Abstract Interpretation Based Program Transformation

Patrick COUSOT

École Normale Supérieure

45 rue d'Ulm

75230 Paris cedex 05, France

Patrick.Cousot@ens.fr

www.di.ens.fr/~cousot

Radhia COUSOT

École Polytechnique

91128 Palaiseau cedex

France

Radhia.Cousot@lix.polytechnique.fr

lix.polytechnique.fr/~rcousot

CLIP Lab. at DIA & DLSIIS/FIM/UPM   Sep 12, 2002

Content

Abstract Interpretation

Abstract Interpretation

- Formalizes the idea of approximation of sets and set operations as considered in set (or category) theory;

- Mainly applied to the approximation of the semantics of programming languages/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;

  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);

- **Mechanical tools**: the idea of effective approximation leads to automatic semantics-based program manipulation tools.

Abstraction

# Abstraction: intuition

- Abstract interpretation formalizes the intuitive idea that a semantics is more or less precise according to the considered observation level of the program executions;

- Abstract interpretation theory formalizes this notion of approximation/abstraction in a mathematical setting which is independent of particular applications.

Intuition behind abstraction

An [in]finite set of points;
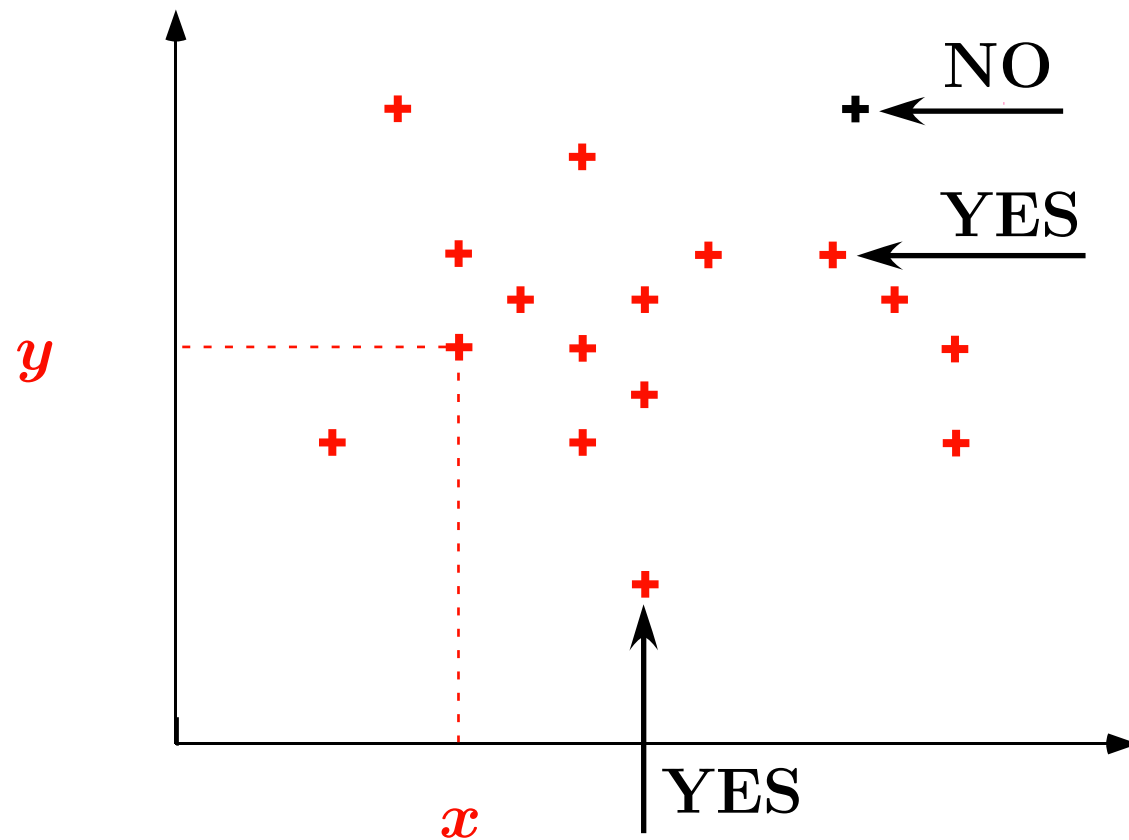
Is that point in the concrete set?

$$\{\ldots, \langle 19,\ 77 \rangle, \ldots, \langle 20,\ 02 \rangle, \ldots\}$$

Approximation of an [in]finite set of points:

From Below

Is that point in the concrete set?

$$\{\ldots, \langle 19, \, 77 \rangle, \ldots, \ldots\}$$
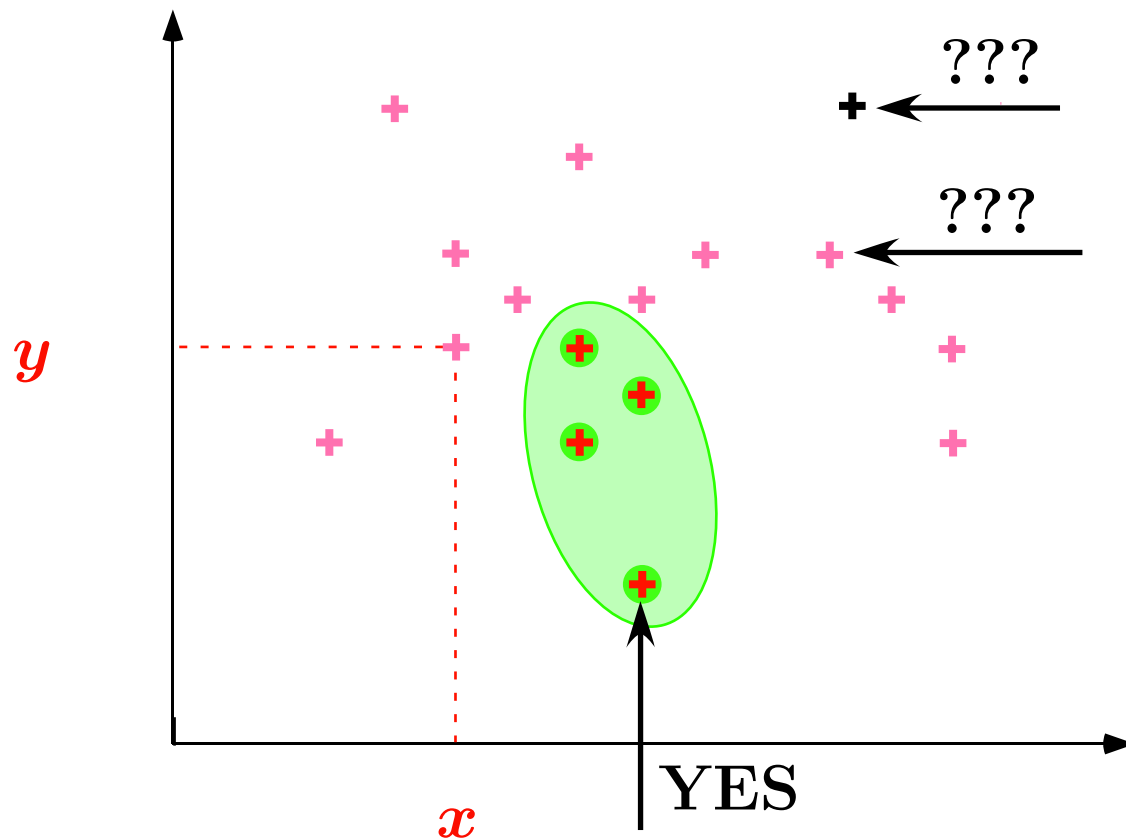
Conservative answer

???

???

$+$

y

YES

x

Approximation of an [in]finite set of points:

From Above

Is that point in the concrete set?

$$\{\ldots, \langle 19,\ 77\rangle, \ldots,$$

$$\langle 20,\ 02\rangle, \langle ?,\ ?\rangle, \ldots\}$$

Conservative answer
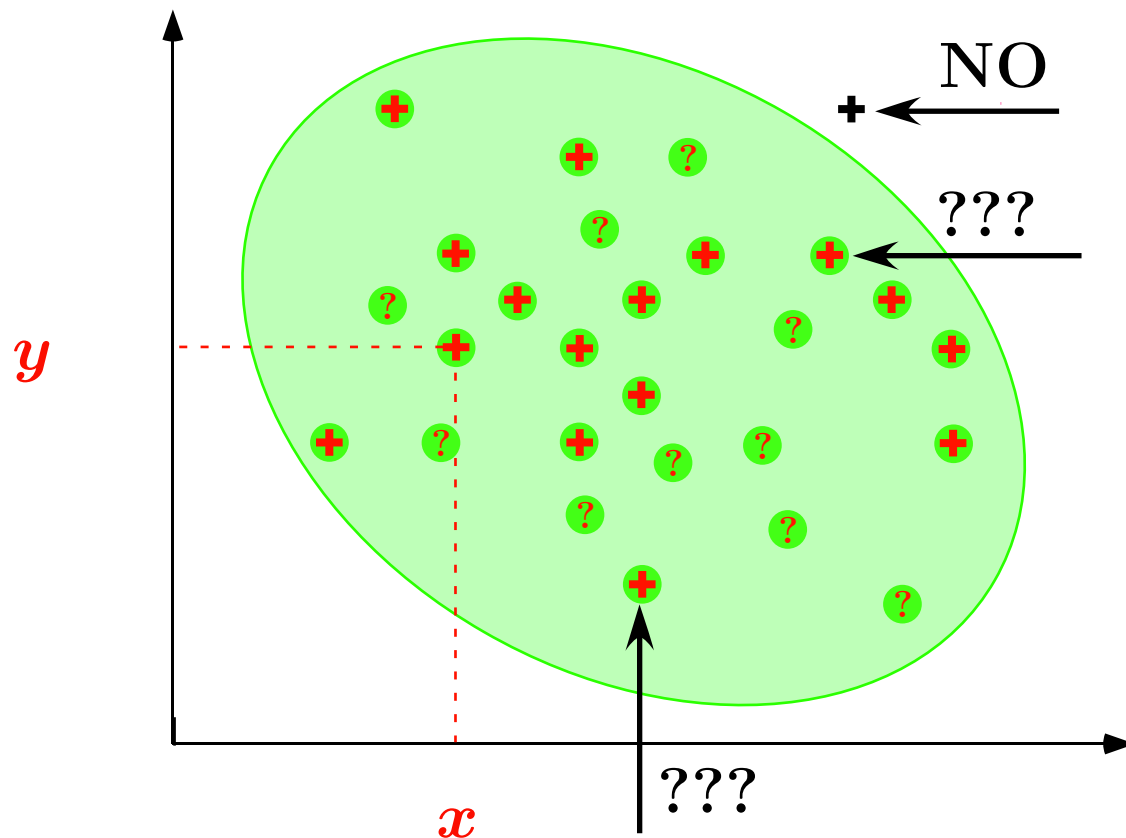
Intuition Behind
Effective Computable Abstraction

Effective computable approximations of an [in]finite set of points; Signs [1]

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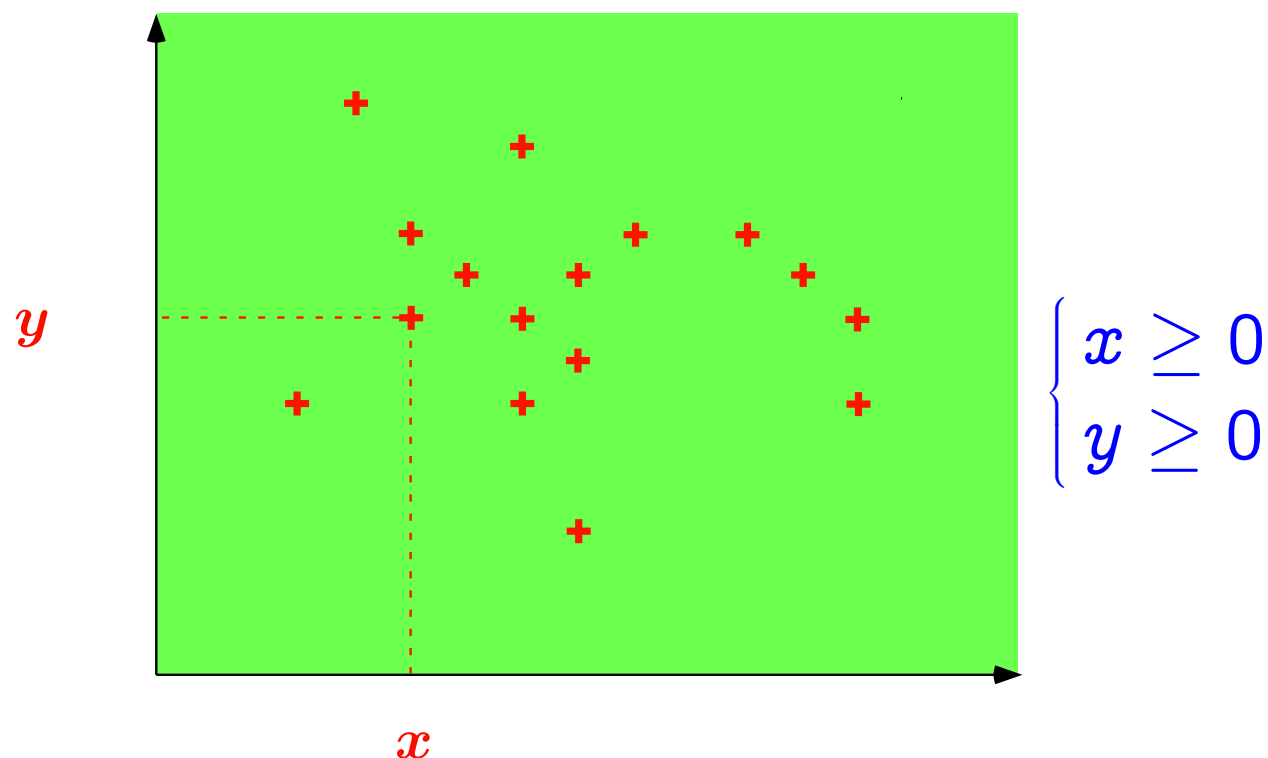
Reference

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

Effective computable approximations of an [in]finite set of points; Intervals [2]

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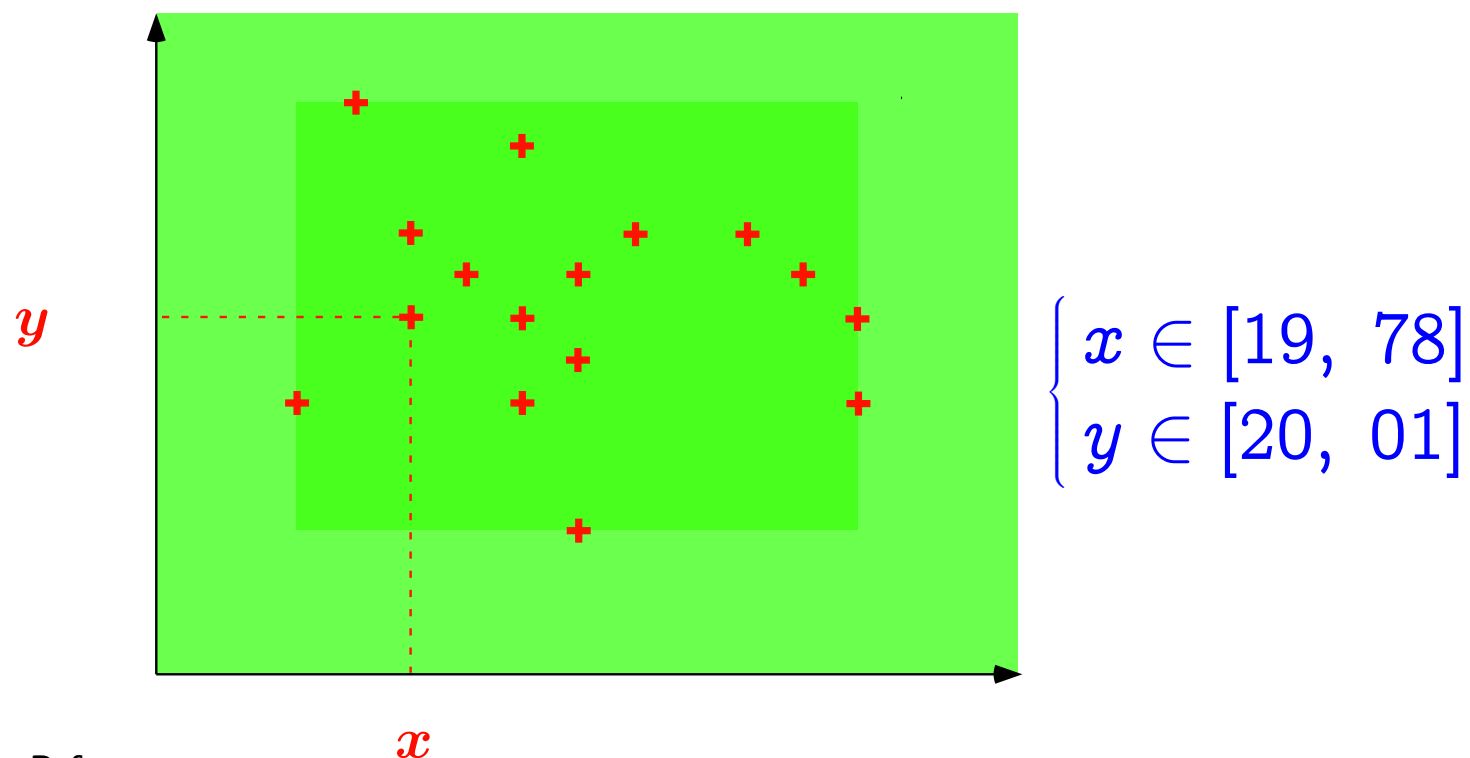
Reference

[2] P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In 2nd Int. Symp. on Programming, pages 106–130. Dunod, 1976.

