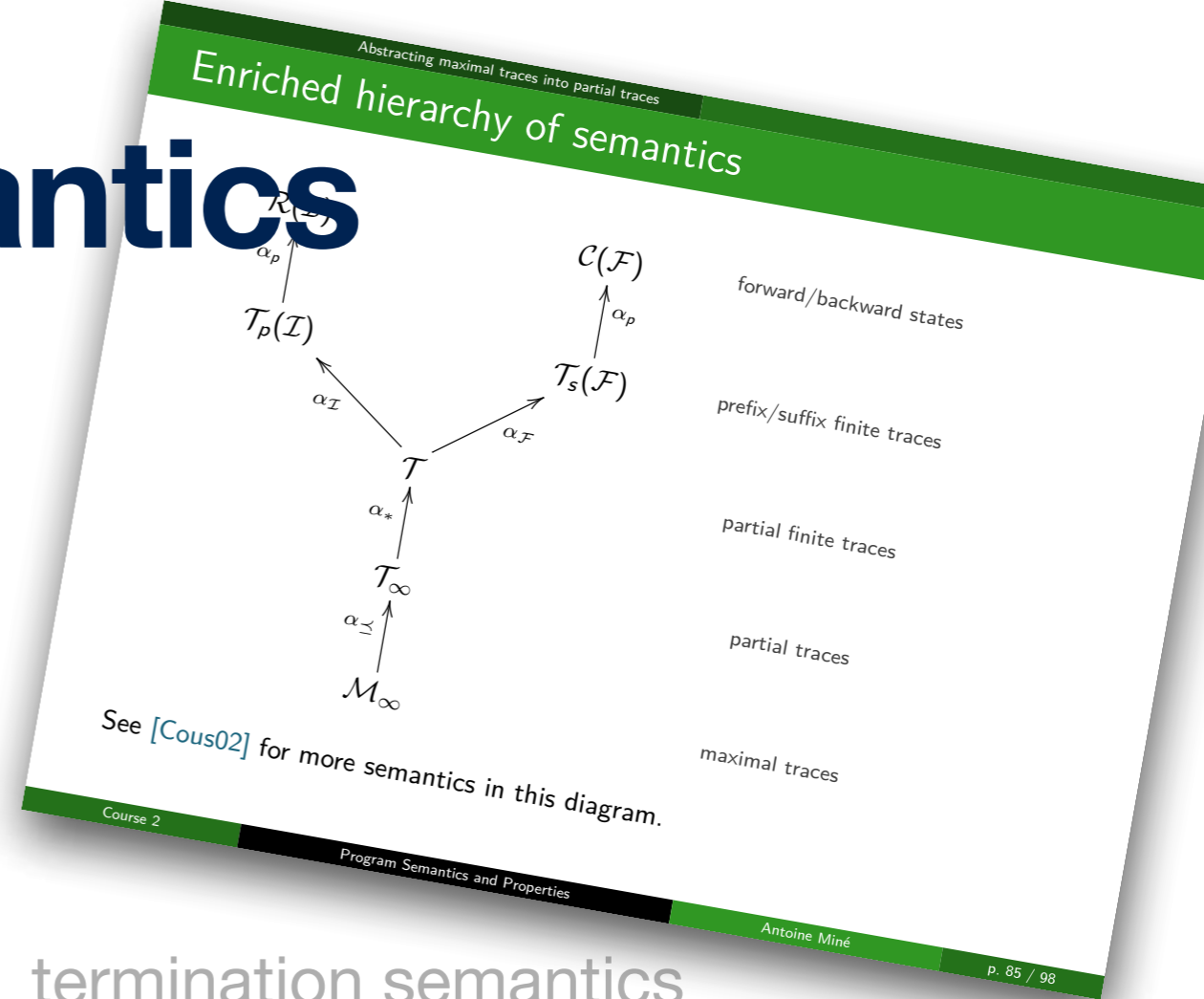
- $A \sqsubseteq B \stackrel{\text{def}}{\iff} (A \cap \Sigma^*) \subseteq (B \cap \Sigma^*) \wedge (A \cap \Sigma^\omega) \supseteq (B \cap \Sigma^\omega)$
- $A \sqcup B \stackrel{\text{def}}{=} ((A \cap \Sigma^*) \cup (B \cap \Sigma^*)) \cup ((A \cap \Sigma^\omega) \cap (B \cap \Sigma^\omega))$
- $A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

In this lattice,  $\mathcal{M}_\infty = \text{lfp } F_s$  where  $F_s(T) \stackrel{\text{def}}{=} \mathcal{B} \cup \tau \cap T$

(proof on next slides)

Course 2 Program Semantics and Properties Antoine Miné p. 76 / 98

# Hierarchy of Semantics



termination semantics

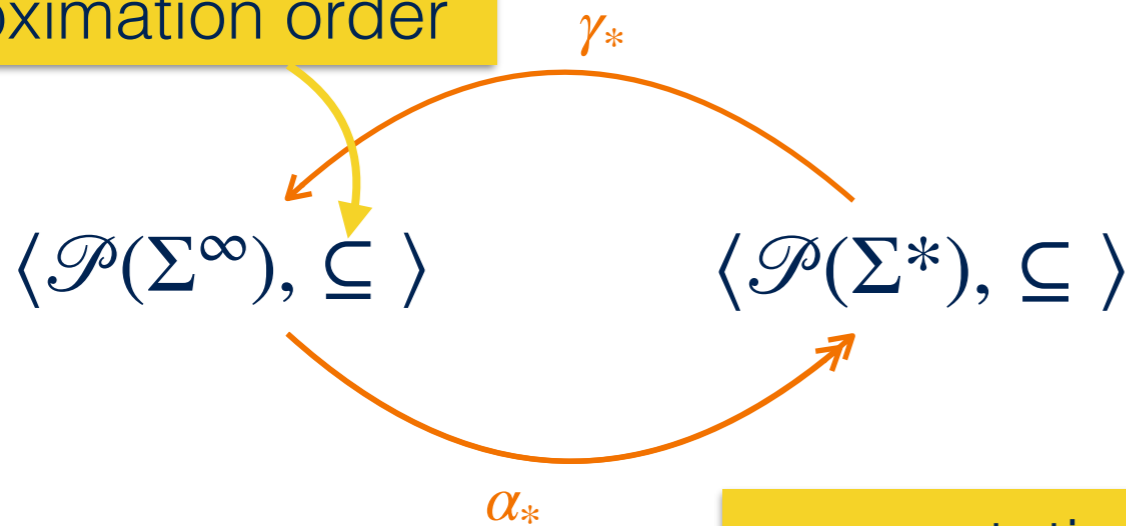
termination trace semantics

maximal trace semantics

# Potential Termination Trace Semantics

## Potential Termination Abstraction

approximation order



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

$$\gamma_*(T) \stackrel{\text{def}}{=} T \cup \Sigma^\omega$$

Example:

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

Galois connections

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

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

which is noted  $(C, \leq) \xleftrightarrow[\alpha]{\gamma} (A, \sqsubseteq)$ .

- $\alpha$  is the upper adjoint or abstraction;  $A$  is the abstract domain.
- $\gamma$  is the lower adjoint or concretization;  $C$  is the concrete domain.

Course 1 Order Theory Antoine Miné p. 48 / 69

### Least fixpoint formulation of maximal traces

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

Fixpoint fusion:

$\mathcal{M}_\infty \cap \Sigma^*$  is best defined on  $(\mathcal{P}(\Sigma^*), \subseteq, \cup, \cap, \emptyset, \Sigma^*)$ .  
 $\mathcal{M}_\infty \cap \Sigma^\omega$  is best defined on  $(\mathcal{P}(\Sigma^\omega), \supseteq, \cap, \cup, \Sigma^\omega, \emptyset)$ , the dual lattice.  
 (we transform the greatest fixpoint into a least fixpoint!)

We mix them into a new complete lattice  $(\mathcal{P}(\Sigma^\infty), \sqsubseteq, \sqcup, \sqcap, \perp, \top)$ :

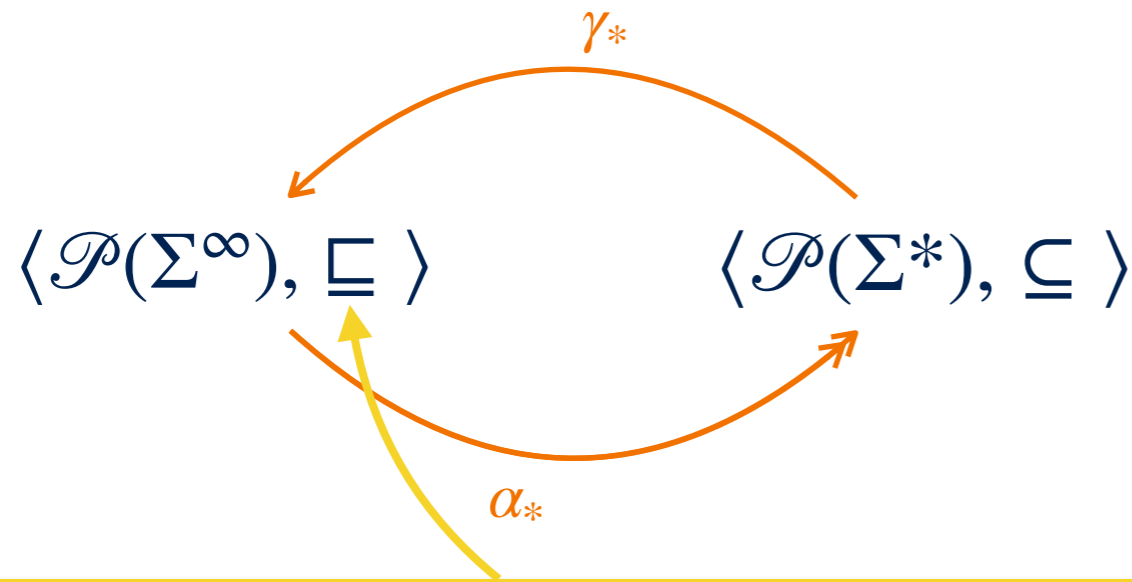
- $A \sqsubseteq B \stackrel{\text{def}}{\iff} (A \cap \Sigma^*) \subseteq (B \cap \Sigma^*) \wedge (A \cap \Sigma^\omega) \supseteq (B \cap \Sigma^\omega)$
- $A \sqcup B \stackrel{\text{def}}{=} ((A \cap \Sigma^*) \cup (B \cap \Sigma^*)) \cup ((A \cap \Sigma^\omega) \cap (B \cap \Sigma^\omega))$
- $A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

In this lattice,  $\mathcal{M}_\infty = \text{lfp } F_s$  where  $F_s(T) \stackrel{\text{def}}{=} B \cup \tau \cap T$

(proof on next slides)

# Potential Termination Trace Semantics

## Finite Trace Abstraction



approximation and computational order coincide

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

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

Galois connections

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

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

which is noted  $(C, \leq) \xleftrightarrow[\alpha]{\gamma} (A, \sqsubseteq)$ .

- $\alpha$  is the upper adjoint or abstraction;  $A$  is the abstract domain.
- $\gamma$  is the lower adjoint or concretization;  $C$  is the concrete domain.

Course 1 Order Theory Antoine Miné p. 48 / 69

Abstracting maximal traces into partial traces

### 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:

$$(\mathcal{P}(\Sigma^\infty), \sqsubseteq) \xleftrightarrow[\alpha_*]{\gamma_*} (\mathcal{P}(\Sigma^*), \subseteq)$$

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

(proof on next slide)

Course 2 Program Semantics and Properties Antoine Miné p. 83 / 98

# Potential Termination Trace Semantics

## Kleenean Fixpoint Transfer

- $\langle \mathcal{P}(\Sigma^\infty), \sqsubseteq \rangle$
- $\mathcal{M}_\infty \stackrel{\text{def}}{=} \text{lfp}^{\sqsubseteq} F_s$   
 $F_s(T) \stackrel{\text{def}}{=} \mathcal{B} \cup \tau \frown T$
- $\langle \mathcal{P}(\Sigma^*), \subseteq \rangle$
- $\alpha_*: \mathcal{P}(\Sigma^\infty) \rightarrow \mathcal{P}(\Sigma^*)$   
 $\alpha_*(T) \stackrel{\text{def}}{=} T \cap \Sigma^*$

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

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

Fixpoint transfer

Fixpoint approximations

if we have:

- a Galois connection  $(C, \leq) \xleftrightarrow[\alpha]{\gamma} (A, \sqsubseteq)$  between CPOs
- monotonic concrete and abstract functions  $f: C \rightarrow C, f^\#: A \rightarrow A$
- a commutation condition  $\alpha \circ f = f^\# \circ \alpha$
- an element  $a$  and its abstraction  $a^\# = \alpha(a)$

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

### Theorem

Let  $\langle C, \leq \rangle$  and  $\langle A, \sqsubseteq \rangle$  be complete partial orders, let  $f: C \rightarrow C$  and  $f^\#: A \rightarrow A$  be monotonic functions, and let  $\alpha: C \rightarrow A$  be a continuous abstraction function 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  $\alpha(\text{lfp}_a^{\leq} f) = \text{lfp}_{a^\#}^{\sqsubseteq} f^\#$ .

# Potential Termination Trace Semantics

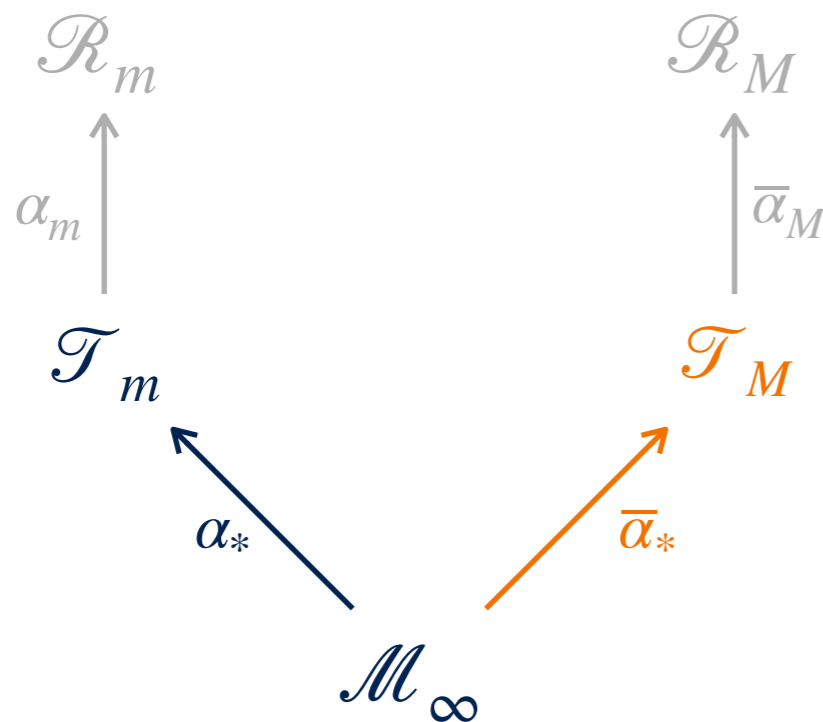
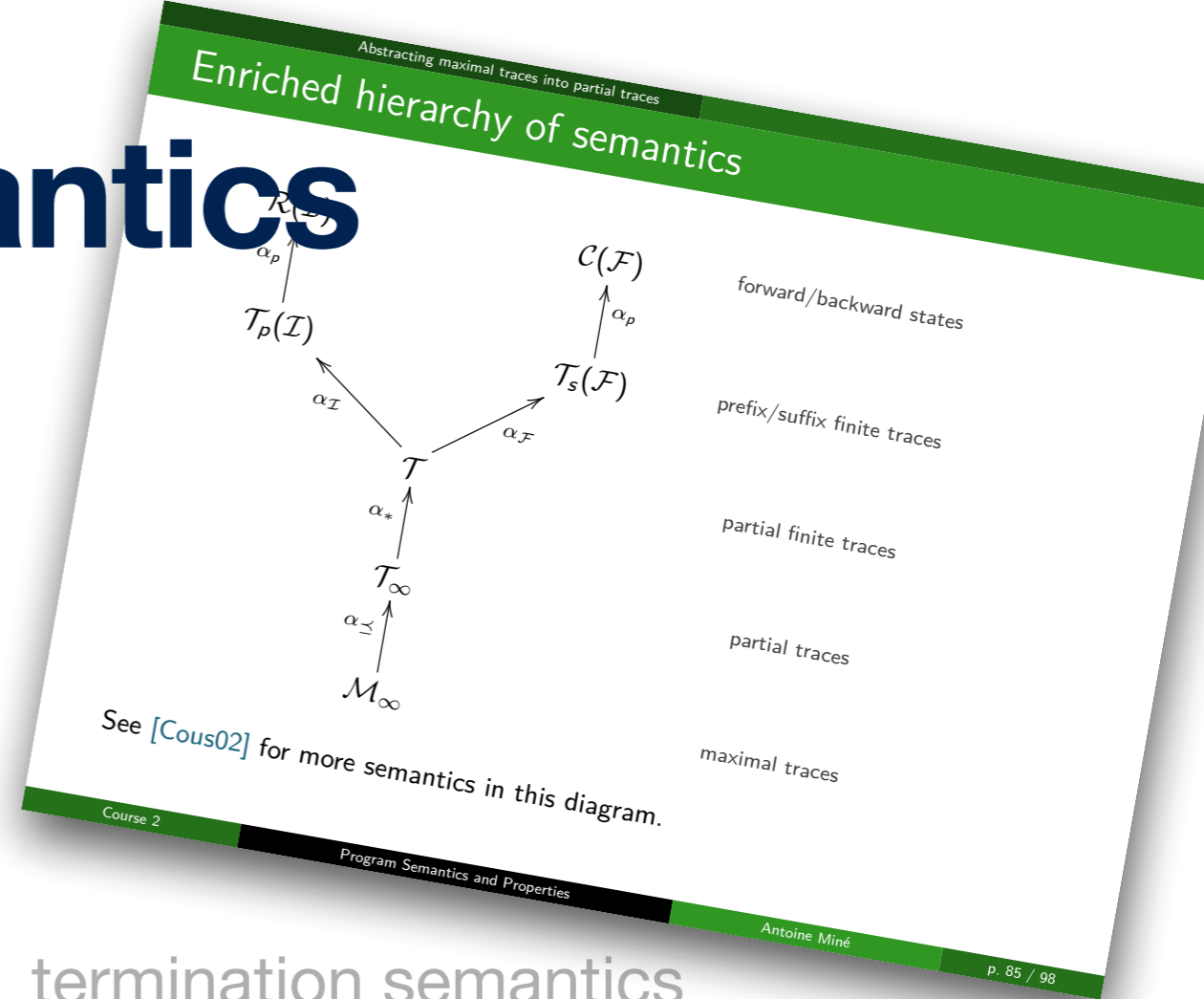
## Example

```
while 1( $[-\infty, +\infty] \neq 0$ ) do  
  2skip  
od3
```

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

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

# Hierarchy of Semantics



termination semantics

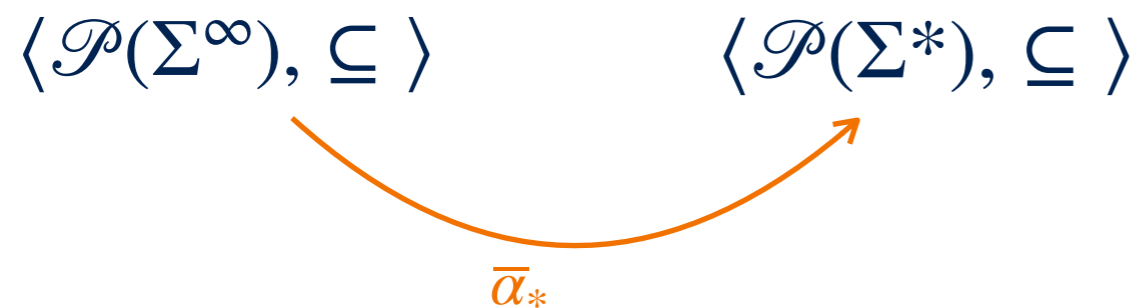
termination trace semantics

maximal trace semantics



# Definite Termination Trace Semantics

## Definite Termination Abstraction



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

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

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

Example:

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

# Definite Termination Trace Semantics

## Tarskian Fixpoint Transfer

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

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

$$\bar{F}_*(T) \stackrel{\text{def}}{=} \mathcal{B} \cup ((\tau \frown T) \cap \neg(\tau \frown \neg T))$$

## Theorem

Let  $\langle C, \leq, \vee, \wedge, \perp, \top \rangle$  and  $\langle A, \sqsubseteq, \sqcup, \sqcap, \perp^\#, \top^\# \rangle$  be complete lattices, let  $f: C \rightarrow C$  and  $f^\#: A \rightarrow A$  be monotonic functions, and let  $\alpha: C \rightarrow A$  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  $\alpha$  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

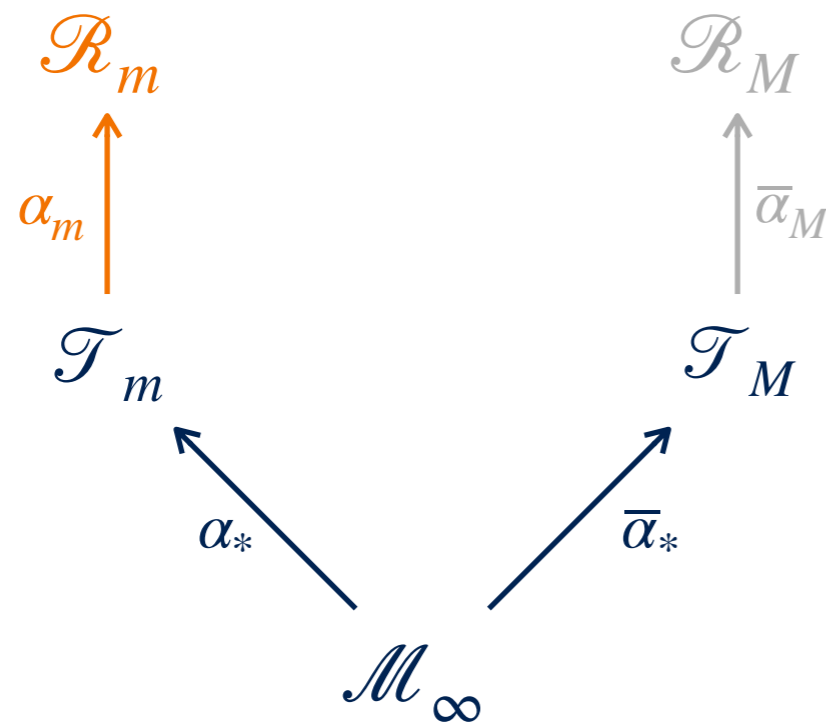
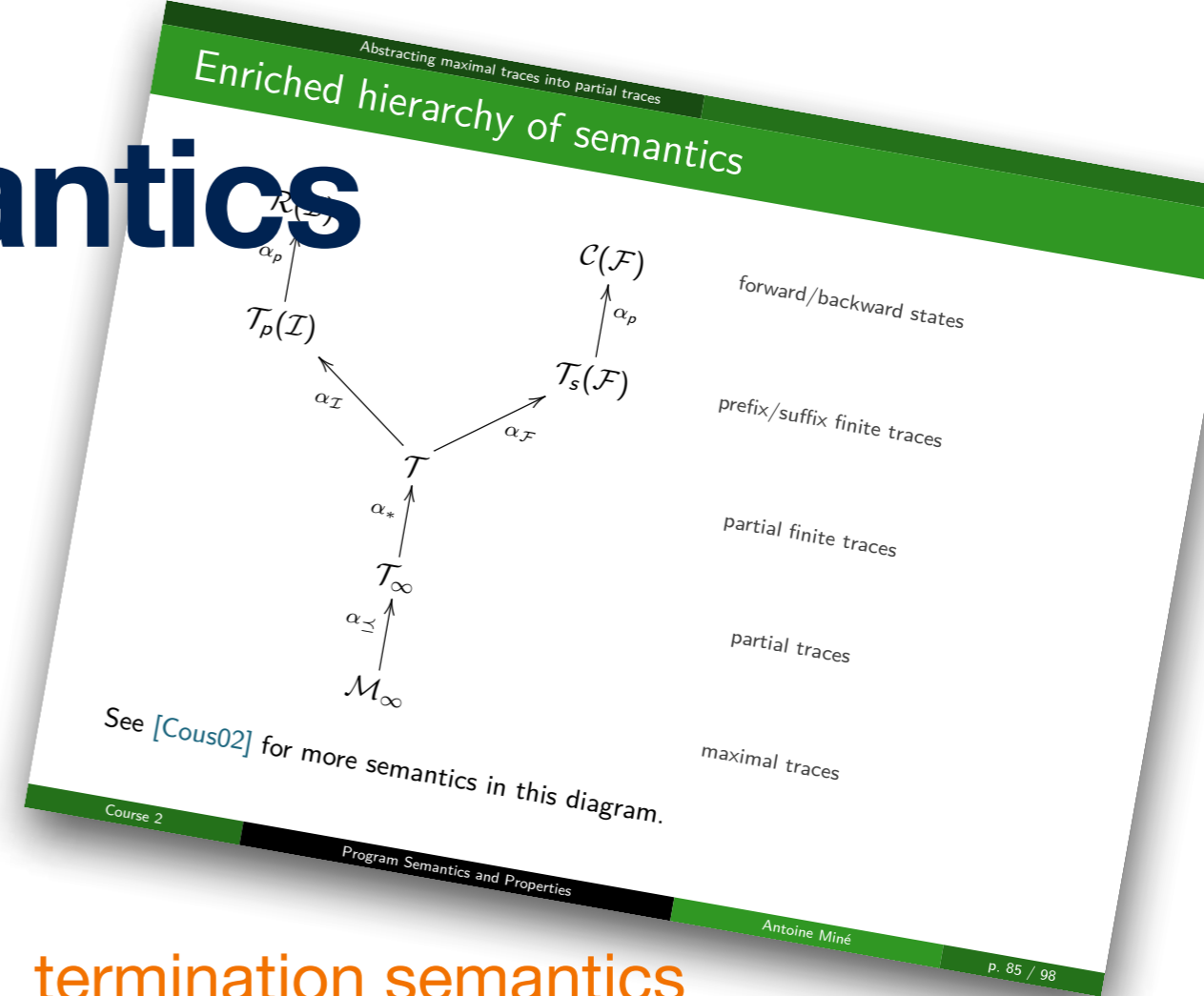
## Example

```
while 1([-∞, +∞] ≠ 0) do  
  2skip  
od3
```

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

$$\mathcal{T}_M \stackrel{\text{def}}{=} \emptyset$$

# Hierarchy of Semantics



termination semantics

termination trace semantics

maximal trace semantics

# Potential Termination Semantics

## Potential Ranking Abstraction



count execution steps backwards

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

$$\langle \Sigma \rightarrow \mathbb{O}, \leq \rangle$$

$\alpha_m$

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

approximation order

$$\alpha_m(T) \stackrel{\text{def}}{=} \alpha_v(\vec{\alpha}(T))$$

$$\alpha_v(\emptyset) \stackrel{\text{def}}{=} \emptyset$$

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

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

# Potential Termination Semantics

approximation and computational order coincide

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

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

Backward state co-reachability semantics

## Backward state co-reachability

$\mathcal{C}(\mathcal{F})$ : states **co-reachable** from  $\mathcal{F}$  in the transition system:

$$\mathcal{C}(\mathcal{F}) \stackrel{\text{def}}{=} \{\sigma \mid \exists n \geq 0, \sigma_0, \dots, \sigma_n: \sigma = \sigma_0, \sigma_n \in \mathcal{F}, \forall i: \sigma_i \rightarrow \sigma_{i+1}\}$$

$$= \bigcup_{n \geq 0} \text{pre}_\tau^n(\mathcal{F})$$

where  $\text{pre}_\tau(S) \stackrel{\text{def}}{=} \{\sigma \mid \exists \sigma' \in S: \sigma \rightarrow \sigma'\}$  ( $\text{pre}_\tau = \text{post}_{\tau^{-1}}$ )

$\mathcal{C}(\mathcal{F})$  can also be expressed in **fixpoint** form:

$$\mathcal{C}(\mathcal{F}) = \text{lfp } F_C \text{ where } F_C(S) \stackrel{\text{def}}{=} \mathcal{F} \cup \text{pre}_\tau(S)$$

**Justification:**  $\mathcal{C}(\mathcal{F})$  in  $\tau$  is exactly  $\mathcal{R}(\mathcal{F})$  in  $\tau^{-1}$

**Alternate characterization:**  $\mathcal{C}(\mathcal{F}) = \text{lfp}_{\mathcal{F}} G_C$  where  $G_C(S) = S \cup \text{pre}_\tau(S)$

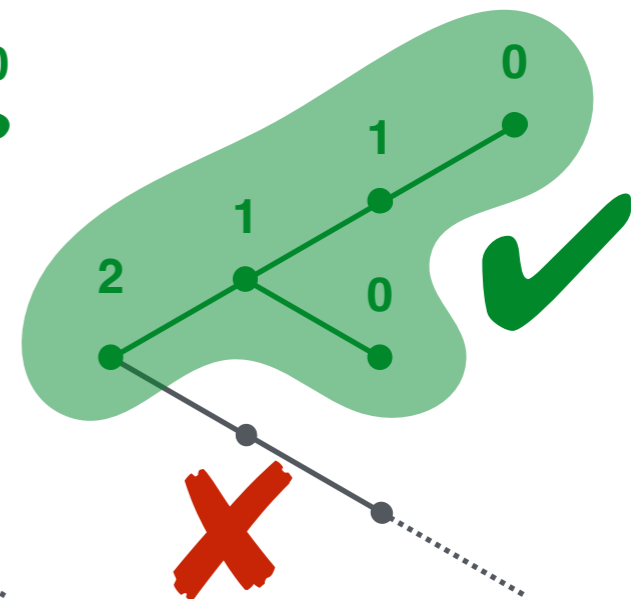
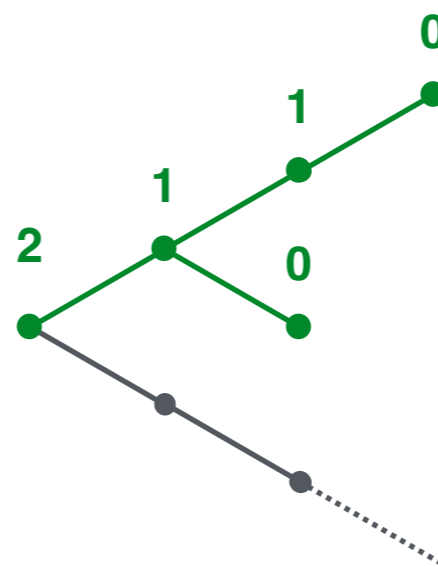
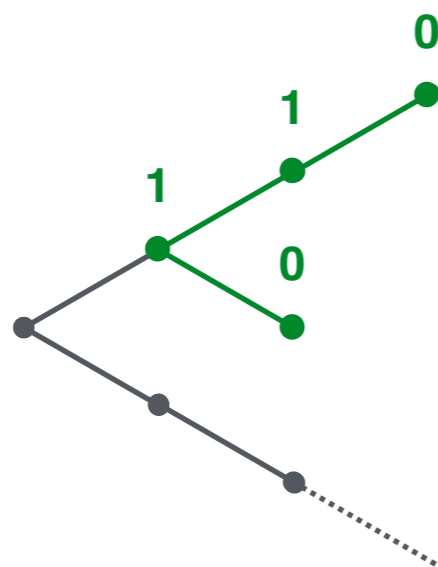
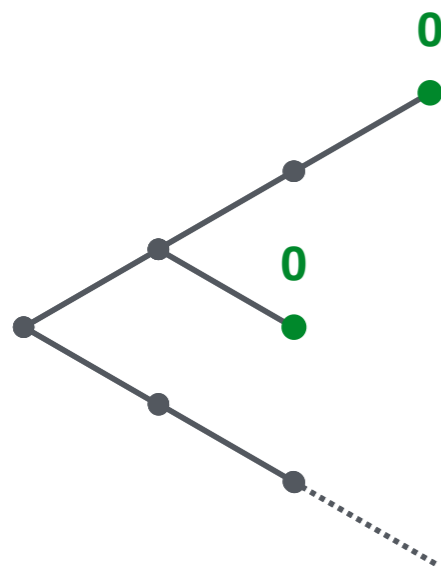
Course 2 Program Semantics and Properties Antoine Miné p. 52 / 98

