Systematic Design of Program Transformation Frameworks by Abstract Interpretation

Patrick COUSOT
École normale supérieure
Patrick.Cousot@ens.fr
www.di.ens.fr/~cousot

Radhia COUSOT
CNRS & École polytechnique
Radhia.Cousot@lix.polytechnique.fr
lix.polytechnique.fr/~rcousot

POPL'02, Portland 16–18 Jan 2002

Motivations

Program Transformation & Abstract Interpretation

In semantics-based (offline) program transformation, such as:

- constant propagation,
- partial evaluation,
- slicing,

abstract interpretation is classically used in a preliminary program static analysis phase:

- to collect the information about the program runtime behaviors,
- and determine which transformations are applicable.

Present Objective

Our present goal is **quite different**:

- Formalize **the program transformation itself**;

  With two objectives:

  - a program transformation correctness proof method;
  - a program transformation design methodology.

- Abstract interpretation is the appropriate framework to reach these objectives.

Abstract Interpretation

Abstract Interpretation

- **Abstract interpretation** formalizes the **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.

Abstract Interpretation (Cont'd)

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

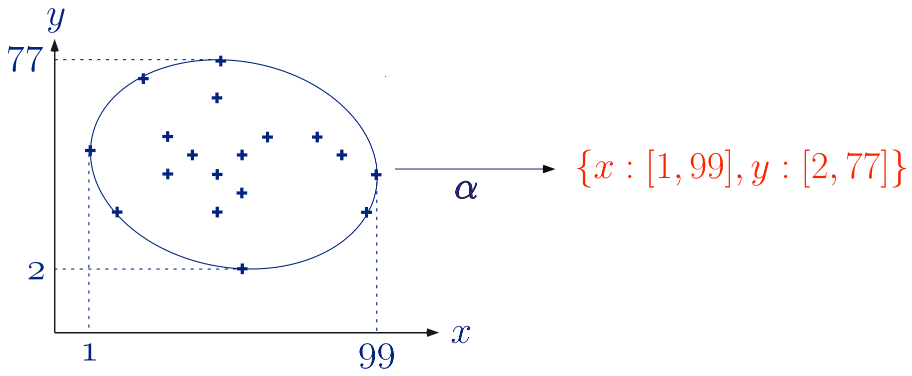
A Few Applications of Abstract Interpretation

Techniques involving approximations are naturally formalized by abstract interpretation:

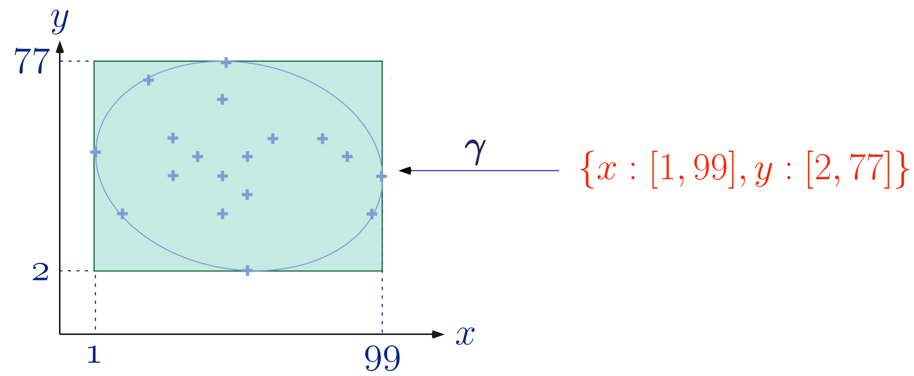
- **Static Program Analysis** [POPL 77,78,79]
- **Hierarchies of Semantics (including Proofs)** [POPL 92, TCS 02]
- **Typing** [POPL 97]
- **Model Checking** [POPL 00]
- **Program Transformation** [POPL 02]

