Introduction to Abstract Interpretation

Bruno Blanchet
Département d'Informatique
École Normale Supérieure, Paris
and Max-Planck-Institut für Informatik
Bruno.Blanchet@ens.fr

November 29, 2002

Abstract

We present the basic theory of abstract interpretation, and its application to static program analysis. The goal is not to give an exhaustive view of abstract interpretation, but to give enough background to make papers on abstract interpretation more understandable.

Notations: $\lambda x.M$ denotes the function that maps x to M. $f[x \mapsto M]$ denotes the function f extended so that x is mapped to M. If f was already defined at x, the new binding replaces the old one.

1 Introduction

The goal of static analysis is to determine runtime properties of programs without executing them. (Scheme of an analyzer: Program → Analyzer → Properties.) Here are some examples of properties can be proved by static analysis:

- For optimization:
  - Dead code elimination
  - Constant propagation
  - Live variables
  - Stack allocation (more complex)
- For verification:
  - Absence of runtime errors: division by zero, square root of negative numbers, overflows, array index out of bounds, pointer dereference outside objects, ...
  - Worst-case execution time (see Wilhelm's work with his team and AbsInt)
  - Security properties (secrecy, authenticity of protocols, ...)

However, most interesting program properties are *undecidable*. The basic result to show undecidability is the undecidability of the halting problem.

Proof By contradiction. Assume that there exists Halt(P,E) returning Yes when P terminates on entry E, No otherwise. Consider P'(P) = if Halt(P,P) then loop else stop.

- If P'(P') loops, Halt(P',P') = true, so P'(P') stops: contradiction.
- If P'(P') stops, Halt(P',P') = false, so P'(P') loops: contradiction.

□

Illustration: Give a program that terminates if and only if Fermat's theorem is wrong.

Most program properties can be reduced to the halting problem. So we have to perform approximations.

These approximations must be *sound* (or correct) approximations: if the system gives a definite answer, then this answer is true. But sometimes the system is going to answer "maybe". We say the system is not *complete*. (This notion of approximation is different from numerical approximations that give an answer close to the exact value.)

We want to formalize these approximations, to be able to prove the soundness of the analyses. First, we formalize the meaning of programs (semantics), then its approximations. (Scheme: semantics, exact, concrete properties → approximate, abstract properties.) Abstract interpretation is a **theory of approximation**.

2 A simple language

2.1 Syntax

Expressions: $E ::= x \mid n \mid E + E \mid E - E \mid E * E \mid E/E \mid E \geq E \mid \ldots$

$C ::=$	commands
end	end
$x := E$	assignment
if E *goto* n	conditional
input x	input
print E	print

A program is a function *Prog* from integers to commands. (Indicates which command is at each address in the program.)

2.2 Trace semantics

The state of the system, here is the program counter (pc), which says what is the next command to execute, and the values of the variables (recorded in an environment). We assume that variables take integer values, and we ignore problems of overflow (that is, we compute in $\mathbb{Z}$).

$\rho \in Env = Var \to \mathbb{Z}$
$pc \in PC$
$s \in State = PC \times Env$
Definition of the semantics:

1. Semantics of expressions: $[\![E]\!]\rho \in \mathbb{Z}$. (In case of error, for instance, division by 0, we say that $[\![E]\!]\rho$ is not defined. We could also have a special value $\perp$ to represent that. In the same line, ρ is a partial map, defined only on variables that have been initialized.)

   $[\![x]\!]\rho = \rho(x)$.

   $[\![E_1 + E_2]\!]\rho = [\![E_1]\!]\rho + [\![E_2]\!]\rho$, etc.

2. Semantics of commands: $pc, \rho \to pc', \rho'$

   If $Prog(pc) = x := E$, $pc, \rho \to pc + 1, \rho[x \mapsto [\![E]\!]\rho]$ (if $[\![E]\!]\rho$ is defined)

   If $Prog(pc) = input\ x$, $pc, \rho \to pc + 1, \rho[x \mapsto v]$ for some $v \in \mathbb{Z}$.

   If $Prog(pc) = print\ E$, $pc, \rho \to pc + 1, \rho$, if $[\![E]\!]\rho$ is defined.

   If $Prog(pc) = if\ E\ goto\ n$,
   $pc, \rho \to n, \rho$ if $[\![E]\!]\rho \neq 0$.
   $pc, \rho \to pc + 1, \rho$ if $[\![E]\!]\rho = 0$.

The semantics of commands is a transition system. For a given program, and given user inputs, we can build a sequence of states, such that we go from one state to the next one following one of the above transitions. Such a sequence of states is said to be a *trace*. A program can have several execution traces, depending of the user input, so the semantics of a program is a *set of traces* rather than a single trace.

