Abstract Interpretation and Applications

Professor Ing. Dr.-Ing. Dr. **Dr.-Ing. E.h.**

Patrick COUSOT

École Normale Supérieure, Département d'Informatique

45 rue d'Ulm, 75230 Paris cedex 05, France

`cousot@ens.fr` `http://www.di.ens.fr/~cousot`

Naturwiss.-Techn. Fakultät I, Universität des Saarlandes

June 29, 2001

■ ◀ ▶ ▷

Ich fühle mich zutiefst geehrt, die mir zugeteilte Ehrendoktorwürde entgegen zu nehmen.

Content

Introduction

Software Costs

- The cost of software is:
  - huge (e.g. 5 to 15 % of the cost of a plane),
  - increasing rapidly with the size of software (frequently 1 up to 40 000 000 lines!);

Software Costs

- The cost of software is:
  - huge (e.g. 5 to 15 % of the cost of a plane),
  - increasing rapidly with the size of software (frequently 1 up to 40 000 000 lines!);
- How to cut down costs and enhance software quality?
  - ...
  - Automate the reasonings about software (the early idea of using computers to reason about computers);
  - ...

Reasoning About Programs

We must be able to reason about programs:

- to design programs;
  - manually: e.g. coding,
  - automatically: e.g. program generation;

Reasoning About Programs

We must be able to reason about programs:

- to design programs;
  - manually: e.g. coding,
  - automatically: e.g. program generation;
- to manipulate programs:
  - manually: e.g. modification of a reused program,
  - automatically: e.g. compilation;

Reasoning About Programs

We must be able to reason about programs:

- to design programs;
  - manually: e.g. coding,
  - automatically: e.g. program generation;
- to manipulate programs:
  - manually: e.g. modification of a reused program,
  - automatically: e.g. compilation;
- to check program correctness:
  - manually: e.g. debuggers,
  - automatically: e.g. analyzers, provers.

Basis for Reasoning about Programs: Semantics

- The semantics of a computer system is the description of the behavior of this computer system when running in interaction with its environment.

Undecidability

- All questions about the semantics of a program are undecidable

Undecidability

- All (interesting) questions about the semantics of a program (written in a non trivial computer language) are undecidable (i.e. cannot be always and fully automatically answered with a computer in finite time);

Undecidability

- All (interesting) questions about the semantics of a program (written in a non trivial computer language) are undecidable (i.e. cannot be always and fully automatically answered with a computer in finite time);
- Examples of undecidable questions:
  - Is my program bug-free? (i.e. correct with respect to a given specification);
  - Can a program variable take two different values during execution?

Coping With Undecidable Questions on the Semantics

- Consider simple specifications or programs (hopeless);

Coping With Undecidable Questions on the Semantics

- Consider simple specifications or programs (hopeless);
- Consider decidable questions only or semi-algorithms (e.g. model-checking);

Coping With Undecidable Questions on the Semantics

- Consider simple specifications or programs (hopeless);
- Consider decidable questions only or semi-algorithms (e.g. model-checking);
- Ask the programmer to help (e.g. theorem proving);

Coping With Undecidable Questions on the Semantics

- Consider simple specifications or programs (hopeless);
- Consider decidable questions only or semi-algorithms (e.g. model-checking);
- Ask the programmer to help (e.g. theorem proving);
- Consider approximations to handle practical complexity limitations (the whole purpose of abstract interpretation).

Semantics

Semantics: intuition

- The semantics of a language defines the semantics of any program written in this language;

Semantics: intuition

- The semantics of a language defines the semantics of any program written in this language;
- The semantics of a program provides a formal mathematical model of all possible behaviors of a computer system executing this program (interacting with any possible environment);

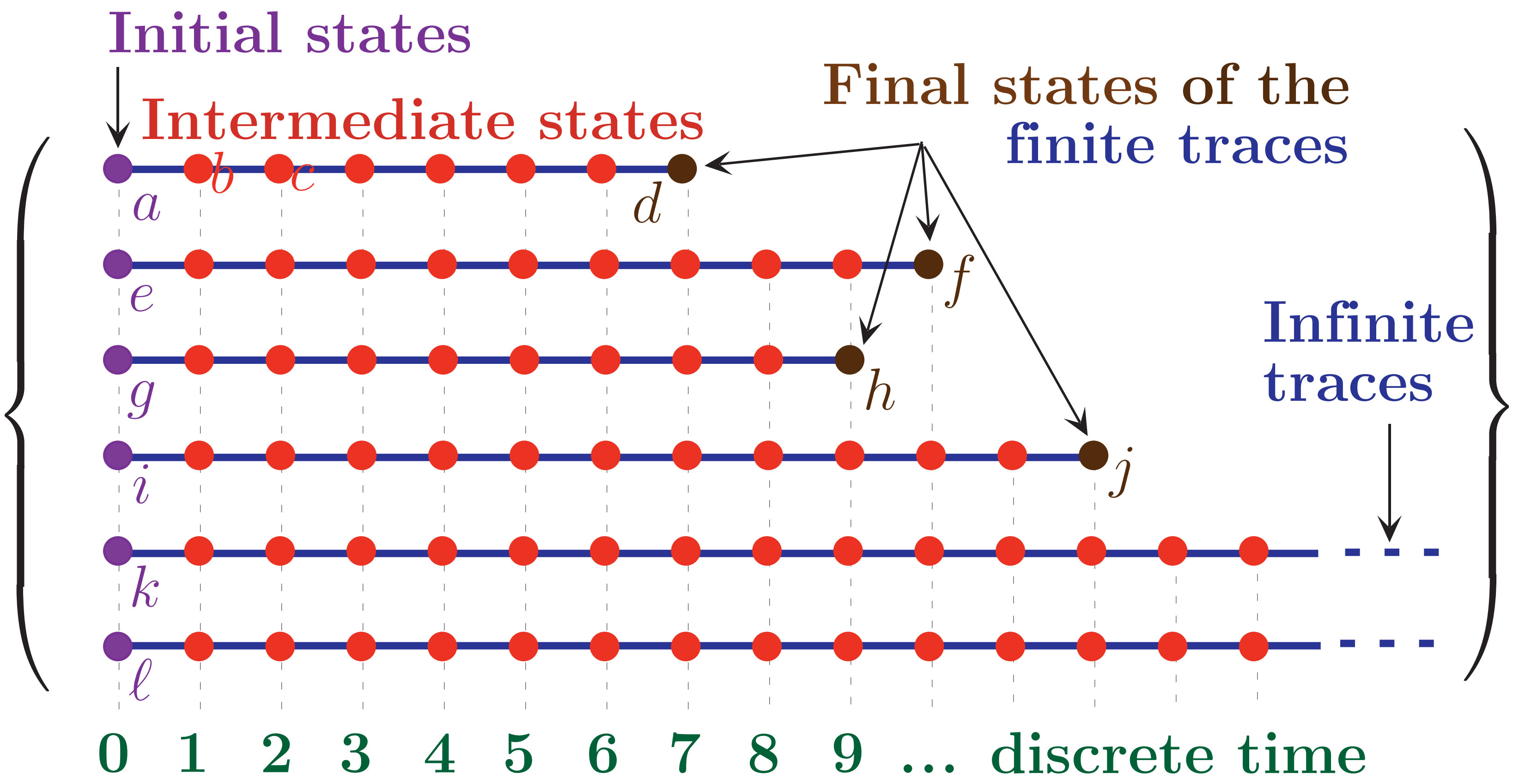
Semantics: intuition

- The semantics of a language defines the semantics of any program written in this language;
- The semantics of a program provides a formal mathematical model of all possible behaviors of a computer system executing this program (interacting with any possible environment);
- Any semantics of a program can be defined as the solution of a fixpoint equation;

Semantics: intuition

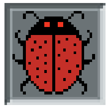
- The semantics of a language defines the semantics of any program written in this language;
- The semantics of a program provides a formal mathematical model of all possible behaviors of a computer system executing this program (interacting with any possible environment);
- Any semantics of a program can be defined as the solution of a fixpoint equation;
- All semantics of a program can be organized in a hierarchy by abstraction.

Example: Trace semantics

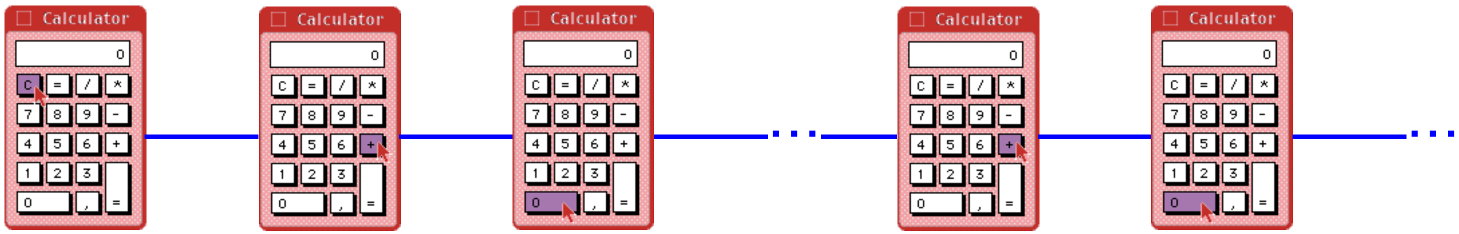


Examples of computation traces

- Finite (C1+1=):

- Erroneous  (C1+1+1+1...):

- Infinite (C+0+0+0...):



Fixpoints: intuition

Behaviors $=$

Fixpoints: intuition

$$\text{Behaviors} = \{\bullet \mid \bullet \text{ is a final state}\}$$

Fixpoints: intuition

$$\begin{aligned}\text{Behaviors} = \; & \{\bullet \mid \bullet \text{ is a final state}\} \\ \cup \; & \{\bullet\!\!-\!\!\!-\!\!\!-\!\!\bullet\!\!-\!\!\!-\!\!\!-\ldots-\!\!\!-\!\!\!-\!\!\bullet \mid \bullet\!\!-\!\!\!-\!\!\!-\!\!\bullet \text{ is an elementary step } \& \\ & \qquad \bullet\!\!-\!\!\!-\!\!\!-\ldots-\!\!\!-\!\!\!-\!\!\bullet \in \text{Behaviors}^{+}\}\end{aligned}$$

Fixpoints: intuition

$$\begin{aligned}
\text{Behaviors} = \; & \{\bullet \mid \bullet \text{ is a final state}\} \\
\cup \; & \{\bullet\!\!-\!\!-\!\!-\!\!\bullet\!\!-\!\!-\!\!-\!\!\ldots\!\!-\!\!-\!\!-\!\!\bullet \mid \bullet\!\!-\!\!-\!\!-\!\!\bullet \text{ is an elementary step } \& \\
& \quad \bullet\!\!-\!\!-\!\!-\!\!\ldots\!\!-\!\!-\!\!-\!\!\bullet \in \text{Behaviors}^{+}\} \\
\cup \; & \{\bullet\!\!-\!\!-\!\!-\!\!\bullet\!\!-\!\!-\!\!-\!\!\ldots\!\!-\!\!-\!\!-\!\!\ldots \mid \bullet\!\!-\!\!-\!\!-\!\!\bullet \text{ is an elementary step } \& \\
& \quad \bullet\!\!-\!\!-\!\!-\!\!\ldots\!\!-\!\!-\!\!-\!\!\ldots \in \text{Behaviors}^{\infty}\}
\end{aligned}$$

Fixpoints: intuition

$$
\begin{aligned}
\mathbf{Behaviors} = \;& \{\bullet \mid \bullet \text{ is a final state}\} \\
\cup \;& \{\bullet\!-\!\!-\!\!-\bullet\!-\!\!-\!\!-\ldots-\!\!-\!\!-\bullet \mid \bullet\!-\!\!-\!\!-\bullet \text{ is an elementary step } \& \\
& \qquad \bullet\!-\!\!-\!\!-\ldots-\!\!-\!\!-\bullet \in \mathbf{Behaviors}^{+}\} \\
\cup \;& \{\bullet\!-\!\!-\!\!-\bullet\!-\!\!-\!\!-\ldots-\!\!-\!\!-\ldots \mid \bullet\!-\!\!-\!\!-\bullet \text{ is an elementary step } \& \\
& \qquad \bullet\!-\!\!-\!\!-\ldots-\!\!-\!\!-\ldots \in \mathbf{Behaviors}^{\infty}\}
\end{aligned}
$$

- In general, the equation has multiple solutions.

Least Fixpoints: Intuition

$$
\begin{aligned}
\text{Behaviors} = \; & \{\bullet \mid \bullet \text{ is a final state}\} \\
\cup \; & \{\bullet\!\longrightarrow\!\bullet\!\longrightarrow \ldots \longrightarrow\!\bullet \mid \bullet\!\longrightarrow\!\bullet \text{ is an elementary step } \& \\
& \qquad \bullet\!\longrightarrow \ldots \longrightarrow\!\bullet \in \text{Behaviors}^{+}\} \\
\cup \; & \{\bullet\!\longrightarrow\!\bullet\!\longrightarrow \ldots \longrightarrow \ldots \mid \bullet\!\longrightarrow\!\bullet \text{ is an elementary step } \& \\
& \qquad \bullet\!\longrightarrow \ldots \longrightarrow \ldots \in \text{Behaviors}^{\infty}\}
\end{aligned}
$$

- In general, the equation has multiple solutions.
- Choose the least one for the partial ordering:

« *more finite traces & less infinite traces* ».

Abstract Interpretation

The Theory of Abstract Interpretation

- **Abstract interpretation** is a theory of conservative approximation of the semantics of computer systems.

The Theory of Abstract Interpretation

- **Abstract interpretation** is a theory of conservative approximation of the semantics of computer systems.

  Approximation: observation of the behavior of a computer system at some level of abstraction, ignoring irrelevant details;

The Theory of Abstract Interpretation

- **Abstract interpretation** is a theory of conservative approximation of the semantics of computer systems.

  Approximation: observation of the behavior of a computer system at some level of abstraction, ignoring irrelevant details;

  Conservative: the approximation cannot lead to any erroneous conclusion.

Usefulness of Abstract Interpretation

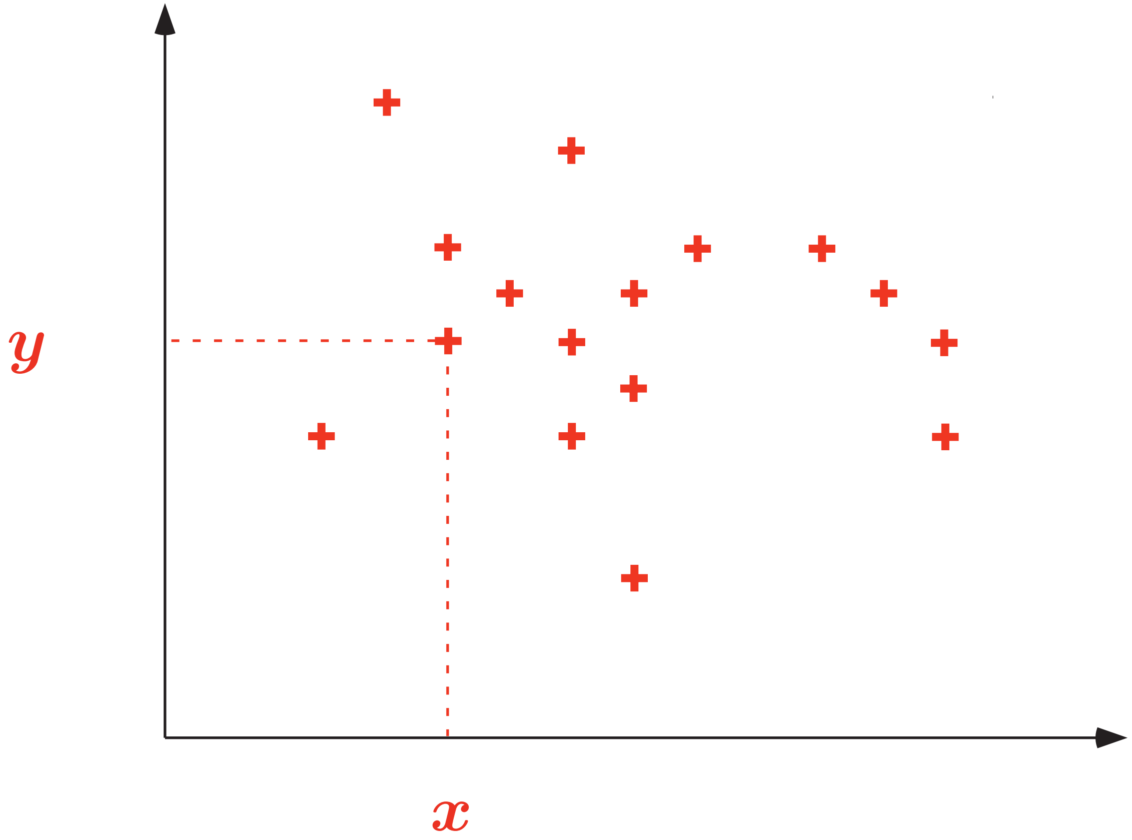
- **Thinking tools**: the idea of abstraction is central to reasoning (in particular on computer systems);

Usefulness of Abstract Interpretation

- **Thinking tools**: the idea of abstraction is central to reasoning (in particular on computer systems);
- **Mechanical tools**: the idea of effective approximation leads to automatic semantics-based program manipulation tools.

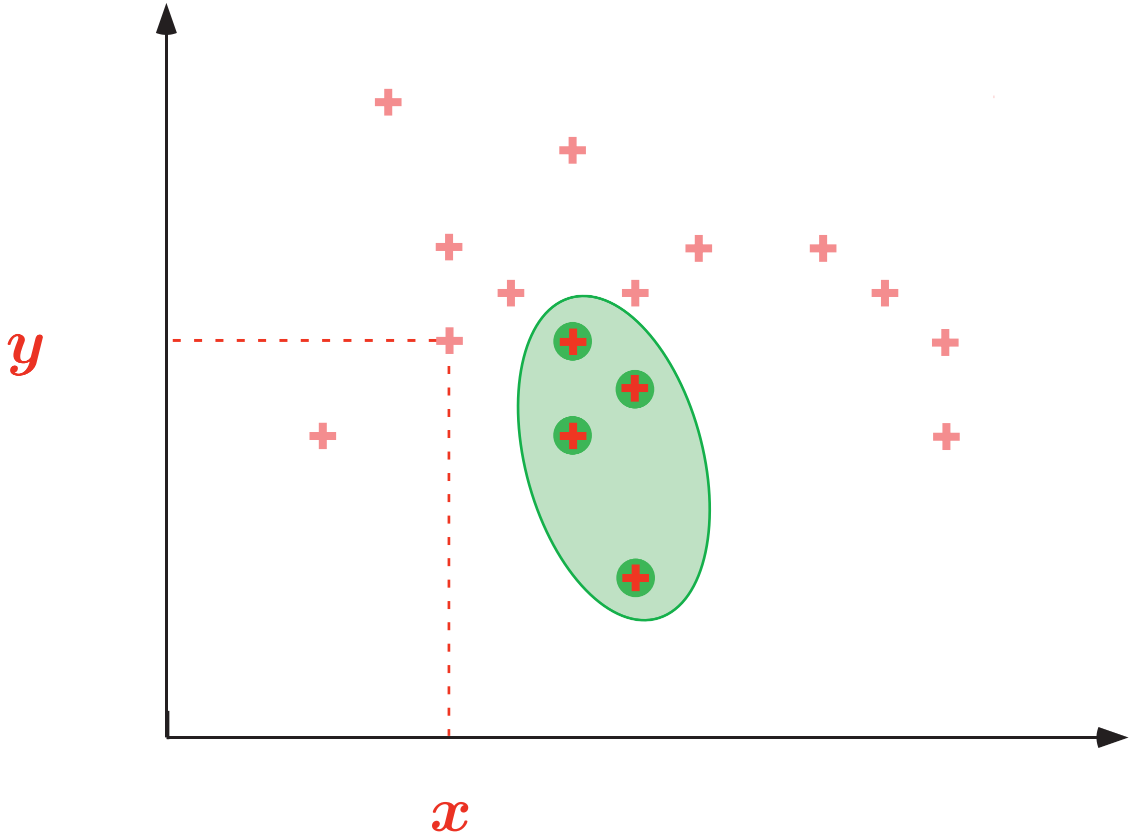
Intuition behind abstraction

Approximations of an [in]finite set of points;



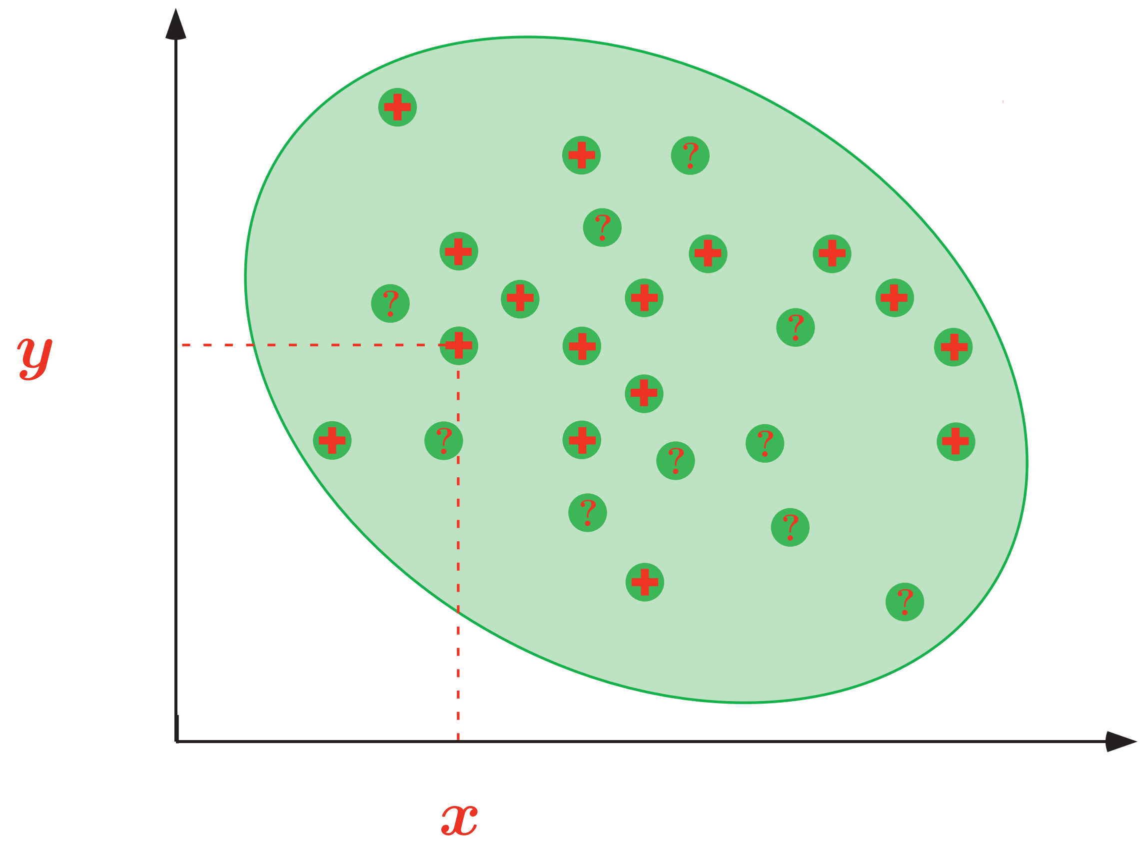
$$\{\ldots,\langle 19,\ 78\rangle,\ldots,\\ \langle 20,\ 01\rangle,\ldots\}$$