Very Basic Elements of Abstract Interpretation Theory

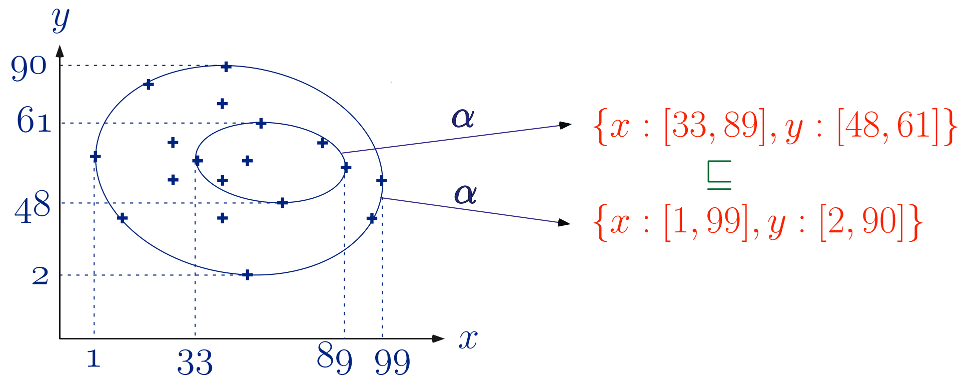
Abstraction α



Concretization γ

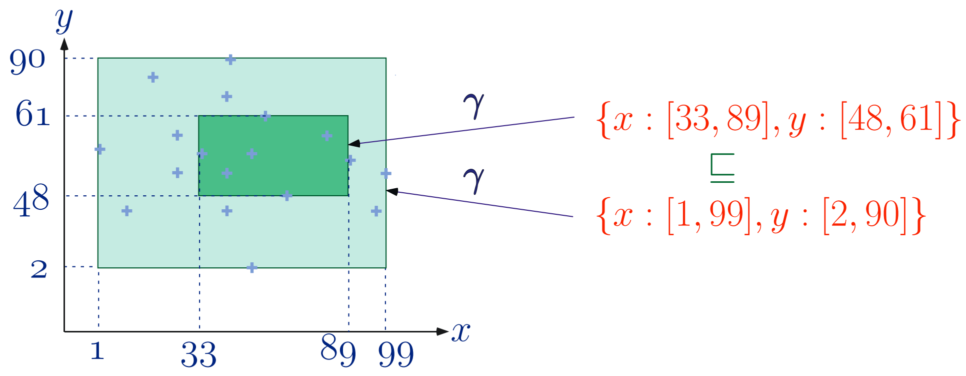


The Abstraction α is Monotone



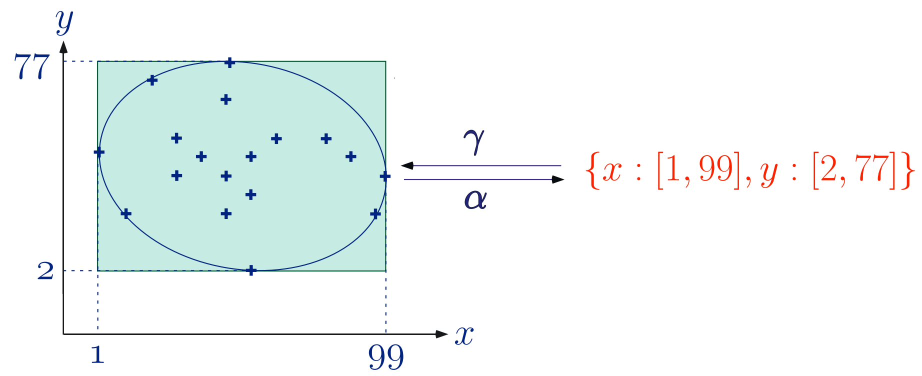
$X \subseteq Y \Rightarrow \alpha(X) \sqsubseteq \alpha(Y)$