# Potential Termination Semantics

approximation and computational order coincide

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

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



## 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**:

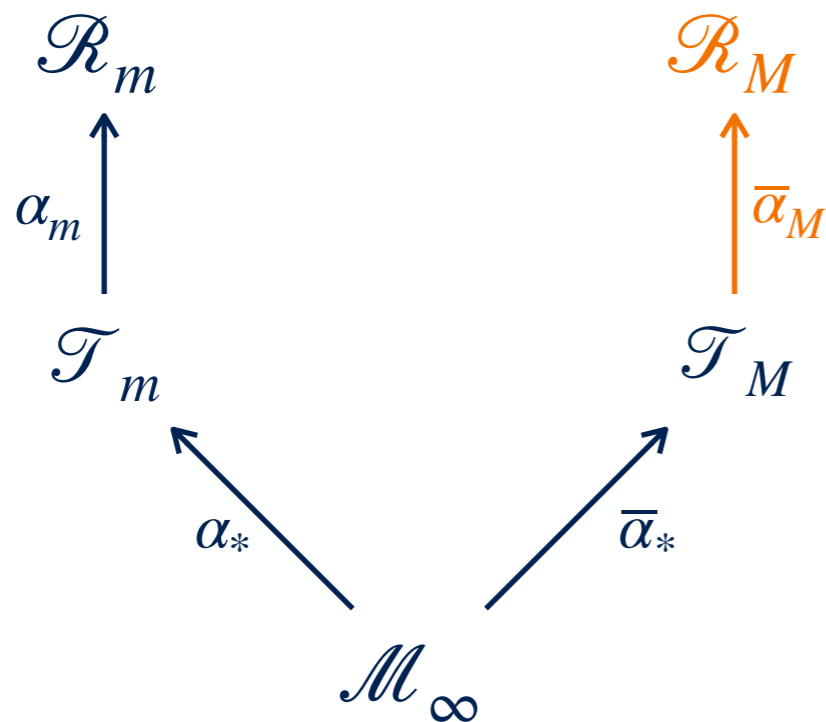
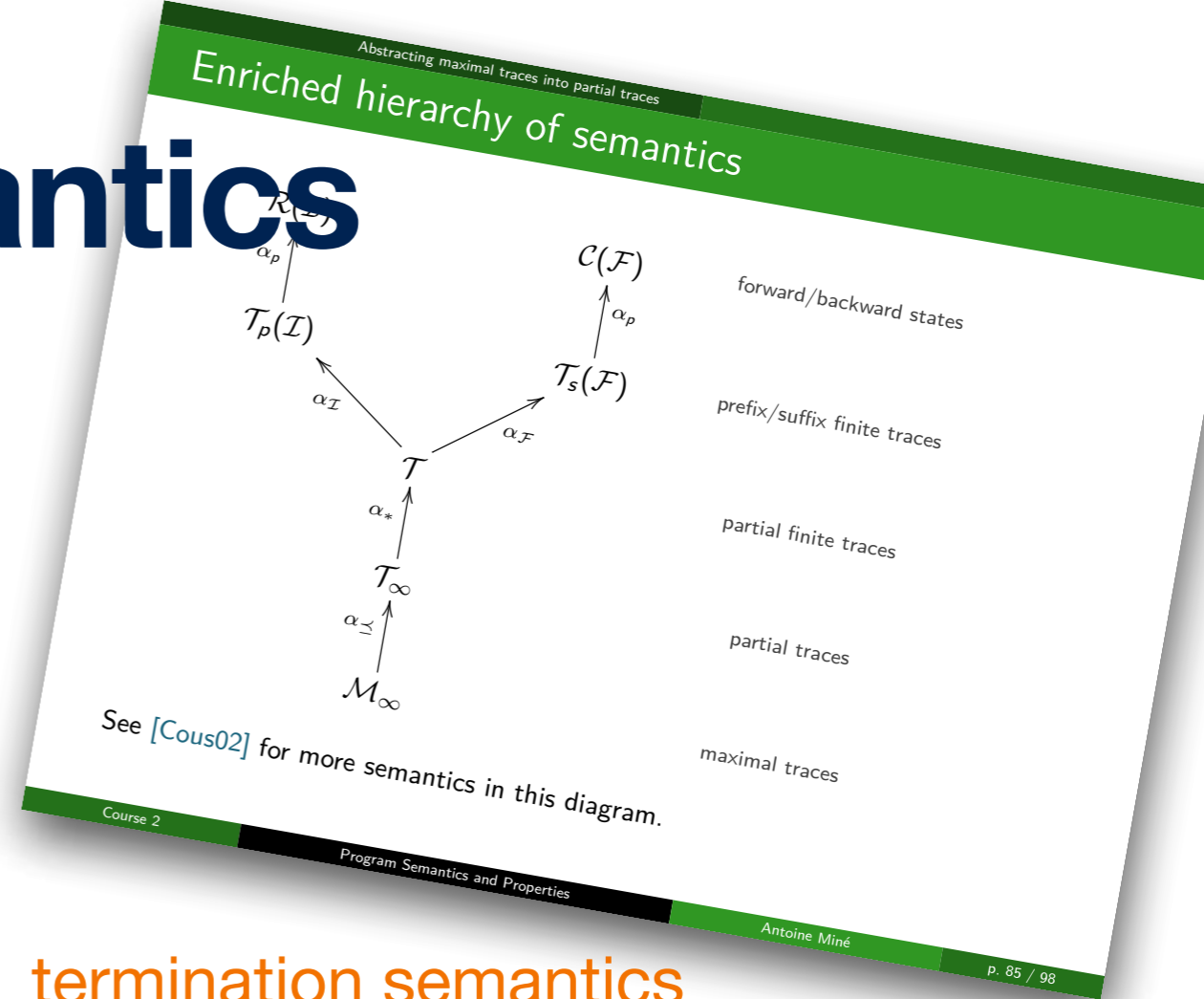
$$\mathcal{R}_m \stackrel{\text{def}}{=} \alpha_m(\mathcal{T}_m) = \text{lfp}^{\preceq} F_m$$

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

**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

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

$$\langle \Sigma \rightarrow \mathbb{O}, \preceq \rangle$$

$\alpha_m$

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

approximation order

$$\bar{\alpha}_M(T) \stackrel{\text{def}}{=} \bar{\alpha}_V(\vec{\alpha}(T))$$

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

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

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

# Definite Termination Semantics

$$\mathcal{R}_M \stackrel{\text{def}}{=} \bar{\alpha}_M(\mathcal{T}_M) = \text{lfp}^{\preceq} \bar{F}_M$$

computational order

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

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

Sufficient precondition state semantics

## Sufficient preconditions

$\mathcal{S}(\mathcal{Y})$ : states with executions **staying in  $\mathcal{Y}$**

$$\mathcal{S}(\mathcal{Y}) \stackrel{\text{def}}{=} \{ \sigma \mid \forall n \geq 0, \sigma_0, \dots, \sigma_n: (\sigma = \sigma_0 \wedge \forall i: \sigma_i \rightarrow \sigma_{i+1}) \implies \sigma_n \in \mathcal{Y} \}$$

$$= \bigcap_{n \geq 0} \tilde{\text{pre}}_\tau^n(\mathcal{Y})$$

where  $\tilde{\text{pre}}_\tau(S) \stackrel{\text{def}}{=} \{ \sigma \mid \forall \sigma': \sigma \rightarrow \sigma' \implies \sigma' \in S \}$   
 (states such that **all** successors satisfy  $S$ ,  $\tilde{\text{pre}}$  is a complete  $\cap$ -morphism)

$\mathcal{S}(\mathcal{Y})$  can be expressed in **fixpoint form**:

$$\mathcal{S}(\mathcal{Y}) = \text{gfp } F_S \text{ where } F_S(S) \stackrel{\text{def}}{=} \mathcal{Y} \cap \tilde{\text{pre}}_\tau(S)$$

proof sketch: similar to that of  $\mathcal{R}(\mathcal{I})$ , in the dual.

$F_S$  is continuous in the dual CPO  $(\mathcal{P}(\Sigma), \supseteq)$ , because  $\tilde{\text{pre}}_\tau$  is:  $F_S(\bigcap_{i \in I} A_i) = \bigcap_{i \in I} F_S(A_i)$ .  
 By Kleene's theorem in the dual,  $\text{gfp } F_S = \bigcap_{n \in \mathbb{N}} F_S^n(\Sigma)$ .  
 We would prove by recurrence that  $F_S^n(\Sigma) = \bigcap_{i < n} \tilde{\text{pre}}_\tau^i(\mathcal{Y})$ .

Course 2      Program Semantics and Properties      Antoine Miné      p. 64 / 98

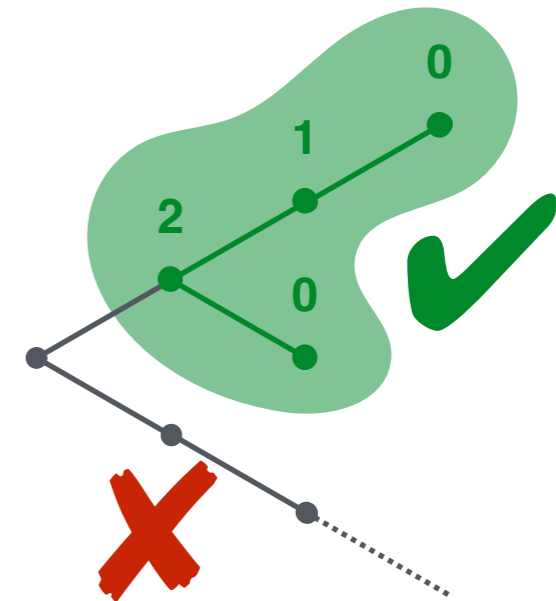
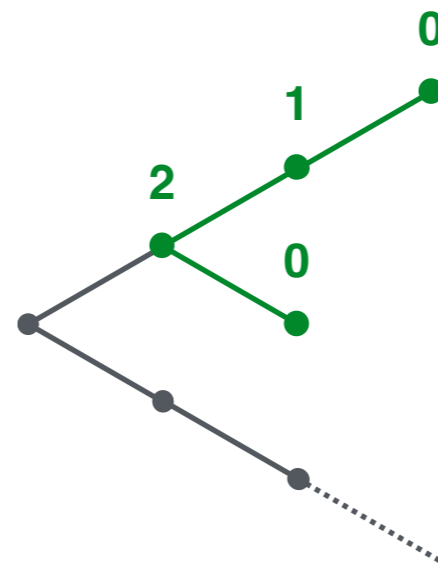
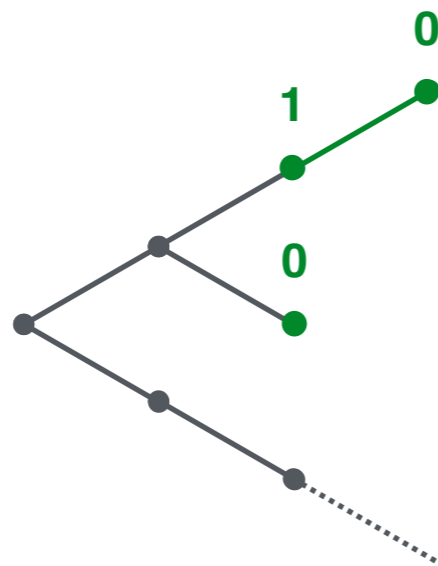
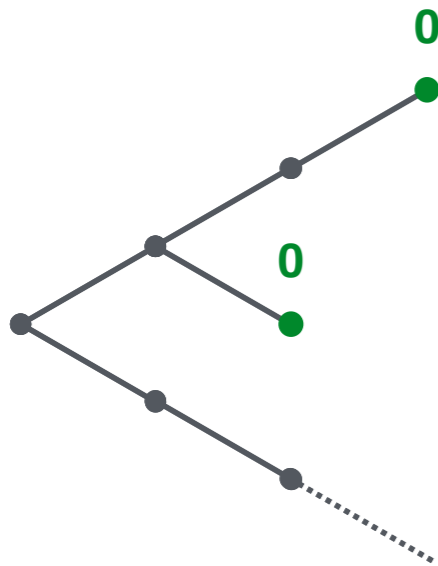
# Definite Termination Semantics

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

$$\mathcal{R}_M \stackrel{\text{def}}{=} \bar{\alpha}_M(\mathcal{T}_M) = \text{lfp}^{\preceq} \bar{F}_M$$

computational order

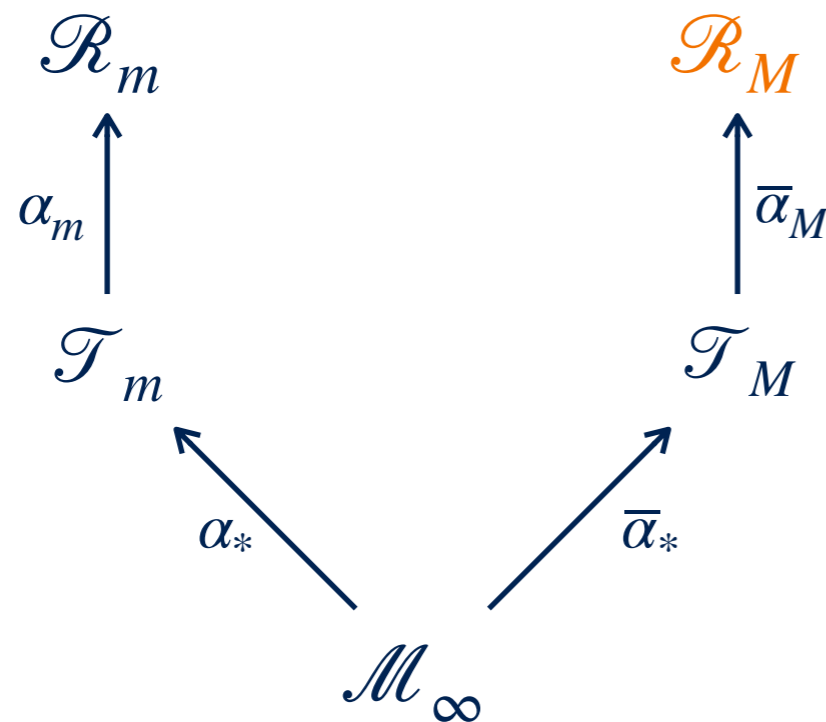
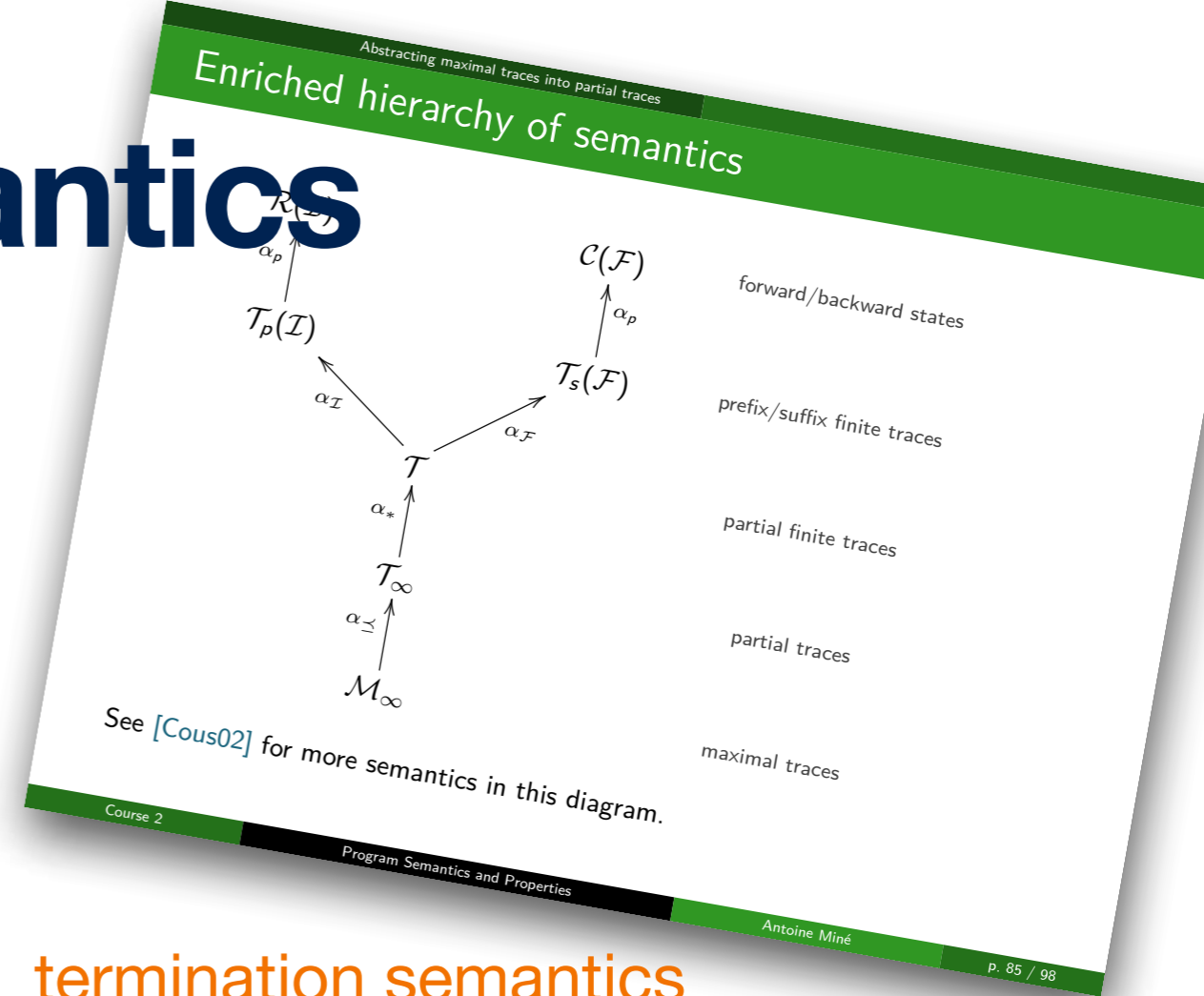
$$\bar{F}_M(f)\sigma \stackrel{\text{def}}{=} \begin{cases} 0 & \sigma \in \mathcal{B} \\ \sup\{f(\sigma') + 1 \mid (\sigma, \sigma') \in \tau\} & \sigma \in \tilde{\text{pre}}_\tau(\text{dom}(f)) \\ \text{undefined} & \text{otherwise} \end{cases}$$



## 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

$\mathcal{R}_M: \Sigma \rightarrow \mathbb{O}$  by partitioning with respect to the program control points, i.e.,

$\mathcal{R}_M: \mathcal{L} \rightarrow (\mathcal{E} \rightarrow \mathbb{O})$ .

Thus, for each program instruction  $\text{stat}$ , we define a transformer

$\mathcal{R}_M[\text{stat}]: (\mathcal{E} \rightarrow \mathbb{O}) \rightarrow (\mathcal{E} \rightarrow \mathbb{O})$ :

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

Language syntax

Programs and executions

${}^l\text{stat}^l$	$::=$	${}^lX \leftarrow \text{exp}^l$	(assignment)
		$\text{if } \text{exp} \bowtie 0 \text{ then } {}^l\text{stat}^l$	(conditional)
		$\text{while } \text{exp} \bowtie 0 \text{ do } {}^l\text{stat}^l \text{ done}$	(loop)
		${}^l\text{stat}^l; {}^l\text{stat}^l$	(sequence)
$\text{exp}$	$::=$	$X$	(variable)
		$-\text{exp}$	(negation)
		$\text{exp} \diamond \text{exp}$	(binary operation)
		$c$	(constant $c \in \mathbb{Z}$ )
		$[c, c']$	(random input, $c, c' \in \mathbb{Z} \cup \{\pm\infty\}$ )

Simple structured, numeric language

- $X \in \mathbb{V}$ , where  $\mathbb{V}$  is a finite set of **program variables**
- $l \in \mathcal{L}$ , where  $\mathcal{L}$  is a finite set of **control points**
- numeric expressions:  $\bowtie \in \{=, \leq, \dots\}$ ,  $\diamond \in \{+, -, \times, /\}$
- **random inputs**:  $X \leftarrow [c, c']$   
model environment, parametric programs, unknown functions, ...

Course 2

Program Semantics and Properties

Antoine Miné

p. 3 / 98

# Denotational Definite Termination Semantics

$$\mathcal{R}_M \llbracket \ell X \leftarrow e \rrbracket$$

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

Example:

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

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

We have

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

# Denotational Definite Termination Semantics

$\mathcal{R}_M[\text{if } e \neq 0 \text{ then } s]$

$\mathcal{R}_M[\text{if } e \neq 0 \text{ then } s]f \stackrel{\text{def}}{=} \lambda\rho. \begin{cases} \textcircled{1} \\ \textcircled{2} \\ \textcircled{3} \\ \text{undefined} & \text{otherwise} \end{cases}$

$\textcircled{1}$   $\sup\{\mathcal{R}_M[s]f(\rho) + 1, f(\rho) + 1\}$   $\rho \in \text{dom}(\mathcal{R}_M[s]f) \cap \text{dom}(f) \wedge \exists v_1, v_2 \in E[e]\rho: v_1 \neq 0 \wedge v_2 \neq 0$

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

$\textcircled{3}$   $f(\rho) + 1$   $\rho \in \text{dom}(f) \wedge \forall v \in E[e]\rho: v \neq 0$



# Denotational Definite Termination Semantics

$\mathcal{R}_M \llbracket \text{if } e \bowtie 0 \text{ then } s \rrbracket$  (continue)

Example:

Let  $\mathbb{V} = \{x\}$  and  $f: \mathcal{E} \rightarrow \mathbb{O}$ , and  $\mathcal{R}_M \llbracket s \rrbracket f$  defined as follows:

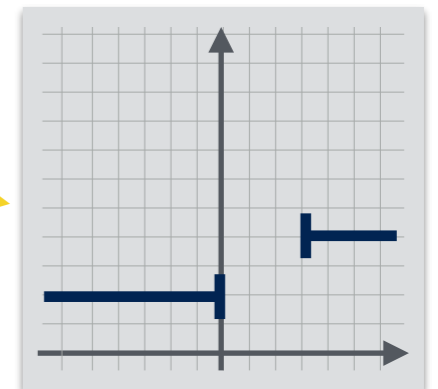
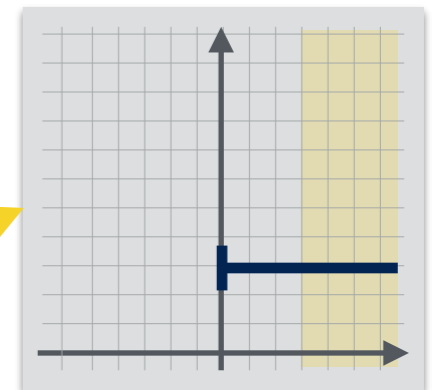
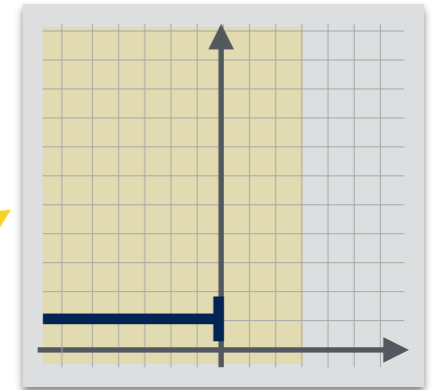
$$f \stackrel{\text{def}}{=} \lambda \rho . \begin{cases} 1 & \rho(x) \leq 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\mathcal{R}_M \llbracket s \rrbracket f \stackrel{\text{def}}{=} \lambda \rho . \begin{cases} 3 & 0 \leq \rho(x) \\ \text{undefined} & \text{otherwise} \end{cases}$$

We have

$$\mathcal{R}_M \llbracket \text{if } 3 - x < 0 \text{ then } s \rrbracket f \stackrel{\text{def}}{=} \lambda \rho . \begin{cases} 2 & \rho(x) \leq 0 \\ 4 & 3 < \rho(x) \\ \text{undefined} & \text{otherwise} \end{cases}$$

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



# Denotational Definite Termination Semantics

$$\mathcal{R}_M[\text{while } e \not\approx 0 \text{ do } s \text{ done}]$$

$$\mathcal{R}_M[\text{while } e \not\approx 0 \text{ do } s \text{ done}]f \stackrel{\text{def}}{=} \text{lfp}_{\emptyset}^{\leq} \bar{F}_M$$

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

computational order

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

- $\textcircled{1}$   $\sup\{\mathcal{R}_M[[s]]x(\rho) + 1, f(\rho) + 1\}$   $\rho \in \text{dom}(\mathcal{R}_M[[s]]x) \cap \text{dom}(f) \wedge \exists v_1, v_2 \in E[[e]]\rho: v_1 \not\approx 0 \wedge v_2 \not\approx 0$
- $\textcircled{2}$   $\mathcal{R}_M[[s]]x(\rho) + 1$   $\rho \in \text{dom}(\mathcal{R}_M[[s]]x) \wedge \forall v \in E[[e]]\rho: v \not\approx 0$
- $\textcircled{3}$   $f(\rho) + 1$   $\rho \in \text{dom}(f) \wedge \forall v \in E[[e]]\rho: v \not\approx 0$

# 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**  $\mathcal{R}_M[[\text{stat}^\ell]] : \mathcal{E} \rightarrow \mathbb{O}$  of a program  $\text{stat}^\ell$  is:

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

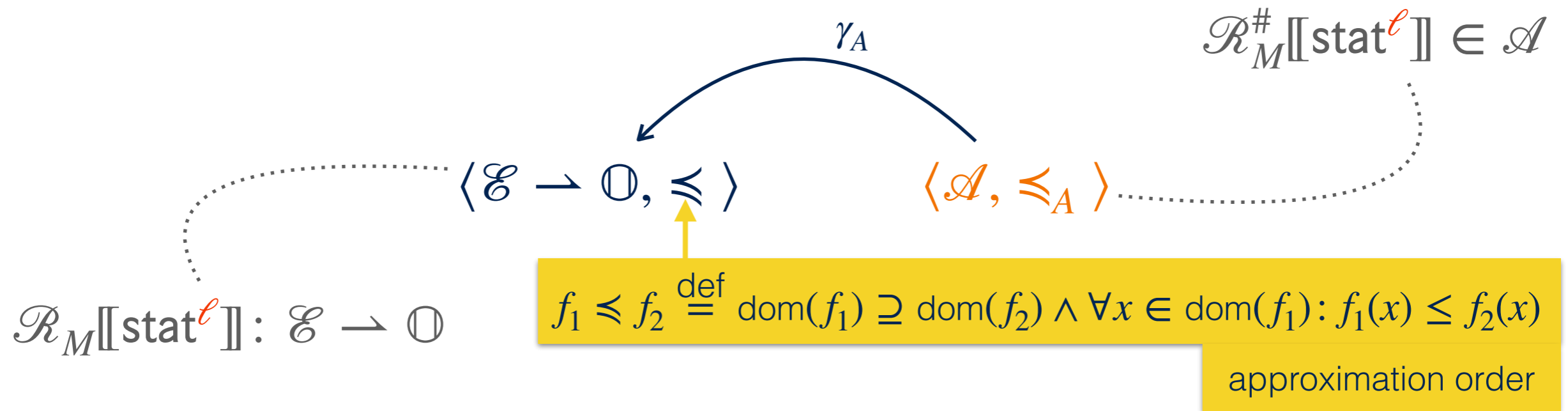
where  $\mathcal{R}_M[[\text{stat}]] : (\mathcal{E} \rightarrow \mathbb{O}) \rightarrow (\mathcal{E} \rightarrow \mathbb{O})$  is the definite termination semantics of each program instruction  $\text{stat}$

## Theorem

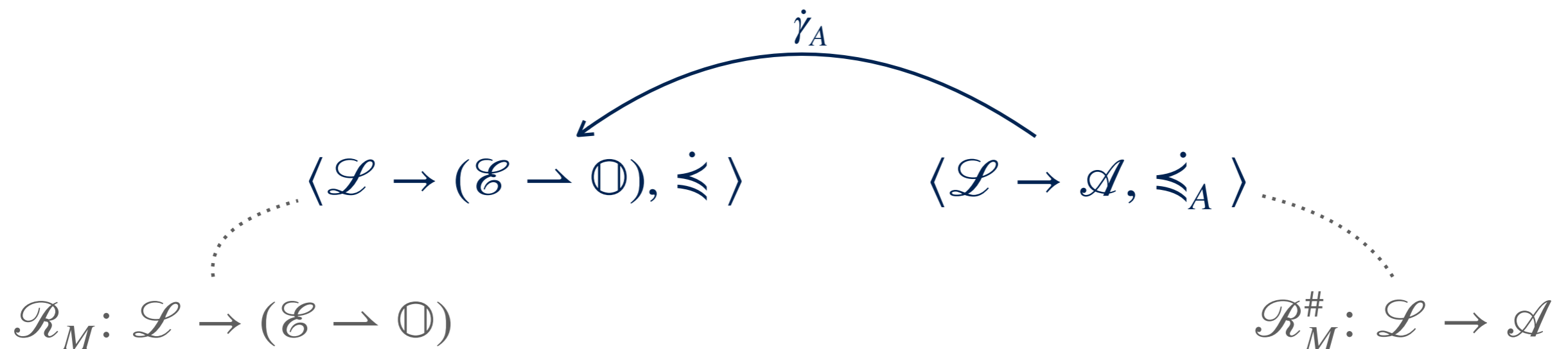
A program  $\text{stat}^\ell$  **must terminate** for traces starting from a set of initial states  $\mathcal{I}$  if  $\mathcal{I} \subseteq \text{dom}(\mathcal{R}_m[[\text{stat}^\ell]])$