Approximations of an [in]finite set of points: From Below



$$\{\ldots, \langle 19,\ 78\rangle, \ldots,\ \ldots\}$$

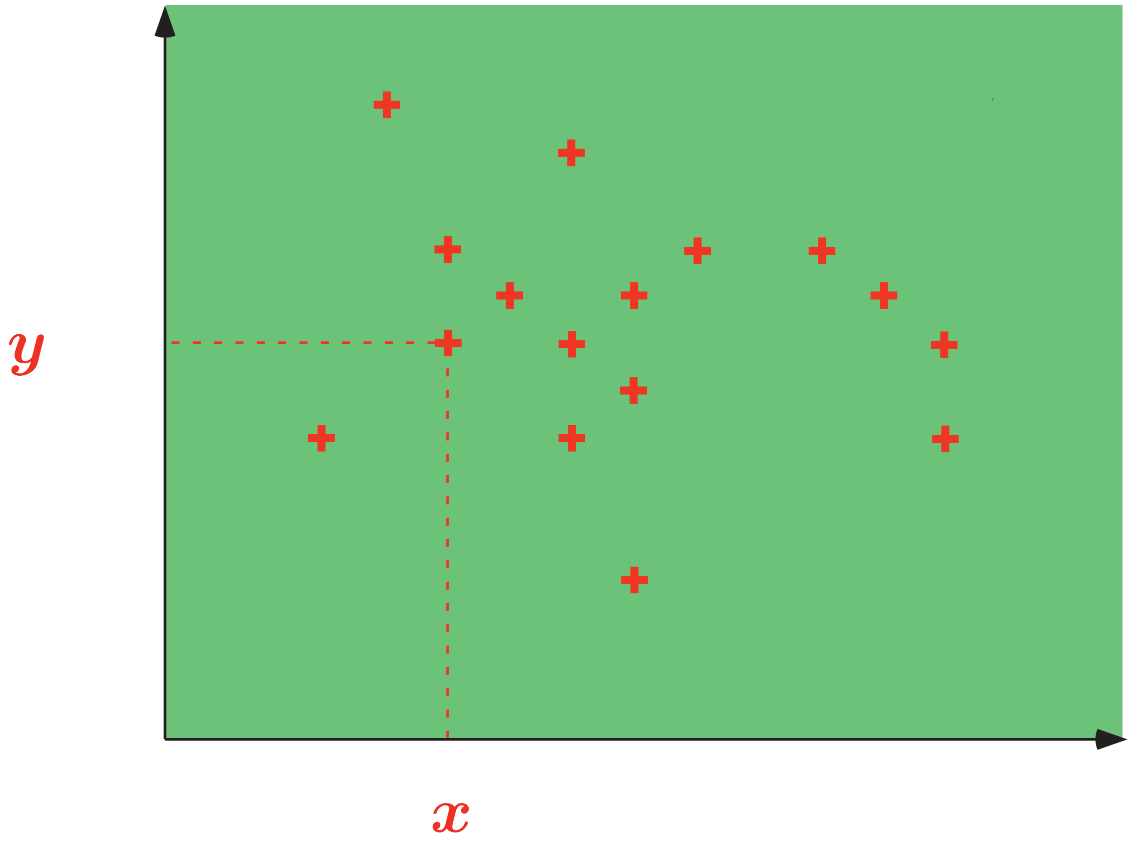
Approximations of an [in]finite set of points: From Above



$$\{\ldots, \langle 19,\ 78\rangle, \ldots,$$
$$\langle 20,\ 01\rangle, \langle ?,\ ?\rangle, \ldots\}$$

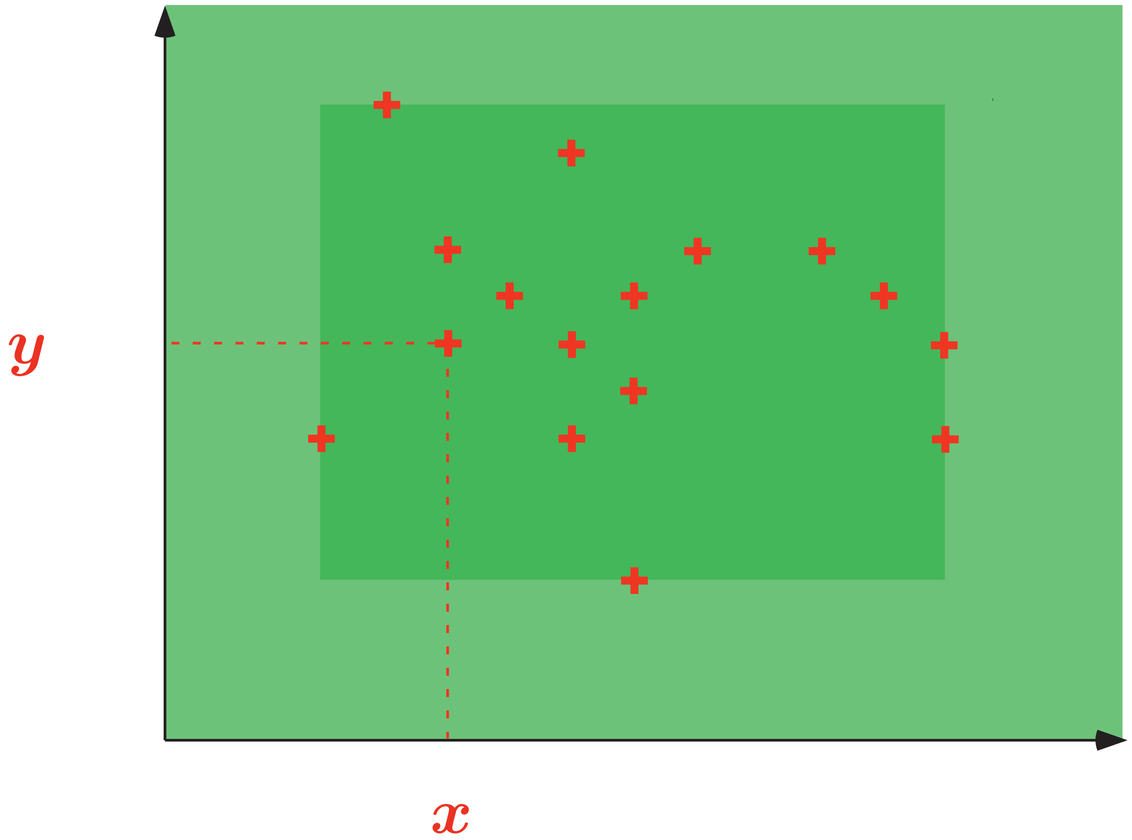
Intuition Behind Effective Computable Abstraction

Effective computable approximations of an [in]finite set of points; Signs



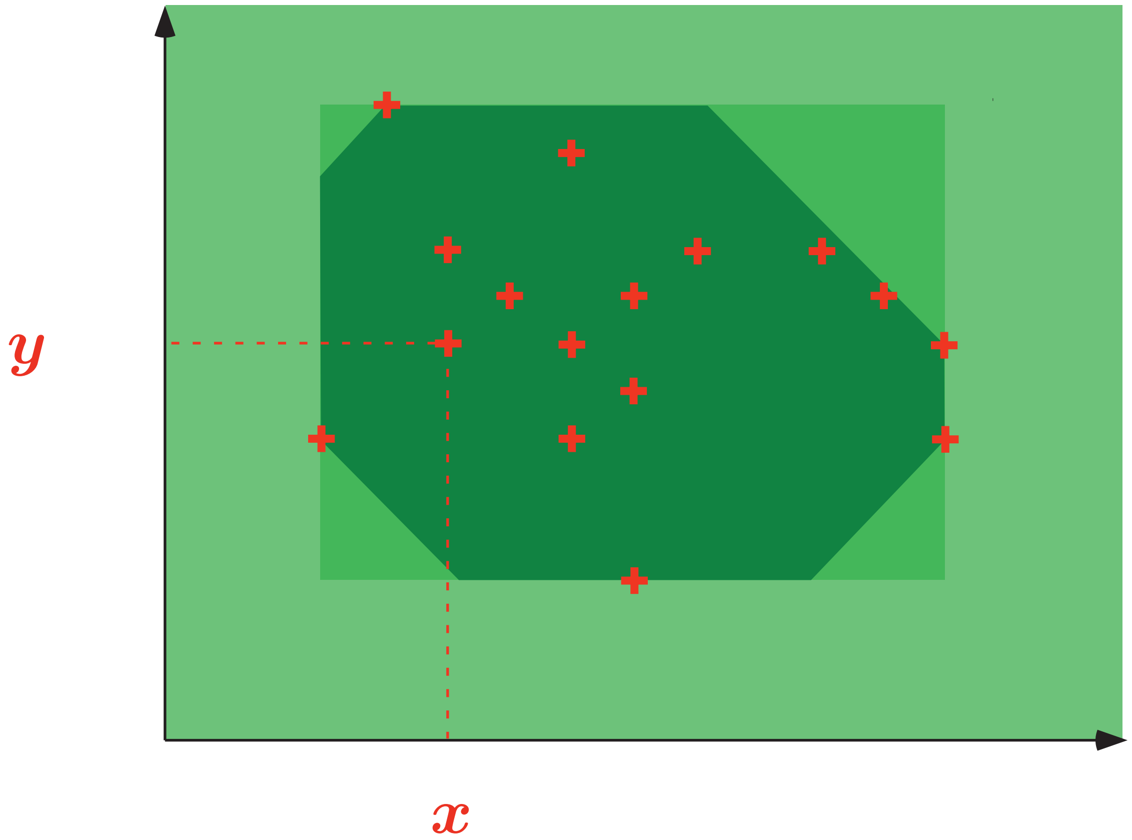
$$\begin{cases} x \geq 0 \\ y \geq 0 \end{cases}$$

Effective computable approximations of an [in]finite set of points; Intervals



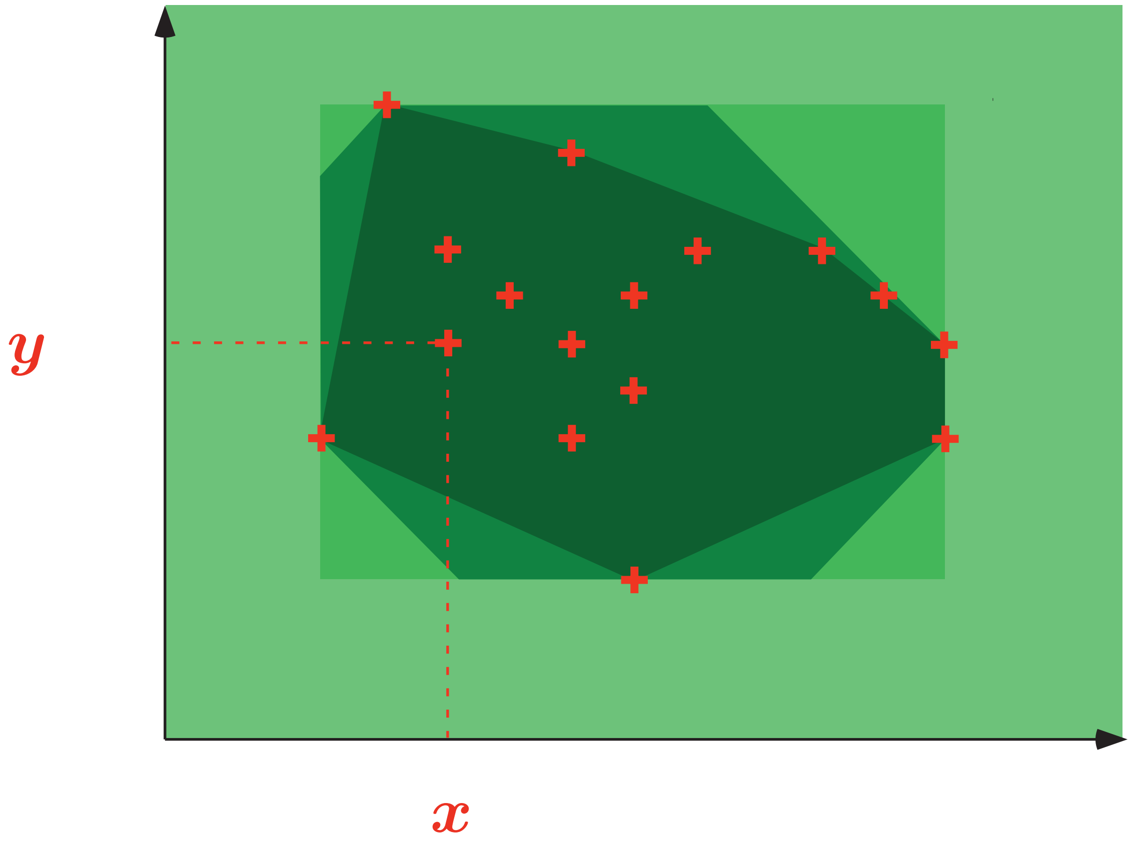
$$\begin{cases} x \in [19,\ 78] \\ y \in [20,\ 01] \end{cases}$$

Effective computable approximations of an [in]finite set of points; Octagons



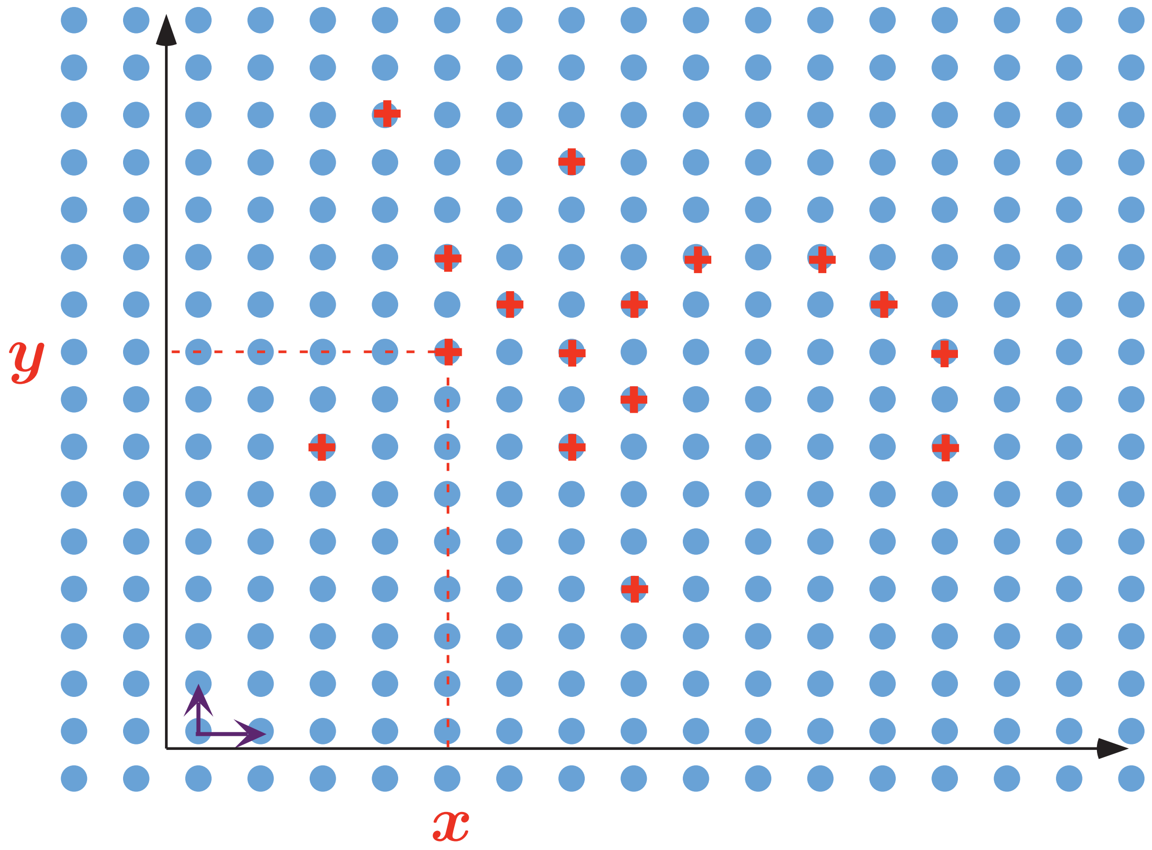
$$\begin{cases} 1 \leq x \leq 9 \\ x + y \leq 78 \\ 1 \leq y \leq 9 \\ x - y \leq 99 \end{cases}$$

Effective computable approximations of an [in]finite set of points; Polyhedra



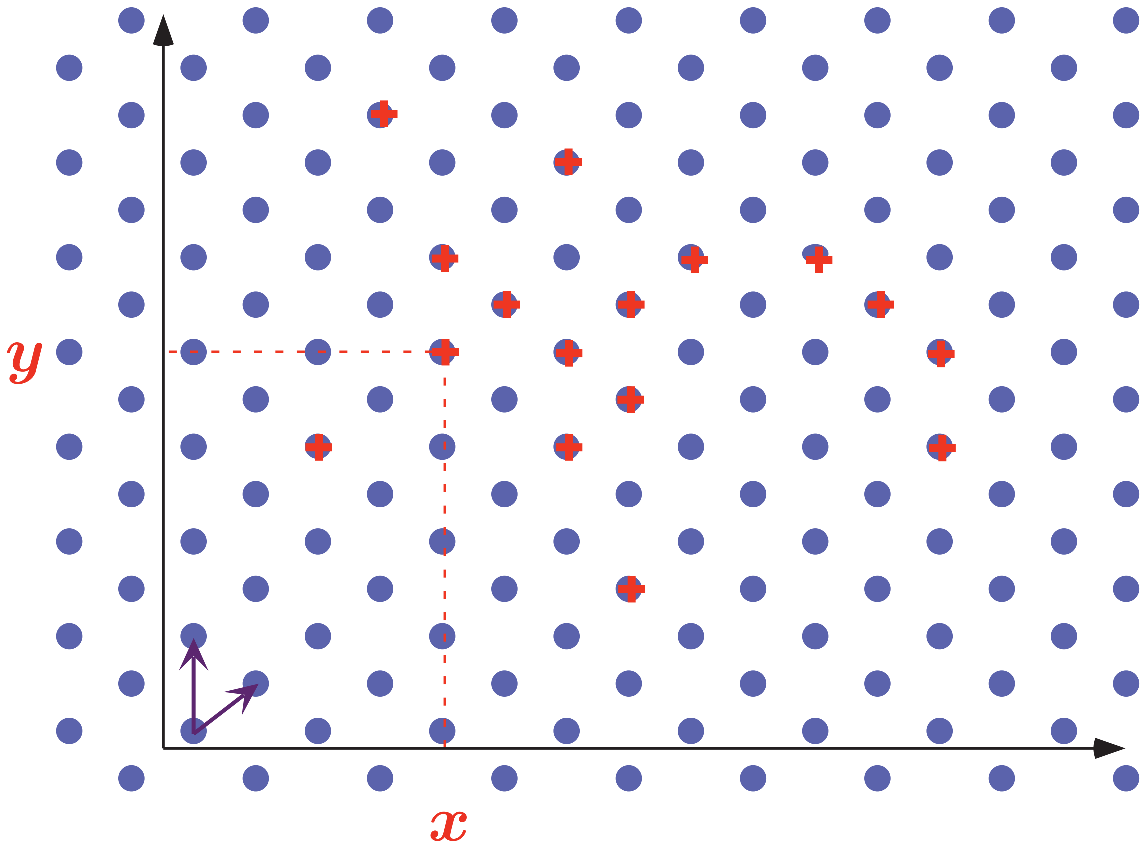
$$\begin{cases} 19x + 78y \le 2000 \\ 20x + 01y \ge 0 \end{cases}$$

Effective computable approximations of an [in]finite set of points; Simple congruences



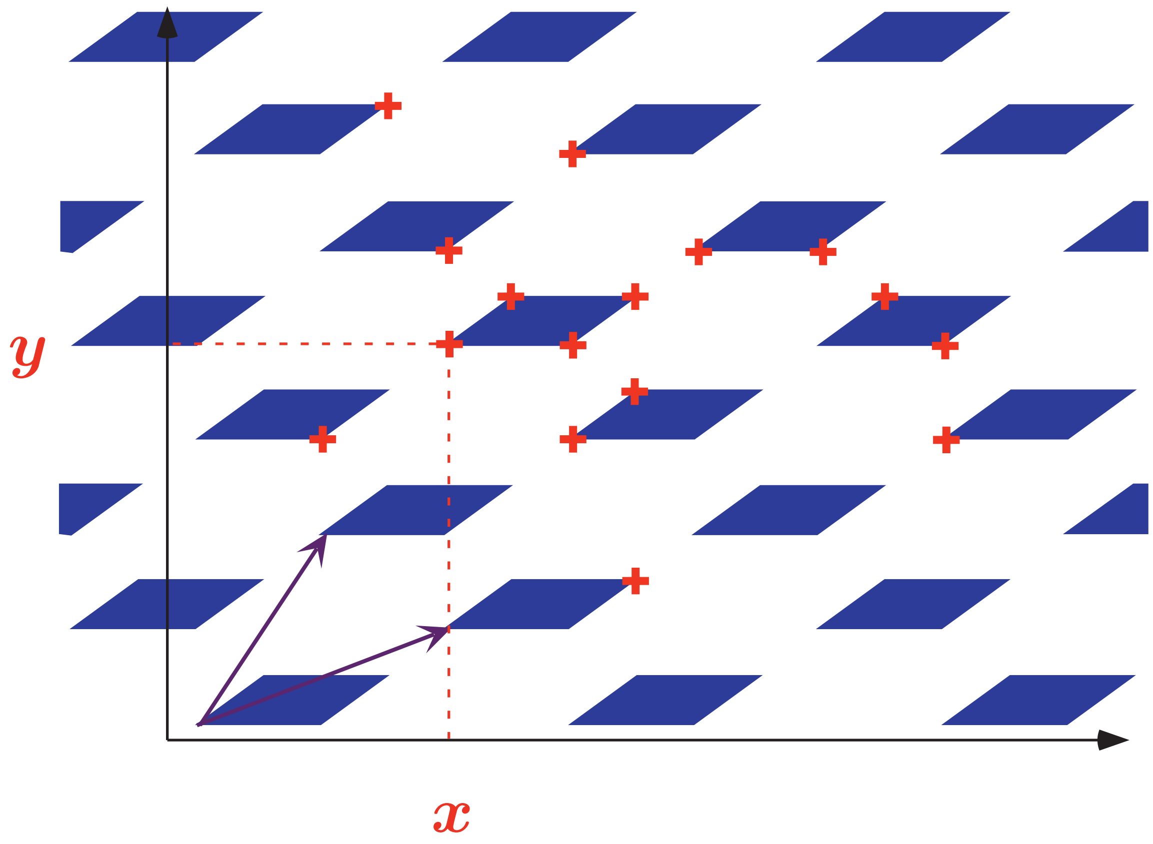
$$\begin{cases} x = 19 \bmod 78 \\ y = 20 \bmod 99 \end{cases}$$