The Concretization γ is Monotone



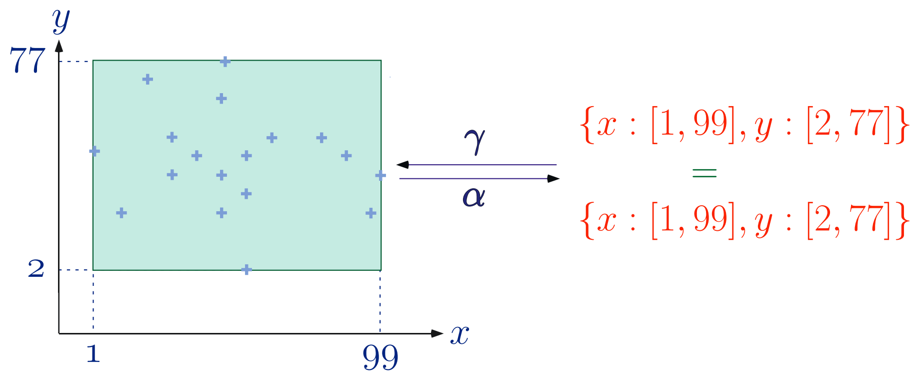
$$X \sqsubseteq Y \Rightarrow \gamma(X) \subseteq \gamma(Y)$$

The $\gamma \circ \alpha$ Composition



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

The $\alpha \circ \gamma$ Composition



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

Galois Connection [1, 2]

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

is defined as

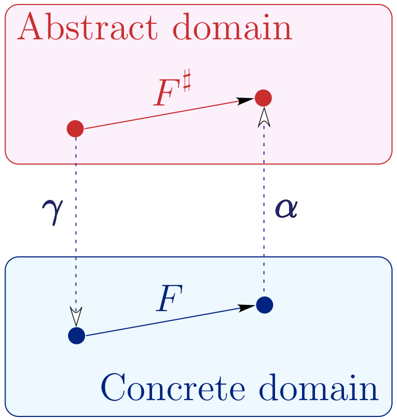
- α is monotone
- γ is monotone
- $X \subseteq \gamma \circ \alpha(X)$
- $\alpha \circ \gamma(Y) \sqsubseteq Y$

iff

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

[1] for short, more precisely "semi-dual Galois connections".

[2] see [POPL 79] for equivalent formalizations using closure operators, ideals, etc. and [JLC 92] for weaker hypotheses if no best approximation.



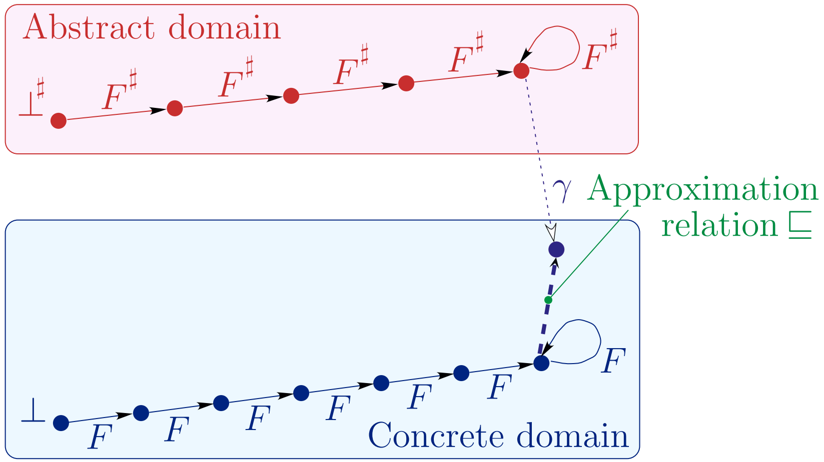
Function Abstraction

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

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

$$\langle P \xrightarrow{\text{mon}} P,\ \dot{\subseteq}\rangle \xrightleftharpoons[\lambda F \cdot \alpha\circ F\circ\gamma]{\lambda F^\sharp \cdot \gamma\circ F^\sharp\circ\alpha} \langle Q \xrightarrow{\text{mon}} Q,\ \dot{\sqsubseteq}\rangle$$

Approximate Fixpoint Abstraction



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

Online Program Transformation

(1) Online Program Transformation

- Program transformation is a syntactic process;
- maps a subject program into a transformed program;
- Both subject and transformed programs are syntactic objects.

