Formalization of Program Transformation by Abstract Interpretation

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

CS Dept., Courant Inst. of Math. Sci., NYU Jan 11, 2002

Content

This work was supported in part by the RTD project IST-1999-20527 DAEDALUS of the european IST FP5 programme.

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

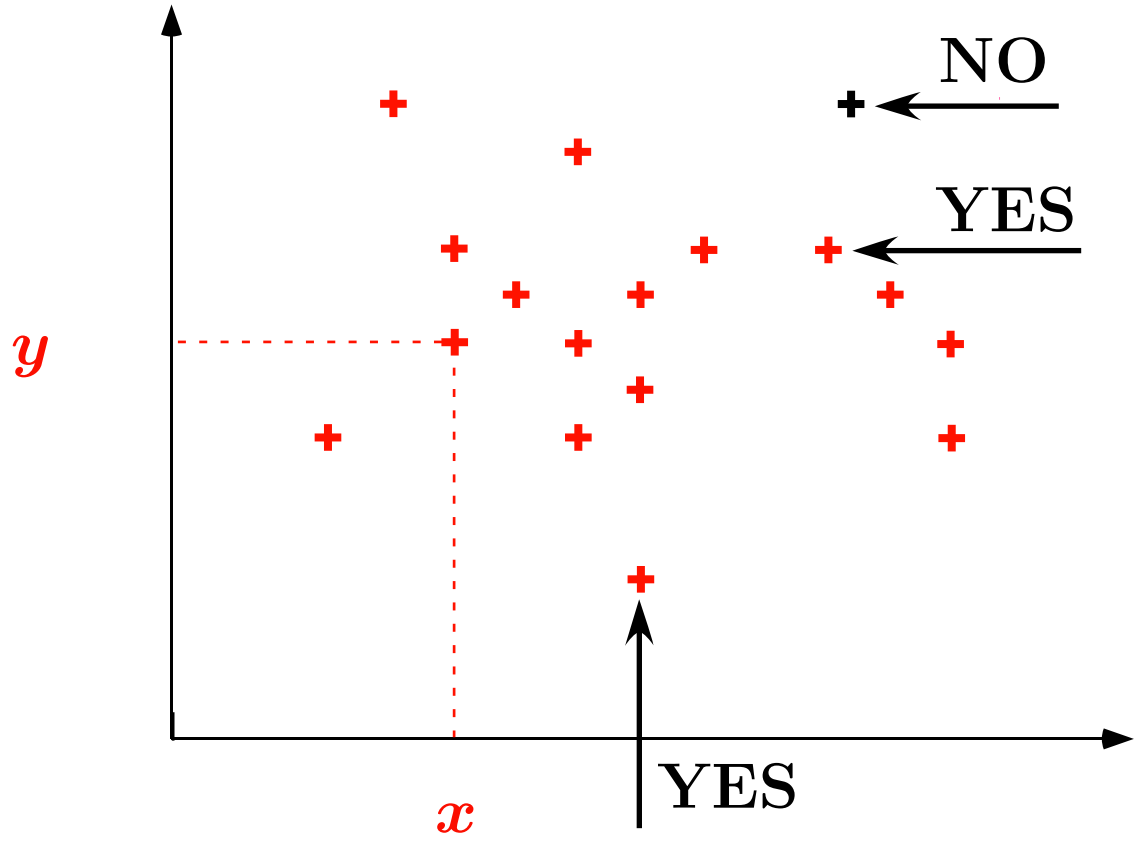
Abstraction: intuition

- 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

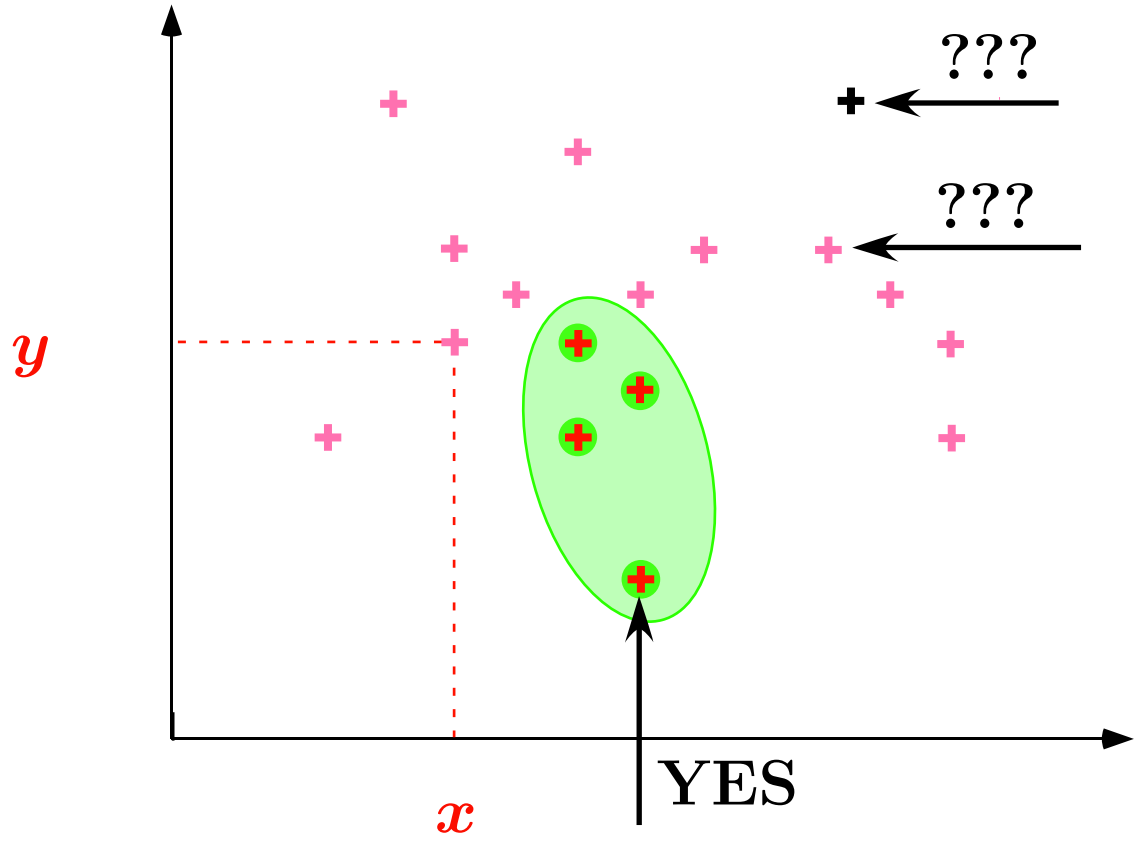
An [in]finite set of points;



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:



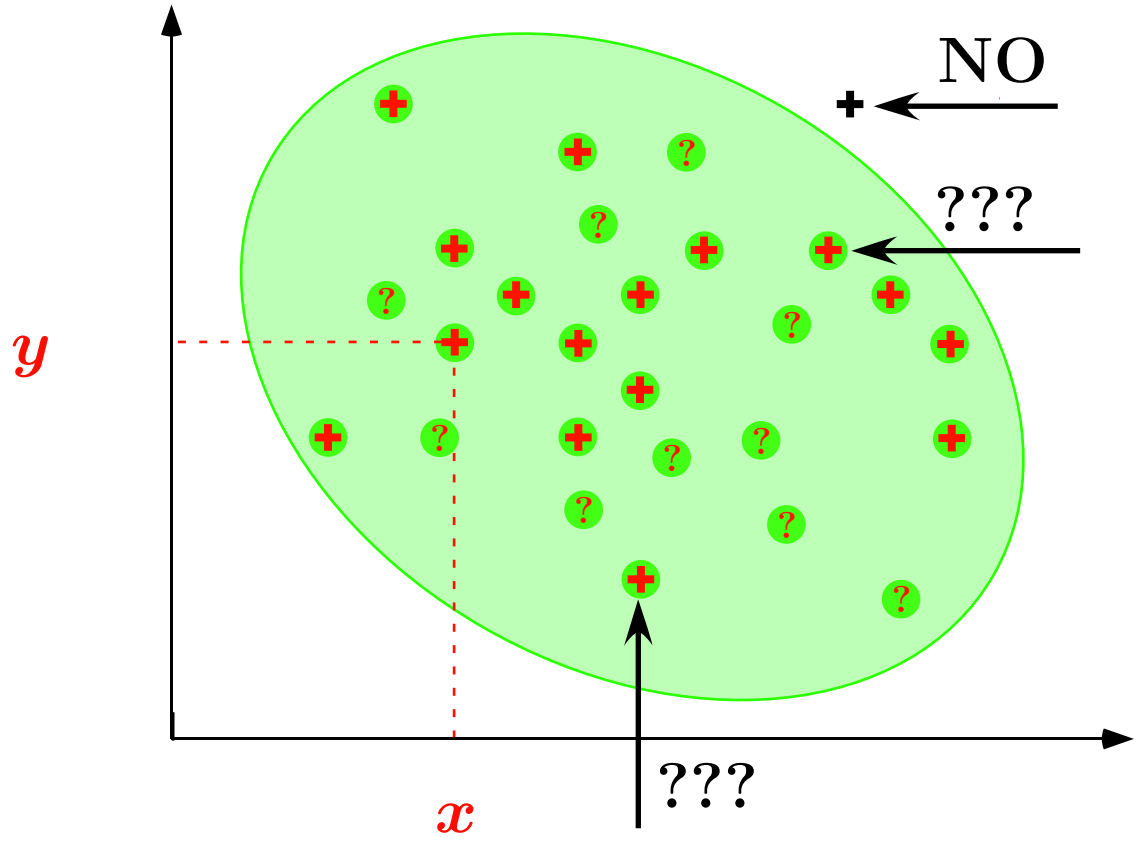
From Below

Is that point in the concrete set?

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

Conservative answer

Approximation of an [in]finite set of points:



From Above

Is that point in the concrete set?

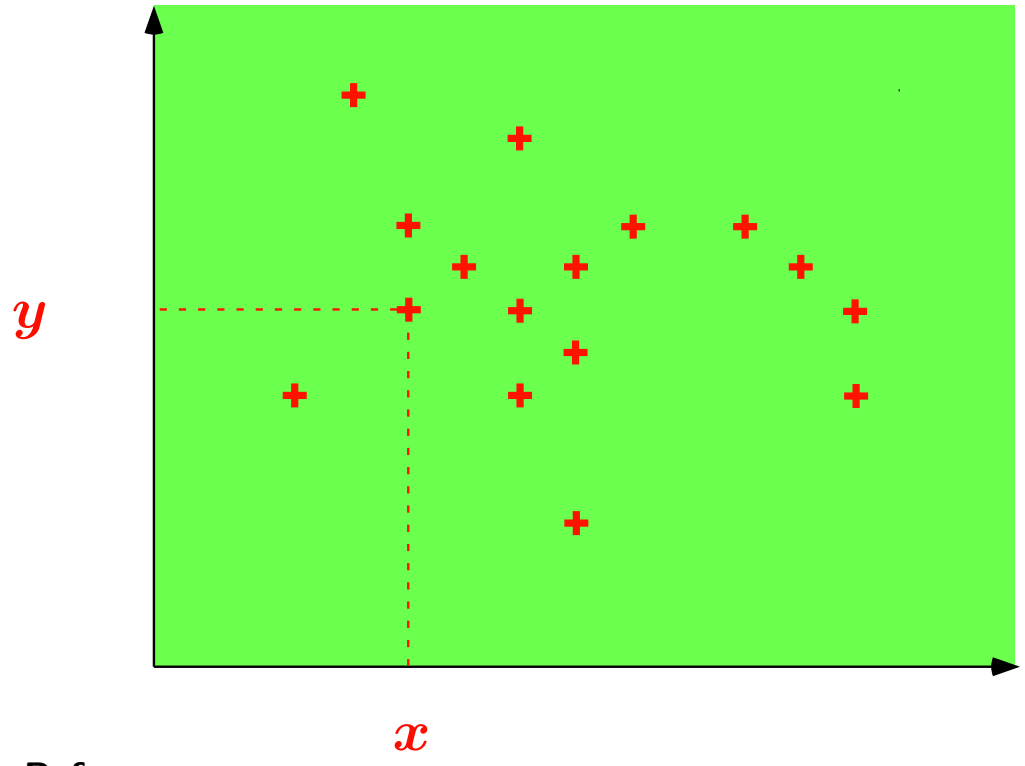
$\{\ldots, \langle 19,\ 77\rangle, \ldots,$