Effective computable approximations of an [in]finite set of points; Linear congruences



$$\begin{cases} 1x + 9y = 7 \bmod 8 \\ 2x - 1y = 9 \bmod 9 \end{cases}$$

Effective computable approximations of an [in]finite set of points; Trapezoidal linear congruences



$$\begin{cases} 1x + 9y \in [0, 78] \bmod 10 \\ 2x - 1y \in [0, 99] \bmod 11 \end{cases}$$

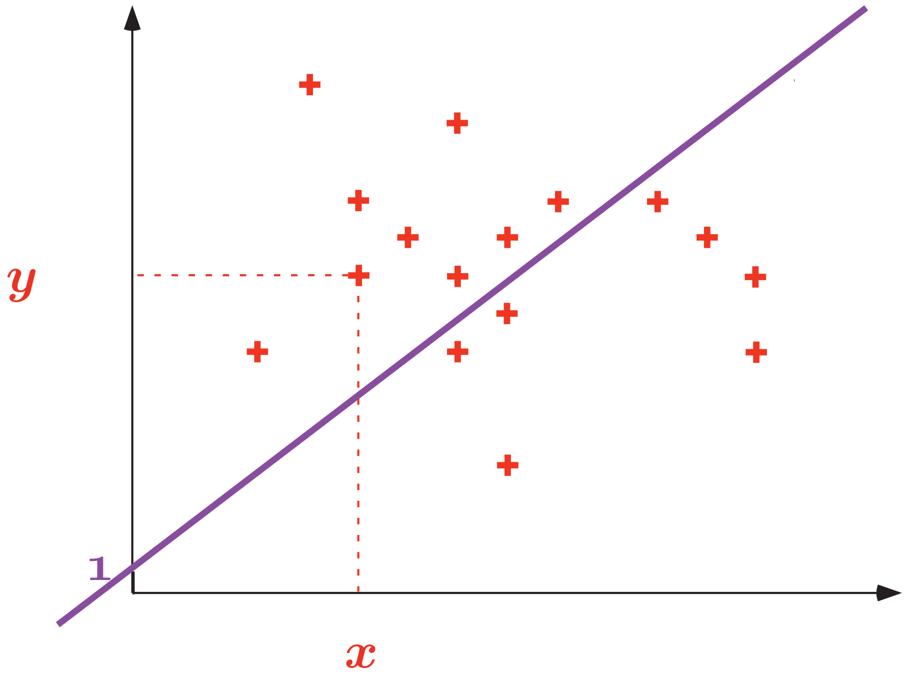
Intuition Behind Sound/Conservative Approximation

Conservative Approximation

- Is the operation `1/(x+1-y)` well defined at run-time?

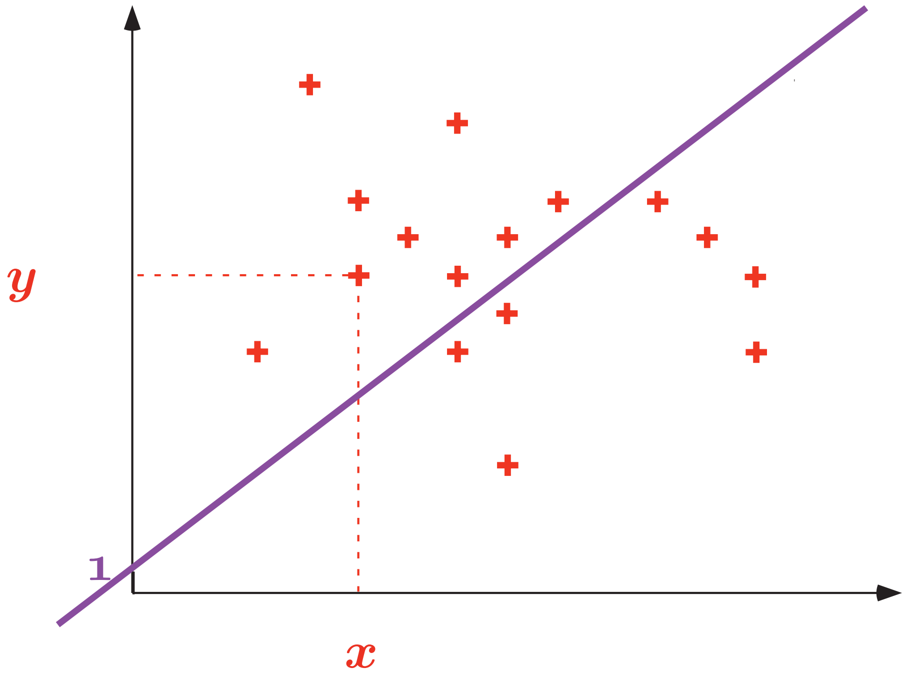
Conservative Approximation

- Is the operation `1/(x+1-y)` well defined at run-time?
- Concrete semantics:



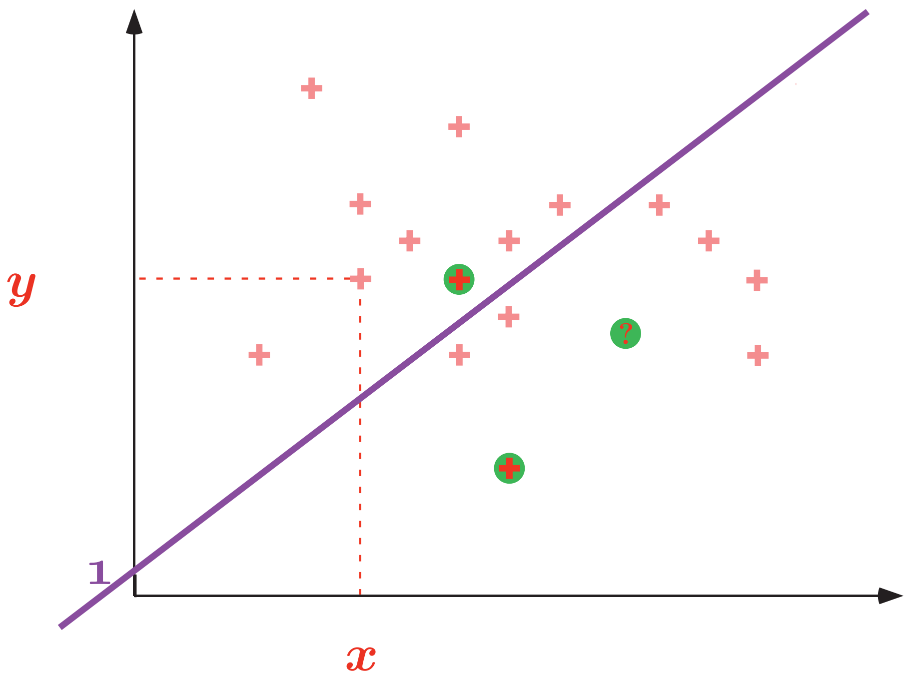
Conservative Approximation

- Is the operation `1/(x+1-y)` well defined at run-time?
- Concrete semantics: **yes**



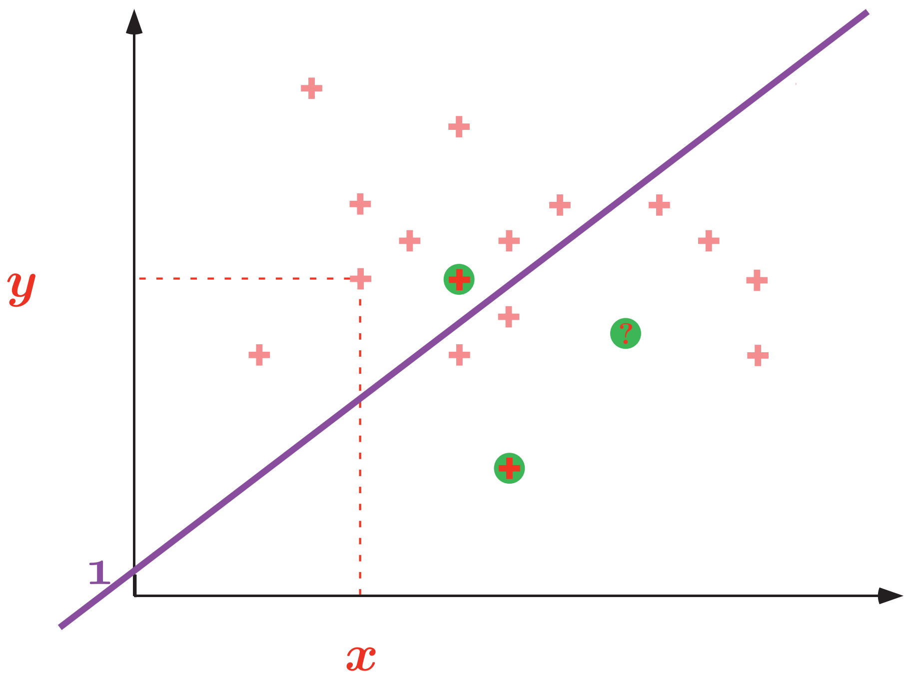
Conservative Approximation

- Is the operation `1/(x+1-y)` well defined at run-time?
- Testing :



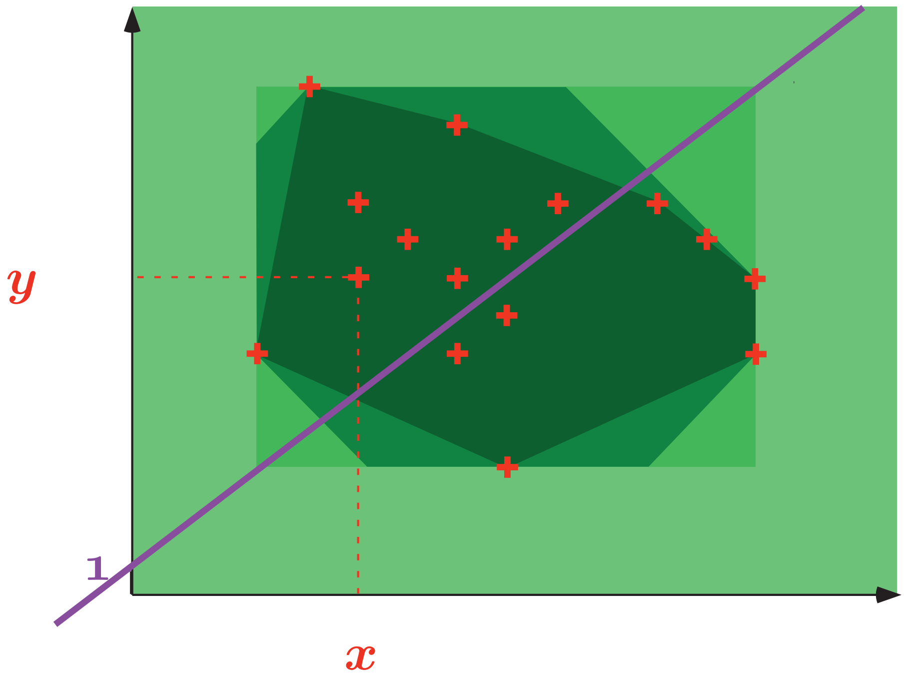
Conservative approximation

- Is the operation `1/(x+1-y)` well defined at run-time?
- Testing : **You <u>never</u> know!**



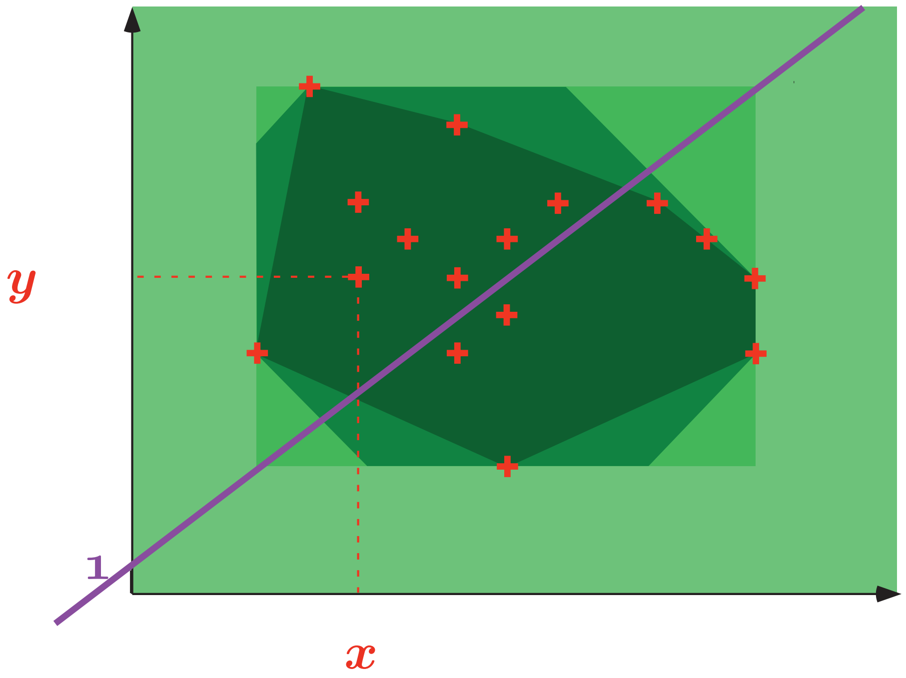
Conservative Approximation

- Is the operation `1/(x+1-y)` well defined at run-time?
- Abstract semantics 1:



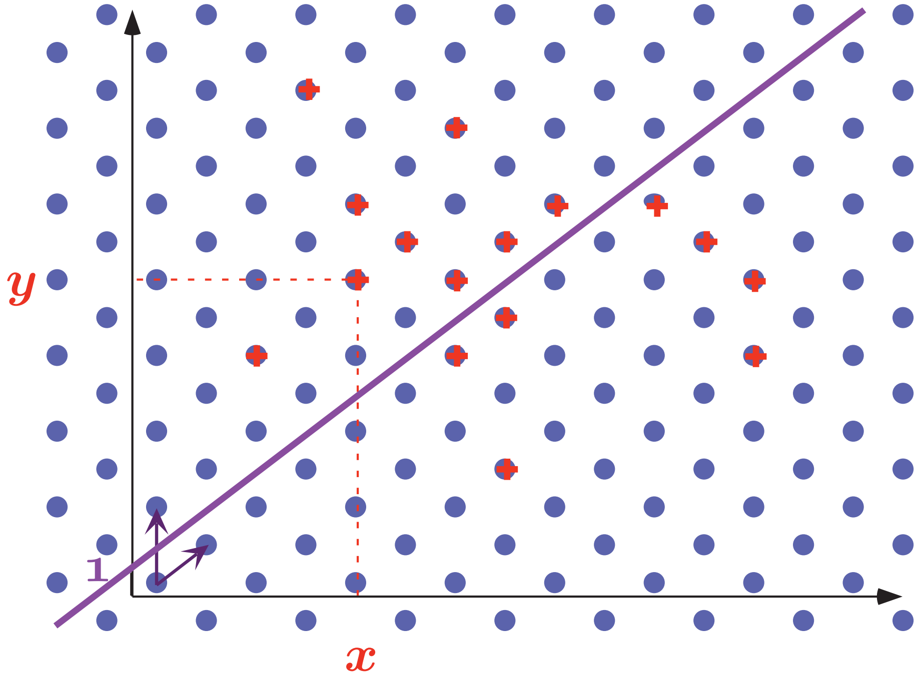
Conservative Approximation

- Is the operation `1/(x+1-y)` well defined at run-time?
- Abstract semantics 1: **I don't know**



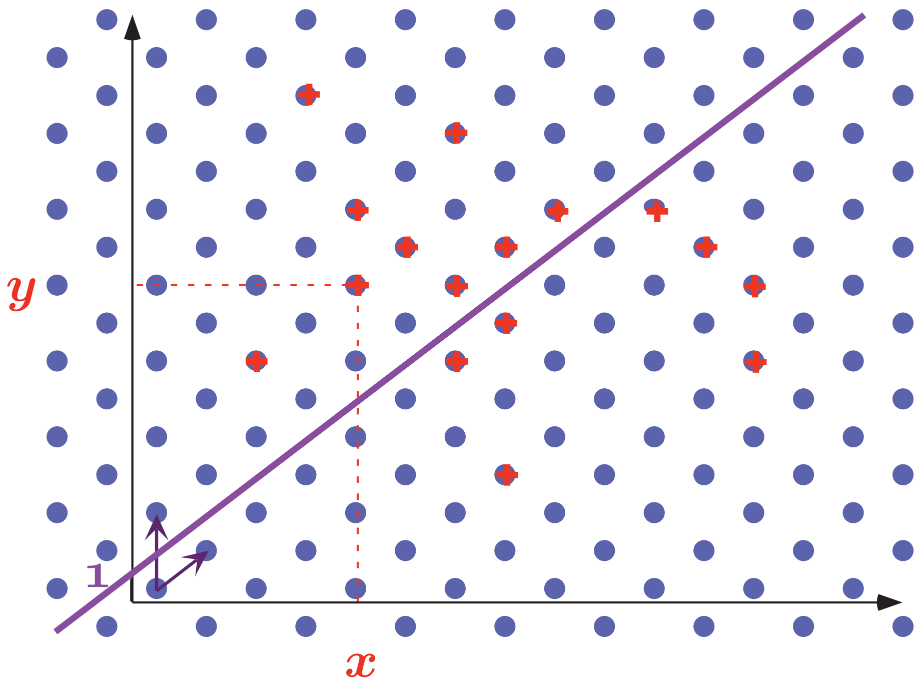
Conservative Approximation

- Is the operation `1/(x+1-y)` well defined at run-time?
- Abstract semantics 2:



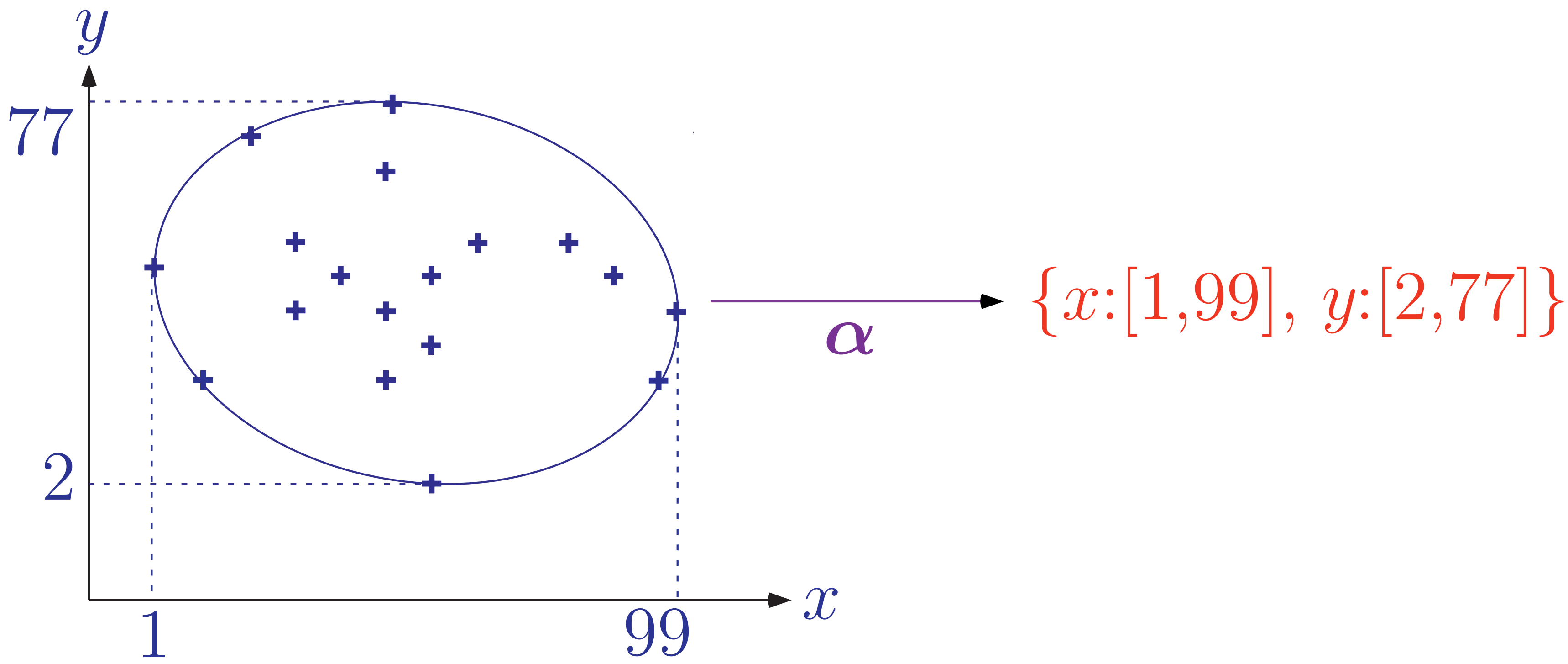
Conservative Approximation

- Is the operation `1/(x+1-y)` well defined at run-time?
- Abstract semantics 2: **yes**