# Piecewise-Defined Ranking Functions Abstract Domain

# Concretization-Based Piecewise Abstraction



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



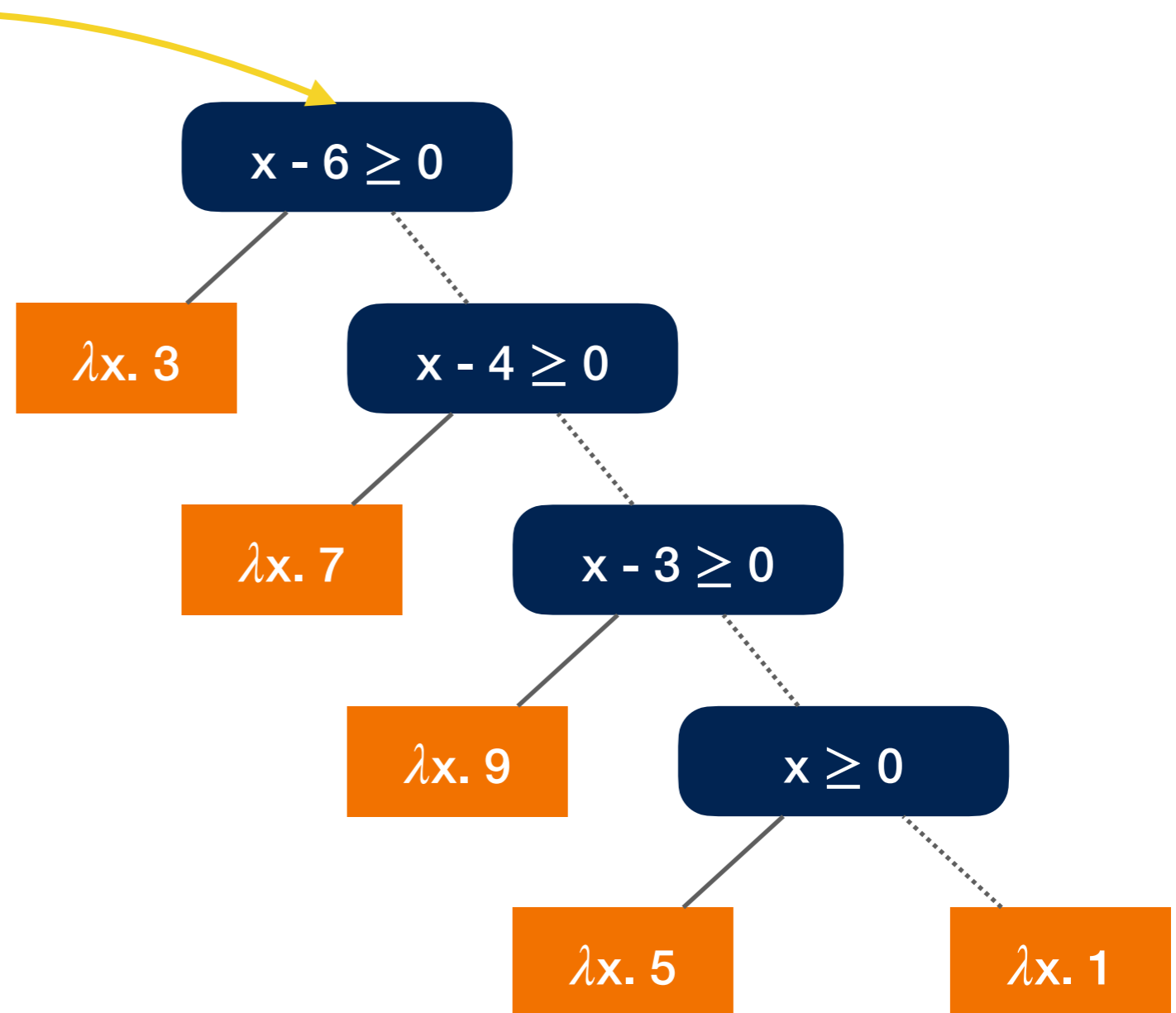
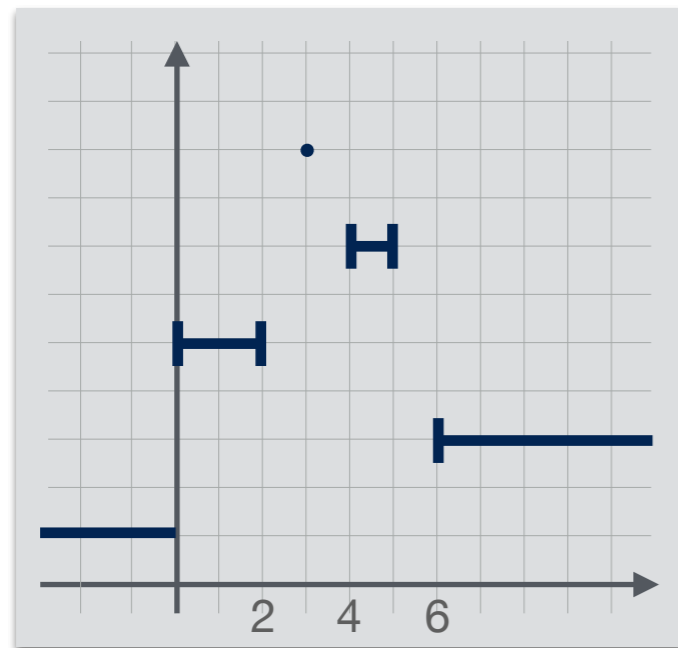
# Piecewise-Defined Ranking Functions Abstract Domain

$\langle \mathcal{A}, \preceq_A \rangle$

## Example

```

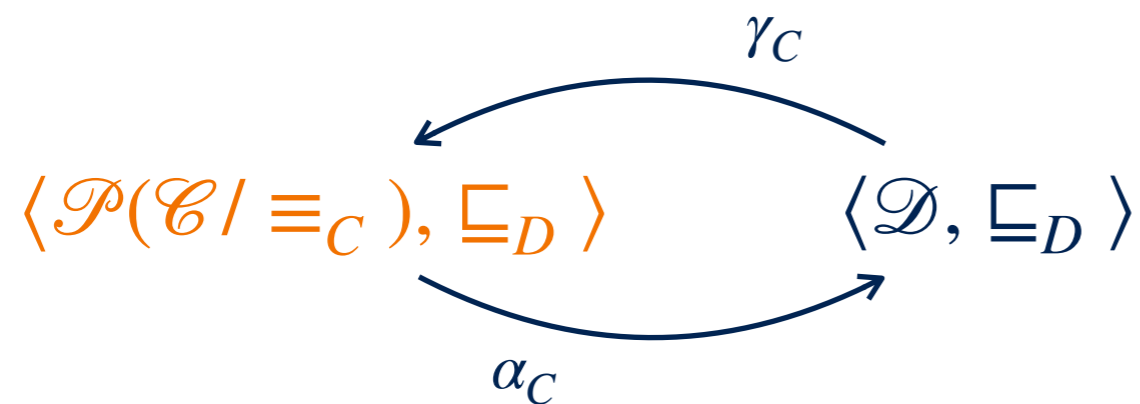
1 $x \leftarrow [-\infty, +\infty]$ 
while 2 $(x \geq 0)$  do
  3 $x \leftarrow -2 \cdot x + 10$ 
od4
  
```



# Piecewise-Defined Ranking Functions Abstract Domain

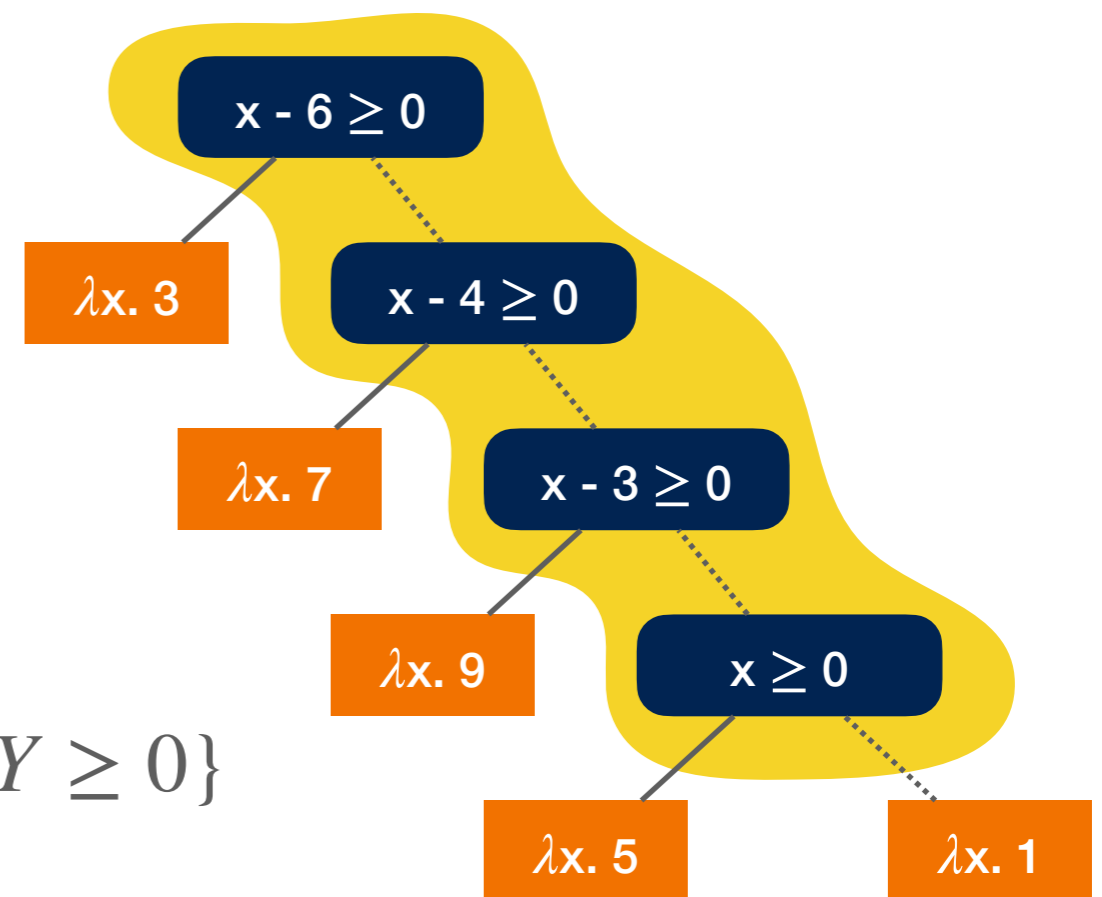
## Linear Constraints Auxiliary Abstract Domain

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



Example:

$$\begin{array}{l}
 X \rightarrow [-\infty, 3] \\
 Y \rightarrow [0, +\infty]
 \end{array}
 \xrightarrow{\gamma_C}
 \{3 - X \geq 0, Y \geq 0\}$$

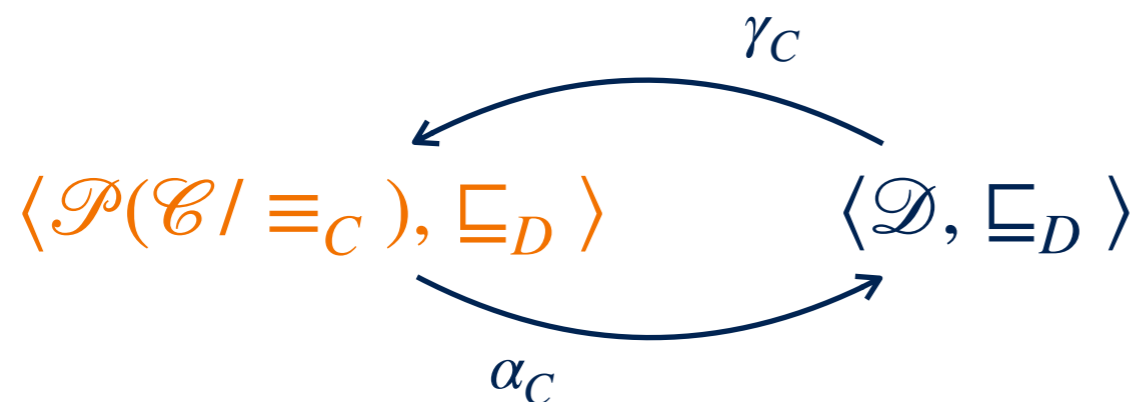




# Piecewise-Defined Ranking Functions Abstract Domain

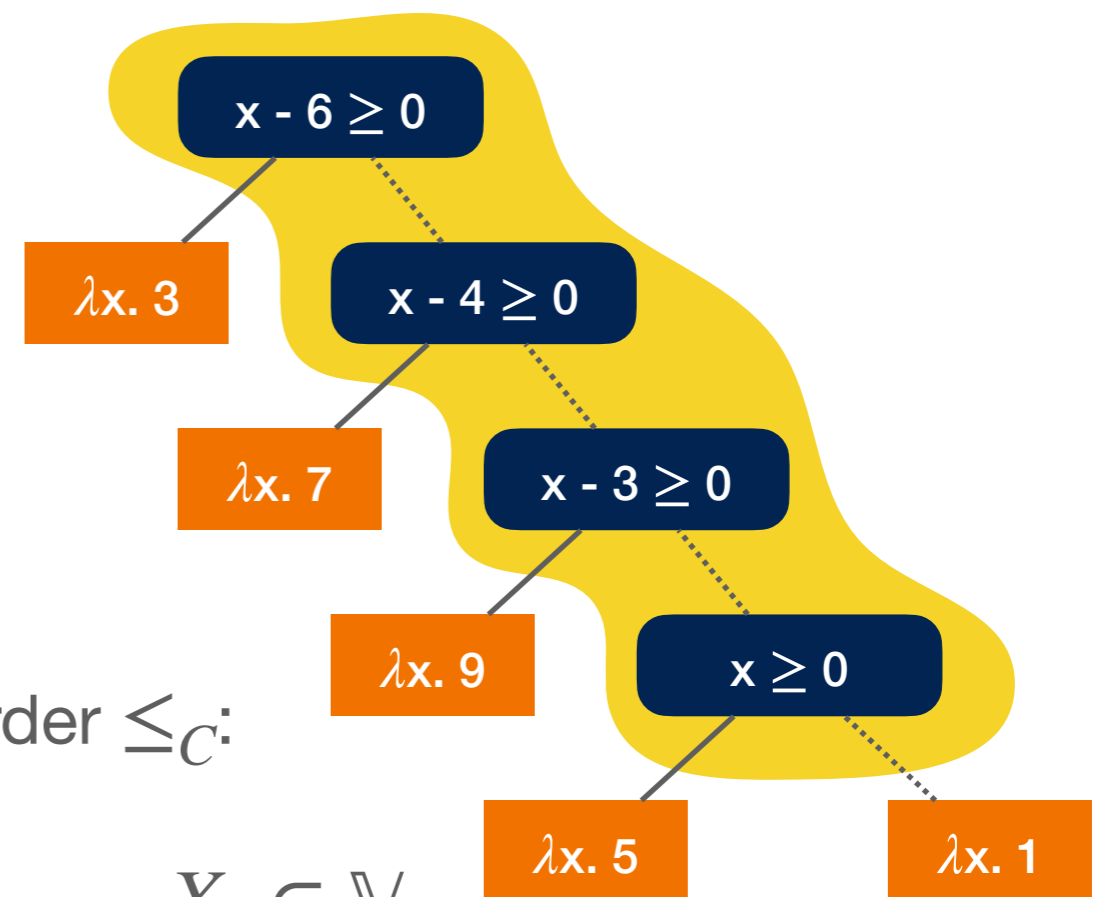
## Linear Constraints Auxiliary Abstract Domain

- 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  $\leq_C$ :

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



# Piecewise-Defined Ranking Functions Abstract Domain

## Functions Auxiliary Abstract Domain

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

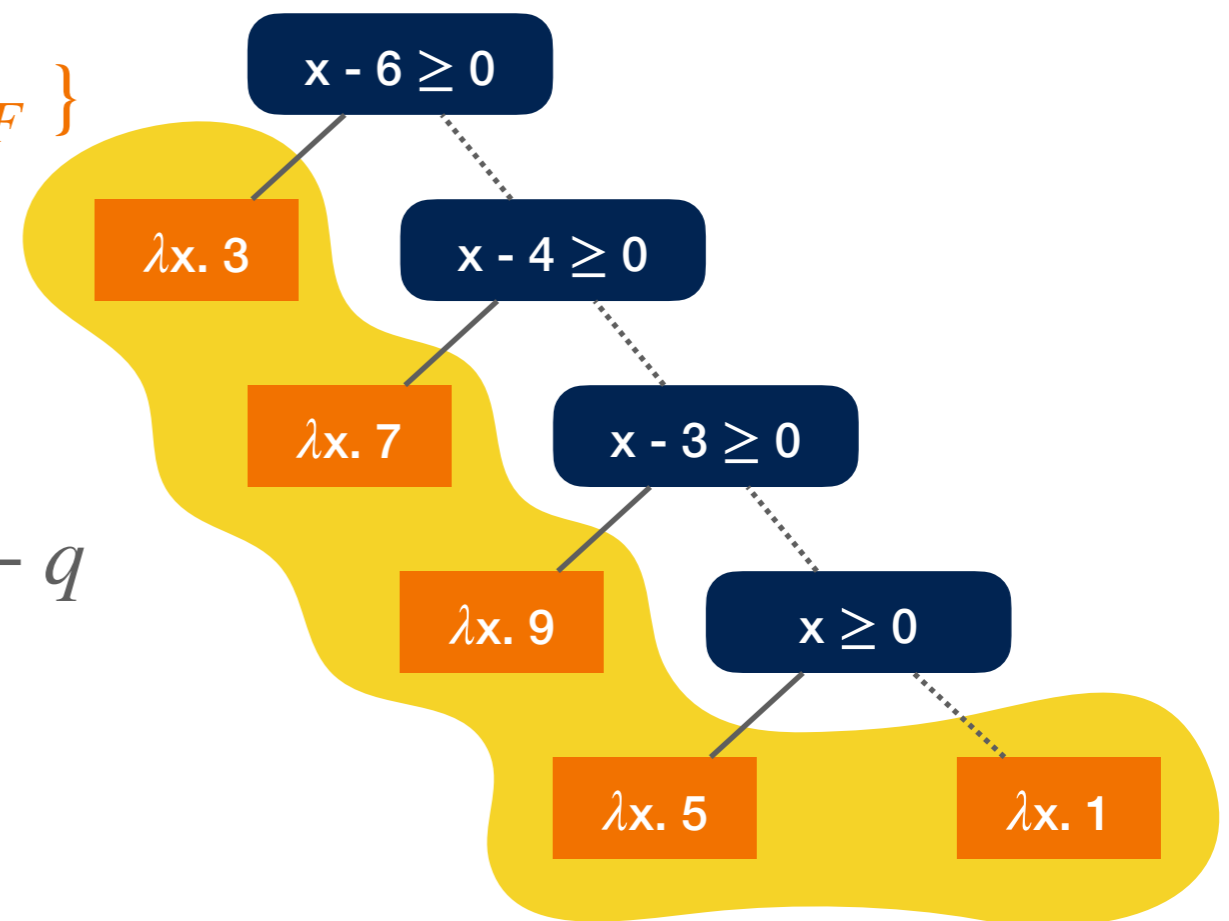
- $\mathcal{F} \stackrel{\text{def}}{=} \{ \perp_F \} \cup (\mathbb{Z}^{|M|} \rightarrow \mathbb{N}) \cup \{ \top_F \}$

We consider **affine functions**:

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

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

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



# Piecewise-Defined Ranking Functions Abstract Domain

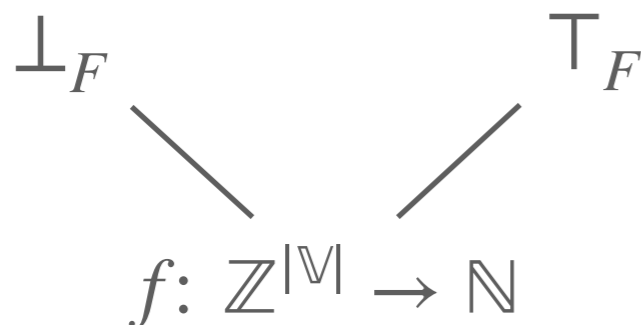
## Functions Auxiliary Abstract Domain (continue)

- **approximation order**  $\preceq_F [D]$ , where  $D \in \mathcal{D}$ :

- between defined leaf nodes:

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

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



# Piecewise-Defined Ranking Functions Abstract Domain

## Functions Auxiliary Abstract Domain (continue)

- **computational order**  $\sqsubseteq_F[D]$ , where  $D \in \mathcal{D}$ :

- between defined leaf nodes:

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

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

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

# Piecewise-Defined Ranking Functions Abstract Domain

- $\mathcal{A} \stackrel{\text{def}}{=} \{\text{LEAF}: f \mid f \in \mathcal{F}\} \cup \{\text{NODE}\{c\}: t_1; t_2 \mid c \in \mathcal{C} \wedge t_1, t_2 \in \mathcal{A}\}$
- **concretization function**  $\gamma_A: \mathcal{A} \rightarrow (\mathcal{E} \rightarrow \mathbb{O})$ :

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

where  $\bar{\gamma}_A: \mathcal{P}(\mathcal{C} / \equiv_C) \rightarrow \mathcal{A} \rightarrow (\mathcal{E} \rightarrow \mathbb{O})$ :

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

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

and  $\gamma_F: \mathcal{D} \rightarrow \mathcal{F} \rightarrow (\mathcal{E} \rightarrow \mathbb{O})$ :

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

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

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

# Piecewise-Defined Ranking Functions Abstract Domain

## Abstract Domain Operators

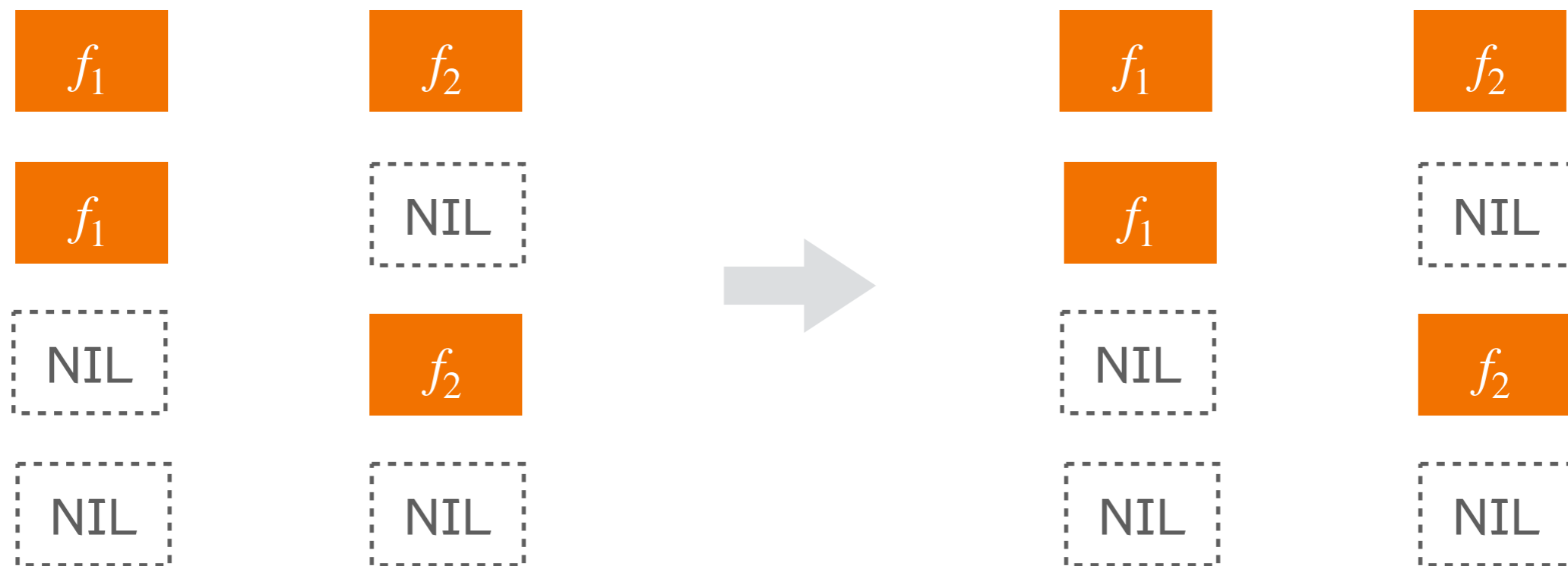
- They manipulate elements in  $\mathcal{A}_{NIL} \stackrel{\text{def}}{=} \{NIL\} \cup \mathcal{A}$
- The **binary operators** rely on a tree unification algorithm
  - approximation order  $\preceq_A$  and computational order  $\sqsubseteq_A$
  - approximation join  $\vee_A$  and computational join  $\sqcup_A$
  - meet  $\wedge_A$
  - widening  $\nabla_A$
- 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

## Tree Unification

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

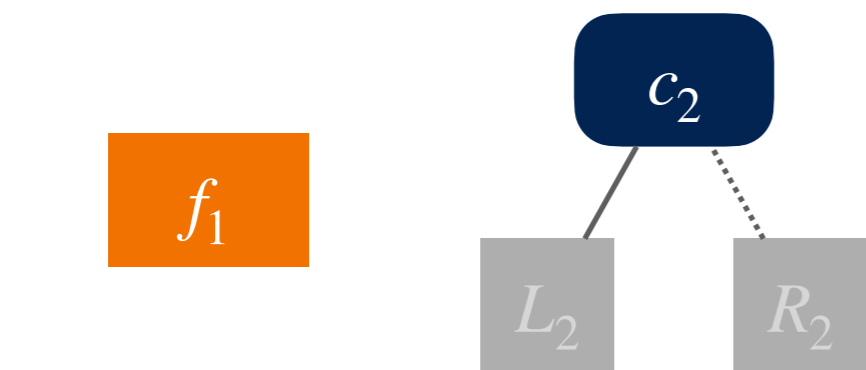
- Base cases:



# Piecewise-Defined Ranking Functions Abstract Domain

## Tree Unification (continue)

- Case ①



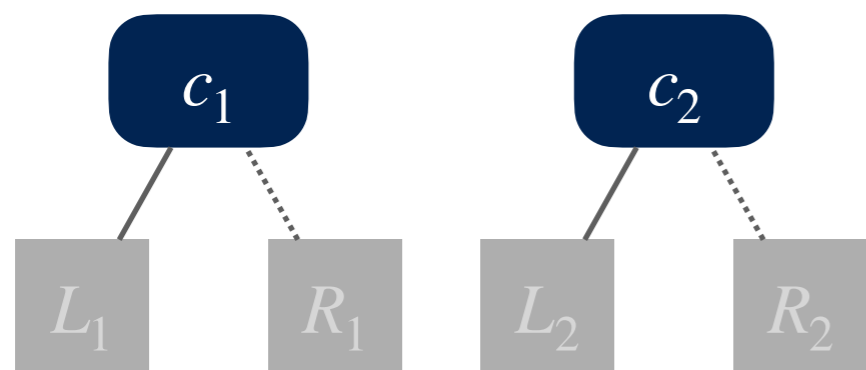
①a)  $c_2$  is redundant



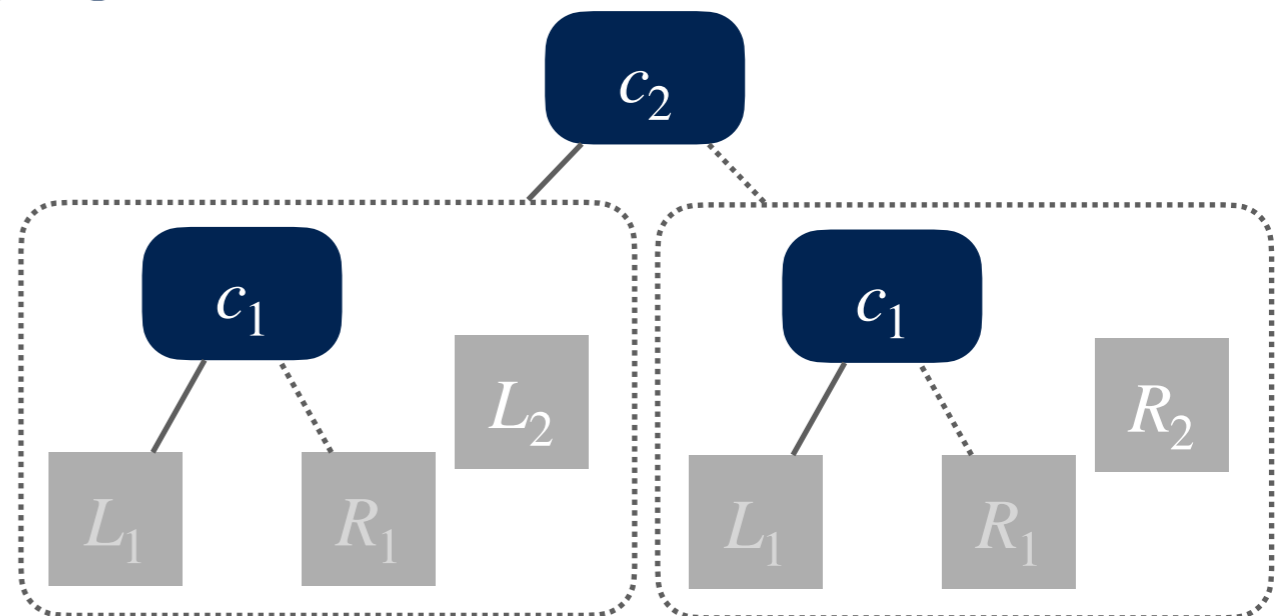
①b)  $\neg c_2$  is redundant



①c)  $c_2$  is added to  $t_1$



$$c_2 \leq_C c_1$$

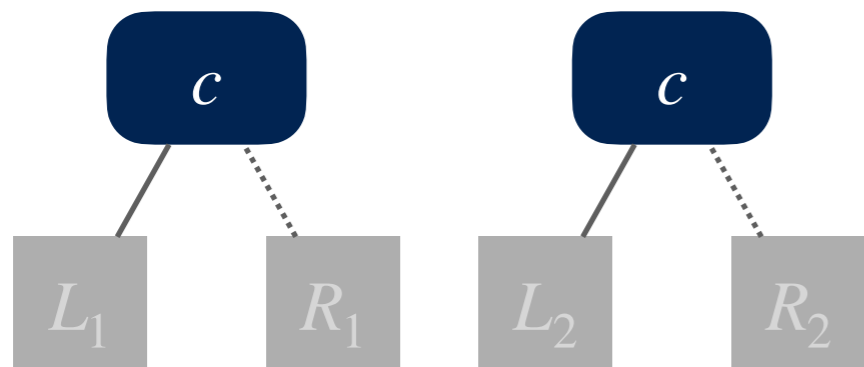




# Piecewise-Defined Ranking Functions Abstract Domain

## Tree Unification (continue)

- Case ② (symmetric to ①)
- Case ③



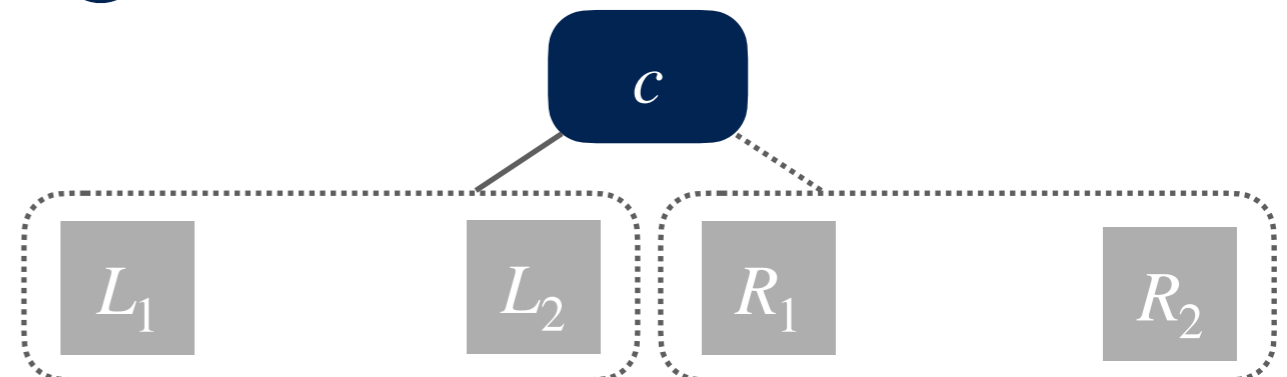
①a)  $c$  is redundant



①b)  $\neg c$  is redundant



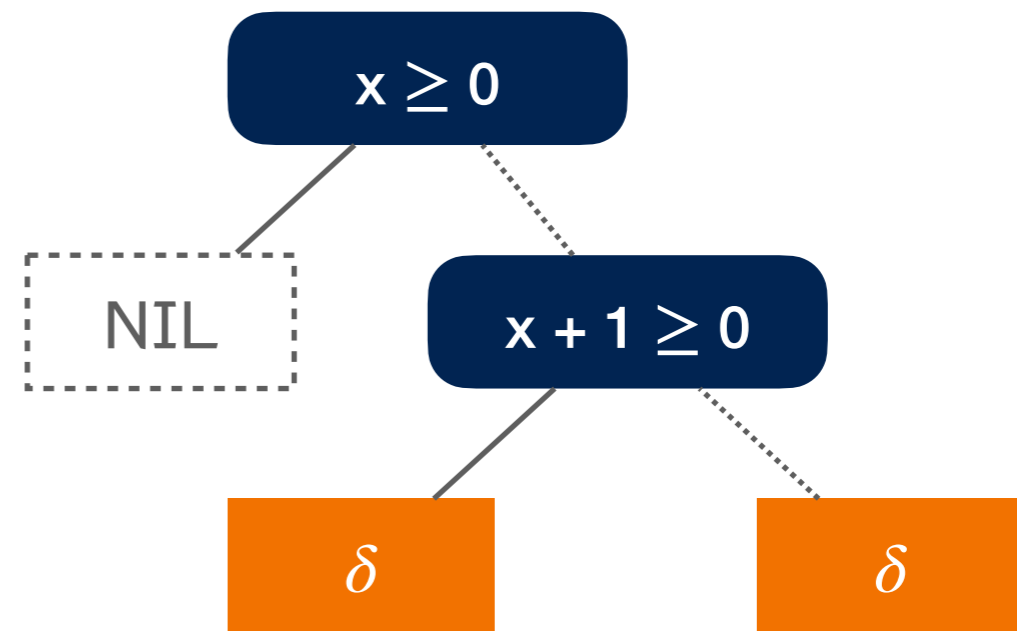
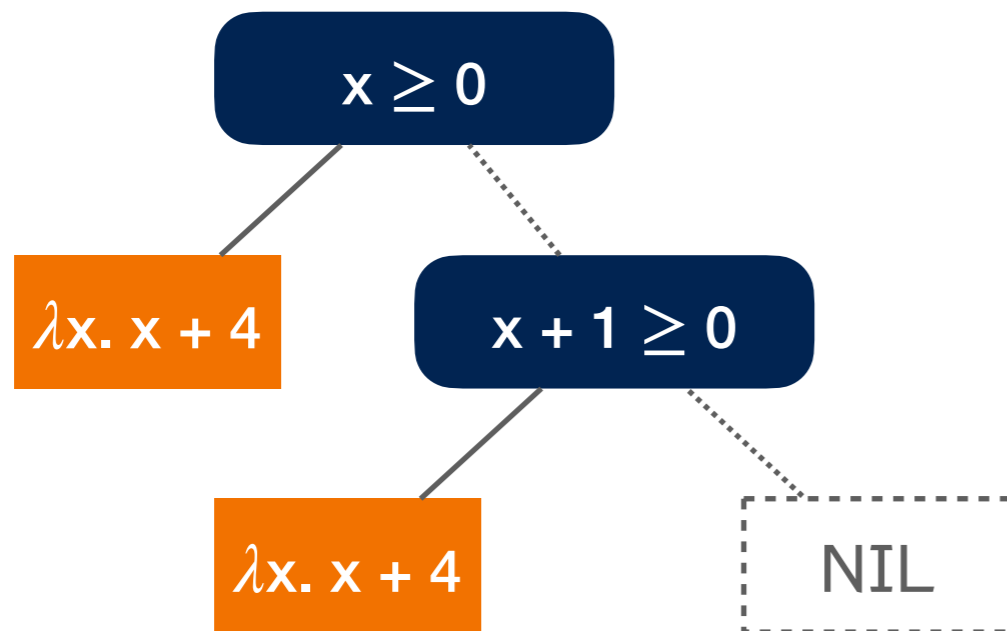
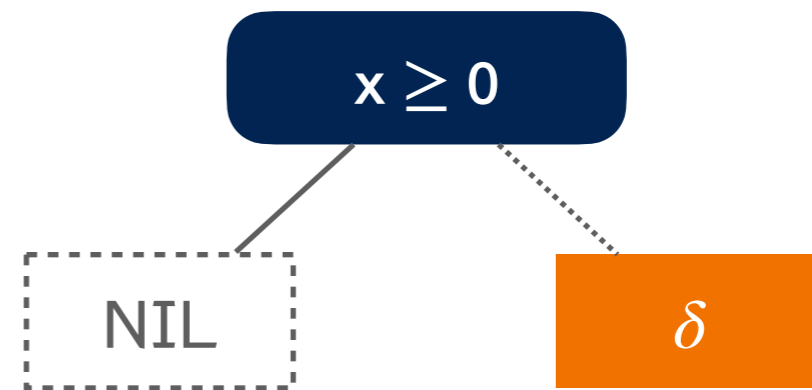
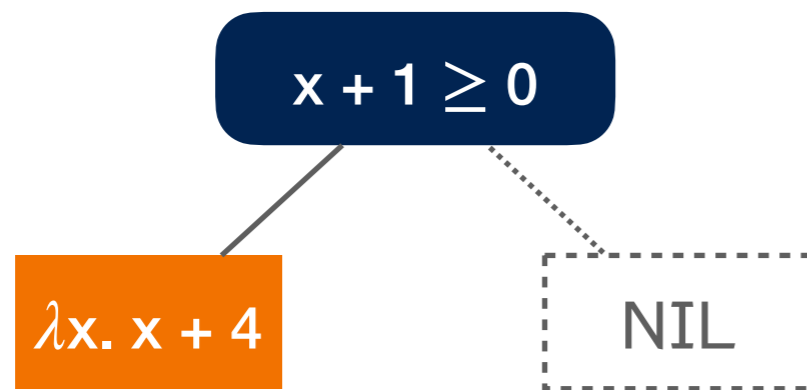
①c)  $c$  is kept in  $t_1$  and  $t_2$



# 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**  $\preceq_F[\alpha_C(C)]$  or the **computational order**  $\sqsubseteq_F[\alpha_C(C)]$

The concretization function  $\gamma_A$  is monotonic with respect to  $\preceq_A$ :

### Lemma

$$\forall t_1, t_2 \in \mathcal{A} : t_1 \preceq_A t_2 \Rightarrow \gamma_A(t_1) \preceq \gamma_A(t_2)$$

# 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. 
$$\text{NIL } \gamma_A t \stackrel{\text{def}}{=} t$$
$$t \gamma_A \text{NIL} \stackrel{\text{def}}{=} t$$
4. Join the leaf nodes using the **approximation join**  $\gamma_F [\alpha_C(C)]$  or the **computational join**  $\sqcup_F [\alpha_C(C)]$

# Piecewise-Defined Ranking Functions Abstract Domain

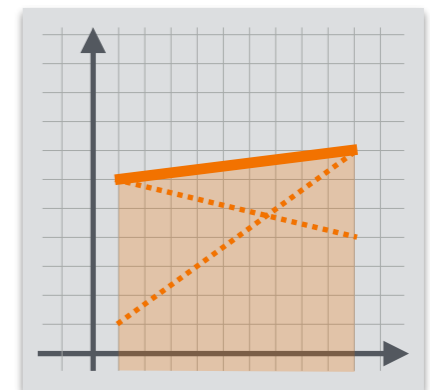
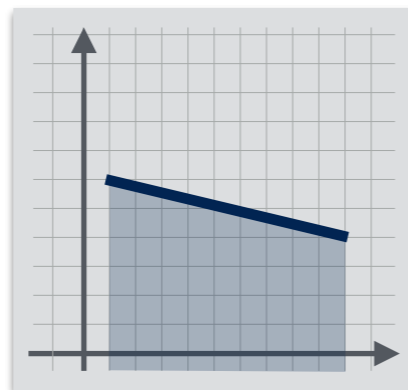
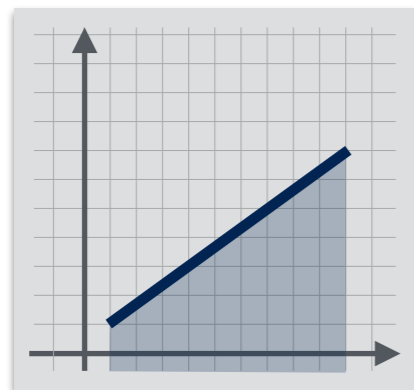
## Join (continue)

- **approximation join**  $\vee_F [D]$ , where  $D \in \mathcal{D}$ :
  - between defined leaf nodes:

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

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

Example:



# Piecewise-Defined Ranking Functions Abstract Domain

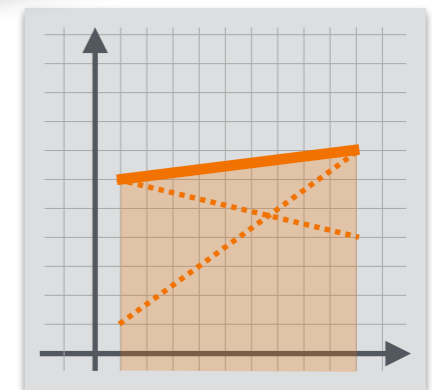
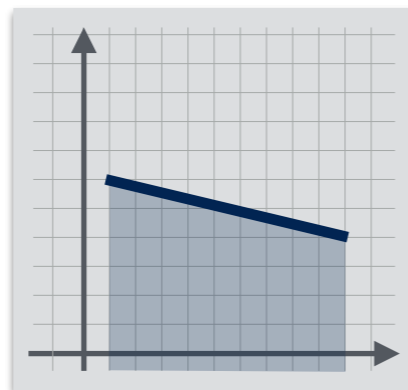
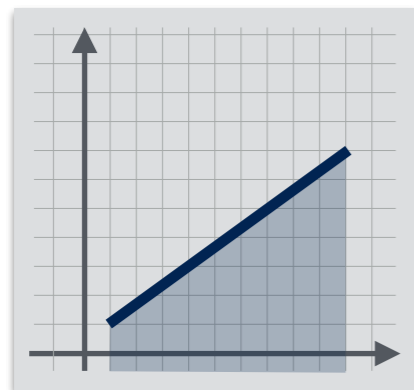
## Join (continue)

- approximation join  $\vee_F [D]$ , where  $D$
- between defined leaf nodes:

$$f_1 \vee_F [D] f_2 \stackrel{\text{def}}{=} \begin{cases} f & f \in \mathcal{F} \setminus \{ \dots \} \\ \top_F & \text{otherwise} \end{cases}$$

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

Example:



Polyhedron domain

### Operators on polyhedra: join

Join:  $\mathcal{X}^\# \cup^\# \mathcal{Y}^\# \stackrel{\text{def}}{=} [[P_{\mathcal{X}^\#} P_{\mathcal{Y}^\#}], [R_{\mathcal{X}^\#} R_{\mathcal{Y}^\#}]]$  (join generator sets)

Examples:

two polytopes

a point and a line

$\cup^\#$  is **optimal**:  
we get the **topological closure of the convex hull** of  $\gamma(\mathcal{X}^\#) \cup \gamma(\mathcal{Y}^\#)$ .

Antoine Miné

p. 30 / 70

Course 4    Relational Numerical Abstract Domains

# Piecewise-Defined Ranking Functions Abstract Domain

## Join (continue)

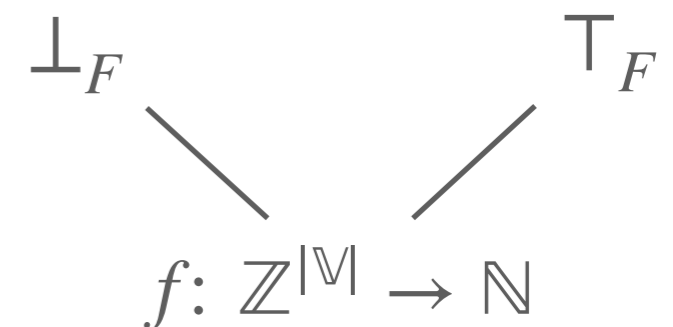
- **approximation join**  $\vee_F [D]$ , where  $D \in \mathcal{D}$ :
  - between defined leaf nodes:

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

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

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

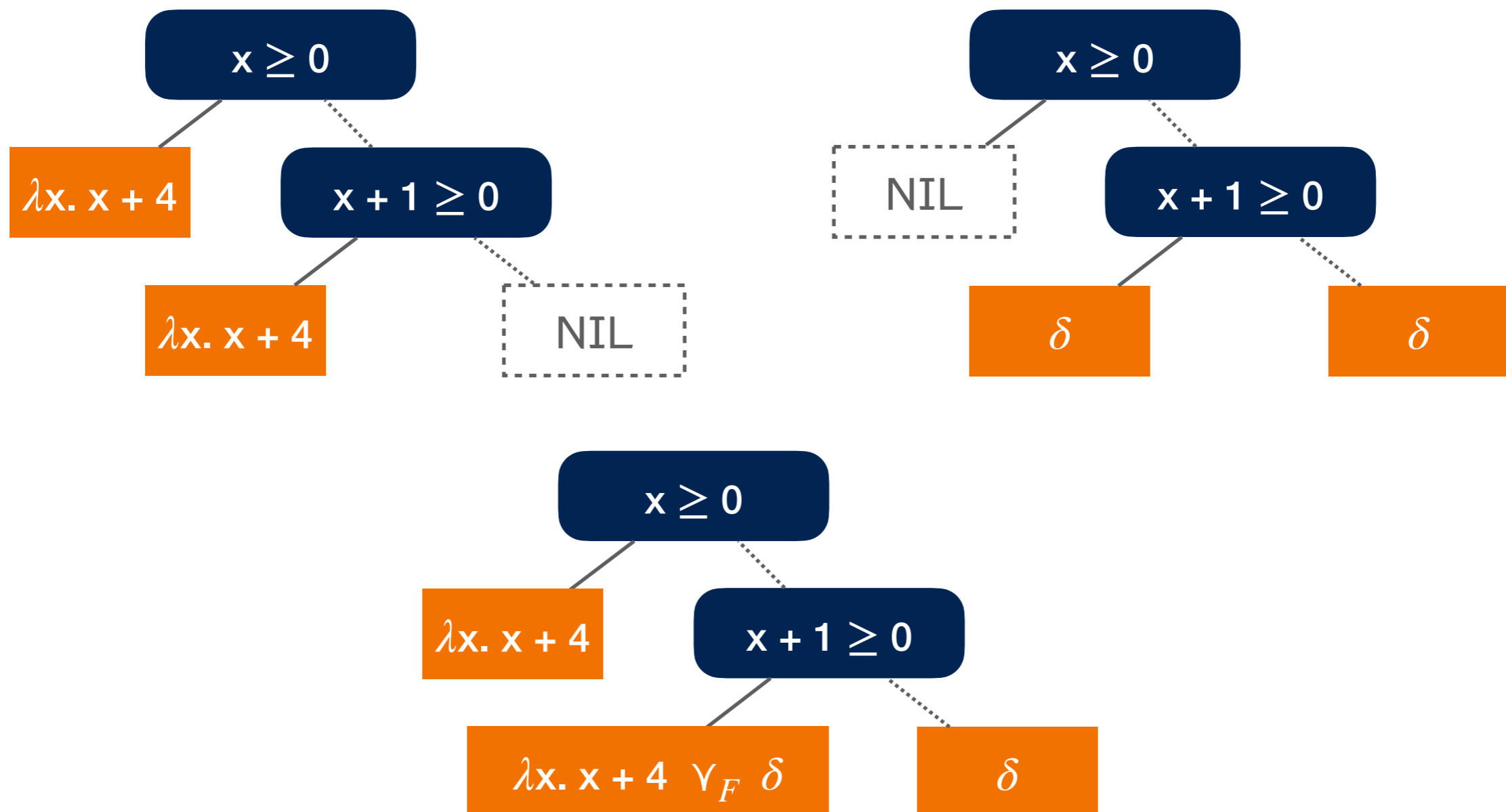
$$\begin{array}{ll} \perp_F \vee_F [D] f \stackrel{\text{def}}{=} \perp_F & f \in \mathcal{F} \setminus \{ \top_F \} \\ f \vee_F [D] \perp_F \stackrel{\text{def}}{=} \perp_F & f \in \mathcal{F} \setminus \{ \top_F \} \\ \top_F \vee_F [D] f \stackrel{\text{def}}{=} \top_F & f \in \mathcal{F} \setminus \{ \perp_F \} \\ f \vee_F [D] \top_F \stackrel{\text{def}}{=} \top_F & f \in \mathcal{F} \setminus \{ \perp_F \} \end{array}$$



# Piecewise-Defined Ranking Functions Abstract Domain

Join (continue)

Example





# Piecewise-Defined Ranking Functions Abstract Domain

## Join (continue)

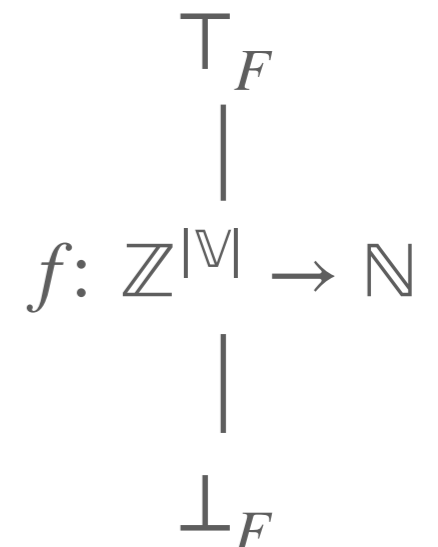
- **computational join**  $\sqcup_F [D]$ , where  $D \in \mathcal{D}$ :
  - between defined leaf nodes:

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

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

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

$$\begin{aligned} \perp_F \sqcup_F [D] f &\stackrel{\text{def}}{=} f & f \in \mathcal{F} \\ f \sqcup_F [D] \perp_F &\stackrel{\text{def}}{=} f & f \in \mathcal{F} \\ \top_F \sqcup_F [D] f &\stackrel{\text{def}}{=} \top_F & f \in \mathcal{F} \\ f \sqcup_F [D] \top_F &\stackrel{\text{def}}{=} \top_F & f \in \mathcal{F} \end{aligned}$$



# Piecewise-Defined Ranking Functions Abstract Domain

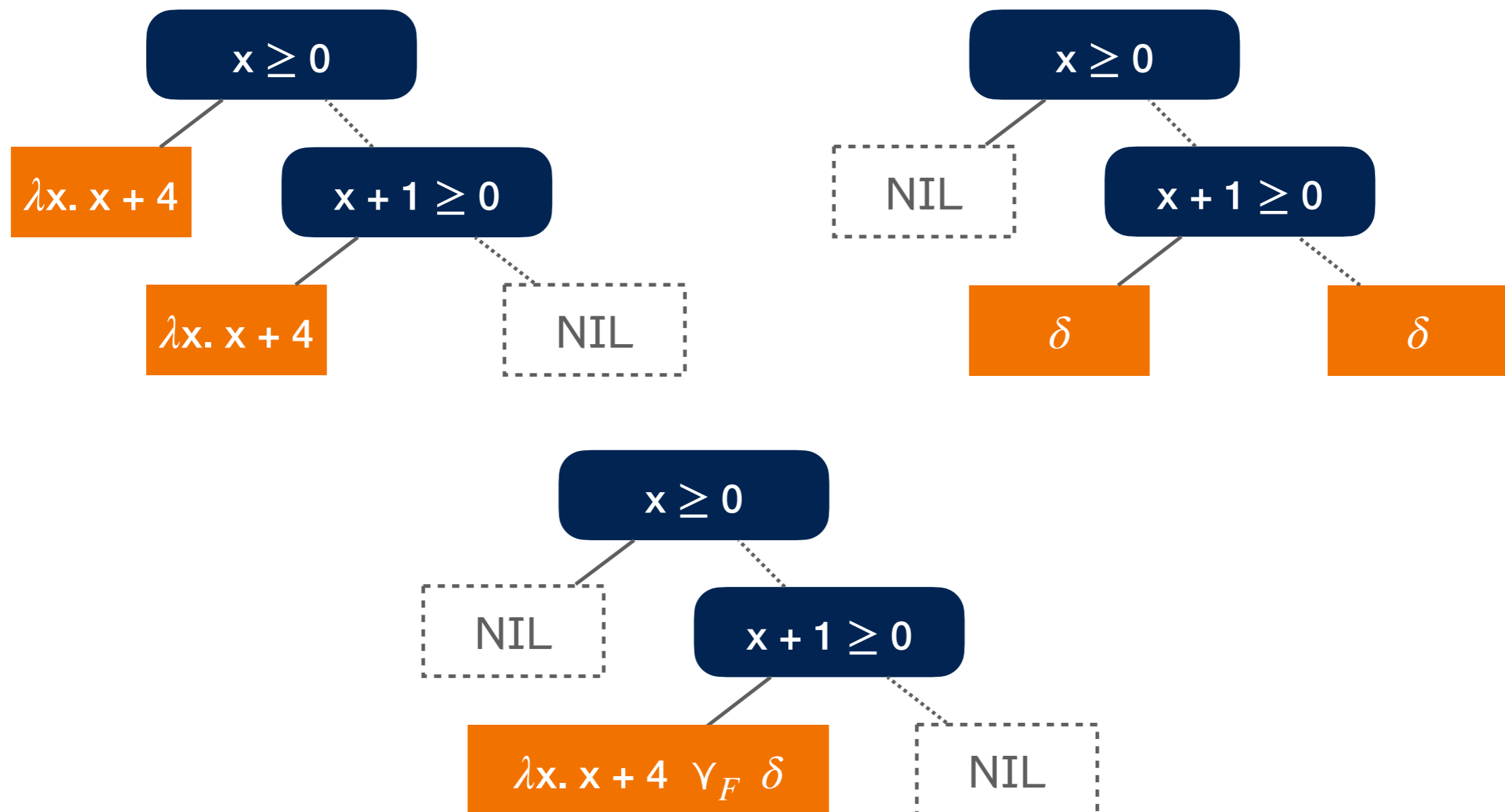
## Meet

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. 
$$\begin{array}{l} \text{NIL } \gamma_A t \stackrel{\text{def}}{=} \text{NIL} \\ t \gamma_A \text{NIL} \stackrel{\text{def}}{=} \text{NIL} \end{array}$$
4. Join the leaf nodes using the **approximation join**  $\gamma_F [\alpha_C(C)]$

# Piecewise-Defined Ranking Functions Abstract Domain

Meet (continue)

Example



# Piecewise-Defined Ranking Functions Abstract Domain

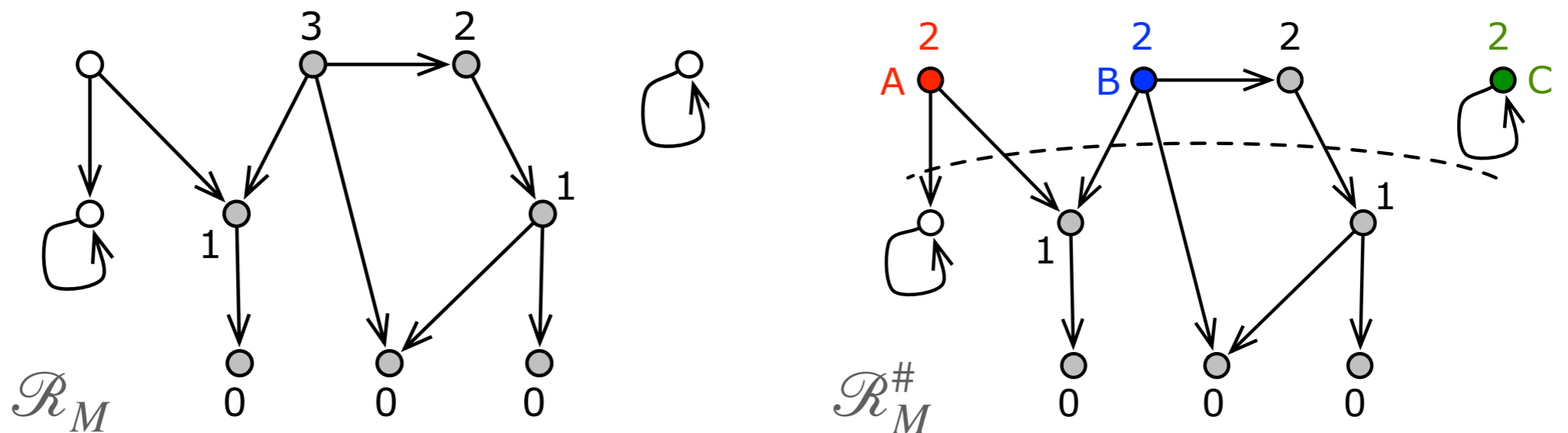
## Widening

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

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

- *under-approximates* the value of  $\mathcal{R}_M$
- and/or
- *over-approximates* the domain  $\text{dom}(\mathcal{R}_M)$  of  $\mathcal{R}_M$

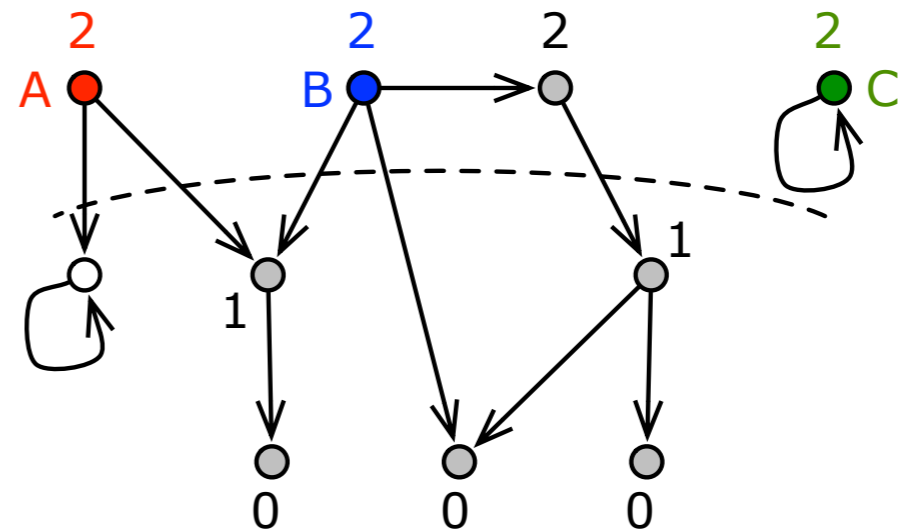
Example



# 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

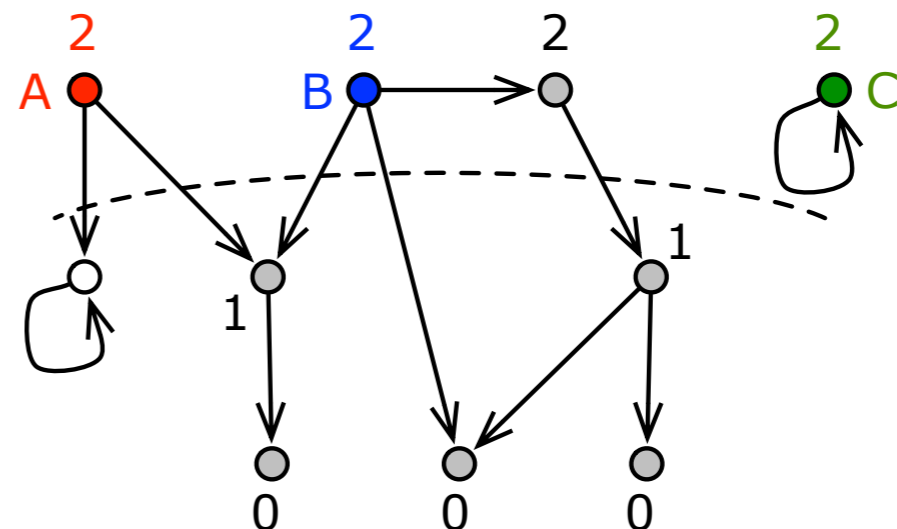
Widening (continue)

Check for Case A

## Lemma

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

(see proof in [Urban15])



# Piecewise-Defined Ranking Functions Abstract Domain

Widening (continue)