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

Conservative answer

Intuition Behind Effective Computable Abstraction

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

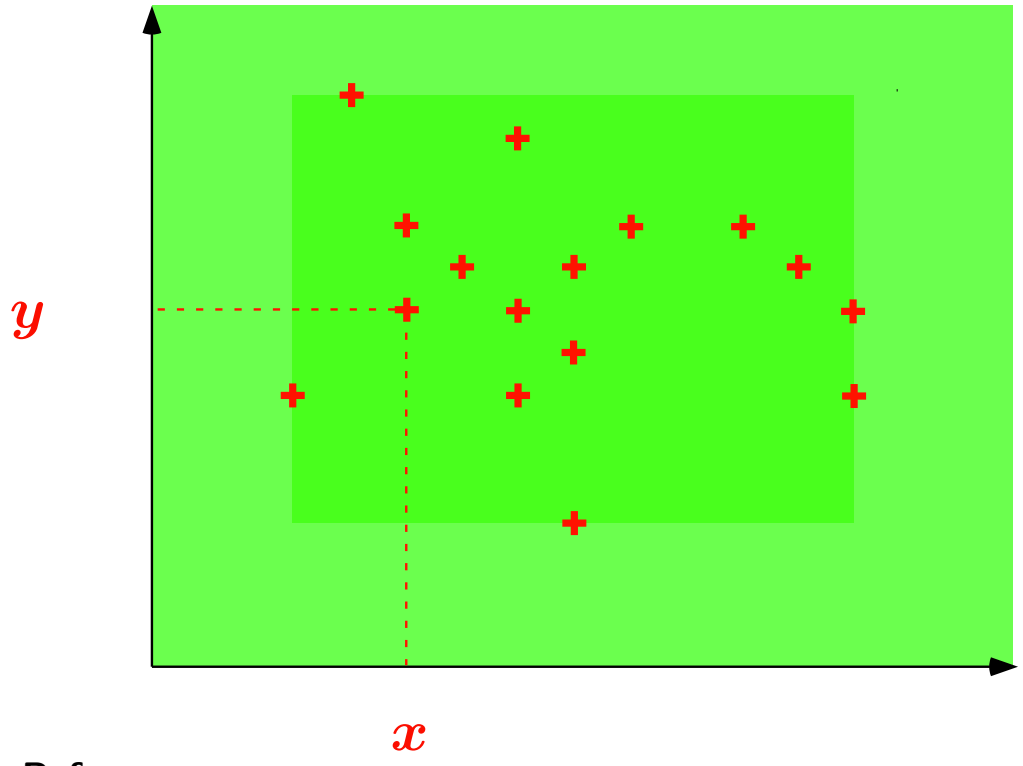


$$\begin{cases} x \geq 0 \\ y \geq 0 \end{cases}$$

Reference

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

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

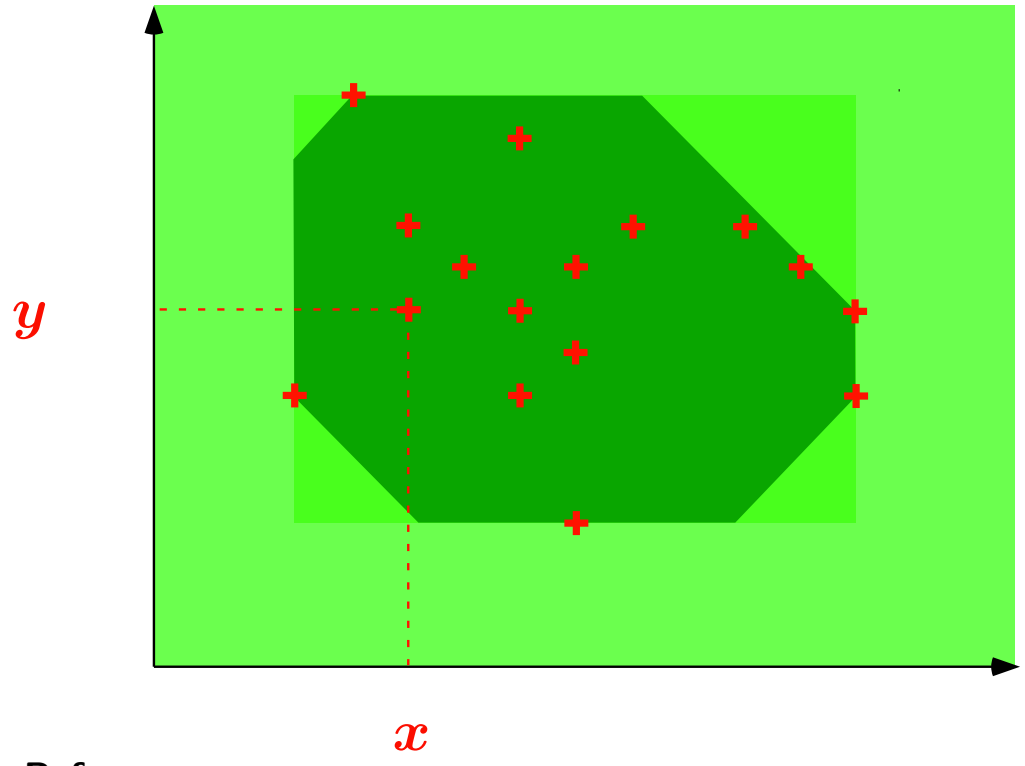


$$\begin{cases} x \in [19,\ 78] \\ y \in [20,\ 01] \end{cases}$$

Reference

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

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

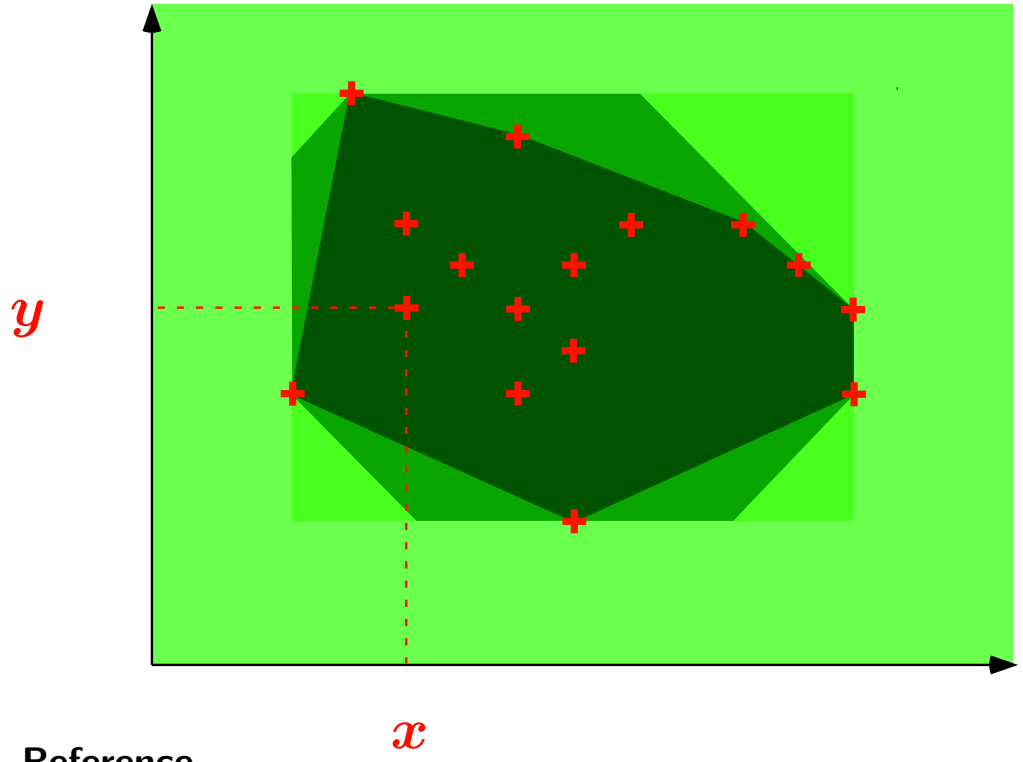


$$\begin{cases} 1 \leq x \leq 9 \\ x + y \leq 78 \\ 1 \leq y \leq 9 \\ x - y \leq 99 \end{cases}$$

Reference

[3] A. Min . 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]

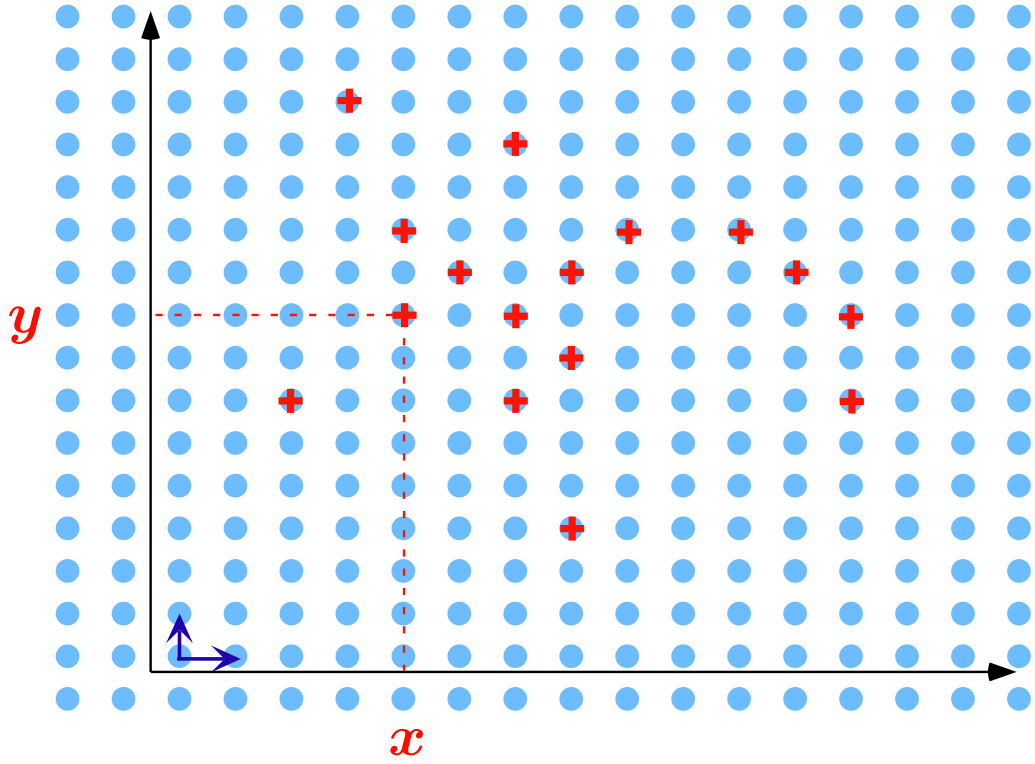


$$\begin{cases} 19x + 78y \leq 2000 \\ 20x + 01y \geq 0 \end{cases}$$

Reference

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

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

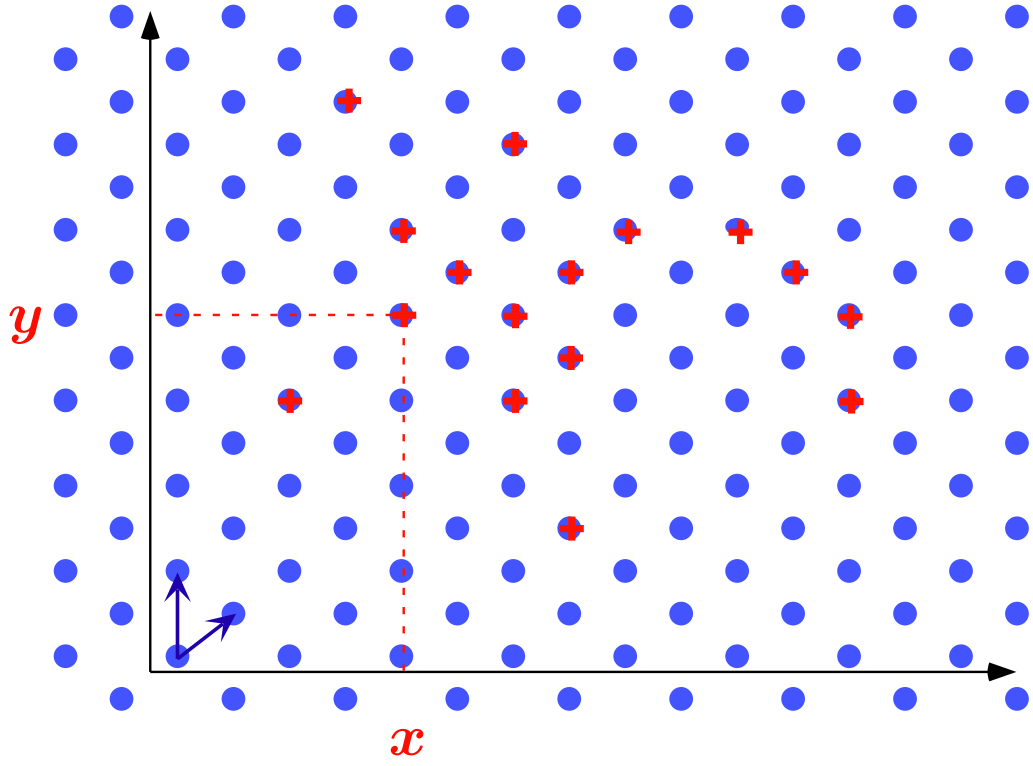


$$\begin{cases} x = 19 \bmod 78 \\ y = 20 \bmod 99 \end{cases}$$

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]

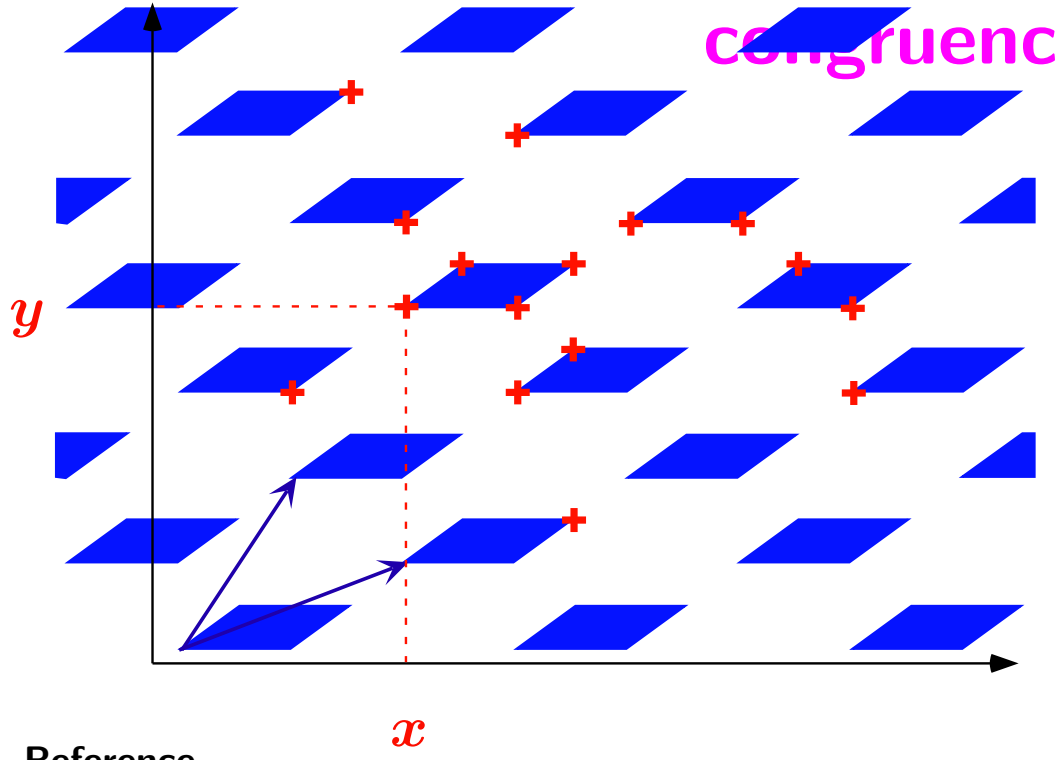


$$\begin{cases} 1x + 9y = 7 \bmod 8 \\ 2x - 1y = 9 \bmod 9 \end{cases}$$

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

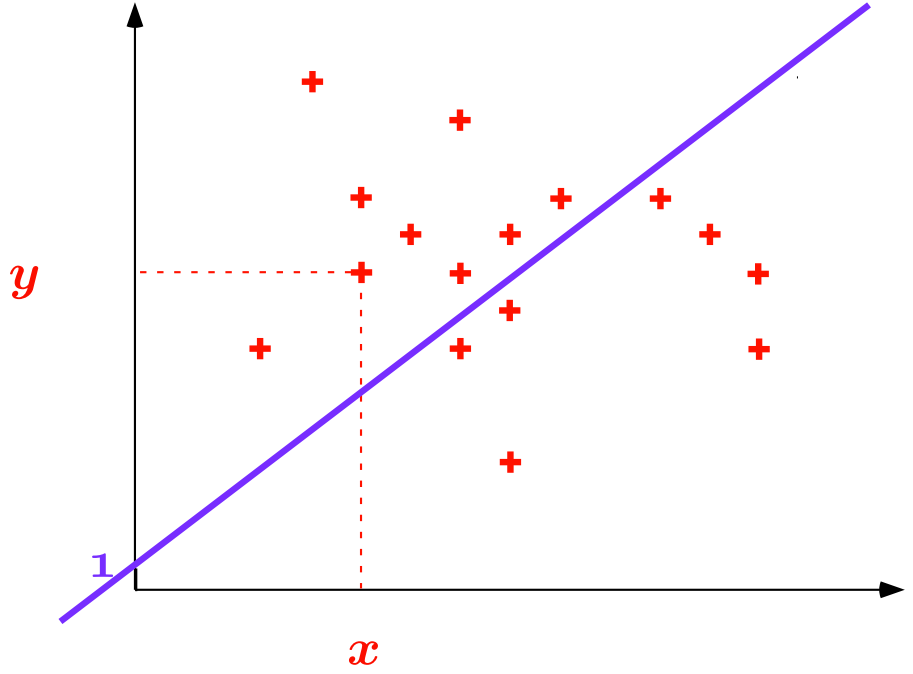
[7] F. Masdupuy. Array operations abstraction using semantic analysis of trapezoid congruences. In *ACM Int. Conf. on Supercomputing, ICS '92*, pages 226–235, 1992.

