Design of Syntactic Program Transformations by Abstract Interpretation of Semantic Transformations

Patrick COUSOT
École Normale Supérieure
45 rue d'Ulm, 75230 Paris cedex 05, France
Patrick.Cousot@ens.fr   www.di.ens.fr/ cousot

ICLP'01, ΠΑΦΟΣ, ΚΥΠΡΟΣ   Nov 26 – Dec 1, 2001

Content

1. A short introduction to abstract interpretation (in the context of program static analysis) 2

2. A new application of abstract interpretation: program transformation 45

3. Conclusion 87

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

A Short Introduction to 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

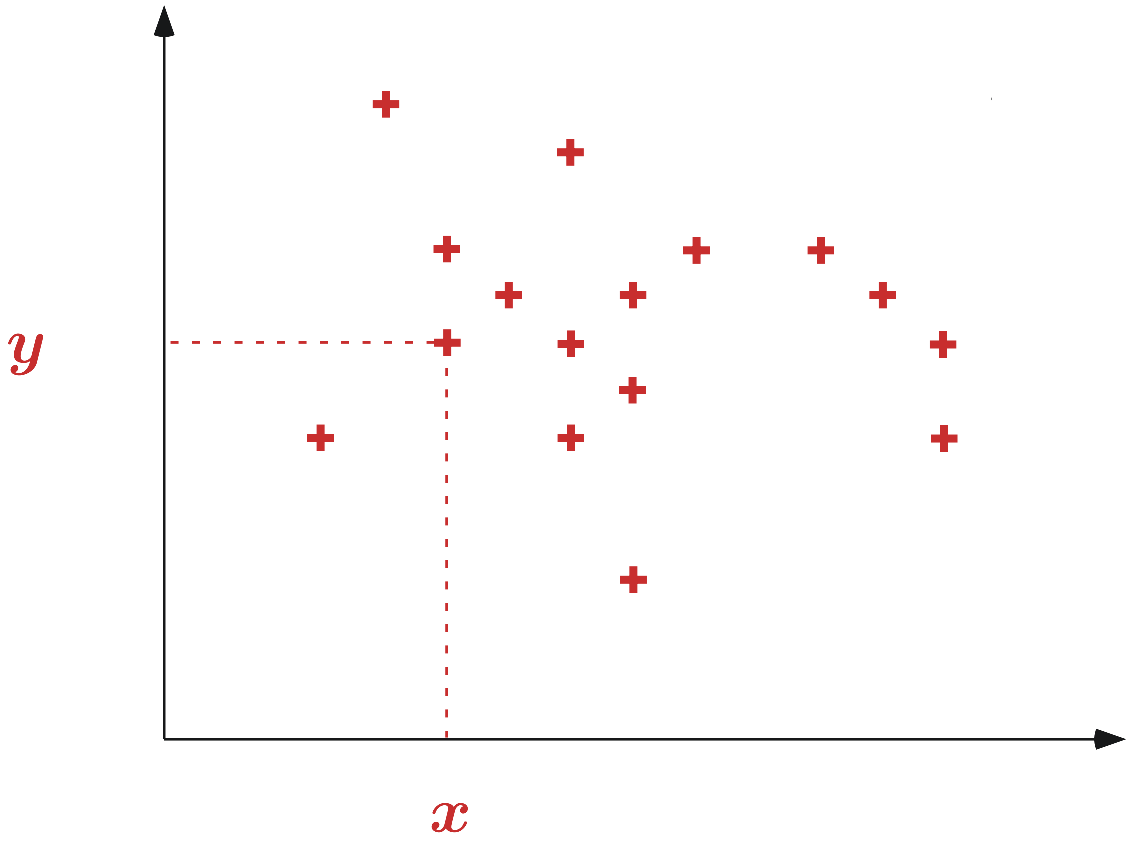
Abstraction: intuition

- 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

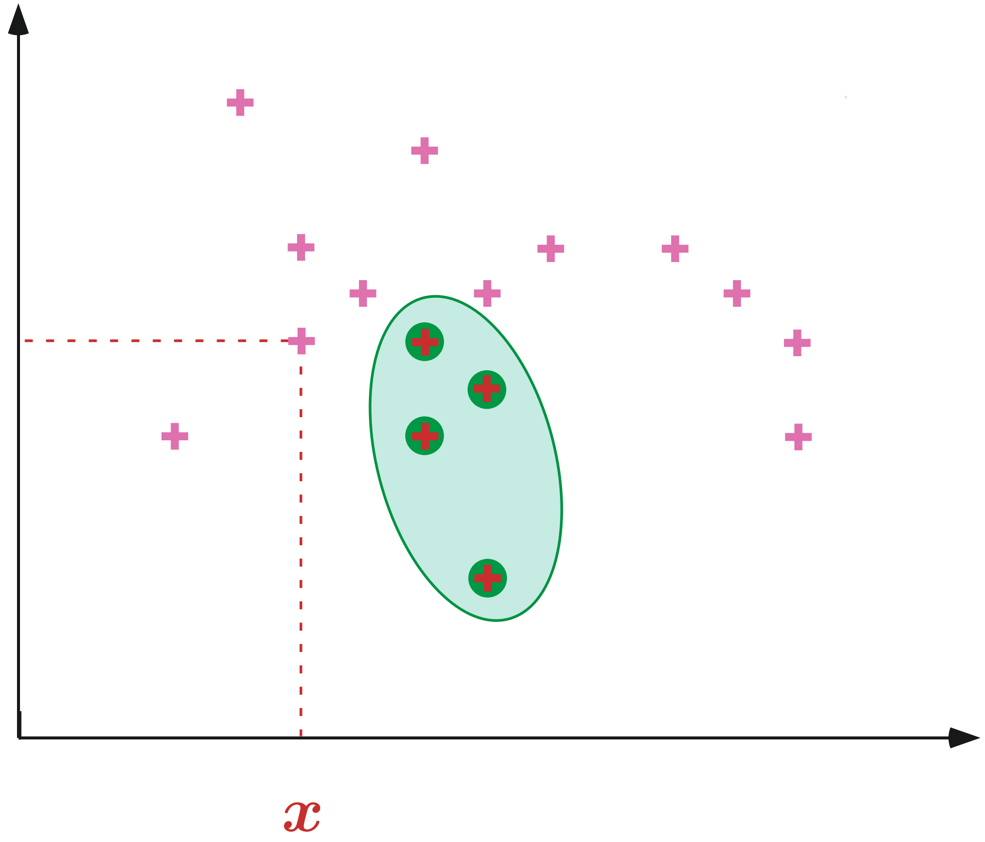
Approximations of an [in]finite set of points;



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

Approximations of an [in]finite set of points:

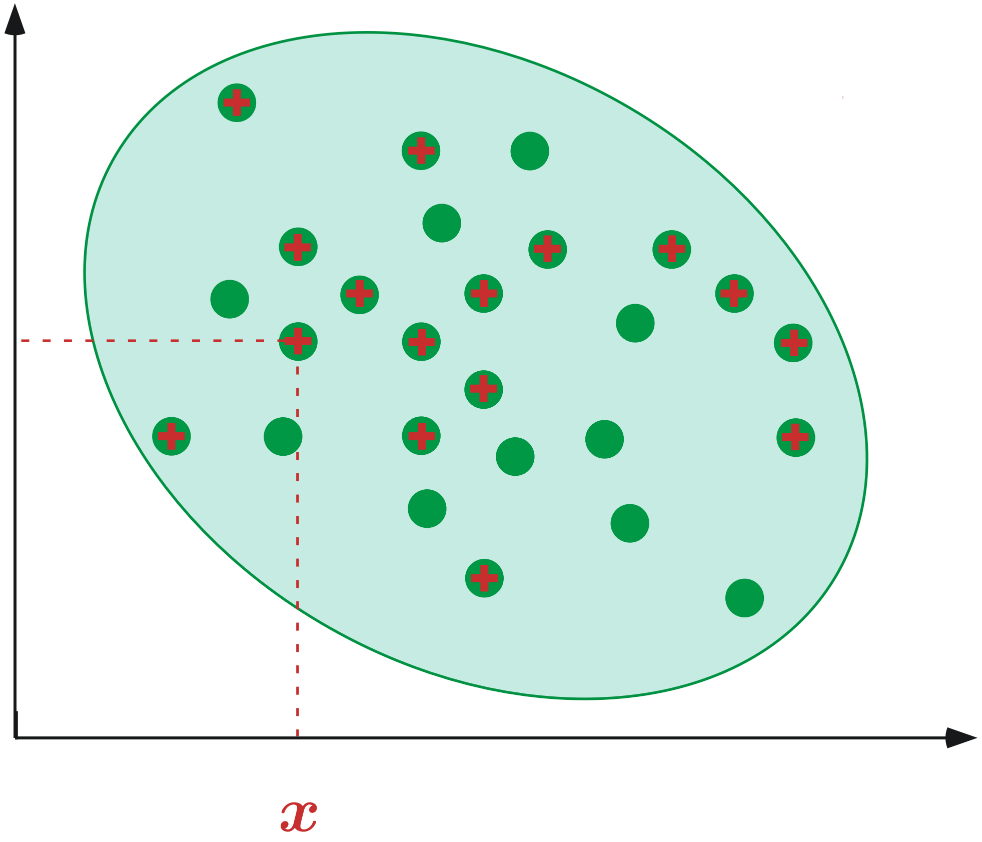
From Below



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

Approximations of an [in]finite set of points:

From Above

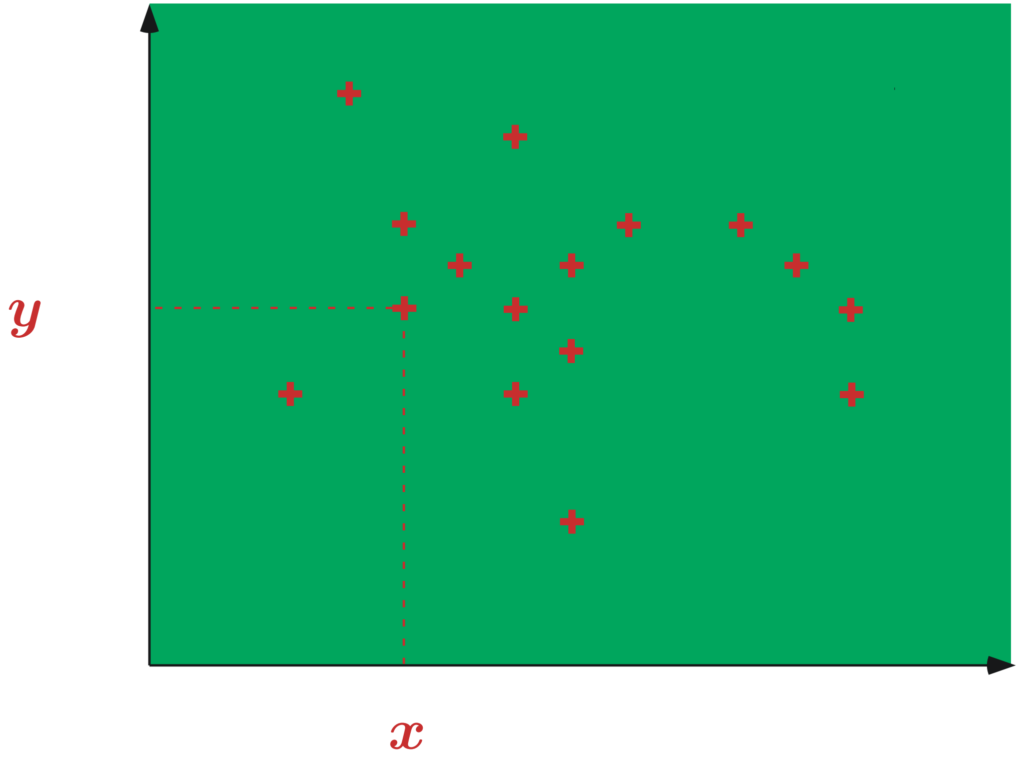


$$\{\ldots, \langle 19,\ 78\rangle, \ldots,$$

$$\langle 20,\ 01\rangle, \langle ?,\ ?\rangle, \ldots\}$$

Intuition Behind Effective Computable Abstraction

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

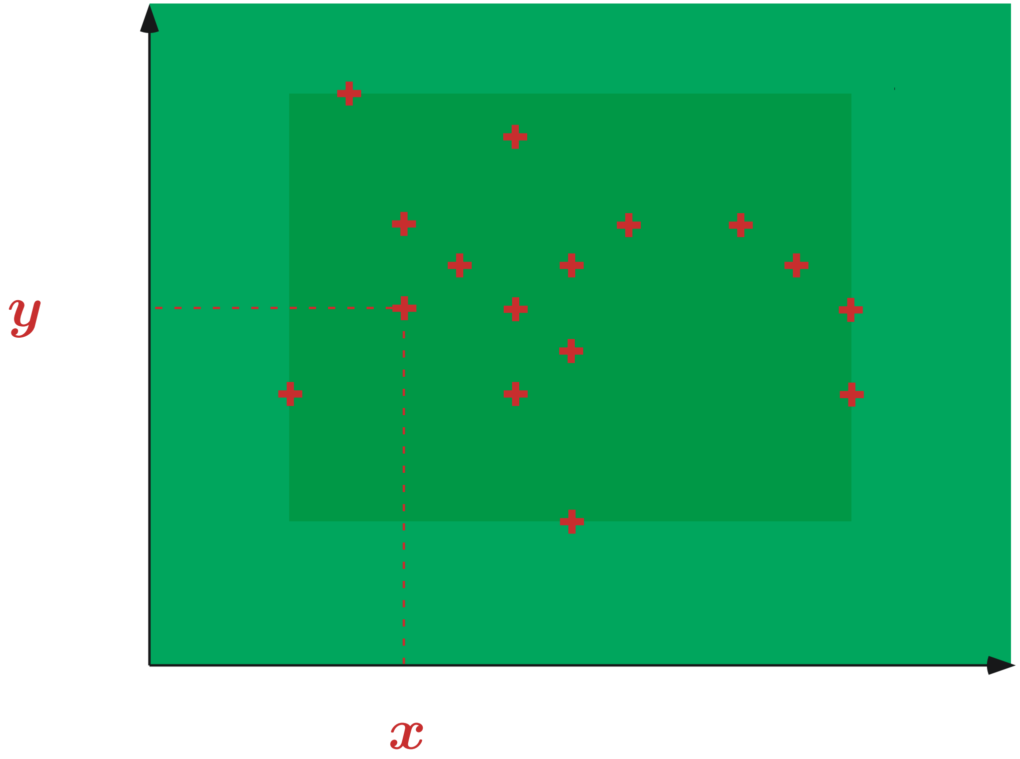


$$\begin{cases} x \geq 0 \\ y \geq 0 \end{cases}$$

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]

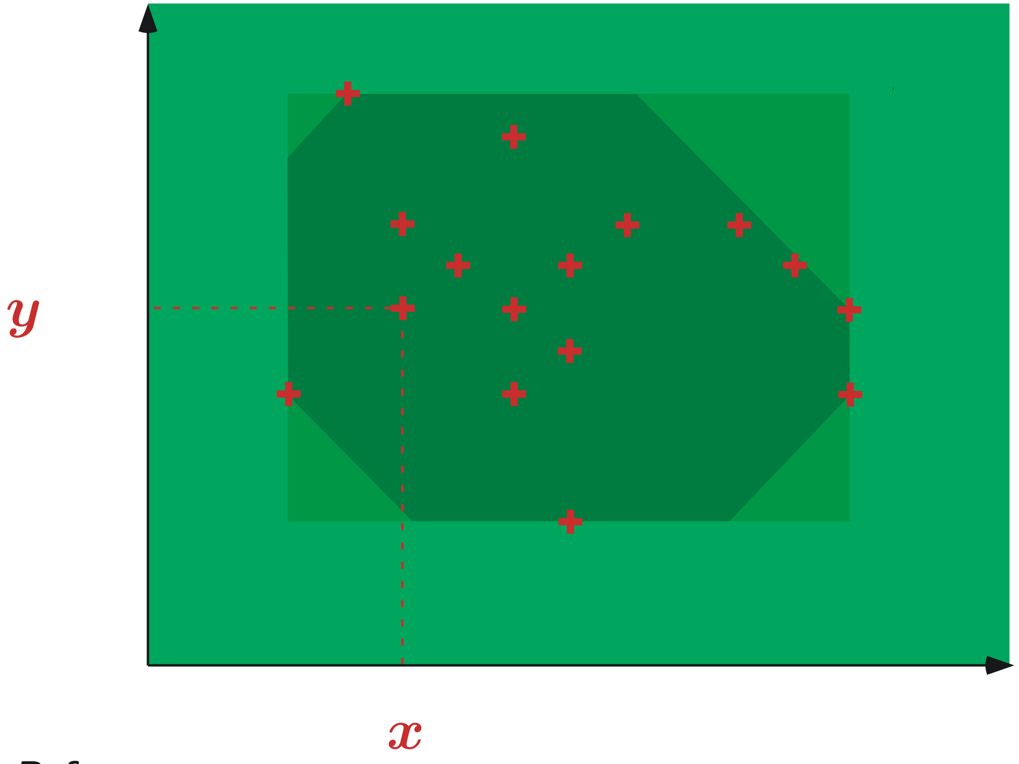


$$\begin{cases} x \in [19,\ 78] \\ y \in [20,\ 01] \end{cases}$$

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]

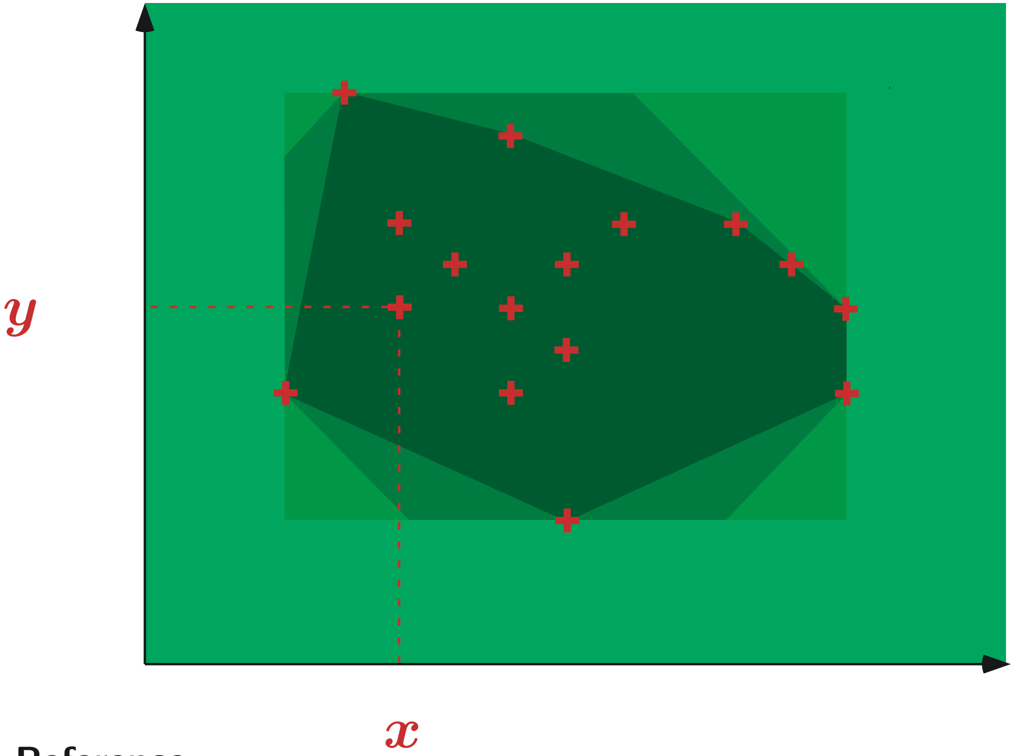


$$\begin{cases} 1 \leq x \leq 9 \\ x + y \leq 78 \\ 1 \leq y \leq 9 \\ x - y \leq 99 \end{cases}$$

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]