Give an example of program, and its semantics.

```
1: input x
2: if x <= 0 goto 4
3: print 10/x
4: input y
5: print y*y*x
```

```
1: x := 0;
2: if x >= y goto 5
3: x := x+1;
4: goto 2:
5: print x
```

3 Approximation

3.1 Intuition

We want to prove properties of a program, in the style: "all traces of the program satisfy a given condition". To do this, we *overapproximate* the set of traces, that is, we compute

a superset of the set of traces, possibly in a more efficient representation. (For properties depending only on one state, that is, invariants, we can also simply overapproximate the set of states of the program.)

For instance, if your program uses two variables x and y, you may wish to approximate the set of values of these variables. This can be done using various approximations (show some schemes, like those of P. Cousot).

We also would like to compare analyses: which analysis is more precise than the other ? An analysis is more precise when it yields a smaller superset of the set of traces. The precision yields an ordering on the approximations of the set of traces. All correct approximations are the ones that are greater (in the precision ordering) than the smallest correct one, namely the set of traces itself.

When we represent sets of traces/states by other structures, we have a similar precision ordering on those structures (example: intervals $[a, b] \leq [a', b']$ iff $a \geq a'$ and $b \leq b'$).

3.2 Order, Lattices, complete lattices.

Definition 1 (Partially ordered set) A ***partially ordered set*** (poset) is a set S equipped with a binary relation $\leq$ such that:

1. $\leq$ is reflexive: $\forall a \in S, a \leq a$.
2. $\leq$ is transitive: $\forall a, b, c \in S$, if $a \leq b$ and $b \leq c$ then $a \leq c$.
3. $\leq$ is antisymmetric: $\forall a, b \in S$, if $a \leq b$ and $b \leq a$, then $a = b$.

$c \in S$ is an ***upper bound*** of $X \subseteq S$ if and only if $\forall c' \in X, c' \leq c$.
$c \in S$ is a ***lower bound*** of $X \subseteq S$ if and only if $\forall c' \in X, c \leq c'$.
$c \in S$ is the ***least upper bound*** of $X \subseteq S$ if and only if $\forall c' \in X, c' \leq c$, and $\forall c'' \in S$ such that $\forall c' \in X, c' \leq c''$, we have $c \leq c''$. When it exists, the least upper bound is unique. The ***greatest lower bound*** is similar.

The ordering is not necessarily total, that is, we may not have $a \leq b$ or $b \leq a$. Two elements can be incomparable. (Example: parts of S ordered by inclusion. $\{a\}$ incomparable with $\{b\}$ if $a \neq b$. $\bot = \emptyset$, $\top = S$, $\sqcup = \cup$, $\sqcap = \cap$)

Definition 2 (Lattice) A ***lattice*** is a partially ordered set $L(\leq)$ such that for all a, b in L, a and b have a least upper bound $a \sqcup b$ and a greatest lower bound $a \sqcap b$. This lattice will be denoted by $L(\leq, \sqcup, \sqcap)$.

In a lattice, all finite sets have least upper bounds and greatest lower bounds, but not necessarily infinite sets.

Definition 3 (Complete lattice) A ***complete lattice*** is a partially ordered set $L(\leq)$ such that every subset X of L has a least upper bound $\sqcup X$ and a greatest lower bound $\sqcap X$. In particular, L has a least element $\bot = \sqcup\emptyset$ and a greatest element $\top = \sqcup L$. This lattice will be denoted by $L(\leq, \bot, \top, \sqcup, \sqcap)$.

Proposition 1 *If S is a set, $\mathcal{P}(S)(\subseteq, \emptyset, S, \cup, \cap)$ is a complete lattice.*

We adopt the precision ordering, that is, $a \leq b$ means intuitively that a is more precise than b; a gives more information than b. (Example on intervals $[a, b] \leq [a', b']$ iff $a \geq a'$ and $b \leq b'$. The lattice of intervals is $\{\emptyset\} \cup \{[a, b] \mid a \leq b, a \in \mathbb{Z} \cup \{-\infty\}, b \in \mathbb{Z} \cup \{\infty\}\}$. By a slight abuse of notation, we denote the interval $[a, b]$ even when one bound is infinite. Normally, the bracket is in the other direction for infinite bounds: $]-\infty, +\infty[$.)

3.3 Collecting semantics

The collecting semantics is the semantics that computes the set of possible traces. Formally, we can define it by

$$T_r = \{s_1 \to \cdots \to s_n \mid s_1 \text{ is the initial state}, s_i \to s_{i+1} \text{ is an allowed transition}\}$$

The set of reachable states is defined by

$$S_r = \{s \mid s_1 \to \cdots \to s \in T_r\}$$

The set of all traces, of all states, are ordered by inclusion. Then, they form a complete lattice (the *concrete* lattice).