Effective computable approximations of an [in]finite set of points; Octagons [3]

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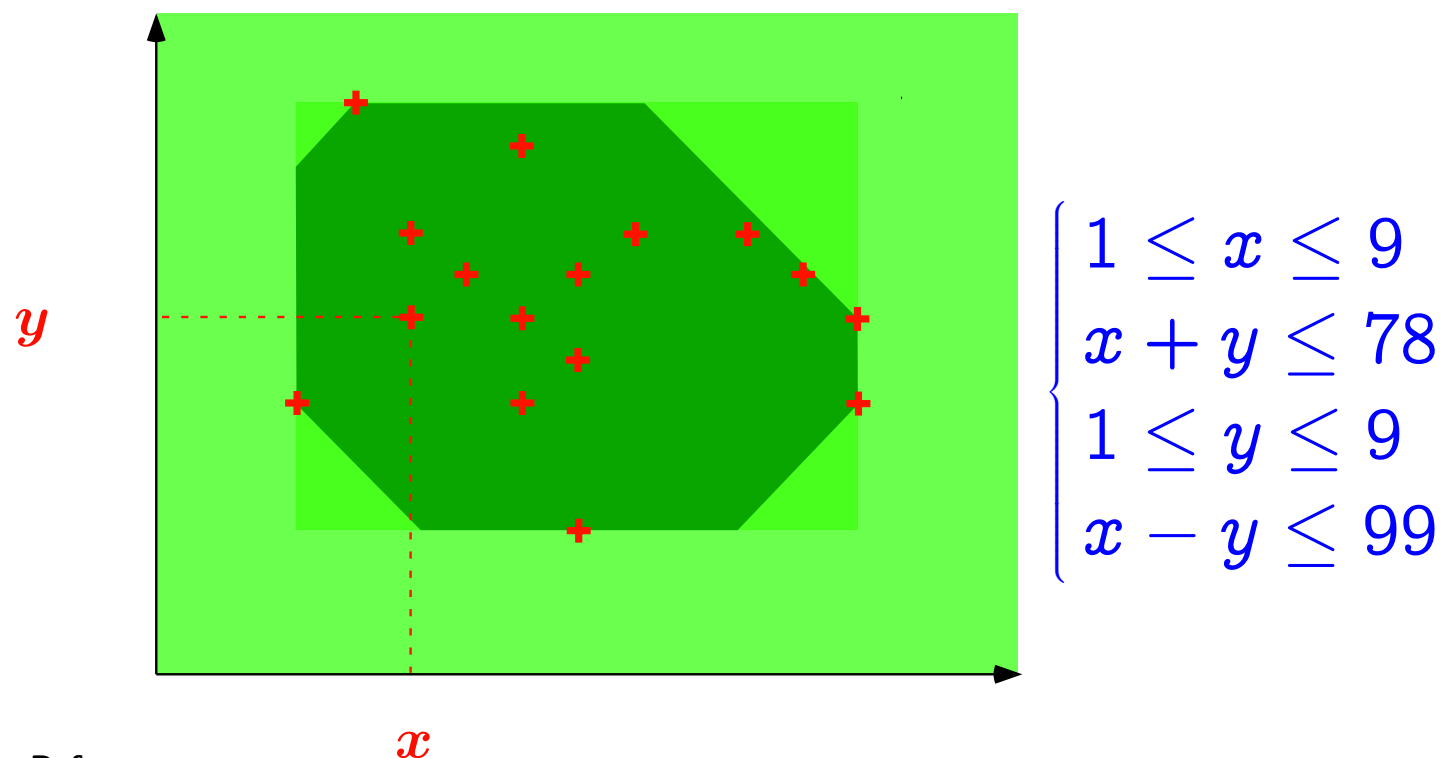
Reference

[3] A. Miné. A New Numerical Abstract Domain Based on Difference-Bound Matrices. In *PADO'2001*, LNCS 2053, Springer, 2001, pp. 155–172.

Effective computable approximations of an [in]finite set of points; Polyhedra [4]
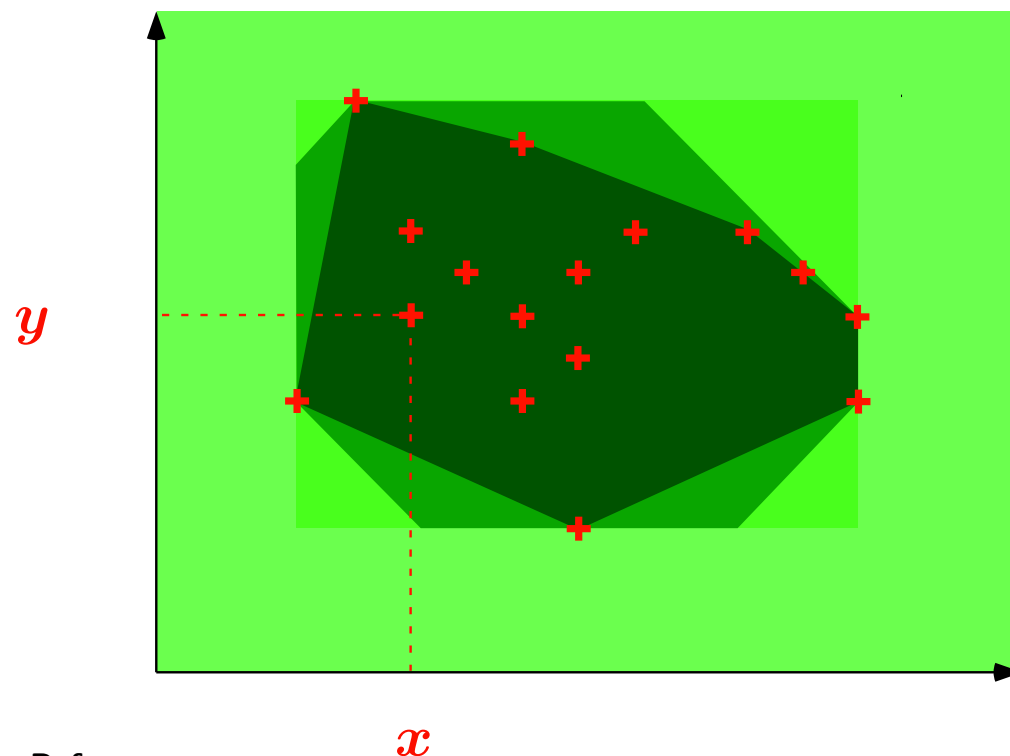
$$\begin{cases} 19x + 78y \leq 2000 \\ 20x + 01y \geq 0 \end{cases}$$

Reference

[4] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In 5th POPL, pages 84–97, Tucson, AZ, 1978. ACM Press.

Effective computable approximations of an [in]finite set of points; Simple congruences [5]

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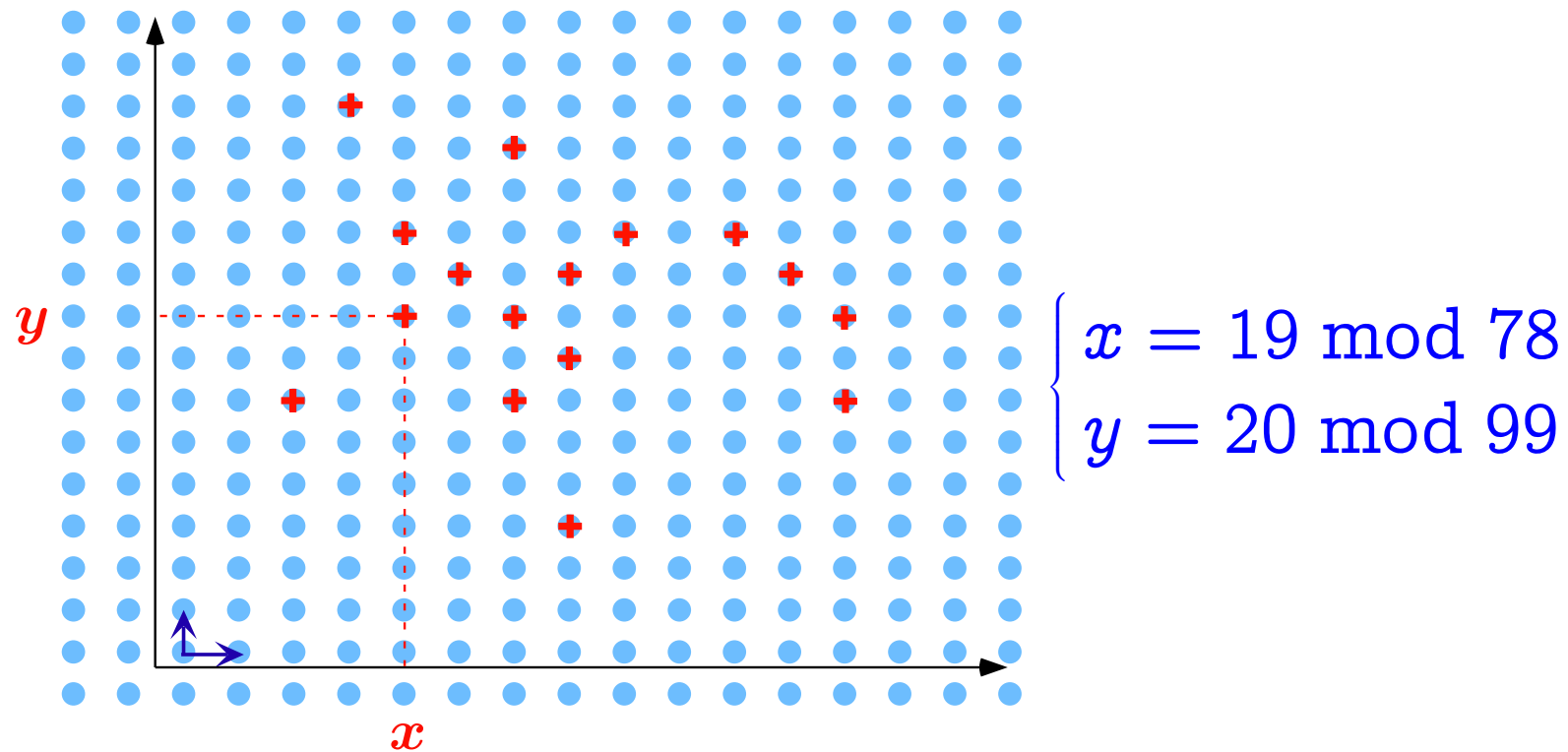
Reference

[5] P. Granger. Static analysis of arithmetical congruences. *Int. J. Comput. Math.*, 30:165–190, 1989.

Effective computable approximations of an [in]finite set of points; Linear congruences [6]

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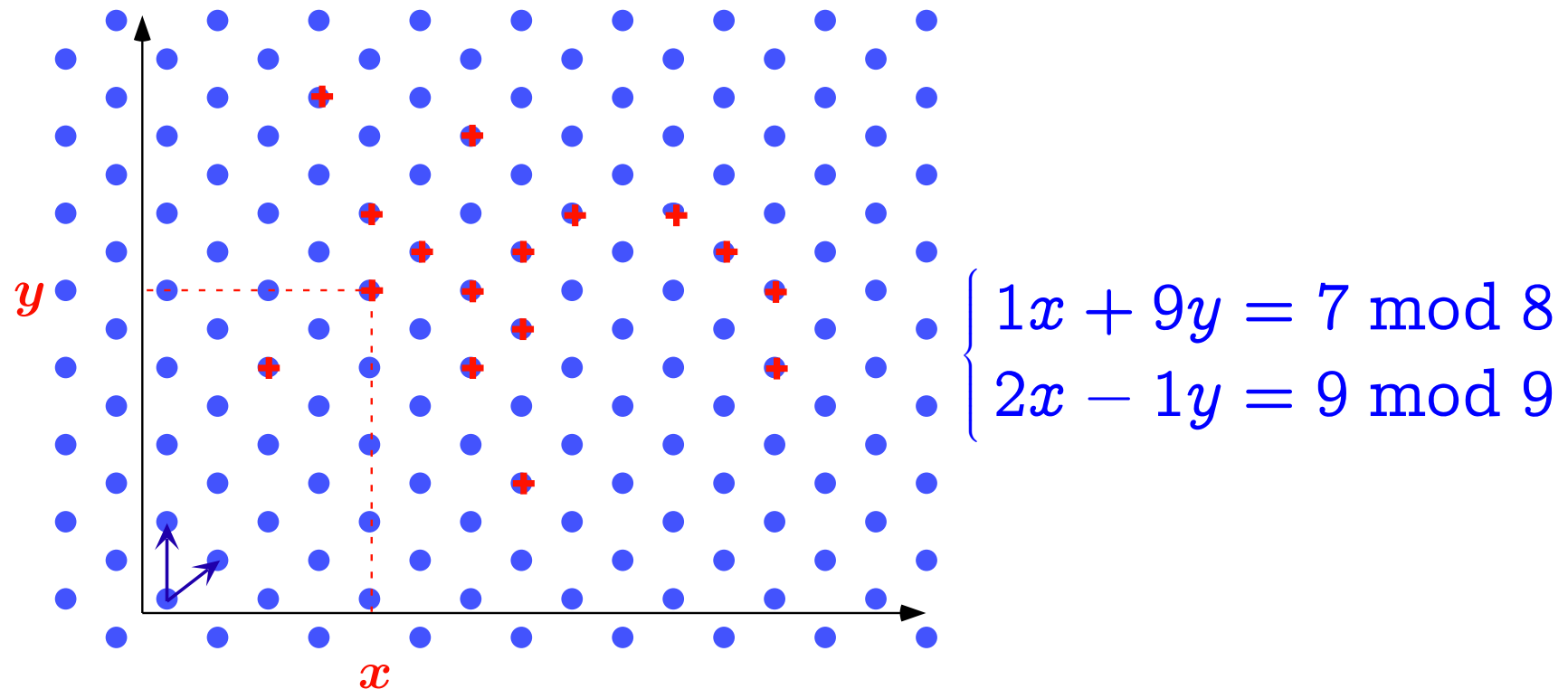
Reference

[6] P. Granger. Static analysis of linear congruence equalities among variables of a program. *CAAP '91*, LNCS 493, pp. 169–192. Springer, 1991.

Effective computable approximations of an [in]finite set of points; Trapezoidal linear congruences [7]

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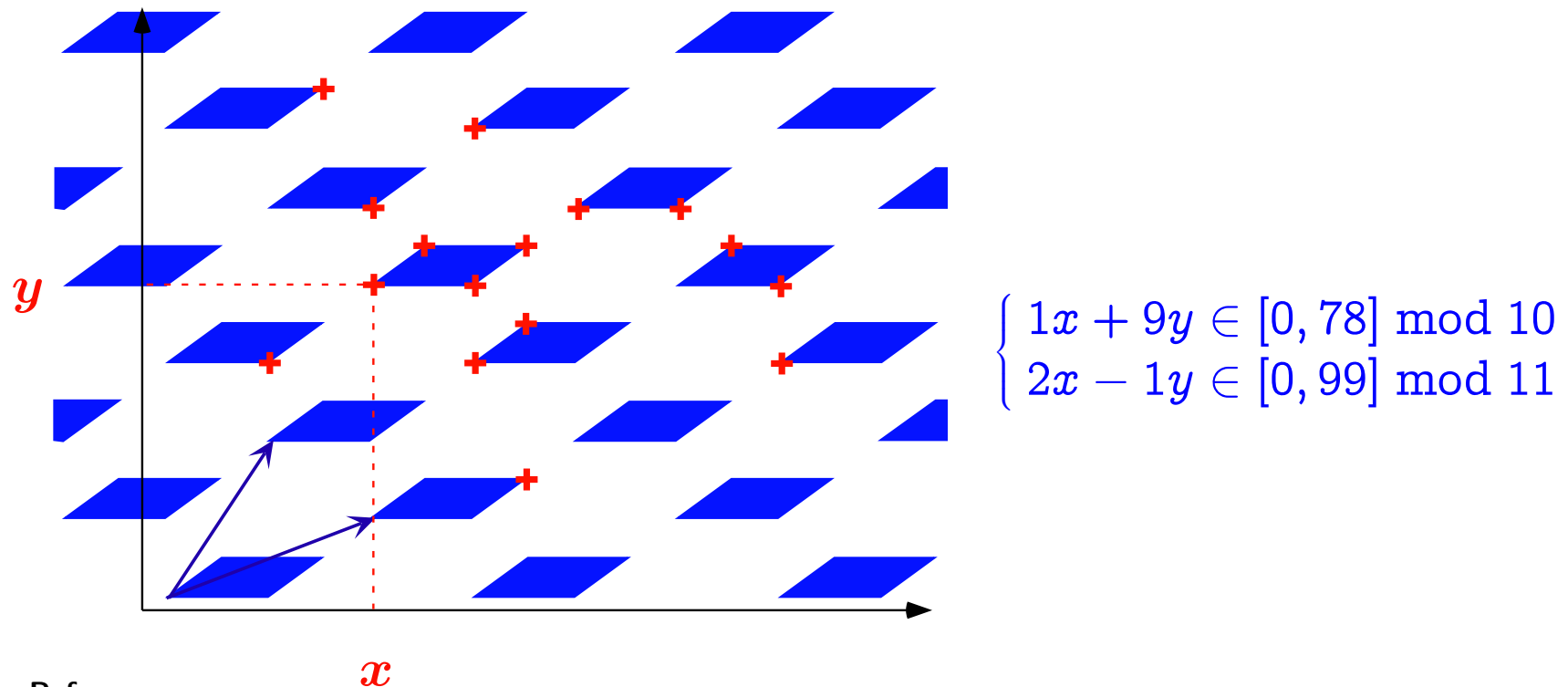
Reference

[7] F. Masdupuy. Array operations abstraction using semantic analysis of trapezoid congruences. In *ACM Int. Conf. on Supercomputing, ICS '92*, pages 226–235, 1992.