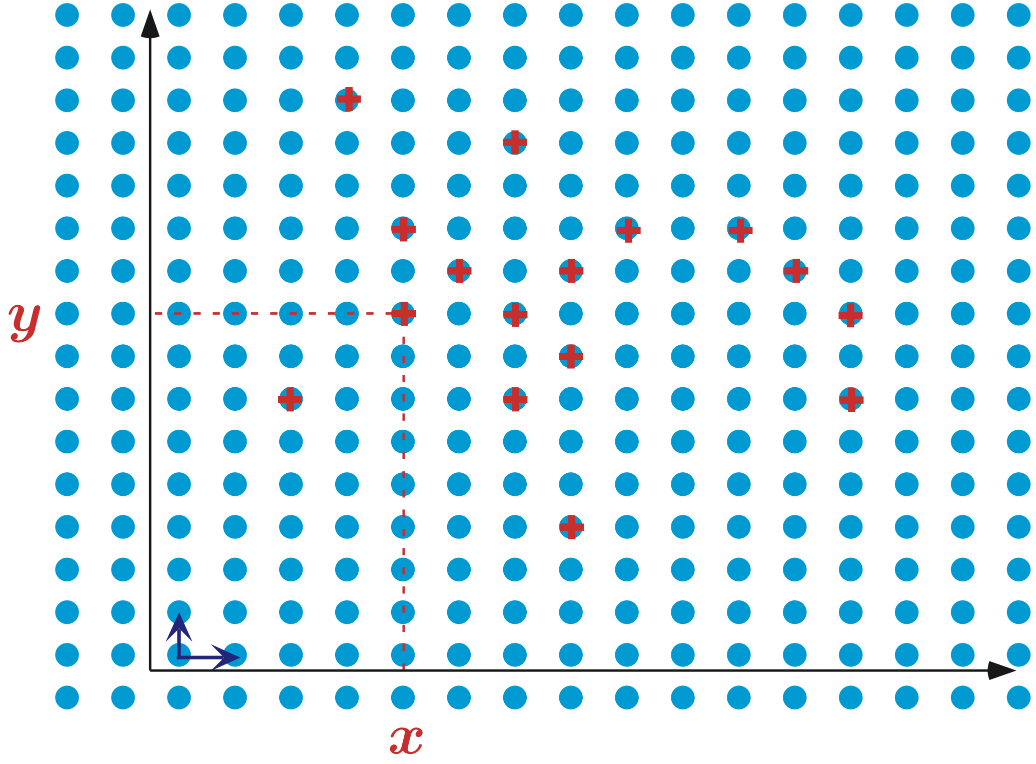


$$\begin{cases} 19x + 78y \leq 2000 \\ 20x + 01y \geq 0 \end{cases}$$

Reference

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

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

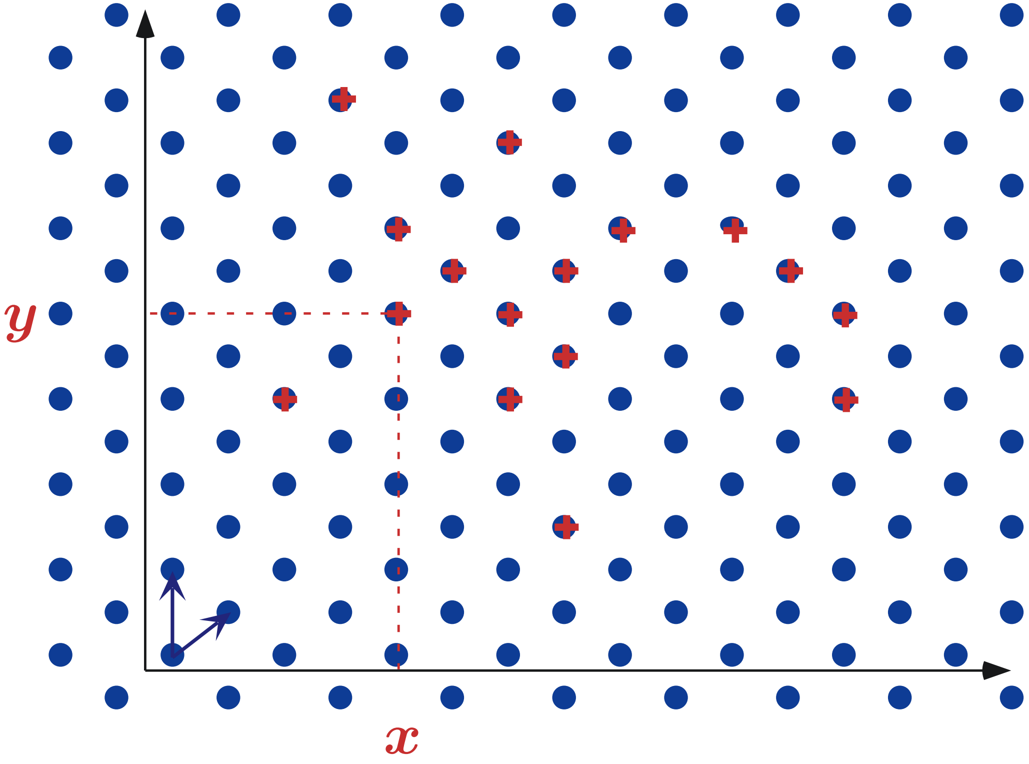


$$\begin{cases} x = 19 \bmod 78 \\ y = 20 \bmod 99 \end{cases}$$

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]

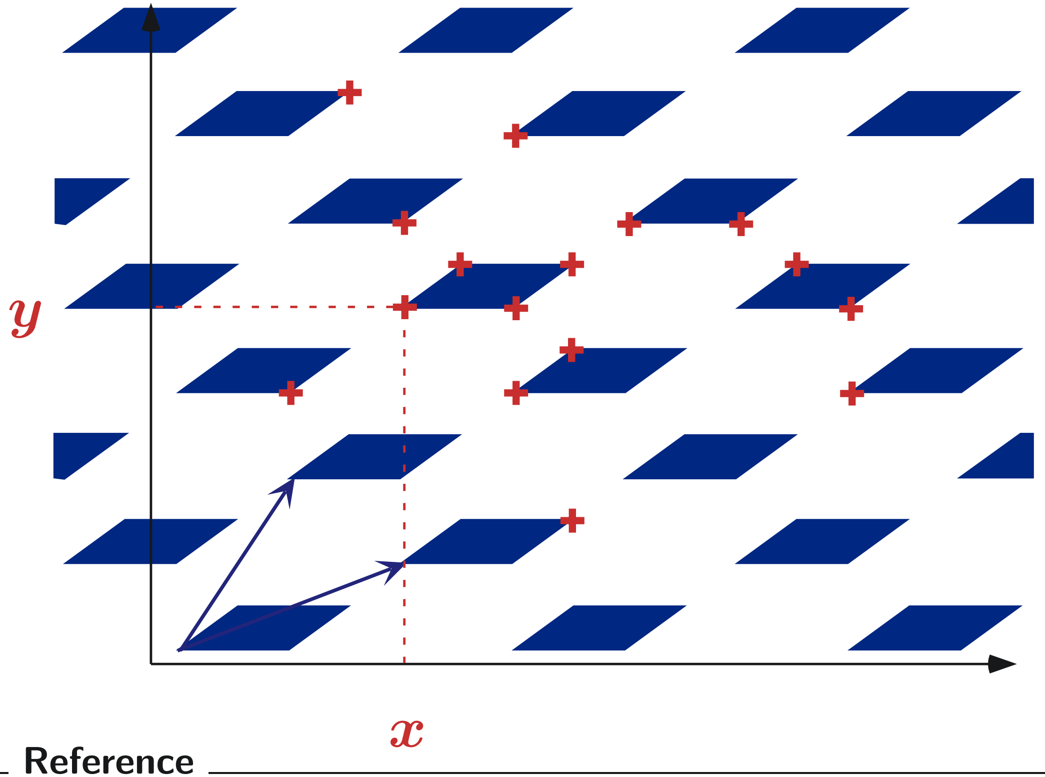


$$\begin{cases} 1x + 9y = 7 \bmod 8 \\ 2x - 1y = 9 \bmod 9 \end{cases}$$

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

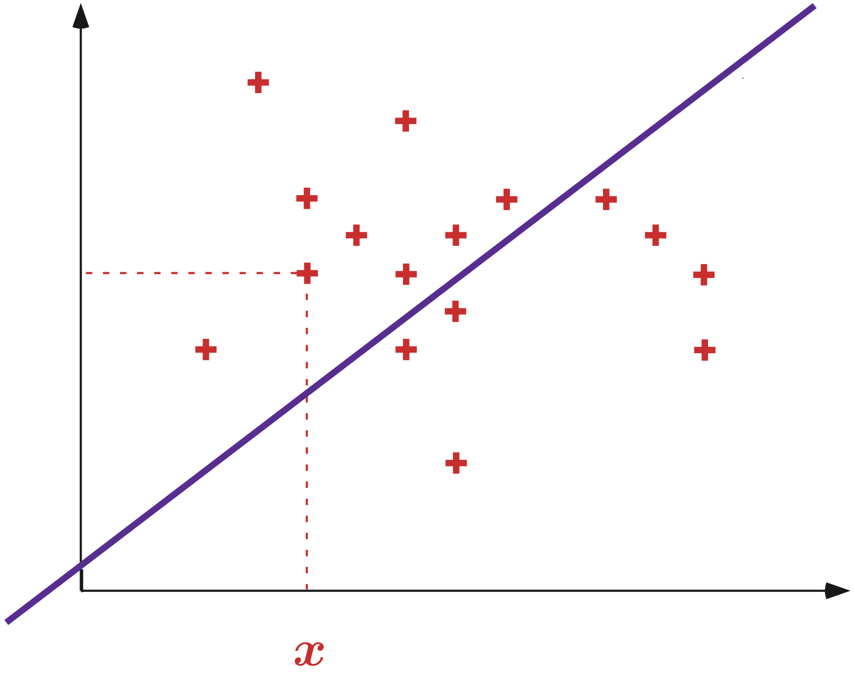
[7] F. Masdupuy. Array operations abstraction using semantic analysis of trapezoid congruences. In *ACM Int. Conf. on Supercomputing, ICS '92*, pages 226–235, 1992.