(2) Online Program Transformation

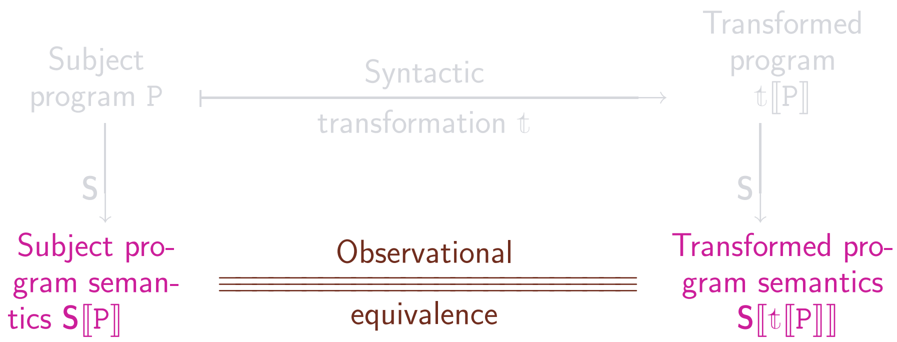
- Program transformations refer to the semantics of the subject and transformed programs:
  - Online program transformations use values manipulated during program execution, hence directly refer to the source concrete semantics;
  - Offline program transformations use a preliminary static analysis of the source program, hence refer to its abstract semantics;

(2) Online Program Transformation

(3) Online Program Transformation

- The subject semantics and transformed semantics are different in general;
- However they should be equivalent, at some level of observation.

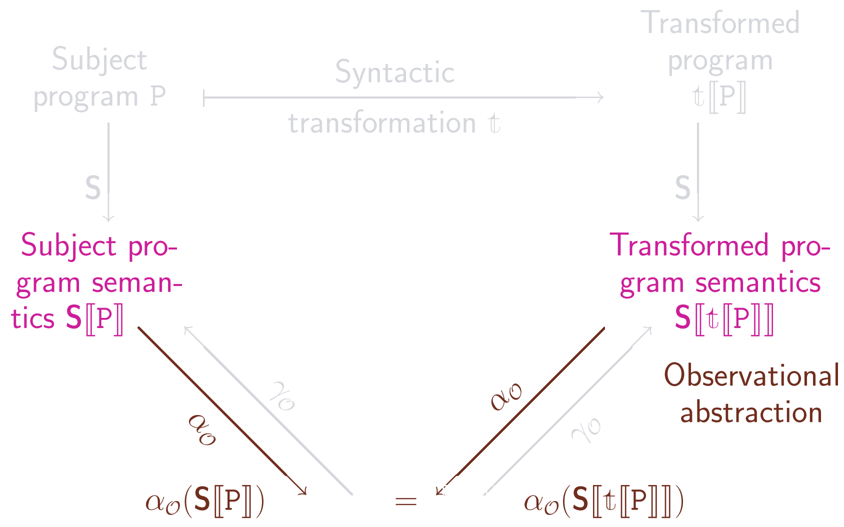
(3) Online Program Transformation



(3) Online Program Transformation

- 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!

(3) Online Program Transformation



(3) Example: Partial Evaluation

Subject program:
```
Y := 1
X := Y - 1
```
Transformed program:
```
Y := 1
X := 0
```