Conservative Approximation and Information Loss

Intuition Behind Sound/Conservative Approximation

Conservative Approximation

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

- Concrete semantics: yes
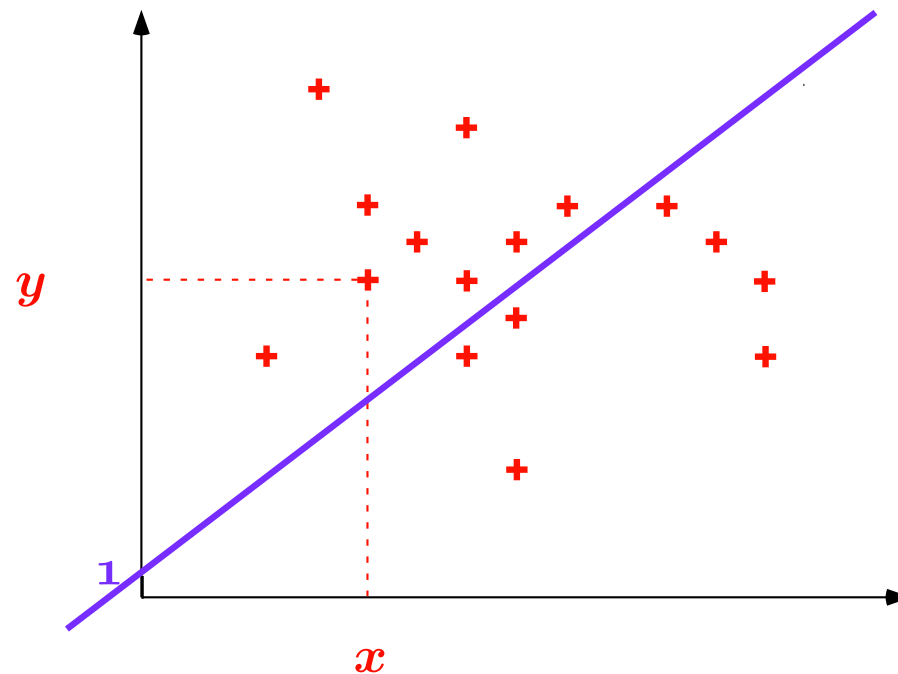
Conservative Approximation

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

- Testing : **You <u>never</u> know!**
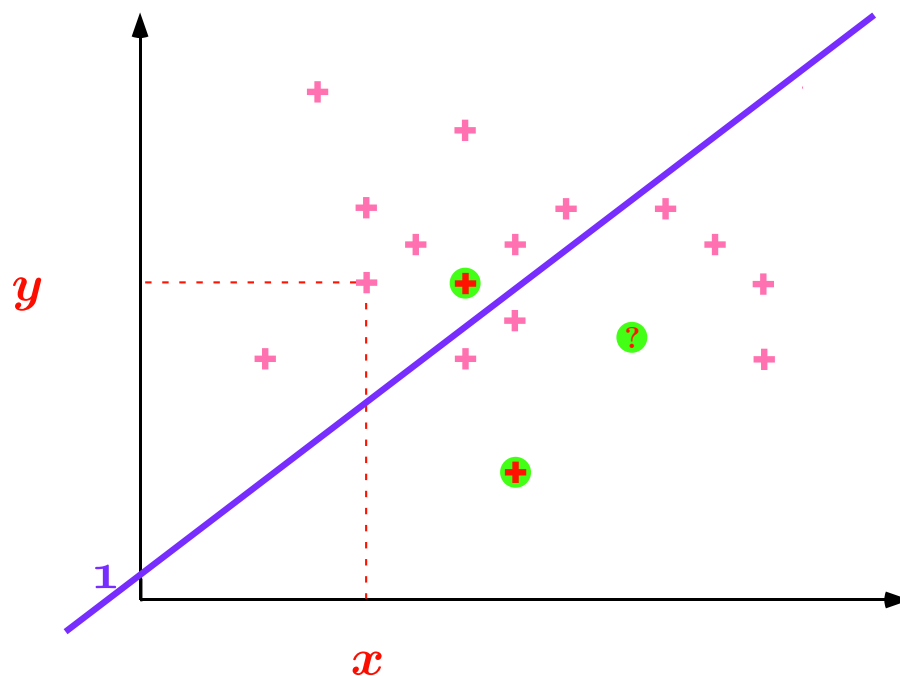
Conservative Approximation

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

- Abstract semantics 1: **I don't know**
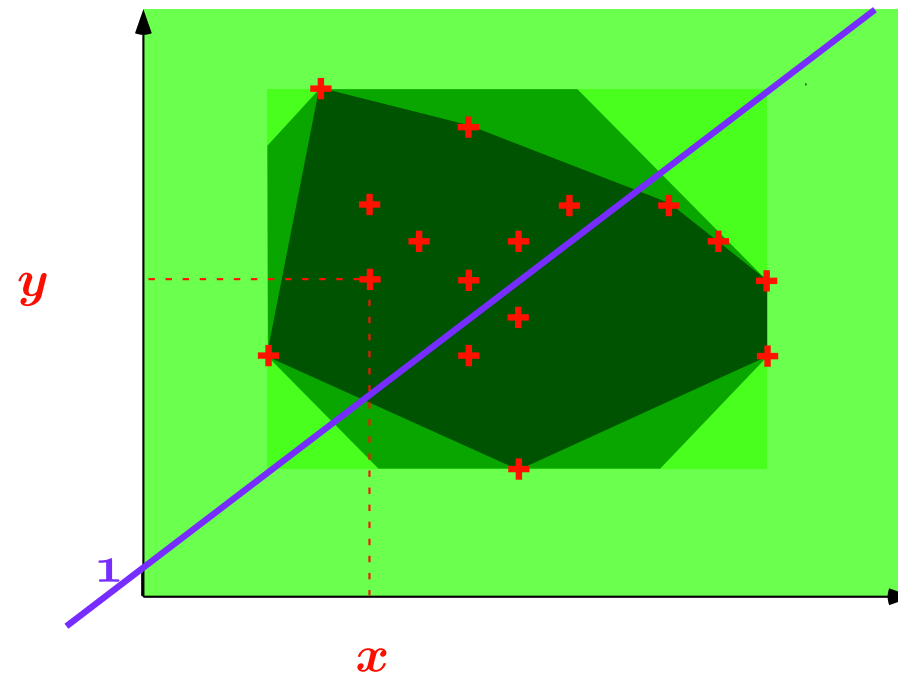
Conservative Approximation

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

- Abstract semantics 2: **yes**
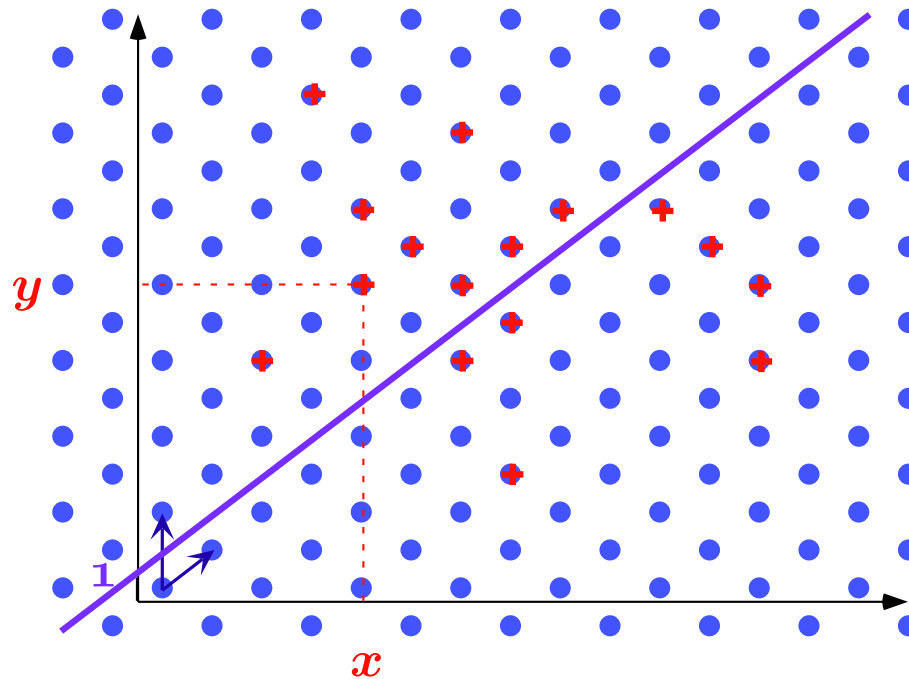
Intuition Behind Information Loss

Information Loss

- All answers given by the abstract semantics are always correct with respect to the concrete semantics;

- Because of the information loss, not all questions can be definitely answered with the abstract semantics;

- The more concrete semantics can answer more questions;

- The more abstract semantics are more simple.

Very Basic Elements of
Abstract Interpretation Theory

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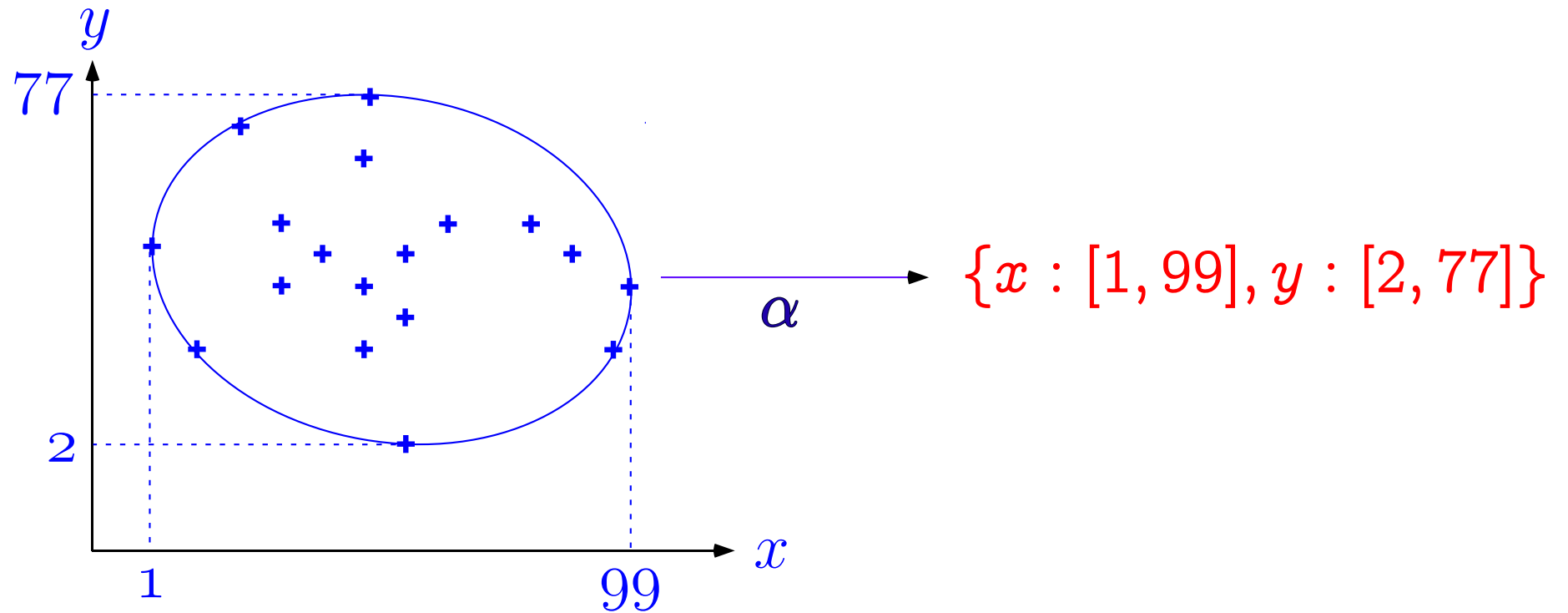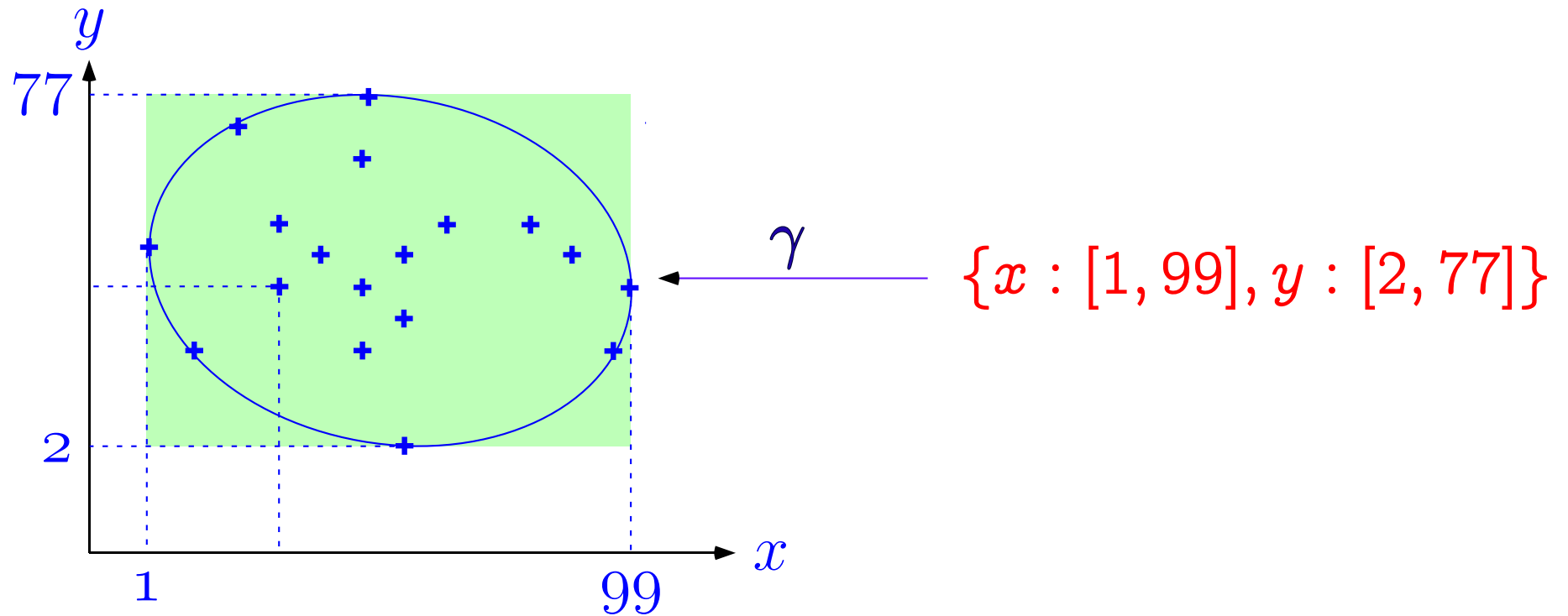
$$\{x : [1, 99], y : [2, 77]\}$$