Conservative Approximation and Information Loss

Intuition Behind Sound/Conservative Approximation

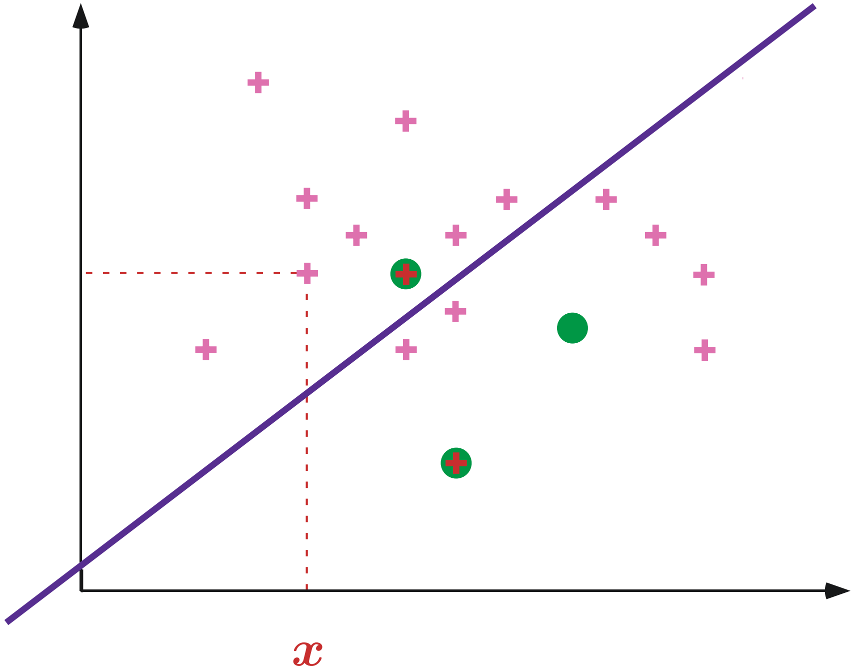
Conservative Approximation

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



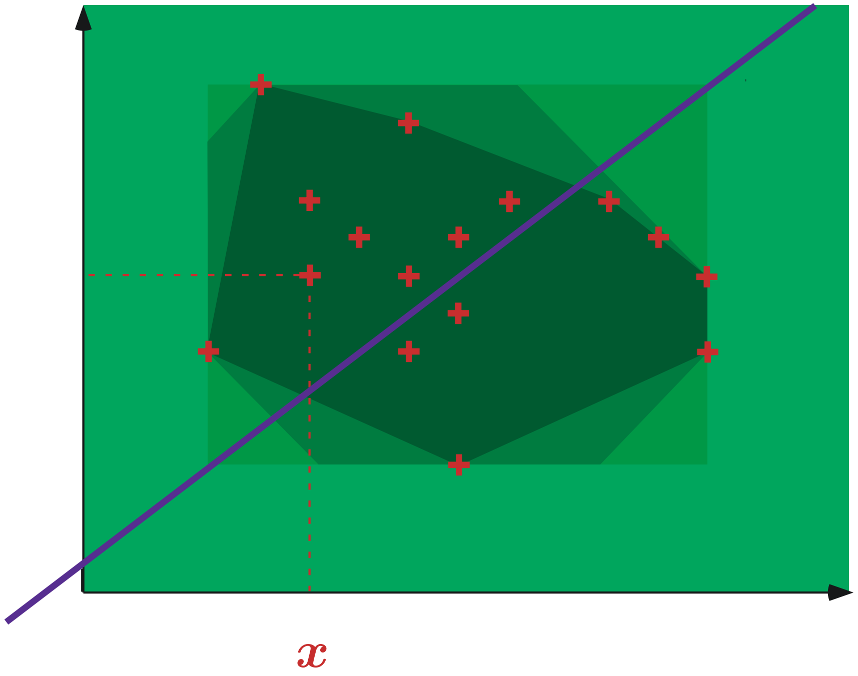
Conservative Approximation

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



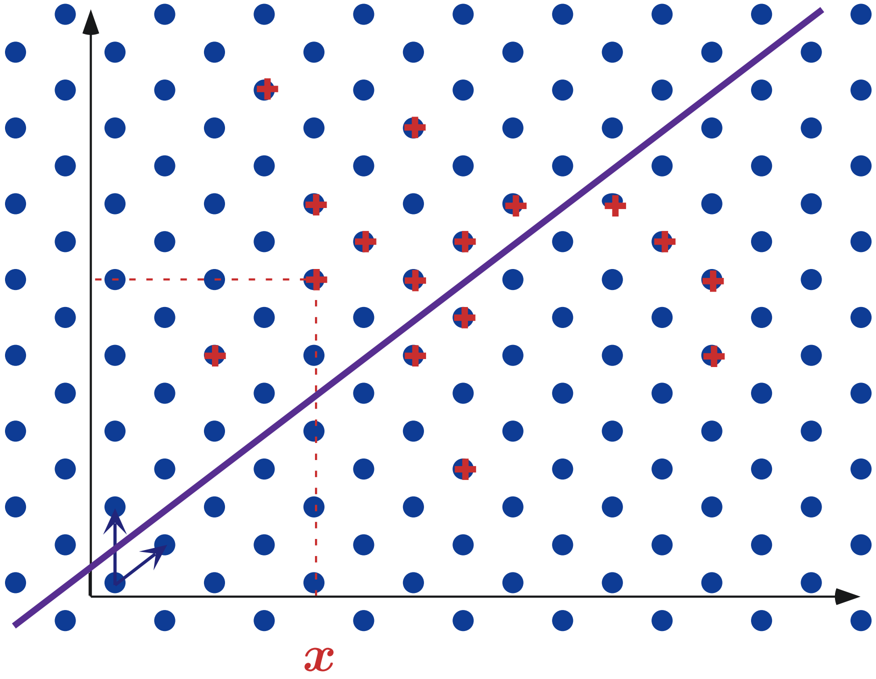
Conservative Approximation

- 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**



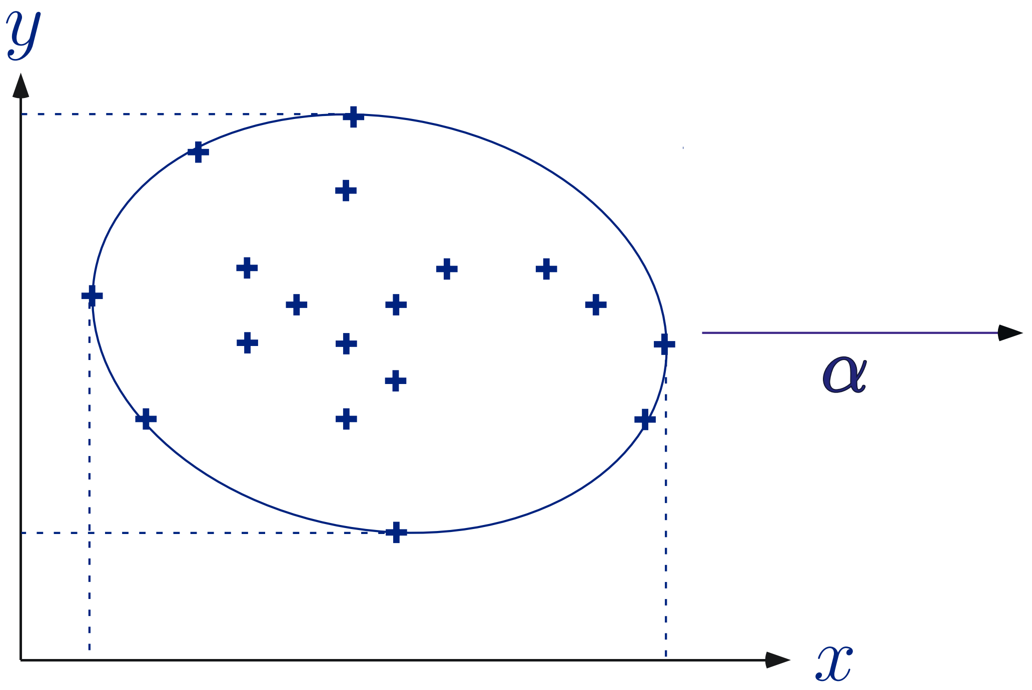
Intuition Behind Information Loss

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.

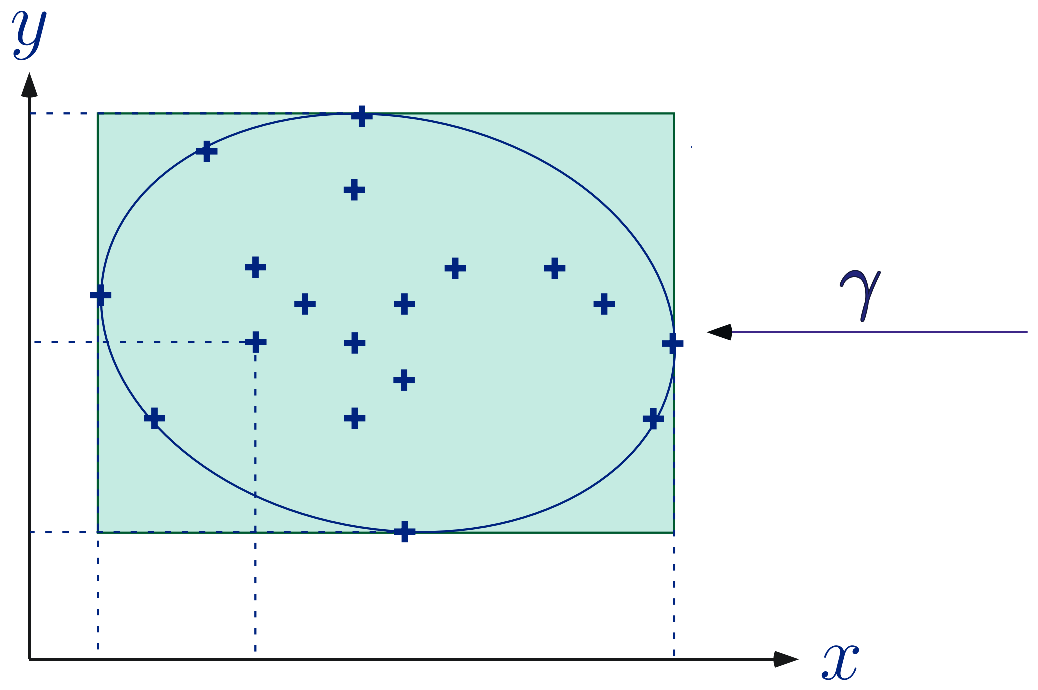
Very Basic Elements of Abstract Interpretation Theory