Check for Case A

## Lemma

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

(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  $C$



# Piecewise-Defined Ranking Functions Abstract 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:

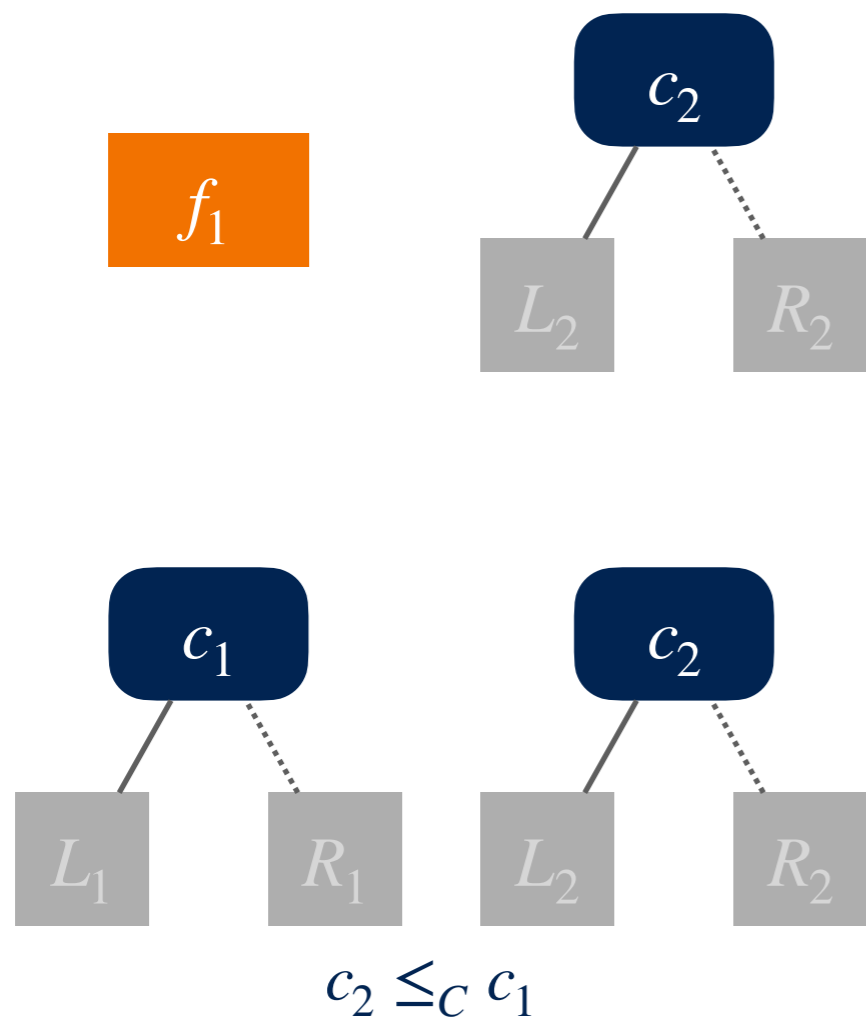




# Piecewise-Defined Ranking Functions Abstract Domain

## Widening (continue)

- Case ①



## Domain Widening

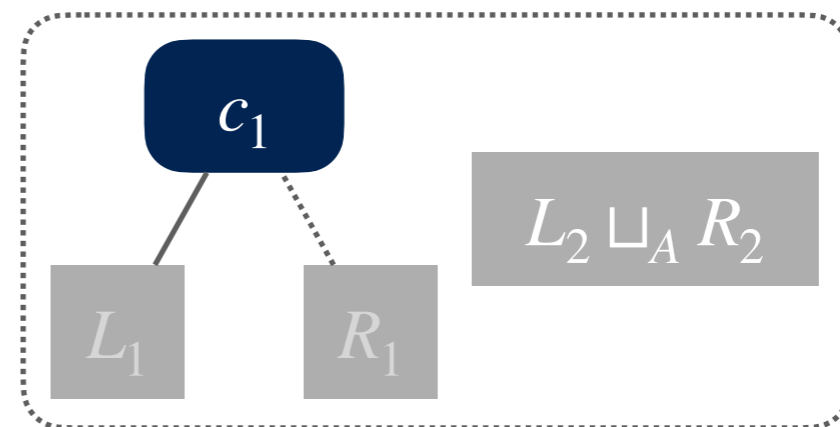
- ①a)  $c_2$  is redundant



- ①b)  $\neg c_2$  is redundant



- ①c)  $c_2$  is removed from  $t_2$

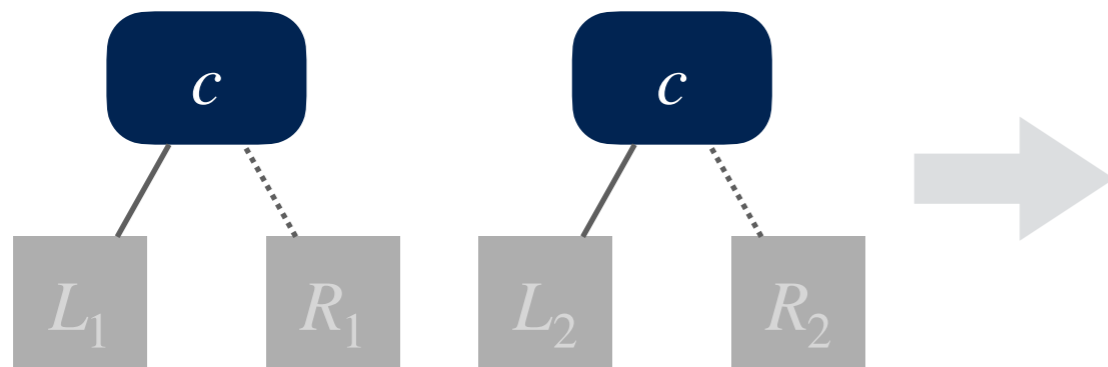


# Piecewise-Defined Ranking Functions Abstract Domain

## Widening (continue)

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

## Domain Widening



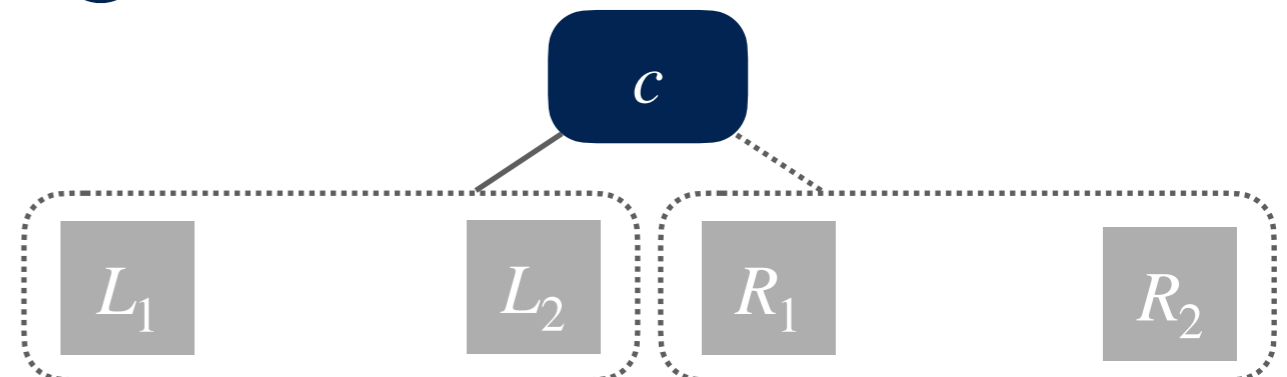
①a)  $c$  is redundant



①b)  $\neg c$  is redundant



①c)  $c$  is kept in  $t_1$  and  $t_2$



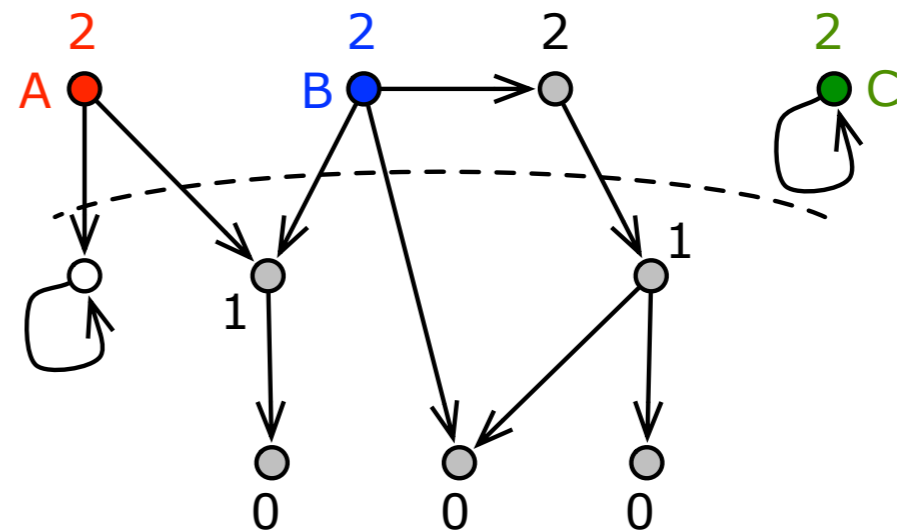
# Piecewise-Defined Ranking Functions Abstract Domain

Widening (continue)

Check for Case B or C

## Lemma

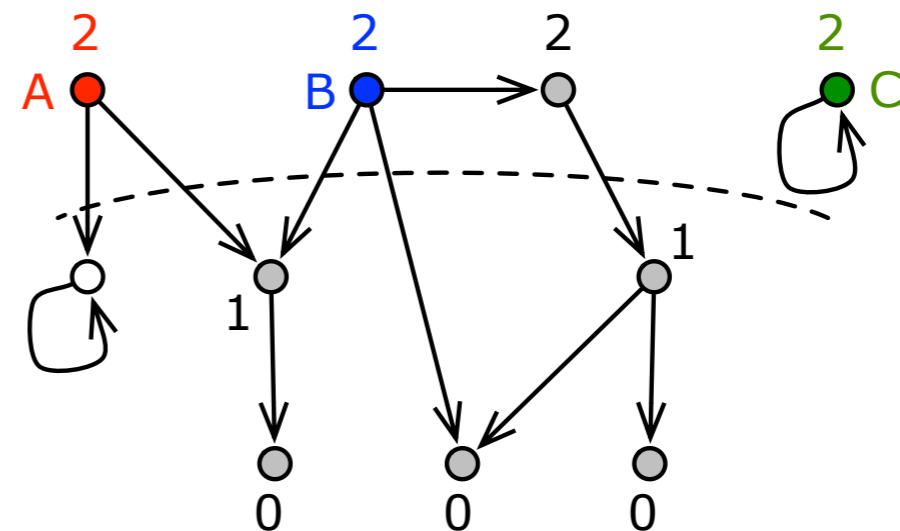
Let  $\gamma_A(\mathcal{R}_M^{\#n}(\ell))(\bar{\rho}) < \mathcal{R}_M(\ell)(\bar{\rho})$  for some  $\bar{\rho} \in \text{dom}(\mathcal{R}_M(\ell)) \cap \text{dom}(\gamma_A(\mathcal{R}_M^{\#n}(\ell)))$  (case B). Then, there exists  $\rho \in \text{dom}(\gamma_A(\mathcal{R}_M^{\#n+1}(\ell))) \cap \text{dom}(\mathcal{R}_M^{\#n}(\ell))$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



# Piecewise-Defined Ranking Functions Abstract Domain

Widening (continue)

Check for Case B or C



## Lemma

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

(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  $C$



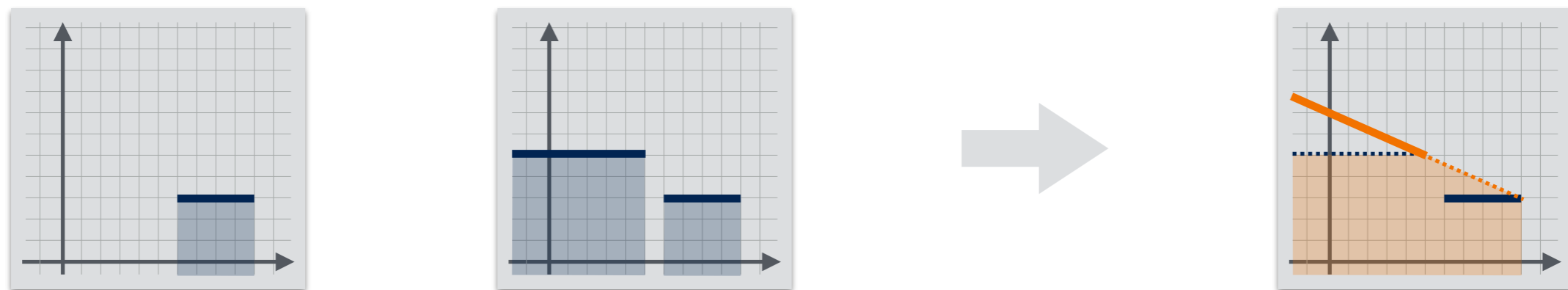
# Piecewise-Defined Ranking Functions Abstract Domain

## 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**  
 $\nabla_F [\alpha_C(\bar{C}), \alpha_C(C)]$ , where  $\bar{C}$  is the set of constraints along the path to  $\bar{f}$

Example:

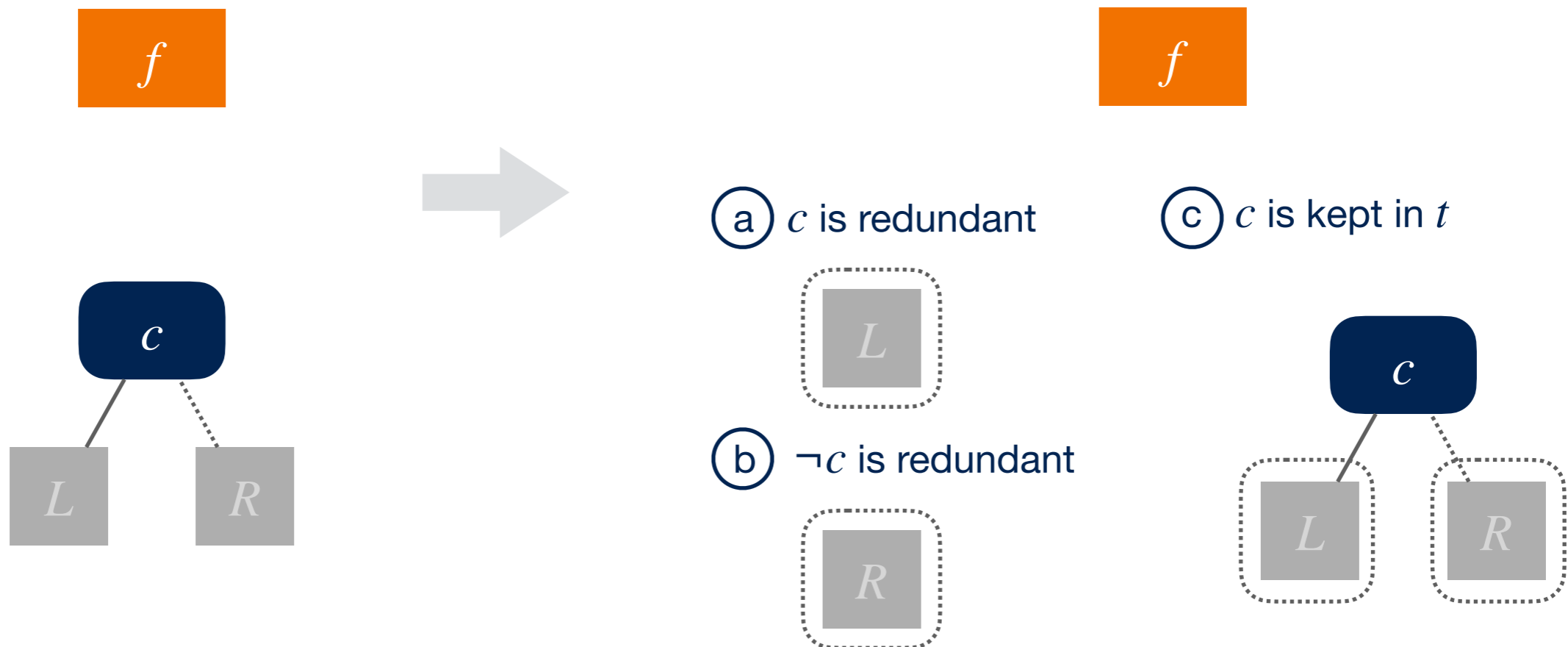


# Piecewise-Defined Ranking Functions Abstract Domain

## Tree Pruning

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

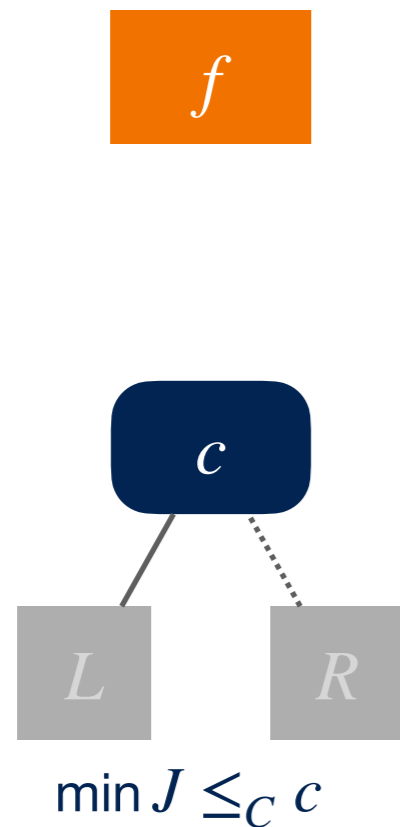
- Base case ( $J = \emptyset$ )



# Piecewise-Defined Ranking Functions Abstract Domain

## Tree Pruning (continue)

- Case ①



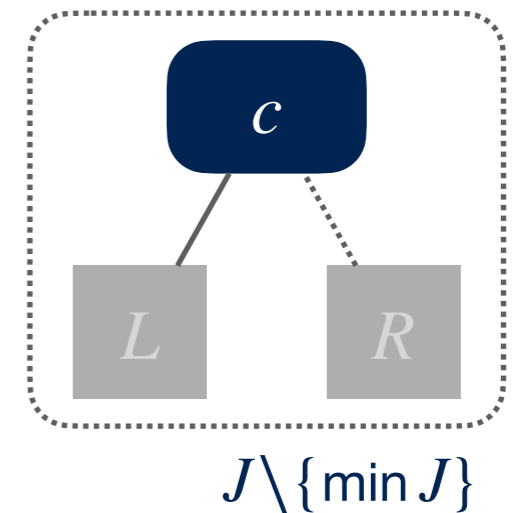
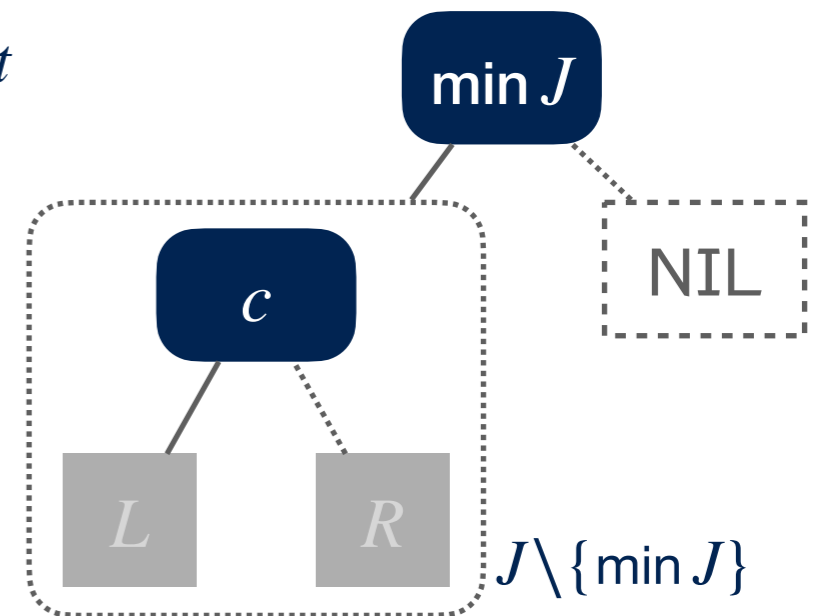
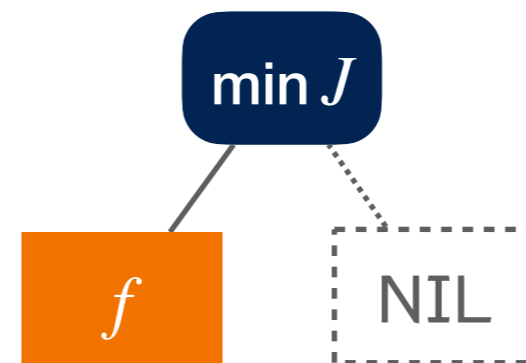
①a)  $\min J$  is redundant



①b)  $\neg \min J$  is redundant



①c)  $\min J$  is added to  $t$

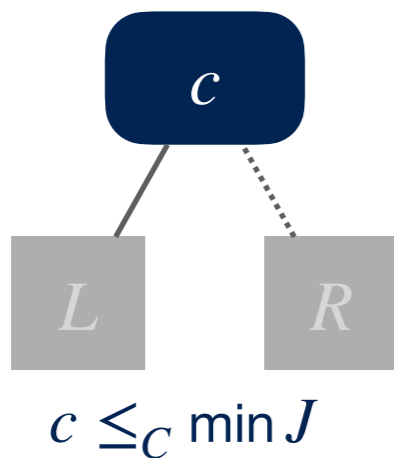




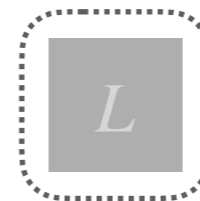
# Piecewise-Defined Ranking Functions Abstract Domain

## Tree Pruning (continue)

- Case ②



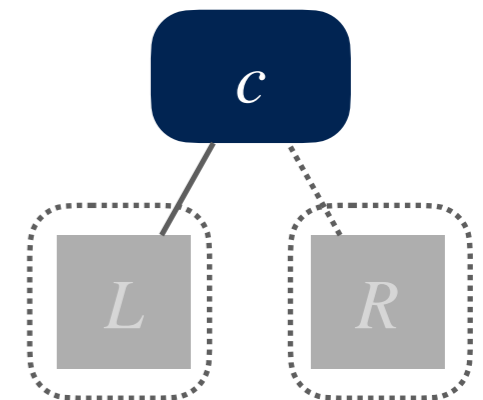
- ②a)  $c$  is redundant



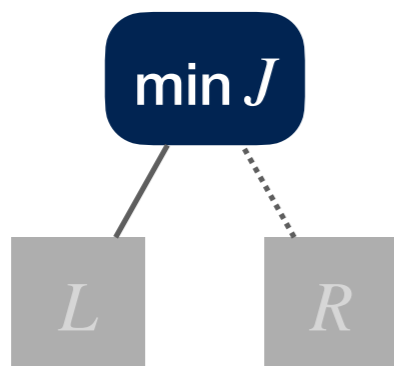
- ②b)  $\neg c$  is redundant



- ②c)  $c$  is kept in  $t$



- Case ③



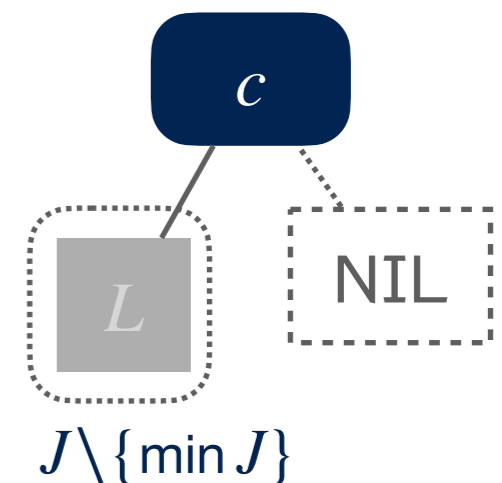
- ③a)  $\min J$  is redundant



- ③b)  $\neg \min J$  is redundant



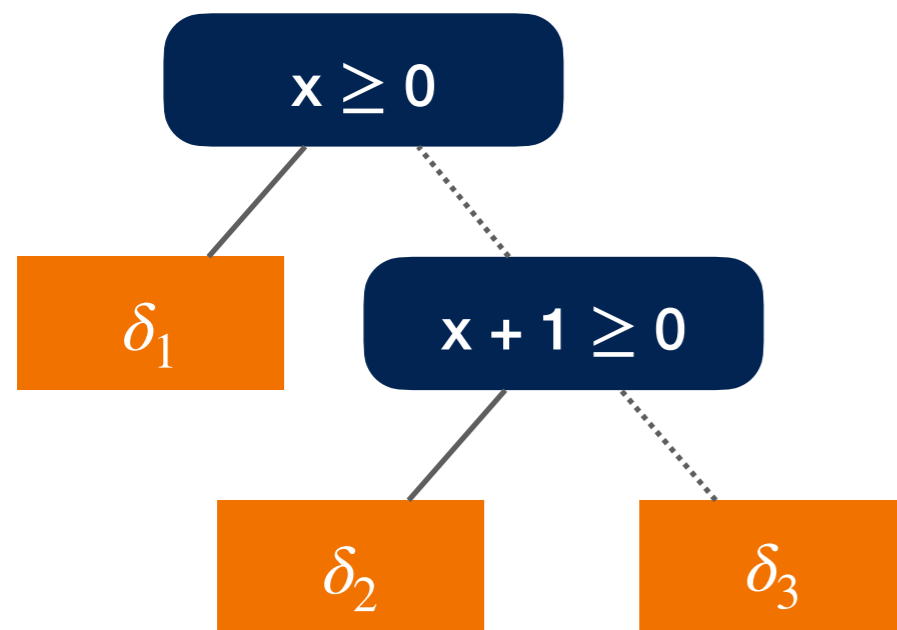
- ③c)  $\min J$  is kept in  $t$



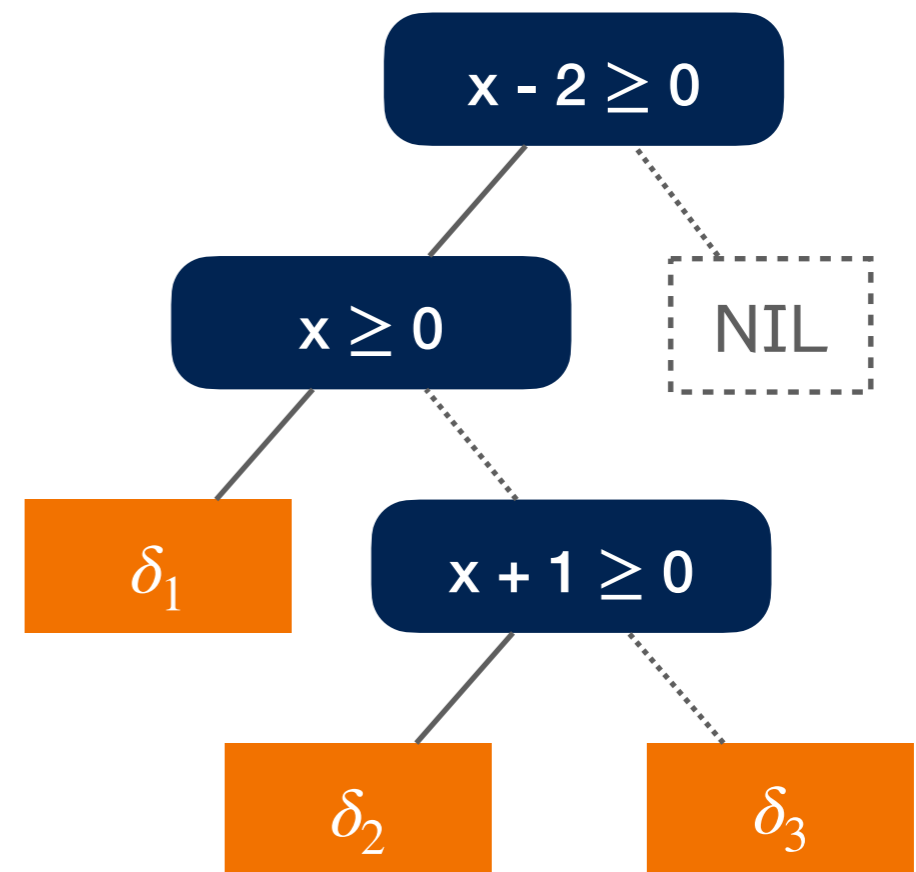
# Piecewise-Defined Ranking Functions Abstract Domain

Tree Pruning (continue)

Example



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



# 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 defined leaf nodes

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

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

Example:

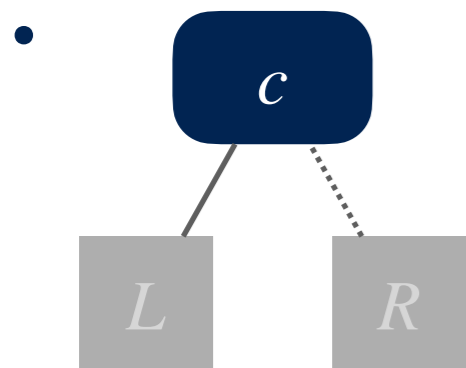
$$\overleftarrow{\text{ASSIGN}}_F[[x \leftarrow x + [1,2]]][\top_D](\lambda x . x + 1) = \lambda x . x + 4$$

(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

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

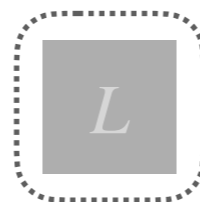


Convert  $\overleftarrow{\text{ASSIGN}}_D[[X \leftarrow e]](\alpha_C(\{c\}))$  and  $\overleftarrow{\text{ASSIGN}}_D[[X \leftarrow e]](\alpha_C(\{\neg c\}))$  into sets  $I$  and  $J$  of linear constraints *in canonical form*

case ①  $I = J = \emptyset$



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



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



case ④

1. perform **tree pruning** on  and 

2. join the results with  $Y_A$

# Piecewise-Defined Ranking Functions Abstract Domain

## Tests

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

1. Recursively descend the tree and apply  $\text{STEP}_F$  on the defined leaf nodes to account for one more execution step needed before termination:

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

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

Example:  $\alpha_C(\text{FILTER}_D[[e]] \top_D)$

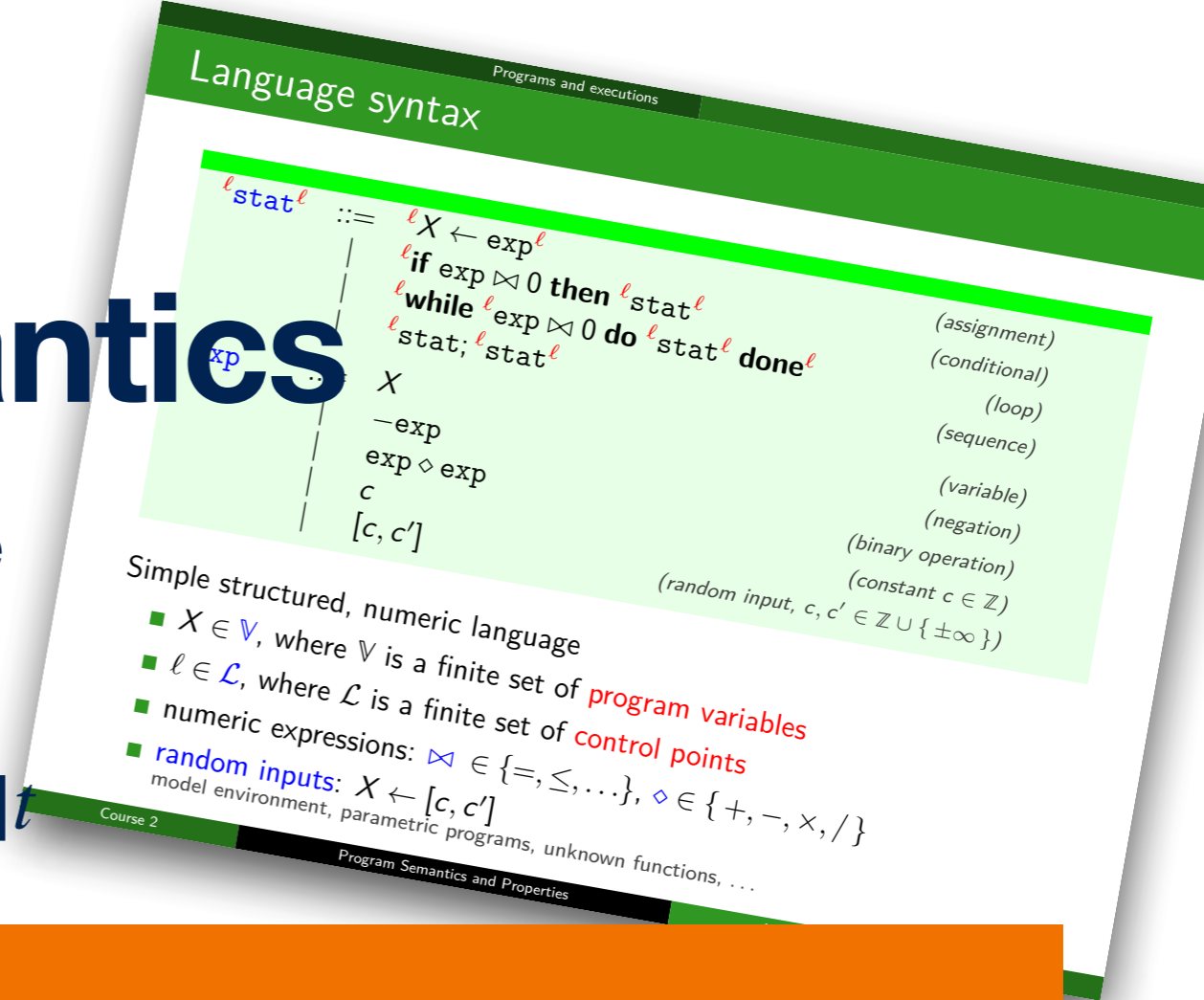
where  $\langle \mathcal{D}, \sqsubseteq_D \rangle$  is the underlying numerical domain