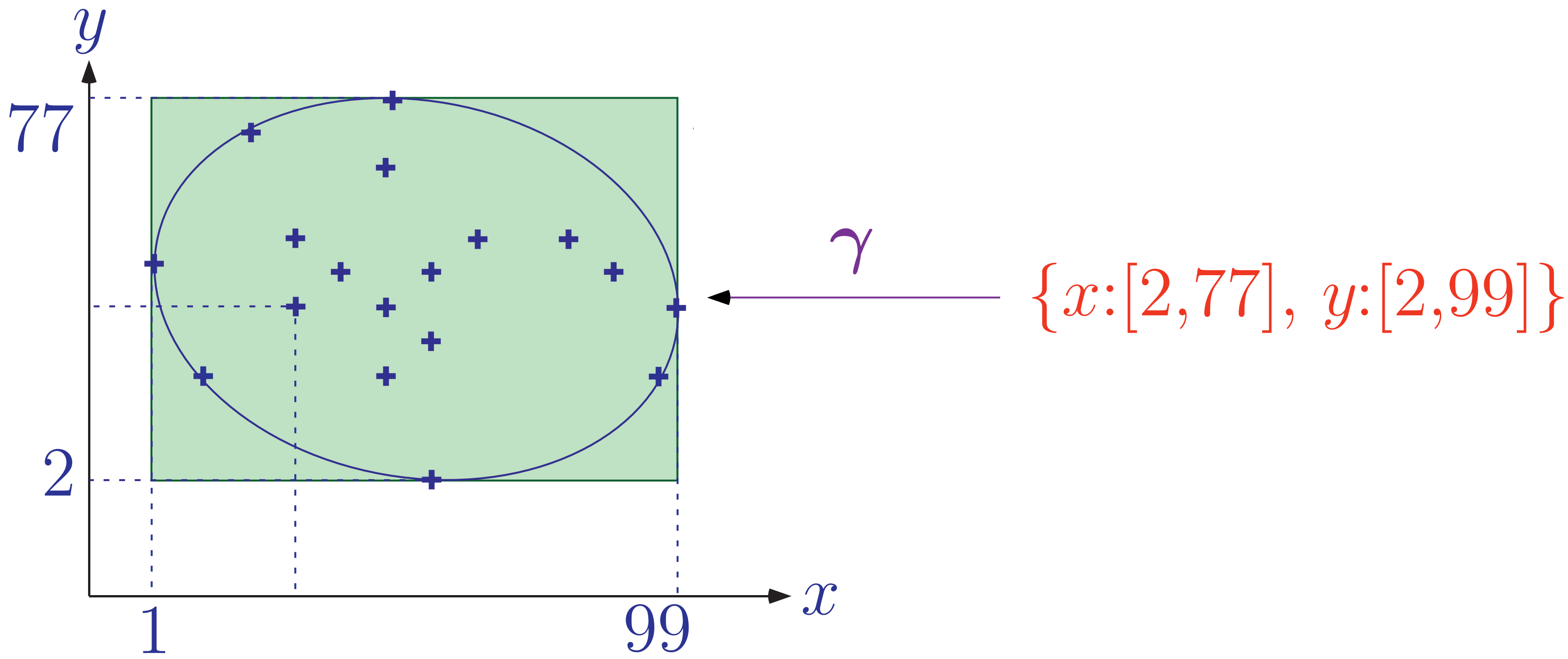


Basic Elements of Abstract Interpretation Theory

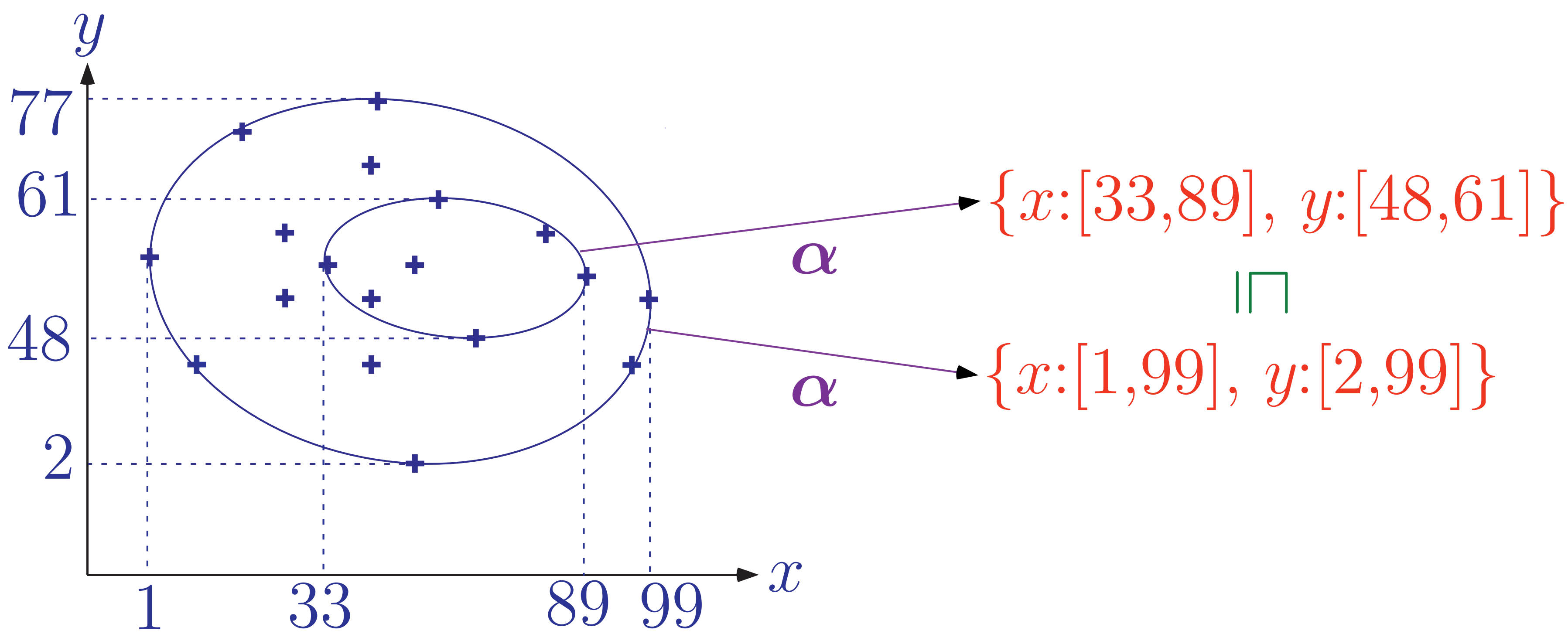
Abstraction α



Concretization γ

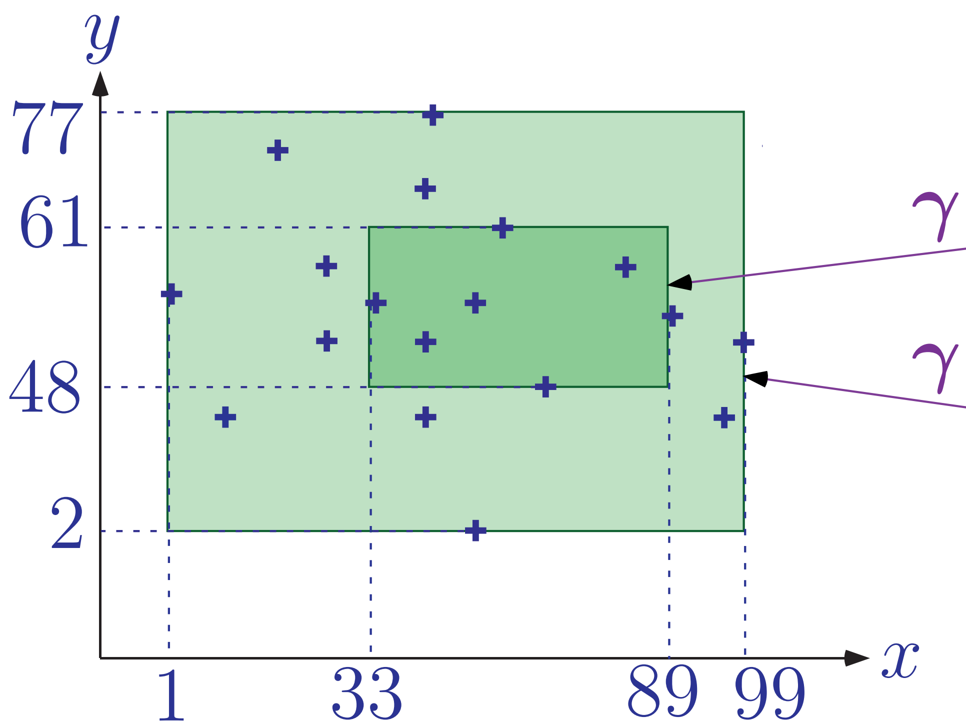


The Abstraction α is Monotone



$$X \subseteq Y \Rightarrow \alpha(X) \sqsubseteq \alpha(Y)$$

The Concretization γ is Monotone



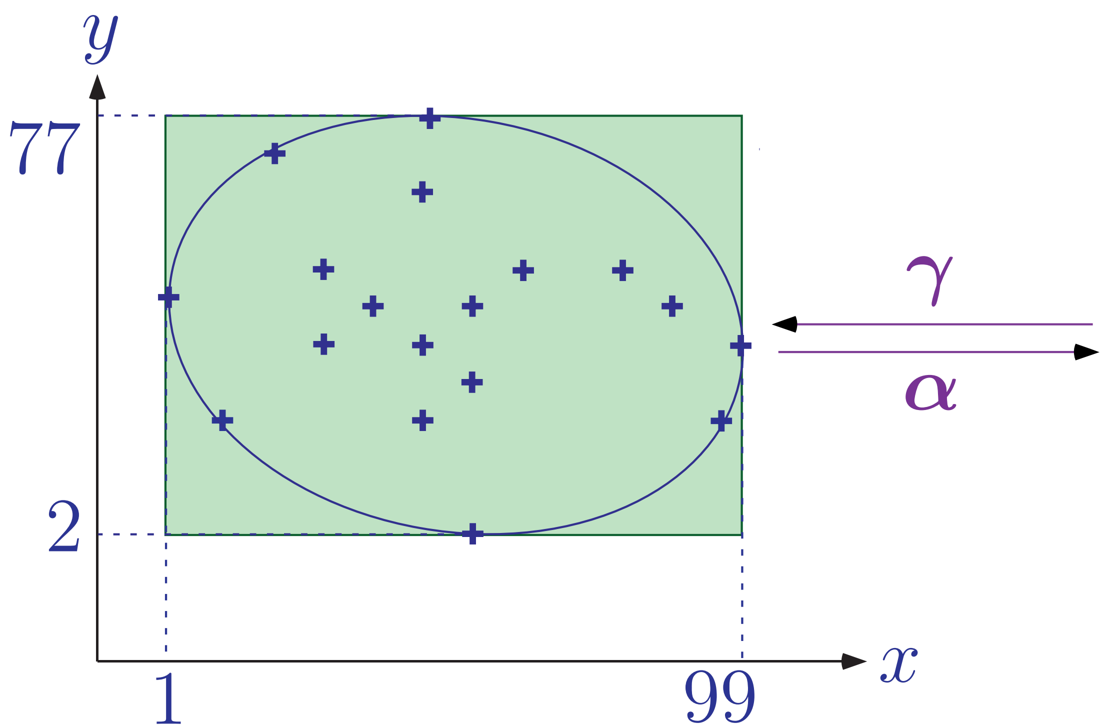
$\{x{:}[33,89],\ y{:}[48,61]\}$

$\sqsupseteq$

$\{x{:}[1,99],\ y{:}[2,99]\}$

$$X \sqsubseteq Y \Rightarrow \gamma(X) \subseteq \gamma(Y)$$

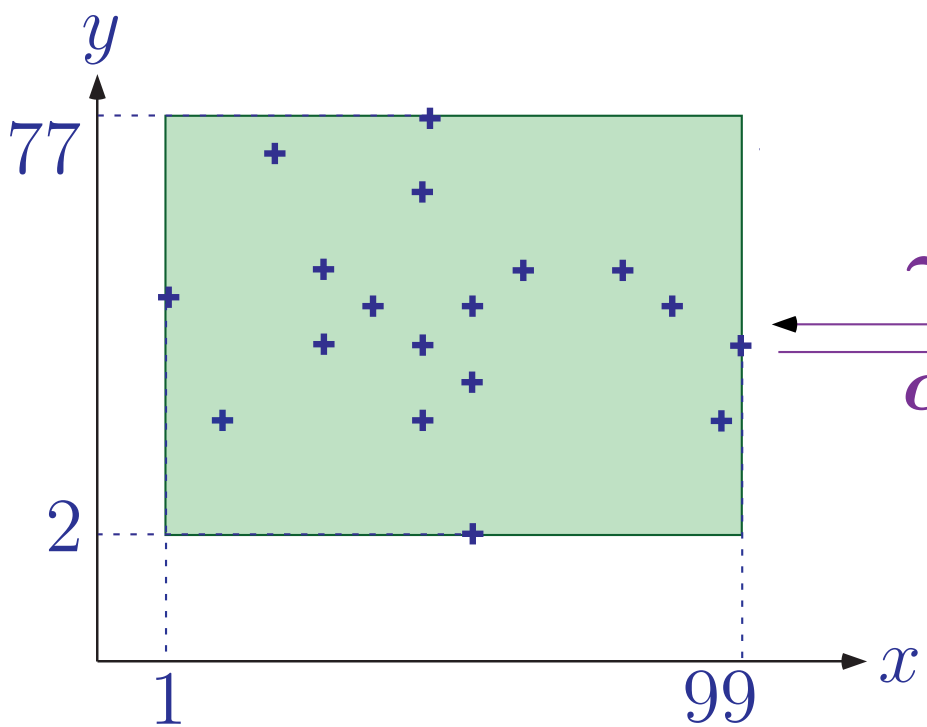
The $\gamma \circ \alpha$ Composition



$\{x{:}[1,99],\ y{:}[2,77]\}$

$$X \subseteq \gamma \circ \alpha(X)$$

The $\alpha \circ \gamma$ Composition



$$\xleftarrow{\gamma} \quad \{x{:}[1,99],\ y{:}[2,77]\}$$
$$\xrightarrow[\alpha]{} \quad \{x{:}[1,99],\ y{:}[2,77]\}$$

$$\alpha \circ \gamma(Y) = Y$$

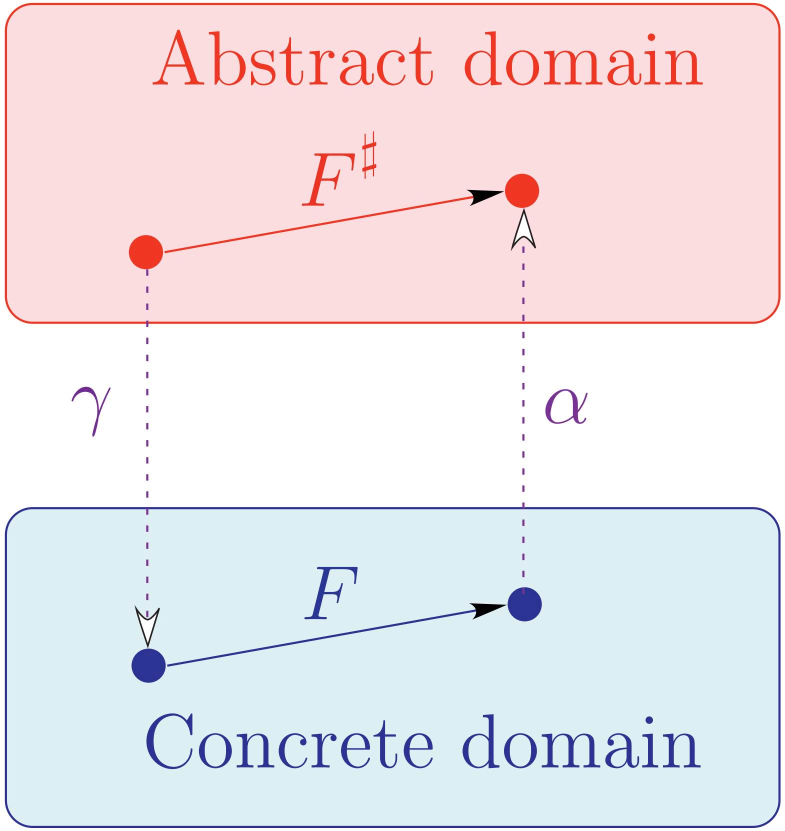
Galois Connection [1]

$$\langle P, \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle Q, \sqsubseteq \rangle$$

iff

- α is monotone
- γ is monotone
- $X \subseteq \gamma \circ \alpha(X)$
- $\alpha \circ \gamma(Y) \sqsubseteq Y$

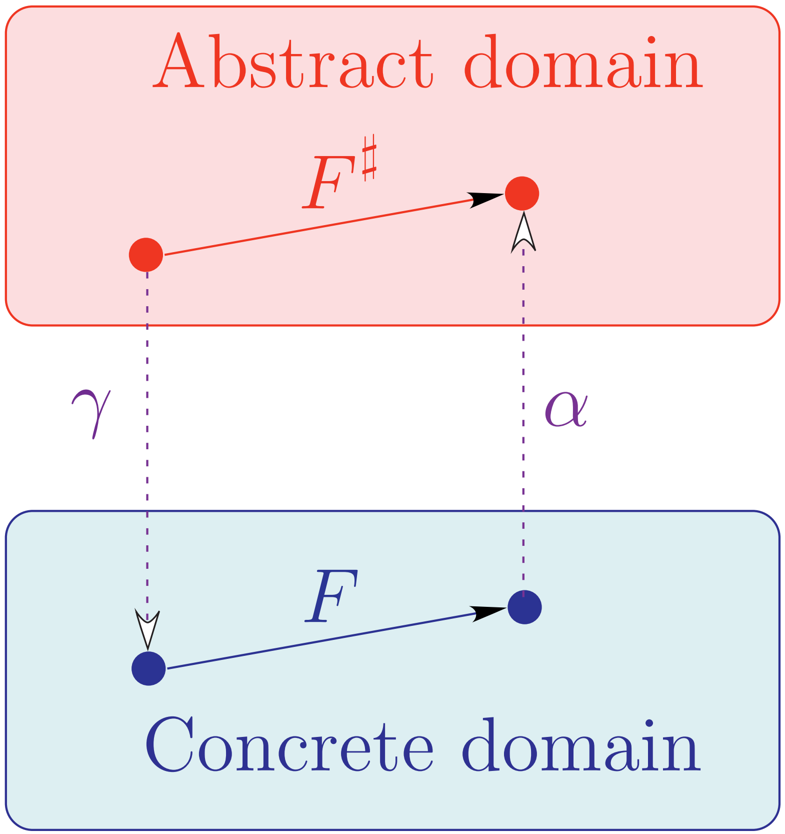
[1] formalizations using closure operators, ideals, etc. are equivalent.



Function Abstraction

$$F^\sharp = \alpha \circ F \circ \gamma$$

Function Abstraction

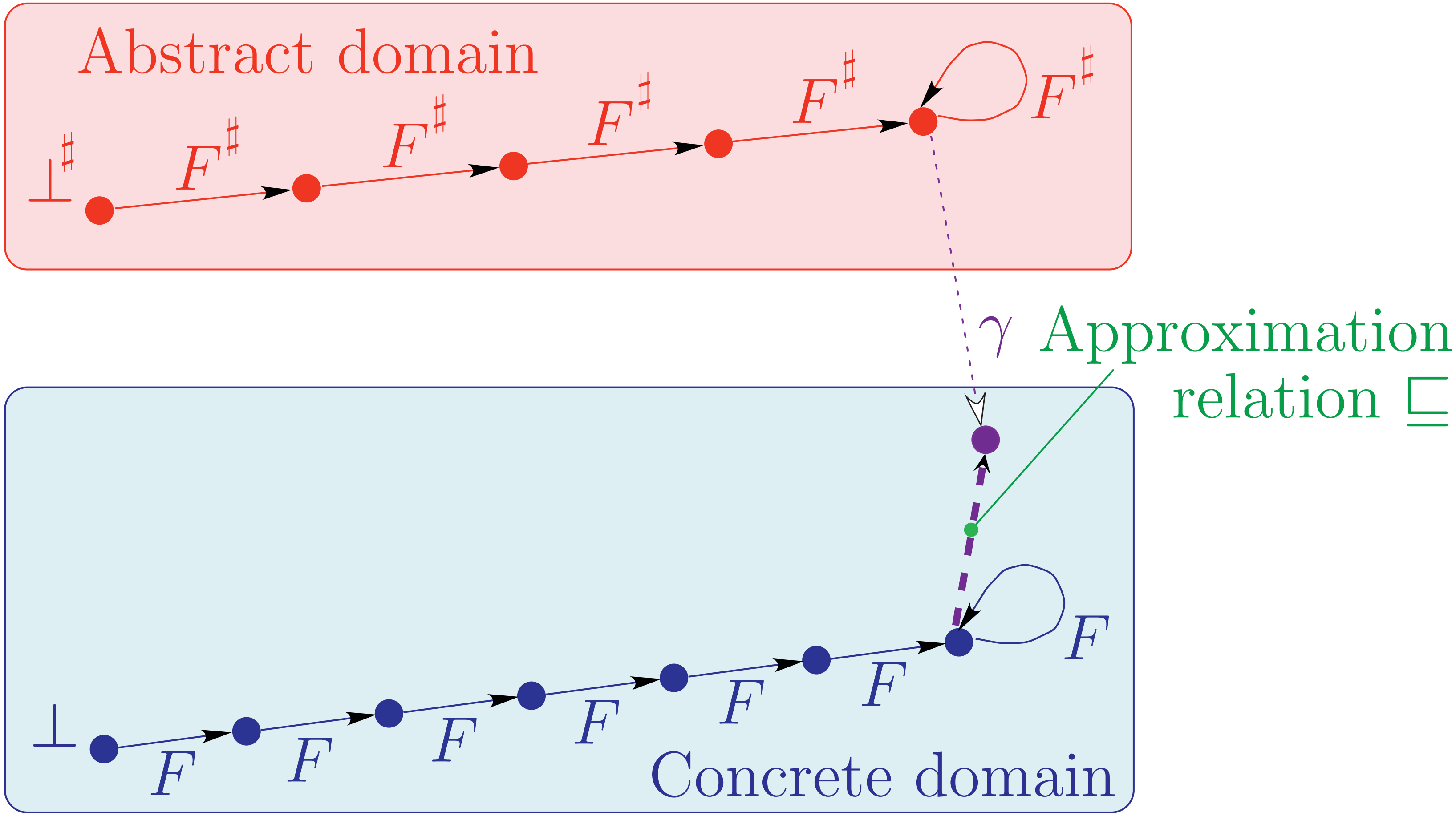


$$F^\sharp = \alpha \circ F \circ \gamma$$

$$\langle P, \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle Q, \sqsubseteq \rangle \Rightarrow$$