Conservative Approximation and Information Loss

Intuition Behind Sound/Conservative Approximation

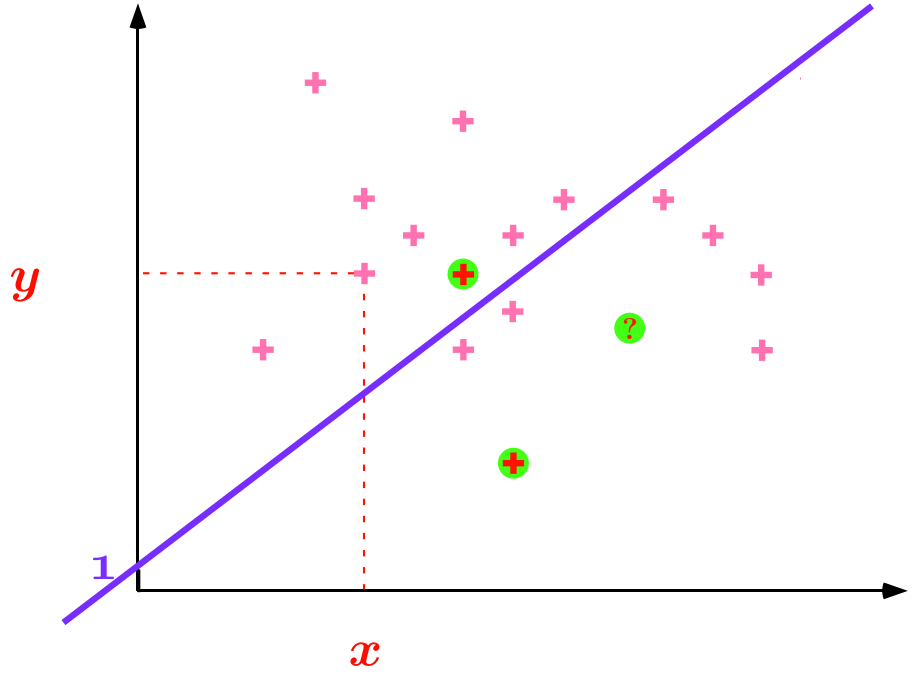
Conservative Approximation

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



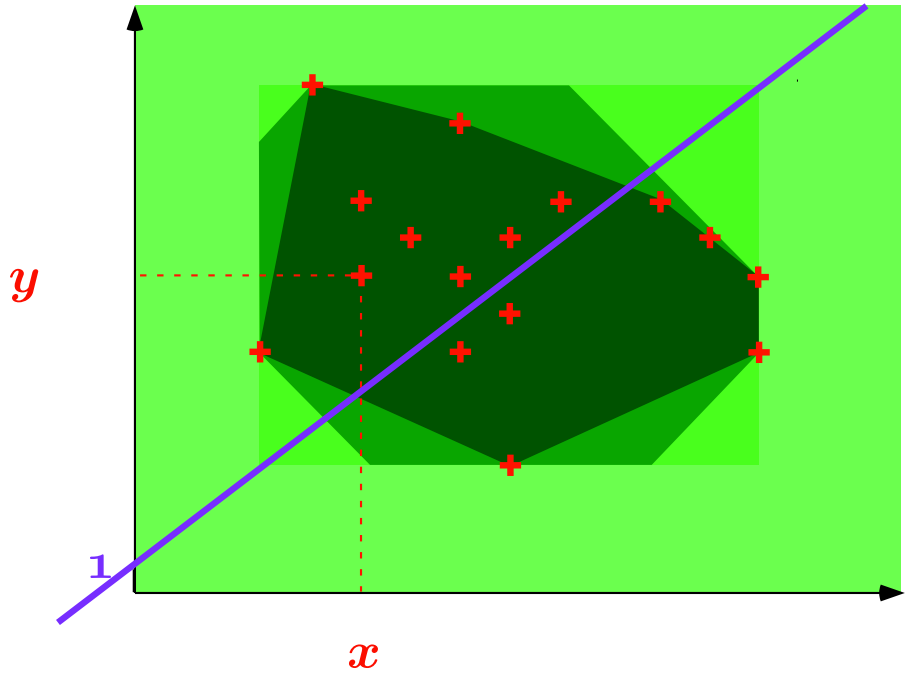
Conservative Approximation

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



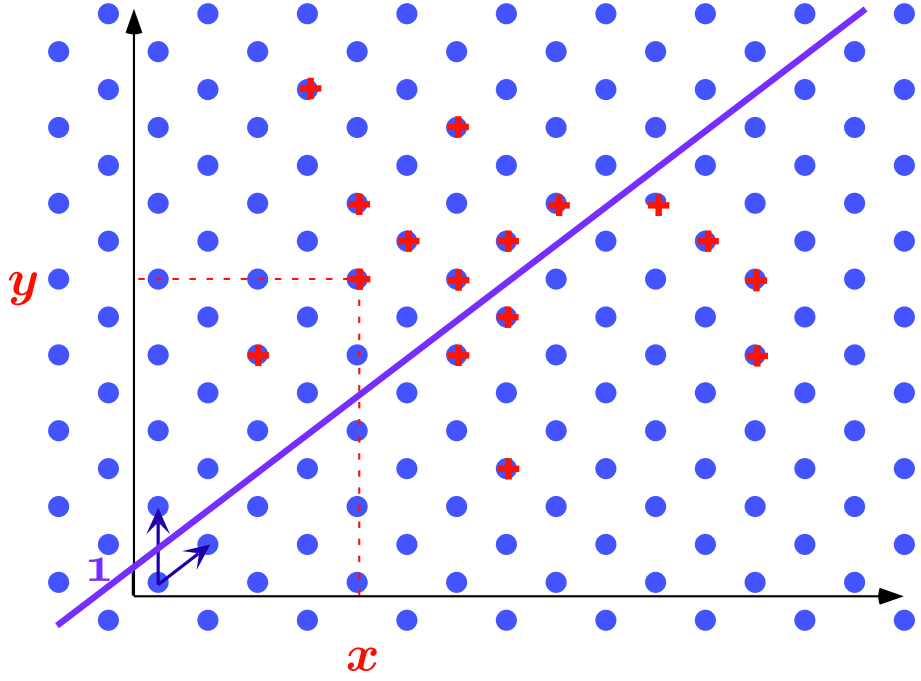
Conservative Approximation

- 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**



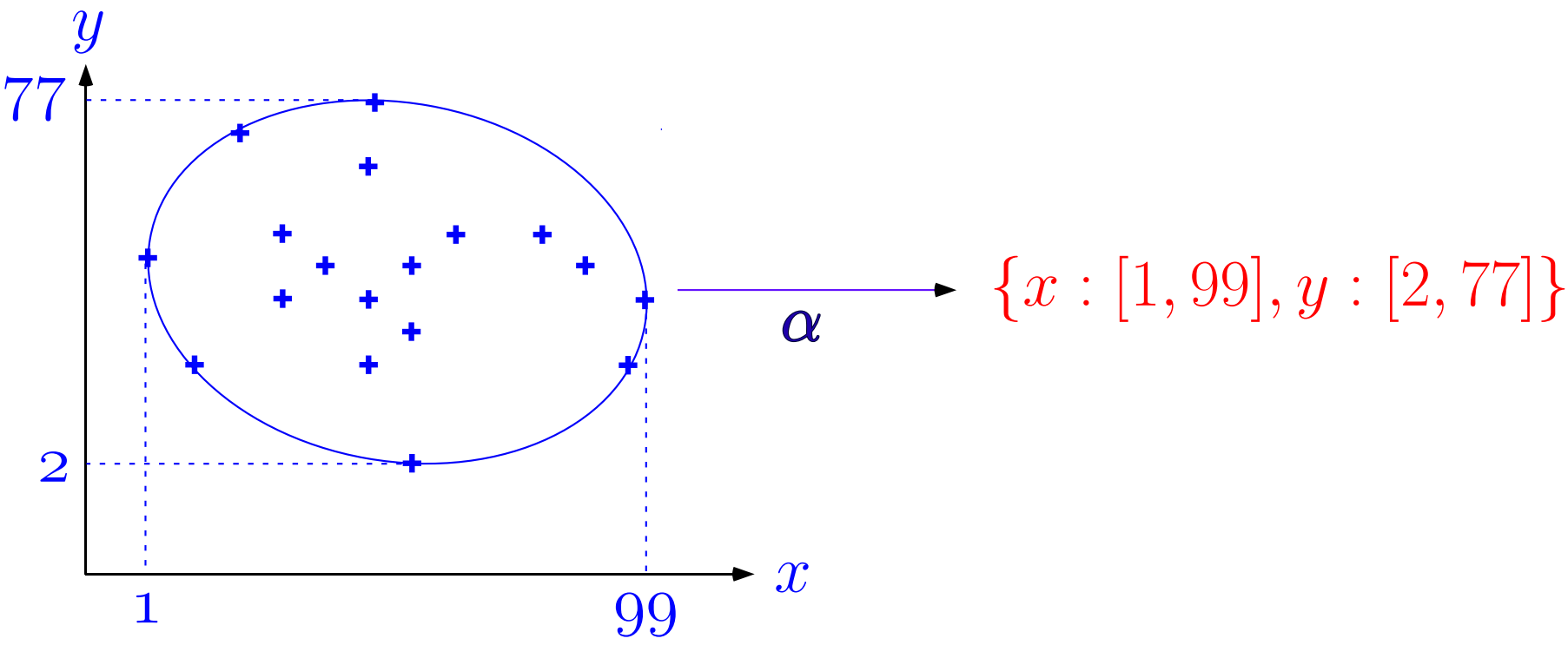
Intuition Behind Information Loss

Information Loss

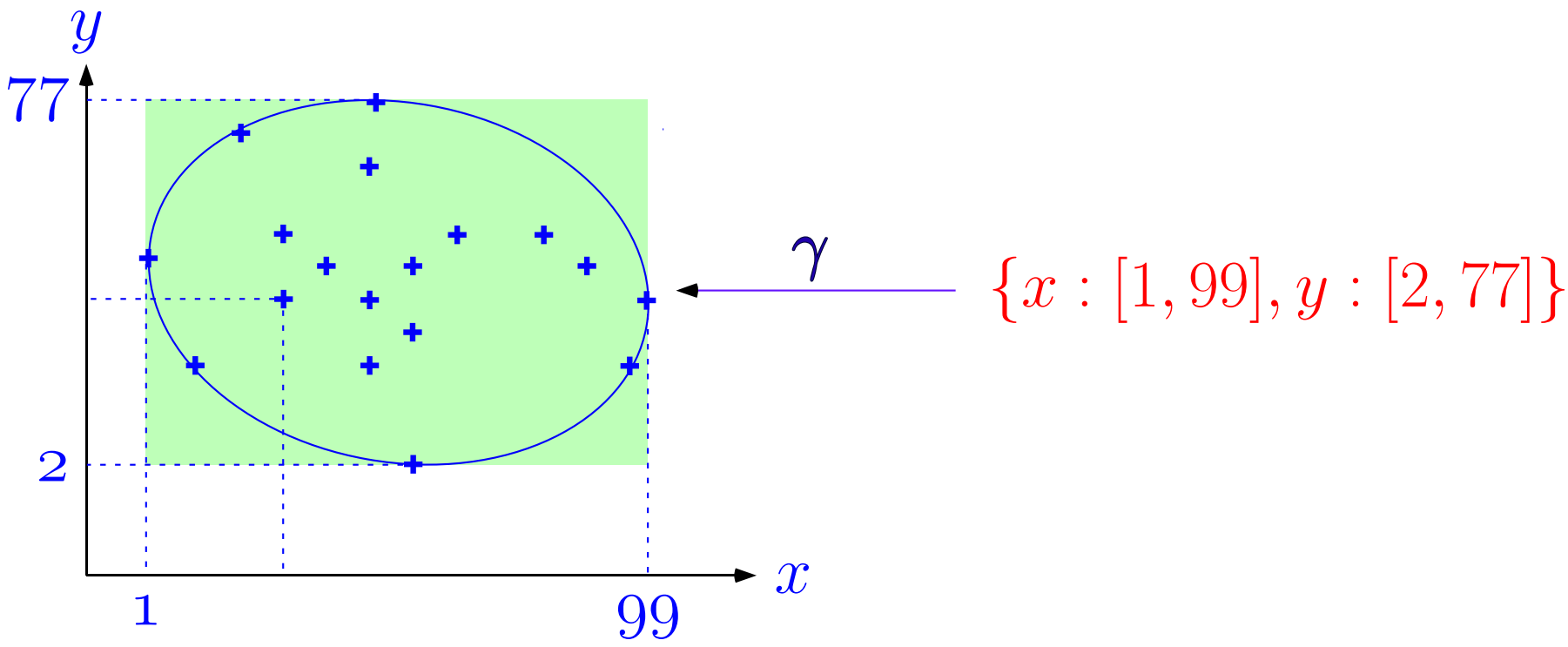
- All answers given by the abstract semantics are always correct with respect to the concrete semantics;
- Because of the information loss, not all questions can be definitely answered with the abstract semantics;
- The more concrete semantics can answer more questions;
- The more abstract semantics are more simple.