3.4 Galois connections

A few words on Evariste Galois (1811-1832), French mathematician. He died in a duel, and in the night before, he wrote down much of his theory. He proved that no algebraic formula in radicals exists that gives the general solution of algebraic equations of degree at least 5. This result is based on group theory, of which he is a precursor. He uses so-called "Galois connections" between groups. In the definition below, one of the orderings is reversed with respect to Galois' original definition, so purists say *semi-dual* Galois connections. (The reversal of the ordering gives better properties for the composition of Galois connections, see 4.2 below.)

Definition 4 (Galois connection [10, Definition 5.3.0.1][11, Example 10]) Let $L_1(\leq_1)$ and $L_2(\leq_2)$ be posets. (α, γ) is a ***Galois connection*** (or pair of adjoined functions) between L_1 and L_2 if and only if $\alpha \in L_1 \to L_2$, $\gamma \in L_2 \to L_1$ and

$$\forall x \in L_1, \forall y \in L_2, \alpha(x) \leq_2 y \Leftrightarrow x \leq_1 \gamma(y).$$

This is denoted by:

$$(L_1, \leq_1) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (L_2, \leq_2).$$

$(L_1, \leq_1)$ is the ***concrete*** lattice, $(L_2, \leq_2)$ is the ***abstract*** lattice, α is said to be the ***abstraction*** and γ the ***concretization***. (Scheme)

The two properties $\alpha(x) \leq_2 y$ and $x \leq_1 \gamma(y)$ both mean that y is a correct approximation of the concrete property x. $\alpha(x)$ is the most precise approximation of $x \in L_1$ in L_2. $\gamma(y)$ is the least precise element of L_1 which can be correctly approximated by $y \in L_2$.

3.5 Examples of Galois connections

Examples: abstraction of a set of values into

- constants: The concrete lattice is $\mathcal{P}(S)$ ordered by inclusion, for some set S. The abstract lattice contains $\bot, c, \top$, for each constant $c \in S$, ordered by $\bot \leq c \leq \top$.

  The abstraction is defined by $\alpha(\emptyset) = \bot$, $\alpha(\{c\}) = c$, $\alpha(S) = \top$ otherwise.

  The concretization is defined by $\gamma(\bot) = \emptyset$, $\gamma(c) = \{c\}$, $\gamma(\top) = S$.

- sign. We can use parts of $\{-, 0, +\}$ to represent signs. So for instance a value that may positive or zero is represented by $\{0, +\}$. The concrete lattice is $\mathbb{Z}$, the abstract lattice $\mathcal{P}(\{-, 0, +\})$. Both are ordered by inclusion.

  The abstraction is then defined by

$$\begin{aligned}\alpha(S) = \{+\} \text{ if } S \cap]0, +\infty[\neq \emptyset \\ \cup \{0\} \text{ if } 0 \in S \\ \cup \{-\} \text{ if } S \cap]-\infty, 0[\neq \emptyset\end{aligned}$$

  The concretization is defined by

$$\begin{aligned}\gamma(S^\sharp) =]-\infty, 0[\text{ if } - \in S^\sharp \\ \cup \{0\} \text{ if } 0 \in S^\sharp \\ \cup]0, +\infty[\text{ if } + \in S^\sharp\end{aligned}$$

- interval. The concrete lattice is $\mathcal{P}(\mathbb{Z})$ ordered by inclusion. The abstract lattice is $\{\emptyset\} \cup \{[a, b] \mid a \leq b, a \in \mathbb{Z} \cup \{-\infty\}, b \in \mathbb{Z} \cup \{\infty\}\}$. The ordering is $[a, b] \leq [a', b']$ if and only if $a' \leq a$ and $b \leq b'$ (the ordering corresponds to the inclusion of the corresponding intervals), and $\emptyset \leq [a, b]$.

  The abstraction is defined by: if $S = \emptyset$, $\alpha(S) = \emptyset$, otherwise $\alpha(S) = [\min S, \max S]$).

  The concretization is defined by $\gamma(\emptyset) = \emptyset$, $\gamma([a, b]) = [a, b]$.

- congruence. The concrete lattice is again $\mathcal{P}(\mathbb{Z})$ ordered by inclusion. The abstract lattice contains $\emptyset$ and $(a, b), a \in [0, b-1], b > 0$ means $x \equiv a(b)$, that is $\exists y, x = a + by$.