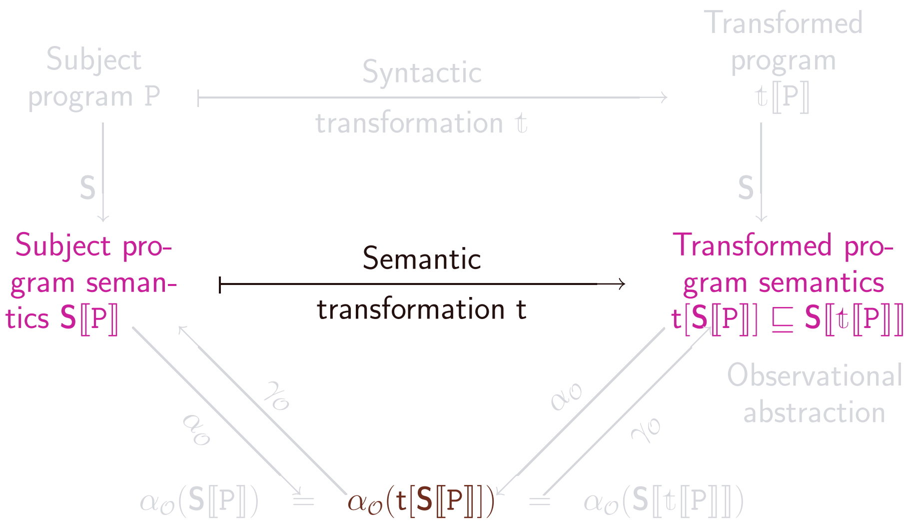
Subject semantics	Transformed semantics	$\xrightarrow{\alpha_{\mathcal{O}}}$	Observational semantics
[X:℧,Y:℧]	[X:℧,Y:℧]	$\xrightarrow{\alpha_{\mathcal{O}}}$	[X:℧,Y:℧]
↓ `Y := 1`	↓ `Y := 1`	$\xrightarrow{\alpha_{\mathcal{O}}}$	↓
[X:℧,Y:1]	[X:℧,Y:1]	$\xrightarrow{\alpha_{\mathcal{O}}}$	[X:℧,Y:1]
↓ `X := Y - 1`	↓ `X := 0`	$\xrightarrow{\alpha_{\mathcal{O}}}$	↓
[X:0,Y:1]	[X:0,Y:1]	$\xrightarrow{\alpha_{\mathcal{O}}}$	[X:0,Y:1]

(4) Online Program Transformation

- The syntactic transformation induces a semantic transformation:

  The subject semantics is mapped to the transformed semantics;
- 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).

(4) Online Program Transformation



(5) 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);

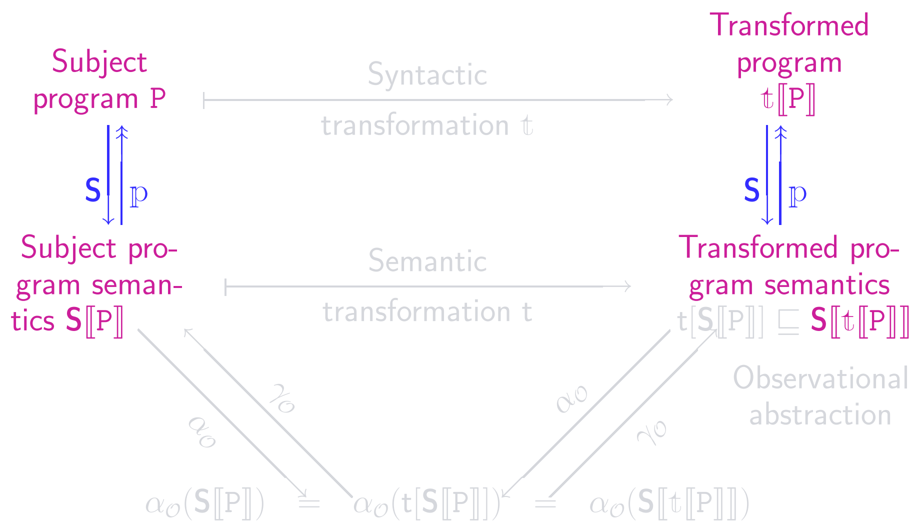
(5) Correspondence Between Syntax and Semantics, Cont'd

- The correspondence between syntax and semantics is an abstraction:

$$\text{po}\langle \mathfrak{D};\ \sqsubseteq\rangle \underset{\mathbb{p}}{\overset{\mathsf{S}}{\leftrightarrows}} \text{po}\langle \mathbb{P}/_{\equiv};\ \sqsubseteq\rangle$$