Concretization γ

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

The Abstraction α is Monotone

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

$\sqsubseteq$

$\{x : [1, 99], y : [2, 90]\}$

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

The Concretization γ is Monotone

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

$\sqsubseteq$

$\{x : [1, 99], y : [2, 90]\}$

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

The $\gamma \circ \alpha$ Composition

$$\{x : [1, 99], y : [2, 77]\}$$
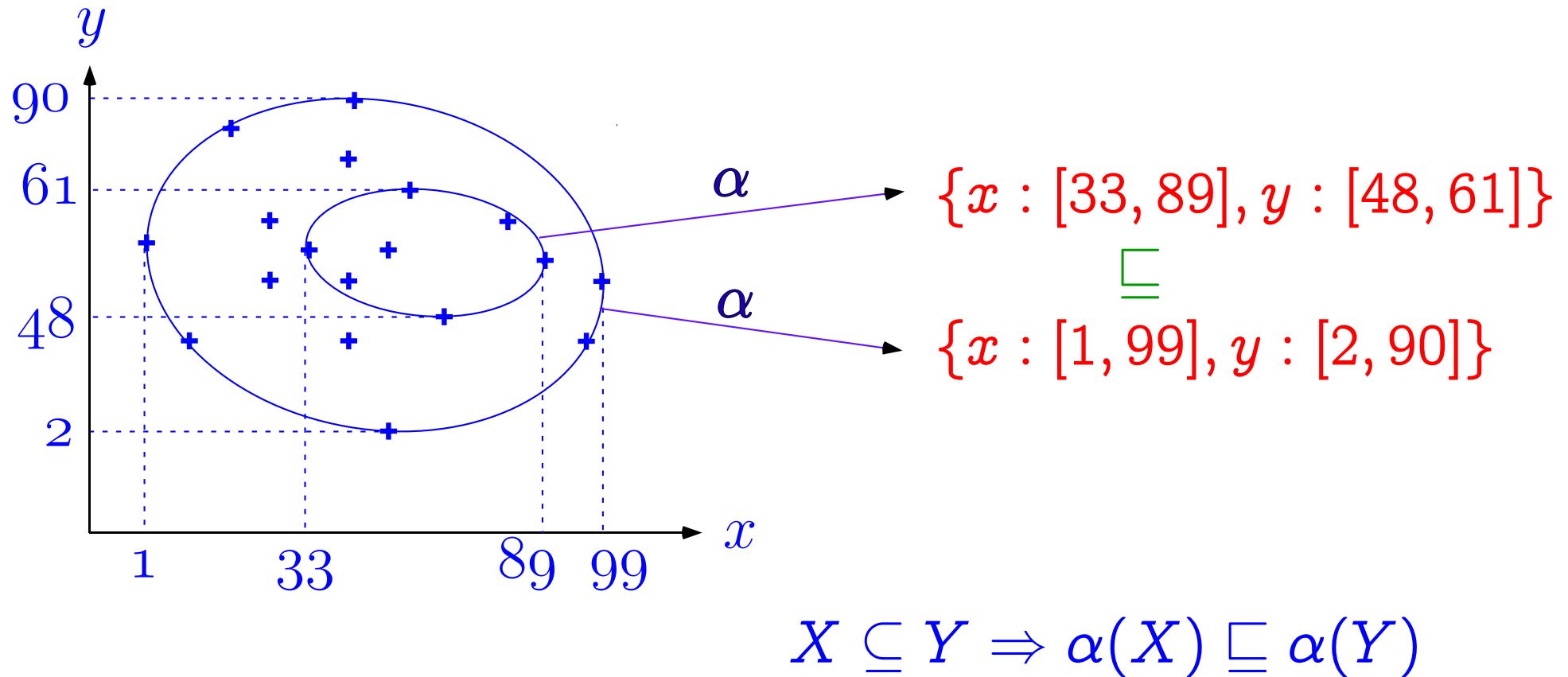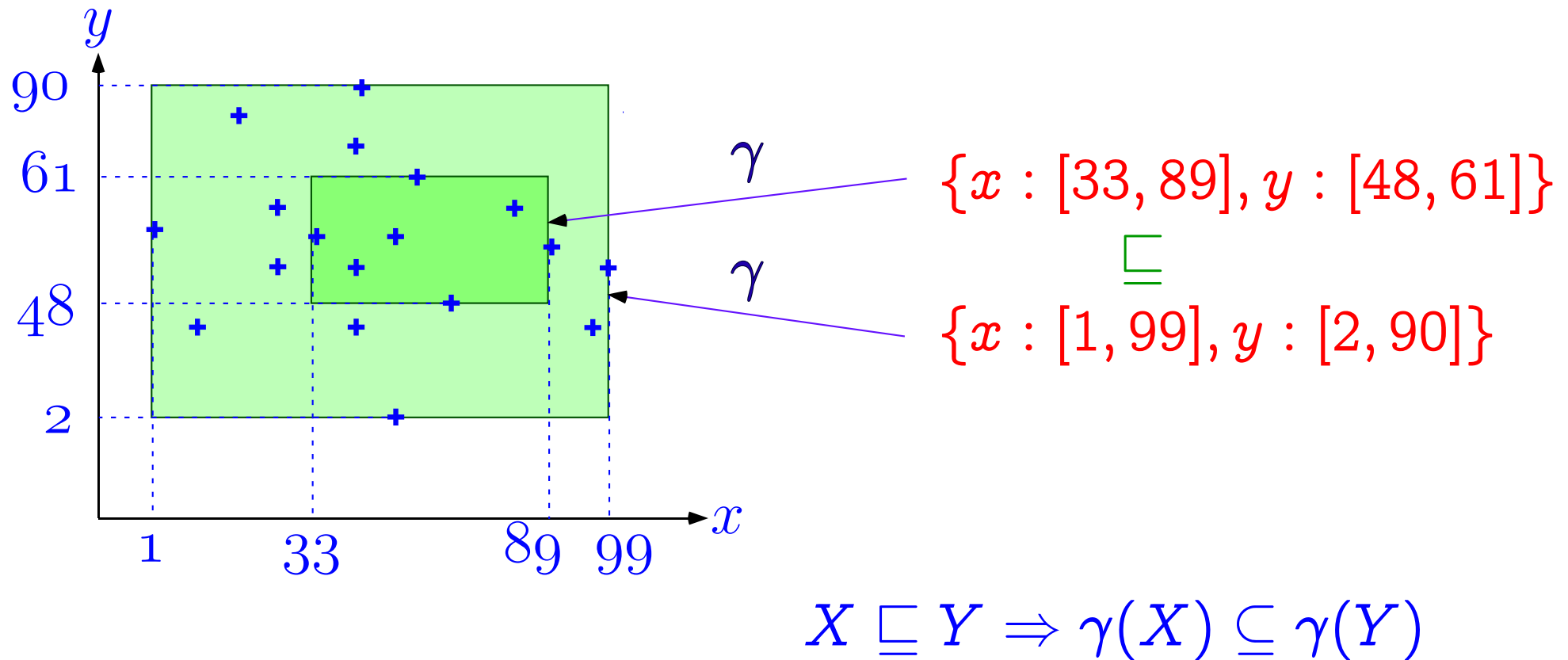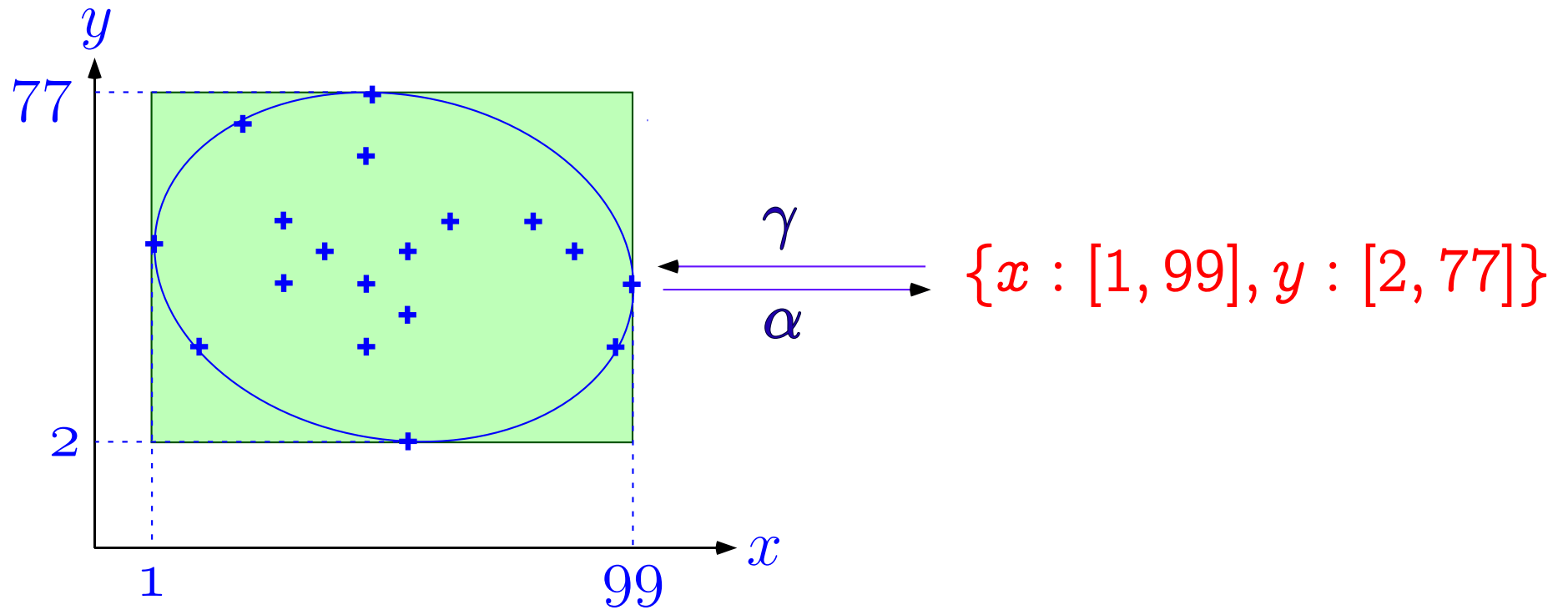
$$X \subseteq \gamma \circ \alpha(X)$$

The $\alpha \circ \gamma$ Composition

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

γ

α

$$\alpha \circ \gamma(Y) = Y$$
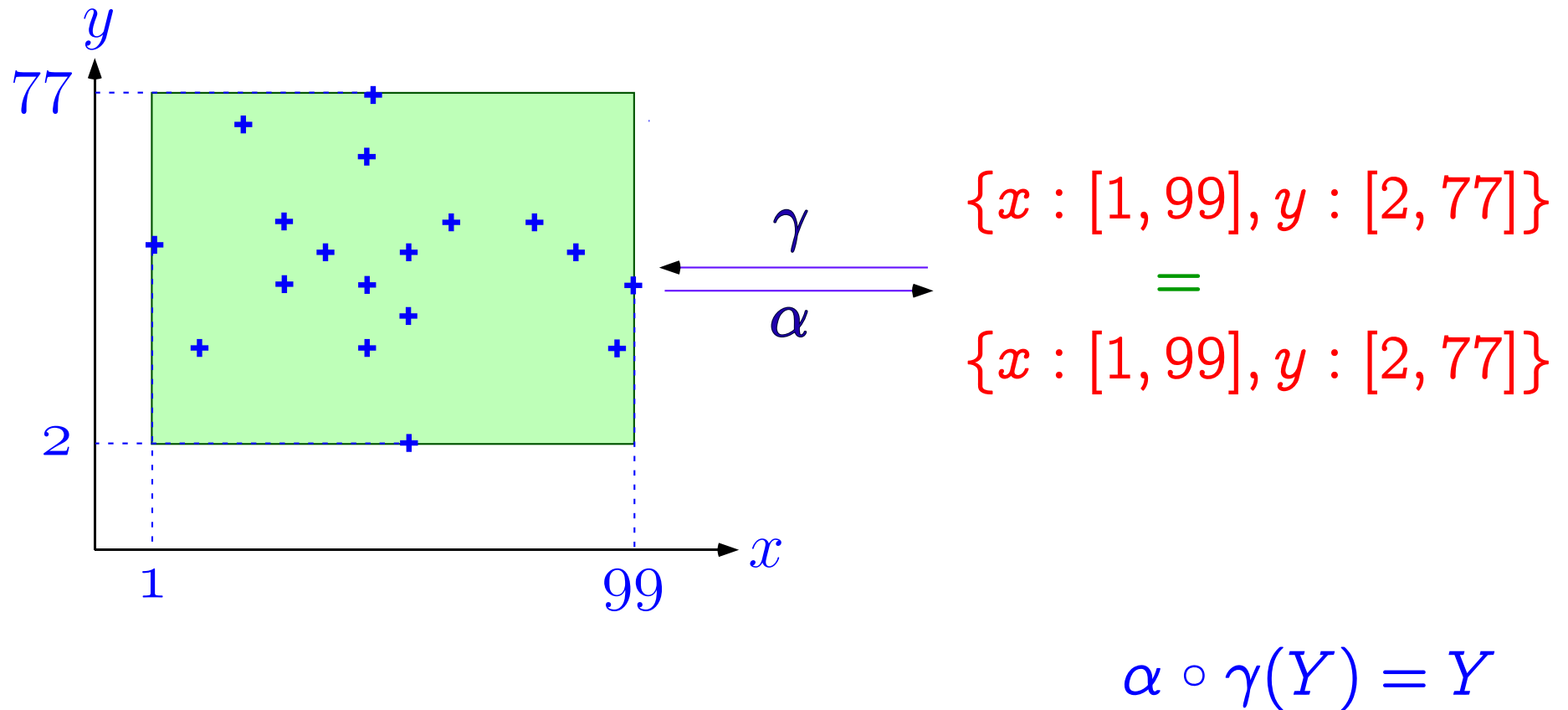
Galois Connection [1]

$$\langle P, \subseteq \rangle \xleftarrow[\alpha]{\gamma} \langle Q, \sqsubseteq \rangle$$

is defined as

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

iff

$$\alpha(X) \sqsubseteq Y \quad \text{iff} \quad X \subseteq \gamma(Y)$$

---

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

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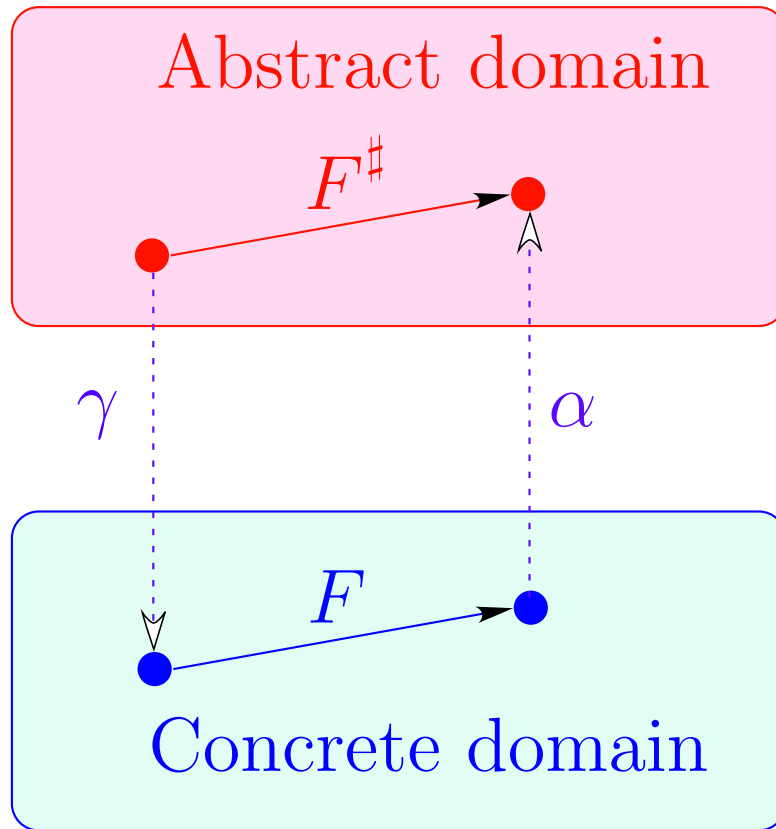
$$F^\sharp = \alpha \circ F \circ \gamma$$

$$\langle P,\ \subseteq \rangle \xleftrightarrow[\alpha]{\gamma} \langle Q,\ \sqsubseteq \rangle \Rightarrow$$

$$\langle P \xmapsto{\mathrm{mon}} P,\ \dot{\subseteq} \rangle \xleftrightarrow[\lambda F\, .\, \alpha \circ F \circ \gamma]{\lambda F^\sharp\, .\, \gamma \circ F^\sharp \circ \alpha} \langle Q \xmapsto{\mathrm{mon}} Q,\ \dot{\sqsubseteq} \rangle$$

Approximate Fixpoint Abstraction
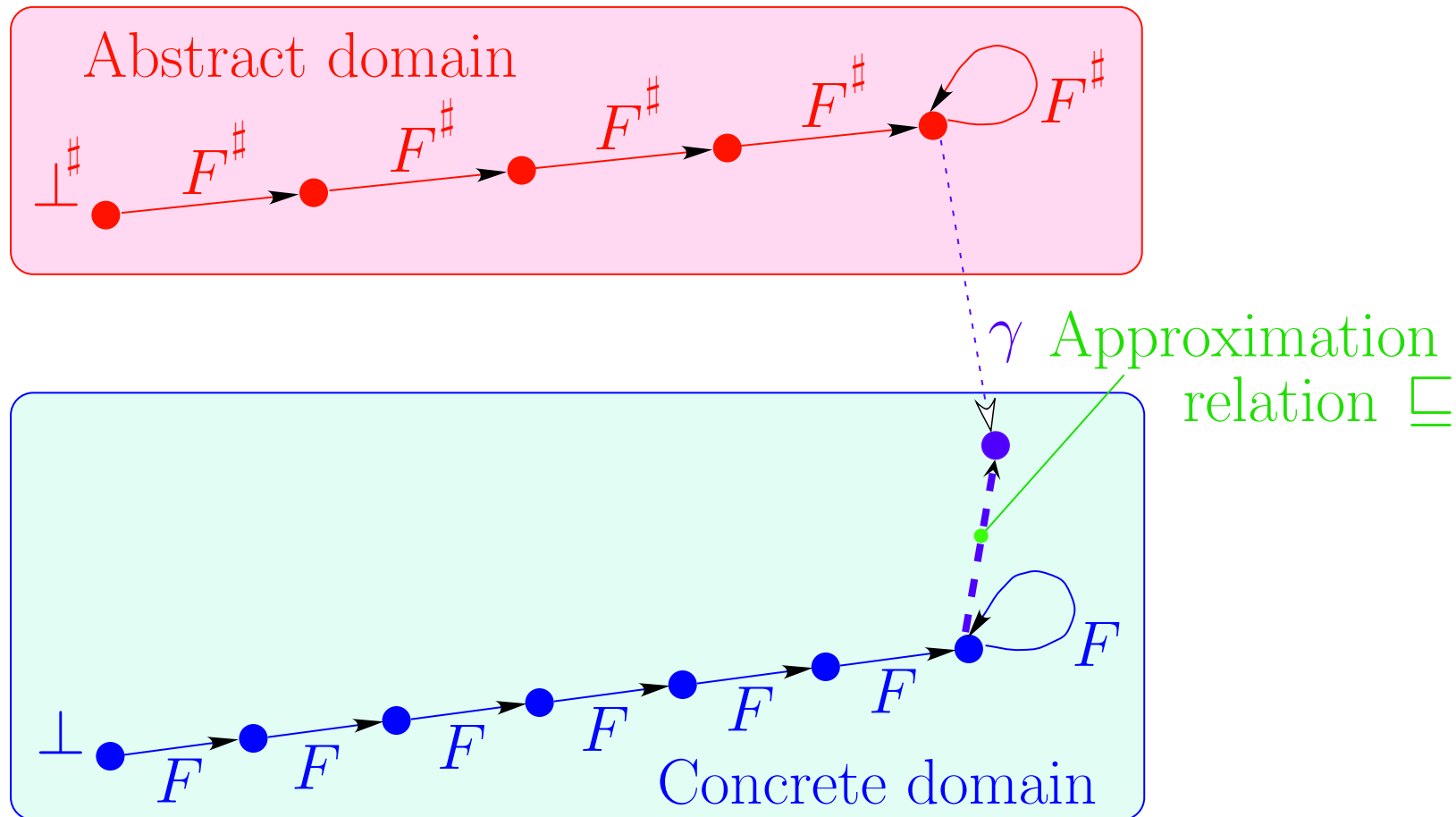
$$\alpha(\textsf{lfp}\, F) \sqsubseteq \textsf{lfp}\, F^{\sharp}$$