  The abstraction is defined as follows: $\alpha(\emptyset) = \emptyset$. Take S non empty, $x_0 \in S$, let b be the greatest common divisor of $x - x_0$ for $x \in S$, and $a = x_0 \mod b$, then $\alpha(S) = (a, b)$.

3.6 Properties

Proposition 2 *(α, γ) is a Galois connection if and only if α and γ are monotone, $(\alpha \circ \gamma)(y) \leq_2 y$, and $(\gamma \circ \alpha)(x) \geq_1 x$ [10, Theorem 5.3.0.4].*

Intuitively, α monotone means that the best approximation of a more precise information is also more precise. $(\gamma \circ \alpha)(x) \geq_1 x$ means that by first approximating then concretizing, you have necessarily less information than at the beginning.

Proof First assume that (α, γ) is a Galois connection.

1. $(\alpha \circ \gamma)(y) \leq_2 y$ if and only if $\gamma(y) \leq_1 \gamma(y)$ which is true.
2. $x \leq_1 (\gamma \circ \alpha)(x)$ if and only if $\alpha(x) \leq_2 \alpha(x)$, which is true.
3. α is monotone: let $x \leq_1 x'$. $\alpha(x) \leq_2 \alpha(x')$ if and only if $x \leq_1 \gamma(\alpha(x'))$, which is true, since $x \leq_1 x'$ and $x' \leq_1 (\gamma \circ \alpha)(x')$.
4. γ is monotone: let $y \leq_2 y'$. $\gamma(y) \leq_1 \gamma(y')$ if and only if $\alpha(\gamma(y)) \leq_2 y'$, which is true since $(\alpha \circ \gamma)(y) \leq_2 y$ and $y \leq_2 y'$.

For the converse, assume $\alpha(x) \leq_2 y$. Then $\gamma(\alpha(x)) \leq_1 \gamma(y)$ since γ is monotone. Moreover, $x \leq_1 (\gamma \circ \alpha)(x)$, so $x \leq_1 \gamma(y)$. Conversely, assume $x \leq_1 \gamma(y)$. Then $\alpha(x) \leq_2 \alpha(\gamma(y))$ since α is monotone. Moreover, $(\alpha \circ \gamma)(y) \leq_2 y$, so $\alpha(x) \leq_2 y$. □

Proposition 3 *Let $L_1(\leq_1, \bot_1, \top_1, \sqcup_1, \sqcap_1)$ and $L_2(\leq_2, \bot_2, \top_2, \sqcup_2, \sqcap_2)$ be complete lattices. Let (α, γ) be a Galois connection between L_1 and L_2.*

1. *[10, Corollary 5.3.0.5, (3)]: Each function in the pair (α, γ) uniquely determines the other:*

$$\alpha(x) = \sqcap_2\{y \in L_2 \mid x \leq_1 \gamma(y)\},$$
$$\gamma(y) = \sqcup_1\{x \in L_1 \mid \alpha(x) \leq_2 y\}.$$

2. *[10, Corollary 5.3.0.5, (4)] α is a complete join-morphism (i.e. is additive: $\alpha(\sqcup S) = \sqcup\{\alpha(x) \mid x \in S)$, $\alpha(\bot_1) = \bot_2$.*

   γ is a complete meet-morphism (i.e. $\gamma(\sqcap S) = \sqcap\{\gamma(x) \mid x \in S\}$), $\gamma(\top_2) = \top_1$.

Proof 1. If $x \leq_1 \gamma(y)$, then $\alpha(x) \leq y$, so $\alpha(x) \leq \sqcap_2\{y \in L_2 \mid x \leq_1 \gamma(y)\}$. Moreover, $\alpha(x) \in L_2$ and $x \leq_1 \gamma(\alpha(x))$, so $\alpha(x) \in \{y \in L_2 \mid x \leq_1 \gamma(y)\}$, so $\alpha(x) \geq \sqcap_2\{y \in L_2 \mid x \leq_1 \gamma(y)\}$, hence $\alpha(x) = \sqcap_2\{y \in L_2 \mid x \leq_1 \gamma(y)\}$.

The proof is similar for γ.

2. Let $X \subseteq L_1$. Let us show that $\alpha(\sqcup_1 X) = \sqcup_2\{\alpha(x) \mid x \in X\}$.

   a) $\sqcup_2\{\alpha(x) \mid x \in X\} \leq \alpha(\sqcup_1 X)$ comes from the monotony of α.