Abstraction α



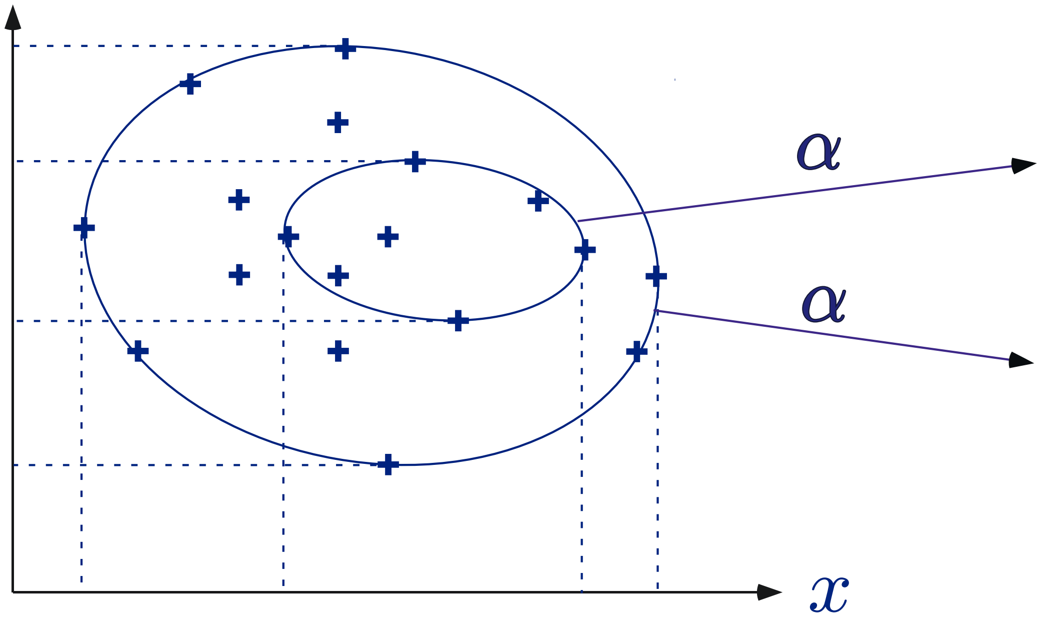
$$\{x : [1,99], y : [2,77]\}$$

Concretization γ



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

The Abstraction α is Monotone



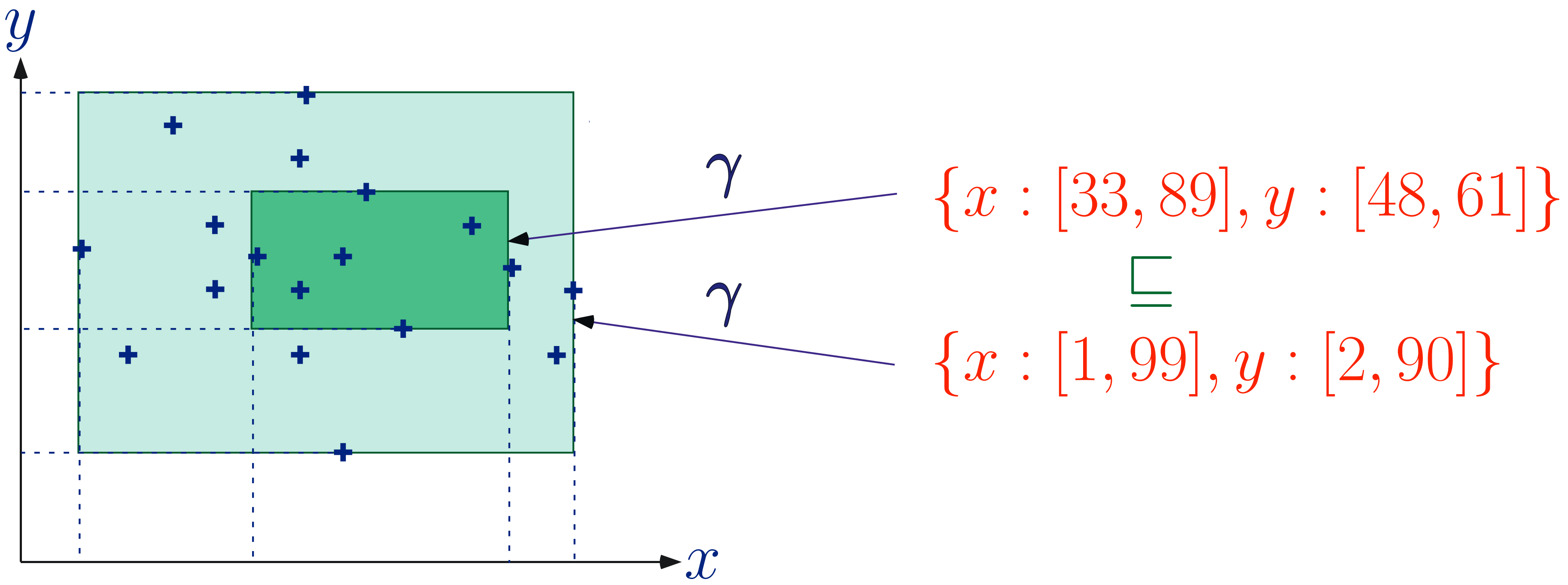
$$\{x : [33, 89], y : [48, 61]\}$$

$$\sqsubseteq$$

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

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

The Concretization γ is Monotone



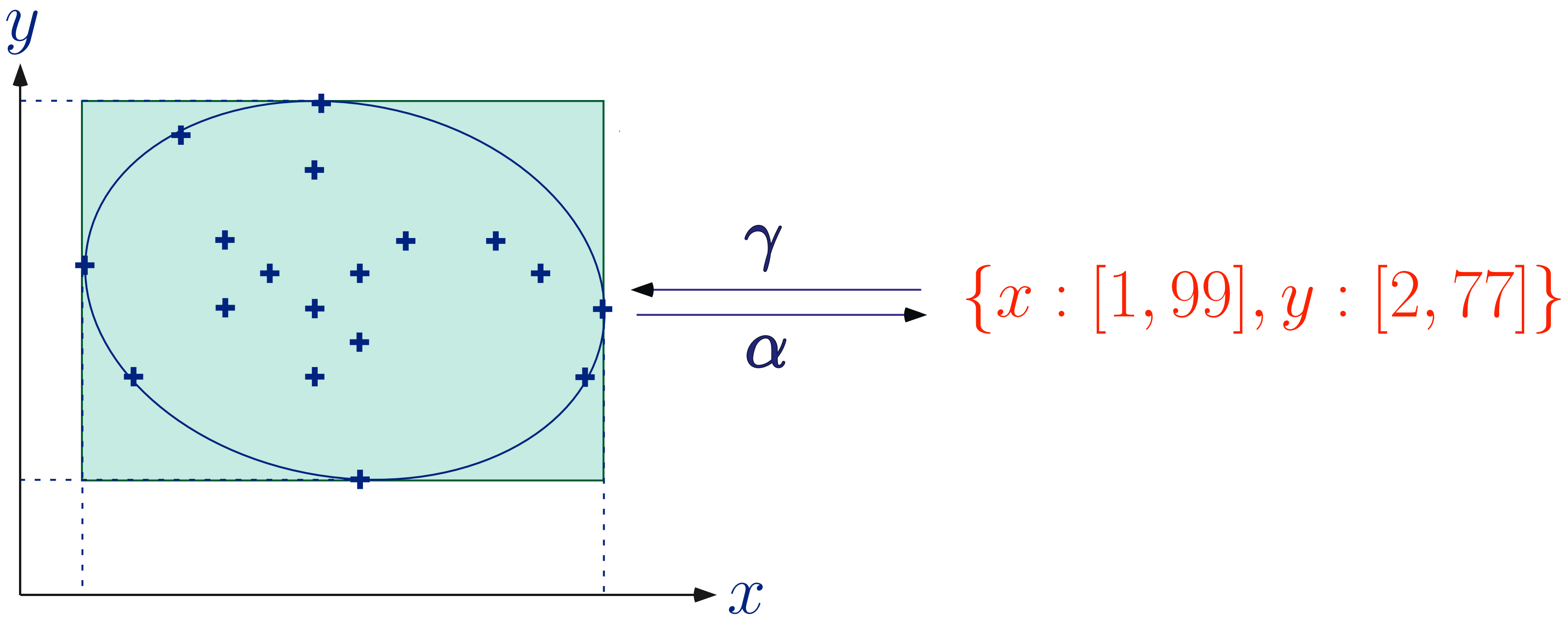
$\{x : [33, 89], y : [48, 61]\}$

$\sqsubseteq$

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

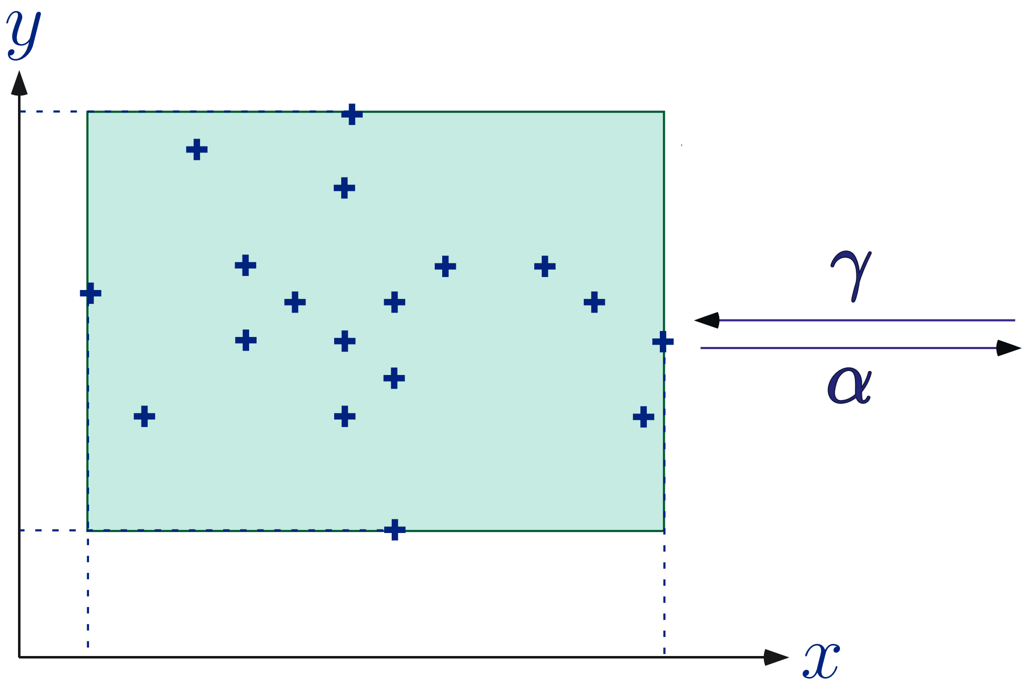
$$X \sqsubseteq Y \Rightarrow \gamma(X) \subseteq \gamma(Y)$$

The $\gamma \circ \alpha$ Composition



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

The $\alpha \circ \gamma$ Composition



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

$$=$$

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

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

Galois Connection [1]

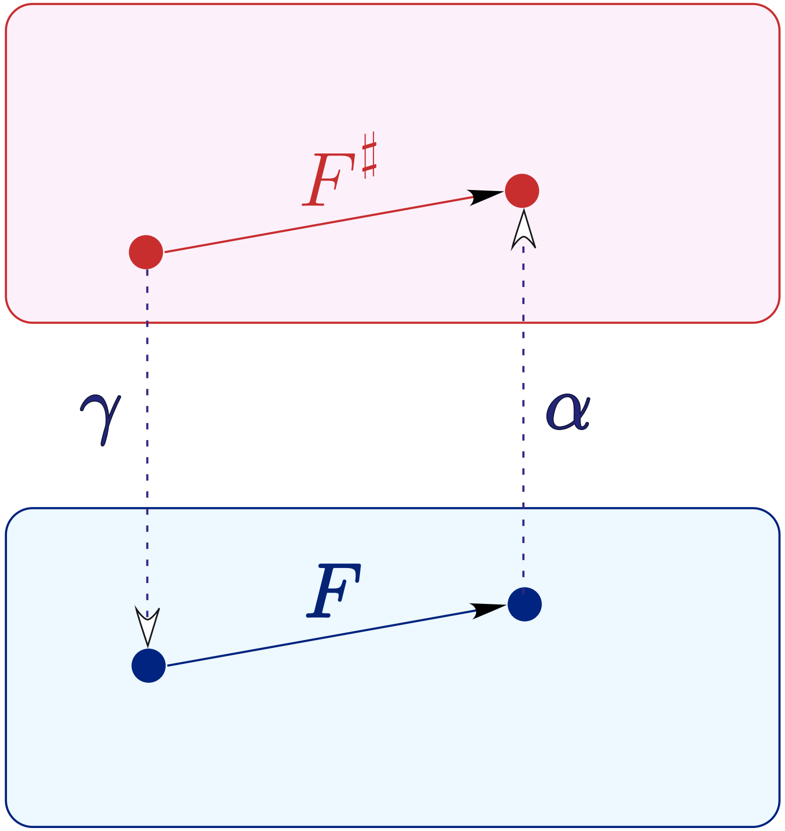
$$\langle P,\ \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle Q,\ \sqsubseteq\rangle$$