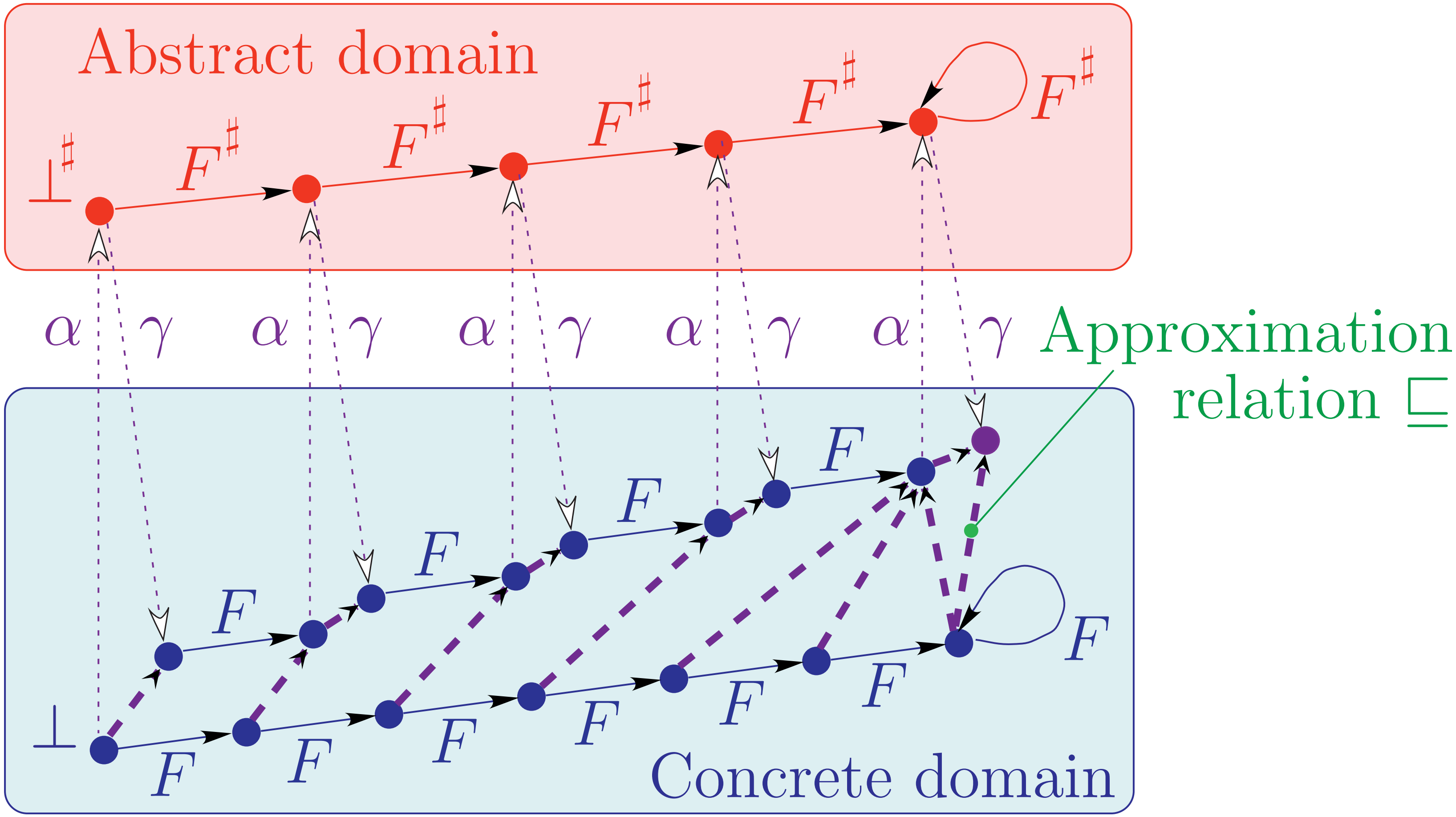
$$\langle P \overset{\textsf{mon}}{\longmapsto} P, \dot{\subseteq} \rangle \underset{\lambda F \cdot \alpha \circ F \circ \gamma}{\overset{\lambda F^\sharp \cdot \gamma \circ F^\sharp \circ \alpha}{\leftrightarrows}} \langle Q \overset{\textsf{mon}}{\longmapsto} Q, \dot{\sqsubseteq} \rangle$$

Fixpoint Abstraction



$$\mathit{lfp}\, F \sqsubseteq \gamma(\mathit{lfp}\, F^\sharp)$$

Fixpoint Abstraction



$$\mathit{lfp}\, F \sqsubseteq \gamma(\mathit{lfp}\, F^\sharp)$$

Exact/Approximate Fixpoint Abstraction

Exact Abstraction:

$$\alpha(\mathit{lfp}\, F) = \mathit{lfp}\, F^\sharp$$

Exact/Approximate Fixpoint Abstraction

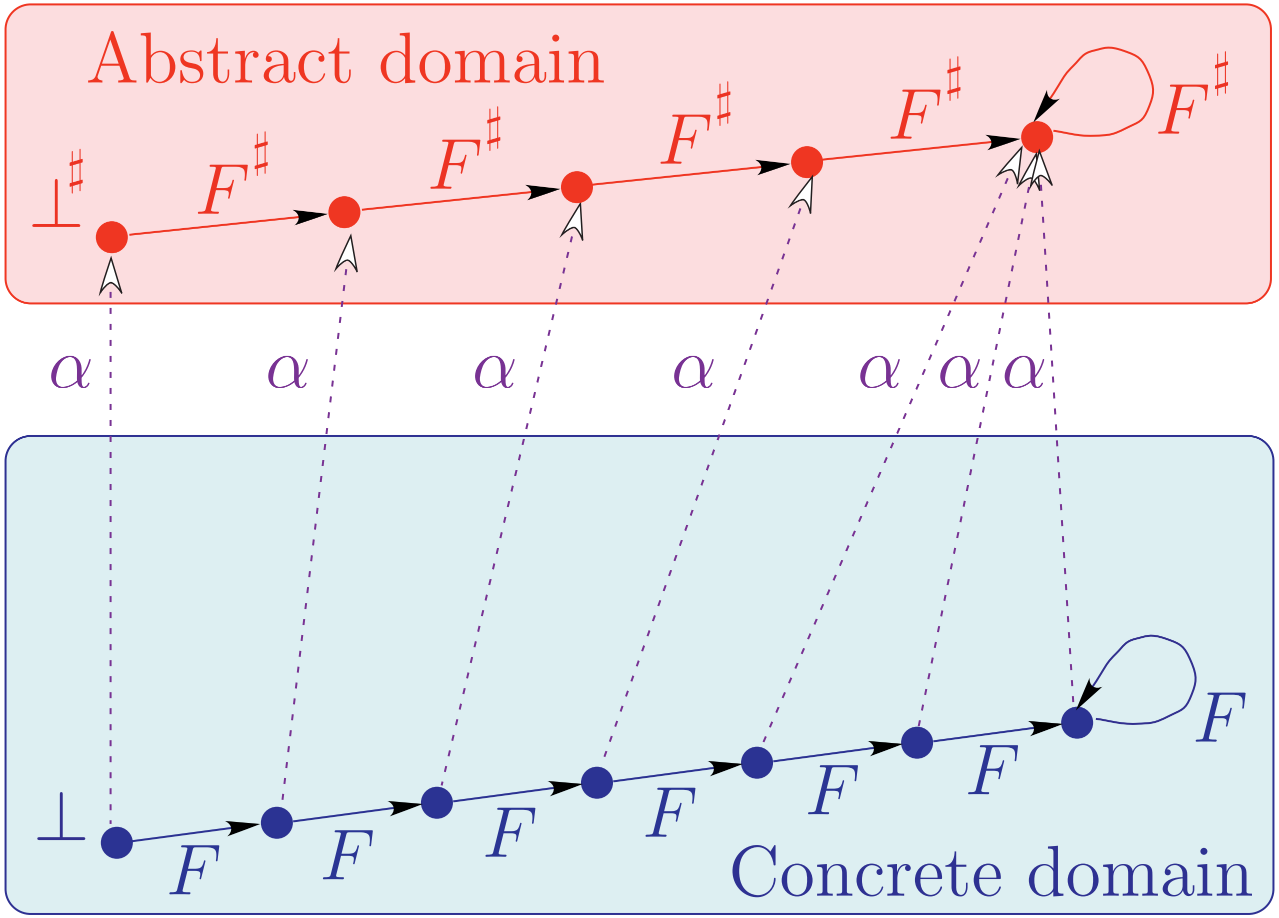
Exact Abstraction:

$$\alpha(\textsf{\textit{lfp}}\, F) = \textsf{\textit{lfp}}\, F^\sharp$$

Approximate Abstraction:

$$\alpha(\textsf{\textit{lfp}}\, F) \sqsubseteq^\sharp \textsf{\textit{lfp}}\, F^\sharp$$

Exact Fixpoint Abstraction



$$\alpha \circ F = F^\sharp \circ \alpha \quad \Rightarrow \quad \alpha(\mathit{lfp}\, F) = \mathit{lfp}\, F^\sharp$$

A Few References on Foundations

[1] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th POPL*, pages 238–252, Los Angeles, CA, 1977. ACM Press.

[2] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *6th POPL*, pages 269–282, San Antonio, TX, 1979. ACM Press.

[3] P. Cousot and R. Cousot. Abstract interpretation frameworks. *J. Logic and Comp.*, 2(4):511–547, 1992.

Applications of Abstract Interpretation

(1) Exact Abstractions

Application to Syntax

The Semantics of Syntax

- Grammar:

$$\begin{aligned} X &:= aY \mid bY \\ Y &:= cY \mid d \end{aligned}$$

The Semantics of Syntax

- Grammar:

$$X := aY \mid bY$$
$$Y := cY \mid d$$

- Equations:

$$\mathcal{X} = \{ay \mid y \in \mathcal{Y}\} \cup \{by \mid y \in \mathcal{Y}\}$$
$$\mathcal{Y} = \{cy \mid y \in \mathcal{Y}\} \cup \{d\}$$

The Semantics of Syntax

- Grammar:

$$\begin{aligned} X &:= aY \mid bY \\ Y &:= cY \mid d \end{aligned}$$

- Equations:

$$\begin{aligned} \mathcal{X} &= \{ay \mid y \in \mathcal{Y}\} \cup \{by \mid y \in \mathcal{Y}\} \\ \mathcal{Y} &= \{cy \mid y \in \mathcal{Y}\} \cup \{d\} \end{aligned}$$

- Transformer F:

$$F(\langle \mathcal{X},\, \mathcal{Y} \rangle) = \\ \langle \{ay \mid y \in \mathcal{Y}\} \cup \{by \mid y \in \mathcal{Y}\},\ \{cy \mid y \in \mathcal{Y}\} \cup \{d\} \rangle$$

- Iterates of the fixpoint *lfp* F:

$$\mathcal{X}^0 = \emptyset$$
$$\mathcal{Y}^0 = \emptyset$$

- Iterates of the fixpoint *lfp* F:

$$\mathcal{X}^0 = \emptyset$$
$$\mathcal{Y}^0 = \emptyset$$

$$\mathcal{X}^1 = \{ay \mid y \in \mathcal{Y}^0\} \cup \{by \mid y \in \mathcal{Y}^0\} = \emptyset$$
$$\mathcal{Y}^1 = \{cy \mid y \in \mathcal{Y}^0\} \cup \{d\} = \{d\}$$

- Iterates of the fixpoint *lfp* F:

$$
\begin{aligned}
\mathcal{X}^0 &= \emptyset\\
\mathcal{Y}^0 &= \emptyset\\[1ex]
\mathcal{X}^1 &= \{ay \mid y \in \mathcal{Y}^0\} \cup \{by \mid y \in \mathcal{Y}^0\} &&= \emptyset\\
\mathcal{Y}^1 &= \{cy \mid y \in \mathcal{Y}^0\} \cup \{d\} &&= \{d\}\\[1ex]
\mathcal{X}^2 &= \{ay \mid y \in \mathcal{Y}^1\} \cup \{by \mid y \in \mathcal{Y}^1\} &&= \{ad, bd\}\\
\mathcal{Y}^2 &= \{cy \mid y \in \mathcal{Y}^1\} \cup \{d\} &&= \{cd, d\}
\end{aligned}
$$

- Iterates of the fixpoint $\mathit{lfp}\, F$:

$$
\begin{aligned}
\mathcal{X}^0 &= \emptyset \\
\mathcal{Y}^0 &= \emptyset \\[1ex]
\mathcal{X}^1 &= \{ay \mid y \in \mathcal{Y}^0\} \cup \{by \mid y \in \mathcal{Y}^0\} && = \emptyset \\
\mathcal{Y}^1 &= \{cy \mid y \in \mathcal{Y}^0\} \cup \{d\} && = \{d\} \\[1ex]
\mathcal{X}^2 &= \{ay \mid y \in \mathcal{Y}^1\} \cup \{by \mid y \in \mathcal{Y}^1\} && = \{ad, bd\} \\
\mathcal{Y}^2 &= \{cy \mid y \in \mathcal{Y}^1\} \cup \{d\} && = \{cd, d\} \\[1ex]
\mathcal{X}^3 &= \{ay \mid y \in \mathcal{Y}^2\} \cup \{by \mid y \in \mathcal{Y}^2\} && = \{acd, ad, bcd, bd\} \\
\mathcal{Y}^3 &= \{cy \mid y \in \mathcal{Y}^2\} \cup \{d\} && = \{ccd, cd, d\} \\
\ldots & \quad \ldots && \quad \ldots
\end{aligned}
$$

$$
\begin{aligned}
\ldots \quad & \ldots \\
\mathcal{X}^n &= \{ay \mid y \in \mathcal{Y}^{n-1}\} \cup \{by \mid y \in \mathcal{Y}^{n-1}\} \\
&= \{ac^{n-2}d, \ldots, acd, ad, bc^{n-2}d, \ldots, bcd, bd\} \\
\mathcal{Y}^n &= \{cy \mid y \in \mathcal{Y}^{n-1}\} \cup \{d\} \\
&= \{c^{n-1}d, \ldots, ccd, cd, d\} \\
\ldots \quad & \ldots
\end{aligned}
$$

$$
\begin{aligned}
\ldots \quad & \ldots \\
\mathcal{X}^n &= \{ay \mid y \in \mathcal{Y}^{n-1}\} \cup \{by \mid y \in \mathcal{Y}^{n-1}\} \\
&= \{ac^{n-2}d, \ldots, acd, ad, bc^{n-2}d, \ldots, bcd, bd\} \\
\mathcal{Y}^n &= \{cy \mid y \in \mathcal{Y}^{n-1}\} \cup \{d\} \\
&= \{c^{n-1}d, \ldots, ccd, cd, d\} \\
\ldots \quad & \ldots
\end{aligned}
$$

- *lfp* $F = \langle \mathcal{X}^\infty, \mathcal{Y}^\infty \rangle$ where:

$$
\begin{aligned}
\mathcal{X}^\infty &= \bigcup_{n \geq 0} \mathcal{X}^n = \{ac^n d, bc^n d \mid n \geq 0\} \\
\mathcal{Y}^\infty &= \bigcup_{n \geq 0} \mathcal{Y}^n = \{c^n d \mid n \geq 0\}
\end{aligned}
$$

FIRST Abstraction

- $\mathsf{FIRST}(\langle \mathcal{X}, \mathcal{Y} \rangle) = \langle \mathsf{FIRST}(\mathcal{X}), \mathsf{FIRST}(\mathcal{Y}) \rangle$

FIRST Abstraction

- $\text{FIRST}(\langle \mathcal{X}, \mathcal{Y} \rangle) = \langle \text{FIRST}(\mathcal{X}), \text{FIRST}(\mathcal{Y}) \rangle$
- $\text{FIRST}(\mathcal{X}) = \{a \mid ax \in \mathcal{X}\}$

FIRST Abstraction

- $\mathsf{FIRST}(\langle \mathcal{X}, \mathcal{Y} \rangle) = \langle \mathsf{FIRST}(\mathcal{X}), \mathsf{FIRST}(\mathcal{Y}) \rangle$
- $\mathsf{FIRST}(\mathcal{X}) = \{a \mid ax \in \mathcal{X}\}$
- $\gamma(Y) = \{ax \mid a \in Y, x \text{ is any sentence}\}$

FIRST Abstraction

- $\mathsf{FIRST}(\langle \mathcal{X}, \mathcal{Y} \rangle) = \langle \mathsf{FIRST}(\mathcal{X}), \mathsf{FIRST}(\mathcal{Y}) \rangle$
- $\mathsf{FIRST}(\mathcal{X}) = \{a \mid ax \in \mathcal{X}\}$
- $\gamma(Y) = \{ax \mid a \in Y, x \text{ is any sentence}\}$
- $\langle \text{Set of sentences}, \subseteq \rangle \underset{\mathsf{FIRST}}{\overset{\gamma}{\leftrightarrows}} \langle \text{Set of symbols}, \subseteq \rangle$

FIRST Abstraction

- $\text{FIRST}(\langle \mathcal{X}, \mathcal{Y} \rangle) = \langle \text{FIRST}(\mathcal{X}), \text{FIRST}(\mathcal{Y}) \rangle$
- $\text{FIRST}(\mathcal{X}) = \{a \mid ax \in \mathcal{X}\}$
- $\gamma(Y) = \{ax \mid a \in Y, x \text{ is any sentence}\}$
- $\langle \text{Set of sentences}, \subseteq \rangle \underset{\text{FIRST}}{\overset{\gamma}{\leftrightarrows}} \langle \text{Set of symbols}, \subseteq \rangle$
- $F^\sharp = \text{FIRST} \circ F \circ \gamma$ is given by the equations:

$$
\begin{aligned}
X &= \text{FIRST}(\{ay \mid y \in \gamma(Y)\} \cup \{by \mid y \in \gamma(Y)\}) \\
&= \{a, b \mid \exists y \in \gamma(Y)\} = \{a, b \mid \exists y \in Y\} = \{a, b \mid Y \neq \emptyset\} \\
Y &= \text{FIRST}(\{cy \mid y \in \gamma(Y)\} \cup \{d\}) = \{c \mid Y \neq \emptyset\} \cup \{d\}
\end{aligned}
$$

- Iterates of the fixpoint $\mathit{lfp}\, F^\sharp$:

$$\mathcal{X}^0 = \emptyset$$
$$\mathcal{Y}^0 = \emptyset$$

- Iterates of the fixpoint *lfp* $F^\sharp$:

$$\mathcal{X}^0 = \emptyset$$
$$\mathcal{Y}^0 = \emptyset$$

$$\mathcal{X}^1 = \{a, b \mid Y^0 \neq \emptyset\} \qquad = \emptyset$$
$$\mathcal{Y}^1 = \{c \mid Y^0 \neq \emptyset\} \cup \{d\} = \{d\}$$

- Iterates of the fixpoint *lfp* $F^\sharp$:

$$\mathcal{X}^0 = \emptyset$$
$$\mathcal{Y}^0 = \emptyset$$

$$\mathcal{X}^1 = \{a, b \mid Y^0 \neq \emptyset\} = \emptyset$$
$$\mathcal{Y}^1 = \{c \mid Y^0 \neq \emptyset\} \cup \{d\} = \{d\}$$

$$\mathcal{X}^2 = \{a, b \mid Y^1 \neq \emptyset\} = \{a, b\}$$
$$\mathcal{Y}^2 = \{c \mid Y^1 \neq \emptyset\} \cup \{d\} = \{c, d\}$$

- Iterates of the fixpoint *lfp* $F^\sharp$:

$$\mathcal{X}^0 = \emptyset$$
$$\mathcal{Y}^0 = \emptyset$$

$$\mathcal{X}^1 = \{a, b \mid Y^0 \neq \emptyset\} = \emptyset$$
$$\mathcal{Y}^1 = \{c \mid Y^0 \neq \emptyset\} \cup \{d\} = \{d\}$$

$$\mathcal{X}^2 = \{a, b \mid Y^1 \neq \emptyset\} = \{a, b\}$$
$$\mathcal{Y}^2 = \{c \mid Y^1 \neq \emptyset\} \cup \{d\} = \{c, d\}$$

$$\mathcal{X}^3 = \{a, b \mid Y^2 \neq \emptyset\} = \{a, b\}$$
$$\mathcal{Y}^3 = \{c \mid Y^2 \neq \emptyset\} \cup \{d\} = \{c, d\}$$

- Iterates of the fixpoint $\mathit{lfp}\, F^\sharp$:

$$\mathcal{X}^0 = \emptyset$$
$$\mathcal{Y}^0 = \emptyset$$

$$\mathcal{X}^1 = \{a, b \mid Y^0 \neq \emptyset\} = \emptyset$$
$$\mathcal{Y}^1 = \{c \mid Y^0 \neq \emptyset\} \cup \{d\} = \{d\}$$

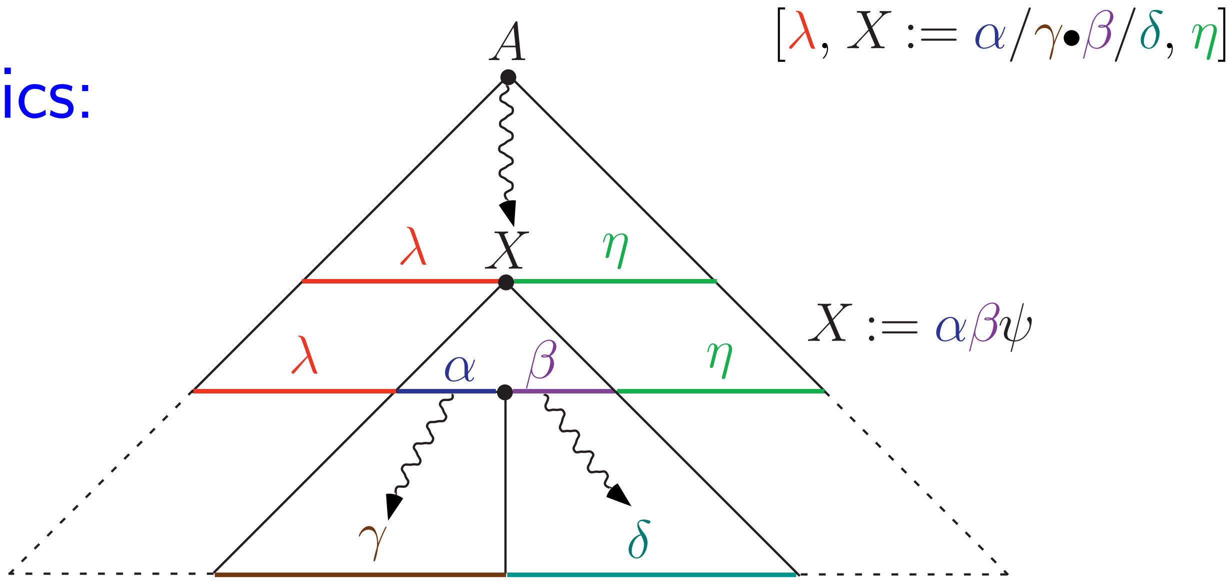
$$\mathcal{X}^2 = \{a, b \mid Y^1 \neq \emptyset\} = \{a, b\}$$
$$\mathcal{Y}^2 = \{c \mid Y^1 \neq \emptyset\} \cup \{d\} = \{c, d\}$$

$$\mathcal{X}^3 = \{a, b \mid Y^2 \neq \emptyset\} = \{a, b\}$$
$$\mathcal{Y}^3 = \{c \mid Y^2 \neq \emptyset\} \cup \{d\} = \{c, d\}$$

- The abstraction is exact so $\mathrm{FIRST}(\mathit{lfp}\, F) = \mathit{lfp}\, F^\sharp$.