3. **Perform tree pruning with  $J$**

# Abstract Definite Termination Semantics

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

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



## Lemma (Soundness)

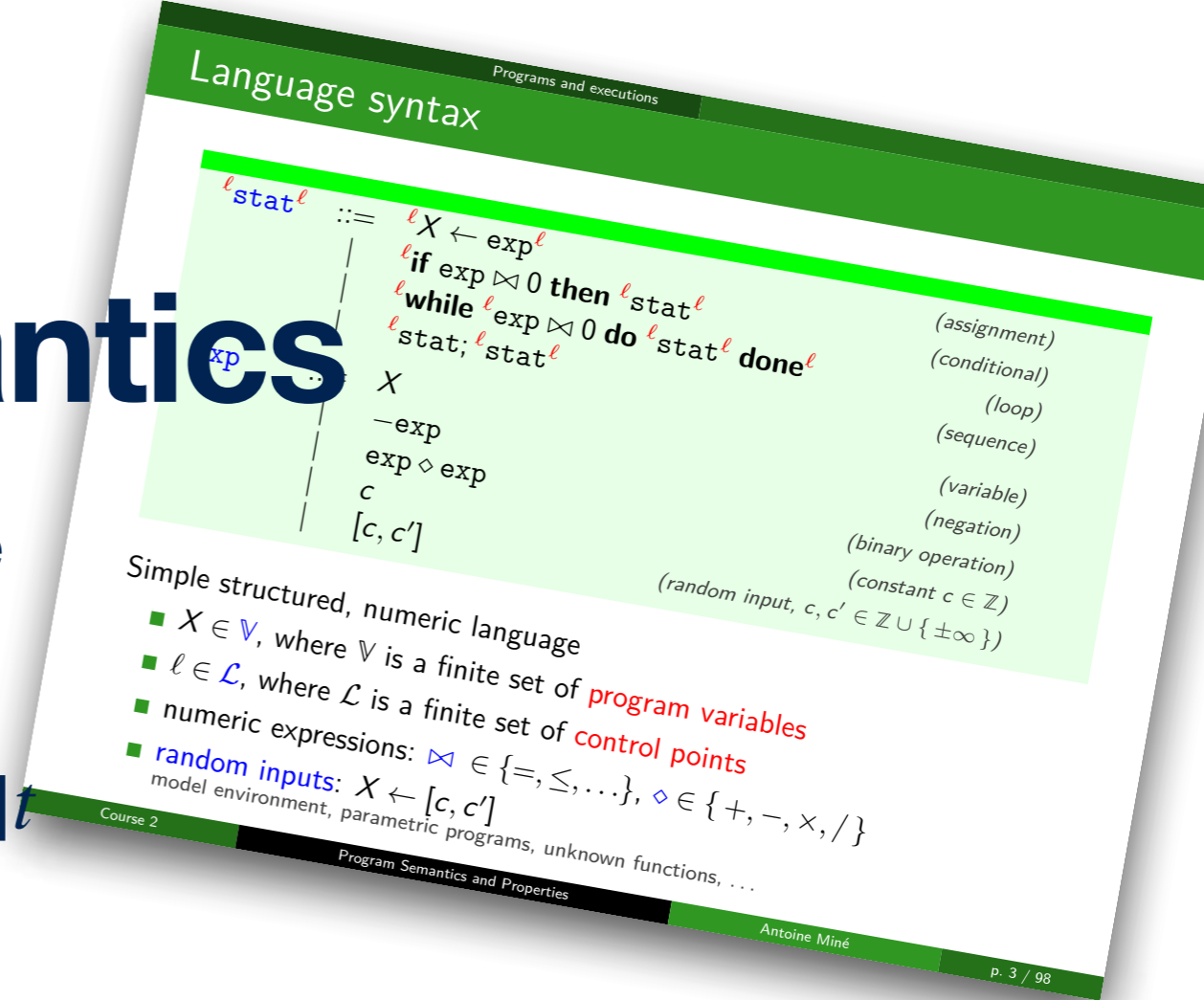
$$\mathcal{R}_M \llbracket \ell X \leftarrow e \rrbracket \gamma_A(t) \preceq \gamma_A(\mathcal{R}_M^\# \llbracket \ell X \leftarrow e \rrbracket t)$$

(see proof in [Urban15])

# Abstract Definite Termination Semantics

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

- $\mathcal{R}_M^\# \llbracket \ell X \leftarrow e \rrbracket t \stackrel{\text{def}}{=} \text{ASSIGN}_A \llbracket X \leftarrow e \rrbracket t$
- $\mathcal{R}_M^\# \llbracket \text{if } \ell e \bowtie 0 \text{ then } s \rrbracket t \stackrel{\text{def}}{=} \text{FILTER}_A \llbracket e \bowtie 0 \rrbracket (\mathcal{R}_M^\# \llbracket s \rrbracket t) \vee_T \text{FILTER}_A \llbracket e \not\bowtie 0 \rrbracket t$



## Lemma (Soundness)

$$\mathcal{R}_M \llbracket \text{if } \ell e \bowtie 0 \text{ then } s \rrbracket \gamma_A(t) \preceq \gamma_A(\mathcal{R}_M^\# \llbracket \text{if } \ell e \bowtie 0 \text{ then } s \rrbracket t)$$

(see proof in [Urban15])

# Abstract Definite Termination Semantics

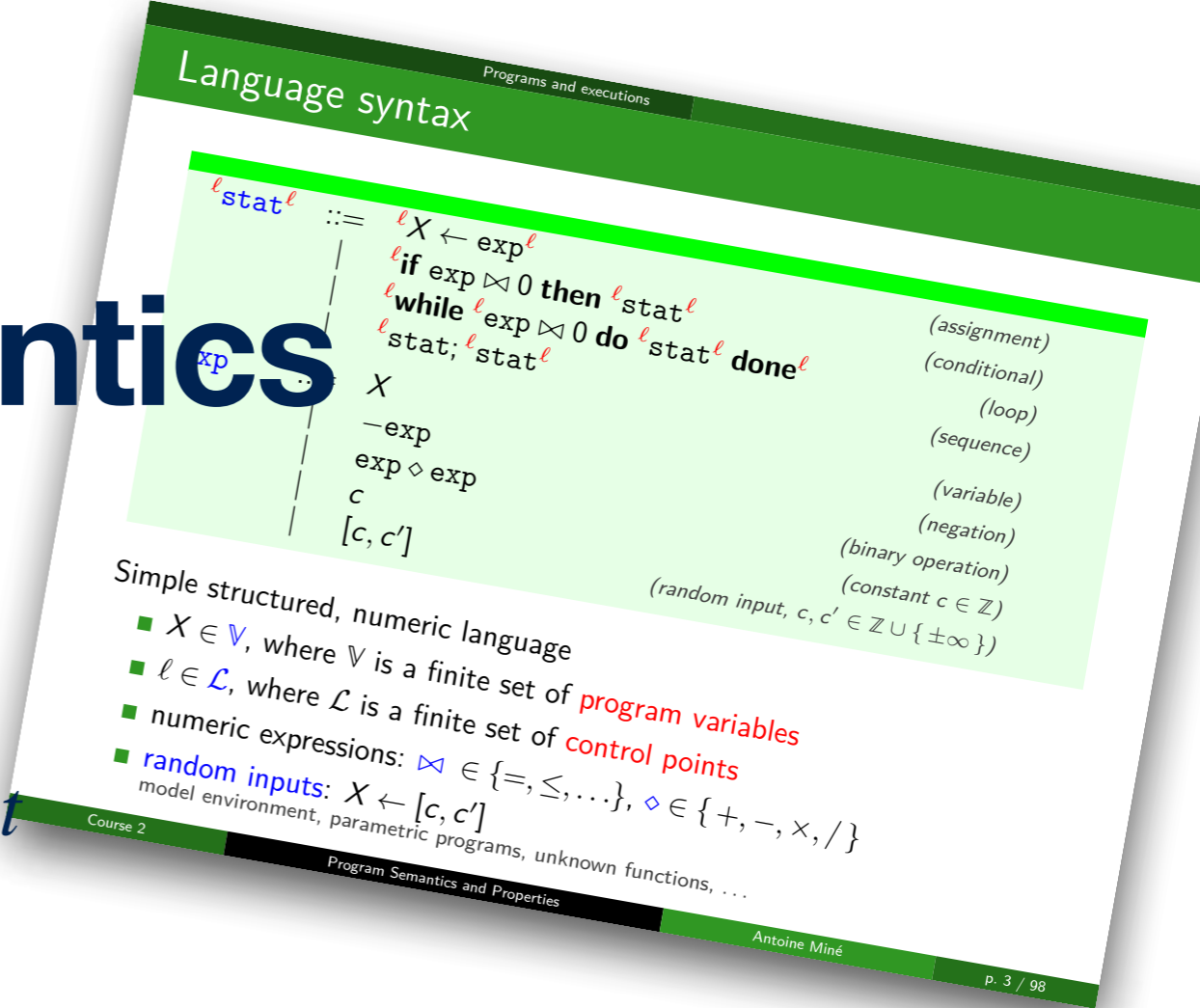
For each program instruction  $\text{stat}$ , we define a transformer  $\mathcal{R}_M^\#[\text{stat}]: \mathcal{A} \rightarrow \mathcal{A}$ :

- $\mathcal{R}_M^\#[\ell X \leftarrow e]t \stackrel{\text{def}}{=} \text{ASSIGN}_A[X \leftarrow e]t$
- $\mathcal{R}_M^\#[\text{if } \ell e \bowtie 0 \text{ then } s]t \stackrel{\text{def}}{=} \text{FILTER}_A[e \bowtie 0](\mathcal{R}_M^\#[s]t) \vee_T \text{FILTER}_A[e \bowtie 0]t$
- $\mathcal{R}_M^\#[\text{while } \ell e \bowtie 0 \text{ do } s \text{ done}]t \stackrel{\text{def}}{=} \text{lfp}^\# \bar{F}_M^\#$   
 where  $\bar{F}_M^\#(x) \stackrel{\text{def}}{=} \text{FILTER}_A[e \bowtie 0](\mathcal{R}_M^\#[s]x) \vee_T \text{FILTER}_A[e \bowtie 0](t)$

## Lemma (Soundness)

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

(see proof in [Urban15])

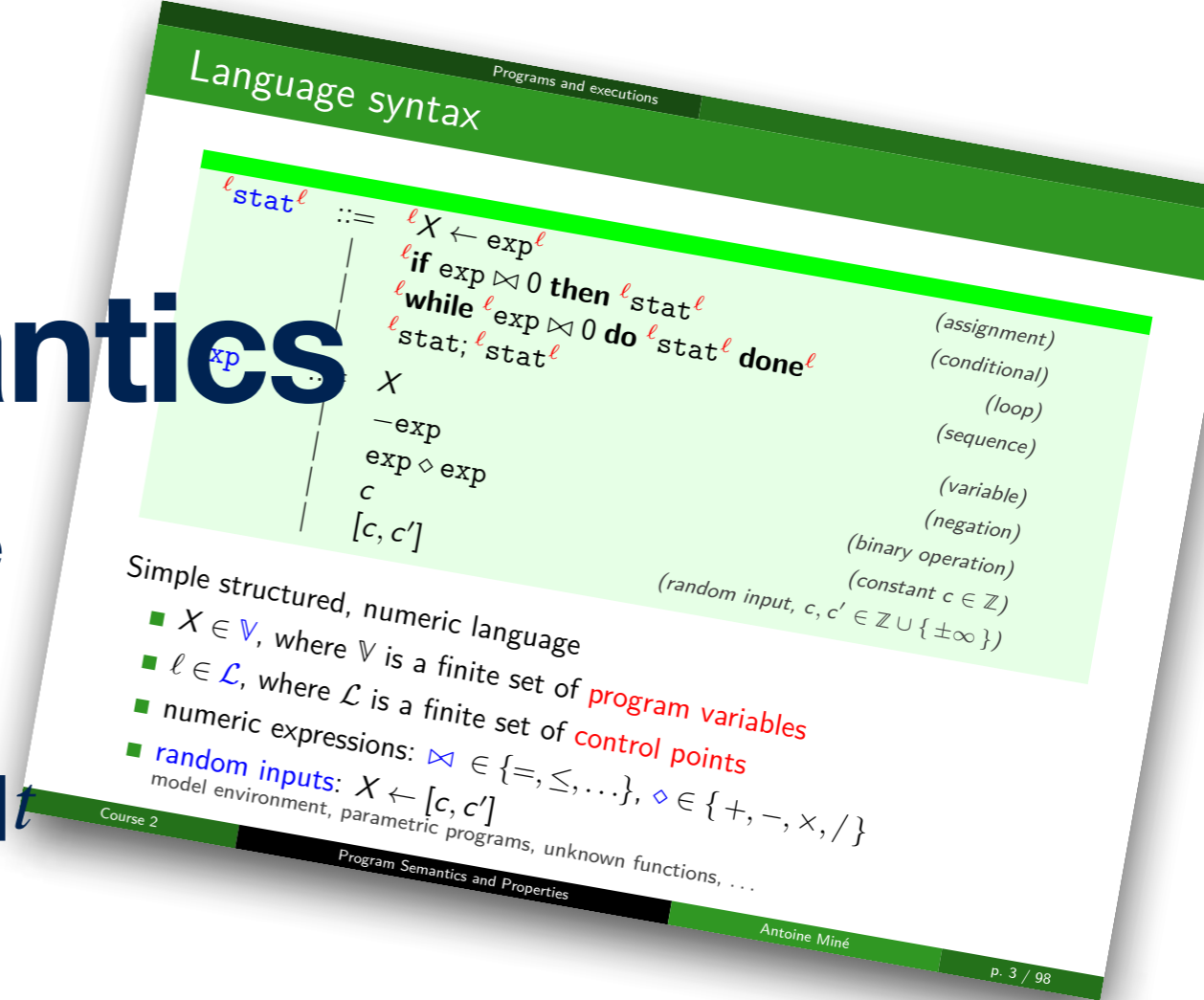




# Abstract Definite Termination Semantics

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

- $\mathcal{R}_M^\#[\ell X \leftarrow e]t \stackrel{\text{def}}{=} \text{ASSIGN}_A[X \leftarrow e]t$
- $\mathcal{R}_M^\#[\text{if } \ell e \bowtie 0 \text{ then } s]t \stackrel{\text{def}}{=} \text{FILTER}_A[e \bowtie 0](\mathcal{R}_M^\#[s]t) \vee_T \text{FILTER}_A[e \bowtie 0]t$
- $\mathcal{R}_M^\#[\text{while } \ell e \bowtie 0 \text{ do } s \text{ done}]t \stackrel{\text{def}}{=} \text{lfp}^\# \bar{F}_M^\#$   
 where  $\bar{F}_M^\#(x) \stackrel{\text{def}}{=} \text{FILTER}_A[e \bowtie 0](\mathcal{R}_M^\#[s]x) \vee_T \text{FILTER}_A[e \bowtie 0](t)$
- $\mathcal{R}_M^\#[s_1; s_2]t \stackrel{\text{def}}{=} \mathcal{R}_M^\#[s_1](\mathcal{R}_M^\#[s_2]t)$



# Abstract Definite Termination Semantics

## Definition

The **abstract definite termination semantics**  $\mathcal{R}_M^\# \llbracket \text{stat}^\ell \rrbracket \in \mathcal{A}$  of a program  $\text{stat}^\ell$  is:

$$\mathcal{R}_M^\# \llbracket \text{stat}^\ell \rrbracket \stackrel{\text{def}}{=} \mathcal{R}_M^\# \llbracket \text{stat} \rrbracket (\text{LEAF} : \lambda X_1, \dots, X_k. 0)$$

where  $\mathcal{R}_M^\# \llbracket \text{stat} \rrbracket : \mathcal{A} \rightarrow \mathcal{A}$  is the abstract definite termination semantics of each program instruction  $\text{stat}$

## Theorem (Soundness)

$$\mathcal{R}_M \llbracket \text{stat}^\ell \rrbracket \preceq \gamma_A(\mathcal{R}_M^\# \llbracket \text{stat}^\ell \rrbracket)$$

## Corollary (Soundness)

A program  $\text{stat}^\ell$  **must terminate** for traces starting from a set of initial states  $\mathcal{I}$  if  $\mathcal{I} \subseteq \text{dom}(\gamma_A(\mathcal{R}_M^\# \llbracket \text{stat}^\ell \rrbracket))$

# Abstract Definite Termination Semantics

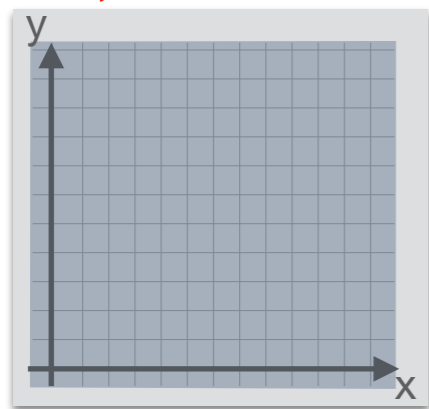
## Example

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

# Abstract Definite Termination Semantics

## Example

```
1  $x \leftarrow [-\infty, +\infty]$   
2  $y \leftarrow [-\infty, +\infty]$   
  while 3 $(x > 0)$  do  
    4 $x \leftarrow x - y$   
  od5
```

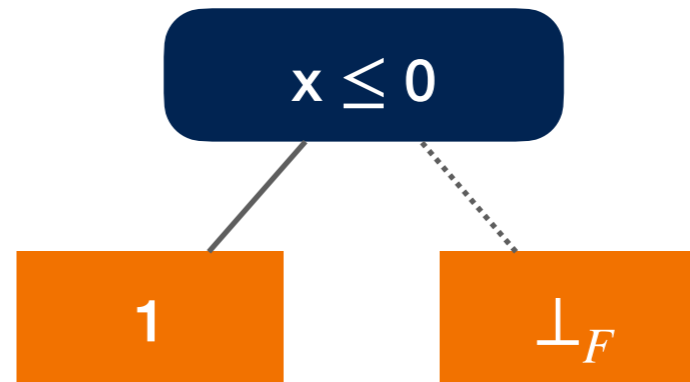
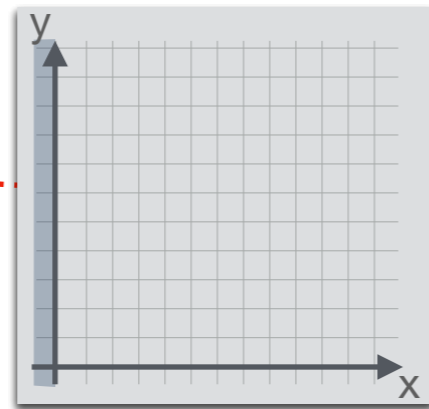


0

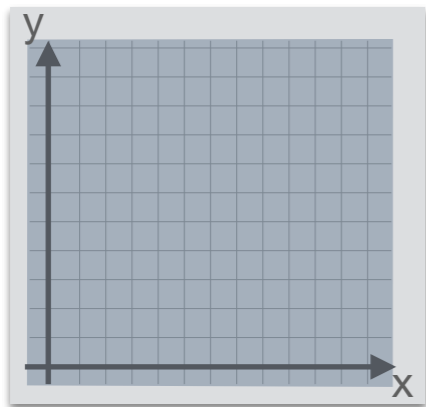
# Abstract Definite Termination Semantics

## Example

```
1  $x \leftarrow [-\infty, +\infty]$   
2  $y \leftarrow [-\infty, +\infty]$   
while 3  $(x > 0)$  do  
  4  $x \leftarrow x - y$   
od 5
```



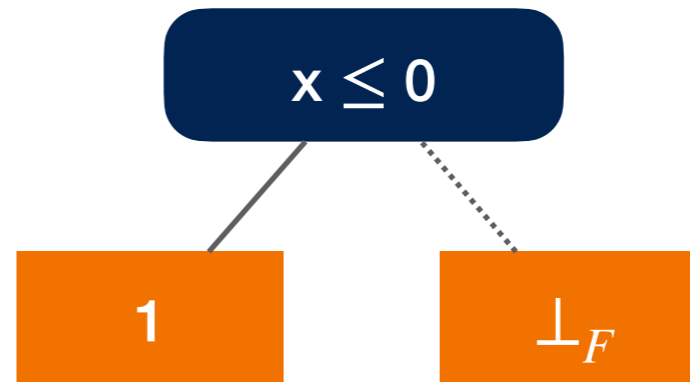
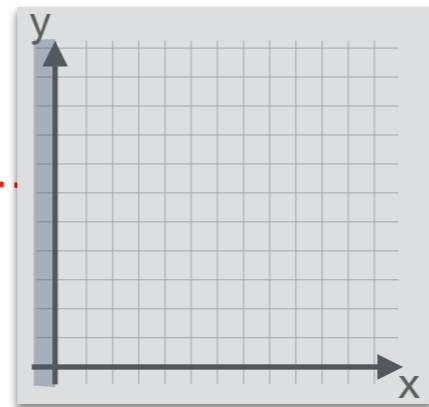
$\text{FILTER}_A[[x \leq 0]]$



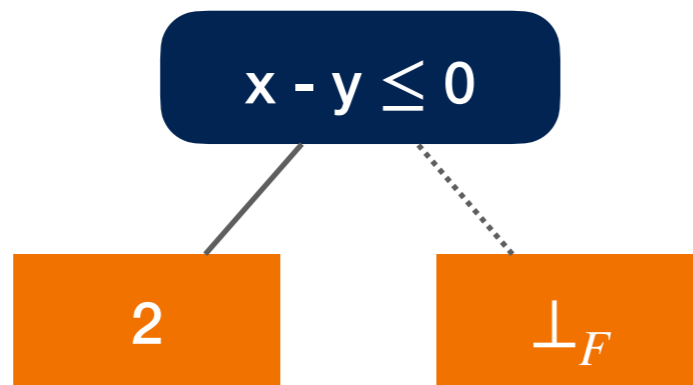
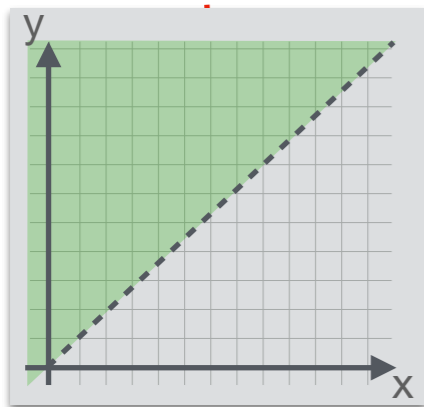
# Abstract Definite Termination Semantics

## Example

```
1  $x \leftarrow [-\infty, +\infty]$   
2  $y \leftarrow [-\infty, +\infty]$   
while 3  $(x > 0)$  do  
4  $x \leftarrow x - y$   
od 5
```



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



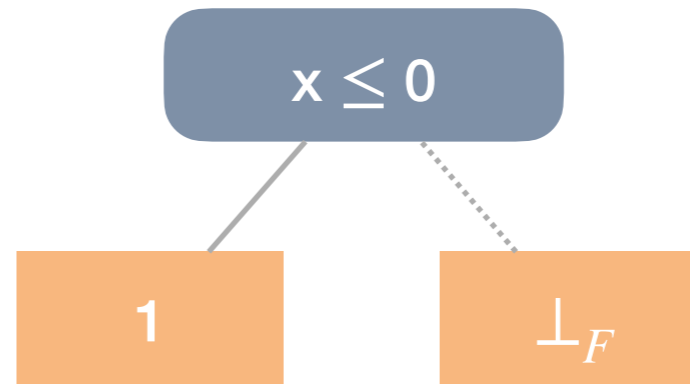
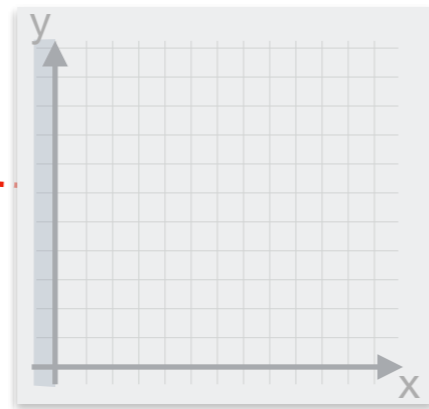
# Abstract Definite Termination Semantics

## Example

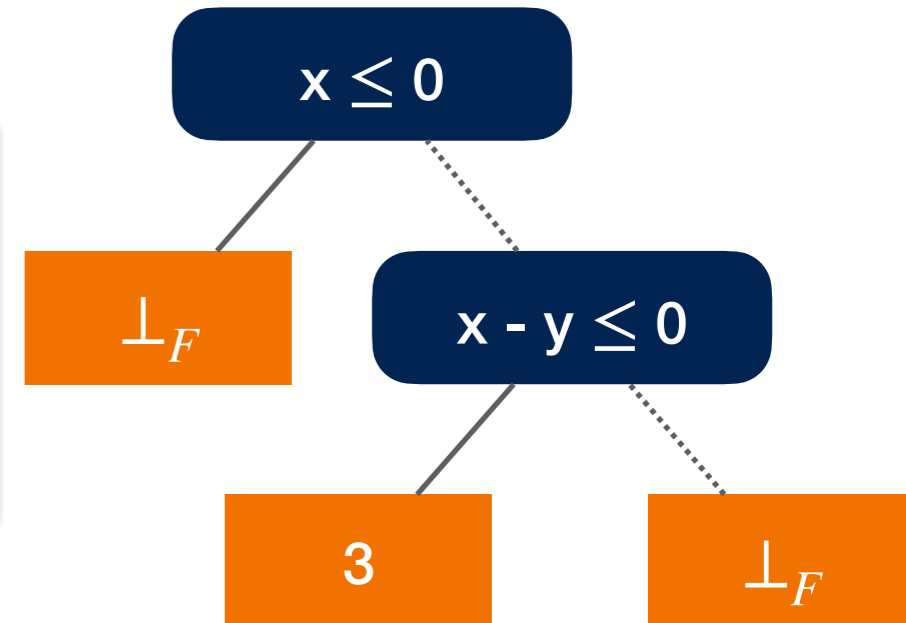
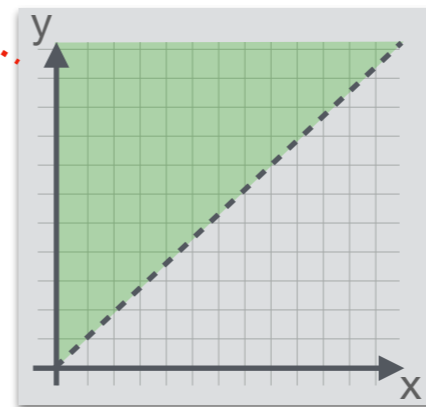
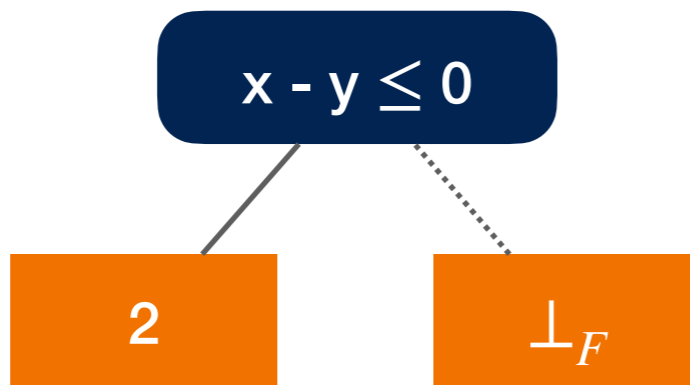
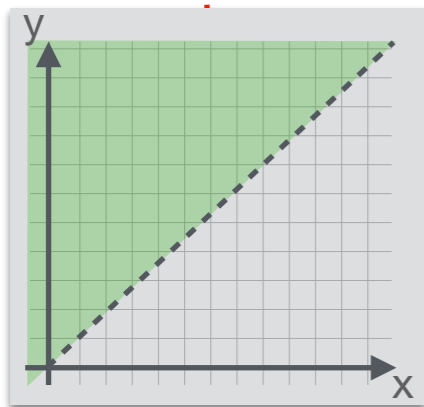
```

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

```



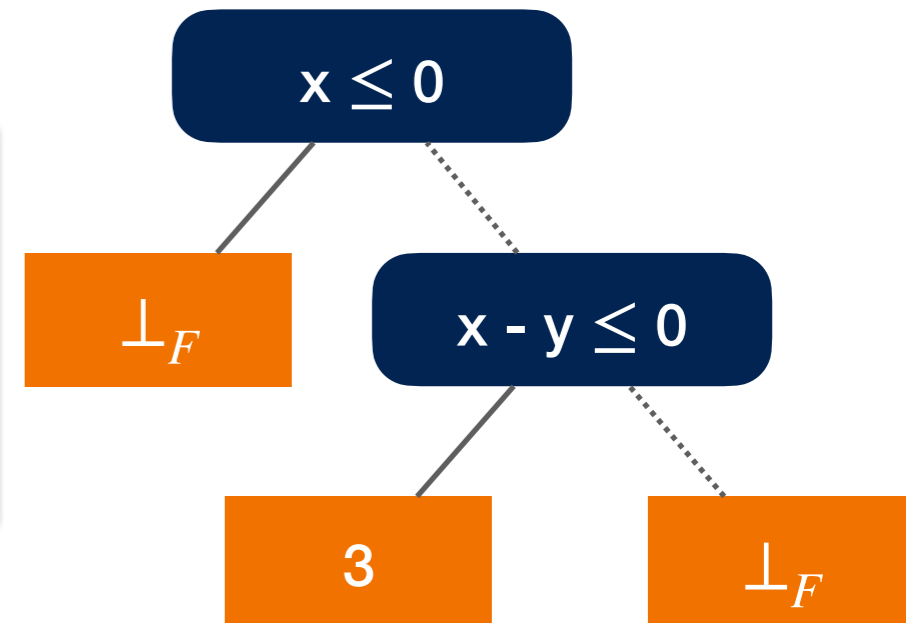
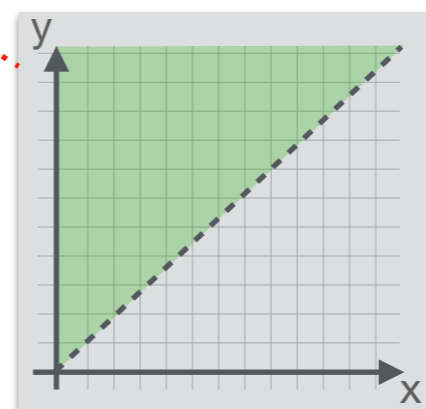
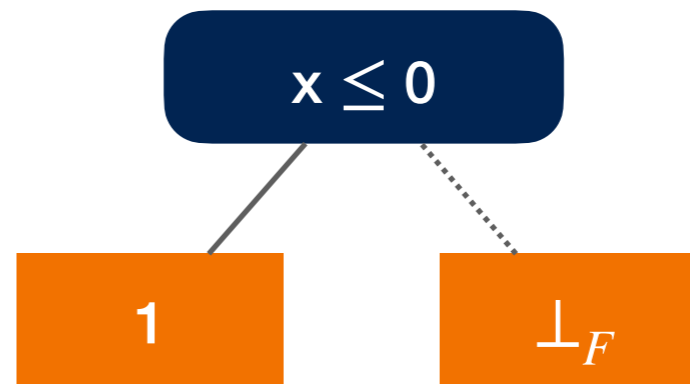
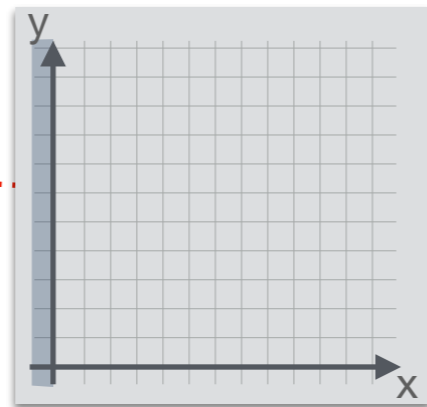
$\text{FILTER}_A[[x > 0]]$



# Abstract Definite Termination Semantics

## Example

```
1  $x \leftarrow [-\infty, +\infty]$   
2  $y \leftarrow [-\infty, +\infty]$   
while 3  $(x > 0)$  do  
    4  $x \leftarrow x - y$   
od 5
```