Exact Fixpoint Abstraction
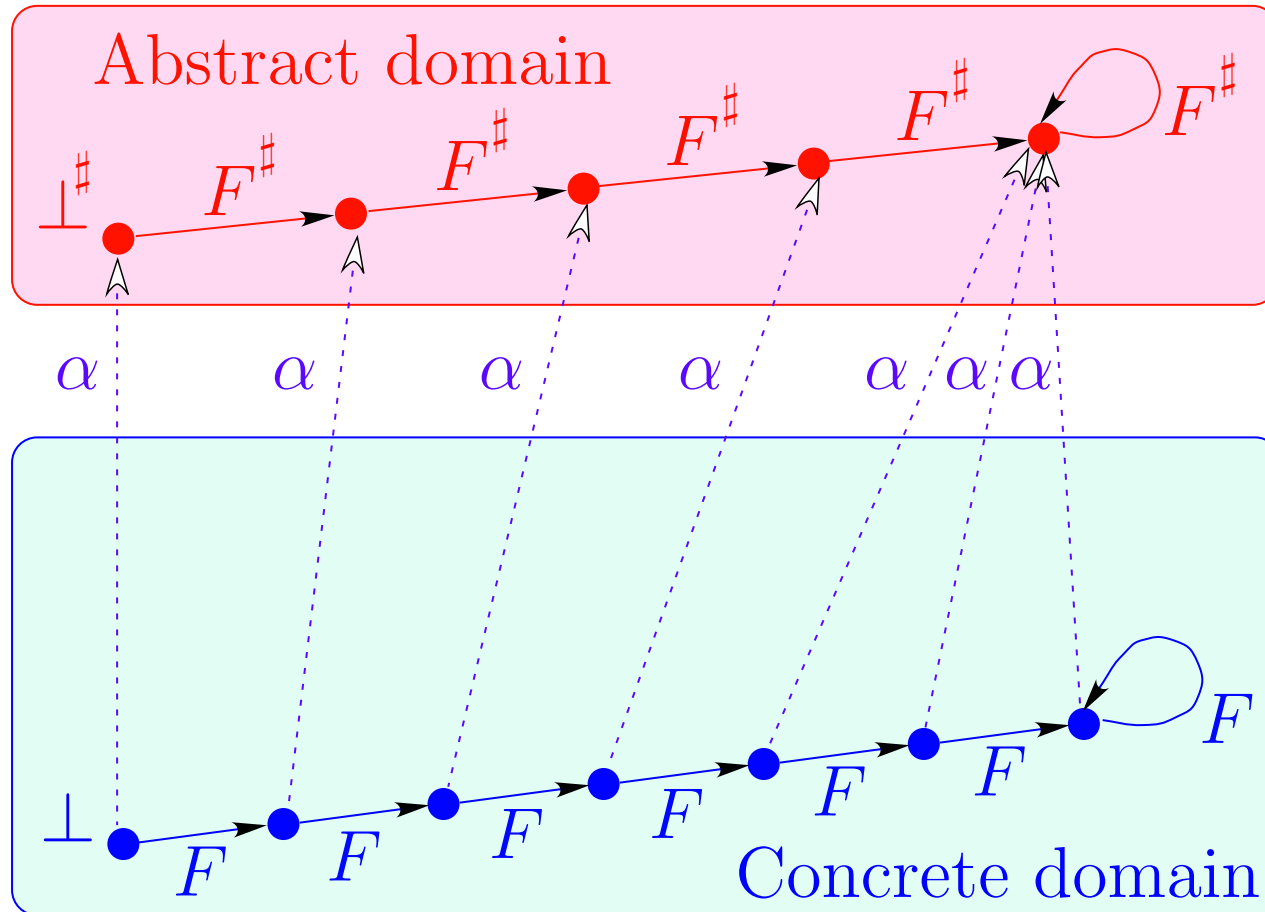
$$\alpha \circ F = F^{\sharp} \circ \alpha \;\Rightarrow\; \alpha\big(\mathsf{lfp}\,F\big) = \mathsf{lfp}\,F^{\sharp}$$

Exact/Approximate Fixpoint Abstraction

Exact Abstraction:

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

Approximate Abstraction:

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

Exact Fixpoint Abstraction
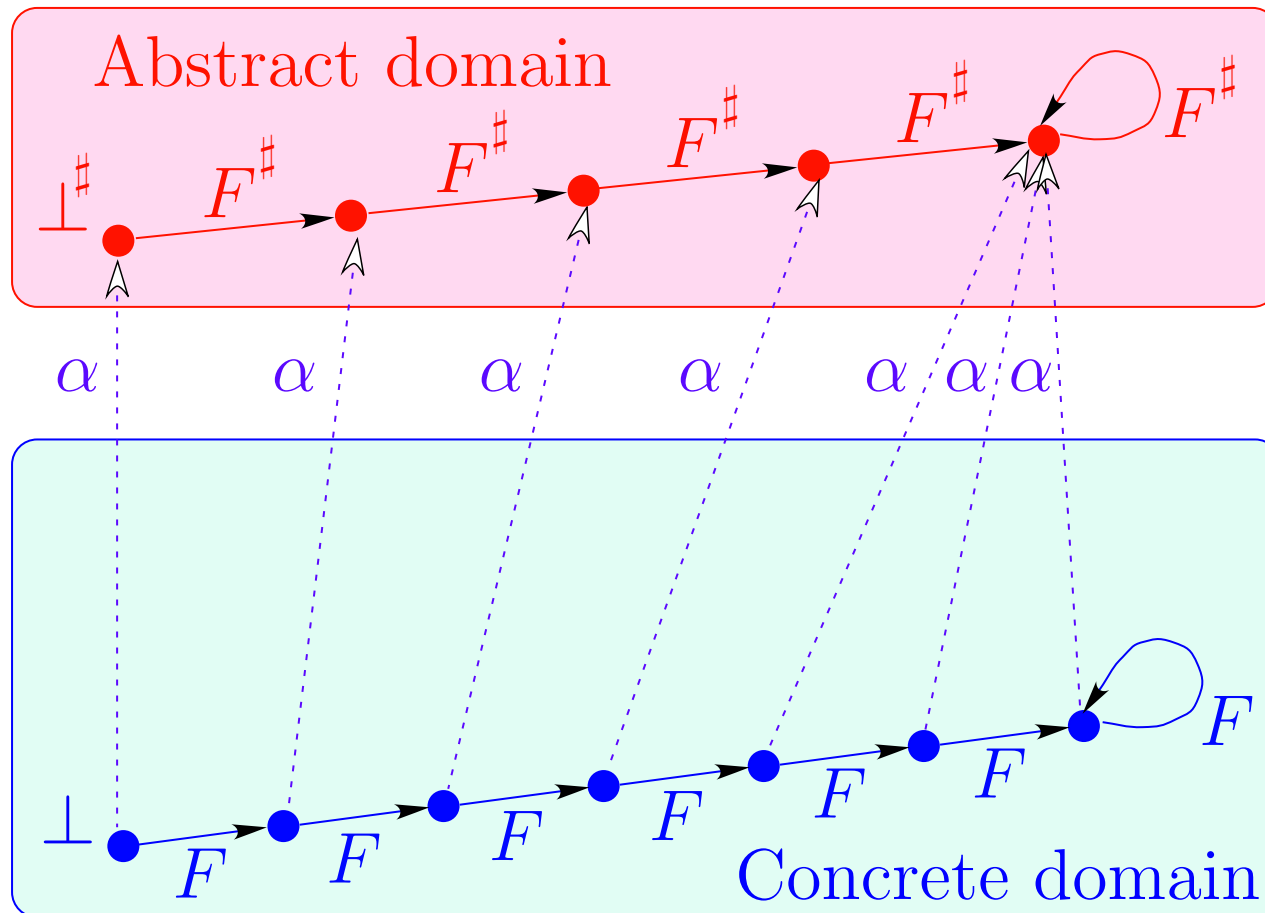
$$\alpha \circ F = F^\sharp \circ \alpha \implies \alpha(\mathsf{lfp}\, F) = \mathsf{lfp}\, F^\sharp$$

A Few References on Foundations

- 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.

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

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

Applications of Abstract Interpretation

Applications of Abstract Interpretation

- **Static Program Analysis** [POPL77,78,79]

- **Syntax Analysis** [TCS, to appear]

- **Hierarchies of Semantics (including Proofs)** [POPL 92, TCS02]

- **Typing** [POPL 97]

- **Model Checking** [POPL 00]

- **Program Transformation** [POPL 02]

All these techniques involves approximations that can be formalized by abstract interpretation.

A New Application of Abstract Interpretation: Program Transformation

Objectives of this Work

Program Transformation & Abstract Interpretation

In semantics-based program transformation, such as:

- constant propagation,
- partial evaluation,
- slicing,

abstract interpretation is used:

- in a preliminary program static analysis phase
- to collect the information about the program run-time behaviors, which is necessary
- to validate the applicable transformations.

Present Objective

Our present objective is **quite different**:

- Formalize the program transformation <u>itself</u> as an abstract interpretation;

- Two subgoals:

  - Understand correctness proofs of program transformations as abstract interpretations;

  - Imagine and apply a program transformation design method by abstract interpretation.

Principle of Online Program Transformation (Explained in Steps with the Constant Propagation Example)

The Programming Language

$$
\begin{array}{ll}
\texttt{a}: \; \texttt{X} := \; ? \; \rightarrow \texttt{b}; & \text{random assignment/input} \\
\texttt{b}: \; \texttt{Y} := 1 \rightarrow \texttt{c}; & \text{assignment} \\
\texttt{c}: \; (\texttt{X} \leq 0) \rightarrow \texttt{f}; & \text{nondeterministic guard} \\
\texttt{c}: \; (\texttt{X} > 0) \rightarrow \texttt{d}; & \\
\texttt{d}: \; \texttt{X} := \texttt{X} - \texttt{Y} \rightarrow \texttt{e}; & \\
\texttt{e}: \; \texttt{skip} \rightarrow \texttt{c}; & \text{branching} \\
\texttt{f}: \; \texttt{stop}; & \text{stop}
\end{array}
$$

Principle of Online Program Transformation (1)

- Program transformation is a syntactic process;

- maps a subject program into a transformed program;

- Both subject and transformed programs are syntactic objects.

Program Transformation: The Syntactic Point of View

Subject program:

a : X := ? $\rightarrow$ b;

b : Y := 1 $\rightarrow$ c;

c : $\left(X \leq 0\right) \rightarrow$ f;

c : $\left(X > 0\right) \rightarrow$ d;

d : X := X $-$ Y $\rightarrow$ e;

e : skip $\rightarrow$ c;

f : stop;

Transformed program:

a : X := ? $\rightarrow$ b;

b : Y := 1 $\rightarrow$ c;

c : $\left(X \leq 0\right) \rightarrow$ f;

c : $\left(X > 0\right) \rightarrow$ d;

d : X := X $-$ $\overset{1}{\cancel{Y}}$ $\rightarrow$ e;

e : skip $\rightarrow$ c;

f : stop;

Principle of Online Program Transformation (1)

Subject
program P

Syntactic
$\xrightarrow{\hspace{4cm}}$
transformation $\mathfrak{t}$

Transformed
program
$\mathfrak{t}[\![P]\!]$
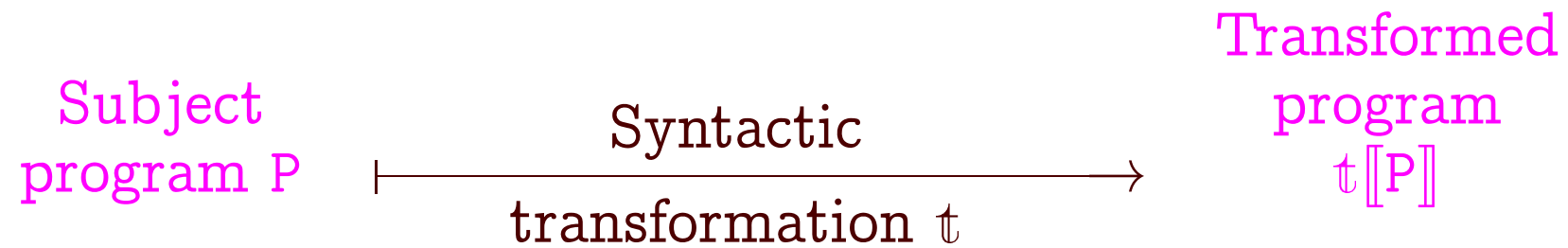
Principle of Online Program Transformation (2)

- Program transformations refer to the semantics of the subject and transformed programs:

  - Online program transformations use values manipulated during program execution, hence directly refer to the source concrete semantics;

  - Offline program transformations use a preliminary static analysis of the source program, hence refer to its abstract semantics;

Principle of Online Program Transformation (2)

$$
\begin{array}{ccc}
\text{Subject} & \xrightarrow{\makebox[3cm]{}} & \text{Transformed} \\
\text{program P} & \text{Syntactic} & \text{program} \\
& \text{transformation } \mathbb{t} & \mathbb{t}[\![P]\!] \\
\Big\downarrow \mathbf{S} & & \Big\downarrow \mathbf{S} \\
\text{Subject pro-} & & \text{Transformed pro-} \\
\text{gram seman-} & & \text{gram semantics} \\
\text{tics } \mathbf{S}[\![P]\!] & & \mathbf{S}[\![\mathbb{t}[\![P]\!]]\!]
\end{array}
$$
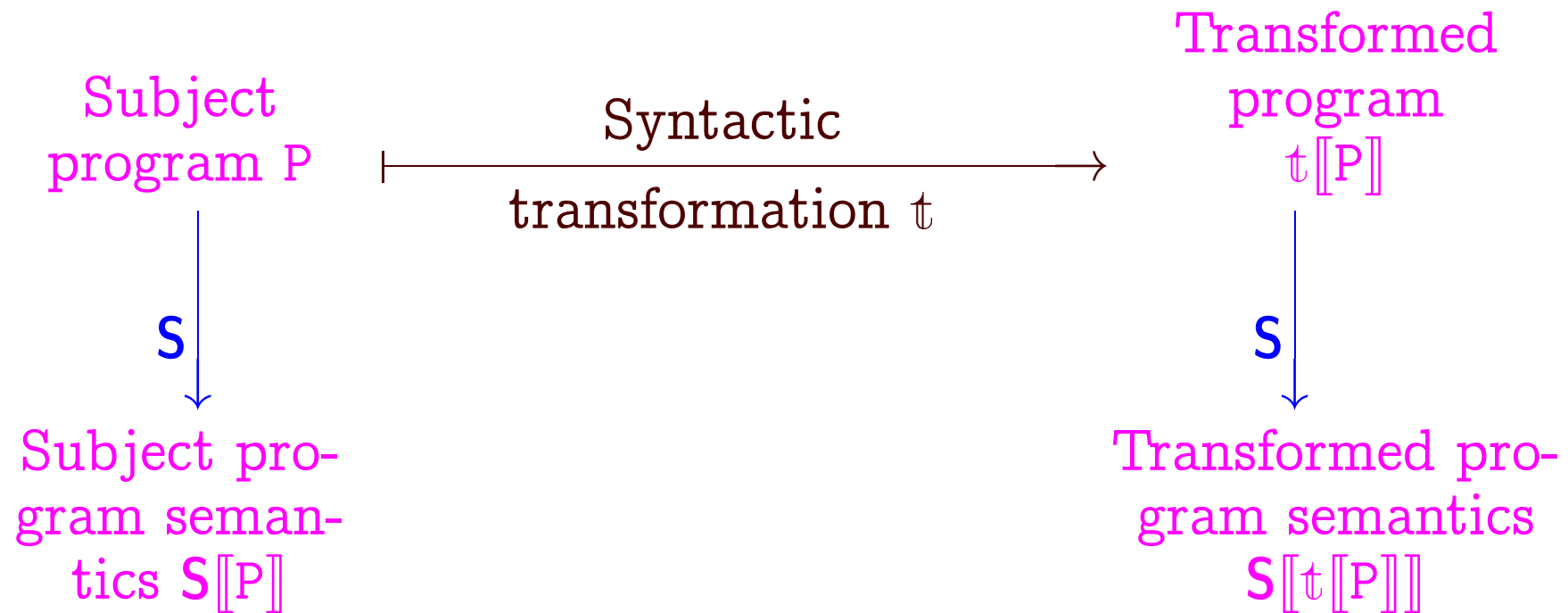
The Prefix Trace Semantics