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

(5) Online Program Transformation



(6) Semantic Transformations as Approximations

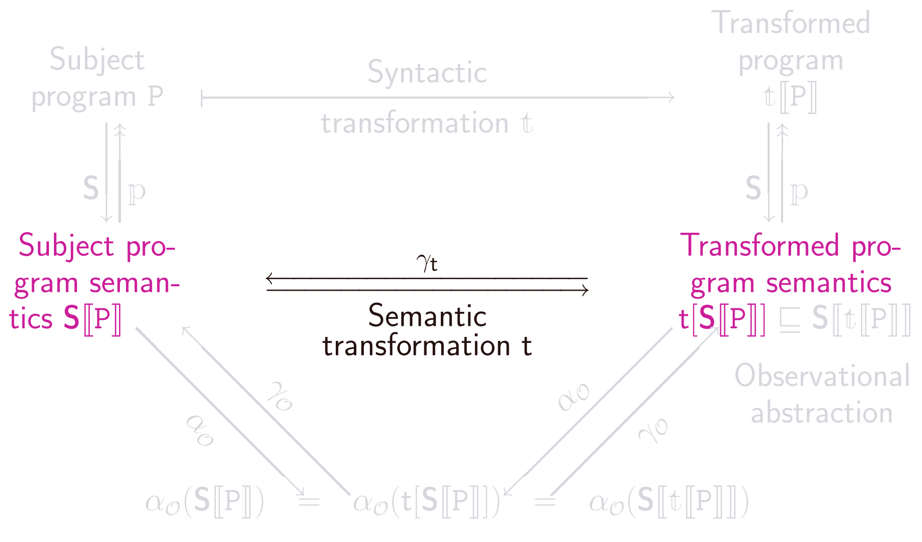
- A semantic program transformation is a loss of information on the semantics of the subject program;

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

(6) Example: Partial Evaluation

$$\left.\begin{array}{l} \mathsf{S}[\![\texttt{Y := 1; X := 0;}]\!] \\ \mathsf{S}[\![\texttt{Y := 1; X := Y - 1;}]\!] \\ \mathsf{S}[\![\texttt{Y := 1; X := 2 * Y - 2;}]\!] \\ \mathsf{S}[\![\texttt{Y := 1; X := Y * (Y - 1);}]\!] \\ \mathsf{S}[\![\ldots]\!] \end{array}\right\} \underset{\text{Semantic transformation t}}{\overset{\gamma_t}{\leftrightarrows}} \mathsf{S}[\![\texttt{Y := 1; X := 0;}]\!]$$