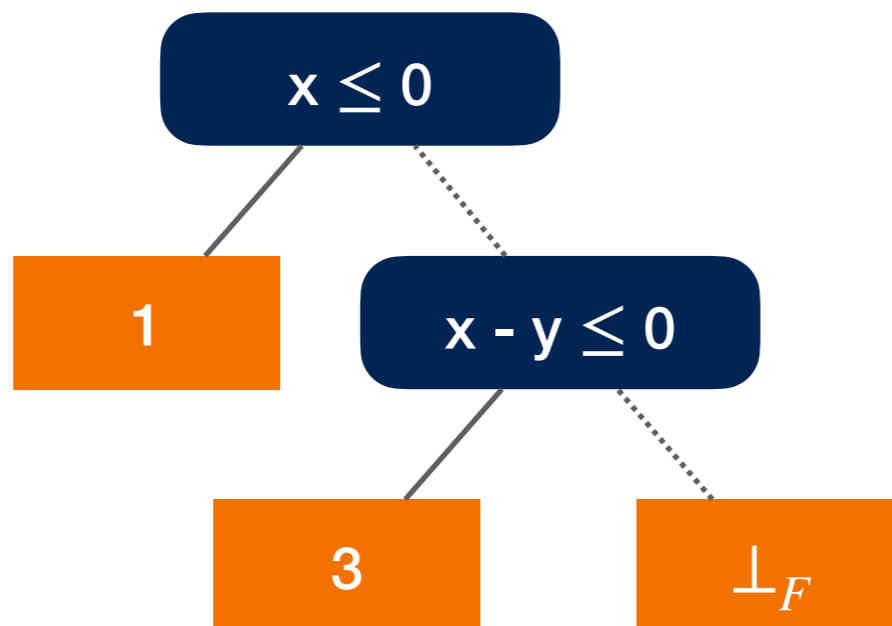
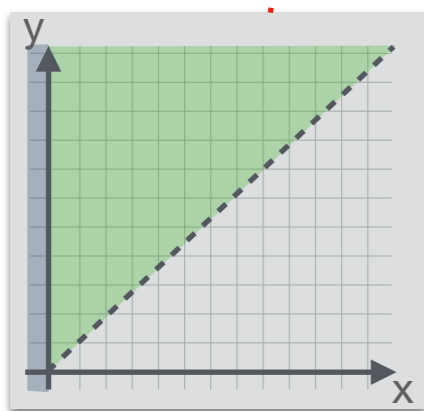


# Abstract Definite Termination Semantics

## Example

```
1  $x \leftarrow [-\infty, +\infty]$   
2  $y \leftarrow [-\infty, +\infty]$   
  while 3 $(x > 0)$  do  
    4 $x \leftarrow x - y$   
  od5
```

$\gamma_A$



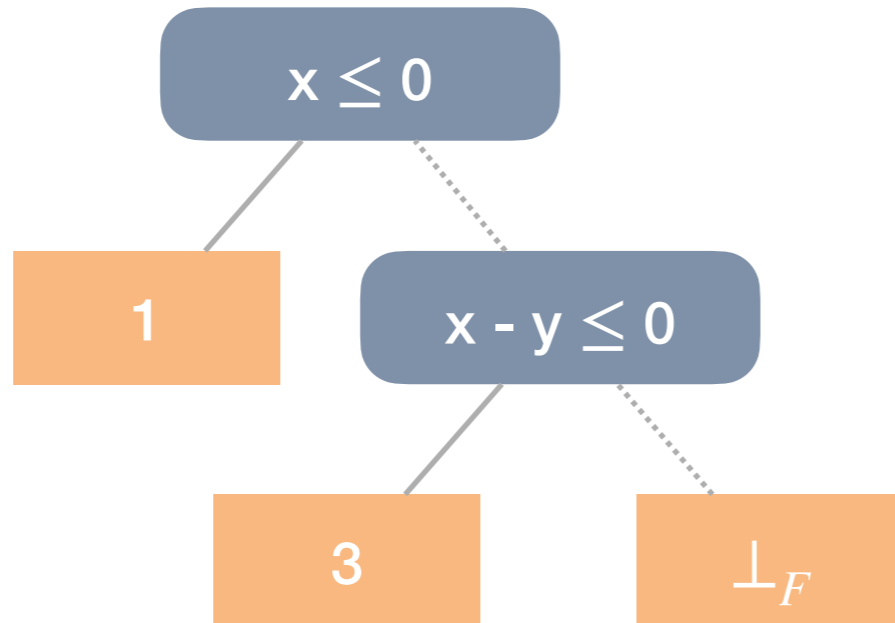
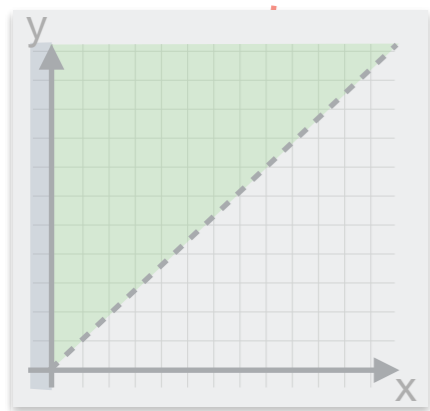
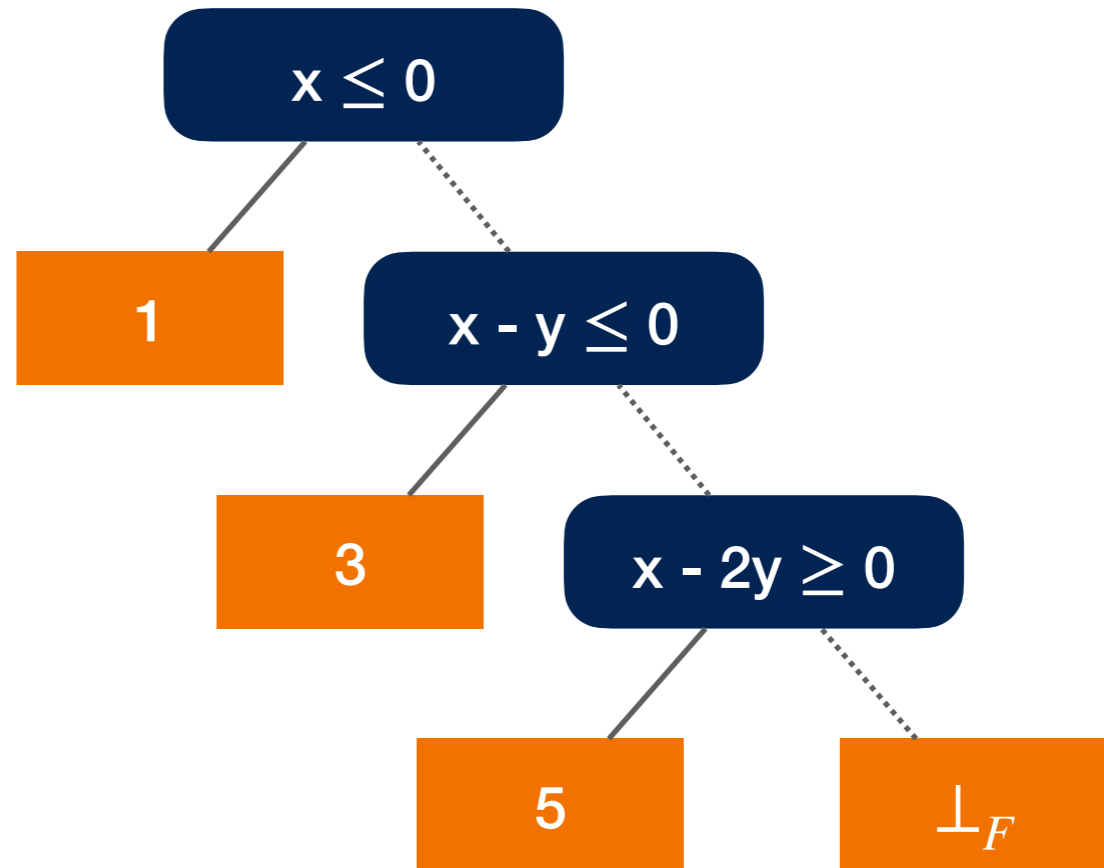
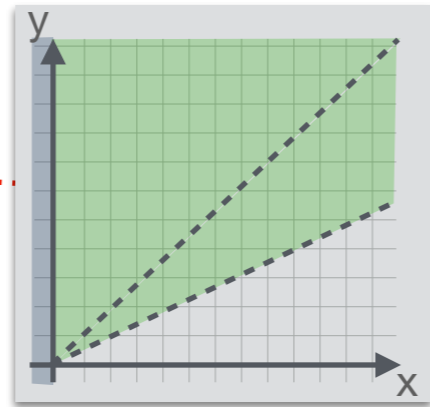
# Abstract Definite Termination Semantics

## Example

```

1  $x \leftarrow [-\infty, +\infty]$ 
2  $y \leftarrow [-\infty, +\infty]$ 
while 3 $(x > 0)$  do
4 $x \leftarrow x - y$ 
od5

```



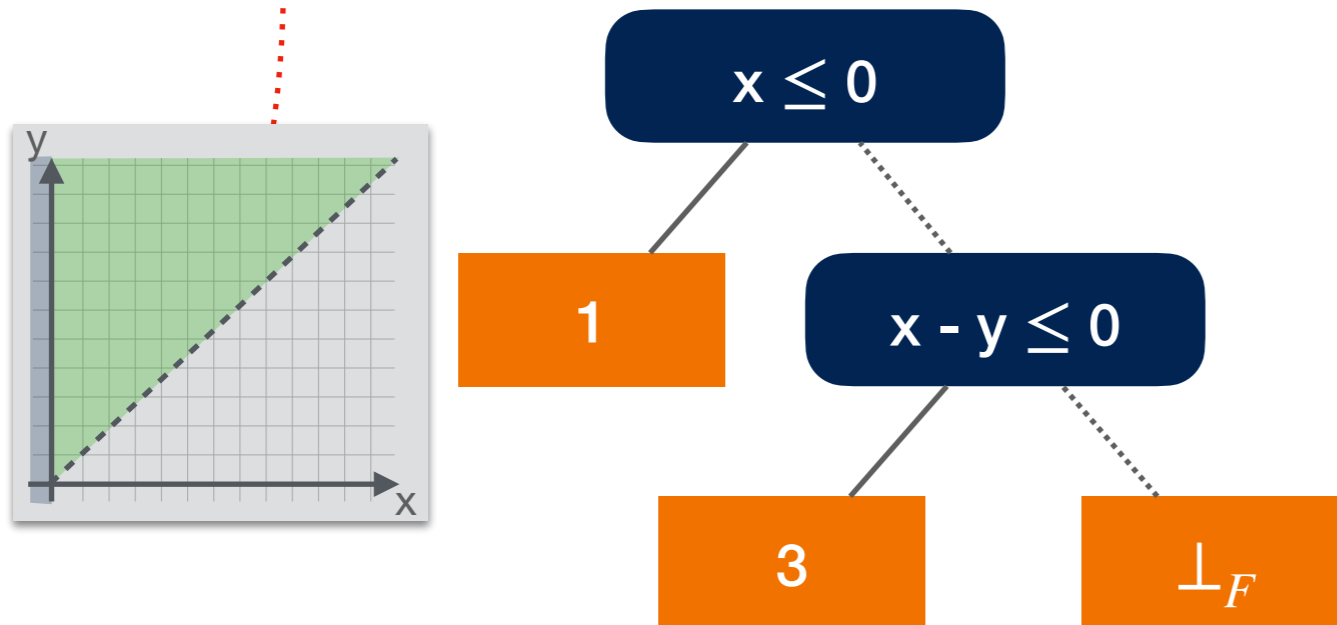
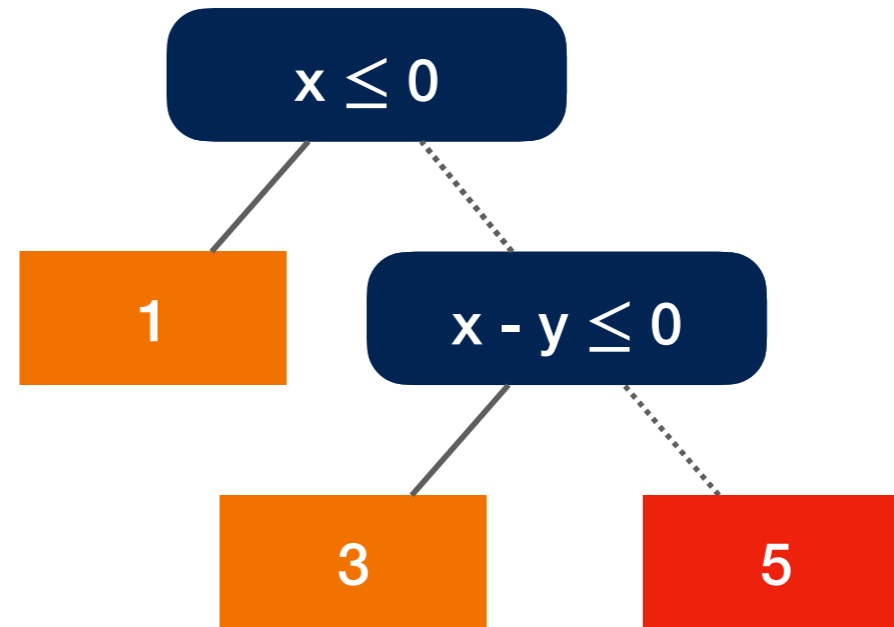
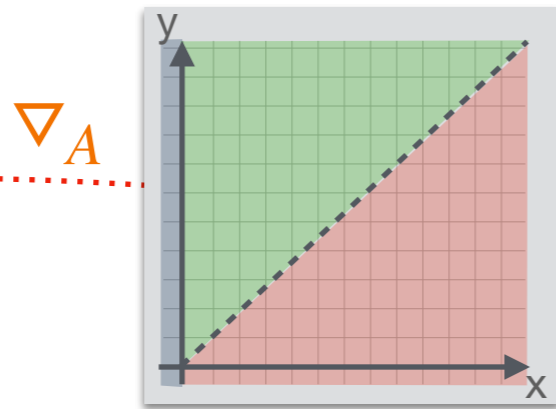
# Abstract Definite Termination Semantics

## Example

```

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

```



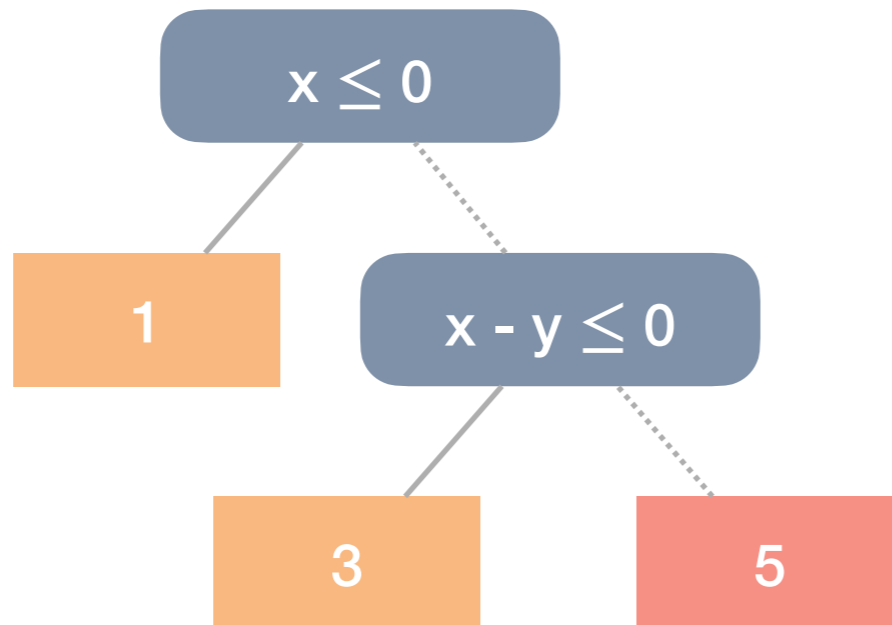
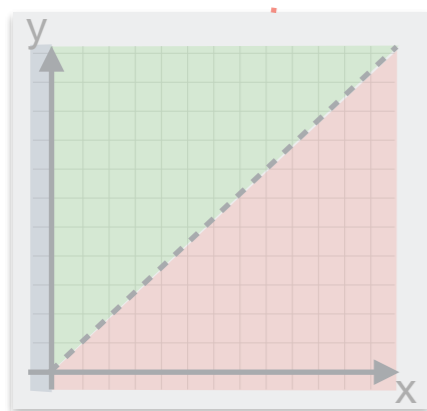
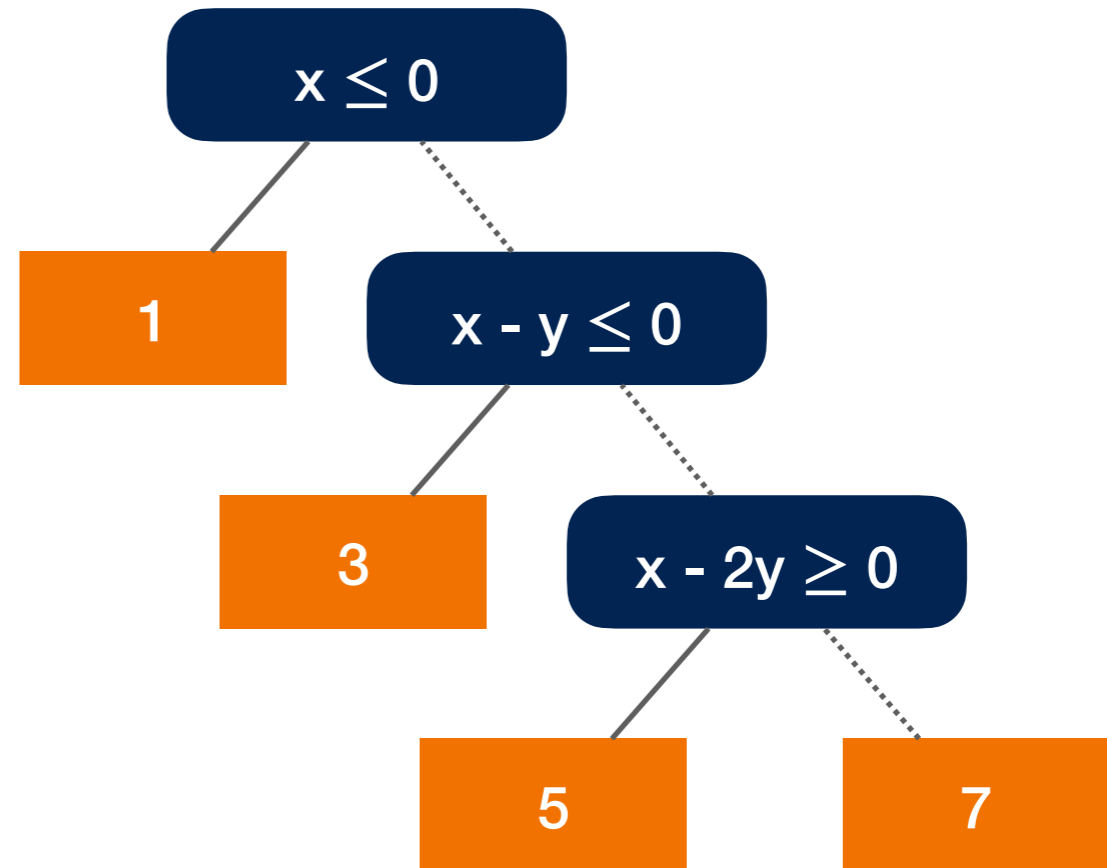
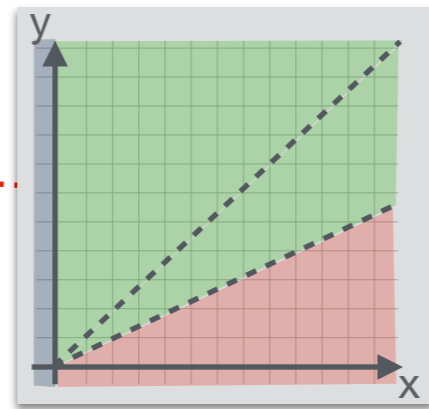
# Abstract Definite Termination Semantics

## Example

```

1  $x \leftarrow [-\infty, +\infty]$ 
2  $y \leftarrow [-\infty, +\infty]$ 
while 3 $(x > 0)$  do
4 $x \leftarrow x - y$ 
od5

```



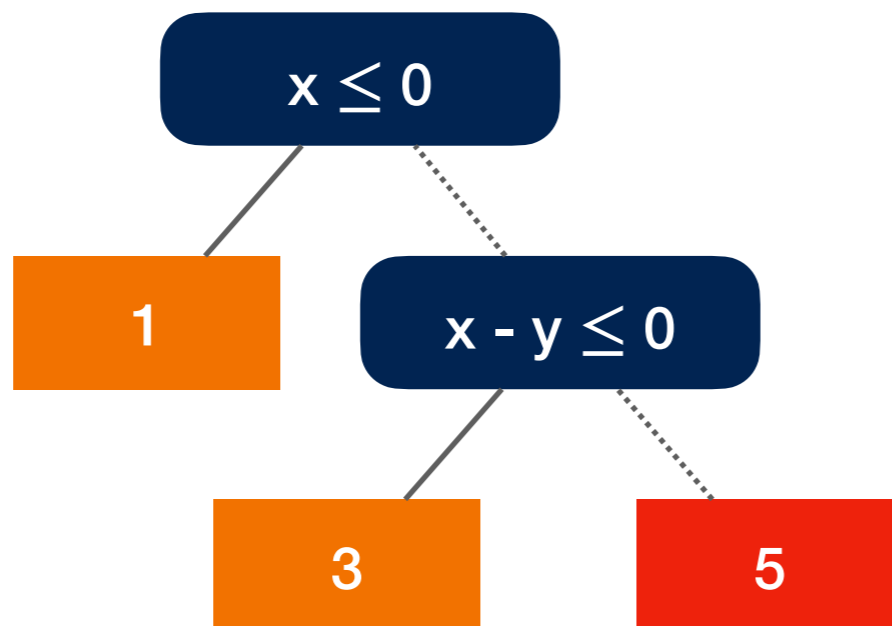
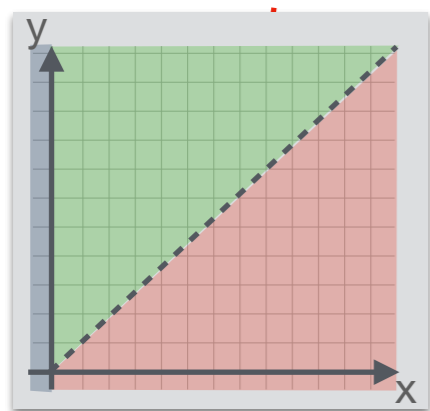
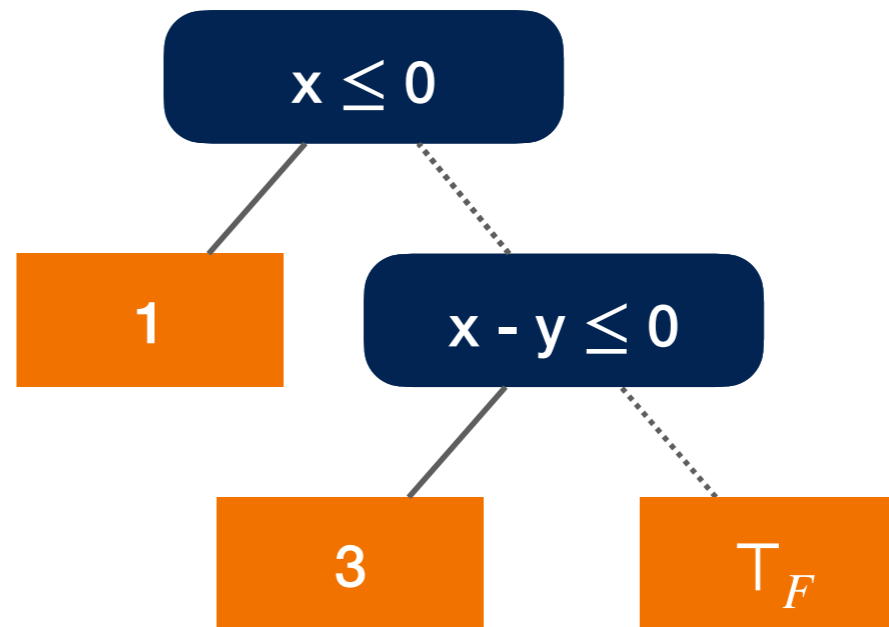
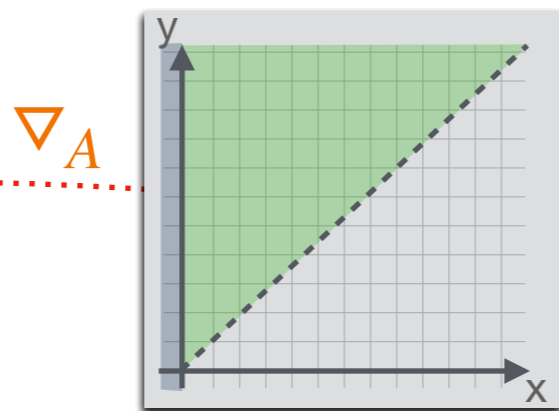
# Abstract Definite Termination Semantics

## Example

```

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

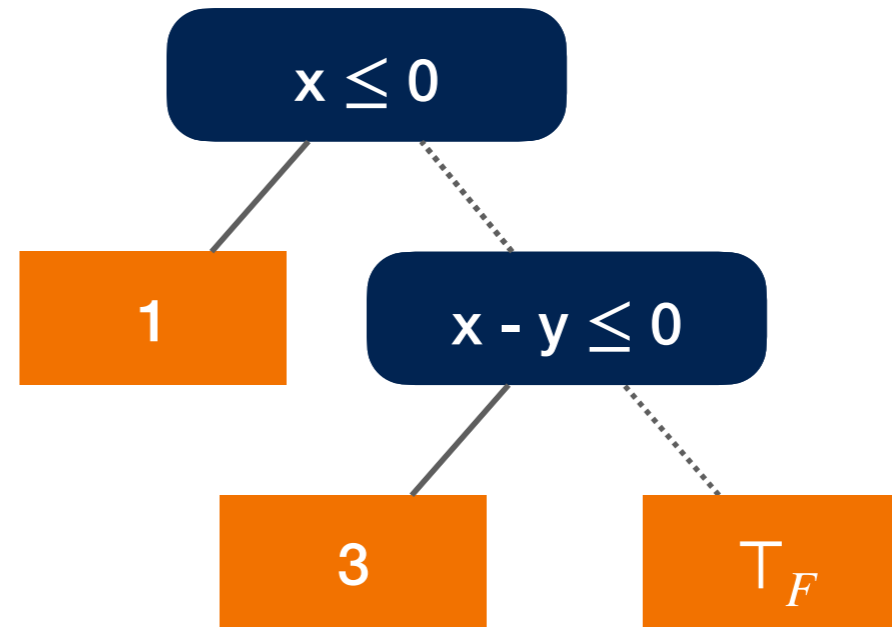
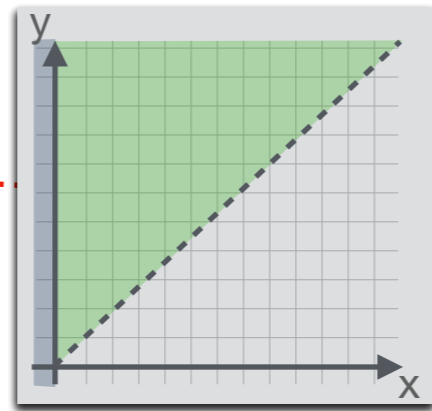
```



# Abstract Definite Termination Semantics

## Example

```
1  $x \leftarrow [-\infty, +\infty]$   
2  $y \leftarrow [-\infty, +\infty]$   
while 3  $(x > 0)$  do  
    4  $x \leftarrow x - y$   
od 5
```