The semantics is the set of prefixes of all traces similar to that one (with different inputs)

$$\langle\, \mathtt{a} : \mathtt{X} := ? \to \mathtt{b};,\ [\mathtt{X} : \mho, \mathtt{Y} : \mho]\,\rangle$$

$$\langle\, \mathtt{b} : \mathtt{Y} := 1 \to \mathtt{c};,\ [\mathtt{X} : 1, \mathtt{Y} : \mho]\,\rangle$$

a : X := ? → b;

b : Y := 1 → c;

c : (X ≤ 0) → f;

c : (X > 0) → d;

$$\langle\, \mathtt{c} : (\mathtt{X} > 0) \to \mathtt{d};,\ [\mathtt{X} : 1, \mathtt{Y} : 1]\,\rangle$$

d : X := X − Y → e;

$$\langle\, \mathtt{d} : \mathtt{X} := \mathtt{X} - \mathtt{Y} \to \mathtt{e};,\ [\mathtt{X} : 1, \mathtt{Y} : 1]\,\rangle$$

e : skip → c;

$$\langle\, \mathtt{e} : \mathtt{skip} \to \mathtt{c};,\ [\mathtt{X} : 0, \mathtt{Y} : 1]\,\rangle$$

$$\langle\, \mathtt{c} : (\mathtt{X} \le 0) \to \mathtt{f};,\ [\mathtt{X} : 0, \mathtt{Y} : 1]\,\rangle$$

f : stop;

$$\langle\, \mathtt{f} : \mathtt{stop};,\ [\mathtt{X} : 0, \mathtt{Y} : 1]\,\rangle$$

Semantics of the Transformed program

a : X := ? → b;

b : Y := 1 → c;

c : (X ≤ 0) → f;

c : (X > 0) → d;

$$d : X := X - \overset{1}{\cancel{Y}} \to e;$$

e : skip → c;

f : stop;

$\langle a : X := ? \to b;, [X : \circlearrowleft, Y : \circlearrowleft] \rangle$

$\langle b : Y := 1 \to c;, [X : 1, Y : \circlearrowleft] \rangle$

$\langle c : (X > 0) \to d;, [X : 1, Y : 1] \rangle$

$\langle d : X := X - \overset{1}{\cancel{Y}} \to e;, [X : 1, Y : 1] \rangle$

$\langle e : \text{skip} \to c;, [X : 0, Y : 1] \rangle$

$\langle c : (X \leq 0) \to f;, [X : 0, Y : 1] \rangle$

$\langle f : \text{stop};, [X : 0, Y : 1] \rangle$

Principle of Online Program Transformation (3)

- The subject semantics and transformed semantics cannot be exactly the same;

- However they should be equivalent, at some level of observation.

Principle of Online Program Transformation (3)

Subject
program P $\xrightarrow{\text{Syntactic}}$ Transformed
program
$\mathfrak{t}[\![P]\!]$

Syntactic
transformation $\mathfrak{t}$

S | S

Subject pro-
gram seman-
tics $\mathbf{S}[\![P]\!]$ $\overline{\underline{\text{Observational}}}$ Transformed pro-
gram semantics
$\mathbf{S}[\![\mathfrak{t}[\![P]\!]]\!]$

equivalence

Principle of Online Program Transformation (3)

- The observational equivalence gets rids of irrelevant details about the subject and transformed program semantics;

- Hence it is an abstract interpretation of the subject and transformed program semantics!

Observational semantics

a : X := ? → b;   ⟨a : X := ? → b;, [X : ℧, Y : ℧]⟩

b : Y := 1 → c;   ⟨b : Y := 1 → c;, [X : 1, Y : ℧]⟩

c : (X ≤ 0) → f;

c : (X > 0) → d;   ⟨c : (X > 0) → d;, [X : 1, Y : 1]⟩

$$d : X := X - \overset{1}{\cancel{Y}} → e;$$ $$⟨d : X := X - \overset{1}{\cancel{Y}} → e;, [X : 1, Y : 1]⟩$$

e : skip → c;   ⟨e : skip → c;, [X : 0, Y : 1]⟩

                                   ⟨c : (X ≤ 0) → f;, [X : 0, Y : 1]⟩

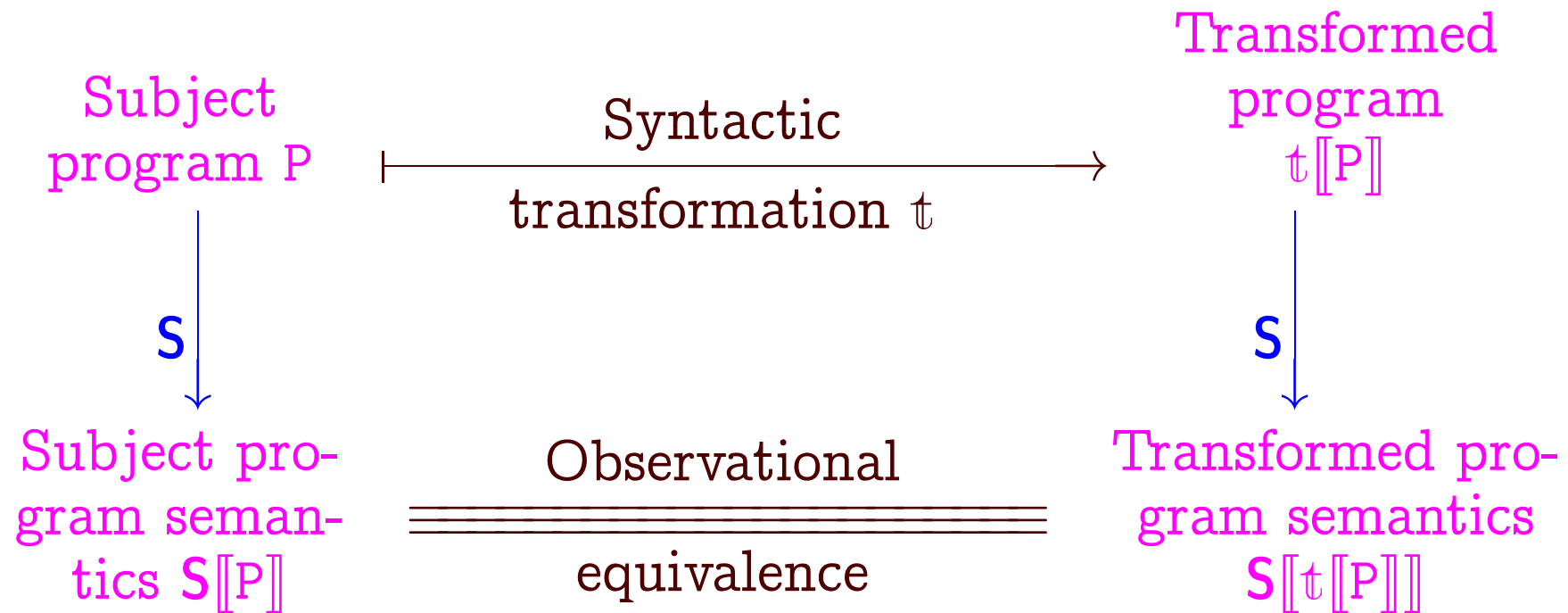f : stop;   ⟨f : stop;, [X : 0, Y : 1]⟩

Principle of Online Program Transformation (3)
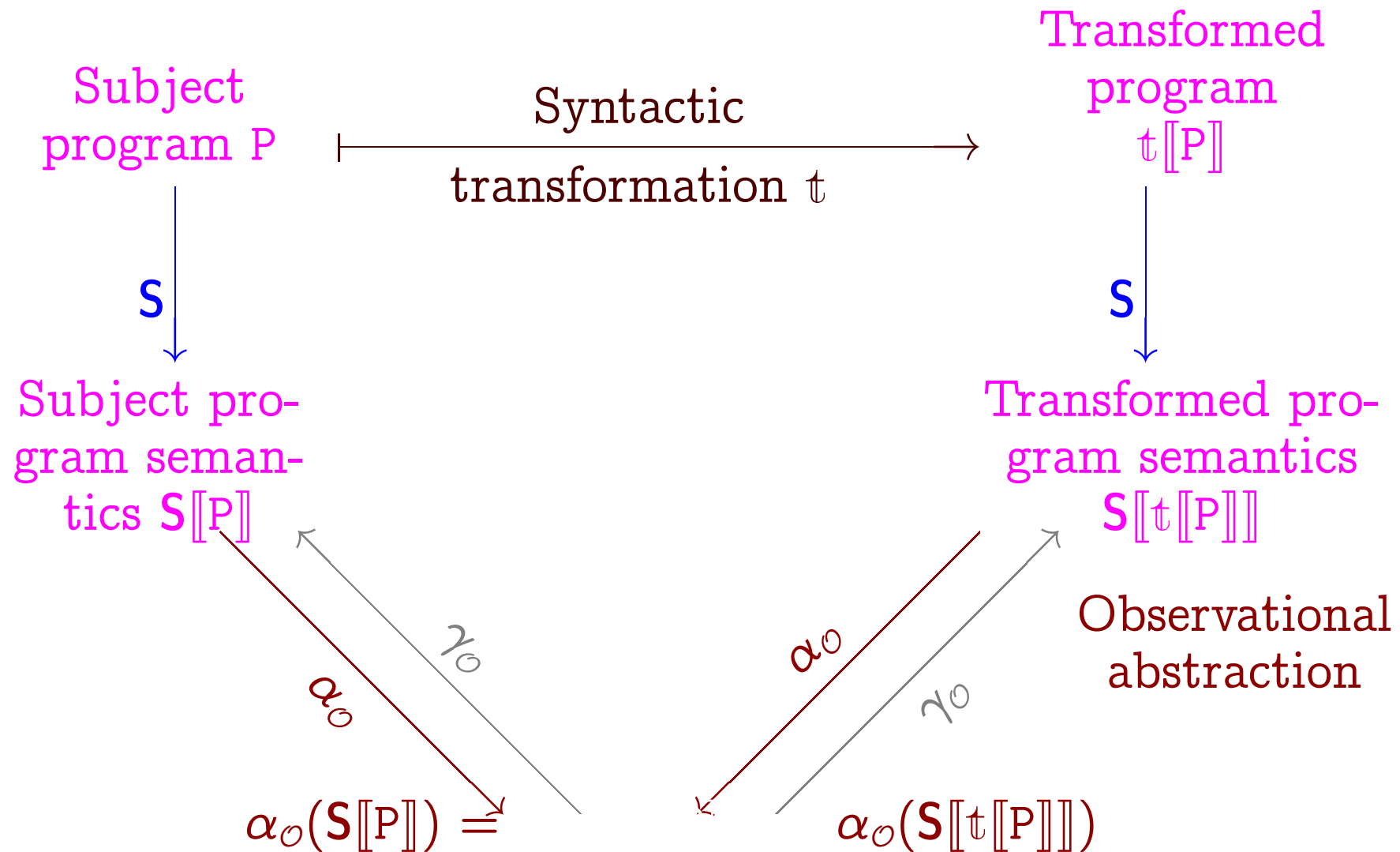
Example: Constant Propagation

Subject/Transformed Semantics

$\langle \mathtt{a} : \mathtt{X} := ? \to \mathtt{b} ;, \; [\mathtt{X} : \mho, \mathtt{Y} : \mho] \rangle$
$\langle \mathtt{b} : \mathtt{Y} := 1 \to \mathtt{c} ;, \; [\mathtt{X} : 1, \mathtt{Y} : \mho] \rangle$
$\langle \mathtt{c} : (\mathtt{X} > 0) \to \mathtt{d} ;, \; [\mathtt{X} : 1, \mathtt{Y} : 1] \rangle$

$\langle \mathtt{d} : \mathtt{X} := \mathtt{X} - \overset{1}{\cancel{\mathtt{Y}}} \to \mathtt{e} ;, \; [\mathtt{X} : 1, \mathtt{Y} : 1] \rangle$
$\langle \mathtt{e} : \mathtt{skip} \to \mathtt{c} ;, \; [\mathtt{X} : 0, \mathtt{Y} : 1] \rangle$
$\langle \mathtt{c} : (\mathtt{X} \le 0) \to \mathtt{f} ;, \; [\mathtt{X} : 0, \mathtt{Y} : 1] \rangle$
$\langle \mathtt{f} : \mathtt{stop} ;, \; [\mathtt{X} : 0, \mathtt{Y} : 1] \rangle$

Observational Semantics

$\langle \mathtt{a} : \quad \to \mathtt{b} ;, \; [\mathtt{X} : \mho, \mathtt{Y} : \mho] \rangle$
$\langle \mathtt{b} : \quad \to \mathtt{c} ;, \; [\mathtt{X} : 1, \mathtt{Y} : \mho] \rangle$
$\langle \mathtt{c} : \quad \to \mathtt{d} ;, \; [\mathtt{X} : 1, \mathtt{Y} : 1] \rangle$

$\langle \mathtt{d} : \quad \to \mathtt{e} ;, \; [\mathtt{X} : 1, \mathtt{Y} : 1] \rangle$
$\langle \mathtt{e} : \quad \to \mathtt{c} ;, \; [\mathtt{X} : 0, \mathtt{Y} : 1] \rangle$
$\langle \mathtt{c} : \quad \to \mathtt{f} ;, \; [\mathtt{X} : 0, \mathtt{Y} : 1] \rangle$
$\langle \mathtt{f} : \mathtt{stop} ;, \; [\mathtt{X} : 0, \mathtt{Y} : 1] \rangle$

Principle of Online Program Transformation (4)

- The syntactic transformation induces a semantic transformation:

    The semantics of the subject program is mapped to the semantics of the transformed program;

- The subject semantics and the transformed semantics should be observationally equivalent;

- The semantic transformation is in general more precise than the algorithmic syntactic transformation (e.g. infinite behaviors are no problem at the semantic level).

Example: Semantic Constant Propagation

$a : X := ? \to b;$

$b : Y := 1 \to c;$

$c : (X \le 0) \to f;$

$c : (X > 0) \to d;$

$d : X := X - Y \to e;$

$e : \text{skip} \to c;$

$f : \text{stop};$

$\langle a : X := ? \to b;, [X : \circlearrowright, Y : \circlearrowright] \rangle$

$\langle b : Y := 1 \to c;, [X : 1, Y : \circlearrowright] \rangle$

$\langle c : (X > 0) \to d;, [X : 1, Y : 1] \rangle$

$\langle d : X := X - \overset{1}{\cancel{Y}} \to e;, [X : 1, Y : 1] \rangle$

$\langle e : \text{skip} \to c;, [X : 0, Y : 1] \rangle$

$\langle c : (X \le 0) \to f;, [X : 0, Y : 1] \rangle$

$\langle f : \text{stop};, [X : 0, Y : 1] \rangle$

Principle of Online Program Transformation (4)

Subject program P — Syntactic transformation $\mathfrak{t}$ → Transformed program $\mathfrak{t}[\![P]\!]$

$\mathbf{S}$

Subject program semantics $\mathbf{S}[\![P]\!]$ — Semantic transformation t → Transformed program semantics $\mathrm{t}[\mathbf{S}[\![P]\!]] \sqsubseteq \mathbf{S}[\![\mathfrak{t}[\![P]\!]]\!]$

Observational abstraction

$\gamma_{\mathcal{O}}$ $\alpha_{\mathcal{O}}$

$\alpha_{\mathcal{O}}$ $\gamma_{\mathcal{O}}$

$$\alpha_{\mathcal{O}}(\mathbf{S}[\![P]\!]) = \alpha_{\mathcal{O}}(\mathrm{t}[\mathbf{S}[\![P]\!]]) = \alpha_{\mathcal{O}}(\mathbf{S}[\![\mathfrak{t}[\![P]\!]]\!])$$

Correspondence Between Syntax and Semantics

Correspondence Between
Syntax and Semantics

- The program syntax forgets details about the program execution semantics:

  – The sequence of values of variables during execution is forgotten, but:

    – their existence and maybe their type are recorded;

    – the sequence (partial order, . . .) of (denotations of) actions performed on these variables is recorded;

  – Program execution times are completely abstracted (but might be included in the operational semantics);

Correspondence Between Syntax and Semantics, Cont'd

- The correspondence between syntax and semantics is an abstraction:

$$\mathsf{po}\langle \mathfrak{D};\ \sqsubseteq \rangle \xleftarrow[\ \mathbb{p}\ ]{\ \mathsf{S}\ } \mathsf{po}\langle \mathbb{P}/\equiv;\ \mathbb{\sqsubseteq} \rangle$$

- The concretization S is the semantics of the program;

- The abstraction $\mathbb{p}$ is the "decompilation" of the semantics.