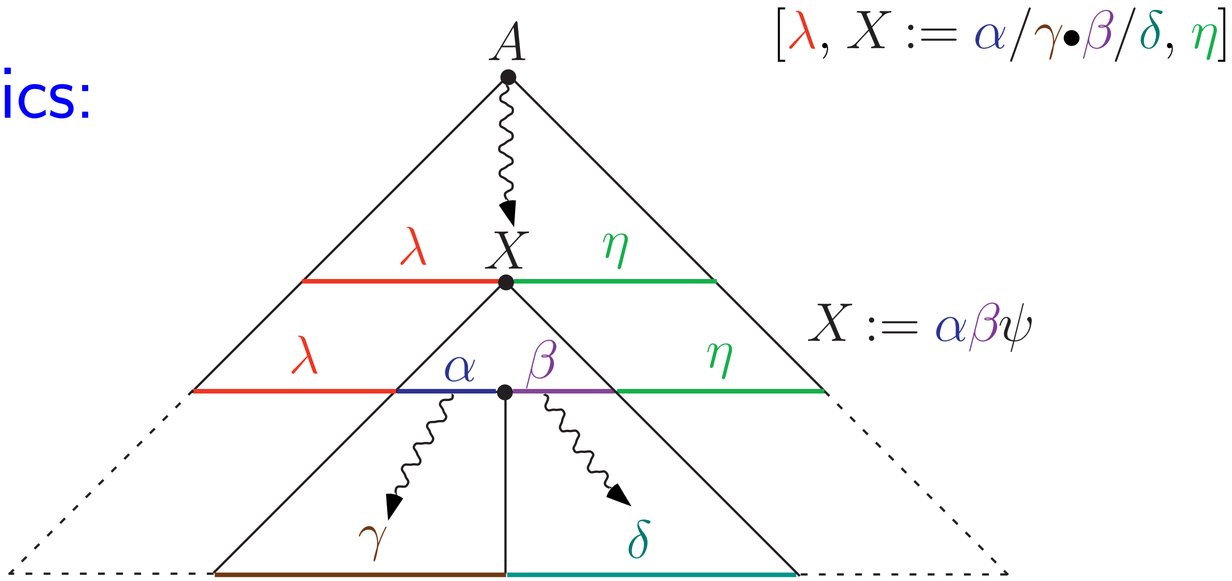
Syntax Analysis

- Refined semantics:



Syntax Analysis

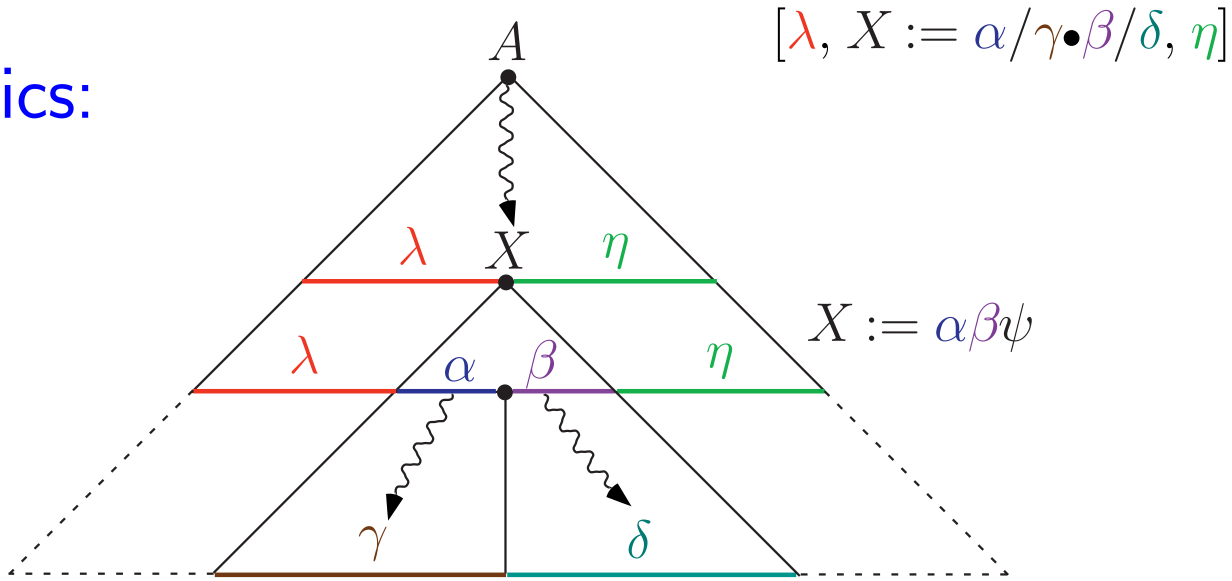
- Refined semantics:



- Abstraction to classical grammar semantics:
  $[\lambda, X := \alpha/\gamma \cdot \beta/\delta, \eta] \longrightarrow \lambda\gamma\delta\eta$, if terminal

Syntax Analysis

- Refined semantics:



- Abstraction to classical grammar semantics:

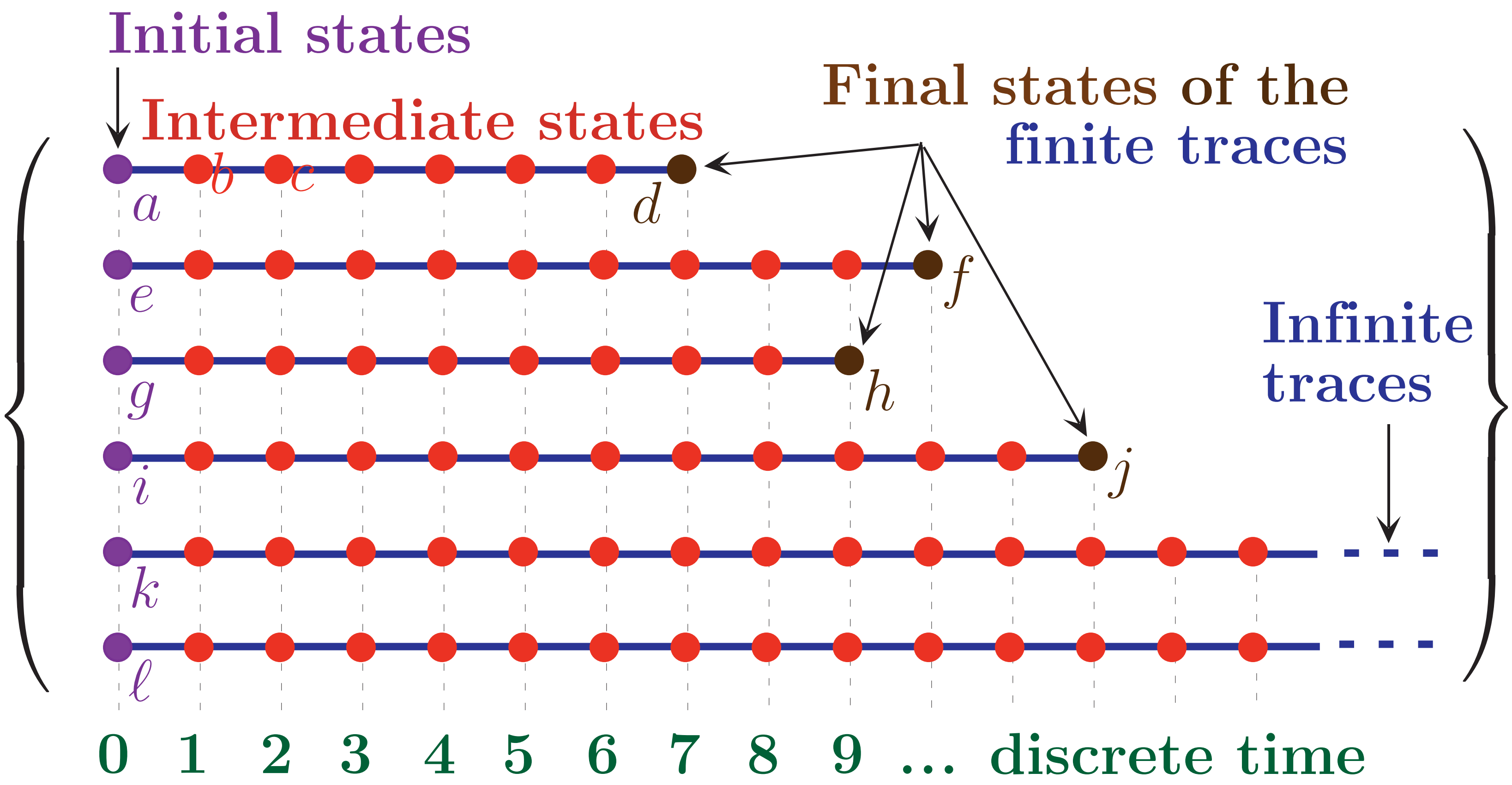
  $[\lambda, X := \alpha/\gamma \cdot \beta/\delta, \eta] \longrightarrow \lambda\gamma\delta\eta$, if terminal

- Abstraction to Earley algorithm:

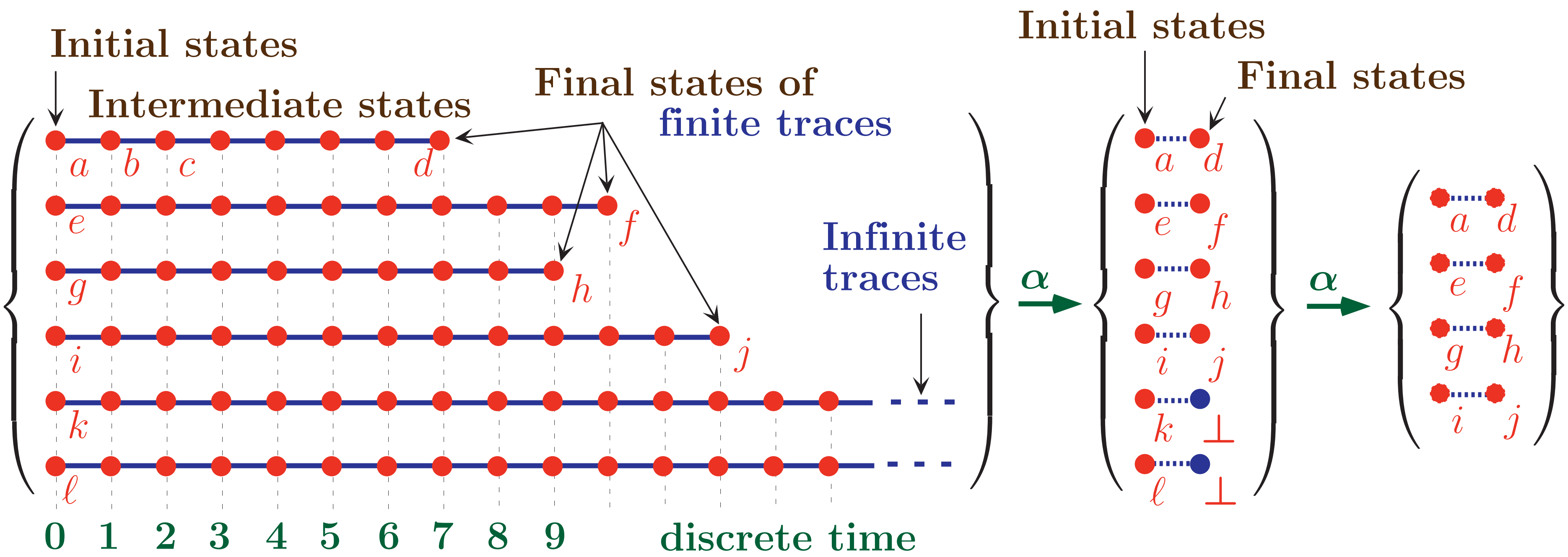
  $[\lambda, X := \alpha/\gamma \cdot \beta/\delta, \eta] \longrightarrow [X := \alpha \cdot \beta, \gamma, \text{FIRST}(\delta)]$

Application to Semantics

Trace Semantics (Once Again)