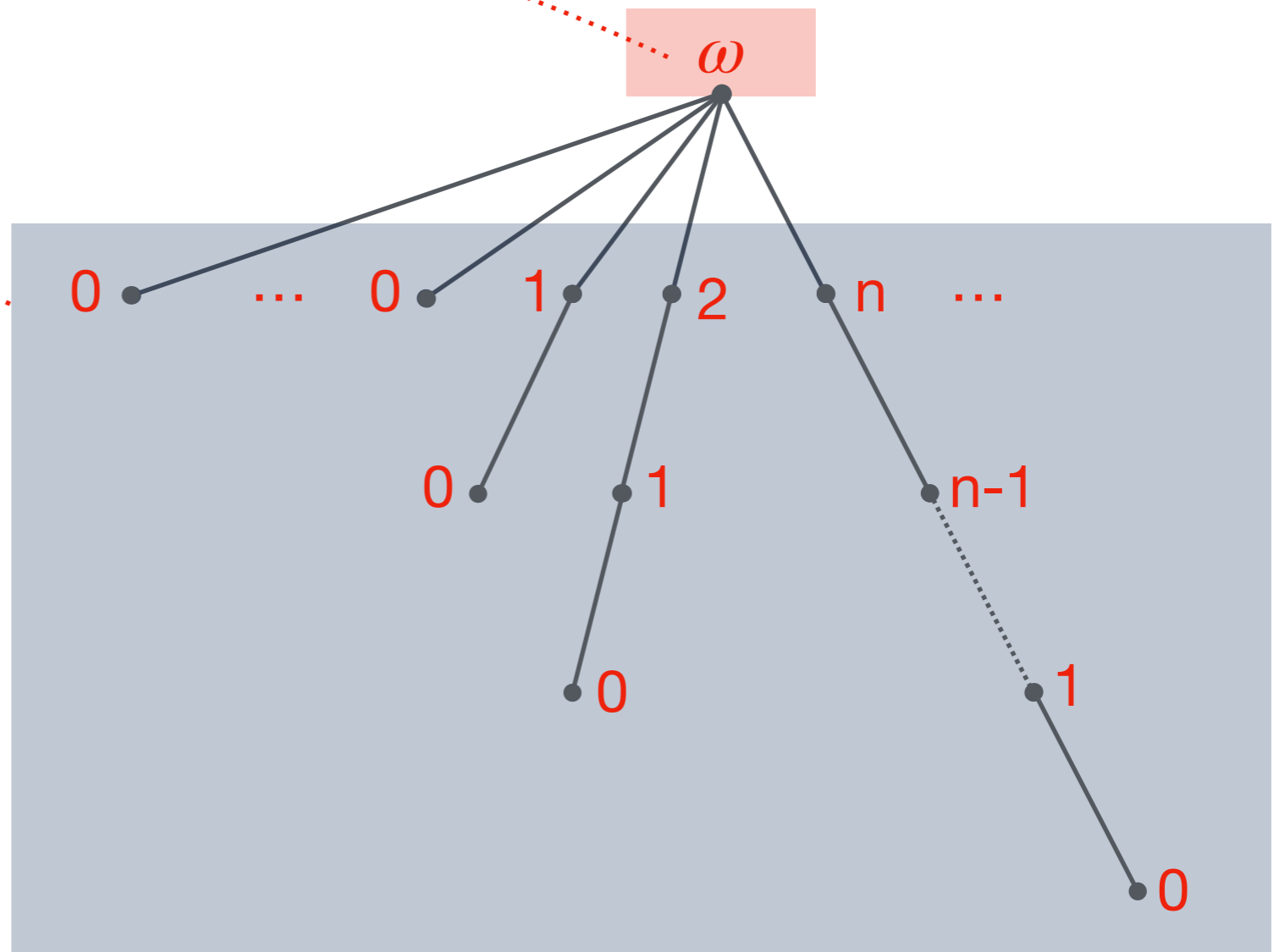


# Ordinal-Valued Raking Functions

# Need for Ordinals

## Example

```
1  $x \leftarrow [-\infty, +\infty]$   
while 2  $(x > 0)$  do  
  3  $x \leftarrow x - 1$   
od 4
```





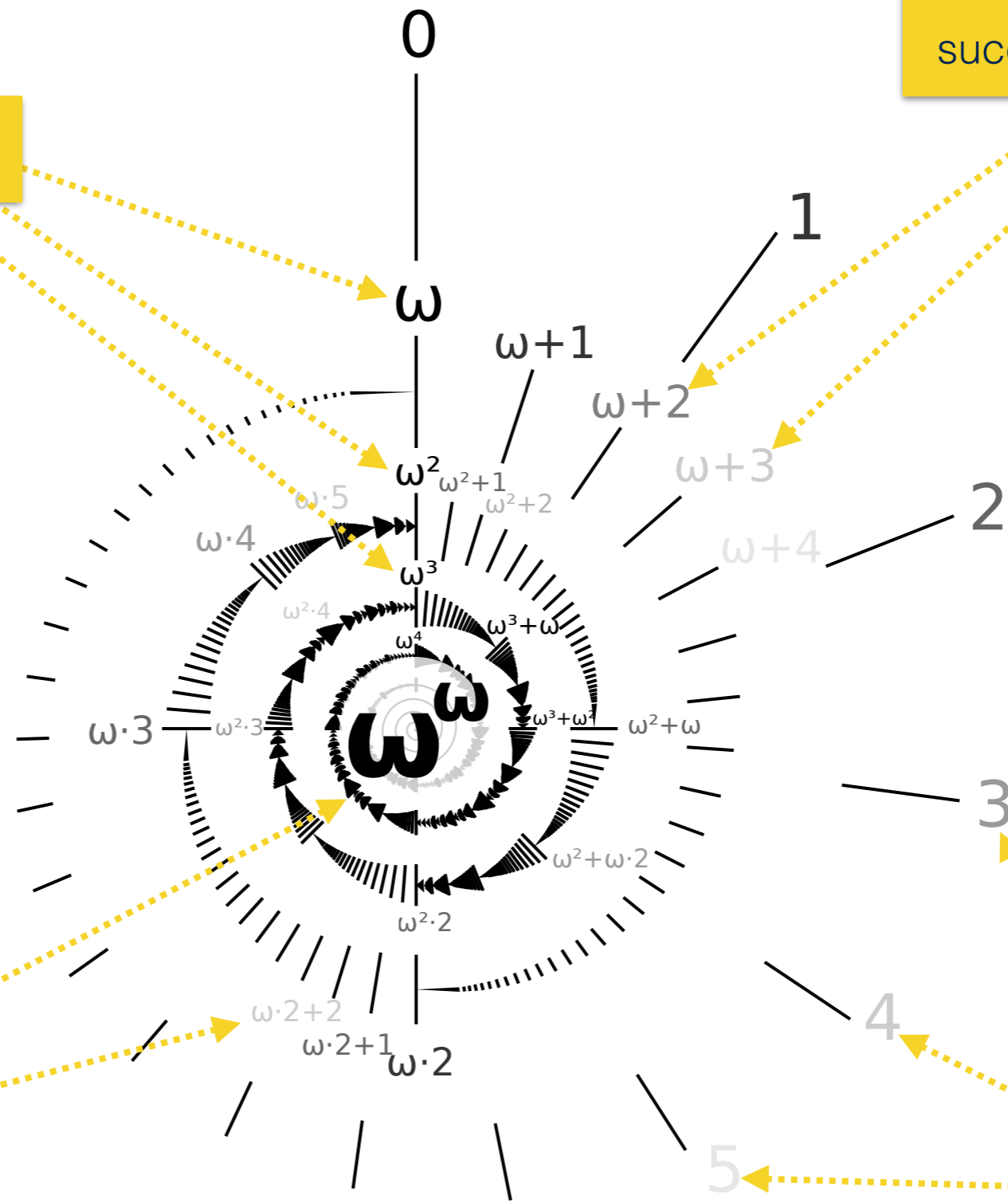
# Ordinals

successor ordinals  
 $\text{succ}(\alpha) \stackrel{\text{def}}{=} \alpha \cup \{\alpha\}$

limit ordinals

transfinite ordinals

finite ordinals



# Ordinal Arithmetic

## Addition

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

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

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

## Properties

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

# Ordinal Arithmetic

## Multiplication

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

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

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

## Properties

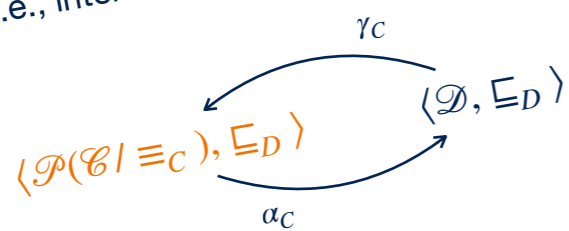
- **associative**  $(\alpha \cdot \beta) \cdot \gamma = \alpha \cdot (\beta \cdot \gamma)$
- **left distributive**  $\alpha \cdot (\beta + \gamma) = (\alpha \cdot \beta) + (\alpha \cdot \gamma)$
- **not commutative**  $2 \cdot \omega = \omega \neq \omega \cdot 2$
- **not right distributive**  $(\omega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

# Piecewise-Defined Ranking Functions Abstract Domain

## Piecewise-Defined Ranking Functions Abstract Domain

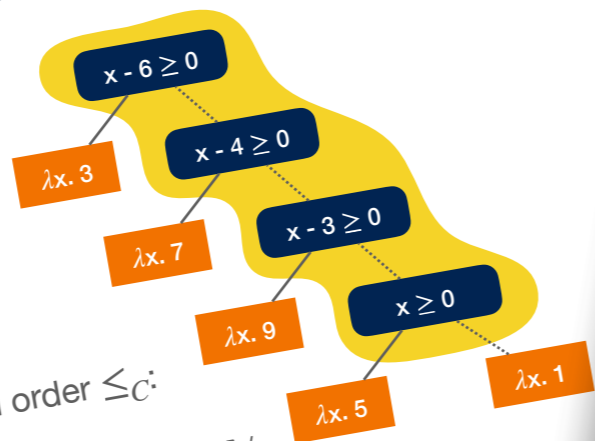
Linear Constraints Auxiliary Abstract Domain

- 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  $\leq_C$ :

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



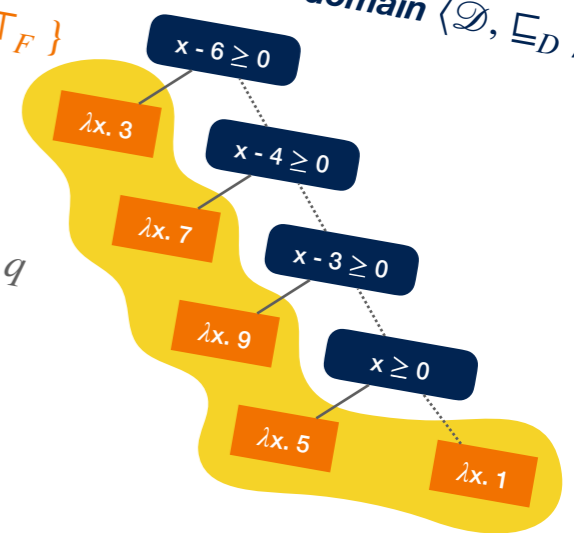
## Piecewise-Defined Ranking Functions Abstract Domain

Linear Constraints Auxiliary Abstract Domain

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

We consider **affine functions**:

$$\mathcal{F}_A \stackrel{\text{def}}{=} \{\perp_F\} \cup \{f: \mathbb{Z}^M \rightarrow \mathbb{N} \mid f(X_1, \dots, X_k) = \sum_{i=1}^k m_i \cdot X_i + q\} \cup \{\top_F\}$$



# Piecewise-Defined Ranking Functions Abstract Domain

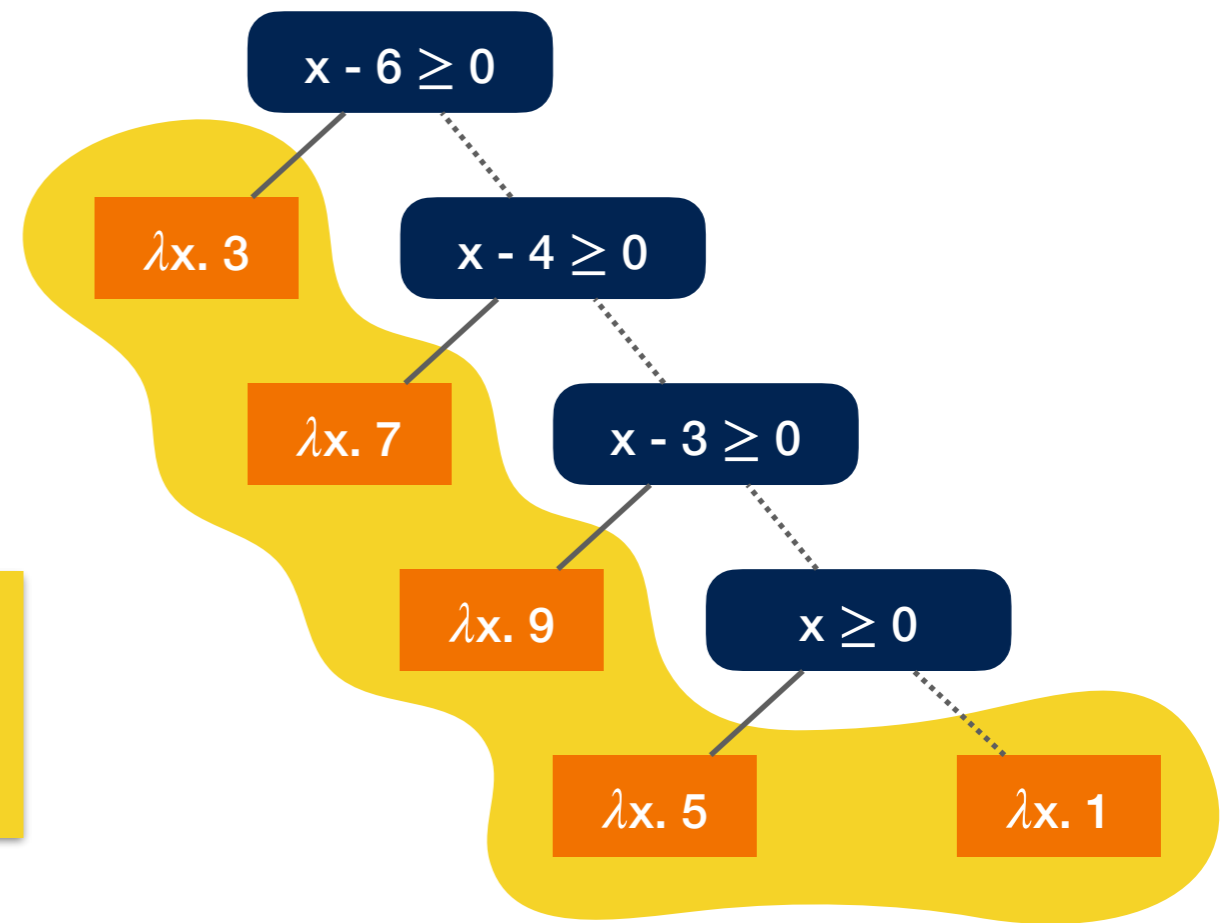
## Ordinal-Valued Functions Auxiliary Domain

- Parameterized by the *underlying functions auxiliary domain*  $\langle \mathcal{F}, \sqsubseteq_F \rangle$

- $$\mathcal{W} \stackrel{\text{def}}{=} \{ \perp_W \} \cup \left\{ \sum_i \omega^i \cdot f_i \mid f_i \in \mathcal{F} \setminus \{ \perp_F, \top_F \} \right\} \cup \{ \top_W \}$$

**Cantor Normal Form**  

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



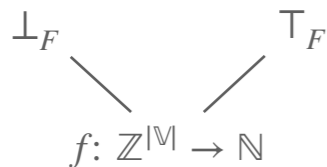
# Piecewise-Defined Ranking Functions Abstract Domain

## Ordinal-Valued Functions Auxiliary Domain (continue)

### Piecewise-Defined Ranking Functions Abstract Domain

Functions Auxiliary Abstract Domain (continue)

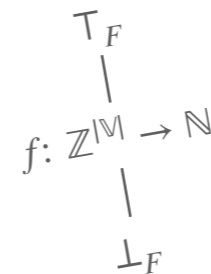
- **approximation order**  $\preceq_F[D]$ , where  $D \in \mathcal{D}$ :
  - between defined leaf nodes:
 
$$f_1 \preceq_F[D] f_2 \stackrel{\text{def}}{=} \forall \rho \in \gamma_D(D): f_1(\dots, \rho(X_i), \dots) \leq f_2(\dots, \rho(X_i), \dots)$$
  - otherwise (i.e., when one or both leaf nodes are undefined):



### Piecewise-Defined Ranking Functions Abstract Domain

Functions Auxiliary Abstract Domain (continue)

- **computational order**  $\sqsubseteq_F[D]$ , where  $D \in \mathcal{D}$ :
  - between defined leaf nodes:
 
$$f_1 \sqsubseteq_F[D] f_2 \stackrel{\text{def}}{=} \forall \rho \in \gamma_D(D): f_1(\dots, \rho(X_i), \dots) \leq f_2(\dots, \rho(X_i), \dots)$$
  - otherwise (i.e., when one or both leaf nodes are undefined):



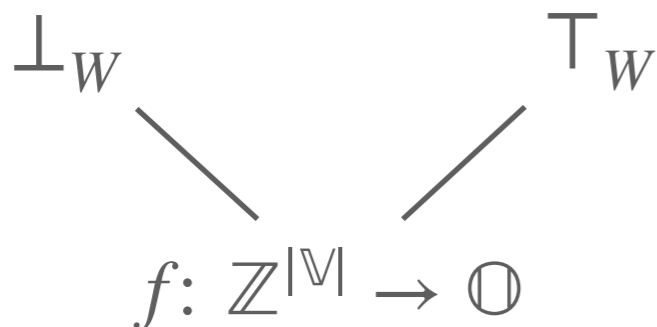
# Piecewise-Defined Ranking Functions Abstract Domain

## Ordinal-Valued Functions Auxiliary Domain (continue)

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

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

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



# Piecewise-Defined Ranking Functions Abstract Domain

## Ordinal-Valued Functions Auxiliary Domain (continue)

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

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

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

$$\begin{array}{c} \top_W \\ | \\ f: \mathbb{Z}^{|M|} \rightarrow \mathbb{O} \\ | \\ \perp_W \end{array}$$



# Piecewise-Defined Ranking Functions Abstract

- $\mathcal{A} \stackrel{\text{def}}{=} \{\text{LEAF}: f \mid f \in \mathcal{W}\} \cup \{\text{NODE}\{c\}: t_1; t_2 \mid c \in \mathcal{C} \wedge t_1, t_2 \in \mathcal{A}\}$
- **concretization function**  $\gamma_A: \mathcal{A} \rightarrow (\mathcal{E} \rightarrow \mathbb{O})$

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

where  $\bar{\gamma}_A: \mathcal{P}(\mathcal{C} / \equiv_C) \rightarrow \mathcal{A} \rightarrow (\mathcal{E} \rightarrow \mathbb{O})$ :

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

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

and  $\gamma_F: \mathcal{D} \rightarrow \mathcal{W} \rightarrow (\mathcal{E} \rightarrow \mathbb{O})$ :

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

$$\gamma_F[D]\left(\sum_i \omega^i \cdot f_i\right) \stackrel{\text{def}}{=} \lambda \rho \in \gamma_D(D): \sum_i \omega^i \cdot f_i(\dots, \rho(X_i), \dots)$$

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

## Piecewise-Defined Ranking Functions Abstract Domain

- $\mathcal{A} \stackrel{\text{def}}{=} \{\text{LEAF}: f \mid f \in \mathcal{F}\} \cup \{\text{NODE}\{c\}: t_1; t_2 \mid c \in \mathcal{C} \wedge t_1, t_2 \in \mathcal{A}\}$
- **concretization function**  $\gamma_A: \mathcal{A} \rightarrow (\mathcal{E} \rightarrow \mathbb{O})$ :

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

where  $\bar{\gamma}_A: \mathcal{P}(\mathcal{C} / \equiv_C) \rightarrow \mathcal{A} \rightarrow (\mathcal{E} \rightarrow \mathbb{O})$ :

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

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

and  $\gamma_F: \mathcal{D} \rightarrow \mathcal{F} \rightarrow (\mathcal{E} \rightarrow \mathbb{O})$ :

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

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

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

# Piecewise-Defined Ranking Functions Abstract Domain

## Abstract Domain Operators

- They manipulate elements in  $\mathcal{A}_{NIL} \stackrel{\text{def}}{=} \{NIL\} \cup \mathcal{A}$
- The **binary operators** rely on a tree unification algorithm
  - approximation order  $\preceq_A$  and computational order  $\sqsubseteq_A$
  - approximation join  $\vee_A$  and computational join  $\sqcup_A$
  - meet  $\wedge_A$
  - widening  $\nabla_A$
- 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

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. 
$$\text{NIL } \gamma_A t \stackrel{\text{def}}{=} t$$
  

$$t \gamma_A \text{NIL} \stackrel{\text{def}}{=} t$$
4. Join the leaf nodes using the **approximation join**  $\gamma_F[\alpha_C(C)]$  or the **computational join**  $\sqcup_F[\alpha_C(C)]$

### Piecewise-Defined Ranking Functions Abstract Domain

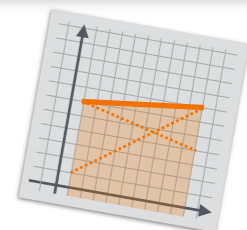
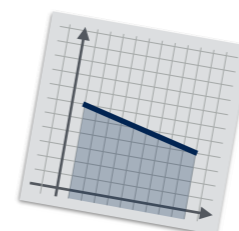
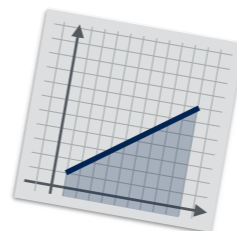
#### Join (continue)

- **approximation join**  $\gamma_F[D]$ , where  $D$  is a set of linear constraints
- between defined leaf nodes:

$$f_1 \gamma_F[D] f_2 \stackrel{\text{def}}{=} \begin{cases} f & f \in \mathcal{F} \setminus \{f_1, f_2\} \\ T_F & \text{otherwise} \end{cases}$$

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

Example:



Operators on polyhedra: join

Join:  $\lambda^{\sharp} \cup^{\sharp} \gamma^{\sharp} \stackrel{\text{def}}{=} [[P_{\lambda^{\sharp}}, P_{\gamma^{\sharp}}], [R_{\lambda^{\sharp}}, R_{\gamma^{\sharp}}]]$  (join generator sets)

Examples:

two polytopes

a point and a line

$\cup^{\sharp}$  is **optimal**:  
we get the topological closure of the convex hull of  $\gamma(\lambda^{\sharp}) \cup \gamma(\gamma^{\sharp})$ .

Course 4 Relational Numerical Abstract Domains Antoine Miné p. 30 / 70

# Piecewise-Defined Ranking Functions Abstract Domain

## Join (continue)

- **approximation join**  $\gamma_W [D]$ , where  $D \in \mathcal{D}$ :
- between defined leaf nodes:

approximation join  $\gamma_F [D]$  in ascending powers of  $\omega$

Example:

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

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

# Piecewise-Defined Ranking Functions Abstract Domain

## Join (continue)

- **approximation join**  $\gamma_W [D]$ , where  $D \in \mathcal{D}$ :
  - between defined leaf nodes:

approximation join  $\gamma_F [D]$  in ascending powers of  $\omega$

Example:

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

# Piecewise-Defined Ranking Functions Abstract Domain

## Join (continue)

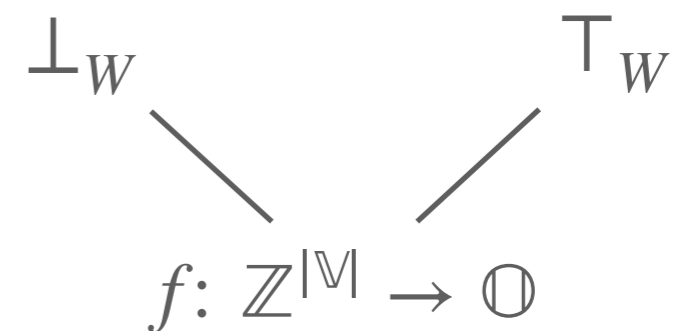
- **approximation join**  $\gamma_W [D]$ , where  $D \in \mathcal{D}$ :

- between defined leaf nodes:

approximation join  $\gamma_F [D]$  in ascending powers of  $\omega$

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

$$\begin{array}{ll}
 \perp_W \gamma_W [D] f \stackrel{\text{def}}{=} \perp_W & f \in \mathcal{W} \setminus \{ \top_W \} \\
 f \gamma_W [D] \perp_W \stackrel{\text{def}}{=} \perp_W & f \in \mathcal{W} \setminus \{ \top_W \} \\
 \top_W \gamma_W [D] f \stackrel{\text{def}}{=} \top_W & f \in \mathcal{W} \setminus \{ \perp_W \} \\
 f \gamma_W [D] \top_W \stackrel{\text{def}}{=} \top_W & f \in \mathcal{W} \setminus \{ \perp_W \}
 \end{array}$$



# Piecewise-Defined Ranking Functions Abstract Domain

## Join (continue)

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

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

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

$$\begin{array}{l} \perp_W \sqcup_W [D] f \stackrel{\text{def}}{=} f \quad f \in \mathcal{W} \\ f \sqcup_W [D] \perp_W \stackrel{\text{def}}{=} f \quad f \in \mathcal{W} \\ \top_W \sqcup_W [D] f \stackrel{\text{def}}{=} \top_W \quad f \in \mathcal{W} \\ f \sqcup_W [D] \top_W \stackrel{\text{def}}{=} \top_W \quad f \in \mathcal{W} \end{array}$$

$$\begin{array}{c} \top_W \\ | \\ f: \mathbb{Z}^{|\mathcal{V}|} \rightarrow \mathbb{O} \\ | \\ \perp_W \end{array}$$

# Piecewise-Defined Ranking Functions Abstract Domain

## Widening

### Piecewise-Defined Ranking Functions Abstract Domain

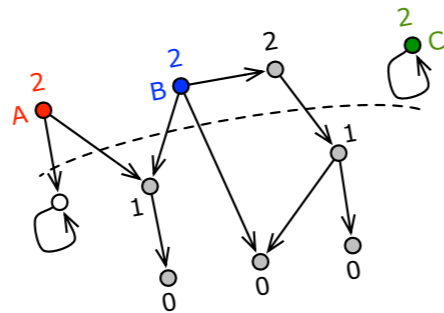
#### Widening (continue)

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**



Caterina Urban

Termination Analysis

Lesson 5

### Piecewise-Defined Ranking Functions Abstract Domain

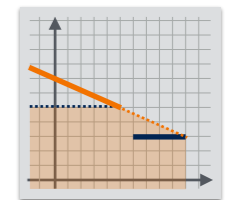
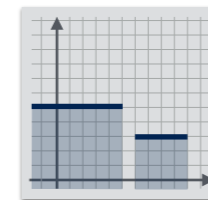
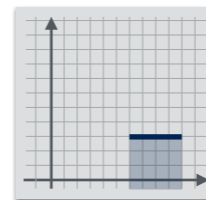
#### Widening (continue)

#### 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**  
 $\nabla_F [\alpha_C(\bar{C}), \alpha_C(C)]$ , where  $\bar{C}$  is the set of constraints along the path to  $\bar{f}$

Example:



Lesson 5

Termination Analysis

Caterina Urban

73



# 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**  
 $\nabla_F [\alpha_C(\bar{C}), \alpha_C(C)]$ , where  $\bar{C}$  is the set of constraints along the path to  $\bar{f}$ ,  
**in ascending powers of  $\omega$**

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

# Piecewise-Defined Ranking Functions Abstract Domain

## Assignments

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

## Piecewise-Defined Ranking Functions Abstract Domain

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

### Assignments

- Base case (  $f$  )

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

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

where  $\tilde{f}(\dots, X_i, X, \dots) \stackrel{\text{def}}{=} \max\{f(\dots, \rho(X_i), v, \dots) + 1 \mid \rho \in \gamma_D(R) \wedge v \in E[[e]]\rho\}$   
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  $\max(3,4) = 4$ )

Caterina Urban

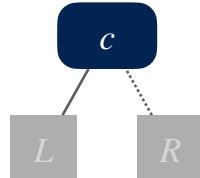
Termination Analysis

Lesson 5

## 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  $I = \emptyset \wedge \perp_C \in J$



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



case ④

- perform **tree pruning** on  and 
- join the results with  $Y_A$

Lesson 5

Termination Analysis

Caterina Urban

79

# Piecewise-Defined Ranking Functions Abstract Domain

Assignments (continue)

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

- Base case (  $f$  )

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

Example:

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

$$\overleftarrow{\text{ASSIGN}}_W[[x_1 \leftarrow [-\infty, +\infty]]][\top_D] \equiv \omega^2 \cdot 1 + \omega \cdot 0 + x_2 + 1$$

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

# Abstract Definite Termination Semantics

## Abstract Definite Termination Semantics

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

- $\mathcal{R}_M^{\#}[[X \leftarrow e]]t \stackrel{\text{def}}{=} \text{ASSIGN}_A[X \leftarrow e]t$
- $\mathcal{R}_M^{\#}[[\text{if } e \bowtie 0 \text{ then } s]]t \stackrel{\text{def}}{=} \text{FILTER}_A[e \bowtie 0](\mathcal{R}_M^{\#}[[s]]t) \vee_T \text{FILTER}_A[e \not\bowtie 0]t$
- $\mathcal{R}_M^{\#}[[\text{while } e \bowtie 0 \text{ do } s \text{ done}]]t \stackrel{\text{def}}{=} \text{lfp}^{\#} \bar{F}_M^{\#}$   
 where  $\bar{F}_M^{\#}(x) \stackrel{\text{def}}{=} \text{FILTER}_A[e \bowtie 0](\mathcal{R}_M^{\#}[[s]]x) \vee_T \text{FILTER}_A[e \not\bowtie 0](t)$
- $\mathcal{R}_M^{\#}[[s_1; s_2]]t \stackrel{\text{def}}{=} \mathcal{R}_M^{\#}[[s_1]](\mathcal{R}_M^{\#}[[s_2]]t)$

Language syntax

```

statℓ ::= X ← expℓ           (assignment)
        | if expℓ ⋈ 0 then statℓ (conditional)
        | while expℓ ⋈ 0 do statℓ doneℓ (loop)
        | statℓ; statℓ         (sequence)
        | X                       (variable)
        | -exp                     (negation)
        | exp ⋄ exp                 (binary operation)
        | c                         (constant c ∈ Z)
        | [c, c']                   (random input, c, c' ∈ Z ∪ {±∞})
    
```

Simple structured, numeric language

- $X \in \mathcal{V}$ , where  $\mathcal{V}$  is a finite set of **program variables**
- $\ell \in \mathcal{L}$ , where  $\mathcal{L}$  is a finite set of **control points**
- numeric expressions:  $\bowtie \in \{=, \leq, \dots\}$ ,  $\diamond \in \{+, -, \times, /\}$
- **random inputs**:  $X \leftarrow [c, c']$   
 model environment, parametric programs, unknown functions, ...

## Abstract Definite Termination Semantics

### Definition

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

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

where  $\mathcal{R}_M^{\#}[[stat]]: \mathcal{A} \rightarrow \mathcal{A}$  is the abstract definite termination semantics of each program instruction  $stat$

### Theorem (Soundness)

$$\mathcal{R}_M^{\#}[[stat^{\ell}]] \preceq \gamma_A(\mathcal{R}_M^{\#}[[stat]])$$

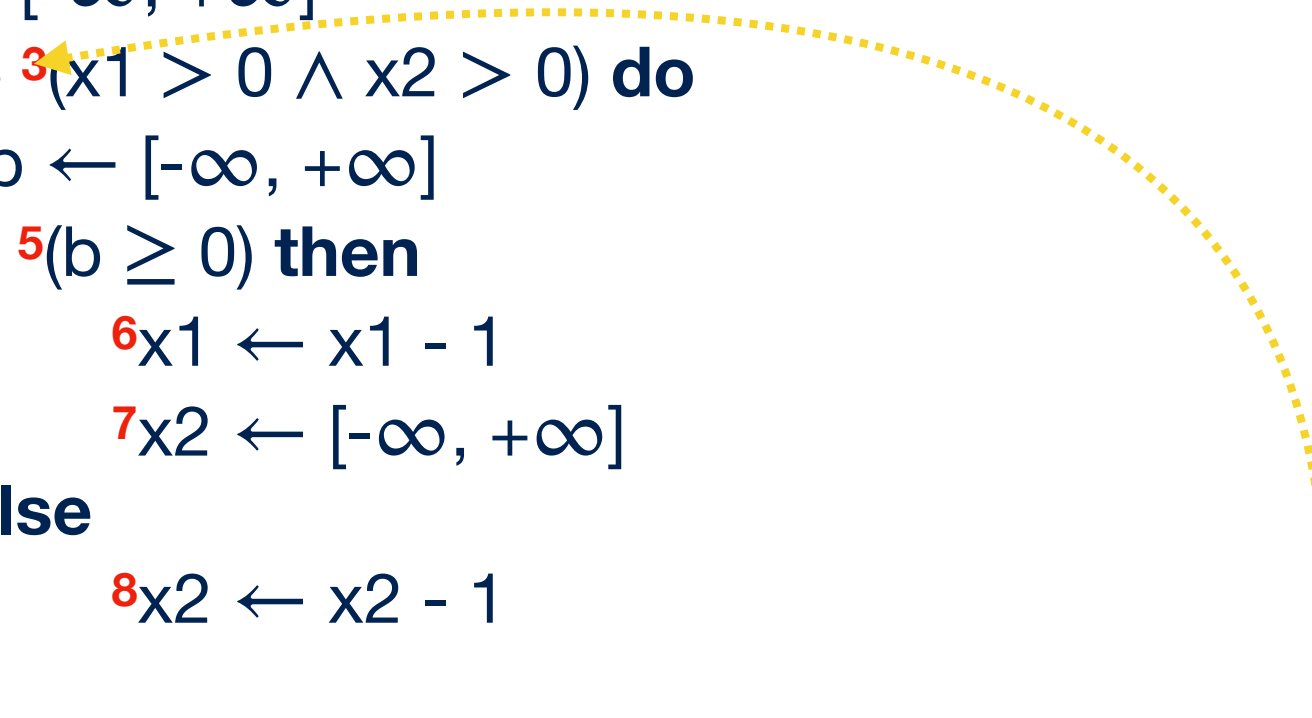
### Corollary (Soundness)

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

# Abstract Definite Termination Semantics

## Example

```
1  $x_1 \leftarrow [-\infty, +\infty]$ 
2  $x_2 \leftarrow [-\infty, +\infty]$ 
while 3  $(x_1 > 0 \wedge x_2 > 0)$  do
  4  $b \leftarrow [-\infty, +\infty]$ 
  if 5  $(b \geq 0)$  then
    6  $x_1 \leftarrow x_1 - 1$ 
    7  $x_2 \leftarrow [-\infty, +\infty]$ 
  else
    8  $x_2 \leftarrow x_2 - 1$ 
od 9
```



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

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 bdeee1 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

# 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**