Principle of Online Program Transformation (5)

Subject
program P

$\xrightarrow{\text{Syntactic transformation } \mathbb{t}}$

Transformed
program
$\mathbb{t}[\![P]\!]$

$\mathbf{S}\,\Vert\,\mathbb{p}$

$\mathbf{S}\,\Vert\,\mathbb{p}$

Subject program semantics $\mathbf{S}[\![P]\!]$

$\xrightarrow{\text{Semantic transformation } t}$

Transformed program semantics
$t[\mathbf{S}[\![P]\!]] \sqsubseteq \mathbf{S}[\![\mathbb{t}[\![P]\!]]\!]$

$\alpha_{\mathcal{O}}$ $\gamma_{\mathcal{O}}$ $\alpha_{\mathcal{O}}$ $\gamma_{\mathcal{O}}$

Observational
abstraction

$$\alpha_{\mathcal{O}}(\mathbf{S}[\![P]\!]) = \alpha_{\mathcal{O}}(t[\mathbf{S}[\![P]\!]]) = \alpha_{\mathcal{O}}(\mathbf{S}[\![\mathbb{t}[\![P]\!]]\!])$$
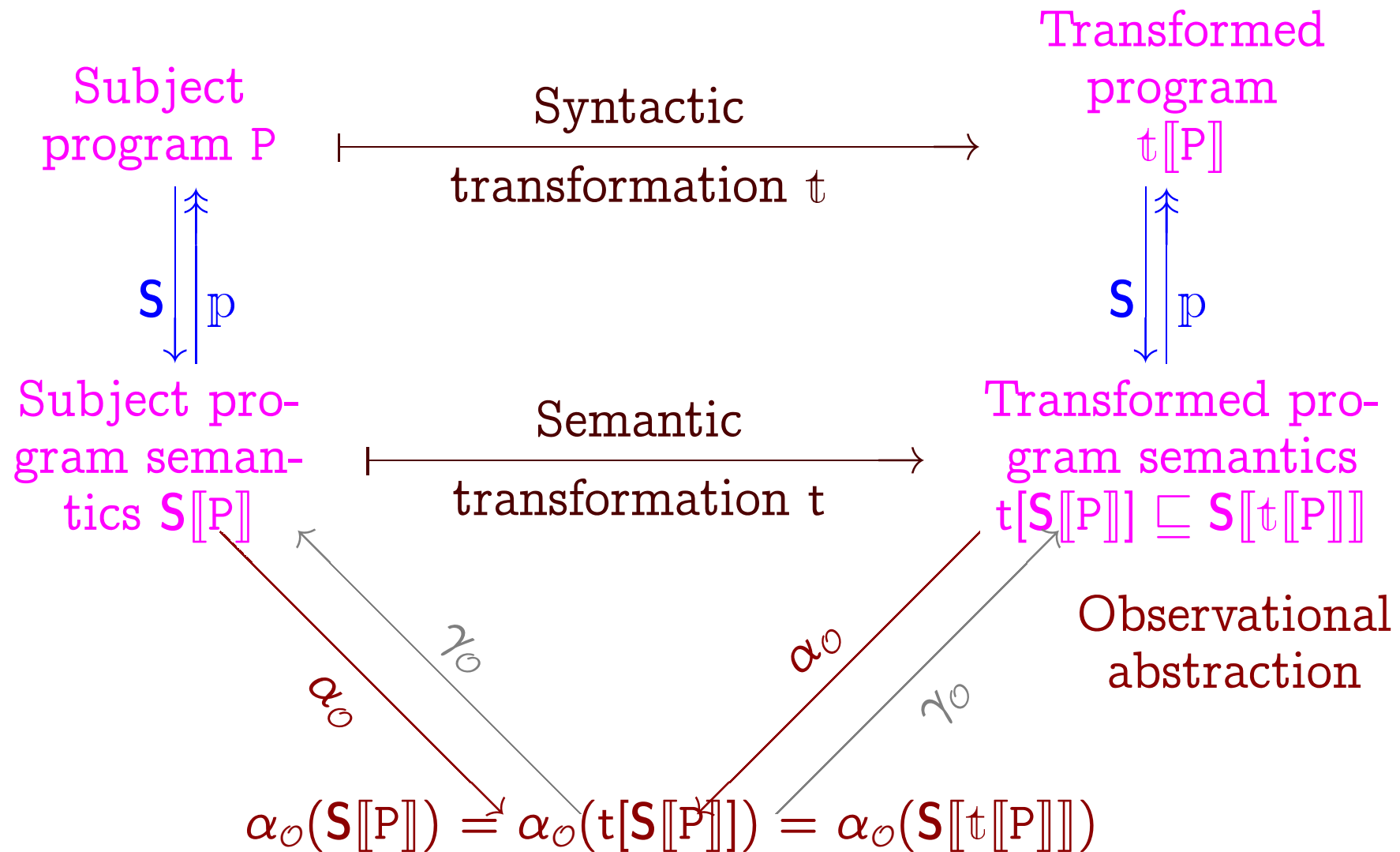
Example: Syntax to Prefix Trace Semantics

- Fixpoint semantics:

$$\mathbf{S}^*[\![\mathrm{P}]\!] = \mathsf{lfp}^{\subseteq} \, \mathbf{F}^*[\![\mathrm{P}]\!]$$

$$\mathbf{F}^*[\![\mathrm{P}]\!]\mathcal{T} = \mathcal{I}[\![\mathrm{P}]\!] \cup \{\sigma s s' \mid \sigma s \in \mathcal{T} \wedge s' \in \mathbf{S}[\![\mathrm{P}]\!]s\},$$

Example: Prefix Trace Semantics to Syntax

• Collect commands along traces.

Correspondence Between the Subject Semantics and the Transformed Semantics

Semantic Transformations as Approximations

- A semantic program transformation is a loss of information on the semantics of the subject program;

  $\longrightarrow$ The semantic program transformation is an abstraction;

Intuition for Transformations as Abstractions

a : X := ? → b;

b : Y := 1 → c;

c : $(X \leq 0)$ → f;

c : $(X > 0)$ → d;

$$\begin{array}{c} \cdots \\ X - (2 * Y - 1) \\ X - Y \end{array}$$

d : X := X − 1 → e;

e : skip → c;

f : stop;

a : X := ? → b;

b : Y := 1 → c;

c : $(X \leq 0)$ → f;

c : $(X > 0)$ → d;

d : X := X − 1 → e;

e : skip → c;

f : stop;

Principle of Online Program Transformation (6)

$$\alpha_{\mathcal{O}}(\mathbf{S}[\![\mathrm{P}]\!]) = \alpha_{\mathcal{O}}(\mathrm{t}[\mathbf{S}[\![\mathrm{P}]\!]]) \stackrel{.}{=} \alpha_{\mathcal{O}}(\mathbf{S}[\![\mathrm{t}[\![\mathrm{P}]\!]]\!])$$
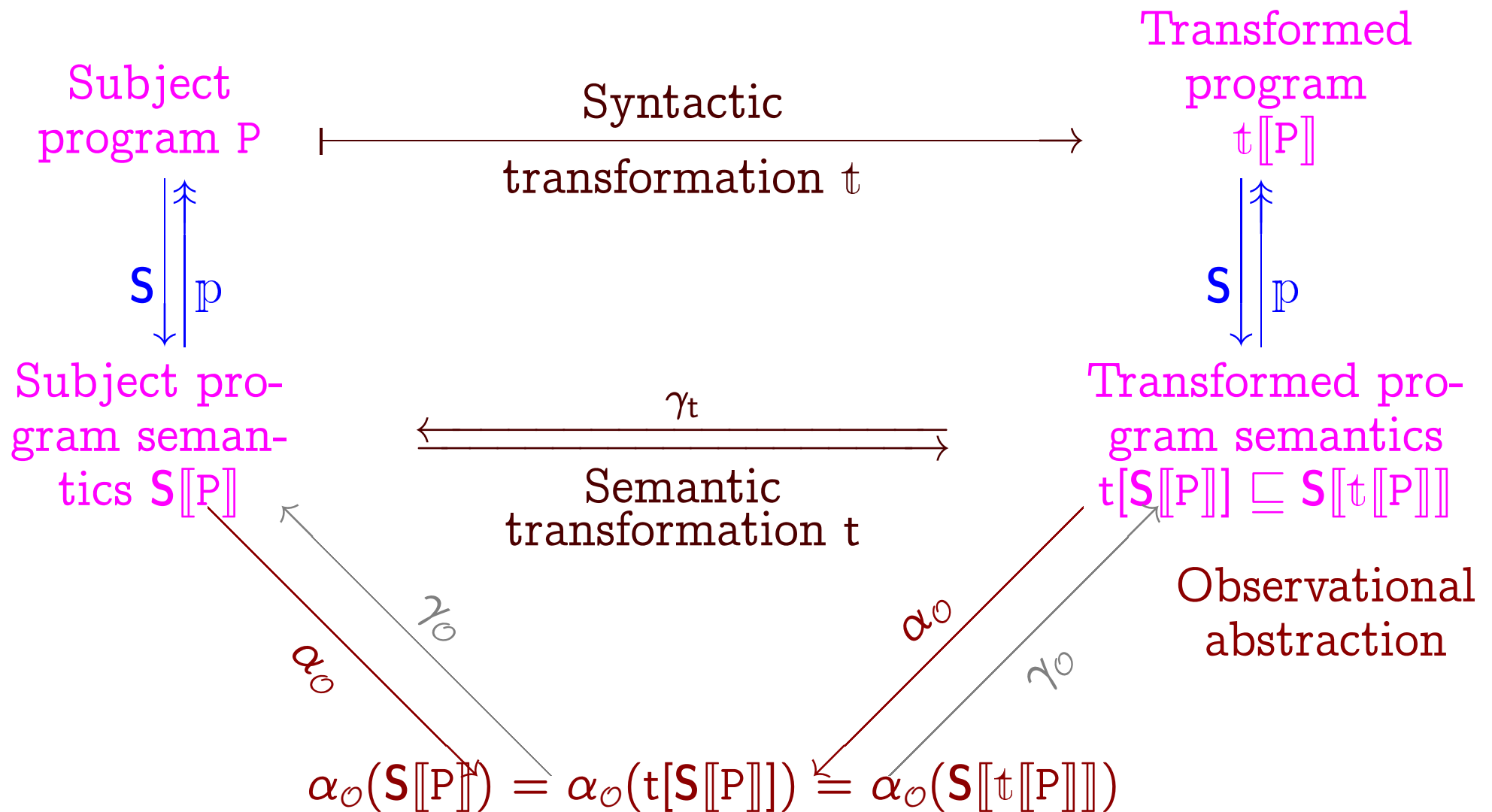
Correspondence Between the Subject Program and the Transformed Program

Syntactic Transformations as Approximations

- By composition, the syntactic program transformation is also a loss of information on subject program;

  $\longrightarrow$ The syntactic program transformation is an abstraction;

Semantic to Syntactic Constant Propagation

$a : X := ? \rightarrow b;$

$b : Y := 1 \rightarrow c;$

$c : (X \leq 0) \rightarrow f;$

$c : (X > 0) \rightarrow d;$

$d : X := X - \overset{1}{\cancel{Y}} \rightarrow e;$

$e : \text{skip} \rightarrow c;$

$f : \text{stop};$

$\langle a : X := ? \rightarrow b;, [X : \circlearrowright, Y : \circlearrowright] \rangle$

$\langle b : Y := 1 \rightarrow c;, [X : 1, Y : \circlearrowright] \rangle$

$\langle c : (X > 0) \rightarrow d;, [X : 1, Y : 1] \rangle$

$\langle d : X := X - \overset{1}{\cancel{Y}} \rightarrow e;, [X : 1, Y : 1] \rangle$

$\langle e : \text{skip} \rightarrow c;, [X : 0, Y : 1] \rangle$

$\langle c : (X \leq 0) \rightarrow f;, [X : 0, Y : 1] \rangle$

$\langle f : \text{stop};, [X : 0, Y : 1] \rangle$

Principle of Online Program Transformation (Done)

Subject
program P

$\gamma_{\mathbb{t}}$

Syntactic
transformation $\mathbb{t}$

Transformed
program
$\mathbb{t}[\![P]\!]$

$\mathbf{S}\ \mathbb{p}$

$\mathbf{S}\ \mathbb{p}$

Subject program semantics $\mathbf{S}[\![P]\!]$

γ_{t}

Semantic
transformation t

Transformed program semantics
$\mathsf{t}[\mathbf{S}[\![P]\!]] \sqsubseteq \mathbf{S}[\![\mathbb{t}[\![P]\!]]\!]$

$\gamma_{\mathcal{O}}$

$\alpha_{\mathcal{O}}$

$\alpha_{\mathcal{O}}$

$\gamma_{\mathcal{O}}$

Observational
abstraction

$$\alpha_{\mathcal{O}}(\mathbf{S}[\![P]\!]) = \alpha_{\mathcal{O}}(\mathsf{t}[\mathbf{S}[\![P]\!]]) = \alpha_{\mathcal{O}}(\mathbf{S}[\![\mathbb{t}[\![P]\!]]\!])$$

Principle of the <u>Formalization</u> of Program Transformation by Abstract Interpretation

Formalization of Program Transformation Correctness by Abstract Interpretation

Correctness of an Online Program Transformation

Subject
program P

Transformed
program
$t[\![P]\!]$

$$\xleftarrow{\quad\gamma_t\quad}$$

Syntactic
transformation t

$S \Big\downarrow \mathbb{p}$

$S \Big\downarrow \mathbb{p}$

Subject program semantics $S[\![P]\!]$

Transformed program semantics
$t[S[\![P]\!]] \sqsubseteq S[\![t[\![P]\!]]\!]$

$$\xleftarrow{\quad\gamma_t\quad}$$

Semantic
transformation t

$\alpha_{\mathcal{O}}$

$\gamma_{\mathcal{O}}$

$\alpha_{\mathcal{O}}$

$\gamma_{\mathcal{O}}$

Observational
abstraction

$\alpha_{\mathcal{O}}(S[\![P]\!]) =$

$\alpha_{\mathcal{O}}(S[\![t[\![P]\!]]\!])$
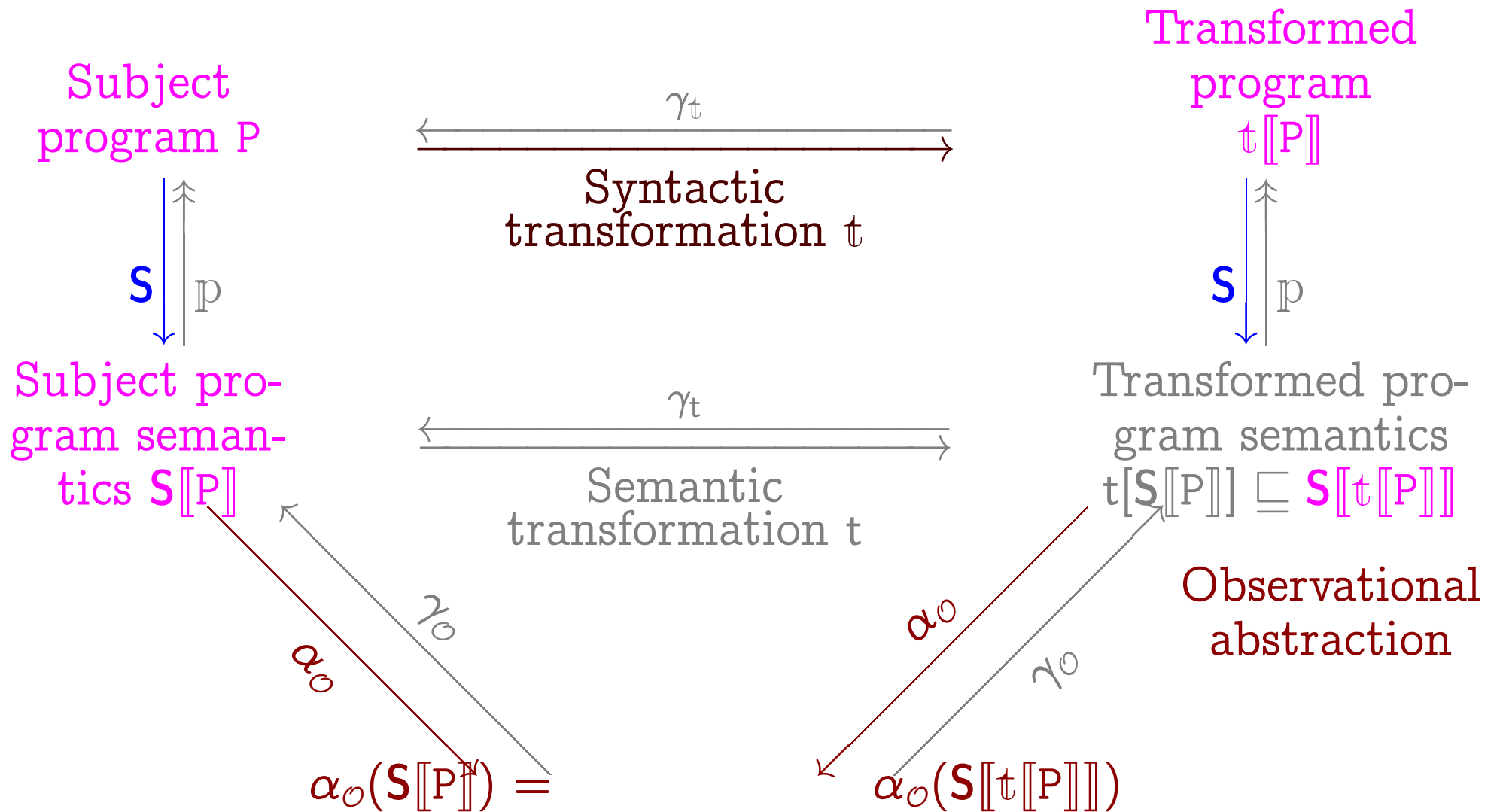
Design of
Program Transformations
by Abstract Interpretation

Design of an Online Program Transformation

Subject
program P

Transformed
program
$$\mathbb{t}[\![P]\!] \sqsupseteq \mathbb{p}[\mathbb{t}[\mathbf{S}[\![P]\!]]]$$