(6) Online Program Transformation

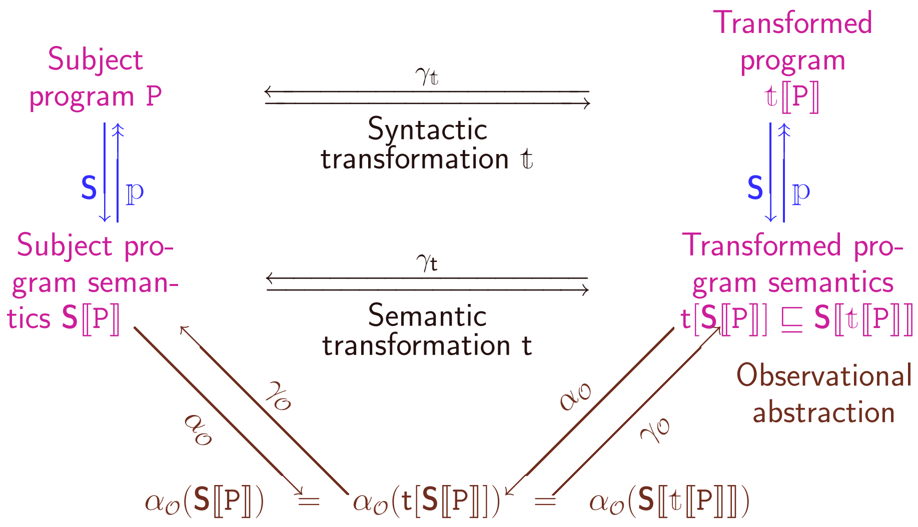


(7) Syntactic Transformations as Approximations

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

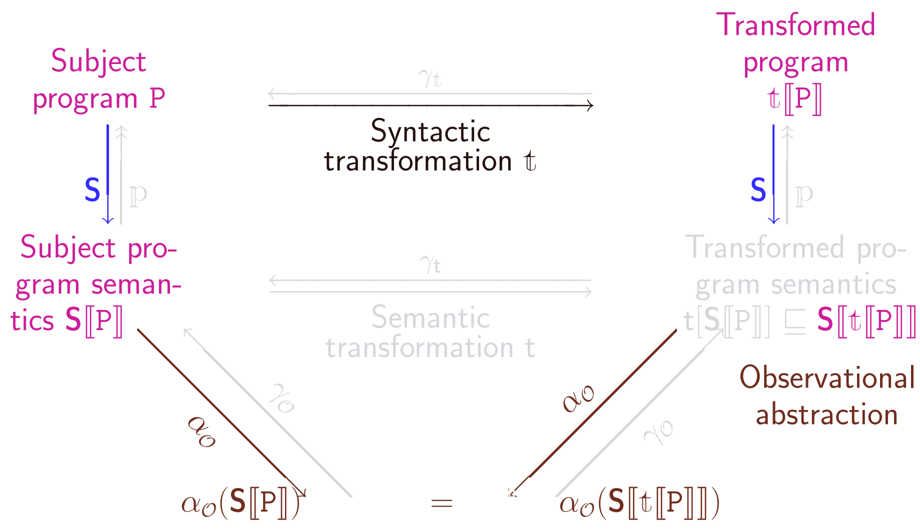
  $\longrightarrow$ The syntactic program transformation is an abstraction;

(7) Online Program Transformation (Done)



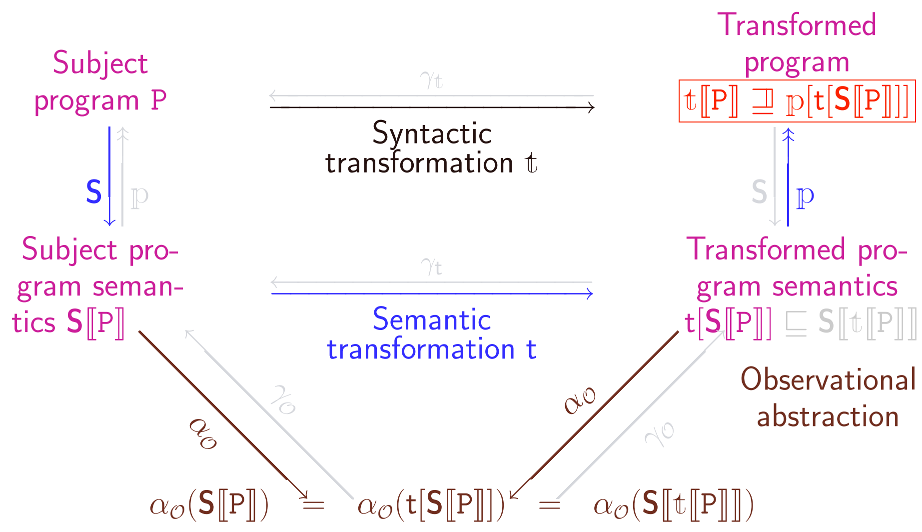
Formalization of Program Transformation Correctness by Abstract Interpretation