Example 1 of Semantics Abstraction

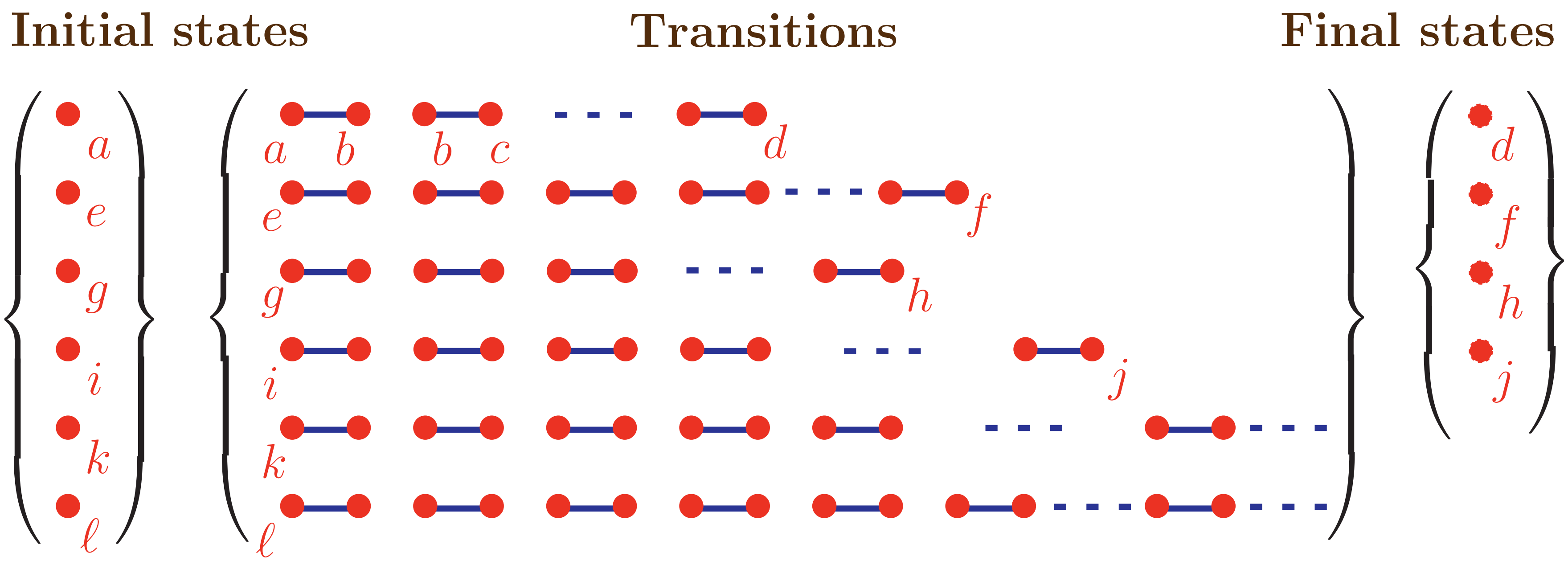


Trace semantics

Denotational semantics

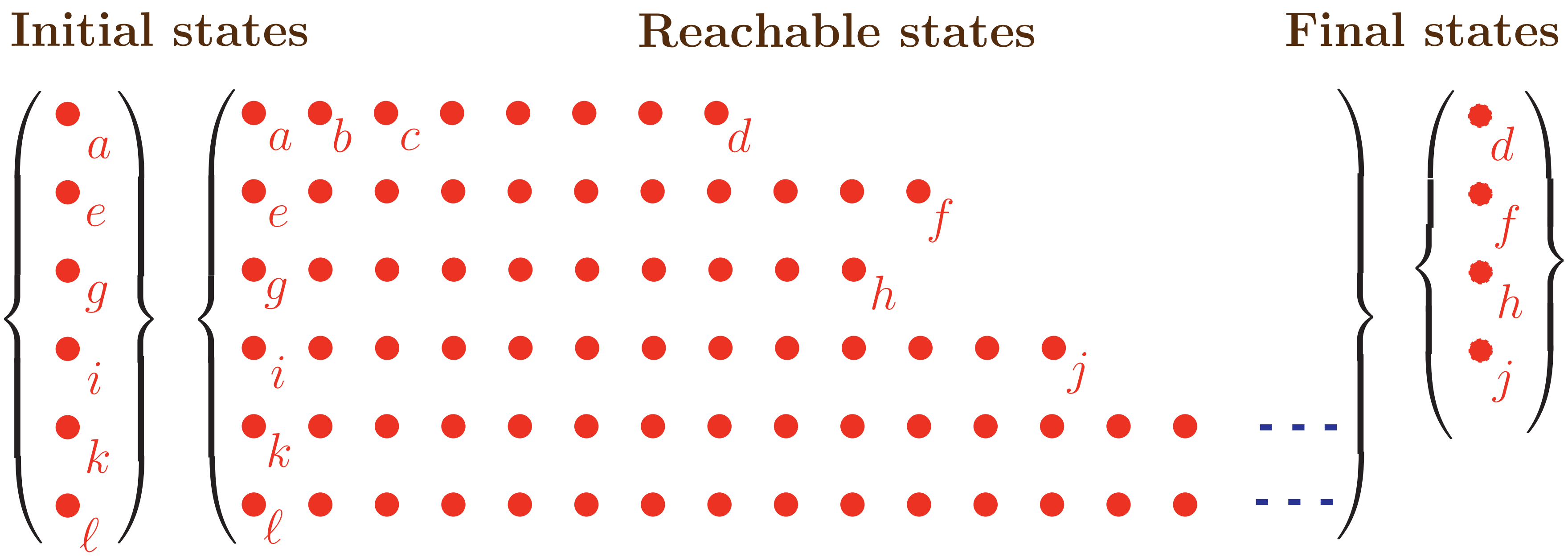
Natural semantics

Example 2 of Semantics Abstraction



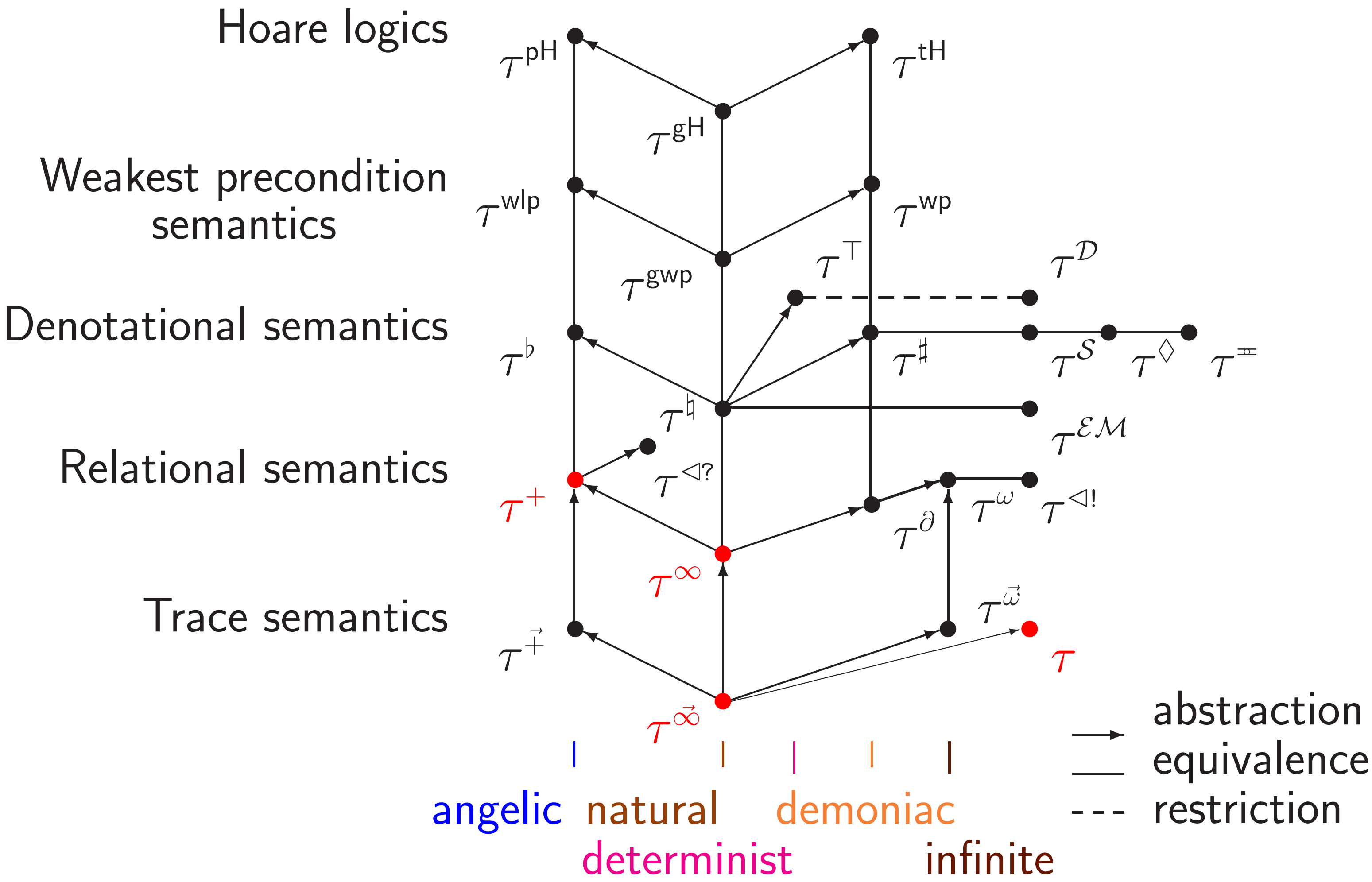
(Small-Step) Operational Semantics

Example 3 of Semantics Abstraction

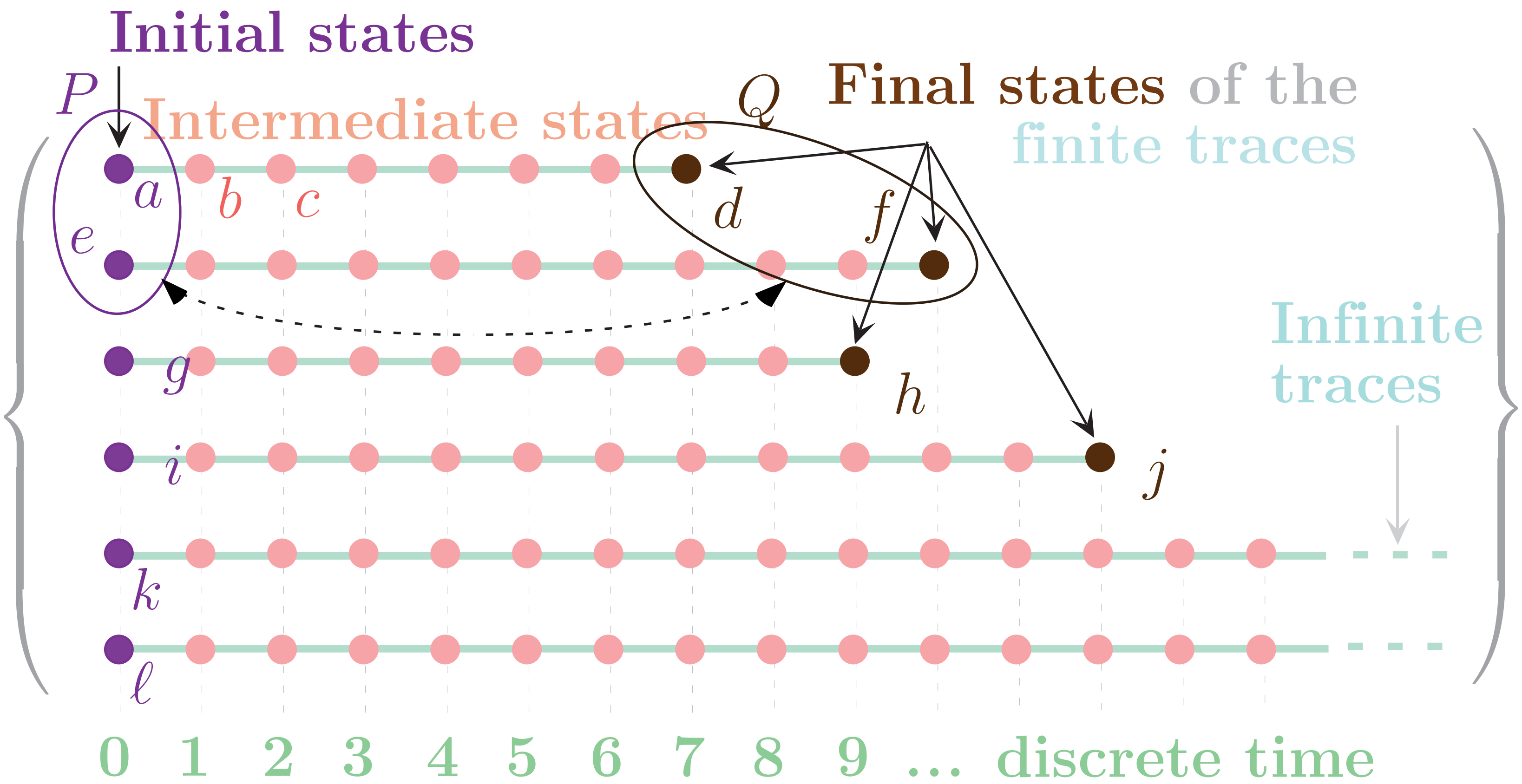


Partial Correctness / Invariance Semantics

Lattice of Semantics



Example 4: Hoare logic for partial correctness



$$\{P\}C\{Q\} \Leftrightarrow P \subseteq \{\bullet \mid \bullet\!-\!\bullet\!-\!\bullet\!-\ldots-\!\bullet \in [\![C]\!] \wedge \bullet \in Q\}$$

The approximation in Hoare logic

For partial correctness:

- Non-terminating behaviors are forgotten;
- The order in which intermediate states may successively appear in a computation is forgotten.

The approximation in Hoare logic

For partial correctness:

- Non-terminating behaviors are forgotten;
- The order in which intermediate states may successively appear in a computation is forgotten.

Conservative approximation:

- Does the program always terminate?

The approximation in Hoare logic

For partial correctness:

- Non-terminating behaviors are forgotten;
- The order in which intermediate states may successively appear in a computation is forgotten.

Conservative approximation:

- Does the program always terminate? **I don't know;**

The approximation in Hoare logic

For partial correctness:

- Non-terminating behaviors are forgotten;
- The order in which intermediate states may successively appear in a computation is forgotten.

Conservative approximation:

- Does the program always terminate? **I don't know;**
- If variable x has always been 0 in the past and is assigned the value 1, will variable y be eventually assigned the value 1?

The approximation in Hoare logic

For partial correctness:

- Non-terminating behaviors are forgotten;
- The order in which intermediate states may successively appear in a computation is forgotten.

Conservative approximation:

- Does the program always terminate? **I don't know;**
- If variable x has always been 0 in the past and is assigned the value 1, will variable y be eventually assigned the value 1? **I don't know.**

Application to Program Transformation

Principle of Online Program Transformation

$$\text{Subject program } P \xrightarrow[\text{transformation } \mathbb{t}[\![\bullet]\!]]{\text{Syntactic}} \text{Transformed program } \mathbb{t}[\![P]\!]$$

Principle of Online Program Transformation

$$
\begin{tikzcd}[column sep=huge, row sep=huge]
\begin{array}{c}\text{Subject}\\ \text{program } P\end{array} \arrow[r, "\text{Syntactic}", "\text{transformation } \mathbb{t}[\![\bullet]\!]"'] \arrow[d, shift right, "\boldsymbol{\mathcal{S}}[\![\bullet]\!]"'] & \begin{array}{c}\text{Transformed program}\\ \mathbb{t}[\![P]\!] = \mathbb{P}[\mathsf{t}[\boldsymbol{\mathcal{S}}[\![P]\!]]]\end{array} \arrow[d, dashed, shift right, "\boldsymbol{\mathcal{S}}[\![\bullet]\!]"'] \\
\begin{array}{c}\text{Subject program}\\ \text{semantics } \boldsymbol{\mathcal{S}}[\![P]\!]\end{array} \arrow[r, "\text{Semantic}", "\text{transformation } \mathsf{t}[\bullet]"'] \arrow[u, dashed, twoheadrightarrow, shift right, "\mathbb{P}[\bullet]"'] & \begin{array}{c}\text{Transformed program}\\ \text{semantics } \mathsf{t}[\boldsymbol{\mathcal{S}}[\![P]\!]]\end{array} \arrow[u, twoheadrightarrow, shift right, "\mathbb{P}[\bullet]"']
\end{tikzcd}
$$

Principle of Online Program Transformation

$$
\begin{tikzcd}[column sep=huge, row sep=huge]
\text{Subject program } P \arrow[r, "\text{Syntactic}", "\text{transformation } \mathbb{t}[\![\bullet]\!]"'] \arrow[d, shift right, "\boldsymbol{\mathcal{S}}[\![\bullet]\!]"'] & \text{Transformed program } \mathbb{t}[\![P]\!] = \mathbb{P}[\mathsf{t}[\boldsymbol{\mathcal{S}}[\![P]\!]]] \arrow[d, shift right, dashed, "\boldsymbol{\mathcal{S}}[\![\bullet]\!]"'] \\
\text{Subject program semantics } \boldsymbol{\mathcal{S}}[\![P]\!] \arrow[u, shift right, dashed, "\mathbb{P}[\bullet]"'] \arrow[r, "\text{Semantic}", "\text{transformation } \mathsf{t}[\bullet]"'] \arrow[dr, shift right, "\alpha_{\mathcal{O}}"'] & \text{Transformed program semantics } \mathsf{t}[\boldsymbol{\mathcal{S}}[\![P]\!]] \arrow[u, shift right, "\mathbb{P}[\bullet]"'] \arrow[d, "\alpha_{\mathcal{O}}"] \\
& \alpha_{\mathcal{O}}(\boldsymbol{\mathcal{S}}[\![P]\!]) = \alpha_{\mathcal{O}}(\mathsf{t}[\boldsymbol{\mathcal{S}}[\![P]\!]]) \arrow[ul, dashed, shift right, "\gamma_{\mathcal{O}}"'] \arrow[u, dashed, shift left, "\gamma_{\mathcal{O}}"]
\end{tikzcd}
$$

Observational abstraction

(2) Approximate Abstractions

Application to Type Systems

Syntax of the Lambda Calculus

$\mathtt{x}, \mathtt{f}, \ldots \in \mathbb{X}$	:	variables
$e \in \mathbb{E}$	:	expressions
e	$::= \mathtt{x}$	variable
	$\mid \boldsymbol{\lambda} \mathtt{x} \cdot e$	abstraction
	$\mid e_1(e_2)$	application
	$\mid \boldsymbol{\mu} \mathtt{f} \cdot \boldsymbol{\lambda} \mathtt{x} \cdot e$	recursion
	$\mid \mathbf{1}$	one
	$\mid e_1 - e_2$	difference
	$\mid (e_1 \mathbin{\mathbf{?}} e_2 \mathbin{\mathbf{:}} e_3)$	conditional

Semantic Domains

Ω	wrong/runtime error value
$\perp$	non-termination
$\mathbb{W} \stackrel{\text{def}}{=} \{\Omega\}$	wrong
$\mathsf{z} \in \mathbb{Z}$	integers
$\mathsf{u}, \mathsf{f}, \varphi \in \mathbb{U} \cong \mathbb{W}_{\perp} \oplus \mathbb{Z}_{\perp} \oplus [\mathbb{U} \mapsto \mathbb{U}]^{2}{}_{\perp}$	values
$\mathsf{R} \in \mathbb{R} \stackrel{\text{def}}{=} \mathbb{X} \mapsto \mathbb{U}$	environments
$\phi \in \mathbb{S} \stackrel{\text{def}}{=} \mathbb{R} \mapsto \mathbb{U}$	semantic domain

2 $[\mathbb{U} \mapsto \mathbb{U}]$: continuous, $\perp$-strict, Ω-strict functions from values $\mathbb{U}$ to values $\mathbb{U}$.

Standard Denotational and Collecting Semantics

- The denotational semantics is:

$$\mathbf{S}[\![\bullet]\!] \in \mathbb{E} \mapsto \mathbb{S}$$

- A concrete property P of a program is a set of possible program behaviors:

$$P \in \mathbb{P} \stackrel{\text{def}}{=} \wp(\mathbb{S})$$

- The standard collecting semantics is the strongest concrete property:

$$\mathbf{C}[\![\bullet]\!] \in \mathbb{E} \mapsto \mathbb{P} \qquad \mathbf{C}[\![e]\!] \stackrel{\text{def}}{=} \{\mathbf{S}[\![e]\!]\}$$

Church/Curry Monotypes

- Simple types are monomorphic:

$$m \in \mathbb{M}^{\mathrm{c}}, \quad m ::= \mathtt{int} \mid m_1 \mathbin{\texttt{->}} m_2 \qquad \text{monotype}$$

- A type environment associates a type to free program variables:

$$H \in \mathbb{H}^{\mathrm{c}} \stackrel{\text{def}}{=} \mathbb{X} \mapsto \mathbb{M}^{\mathrm{c}} \qquad \text{type environment}$$

Church/Curry Monotypes (continued)

- A typing $\langle H,\ m \rangle$ specifies a possible result type m in a given type environment H assigning types to free variables:

$$\theta \in \mathbb{I}^{\mathsf{c}} \stackrel{\text{def}}{=} \mathbb{H}^{\mathsf{c}} \times \mathbb{M}^{\mathsf{c}} \qquad \text{typing}$$

- An abstract property or program type is a set of typings;

$$T \in \mathbb{T}^{\mathsf{c}} \stackrel{\text{def}}{=} \wp(\mathbb{I}^{\mathsf{c}}) \qquad \text{program type}$$

Concretization Function

The meaning of types is a program property, as defined by the concretization function γ^{c}: [3]

- Monotypes $\gamma_1^{\mathsf{c}} \in \mathbb{M}^{\mathsf{c}} \mapsto \wp(\mathbb{U})$:

$$
\begin{aligned}
\gamma_1^{\mathsf{c}}(\mathtt{int}) &\stackrel{\text{def}}{=} \mathbb{Z} \cup \{\perp\} \\
\gamma_1^{\mathsf{c}}(m_1 \mathbin{\texttt{->}} m_2) &\stackrel{\text{def}}{=} \{\varphi \in [\mathbb{U} \mapsto \mathbb{U}] \mid \\
&\qquad\qquad \forall \mathbf{u} \in \gamma_1^{\mathsf{c}}(m_1) : \varphi(\mathbf{u}) \in \gamma_1^{\mathsf{c}}(m_2)\} \\
&\quad \cup \{\perp\}
\end{aligned}
$$

[3] For short up/down lifting/injection are omitted.

- type environment $\gamma_2^{\mathsf{c}} \in \mathbb{H}^{\mathsf{c}} \longmapsto \wp(\mathbb{R})$:

$$\gamma_2^{\mathsf{c}}(H) \stackrel{\mathsf{def}}{=} \{\mathsf{R} \in \mathbb{R} \mid \forall \mathbf{x} \in \mathbb{X} : \mathsf{R}(\mathbf{x}) \in \gamma_1^{\mathsf{c}}(H(\mathbf{x}))\}$$

- type environment $\gamma_2^{\mathsf{c}} \in \mathbb{H}^{\mathsf{c}} \longmapsto \wp(\mathbb{R})$:

$$\gamma_2^{\mathsf{c}}(H) \stackrel{\text{def}}{=} \{\mathsf{R} \in \mathbb{R} \mid \forall \mathbf{x} \in \mathbb{X} : \mathsf{R}(\mathbf{x}) \in \gamma_1^{\mathsf{c}}(H(\mathbf{x}))\}$$

- typing $\gamma_3^{\mathsf{c}} \in \mathbb{I}^{\mathsf{c}} \longmapsto \mathbb{P}$:

$$\gamma_3^{\mathsf{c}}(\langle H,\ m\rangle) \stackrel{\text{def}}{=} \{\phi \in \mathbb{S} \mid \forall \mathsf{R} \in \gamma_2^{\mathsf{c}}(H) : \phi(\mathsf{R}) \in \gamma_1^{\mathsf{c}}(m)\}$$

- type environment $\gamma_2^c \in \mathbb{H}^c \longmapsto \wp(\mathbb{R})$:

$$\gamma_2^c(H) \stackrel{\text{def}}{=} \{\mathsf{R} \in \mathbb{R} \mid \forall \mathbf{x} \in \mathbb{X} : \mathsf{R}(\mathbf{x}) \in \gamma_1^c(H(\mathbf{x}))\}$$

- typing $\gamma_3^c \in \mathbb{I}^c \longmapsto \mathbb{P}$:

$$\gamma_3^c(\langle H,\ m\rangle) \stackrel{\text{def}}{=} \{\phi \in \mathbb{S} \mid \forall \mathsf{R} \in \gamma_2^c(H) : \phi(\mathsf{R}) \in \gamma_1^c(m)\}$$

- program type $\gamma^c \in \mathbb{T}^c \longmapsto \mathbb{P}$:

$$\gamma^c(T) \stackrel{\text{def}}{=} \bigcap_{\theta \in T} \gamma_3^c(\theta)$$

$$\gamma^c(\emptyset) \stackrel{\text{def}}{=} \mathbb{S}$$

Program Types

- Galois connection:

$$\langle \mathbb{P}, \subseteq, \emptyset, \mathbb{S}, \cup, \cap \rangle \underset{\alpha^{\mathsf{c}}}{\overset{\gamma^{\mathsf{c}}}{\leftrightarrows}} \langle \mathbb{T}^{\mathsf{c}}, \supseteq, \mathbb{I}^{\mathsf{c}}, \emptyset, \cap, \cup \rangle$$

Program Types

- Galois connection:

$$\langle \mathbb{P}, \subseteq, \emptyset, \mathbb{S}, \cup, \cap \rangle \xrightleftharpoons[\alpha^{c}]{\gamma^{c}} \langle \mathbb{T}^{c}, \supseteq, \mathbb{I}^{c}, \emptyset, \cap, \cup \rangle$$

- Types $\mathbf{T}[\![e]\!]$ of an expression e:

$$\mathbf{T}[\![e]\!] \subseteq \alpha^{c}(\mathbf{C}[\![e]\!]) = \alpha^{c}(\{\mathbf{S}[\![e]\!]\})$$

Program Types

- Galois connection:

$$\langle \mathbb{P},\ \subseteq,\ \emptyset,\ \mathbb{S},\ \cup,\ \cap \rangle \xrightleftharpoons[\alpha^{\mathrm{c}}]{\gamma^{\mathrm{c}}} \langle \mathbb{T}^{\mathrm{c}},\ \supseteq,\ \mathbb{I}^{\mathrm{c}},\ \emptyset,\ \cap,\ \cup \rangle$$

- Types $\mathbf{T}[\![e]\!]$ of an expression e:

$$\mathbf{T}[\![e]\!] \subseteq \alpha^{\mathrm{c}}(\mathbf{C}[\![e]\!]) = \alpha^{\mathrm{c}}(\{\mathbf{S}[\![e]\!]\})$$

Typable Programs Cannot Go Wrong

$$\Omega \in \gamma^{\mathrm{c}}(\mathbf{T}[\![e]\!]) \quad \Longleftrightarrow \quad \mathbf{T}[\![e]\!] = \emptyset$$

Application to Model Checking

Objective of Model Checking

1) Built a model M of the computer system;

2) Check (i.e. prove enumeratively) that the model satisfies a specification given (as set of traces φ) by a (linear) temporal formula: $M \subseteq \varphi$ or $M \cap \varphi \neq \emptyset$;

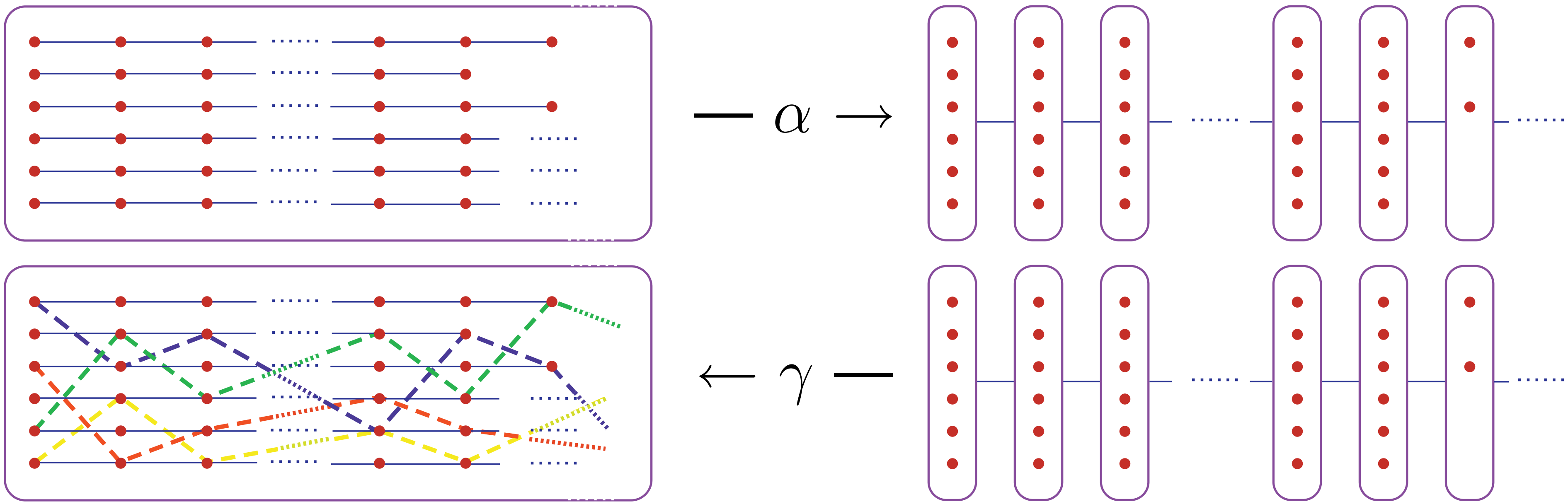
Objective of Model Checking

1) Built a model M of the computer system;
2) Check (i.e. prove enumeratively) that the model satisfies a specification given (as set of traces φ) by a (linear) temporal formula: $M \subseteq \varphi$ or $M \cap \varphi \neq \emptyset$;

Abstract interpretation is involved:

- To prove that the model and specification are correct abstractions of the computer system (often taken for granted);
- Checking is an abstraction;
- Soundness/completeness/refinement arguments.