$\gamma_\mathbb{t}$

Syntactic
transformation $\mathbb{t}$

$\mathbf{S} \Big\downarrow \mathbb{p}$

$\mathbf{S} \Big\downarrow \mathbb{p}$

Subject pro-
gram seman-
tics $\mathbf{S}[\![P]\!]$

γ_t

Semantic
transformation t

Transformed pro-
gram semantics
$\mathsf{t}[\mathbf{S}[\![P]\!]] \sqsubseteq \mathbf{S}[\![\mathbb{t}[\![P]\!]]]$

$\gamma_\mathcal{O}$

$\alpha_\mathcal{O}$

$\alpha_\mathcal{O}$

$\gamma_\mathcal{O}$

Observational
abstraction

$$\alpha_\mathcal{O}(\mathbf{S}[\![P]\!]) = \alpha_\mathcal{O}(\mathsf{t}[\mathbf{S}[\![P]\!]]) = \alpha_\mathcal{O}(\mathbf{S}[\![\mathbb{t}[\![P]\!]]])$$
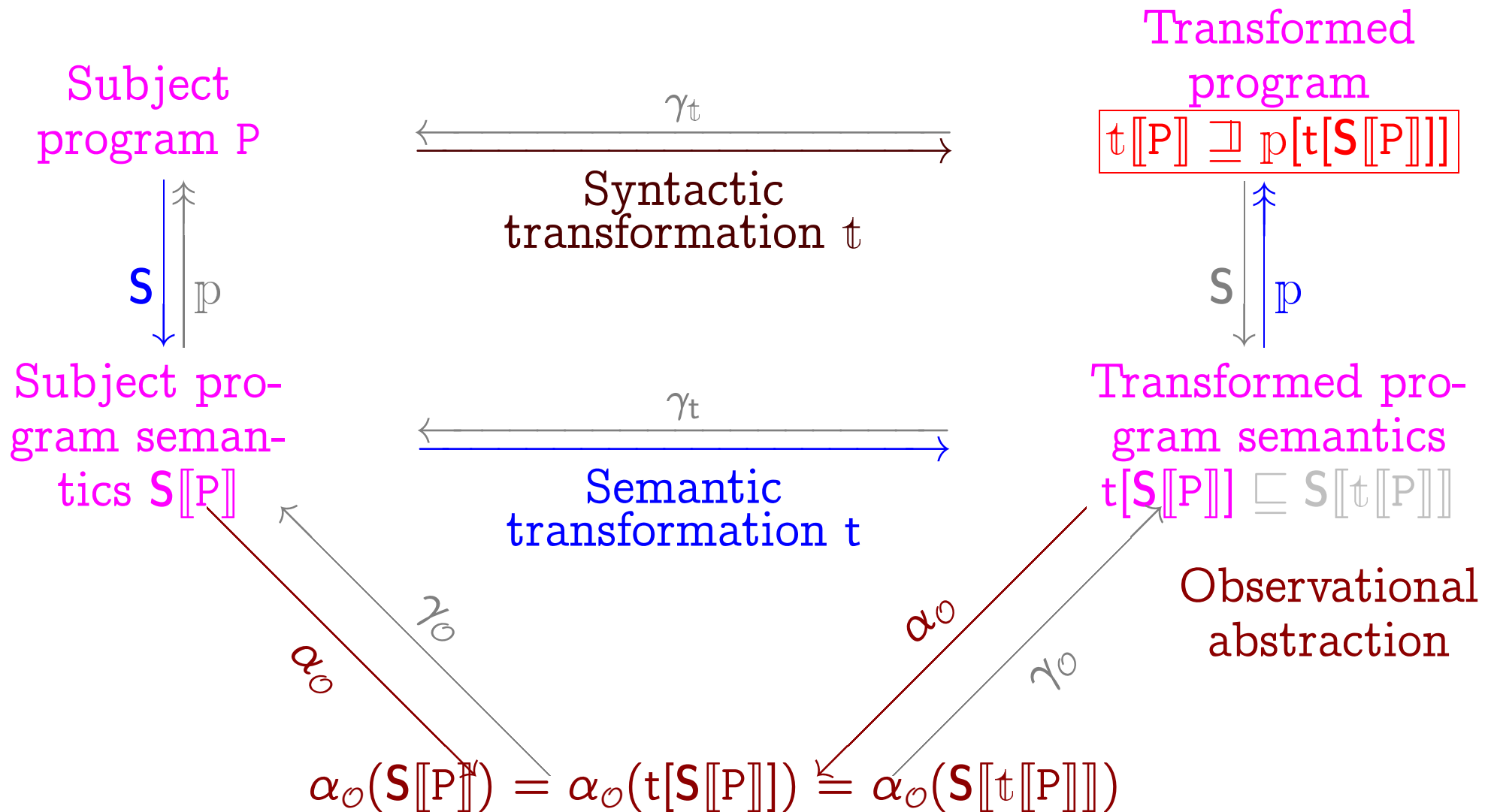
Design of Program Transformation Algorithms

$$\mathbb{t}[\![\mathrm{P}]\!] \sqsupseteq \mathbb{p}[t[\mathbf{S}[\![\mathrm{P}]\!]]]$$

$$= \mathbb{p}[t[\mathsf{lfp}^{\sqsubseteq} \mathbf{F}[\![\mathrm{P}]\!]]]$$

$$\sqsupseteq \ldots \qquad \leftarrow \text{apply fixpoint transfer}$$
$$/\text{approximation} \quad \text{theorems}$$

$$= \mathsf{lfp}^{\sqsubseteq^{\sharp}} \mathbb{F}^{\sharp}[\![\mathrm{P}]\!]$$

We obtain an iterative program transformation algorithm!

The Iterative Constant Propagation Algorithm

ConstantPropagation$(\mathrm{P}, \rho^\sharp) =$
  $\mathrm{Q} := \emptyset$;
  forall label L of P such that $\rho^\sharp(\mathrm{L}) \neq \dot{\perp}$ **do**
    forall $\mathrm{L} : \mathrm{A} \to \mathrm{L}_1; \in \mathrm{P}$ **do**
      $\mathrm{A}_c := \mathbf{Simplify}[\![\mathrm{A}]\!](\rho^\sharp(\mathrm{L}))$;
      $\mathrm{Q} := \mathrm{Q} \cup \{\mathrm{L} : \mathrm{A}_c \to \mathrm{L}_1;\}$
    end;
    if $\mathrm{L} : \mathtt{stop}; \in \mathrm{P}$ **then**
      $\mathrm{Q} := \mathrm{Q} \cup \{\mathrm{L} : \mathtt{stop};\}$
    end
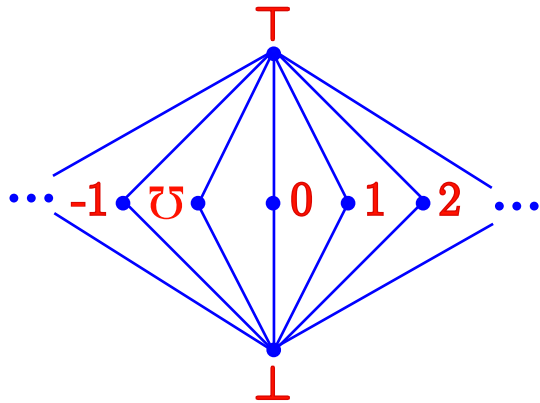 **end**;
  return Q.

Principle of
Offline Program Transformation

Offline Transformations

- A semantic program transformation can be restricted to use the only semantic information which can be discovered by a static program analysis;

  $\longrightarrow$ This can be formalized by abstract interpretation.

Example: Kildall's Constant Propagation

- Kildall's lattice (POPL'73):

$$\gamma^c(\top) = \mathbb{Z} \cup \{\Omega\}$$
$$\gamma^c(x) = \{x\}, \quad x \in \mathbb{Z} \cup \{\Omega\}$$
$$\gamma^c(\bot) = \emptyset$$

- Pointwise extension to variable environments and program labels;

Example: Kildall's Constant Propagation, Cont'd
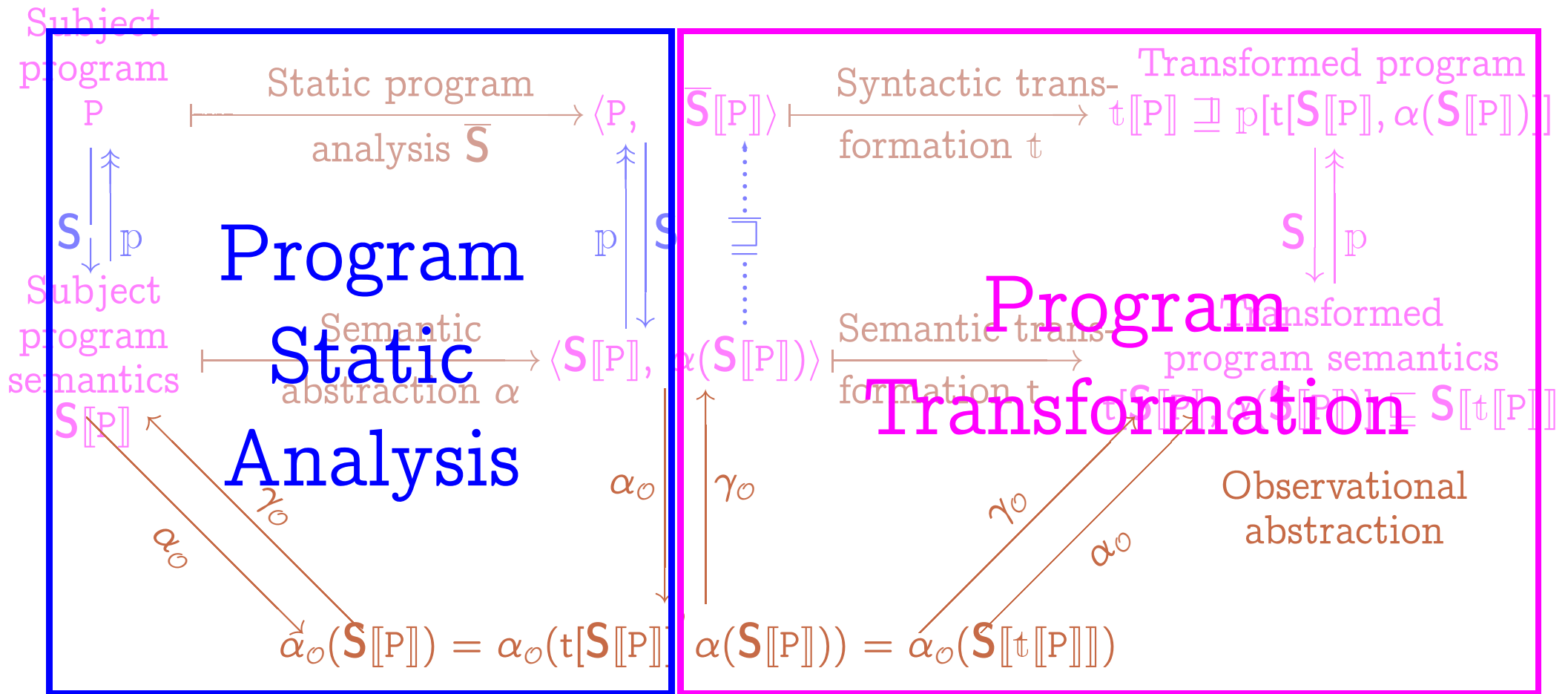
- Elementwise abstraction of a set $\mathcal{T}$ of traces:

$$\alpha^c(\mathcal{T}) = \lambda L \cdot \lambda X \cdot \ddot{\sqcup}\{\rho(X) \mid \exists \sigma \in \mathcal{T} : \exists c \in \mathbb{C} : \exists i :$$
$$\sigma_i = \langle \rho, c \rangle \wedge \mathsf{lab}[\![c]\!] = L\}$$

  where $\ddot{\sqcup}$ is the pointwise extension of the lub in Kildall's lattice

Principle of Offline Program Transformation

Static program analysis $\overline{\mathbf{S}}$ → $\langle P,$

$\mathbf{S} \downarrow \mathbb{p}$

Subject program semantics $\mathbf{S}[\![P]\!]$

Semantic abstraction α → $\langle \mathbf{S}[\![P]\!], \alpha(\mathbf{S}[\![P]\!]) \rangle$

$\mathbb{p}$ $\mathbf{S}$

Program Static Analysis

$\alpha_{\mathcal{O}}$

$\gamma_{\mathcal{O}}$

$\alpha_{\mathcal{O}}$

$\alpha_{\mathcal{O}}(\mathbf{S}[\![P]\!]) = \alpha_{\mathcal{O}}(\mathbf{t}[\mathbf{S}[\![P]\!]]$

$\overline{\mathbf{S}}[\![P]\!] \rangle$ Syntactic transformation $\mathbf{t}$ → $\mathbf{t}[\![P]\!] \sqsupseteq \mathbb{p}[\mathbf{t}[\mathbf{S}[\![P]\!], \alpha(\mathbf{S}[\![P]\!])]]$

Transformed program

$\sqsubseteq$

$\mathbf{S} \downarrow \mathbb{p}$

Program Transformation

Semantic transformation $\mathbf{t}$ → Transformed program semantics $\mathbf{S}[\![\mathbf{t}[\![P]\!]]\!]$

Observational abstraction

$\gamma_{\mathcal{O}}$

$\alpha_{\mathcal{O}}$

$\alpha(\mathbf{S}[\![P]\!])) = \alpha_{\mathcal{O}}(\mathbf{S}[\![\mathbf{t}[\![P]\!]]\!])$

Principle of Offline Program Transformation
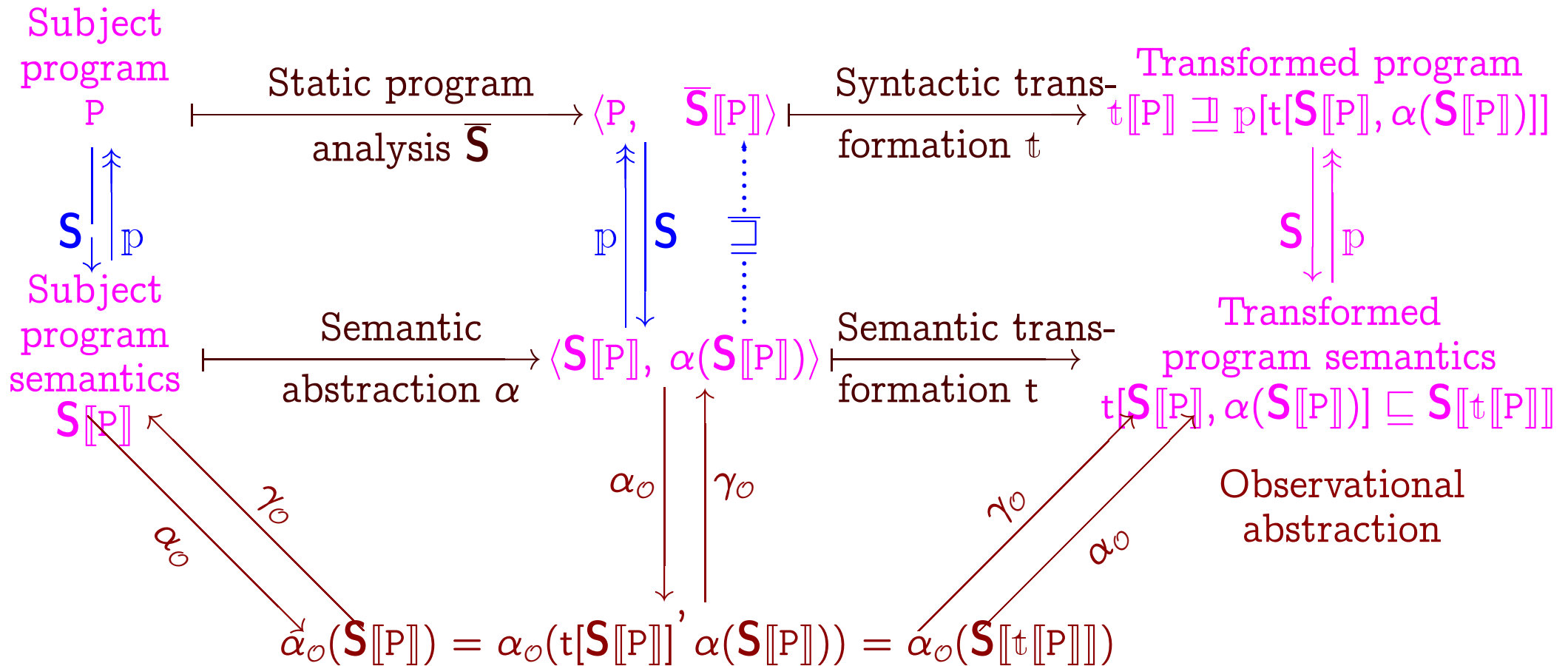
Applications of the formalization of program transformation

Other Program Transformations Formally Handled in the Same Way

- In this talk, the approach was illustrated on the trivial constant propagation example;

- The same approach has been successfully applied to:
  - Blocking command elimination (ENTCS v. 45);
  - Online, offline and mixline partial evaluation (POPL'02);
  - Program monitoring (POPL'02);
  - Program reduction (e.g. transition compression), Slicing, etc.

Conclusion

Conclusion

- Program transformation is understood as an abstraction of a semantic transformation of run-time execution;

- Leads to a unified framework for semantics-based program analysis and transformation;

- The benefit is presently purely foundational and conceptual;

- Practical application: reanalysis of assembler code from source requires the formalization of the compilation process (work of Xavier RIVAL at ENS).