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

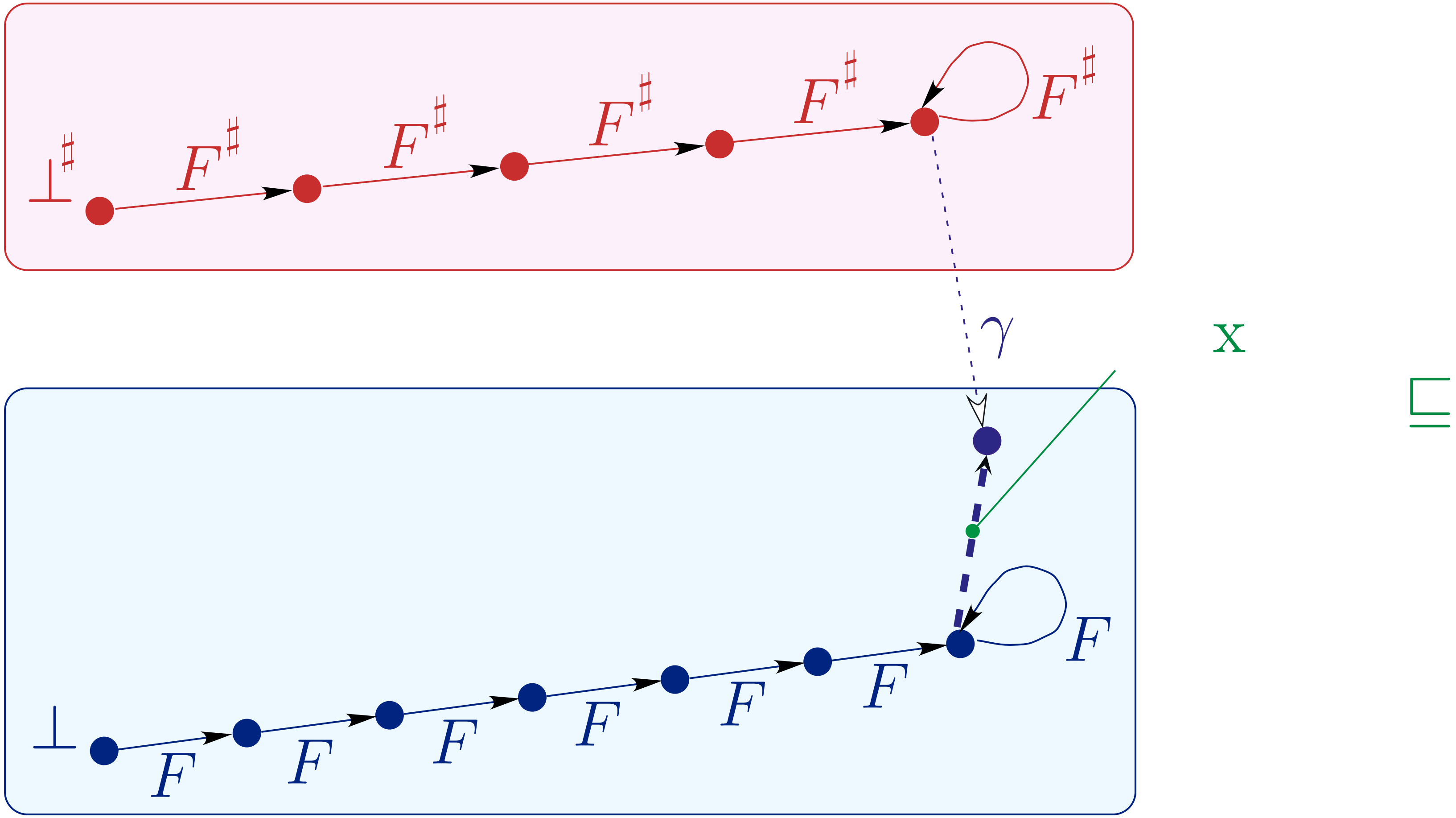


$$F^\sharp = \alpha \circ F \circ \gamma$$

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

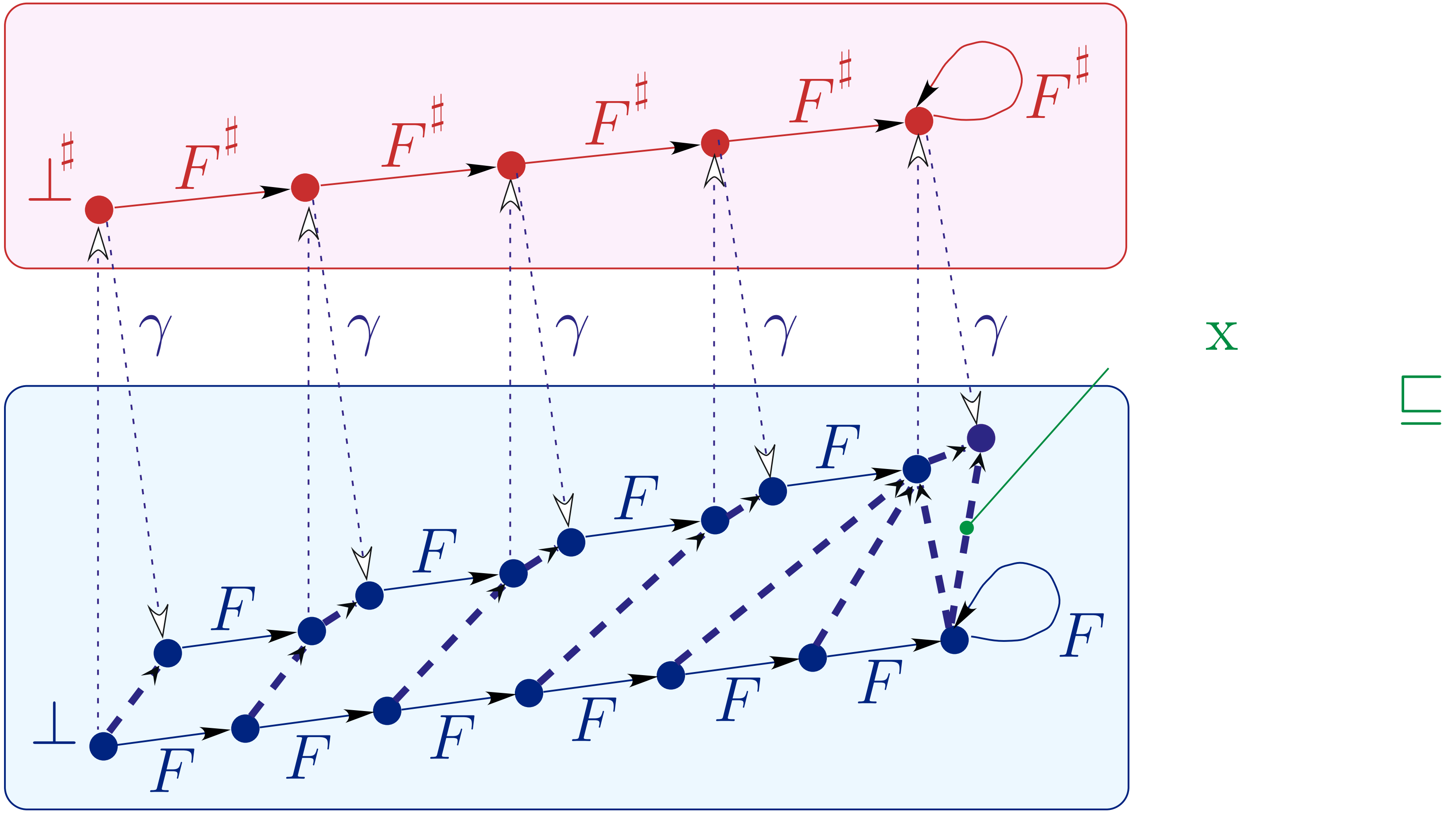
$$\langle P \xmapsto{\text{mon}} P, \dot{\subseteq}\rangle \underset{\lambda F \cdot \alpha\circ F\circ\gamma}{\overset{\lambda F^\sharp \cdot \gamma\circ F^\sharp\circ\alpha}{\leftrightarrows}} \langle Q \xmapsto{\text{mon}} Q, \dot{\sqsubseteq}\rangle$$

Fixpoint Abstraction



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

Fixpoint Abstraction



$$F^\sharp = \alpha \circ F \circ \gamma \Rightarrow \textsf{lfp}\, F \sqsubseteq \gamma(\textsf{lfp}\, F^\sharp)$$

Exact/Approximate Fixpoint Abstraction

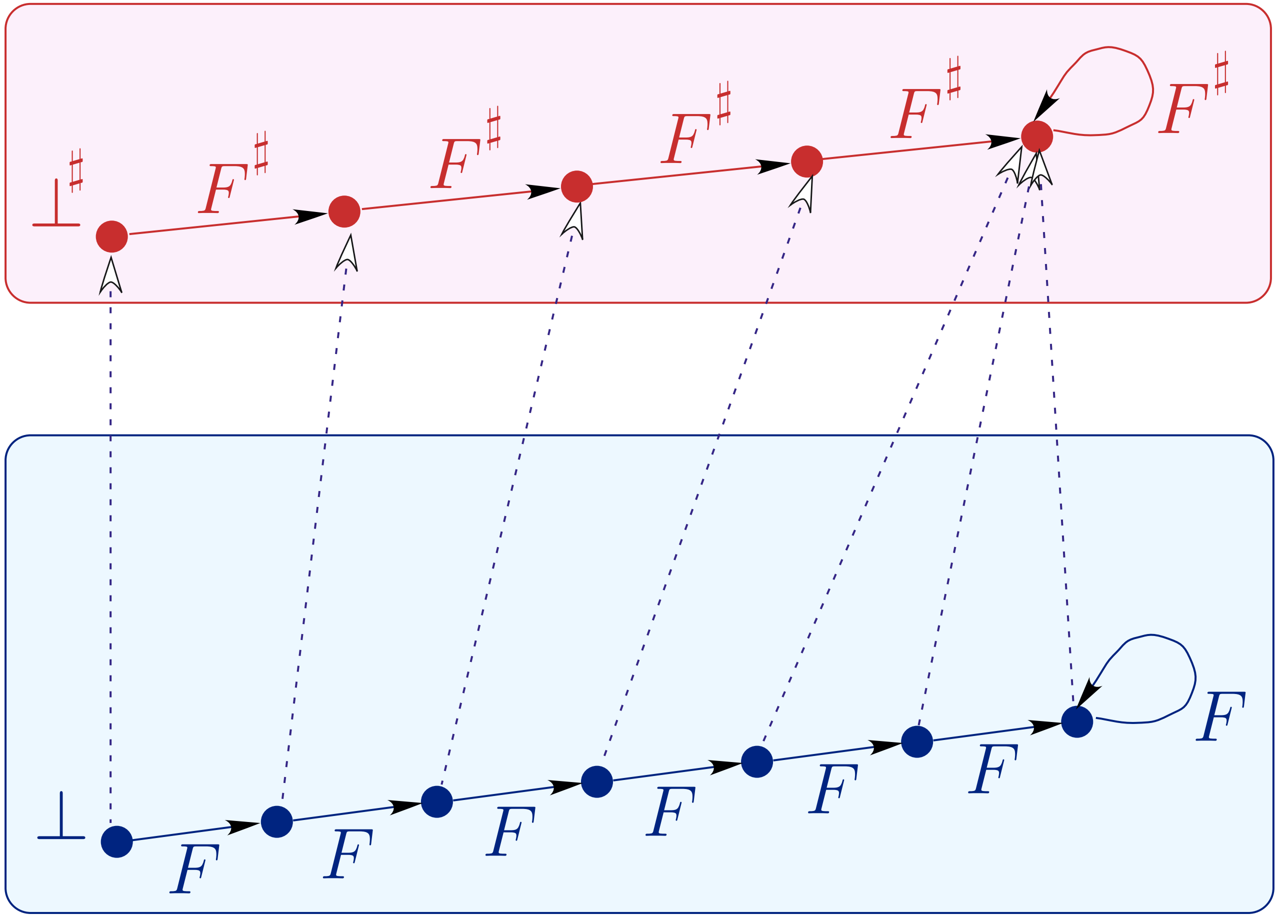
Exact Abstraction:

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

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,...]
- **Program Proofs** [HTCS 90]
- **Hierarchies of Semantics** [POPL 92]
- **Typing** [POPL 97]
- **Model Checking** [POPL 00]
- **Program Transformation** [ICLP'01,POPL 02]

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

Applications of Abstract Interpretation to Logic/Constraint Programming

- **Numerous contributors**, a.o.:

  Gianluca Amato, María Alpuente, María García de la Banda, Roberto Bagnara, Annalisa Bossi, Maurice Bruynooghe, Michael Codish, Philippe Codognet, Marco Comini, Marc-Michel Comini, Mario Coppo, Marc-Michel Corsini, Agostino Cortesi, Saumya K. Debray, Bart Demoen, Daniel De Schreye, Moreno Falaschi, Gilberto Filé, John Gallagher, Roberto Gori, Roberta Giacobazzi, Peter Greenfield, John Hannan, Michael Hanus, Manuel Hermenegildo, Patricia Hill, Gerda Janssens, Tanaka Kanamori, Andy King, Baudouin Le Charlier, Michael Leuschel, Giorgio Levi, Lunjin Lu, Salvador Lucas Alba, Kim Marriott, Fred Mesnard, Maria Chiara Meo, Dale Miller, Anne Mulkers, Ulf Nilsson, Catuscia Palamidessi, Andreas Podelski, Germán Puebla Sánchez, Elisa Quintarelli, Francesco Ranzato, Laura Ricci, Francesca Rossi, Huseyin Saglam, Dan Sahlin, Francesca Scozzari, Ehud Shapiro, Harald Søndergaard, Peter J. Stuckey, Kazunori Ueda, Pascal van Hentenrick, Peter van Roy, Robert Warren, William Winsborough, Enea Zaffanella, ...

and many more!

A New Application of Abstract Interpretation: Program Transformation

Objectives of this (Ongoing) 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.

Example Program Transformation: Constant Propagation

The Programming Language

```
a : X := ? → b;          random assignment/input
b : Y := 1 → c;          assignment
c : (X ≤ 0) → f;         nondeterministic guard
c : (X > 0) → d;
d : X := X − Y → e;
e : skip → c;            branching
f : stop;                stop
```

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:

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

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

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

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

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

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

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

Syntactic program transformations

- Program transformations are performed at the syntactic level;

$$\begin{array}{c}\text{Subject}\\ \text{program } P \in \mathbb{P}\end{array} \xmapsto[\text{transformation } \mathbb{t}]{\text{Syntactic}} \begin{array}{c}\text{Transformed}\\ \text{program } \mathbb{t}[\![P]\!] \in \mathbb{P}\end{array}$$

Program Transformation: The Semantic Point of View

- The subject and transformed programs should have "similar"/ "equivalent" semantics;

  (cannot be the "same" semantics).