Very Basic Elements of Abstract Interpretation Theory

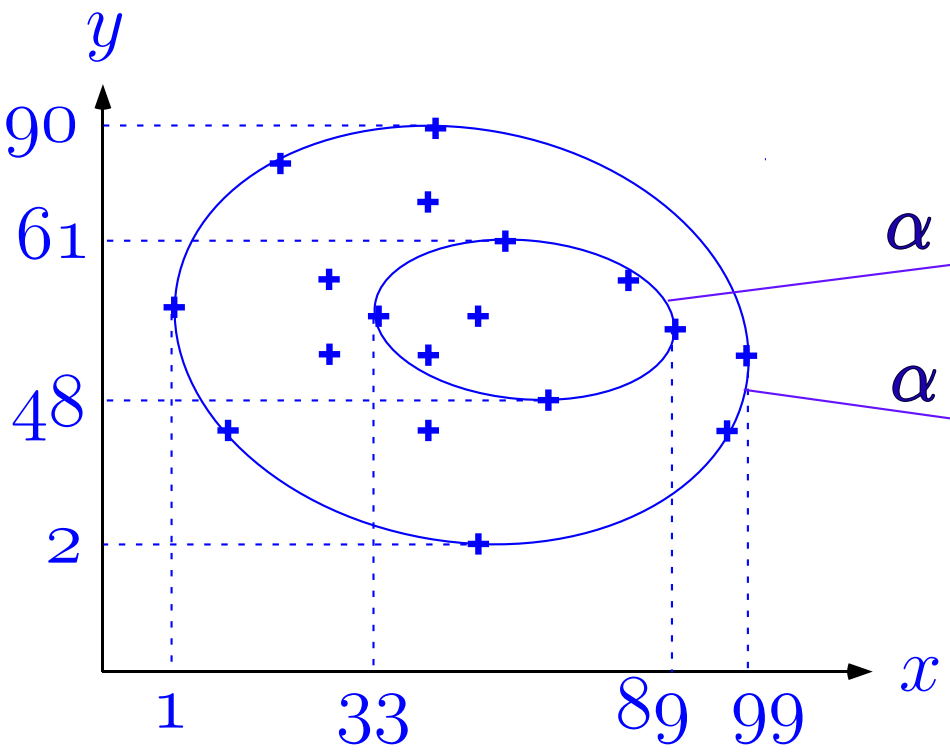
Abstraction α



Concretization γ



The Abstraction α is Monotone



α

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

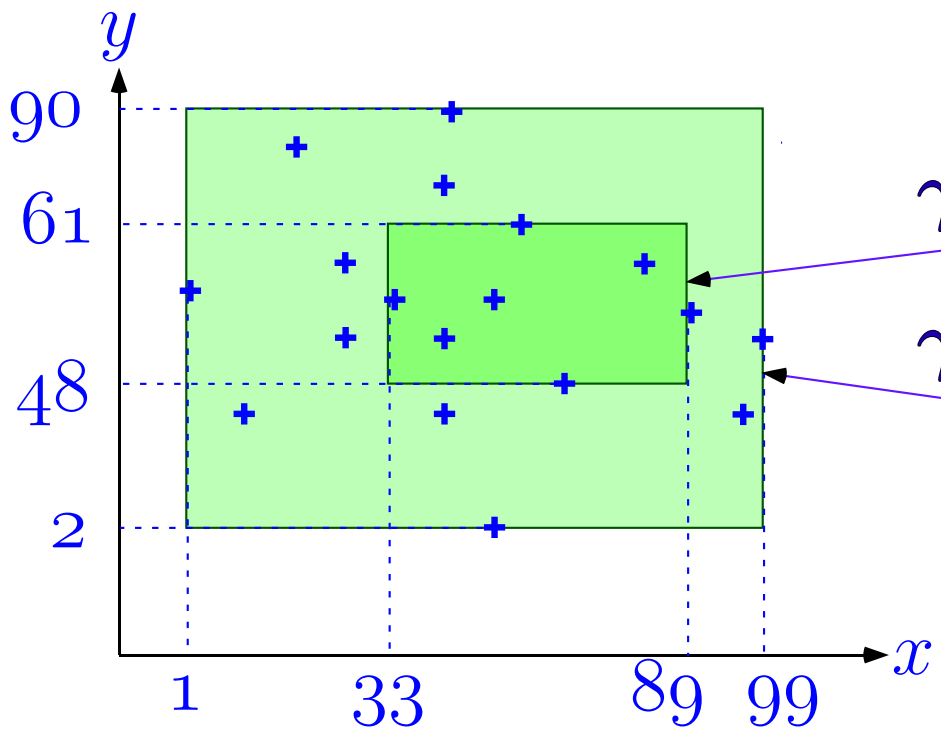
$\sqsubseteq$

α

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

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

The Concretization γ is Monotone



γ

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

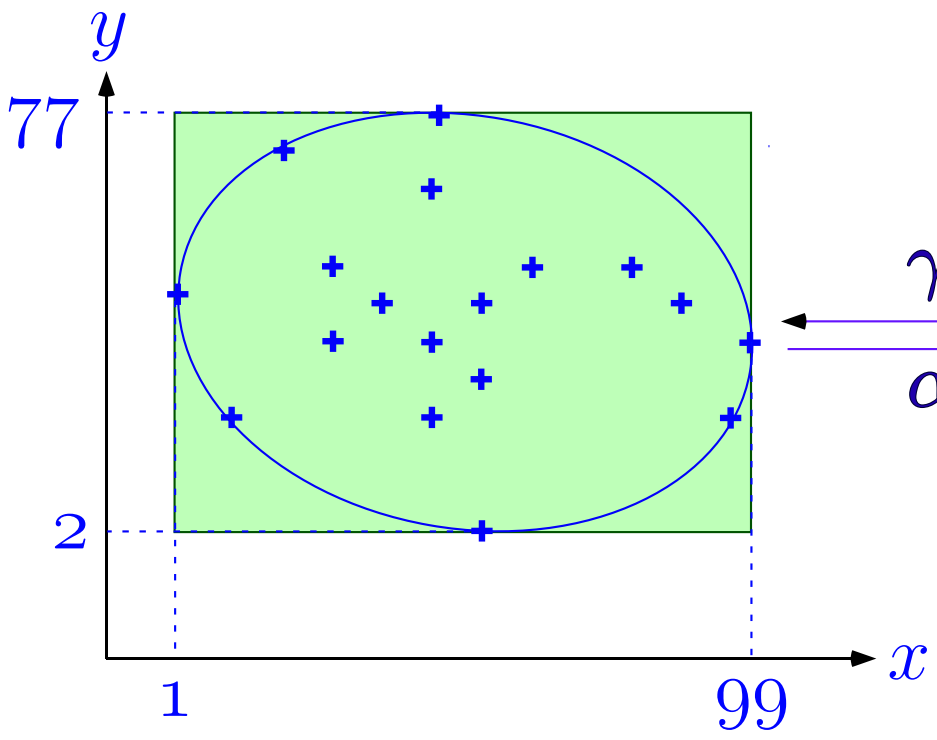
$\sqsubseteq$

γ

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

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

The $\gamma \circ \alpha$ Composition



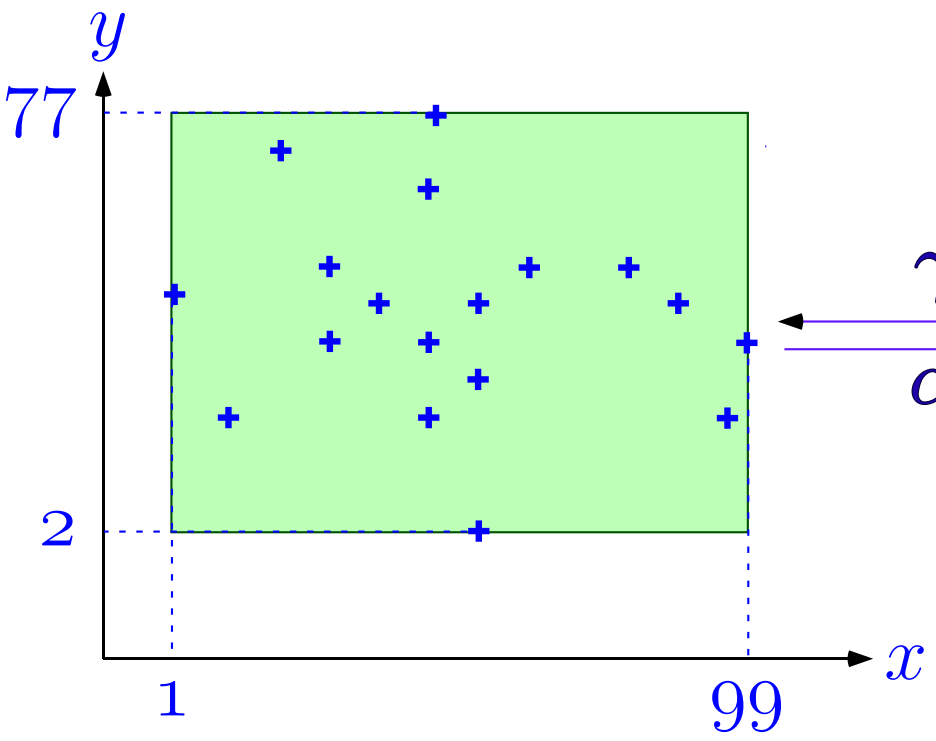
γ

α

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

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

The $\alpha \circ \gamma$ Composition



γ

α

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

$$=$$

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

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

Galois Connection [1]

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

is defined as

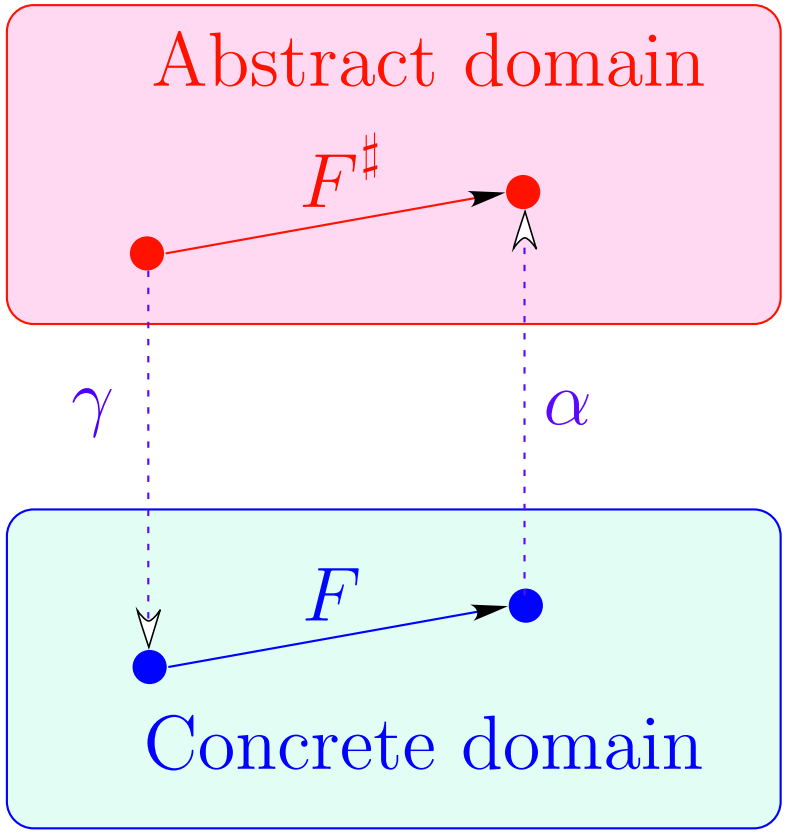
- α is monotone
- γ is monotone
- $X \subseteq \gamma \circ \alpha(X)$
- $\alpha \circ \gamma(Y) \sqsubseteq Y$

iff

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

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

Function Abstraction

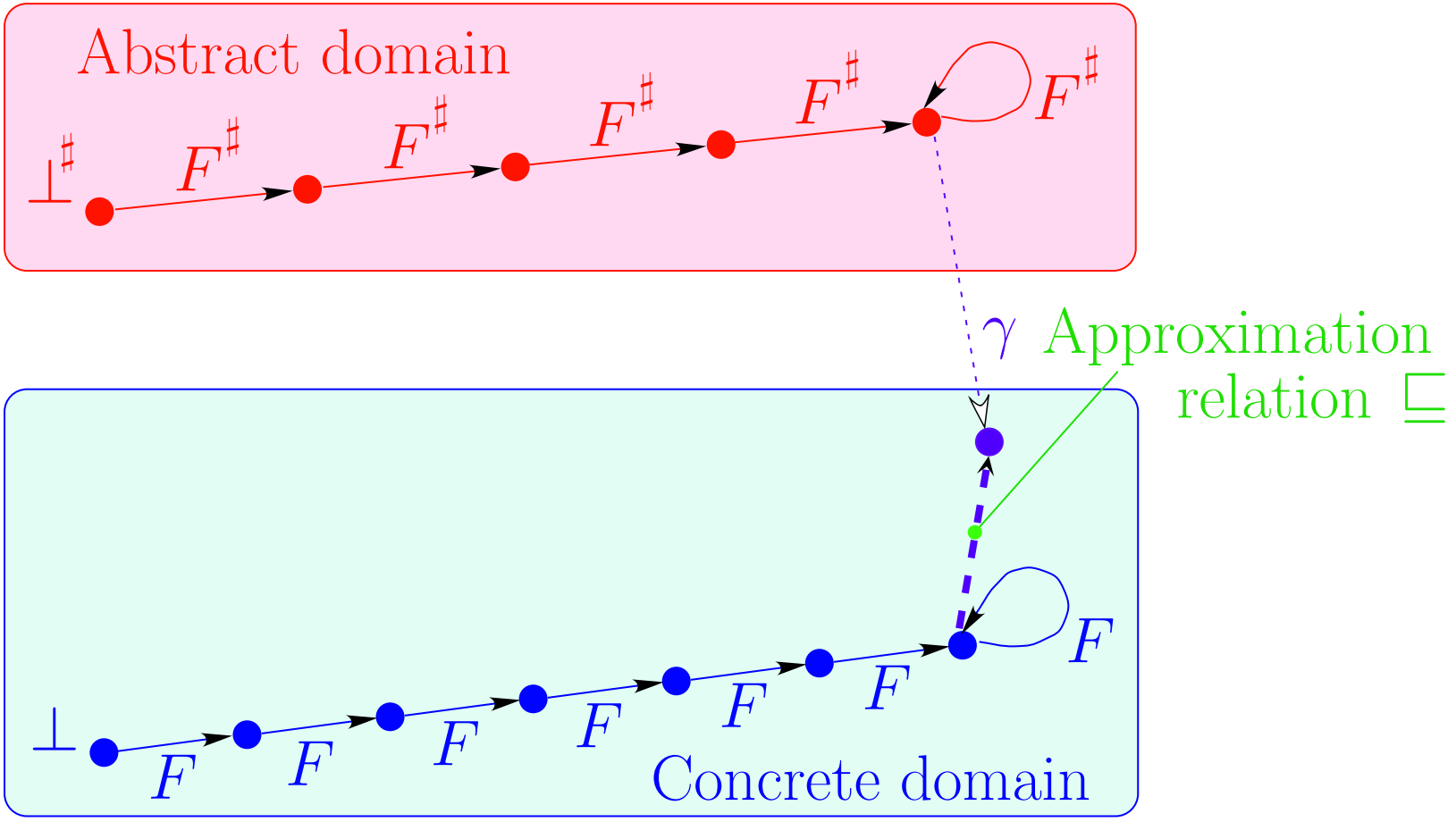


$$F^\sharp = \alpha \circ F \circ \gamma$$

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

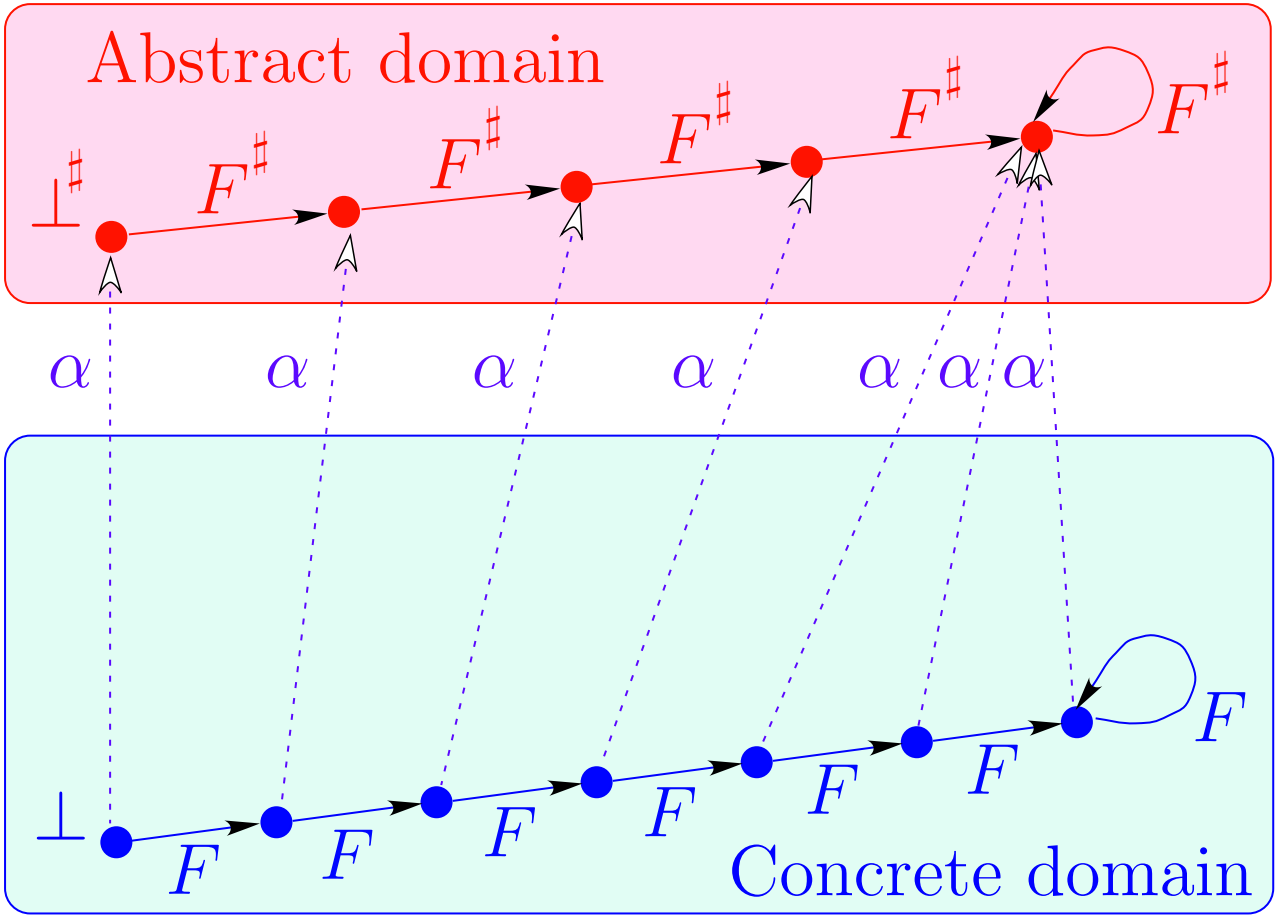
$$\langle P \overset{\text{mon}}{\longmapsto} P, \dot{\subseteq} \rangle \underset{\lambda F \cdot \alpha \circ F \circ \gamma}{\overset{\lambda F^\sharp \cdot \gamma \circ F^\sharp \circ \alpha}{\leftrightarrows}} \langle Q \overset{\text{mon}}{\longmapsto} Q, \dot{\sqsubseteq} \rangle$$