Implicit Abstraction Involved in Model Checking



Spurious traces: ---, ---, ---, ---, ... ;

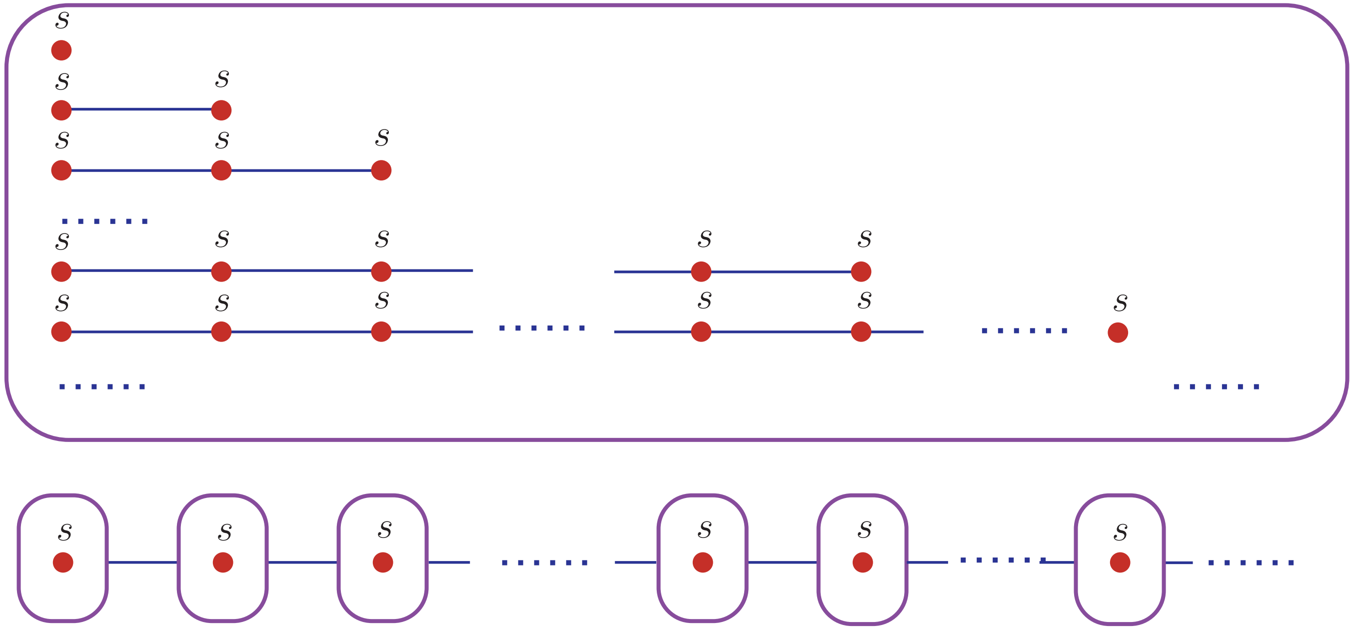
Kozen's μ-calculus is closed under this abstraction.

Soundness

For a *given class* of properties, soundness means that:

> Any property (in the *given class*) of the abstract world must hold in the concrete world;

Example for <u>Un</u>soundness



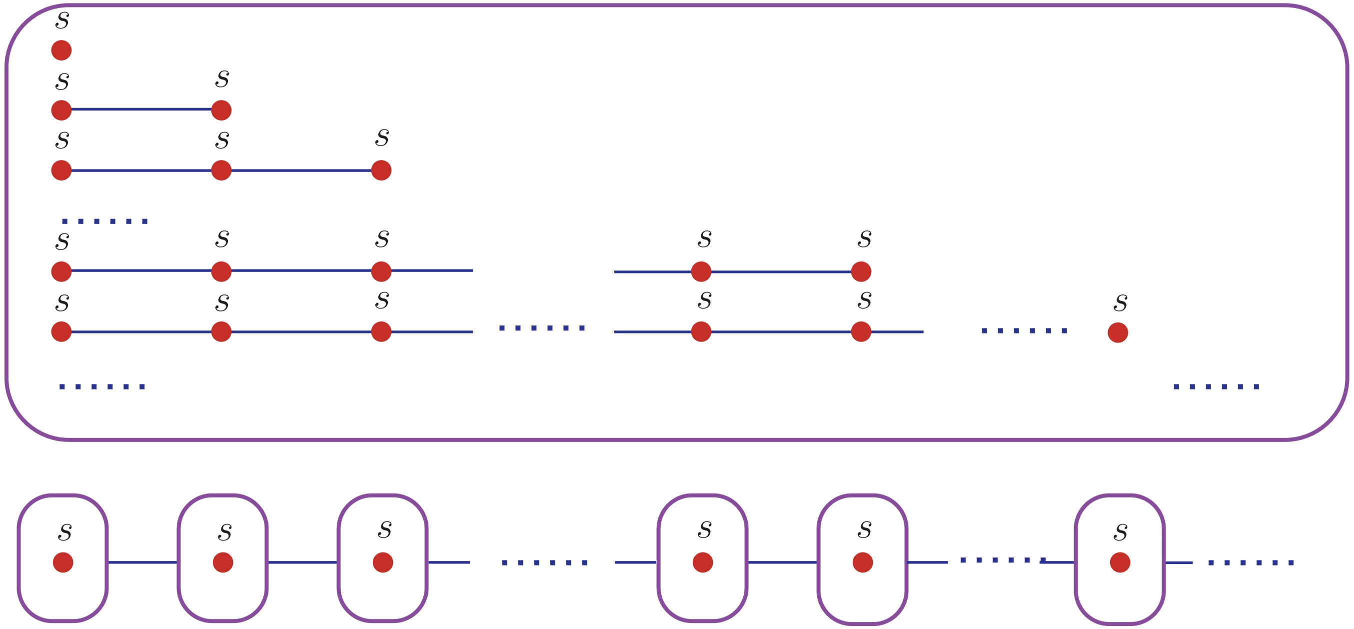
All abstract traces are infinite but not the concrete ones!

Completeness

For a *given class* of properties, completeness means that:

> Any property (in the *given class*) of the concrete world must hold in the abstract world;

Example for Incompleteness



All concrete traces are finite but not the abstract ones!

On the Soundness/Completeness of Model-Checking

- Model checking is sound and complete (for the model);

On the Soundness/Completeness of Model-Checking

- Model checking is sound and complete (for the model);
- This is due to restrictions on the models and specifications (e.g. closure under the implicit abstractions);

On the Soundness/Completeness of Model-Checking

- Model checking is sound and complete (for the model);
- This is due to restrictions on the models and specifications (e.g. closure under the implicit abstractions);
- There are models/specifications (bidirectional traces) for which:
  - The implicit abstraction is incomplete (POPL'00),
  - Any abstraction is incomplete (Ranzato, Esop'01).

Application to Static Program Analysis

What is static program analysis?

- Automatic static/compile time determination of dynamic/run-time properties of programs;

What is static program analysis?

- Automatic static/compile time determination of dynamic/run-time properties of programs;
- **Basic idea:** use effective computable approximations of the program semantics;

  Advantage: fully automatic, no need for error-prone user designed model or costly user interaction;

  Drawback: can only handle properties captured by the approximation.

Static Analysis

- Objective: automatically extract information on the runtime behavior of a program from its text;

Static Analysis

- Objective: automatically extract information on the runtime behavior of a program from its text;
- Method: use abstract interpretation to derive an abstract semantics which is effectively computable by a computer (from the standard semantics);

Static Analysis

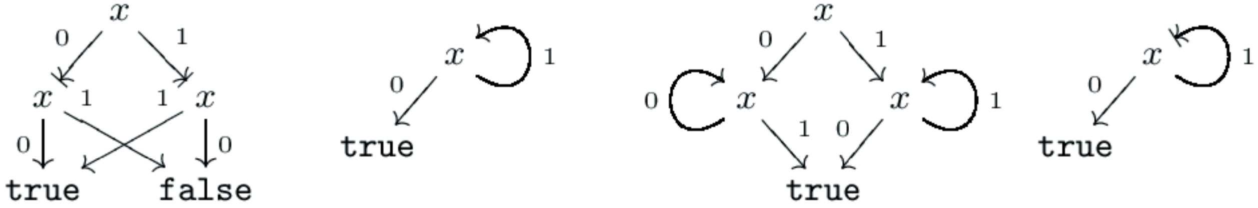
- Objective: automatically extract information on the runtime behavior of a program from its text;
- Method: use abstract interpretation to derive an abstract semantics which is effectively computable by a computer (from the standard semantics);
- Application: analyze the behavior of software before executing it in the real world;

Static Analysis

- Objective: automatically extract information on the runtime behavior of a program from its text;
- Method: use abstract interpretation to derive an abstract semantics which is effectively computable by a computer (from the standard semantics);
- Application: analyze the behavior of software before executing it in the real world;
- Usefulness: essential for safety critical software (as found in planes, launchers, nuclear plants, ...).

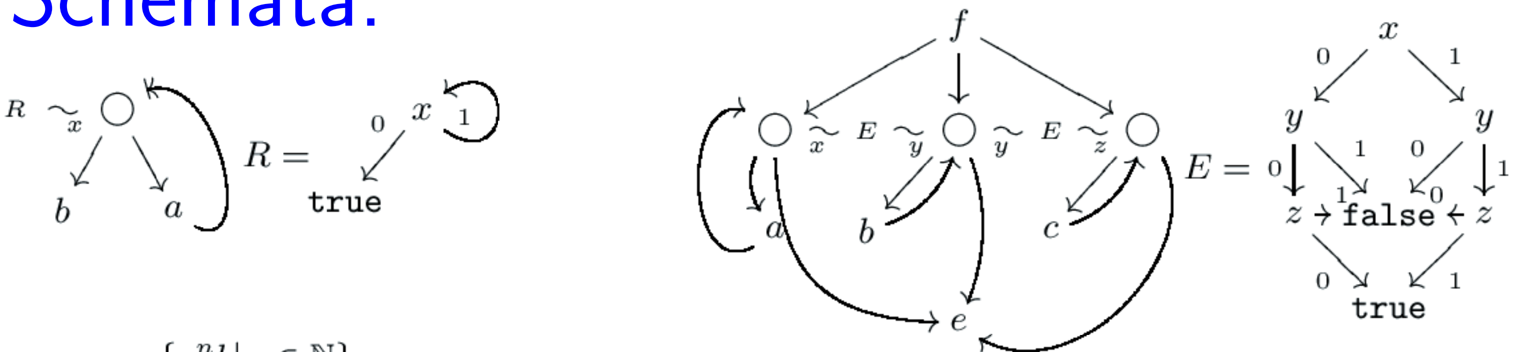
Example of Effective Abstractions of Infinite Sets of Infinite Trees

Binary Decision Graphs:



$\{0^\omega, 1^\omega\}$ infinite number of 0's fair vectors ends by 0^ω

Tree Schemata:

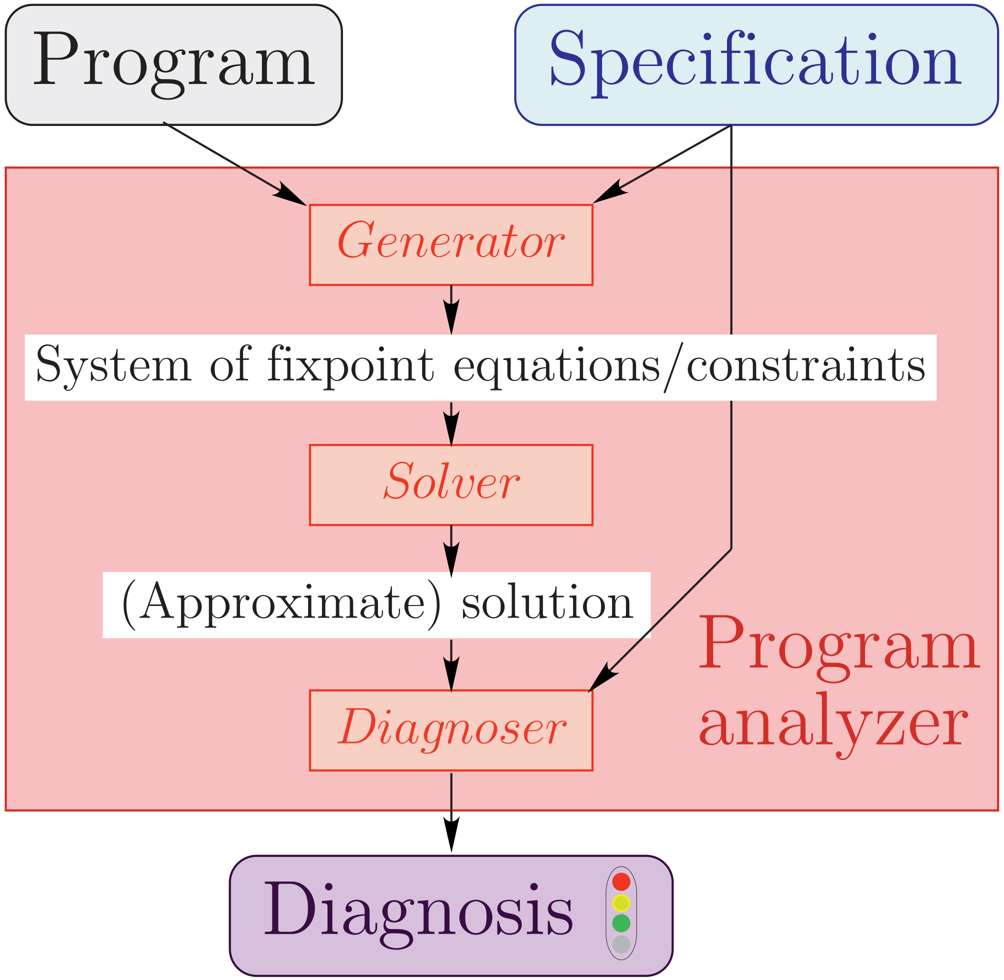


$\{a^n b | n \in \mathbb{N}\}$

$\{f(a^n e, b^n e, c^n e) | n \in \mathbb{N}\}$

Note that E is the equality relation.

Principle of Verification by Static Analysis



Example: Interval Analysis (1975)

Program to be analyzed:

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

Example: Interval Analysis (1975)

Equations (abstract interpretation of the semantics):

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$
\begin{cases}
X_1 = [1, 1] \\
X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\
X_3 = X_2 \oplus [1, 1] \\
X_4 = (X_1 \cup X_3) \cap [10000, +\infty]
\end{cases}
$$

Example: Interval Analysis (1975)

Resolution by chaotic increasing iteration:

```
    x := 1;
1:
    while x < 10000 do
2:
        x := x + 1
3:
    od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = \emptyset \\ X_2 = \emptyset \\ X_3 = \emptyset \\ X_4 = \emptyset \end{cases}$$

Example: Interval Analysis (1975)

Increasing chaotic iteration:

```
    x := 1;
1:
    while x < 10000 do
2:
        x := x + 1
3:
    od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = \emptyset \\ X_3 = \emptyset \\ X_4 = \emptyset \end{cases}$$

Example: Interval Analysis (1975)

Increasing chaotic iteration:

```
    x := 1;
1:
    while x < 10000 do
2:
        x := x + 1
3:
    od;
4:
```

$$\begin{cases} X_1 = [1, 1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1, 1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1, 1] \\ X_2 = [1, 1] \\ X_3 = \emptyset \\ X_4 = \emptyset \end{cases}$$

Example: Interval Analysis (1975)

Increasing chaotic iteration:

```
    x := 1;
1:
    while x < 10000 do
2:
        x := x + 1
3:
    od;
4:
```

$$\begin{cases} X_1 = [1, 1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1, 1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1, 1] \\ X_2 = [1, 1] \\ X_3 = [2, 2] \\ X_4 = \emptyset \end{cases}$$

Example: Interval Analysis (1975)

Increasing chaotic iteration:

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1, 1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1, 1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1, 1] \\ X_2 = [1, 2] \\ X_3 = [2, 2] \\ X_4 = \emptyset \end{cases}$$

Example: Interval Analysis (1975)

Increasing chaotic iteration: **convergence !**

```
   x := 1;
1:
   while x < 10000 do
2:
         x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1, 1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1, 1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1, 1] \\ X_2 = [1, 2] \\ X_3 = [2, 3] \\ X_4 = \emptyset \end{cases}$$

Example: Interval Analysis (1975)

Increasing chaotic iteration: **convergence !!**

```
    x := 1;
1:
    while x < 10000 do
2:
        x := x + 1
3:
    od;
4:
```

$$\begin{cases} X_1 = [1, 1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1, 1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1, 1] \\ X_2 = [1, 3] \\ X_3 = [2, 3] \\ X_4 = \emptyset \end{cases}$$

Example: Interval Analysis (1975)

Increasing chaotic iteration: **convergence !!!**

```
   x := 1;
1:
   while x < 10000 do
2:
      x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1, 1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1, 1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1, 1] \\ X_2 = [1, 3] \\ X_3 = [2, 4] \\ X_4 = \emptyset \end{cases}$$

Example: Interval Analysis (1975)

Increasing chaotic iteration: **convergence !!!!**

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1,4] \\ X_3 = [2,4] \\ X_4 = \emptyset \end{cases}$$

Example: Interval Analysis (1975)

Increasing chaotic iteration: **convergence !!!!!**

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1,4] \\ X_3 = [2,5] \\ X_4 = \emptyset \end{cases}$$

Example: Interval Analysis (1975)

Increasing chaotic iteration: **convergence !!!!!!**

```
    x := 1;
1:
    while x < 10000 do
2:
        x := x + 1
3:
    od;
4:
```

$$\begin{cases} X_1 = [1, 1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1, 1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1, 1] \\ X_2 = [1, 5] \\ X_3 = [2, 5] \\ X_4 = \emptyset \end{cases}$$

Example: Interval Analysis (1975)

Increasing chaotic iteration: **convergence !!!!!!!**

```
    x := 1;
1:
    while x < 10000 do
2:
        x := x + 1
3:
    od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1,5] \\ X_3 = [2,6] \\ X_4 = \emptyset \end{cases}$$

Example: Interval Analysis (1975)

Convergence speed-up by widening:

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1, 1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1, 1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1, 1] \\ X_2 = [1, +\infty] \quad \Leftarrow \text{widening} \\ X_3 = [2, 6] \\ X_4 = \emptyset \end{cases}$$

Example: Interval Analysis (1975)

Decreasing chaotic iteration:

```
    x := 1;
1:
    while x < 10000 do
2:
        x := x + 1
3:
    od;
4:
```

$$\begin{cases} X_1 = [1, 1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1, 1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1, 1] \\ X_2 = [1, +\infty] \\ X_3 = [2, +\infty] \\ X_4 = \emptyset \end{cases}$$

Example: Interval Analysis (1975)

Decreasing chaotic iteration:

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1, 1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1, 1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1, 1] \\ X_2 = [1, 9999] \\ X_3 = [2, +\infty] \\ X_4 = \emptyset \end{cases}$$

Example: Interval Analysis (1975)

Decreasing chaotic iteration:

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1, 9999] \\ X_3 = [2, +10000] \\ X_4 = \emptyset \end{cases}$$

Example: Interval Analysis (1975)

Final solution:

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1, 9999] \\ X_3 = [2, +10000] \\ X_4 = [+10000, +10000] \end{cases}$$

Example: Interval Analysis (1975)

Result of the interval analysis:

```
   x := 1;
1: {x = 1}
   while x < 10000 do
2: {x ∈ [1, 9999]}
         x := x + 1
3: {x ∈ [2, +10000]}
   od;
4: {x = 10000}
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1, 9999] \\ X_3 = [2, +10000] \\ X_4 = [+10000, +10000] \end{cases}$$

Example: Interval Analysis (1975)

Checking absence of runtime errors with interval analysis:

```
   x := 1;
1: {x = 1}
   while x < 10000 do
2: {x ∈ [1, 9999]}
       x := x + 1          ←— no overflow
3: {x ∈ [2, +10000]}
   od;
4: {x = 10000}
```

Application to Abstract Program Testing

Static Analysis

- Static analysis: specification derived automatically from the program (e.g. using the language specification for run-time errors);

Static Analysis versus Abstract Testing

- Static analysis: specification derived automatically from the program (e.g. using the language specification for run-time errors);
- Abstract testing: specification given by the programmer.

A tiny example

```
read(n);
f := 1;
while (n <> 0) do
    f := (f * n);
    n := (n - 1)
od;
```

- user program

A tiny example

```
read(n);
f := 1;
while (n <> 0) do
    f := (f * n);
    n := (n - 1)
od;
sometime true;;
```

- user program
- user specification

A tiny example

```
0: { n:[−∞,+∞]?; f:[−∞,+∞]? }
  read(n);
1: { n:[0,+∞]; f:[−∞,+∞]? }
  f := 1;
2: { n:[0,+∞]; f:[1,+∞] }
  while (n <> 0) do
    3: { n:[1,+∞]; f:[1,+∞] }
      f := (f * n);
    4: { n:[1,+∞]; f:[1,+∞] }
      n := (n - 1)
    5: { n:[0,+∞]; f:[1,+∞] }
  od;
6: { n:[0,0]; f:[1,+∞] }
sometime true;;
```

- static analyzer inference
- user program
- user specification

A tiny example

```
0: { n:[−∞,+∞]?; f:[−∞,+∞]? }
  read(n);
1: { n:[0,+∞]; f:[−∞,+∞]? }
  f := 1;
2: { n:[0,+∞]; f:[1,+∞] }
  while (n <> 0) do
    3: { n:[1,+∞]; f:[1,+∞] }
      f := (f * n);
    4: { n:[1,+∞]; f:[1,+∞] }
      n := (n - 1)
    5: { n:[0,+∞]; f:[1,+∞] }
  od;
6: { n:[0,0]; f:[1,+∞] }
sometime true;;
```

- static analyzer inference
- definite error
- no error
- potential error
- user program
- user specification

Which properties can be handled? (examples)

Invariance/set of states properties: absence of runtime errors (overflows, division by zero, null pointer dereferencing, etc);

Which properties can be handled? (examples)

Invariance/set of states properties: absence of runtime errors (overflows, division by zero, null pointer dereferencing, etc);

Trace properties: accuracy of floating point computations, inevitable reaction to events, properties specified by the CTL temporal logic/the μ-calculus;

Which properties can be handled? (examples)

Invariance/set of states properties: absence of runtime errors (overflows, division by zero, null pointer dereferencing, etc);

Trace properties: accuracy of floating point computations, inevitable reaction to events, properties specified by the CTL temporal logic/the μ-calculus;

Temporal properties: termination, execution time, etc;

Types of analyzers

- Universal analyzers: based on general purpose approximations of wide spectrum properties to check common specifications for widely used programming languages (e.g. absence of run-time errors in C/ADA);

Types of analyzers

- Universal analyzers: based on general purpose approximations of wide spectrum properties to check common specifications for widely used programming languages (e.g. absence of run-time errors in C/ADA);
- Special purpose analyzers: based on specific approximations of problem specific properties to check user oriented specifications for a well-defined application (e.g. execution time on a given computer).

Conclusions and References

Conclusion

Future Objectives:

- Abstract interpretation as a thinking tool: a basis for reasoning about programs (from semantics to compilation, ...);
- Abstract interpretation applied to mechanical tools: scale up for large-scale industrialization;

Short Introductive Survey on Abstract Interpretation (with Numerous References)

[4] P. Cousot. Abstract interpretation based formal methods and future challenges. In R. Wilhelm, editor, « *Informatics — 10 Years Back, 10 Years Ahead* », volume 2000 of *LNCS*, pages 138–156. Springer-Verlag, 2001.

THE END