Correctness of an Online Program Transformation



Design of Program Transformations by Abstract Interpretation

Design of an Online Program Transformation



Design of Program Transformation Algorithms

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

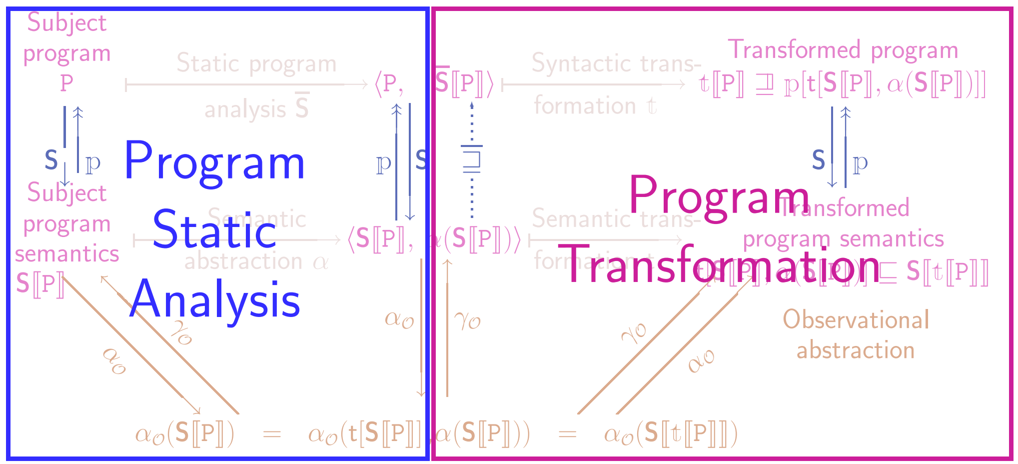
← apply fixpoint transfer /approximation theorems (with widening)

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

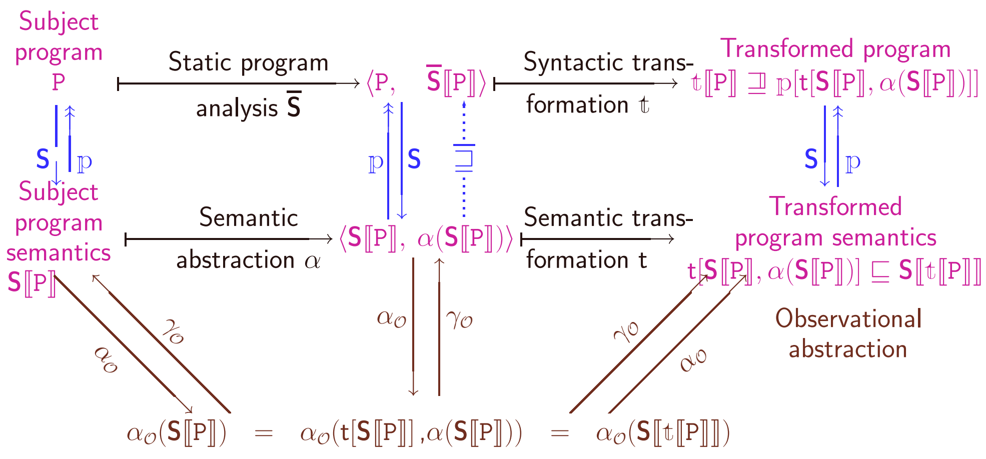
We obtain an iterative program transformation algorithm;
This algorithm is classical or new!

Principle of Offline Program Transformation

Principle of Offline Program Transformation



Principle of Offline Program Transformation



— 45 —

Illustrative Examples

Program Transformations Formalized in the Paper

- Constant propagation;
- Online & offline partial evaluation;
- Mixline partial evaluation (with widening);
- Static program monitoring $\mathsf{S}[\![\mathbb{t}[\![P, M]\!]]\!] = \mathsf{S}[\![P]\!] \cap \mathsf{S}[\![M]\!]$:
  - Example 1: run-time checks elimination ,
  - Example 2: security ,
  - Example 3: proof by transformation ($P \equiv \mathbb{t}[\![P, M]\!]$).

— 47 —

Conclusion

Conclusion

- Program transformation is formalized 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;
- Pave the way to:
  - machine-checked program transformations,
  - a formalization of compilation.