Approximate Fixpoint Abstraction



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

Exact Fixpoint Abstraction



$$\alpha \circ F = F^\sharp \circ \alpha \;\Rightarrow\; \alpha(\mathbf{lfp}\, F) = \mathbf{lfp}\, F^\sharp$$

Exact/Approximate Fixpoint Abstraction

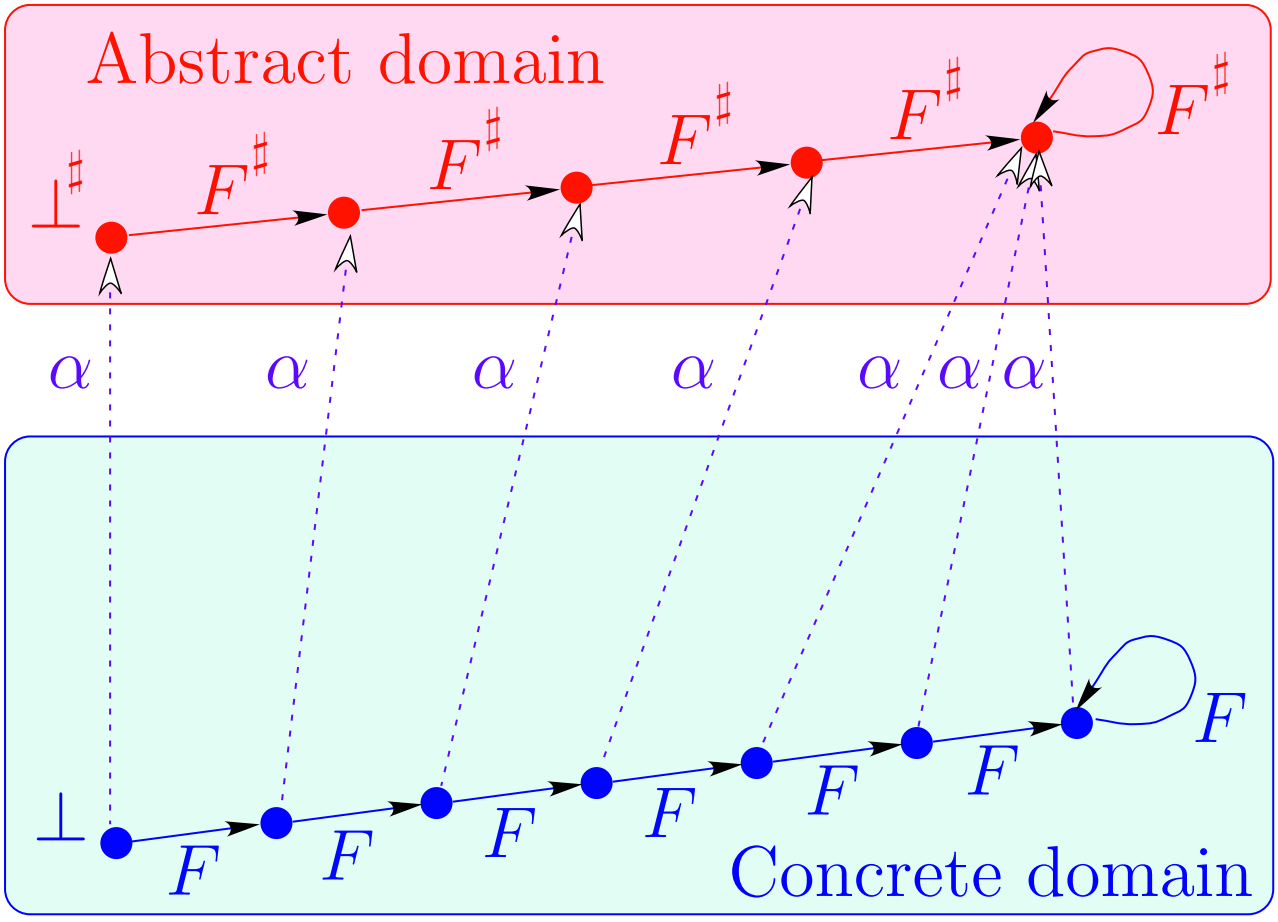
Exact Abstraction:

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

Approximate Abstraction:

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

Exact Fixpoint Abstraction



$$\alpha \circ F = F^\sharp \circ \alpha \;\Rightarrow\; \alpha(\mathbf{lfp}\, F) = \mathbf{lfp}\, F^\sharp$$

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]
- **Hierarchies of Semantics (including Proofs)** [POPL 92]
- **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 runtime behaviors, which is necessary
- to validate the applicable transformations.

Present Objective

Our present objective is **quite different**:

- Formalize the program transformation itself 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

$\mathtt{a}: \mathtt{X} := ? \longrightarrow \mathtt{b};$	random assignment/input
$\mathtt{b}: \mathtt{Y} := 1 \longrightarrow \mathtt{c};$	assignment
$\mathtt{c}: (\mathtt{X} \leq 0) \longrightarrow \mathtt{f};$	nondeterministic guard
$\mathtt{c}: (\mathtt{X} > 0) \longrightarrow \mathtt{d};$	
$\mathtt{d}: \mathtt{X} := \mathtt{X} - \mathtt{Y} \longrightarrow \mathtt{e};$	
$\mathtt{e}: \mathtt{skip} \longrightarrow \mathtt{c};$	branching
$\mathtt{f}: \mathtt{stop};$	stop

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:

$\mathtt{a} : \mathtt{X} := ? \longrightarrow \mathtt{b};$

$\mathtt{b} : \mathtt{Y} := 1 \longrightarrow \mathtt{c};$

$\mathtt{c} : (\mathtt{X} \leq 0) \longrightarrow \mathtt{f};$

$\mathtt{c} : (\mathtt{X} > 0) \longrightarrow \mathtt{d};$

$\mathtt{d} : \mathtt{X} := \mathtt{X} - \mathtt{Y} \longrightarrow \mathtt{e};$

$\mathtt{e} : \mathtt{skip} \longrightarrow \mathtt{c};$

$\mathtt{f} : \mathtt{stop};$

Transformed program:

$\mathtt{a} : \mathtt{X} := ? \longrightarrow \mathtt{b};$

$\mathtt{b} : \mathtt{Y} := 1 \longrightarrow \mathtt{c};$

$\mathtt{c} : (\mathtt{X} \leq 0) \longrightarrow \mathtt{f};$

$\mathtt{c} : (\mathtt{X} > 0) \longrightarrow \mathtt{d};$

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

$\mathtt{e} : \mathtt{skip} \longrightarrow \mathtt{c};$

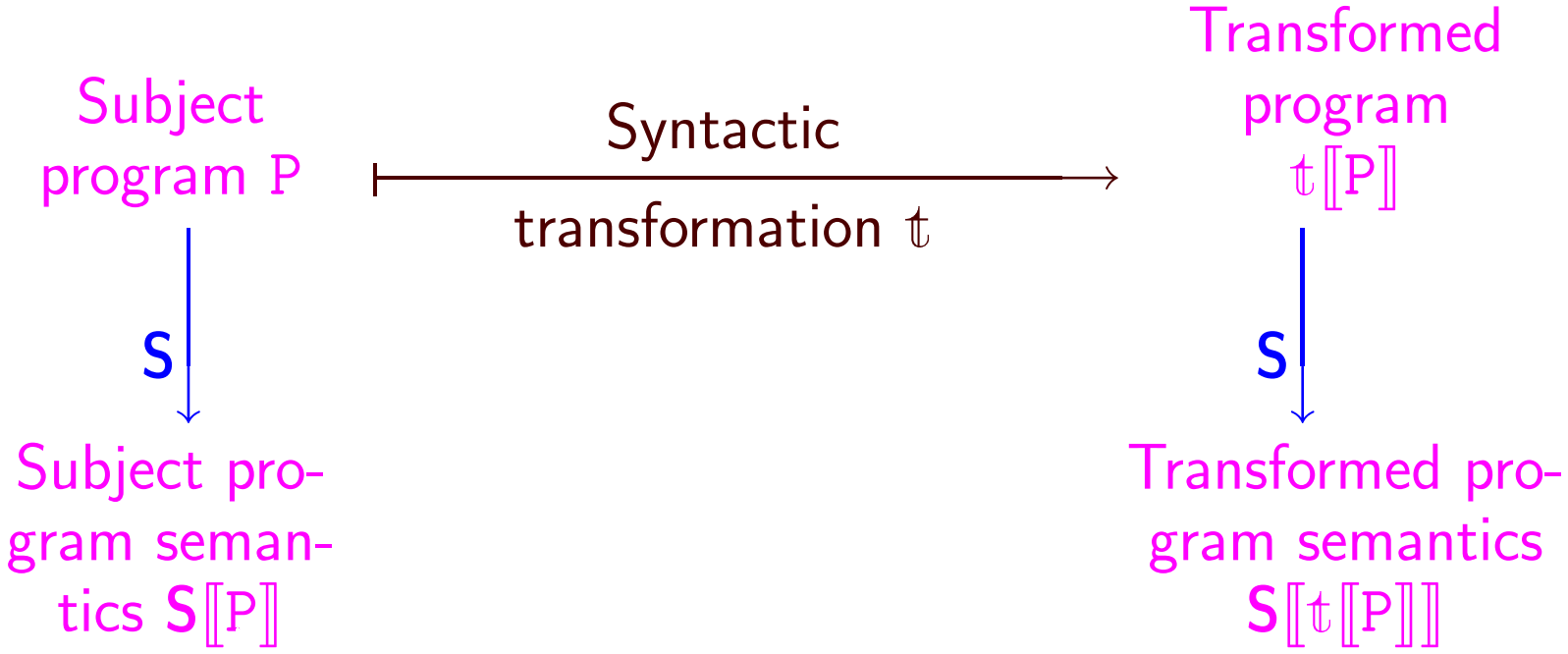
$\mathtt{f} : \mathtt{stop};$

Principle of Online Program Transformation (1)

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)



The Prefix Trace Semantics

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

$$\mathtt{a} : \mathtt{X} := ? \longrightarrow \mathtt{b};$$
$$\mathtt{b} : \mathtt{Y} := 1 \longrightarrow \mathtt{c};$$
$$\mathtt{c} : (\mathtt{X} \leq 0) \longrightarrow \mathtt{f};$$
$$\mathtt{c} : (\mathtt{X} > 0) \longrightarrow \mathtt{d};$$
$$\mathtt{d} : \mathtt{X} := \mathtt{X} - \mathtt{Y} \longrightarrow \mathtt{e};$$
$$\mathtt{e} : \mathtt{skip} \longrightarrow \mathtt{c};$$
$$\mathtt{f} : \mathtt{stop};$$

$$\langle \mathtt{a} : \mathtt{X} := ? \longrightarrow \mathtt{b};,\ [\mathtt{X} : \mho, \mathtt{Y} : \mho] \rangle$$
$$\langle \mathtt{b} : \mathtt{Y} := 1 \longrightarrow \mathtt{c};,\ [\mathtt{X} : 1, \mathtt{Y} : \mho] \rangle$$
$$\langle \mathtt{c} : (\mathtt{X} > 0) \longrightarrow \mathtt{d};,\ [\mathtt{X} : 1, \mathtt{Y} : 1] \rangle$$
$$\langle \mathtt{d} : \mathtt{X} := \mathtt{X} - \mathtt{Y} \longrightarrow \mathtt{e};,\ [\mathtt{X} : 1, \mathtt{Y} : 1] \rangle$$
$$\langle \mathtt{e} : \mathtt{skip} \longrightarrow \mathtt{c};,\ [\mathtt{X} : 0, \mathtt{Y} : 1] \rangle$$
$$\langle \mathtt{c} : (\mathtt{X} \leq 0) \longrightarrow \mathtt{f};,\ [\mathtt{X} : 0, \mathtt{Y} : 1] \rangle$$
$$\langle \mathtt{f} : \mathtt{stop};,\ [\mathtt{X} : 0, \mathtt{Y} : 1] \rangle$$

Semantics of the Transformed program

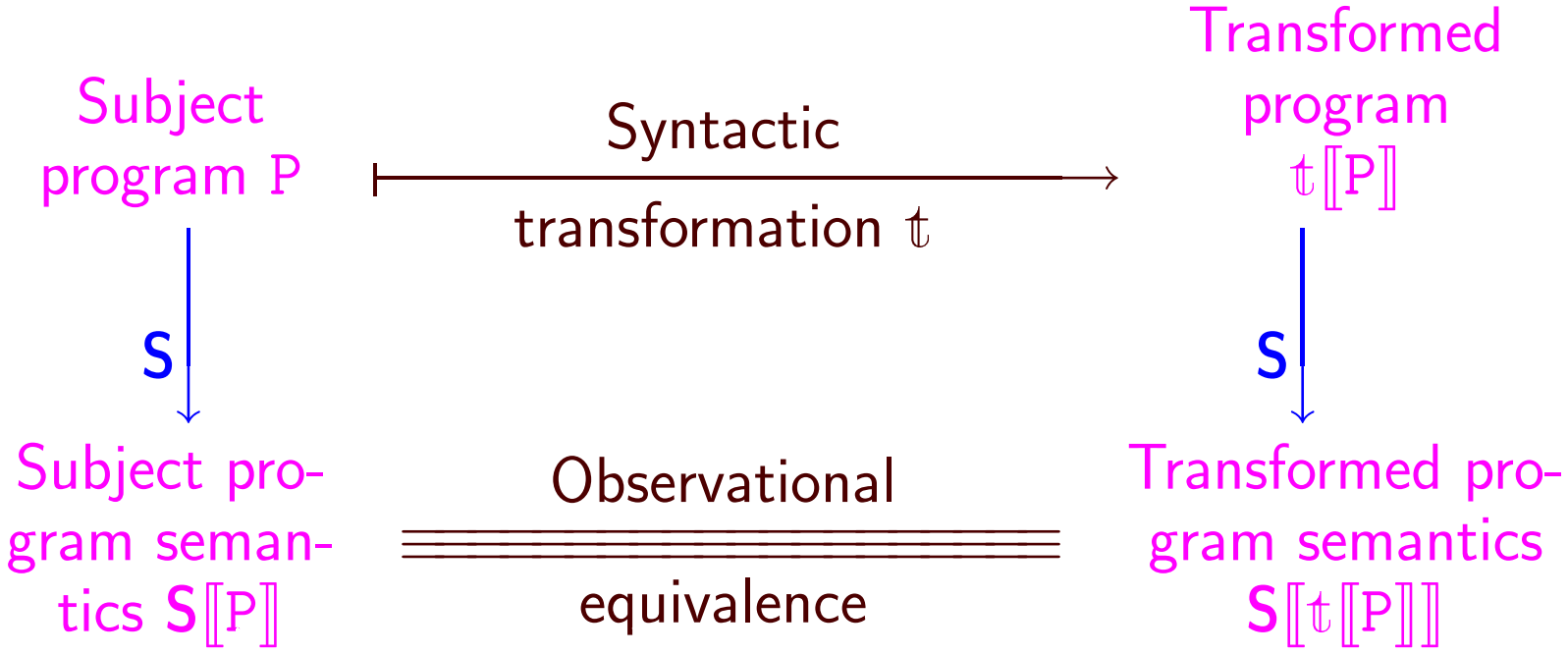
$$\mathtt{a} : \mathtt{X} := ? \longrightarrow \mathtt{b};$$
$$\mathtt{b} : \mathtt{Y} := 1 \longrightarrow \mathtt{c};$$
$$\mathtt{c} : (\mathtt{X} \leq 0) \longrightarrow \mathtt{f};$$
$$\mathtt{c} : (\mathtt{X} > 0) \longrightarrow \mathtt{d};$$
$$\mathtt{d} : \mathtt{X} := \mathtt{X} - \overset{1}{\cancel{\mathtt{Y}}} \longrightarrow \mathtt{e};$$
$$\mathtt{e} : \mathtt{skip} \longrightarrow \mathtt{c};$$
$$\mathtt{f} : \mathtt{stop};$$

$$\langle \mathtt{a} : \mathtt{X} := ? \longrightarrow \mathtt{b};,\ [\mathtt{X} : \mho, \mathtt{Y} : \mho] \rangle$$
$$\langle \mathtt{b} : \mathtt{Y} := 1 \longrightarrow \mathtt{c};,\ [\mathtt{X} : 1, \mathtt{Y} : \mho] \rangle$$
$$\langle \mathtt{c} : (\mathtt{X} > 0) \longrightarrow \mathtt{d};,\ [\mathtt{X} : 1, \mathtt{Y} : 1] \rangle$$
$$\langle \mathtt{d} : \mathtt{X} := \mathtt{X} - \overset{1}{\cancel{\mathtt{Y}}} \longrightarrow \mathtt{e};,\ [\mathtt{X} : 1, \mathtt{Y} : 1] \rangle$$
$$\langle \mathtt{e} : \mathtt{skip} \longrightarrow \mathtt{c};,\ [\mathtt{X} : 0, \mathtt{Y} : 1] \rangle$$
$$\langle \mathtt{c} : (\mathtt{X} \leq 0) \longrightarrow \mathtt{f};,\ [\mathtt{X} : 0, \mathtt{Y} : 1] \rangle$$
$$\langle \mathtt{f} : \mathtt{stop};,\ [\mathtt{X} : 0, \mathtt{Y} : 1] \rangle$$