The Prefix Trace Semantics

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

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

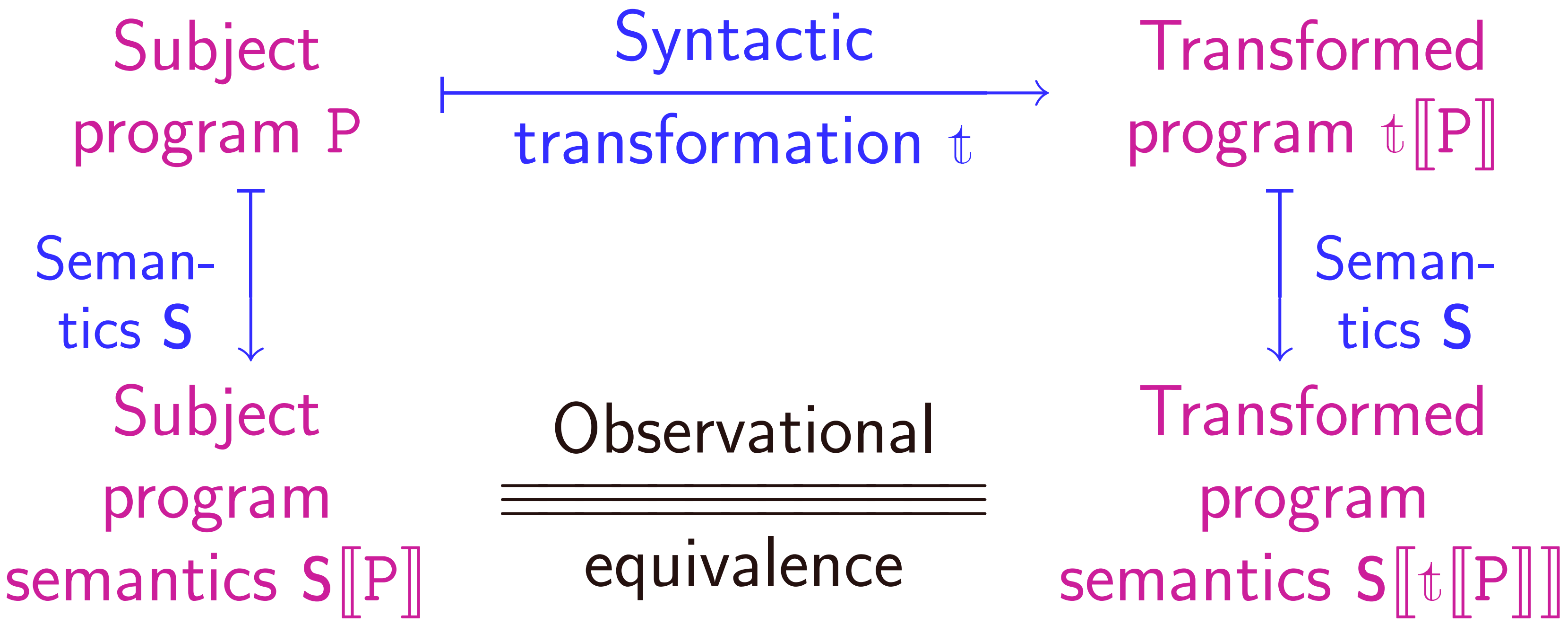
$$
\begin{array}{l}
\langle \texttt{a} : \texttt{X} := \texttt{?} \longrightarrow \texttt{b};, [\texttt{X} : \mho, \texttt{Y} : \mho] \rangle \\
\langle \texttt{b} : \texttt{Y} := \texttt{1} \longrightarrow \texttt{c};, [\texttt{X} : 1, \texttt{Y} : \mho] \rangle \\
\langle \texttt{c} : (\texttt{X} >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

Semantics of the Transformed program

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

Validation of program transformations

- Program transformations are validated at a semantic level.



Subject program P

Syntactic transformation $\mathbb{t}$

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

Semantics $\mathbf{S}$

Semantics $\mathbf{S}$

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

Observational equivalence

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

Observational semantics

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

Online Versus Offline Program Transformation

Online Transformation : use the actual values of variables (i.e. the program concrete semantics);

Offline Transformation : use a preliminary static analysis of the program (i.e. the program abstract semantics).

(Constant propagation is an offline program transformation, otherwise the transformation might loop for non-constants.)

Basic Elements of Program Transformation Design by Abstract Interpretation

Element 1: Observational Equivalence

- The semantics of the subject and transformed programs should be equivalent at some level of observation:

$$\begin{tikzcd}
\text{Subject program } P \arrow[r, mapsto, "\text{Syntactic}", "\text{transformation } \mathbb{t}"'] \arrow[d, mapsto, "\text{Semantics } \mathsf{S}"'] & \text{Transformed program } \mathbb{t}[\![P]\!] \arrow[d, mapsto, "\text{Semantics } \mathsf{S}"] \\
\text{Subject program semantics } \mathsf{S}[\![P]\!] \arrow[r, equal, "\text{Observational}", "\text{equivalence } \equiv_{\mathcal{O}}"'] & \text{Transformed program semantics } \mathsf{S}[\![\mathbb{t}[\![P]\!]]\!]
\end{tikzcd}$$

Element 1: Observational Equivalence, Cont'd

- Observational equivalence can be formalized as an abstract interpretation:

$$\begin{tikzcd}
\begin{array}{c}\text{Subject}\\ \text{program}\\ \text{semantics}\\ \mathsf{S}[\![\mathrm{P}]\!]\end{array} \arrow[rr, equal, "\text{Observational}", "\text{equivalence } \equiv_{\mathcal{O}}"'] \arrow[dr, mapsto, "\alpha_{\mathcal{O}}"'] & & \begin{array}{c}\text{Transformed}\\ \text{program}\\ \text{semantics}\\ \mathsf{S}[\![\mathfrak{t}[\![\mathrm{P}]\!]]\!]\end{array} \arrow[dl, mapsto, "\alpha_{\mathcal{O}}"] \\
& \text{Observational semantics} &
\end{tikzcd}$$

Example: Constant Propagation

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

Element 2: Induced Semantic Transformation

- A syntactic program transformation induces a semantic program transformation:

$$\begin{tikzcd}
\text{Subject program } P \arrow[r, mapsto, "\text{Syntactic}", "\text{transformation } \mathbb{t}"'] \arrow[d, mapsto, "\text{Semantics } \mathbf{S}"'] & \text{Transformed program } \mathbb{t}\llbracket P \rrbracket \arrow[d, mapsto, "\text{Semantics } \mathbf{S}"'] \\
\text{Subject semantics } \mathbf{S}\llbracket P \rrbracket \arrow[r, mapsto, "\text{Semantic}", "\text{transformation } t"'] & \begin{array}{c}\text{Transformed semantics}\\ t[\mathbf{S}\llbracket P \rrbracket] \equiv_{\mathcal{O}} \mathbf{S}\llbracket \mathbb{t}\llbracket P \rrbracket \rrbracket\end{array}
\end{tikzcd}$$

- We study this semantic transformation from an abstract interpretation point of view.

Example: Semantic Constant Propagation

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

Element 3: Semantic Correctness

- From a validation point of view, the correctness of a syntactic transformation can be proved by reasoning on the induced semantic transformation:

$$\begin{tikzcd}
\text{Subject program } \mathrm{P} \arrow[r, mapsto, "\text{Syntactic}", "\text{transformation } \mathbb{t}"'] \arrow[d, mapsto, "\text{Seman-tics } \mathbf{S}"'] & \text{Transformed program } \mathbb{t}[\![\mathrm{P}]\!] \arrow[d, mapsto, "\text{Seman-tics } \mathbf{S}"'] \\
\text{Subject semantics } \mathbf{S}[\![\mathrm{P}]\!] \arrow[r, mapsto, "\text{Semantic}", "\text{transformation } \mathrm{t}"'] & \text{Transformed semantics } \mathrm{t}[\mathbf{S}[\![\mathrm{P}]\!]]
\end{tikzcd}$$

$$\mathbf{S}[\![\mathrm{P}]\!] \equiv_{\mathcal{O}} \mathrm{t}[\mathbf{S}[\![\mathrm{P}]\!]] \equiv_{\mathcal{O}} \mathbf{S}[\![\mathbb{t}[\![\mathrm{P}]\!]]\!]$$

Element 4: 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) the performed actions is recorded;
  - Program execution times are completely abstracted (but might be included in the operational semantics);

Element 4: Correspondence Between Syntax and Semantics, Cont'd

- The correspondence between syntax and semantics is an abstract interpretation:

$$\mathsf{po}\langle\mathfrak{D};\ \sqsubseteq\rangle \;\underset{\mathbb{p}}{\overset{\mathsf{S}}{\mathrel{\substack{\longleftarrow\\ \twoheadrightarrow}}}}\; \mathsf{po}\langle\mathbb{P}/_{\equiv};\ \sqsubseteq\rangle$$

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.

Element 5: Semantics-Based Design

- From a constructive design point of view, the syntactic transformation can be formally derived from a correct semantic transformation:

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

Semantic to Syntactic Constant Propagation

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

Element 6: Transformations as Approximations

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

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

$$\begin{tikzcd}[column sep=huge, row sep=huge]
\text{Subject program } P \arrow[r, mapsto, "\text{Syntactic}", "\text{transformation } \mathbb{t}"'] \arrow[d, shift right, "\mathsf{S}"'] & \text{Transformed program } \mathbb{t}[\![P]\!] \sqsupseteq \mathbb{p}[t[\mathsf{S}[\![P]\!]]] \arrow[d, shift right, "\mathsf{S}"'] \\
\text{Subject program semantics } \mathsf{S}[\![P]\!] \arrow[u, shift right, "\mathbb{p}"'] \arrow[r, shift right, "\text{Semantic transformation } t"'] & \text{Transformed program semantics } t[\mathsf{S}[\![P]\!]] \sqsubseteq \mathsf{S}[\![\mathbb{t}[\![P]\!]]\!] \arrow[u, shift right, "\mathbb{p}"'] \arrow[l, shift right, "\gamma_t"']
\end{tikzcd}$$

Element 6: Transformations as Approximations (Cont'd)

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

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

$$\begin{tikzcd}
\text{Subject program } P \arrow[r, shift right, "\text{Syntactic transformation } \mathbb{t}"'] \arrow[d, shift right, "\mathsf{S}"'] & \text{Transformed program } \mathbb{t}[\![P]\!] \sqsupseteq \mathbb{p}[\mathrm{t}[\mathsf{S}[\![P]\!]]] \arrow[l, shift right, "\gamma_{\mathbb{t}}"'] \arrow[d, shift right, "\mathsf{S}"'] \\
\text{Subject program semantics } \mathsf{S}[\![P]\!] \arrow[r, shift right, "\text{Semantic transformation } \mathrm{t}"'] \arrow[u, shift right, "\mathbb{p}"'] & \text{Transformed program semantics } \mathrm{t}[\mathsf{S}[\![P]\!]] \sqsubseteq \mathsf{S}[\![\mathbb{t}[\![P]\!]]] \arrow[l, shift right, "\gamma_{\mathrm{t}}"'] \arrow[u, shift right, "\mathbb{p}"']
\end{tikzcd}$$

Intuition for Transformations as Abstractions

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

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

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

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

$\ldots$

$\mathtt{X} - (2 * \mathtt{Y} - 1)$

$\mathtt{X} - \mathtt{Y}$

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

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

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

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

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

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

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

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

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

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

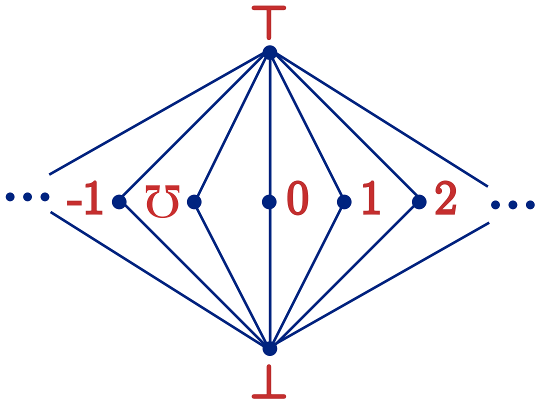
Element 7: offline Transformations

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

  ⟶ This can be formalized by abstract interpretation.