Principle of Online Program Transformation (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)



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

$\mathtt{a} : \mathtt{X} := ? \longrightarrow \mathtt{b};$

$\mathtt{b} : \mathtt{Y} := 1 \longrightarrow \mathtt{c};$

$\mathtt{c} : (\mathtt{X} \leq 0) \longrightarrow \mathtt{f};$

$\mathtt{c} : (\mathtt{X} > 0) \longrightarrow \mathtt{d};$

$\mathtt{d} : \mathtt{X} := \mathtt{X} - \overset{1}{\not\mathtt{Y}} \longrightarrow \mathtt{e};$

$\mathtt{e} : \mathtt{skip} \longrightarrow \mathtt{c};$

$\mathtt{f} : \mathtt{stop};$

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

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

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

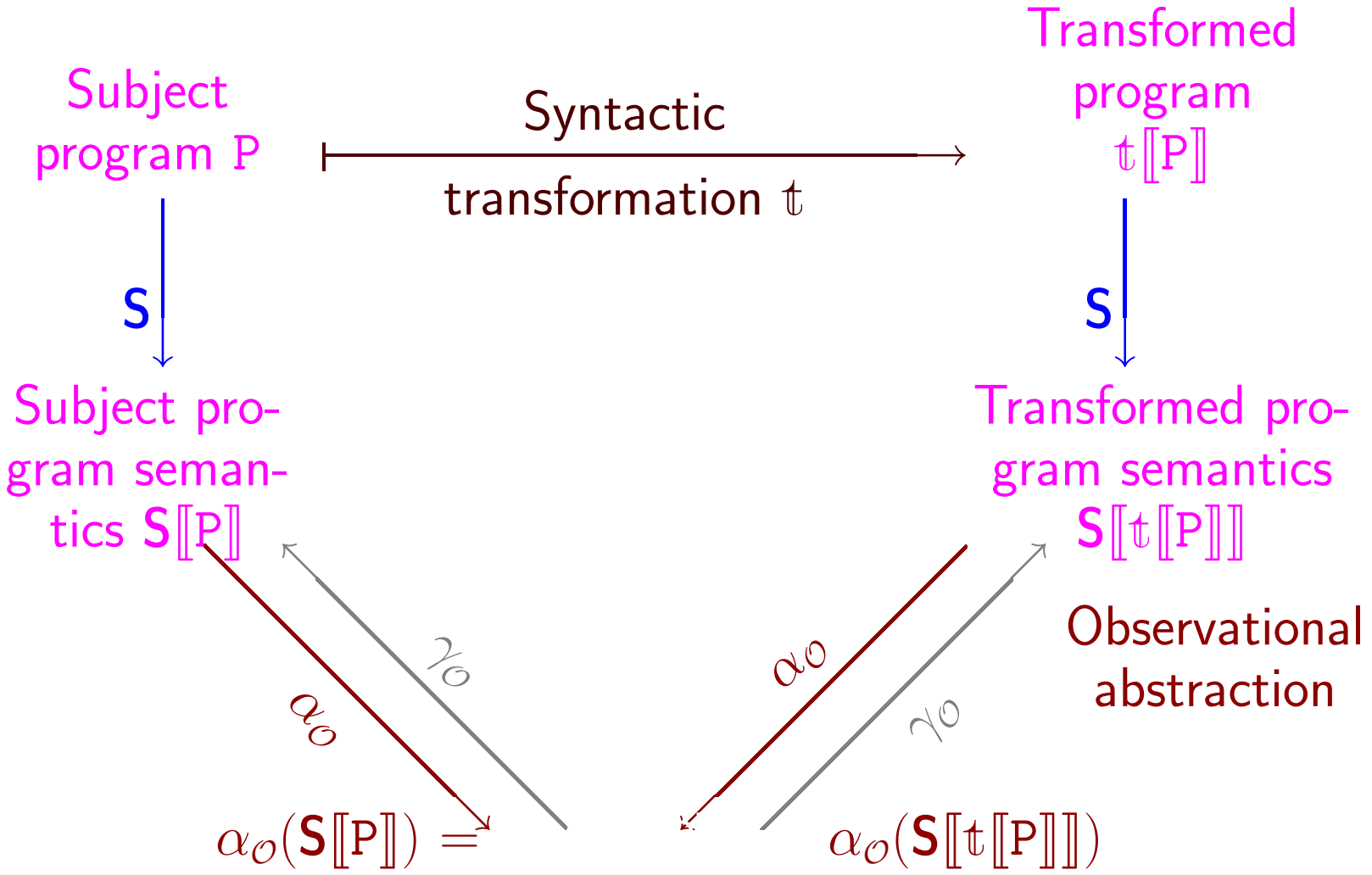
$\langle \mathtt{d} : \mathtt{X} := \mathtt{X} - \overset{1}{\not\mathtt{Y}} \longrightarrow \mathtt{e};, [\mathtt{X} : 1, \mathtt{Y} : 1] \rangle$

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

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

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

Principle of Online Program Transformation (3)



Example: Constant Propagation

Subject/Transformed Semantics

$\langle \mathtt{a} : \mathtt{X} := ? \longrightarrow \mathtt{b};,\ [\mathtt{X} : \mho, \mathtt{Y} : \mho] \rangle$
$\langle \mathtt{b} : \mathtt{Y} := 1 \longrightarrow \mathtt{c};,\ [\mathtt{X} : 1, \mathtt{Y} : \mho] \rangle$
$\langle \mathtt{c} : (\mathtt{X} > 0) \longrightarrow \mathtt{d};,\ [\mathtt{X} : 1, \mathtt{Y} : 1] \rangle$
$\langle \mathtt{d} : \mathtt{X} := \mathtt{X} - \overset{1}{\cancel{\mathtt{Y}}} \longrightarrow \mathtt{e};,\ [\mathtt{X} : 1, \mathtt{Y} : 1] \rangle$
$\langle \mathtt{e} : \mathtt{skip} \longrightarrow \mathtt{c};,\ [\mathtt{X} : 0, \mathtt{Y} : 1] \rangle$
$\langle \mathtt{c} : (\mathtt{X} \leq 0) \longrightarrow \mathtt{f};,\ [\mathtt{X} : 0, \mathtt{Y} : 1] \rangle$
$\langle \mathtt{f} : \mathtt{stop};,\ [\mathtt{X} : 0, \mathtt{Y} : 1] \rangle$

Observational Semantics

$\langle \mathtt{a} : \ \longrightarrow \mathtt{b};,\ [\mathtt{X} : \mho, \mathtt{Y} : \mho] \rangle$
$\langle \mathtt{b} : \ \longrightarrow \mathtt{c};,\ [\mathtt{X} : 1, \mathtt{Y} : \mho] \rangle$
$\langle \mathtt{c} : \ \longrightarrow \mathtt{d};,\ [\mathtt{X} : 1, \mathtt{Y} : 1] \rangle$
$\langle \mathtt{d} : \ \longrightarrow \mathtt{e};,\ [\mathtt{X} : 1, \mathtt{Y} : 1] \rangle$
$\langle \mathtt{e} : \ \longrightarrow \mathtt{c};,\ [\mathtt{X} : 0, \mathtt{Y} : 1] \rangle$
$\langle \mathtt{c} : \ \longrightarrow \mathtt{f};,\ [\mathtt{X} : 0, \mathtt{Y} : 1] \rangle$
$\langle \mathtt{f} : \mathtt{stop};,\ [\mathtt{X} : 0, \mathtt{Y} : 1] \rangle$

Principle of Online Program Transformation (4)

- The syntactic transformation induces a semantic transformation:

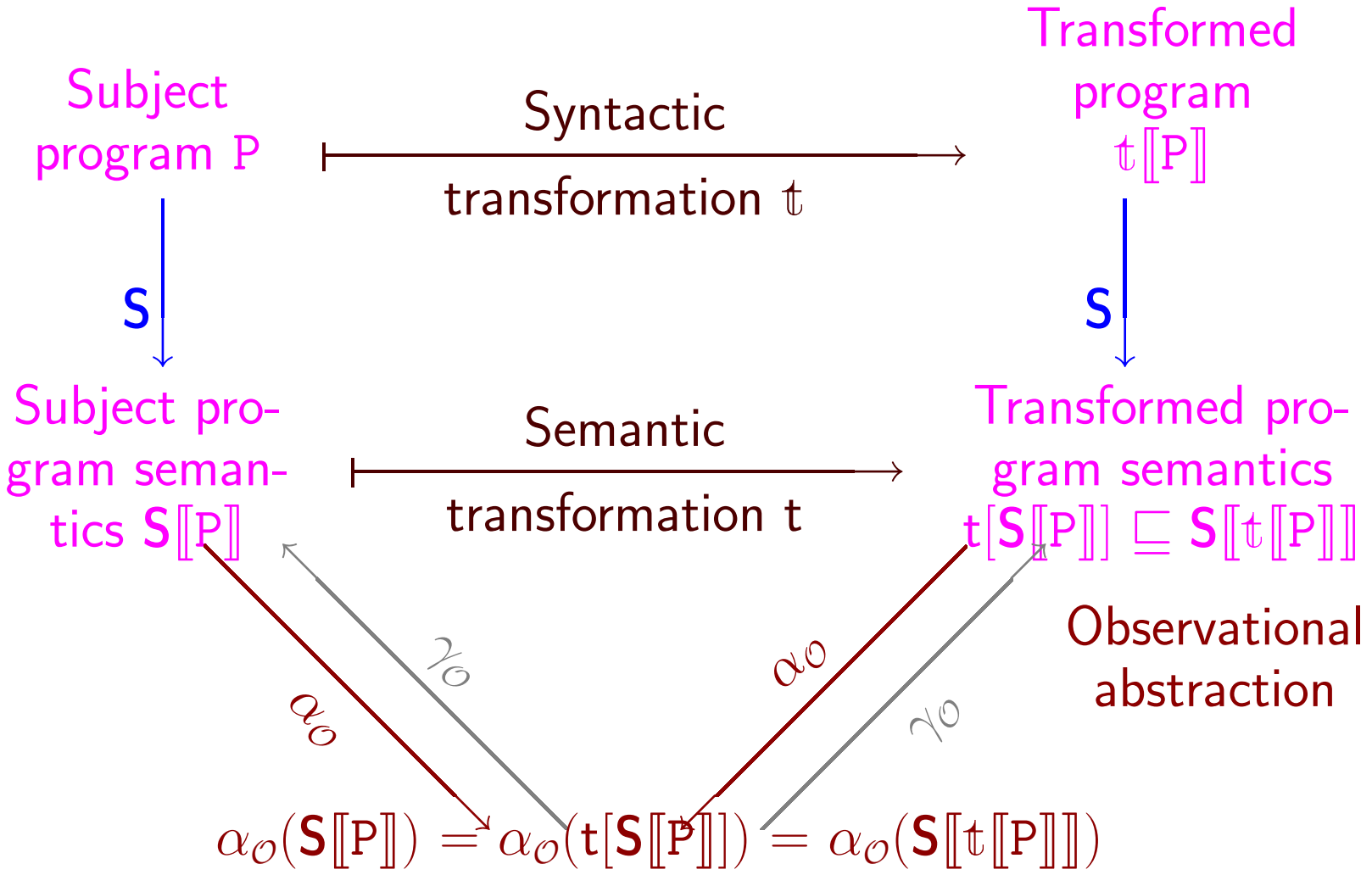
  The semantics of the subject program is mapped to the semantics of the transformed program;

- 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

$$
\begin{array}{ll}
\mathtt{a} : \mathtt{X} := ? \longrightarrow \mathtt{b}; & \langle \mathtt{a} : \mathtt{X} := ? \longrightarrow \mathtt{b};,\ [\mathtt{X} : \mho, \mathtt{Y} : \mho] \rangle \\
\mathtt{b} : \mathtt{Y} := 1 \longrightarrow \mathtt{c}; & \langle \mathtt{b} : \mathtt{Y} := 1 \longrightarrow \mathtt{c};,\ [\mathtt{X} : 1, \mathtt{Y} : \mho] \rangle \\
\mathtt{c} : (\mathtt{X} \leq 0) \longrightarrow \mathtt{f}; & \\
\mathtt{c} : (\mathtt{X} > 0) \longrightarrow \mathtt{d}; & \langle \mathtt{c} : (\mathtt{X} > 0) \longrightarrow \mathtt{d};,\ [\mathtt{X} : 1, \mathtt{Y} : 1] \rangle \\
 & \overset{1}{} \\
\mathtt{d} : \mathtt{X} := \mathtt{X} - \mathtt{Y} \longrightarrow \mathtt{e}; & \langle \mathtt{d} : \mathtt{X} := \mathtt{X} - \cancel{\mathtt{Y}} \longrightarrow \mathtt{e};,\ [\mathtt{X} : 1, \mathtt{Y} : 1] \rangle \\
\mathtt{e} : \mathtt{skip} \longrightarrow \mathtt{c}; & \langle \mathtt{e} : \mathtt{skip} \longrightarrow \mathtt{c};,\ [\mathtt{X} : 0, \mathtt{Y} : 1] \rangle \\
 & \langle \mathtt{c} : (\mathtt{X} \leq 0) \longrightarrow \mathtt{f};,\ [\mathtt{X} : 0, \mathtt{Y} : 1] \rangle \\
\mathtt{f} : \mathtt{stop}; & \langle \mathtt{f} : \mathtt{stop};,\ [\mathtt{X} : 0, \mathtt{Y} : 1] \rangle
\end{array}
$$

Principle of Online Program Transformation (4)



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);

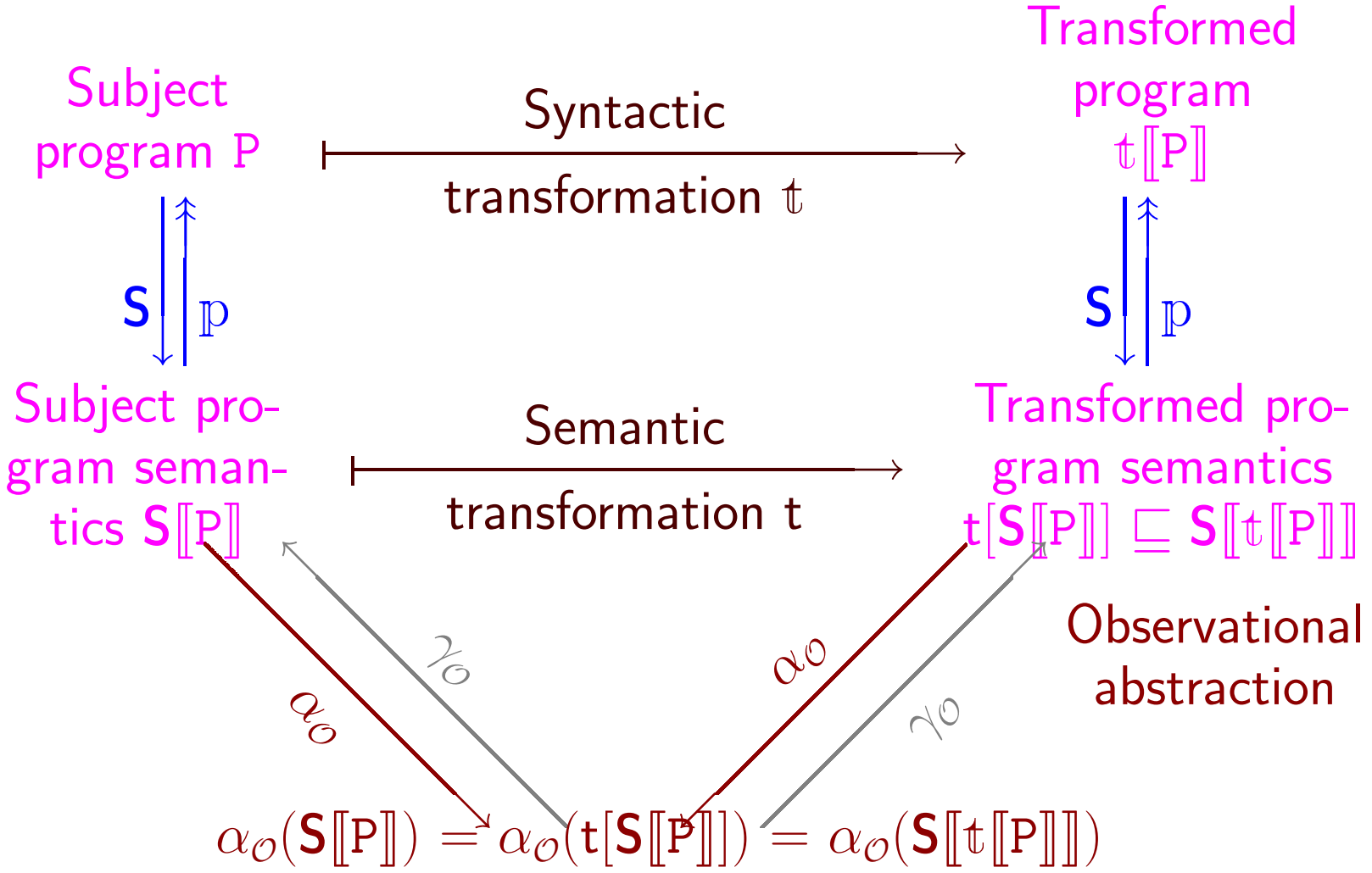
Correspondence Between Syntax and Semantics, Cont'd

- The correspondence between syntax and semantics is an abstraction:

$$\mathrm{po}\langle \mathfrak{D};\ \sqsubseteq \rangle \underset{\mathbb{p}}{\overset{\mathsf{S}}{\leftarrow\!\!\!\!\twoheadrightarrow}} \mathrm{po}\langle \mathbb{P}/\!\!\equiv;\ \mathbb{\sqsubseteq} \rangle$$

- 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)



Example: Syntax to Prefix Trace Semantics

- Fixpoint semantics:

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

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

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;

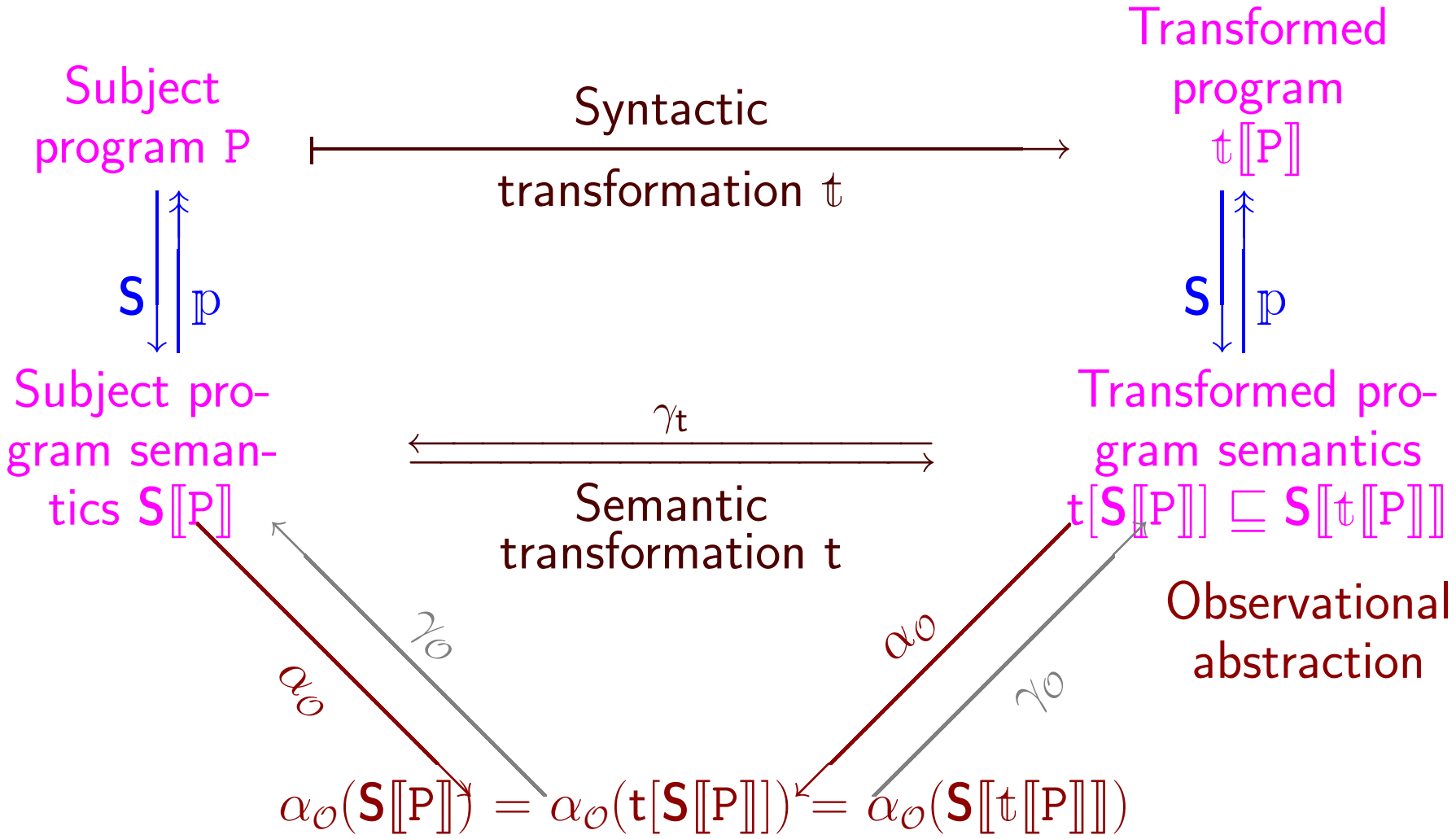
  $\longrightarrow$ The semantic program transformation is an abstraction;

Intuition for Transformations as Abstractions

$$
\begin{array}{l}
\mathtt{a}: \mathtt{X} := ? \longrightarrow \mathtt{b};\\
\mathtt{b}: \mathtt{Y} := 1 \longrightarrow \mathtt{c};\\
\mathtt{c}: (\mathtt{X} \leq 0) \longrightarrow \mathtt{f};\\
\mathtt{c}: (\mathtt{X} > 0) \longrightarrow \mathtt{d};\\
\\
\mathtt{d}: \mathtt{X} := \begin{array}{l}\ldots\\ \mathtt{X} - (2 * \mathtt{Y} - 1)\\ \mathtt{X} - \mathtt{Y}\end{array} \longrightarrow \mathtt{e};\\
\mathtt{e}: \mathtt{skip} \longrightarrow \mathtt{c};\\
\mathtt{f}: \mathtt{stop};
\end{array}
$$

$$
\begin{array}{l}
\mathtt{a}: \mathtt{X} := ? \longrightarrow \mathtt{b};\\
\mathtt{b}: \mathtt{Y} := 1 \longrightarrow \mathtt{c};\\
\mathtt{c}: (\mathtt{X} \leq 0) \longrightarrow \mathtt{f};\\
\mathtt{c}: (\mathtt{X} > 0) \longrightarrow \mathtt{d};\\
\\
\mathtt{d}: \mathtt{X} := \mathtt{X} - 1 \longrightarrow \mathtt{e};\\
\mathtt{e}: \mathtt{skip} \longrightarrow \mathtt{c};\\
\mathtt{f}: \mathtt{stop};
\end{array}
$$

Principle of Online Program Transformation (6)



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

$$\mathtt{a} : \mathtt{X} := ? \longrightarrow \mathtt{b}; \qquad \langle \mathtt{a} : \mathtt{X} := ? \longrightarrow \mathtt{b};, [\mathtt{X} : \mathcal{U}, \mathtt{Y} : \mathcal{U}] \rangle$$

$$\mathtt{b} : \mathtt{Y} := 1 \longrightarrow \mathtt{c}; \qquad \langle \mathtt{b} : \mathtt{Y} := 1 \longrightarrow \mathtt{c};, [\mathtt{X} : 1, \mathtt{Y} : \mathcal{U}] \rangle$$

$$\mathtt{c} : (\mathtt{X} \leq 0) \longrightarrow \mathtt{f};$$

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

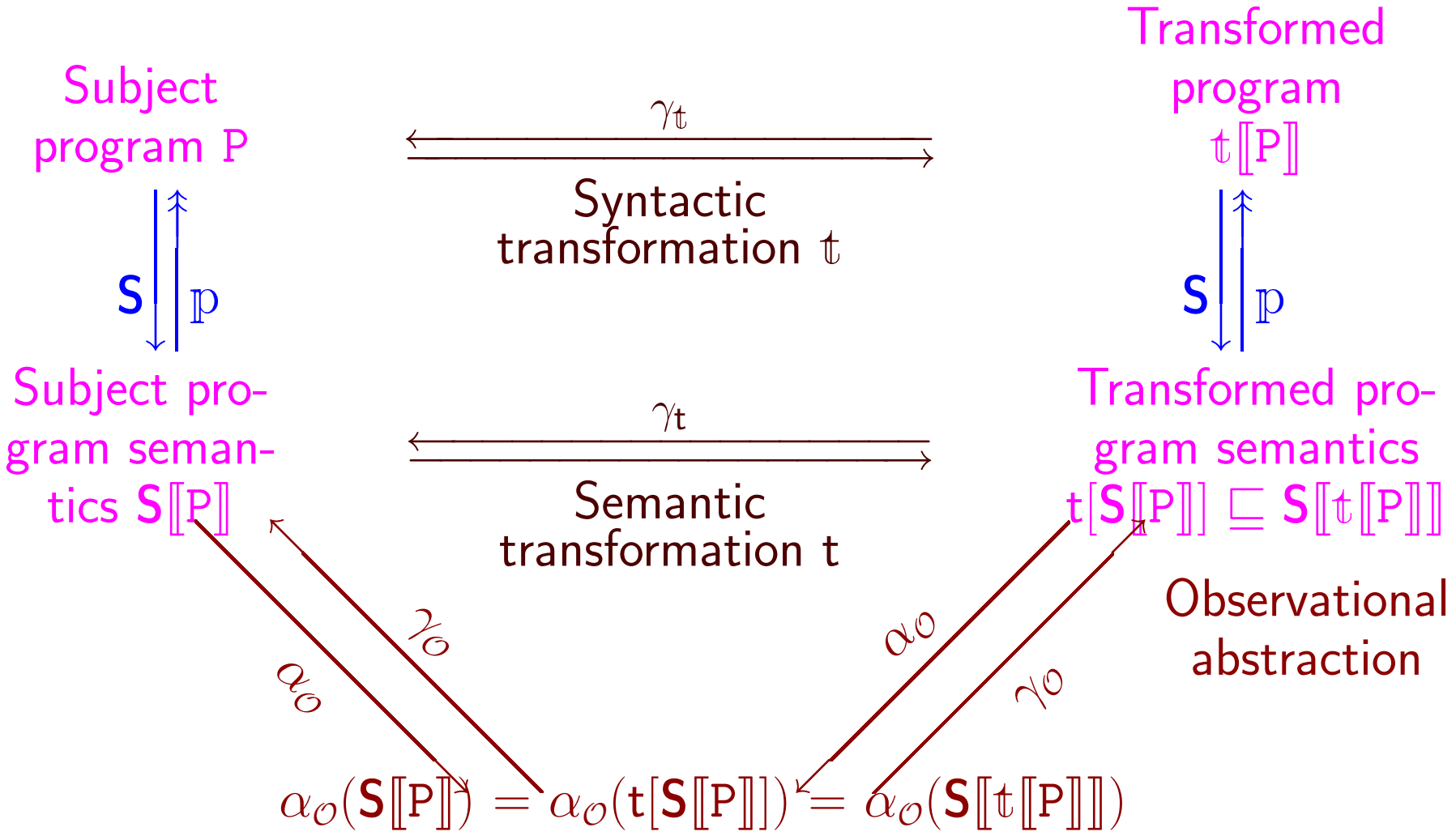
$$\mathtt{d} : \mathtt{X} := \mathtt{X} - \overset{1}{\cancel{\mathtt{Y}}} \longrightarrow \mathtt{e}; \qquad \langle \mathtt{d} : \mathtt{X} := \mathtt{X} - \overset{1}{\cancel{\mathtt{Y}}} \longrightarrow \mathtt{e};, [\mathtt{X} : 1, \mathtt{Y} : 1] \rangle$$

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

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

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

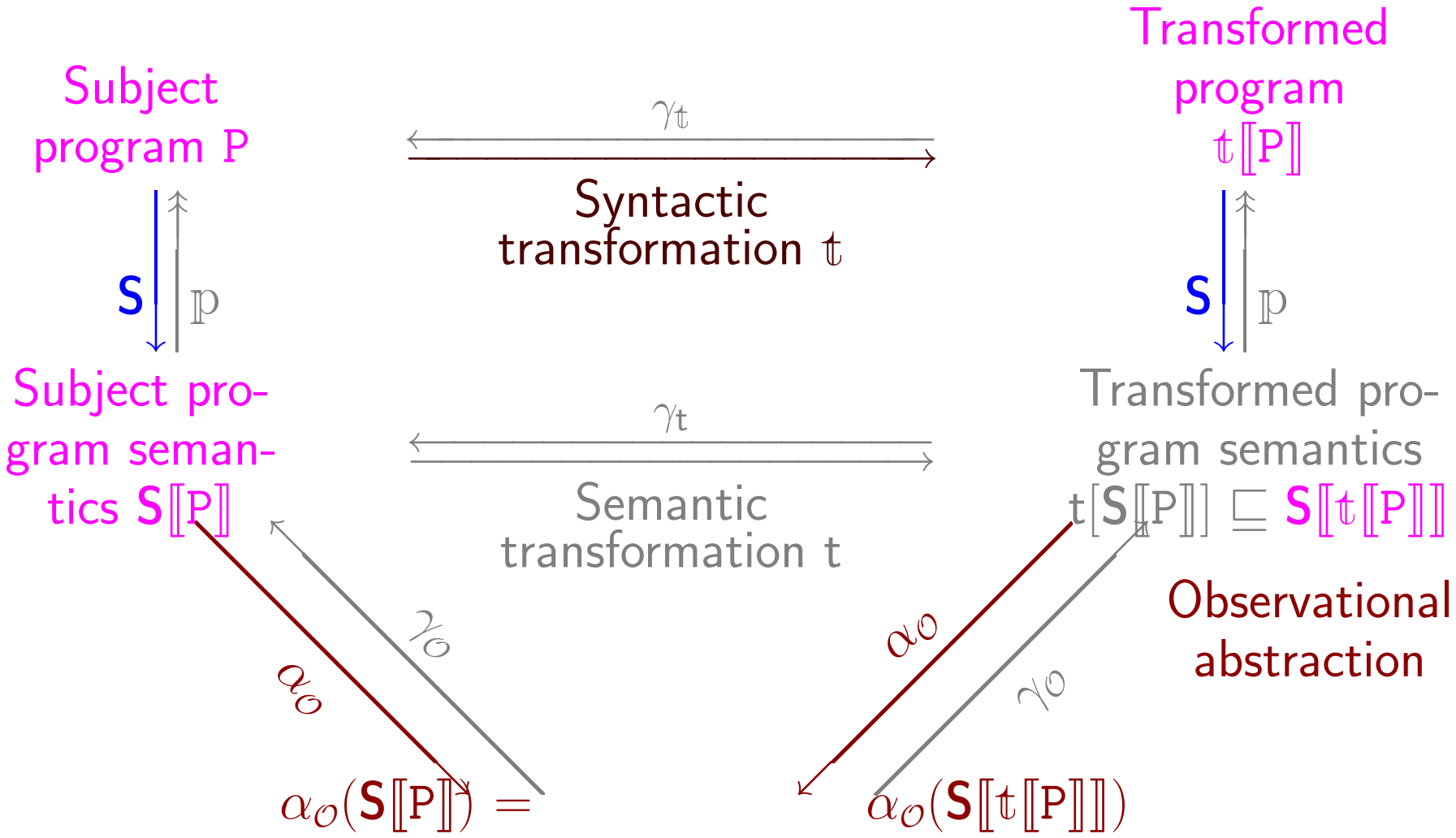
Principle of Online Program Transformation (Done)