Example: Kildall's Constant Propagation

- Kildall's lattice (POPL'73):



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

- Pointwise extension to variable environments and program labels;

Example: Kildall's Constant Propagation, Cont'd

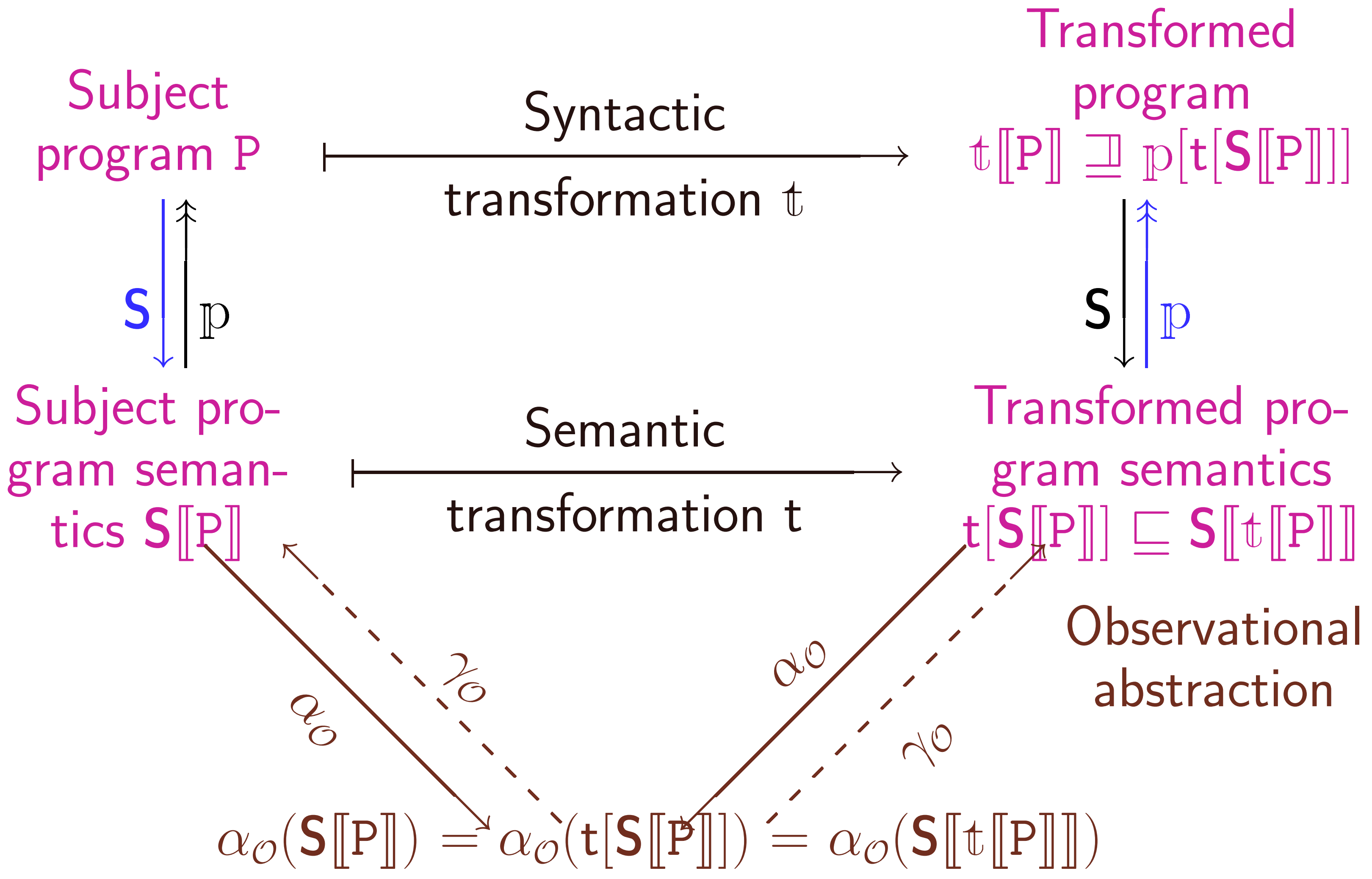
- Elementwise abstraction of a set $\mathcal{T}$ of traces:

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

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

Principle of the Formalization of Program Transformation by Abstract Interpretation

Principle of Online Program Transformation



Subject program P

Syntactic transformation $\mathbb{t}$

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

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

Semantic transformation t

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

Observational abstraction

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

Principle of Offline Program Transformation (1)

$$\begin{tikzcd}[column sep=huge, row sep=huge]
\begin{array}{c}\text{Subject}\\ \text{program } \mathrm{P}\end{array} \arrow[r, mapsto, "\text{Syntactic annotation}", "\mathbb{a}"'] \arrow[d, shift right, "\mathsf{S}"'] & \begin{array}{c}\text{Annotated}\\ \text{subject program}\\ \underline{\mathrm{P}} = \mathbb{a}[\![\mathrm{P}]\!]\end{array} \arrow[r, mapsto, "\text{Syntactic transformation}", "\mathbb{t}"'] \arrow[d, shift left, "\mathsf{S}"] & \begin{array}{c}\text{Transformed}\\ \text{program}\\ \mathbb{t}[\![\underline{\mathrm{P}}]\!] \sqsupseteq \mathbb{p}[\mathrm{t}[\mathsf{S}[\![\underline{\mathrm{P}}]\!]]]\end{array} \arrow[d, shift right, "\mathsf{S}"'] \\
\begin{array}{c}\text{Subject}\\ \text{program}\\ \text{semantics } \mathsf{S}[\![\mathrm{P}]\!]\end{array} \arrow[u, shift right, two heads, "\mathbb{p}"'] \arrow[r, mapsto, "\text{Semantic annotation}", "\mathrm{a}"'] \arrow[dr, "\alpha_{\mathcal{O}}"'] & \begin{array}{c}\text{Annotated}\\ \text{subject program}\\ \text{semantics } \mathsf{S}[\![\underline{\mathrm{P}}]\!]\end{array} \arrow[u, shift left, two heads, "\mathbb{p}"] \arrow[r, mapsto, "\text{Semantic transformation}", "\mathrm{t}"'] \arrow[d, shift right, "\alpha_{\mathcal{O}}"'] & \begin{array}{c}\text{Transformed}\\ \text{program semantics}\\ \mathrm{t}[\mathsf{S}[\![\underline{\mathrm{P}}]\!]] \sqsubseteq \mathsf{S}[\![\mathbb{t}[\![\underline{\mathrm{P}}]\!]]\!]\end{array} \arrow[u, shift right, two heads, "\mathbb{p}"'] \arrow[dl, "\alpha_{\mathcal{O}}"] \\
 & \alpha_{\mathcal{O}}(\mathsf{S}[\![\mathrm{P}]\!]) = \alpha_{\mathcal{O}}(\mathsf{S}[\![\underline{\mathrm{P}}]\!]) = \alpha_{\mathcal{O}}(\mathrm{t}[\mathsf{S}[\![\underline{\mathrm{P}}]\!]]) = \alpha_{\mathcal{O}}(\mathsf{S}[\![\mathbb{t}[\![\underline{\mathrm{P}}]\!]]\!]) \arrow[ul, dashed, "\gamma_{\mathcal{O}}"'] \arrow[u, shift right, dashed, "\gamma_{\mathcal{O}}"'] \arrow[ur, dashed, "\gamma_{\mathcal{O}}"] &
\end{tikzcd}$$

Observational abstraction

Principle of Offline Program Transformation (2)

$$\begin{tikzcd}[column sep=huge, row sep=huge]
\begin{array}{c}\text{Subject}\\ \text{program}\\ P\end{array} \arrow[r, mapsto, "\text{Static program}", "\text{analysis } \overline{\mathsf{S}}"'] \arrow[d, shift right, "\mathsf{S}"'] & \langle P,\ \overline{\mathsf{S}}[\![P]\!]\rangle \arrow[r, mapsto, "\text{Syntactic trans-}", "\text{formation } \mathbb{t}"'] \arrow[d, shift left, "\mathsf{S}"] & \begin{array}{c}\text{Transformed program}\\ \mathbb{t}[\![P]\!] \sqsupseteq \mathbb{p}[\mathrm{t}[\mathsf{S}[\![P]\!], \alpha(\mathsf{S}[\![P]\!])]]\end{array} \arrow[d, shift right, "\mathsf{S}"'] \\
\begin{array}{c}\text{Subject}\\ \text{program}\\ \text{semantics}\\ \mathsf{S}[\![P]\!]\end{array} \arrow[u, shift right, "\mathbb{p}"'] \arrow[r, mapsto, "\text{Semantic}", "\text{abstraction } \alpha"'] \arrow[dr, "\alpha_{\mathcal{O}}"', shift right] & \langle \mathsf{S}[\![P]\!],\ \alpha(\mathsf{S}[\![P]\!])\rangle \arrow[u, shift left, "\mathbb{p}"] \arrow[u, dotted, shift right=6, "\sqsupseteq"] \arrow[r, mapsto, "\text{Semantic trans-}", "\text{formation t}"'] \arrow[d, shift right, "\alpha_{\mathcal{O}}"'] & \begin{array}{c}\text{Transformed}\\ \text{program semantics}\\ \mathrm{t}[\mathsf{S}[\![P]\!], \alpha(\mathsf{S}[\![P]\!])] \sqsubseteq \mathsf{S}[\![\mathbb{t}[\![P]\!]]\!]\end{array} \arrow[u, shift right, "\mathbb{p}"'] \arrow[dl, shift left, "\alpha_{\mathcal{O}}"] \\
& \alpha_{\mathcal{O}}(\mathsf{S}[\![P]\!]) = \alpha_{\mathcal{O}}(\mathrm{t}[\mathsf{S}[\![P]\!], \alpha(\mathsf{S}[\![P]\!])) = \alpha_{\mathcal{O}}(\mathsf{S}[\![\mathbb{t}[\![P]\!]]\!]) \arrow[ul, dashed, shift right, "\gamma_{\mathcal{O}}"'] \arrow[u, dashed, shift right, "\gamma_{\mathcal{O}}"'] \arrow[ur, dashed, shift left, "\gamma_{\mathcal{O}}"] & \begin{array}{c}\text{Observational}\\ \text{abstraction}\end{array}
\end{tikzcd}$$

Design of Program Transformations by Abstract Interpretation

Back to Principles ...

$$\begin{tikzcd}[column sep=huge, row sep=huge]
\text{Subject program } \mathrm{P} \arrow[rr, mapsto, "\text{Syntactic}", "\text{transformation } \mathbb{t}"'] \arrow[d, shift right, "\mathsf{S}"'] & & \text{Transformed program } \boxed{\mathbb{t}\llbracket \mathrm{P} \rrbracket \sqsupseteq \mathbb{p}[\mathsf{t}[\mathsf{S}\llbracket \mathrm{P} \rrbracket]]} \arrow[d, shift right, "\mathsf{S}"'] \\
\text{Subject program semantics } \mathsf{S}\llbracket \mathrm{P} \rrbracket \arrow[u, shift right, two heads, "\mathbb{p}"'] \arrow[rr, mapsto, "\text{Semantic}", "\text{transformation } \mathsf{t}"'] \arrow[dr, "\alpha_{\mathcal{O}}"'] & & \text{Transformed program semantics } \mathsf{t}[\mathsf{S}\llbracket \mathrm{P} \rrbracket] \sqsubseteq \mathsf{S}\llbracket \mathbb{t}\llbracket \mathrm{P} \rrbracket \rrbracket \arrow[u, shift right, two heads, "\mathbb{p}"'] \arrow[dl, "\alpha_{\mathcal{O}}"] \arrow[d, phantom, "\text{Observational abstraction}"] \\
& \alpha_{\mathcal{O}}(\mathsf{S}\llbracket \mathrm{P} \rrbracket) = \alpha_{\mathcal{O}}(\mathsf{t}[\mathsf{S}\llbracket \mathrm{P} \rrbracket]) = \alpha_{\mathcal{O}}(\mathsf{S}\llbracket \mathbb{t}\llbracket \mathrm{P} \rrbracket \rrbracket) \arrow[ul, dashed, bend left, "\gamma_{\mathcal{O}}"'] \arrow[ur, dashed, bend right, "\gamma_{\mathcal{O}}"] & \;
\end{tikzcd}$$

Design of Program Transformation Algorithms

$$\begin{aligned}
\mathbb{t}[\![\mathrm{P}]\!] &\sqsupseteq \mathbb{p}[\mathsf{t}[\mathsf{S}[\![\mathrm{P}]\!]]] \\
&= \mathbb{p}[\mathsf{t}[\mathbf{lfp}^{\subseteq}\ \mathsf{F}^{*}[\![\mathrm{P}]\!]]] \\
&\sqsupseteq \ldots \\
&= \mathbf{lfp}^{\subseteq^{\sharp}}\ \mathbb{F}^{\sharp}[\![\mathrm{P}]\!]
\end{aligned}$$

← apply fixpoint transfer /approximation theorems

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**
            $\mathtt{A}_c :=$ **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** $\mathtt{Q}$.
```

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);
  - Program monitoring (POPL'02);
  - Program reduction (e.g. transition compression);
  - Slicing;

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.