Principle of the Formalization of Program Transformation by Abstract Interpretation

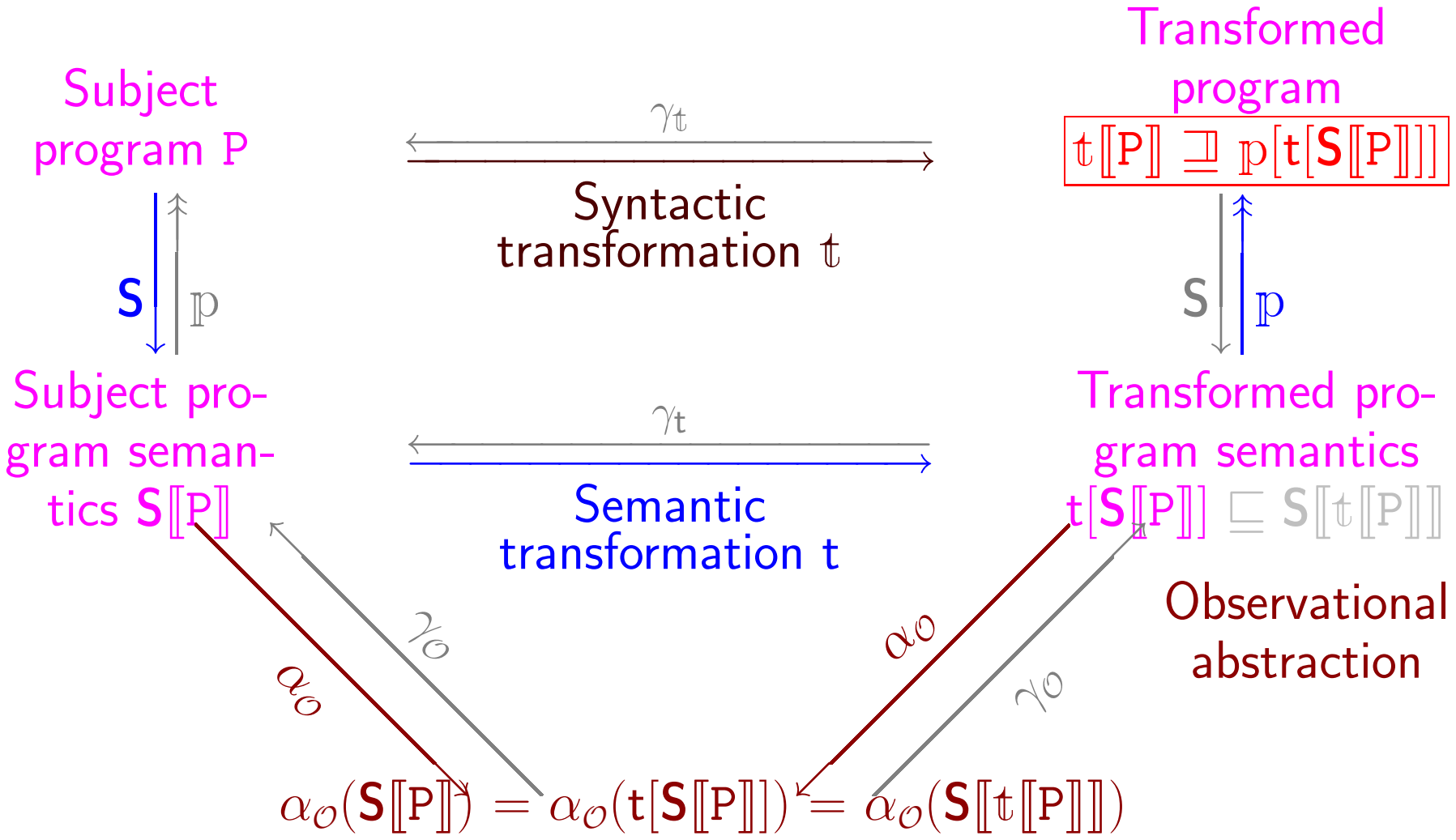
Formalization of Program Transformation Correctness by Abstract Interpretation

Correctness of an Online Program Transformation



Design of Program Transformations by Abstract Interpretation

Design of an Online Program Transformation



Design of Program Transformation Algorithms

$$
\begin{aligned}
\mathtt{t}[\![\mathrm{P}]\!] &\sqsupseteq \mathtt{p}[\mathtt{t}[\mathbf{S}[\![\mathrm{P}]\!]]] \\
&= \mathtt{p}[\mathtt{t}[\mathbf{lfp}^{\sqsubseteq}\, \mathbf{F}[\![\mathrm{P}]\!]]] \\
&\sqsupseteq \ldots \qquad \leftarrow \text{ apply fixpoint transfer} \\
&\qquad\qquad\qquad\quad \text{/approximation theorems} \\
&= \mathbf{lfp}^{\sqsubseteq^\sharp}\, \mathbb{F}^\sharp[\![\mathrm{P}]\!]
\end{aligned}
$$

We obtain an iterative program transformation algorithm!

The Iterative Constant Propagation Algorithm

ConstantPropagation($\mathtt{P}$, $\rho^\sharp$) =

  $\mathtt{Q} := \emptyset$;

  forall label $\mathtt{L}$ of $\mathtt{P}$ such that $\rho^\sharp(\mathtt{L}) \neq \dot{\bot}$ **do**

    forall $\mathtt{L} : \mathtt{A} \longrightarrow \mathtt{L}_1 ; \in \mathtt{P}$ **do**

      $\mathtt{A}_c := \textbf{Simplify}[\![\mathtt{A}]\!](\rho^\sharp(\mathtt{L}))$;

      $\mathtt{Q} := \mathtt{Q} \cup \{\mathtt{L} : \mathtt{A}_c \longrightarrow \mathtt{L}_1 ;\}$

    end;

    if $\mathtt{L} : \mathtt{stop}; \in \mathtt{P}$ **then**

      $\mathtt{Q} := \mathtt{Q} \cup \{\mathtt{L} : \mathtt{stop};\}$

    end

  end;

  return $\mathtt{Q}$.

Principle of Offline Program Transformation

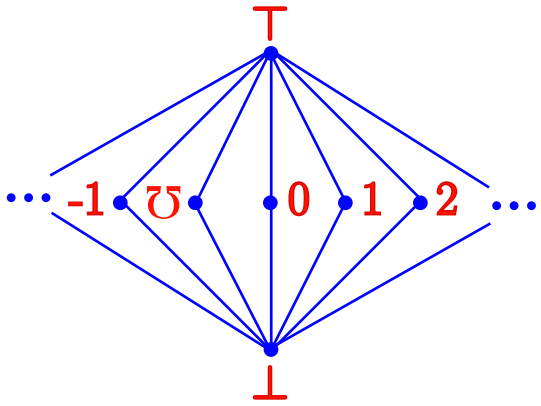
Offline Transformations

- 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):



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

- Pointwise extension to variable environments and program labels;

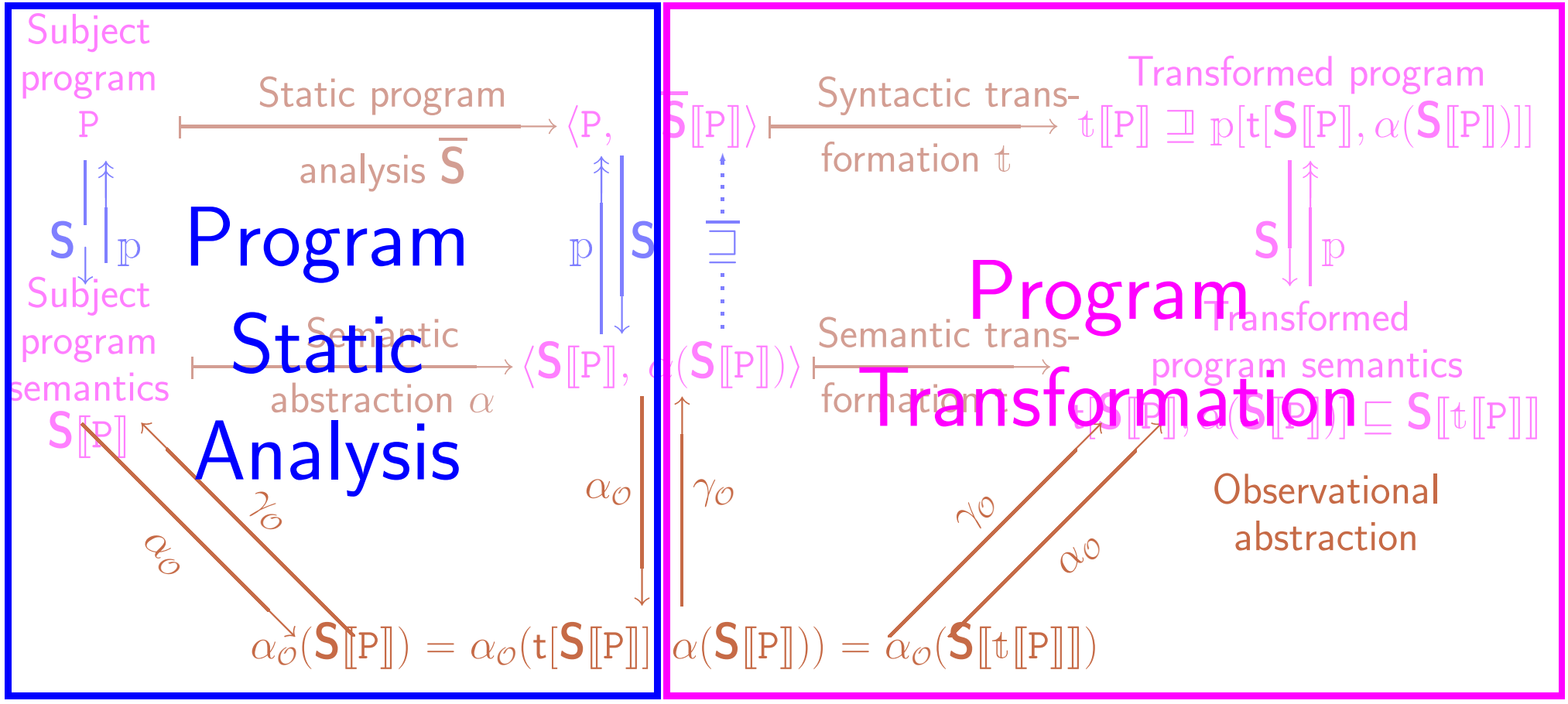
Example: Kildall's Constant Propagation, Cont'd

- Elementwise abstraction of a set $\mathcal{T}$ of traces:

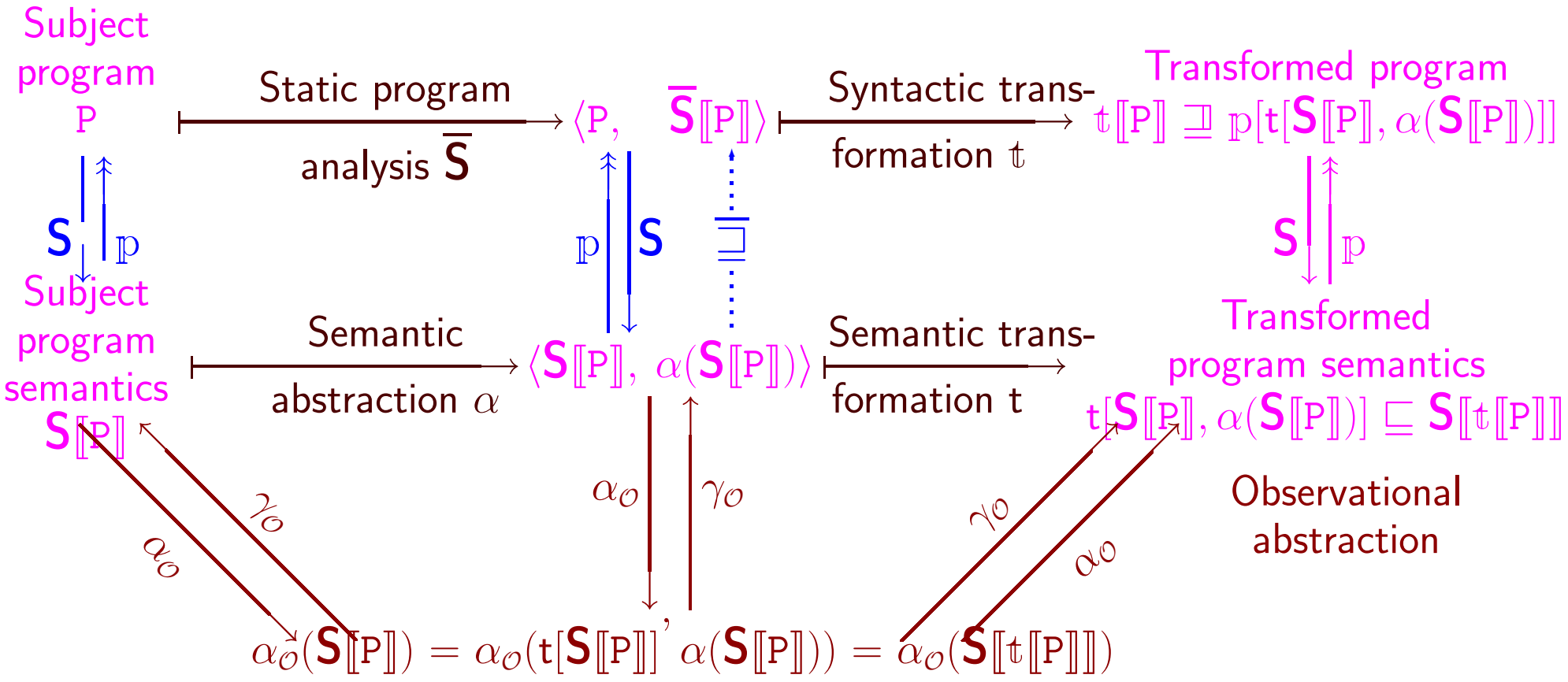
$$\alpha^c(\mathcal{T}) = \lambda \mathtt{L} \cdot \lambda \mathtt{X} \cdot \ddot{\bigsqcup} \{\rho(\mathtt{X}) \mid \exists \sigma \in \mathcal{T} : \exists \mathtt{C} \in \mathbb{C} : \exists i : \sigma_i = \langle \rho,\ \mathtt{C} \rangle \wedge \mathsf{lab}[\![\mathtt{C}]\!] = \mathtt{L}\}$$

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

Principle of Offline Program Transformation



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.