   Precisely, for all $x \in X$, $x \leq_1 \sqcup_1 X$, so $\alpha(x) \leq_2 \alpha(\sqcup_1 X)$ since α is monotone. So $\sqcup_2\{\alpha(x) \mid x \in X\} \leq \alpha(\sqcup_1 X)$.

   b) To show $\alpha(\sqcup_1 X) \leq_2 \sqcup_2\{\alpha(x) \mid x \in X\}$, we show $\sqcup_1 X \leq_1 \gamma(\sqcup_2\{\alpha(x) \mid x \in X\})$. Let $x \in X$. $\alpha(x) \leq_2 \sqcup_2\{\alpha(x) \mid x \in X\}$, so $x \leq_1 \leq_1 \gamma(\sqcup_2\{\alpha(x) \mid x \in X\})$ by the definition of Galois connections. Then $\sqcup_1 X \leq_1 \gamma(\sqcup_2\{\alpha(x) \mid x \in X\})$, hence the result.

   We have $\alpha(\sqcup_1 \emptyset) = \sqcup_2 \emptyset$, so $\alpha(\bot_1) = \bot_2$.

   The proof is similar for γ.

□

Proposition 4 *Let $L_1(\leq_1, \bot_1, \top_1, \sqcup_1, \sqcap_1)$ and $L_2(\leq_2, \bot_2, \top_2, \sqcup_2, \sqcap_2)$ be complete lattices.*

If $\alpha : L_1 \to L_2$ is a complete join-morphism and $\gamma(y) = \sqcup_1\{x \in L_1 \mid \alpha(x) \leq_2 y\}$, then (α, γ) is a Galois connection.

If $\gamma : L_2 \to L_1$ is a complete meet-morphism and $\alpha(x) = \sqcap_2\{y \in L_2 \mid x \leq_1 \gamma(y)\}$, then (α, γ) is a Galois connection.

Proof

- First point: Let $\alpha : L_1 \to L_2$ be a complete join-morphism and $\gamma(y) = \sqcup_1\{x \in L_1 \mid \alpha(x) \leq_2 y\}$.

  a) If $\alpha(x) \leq y$, then $x \in \{x' \in L_1 \mid \alpha(x') \leq_2 y\}$, so $x \leq \gamma(y)$.

  b) If $x \leq \gamma(y)$, $x \leq \sqcup_1\{x' \in L_1 \mid \alpha(x') \leq_2 y\}$, so $\alpha(x) \leq \sqcup_1\{\alpha(x') \in L_1 \mid \alpha(x') \leq_2 y\}$ since α is a complete join-morphism, so $\alpha(x) \leq y$.

  Then (α, γ) is a Galois connection.

- Second point: Let $\gamma : L_2 \to L_1$ be a complete meet-morphism and $\alpha(x) = \sqcap_2\{y \in L_2 \mid x \leq_1 \gamma(y)\}$.

  a) If $\alpha(x) \leq y$, then $\sqcap_2\{y' \in L_2 \mid x \leq_1 \gamma(y')\} \leq y$, so $\sqcap_2\{\gamma(y') \in L_2 \mid x \leq_1 \gamma(y')\} \leq \gamma(y)$, since γ is a complete meet-morphism, so $x \leq \gamma(y)$.

  b) If $x \leq \gamma(y)$, $y \in \{y' \in L_2 \mid x \leq_1 \gamma(y')\}$, so $\alpha(x) \leq y$.

  Then (α, γ) is a Galois connection.

□

Proposition 5 *[10, Theorem 5.3.0.6, (1)] α is onto if and only if γ is one-to-one if and only if $\alpha \circ \gamma = \lambda y.y$.*

Proof If $\alpha \circ \gamma = \mathrm{Id}$, then α is onto (for all $y \in L_2$, $y = \alpha(\gamma(y))$) and γ is one-to-one (if $\gamma(y) = \gamma(y')$ then $y = \alpha(\gamma(y)) = \alpha(\gamma(y')) = y'$). This is a general property of functions, that is not related to Galois connections.

If α is onto, let x' such that $\alpha(x') = y$. Then $x' \leq_1 \gamma(y)$, so $y = \alpha(x') \leq_2 \alpha(\gamma(y))$. In general, we have $\alpha(\gamma(y)) \leq_2 y$, so $y = \alpha(\gamma(y))$.

Assume that γ is one-to-one. We have $\gamma \circ \alpha \circ \gamma(x) \geq_1 \gamma(x)$ since $\gamma \circ \alpha \geq_1 \mathrm{Id}$, and $\gamma \circ \alpha \circ \gamma(x) \leq_1 \gamma(x)$ since $\alpha \circ \gamma \leq_2 \mathrm{Id}$ and γ is monotone. So $\gamma \circ \alpha \circ \gamma(x) = \gamma(x)$. Since γ is one-to-one, we have $\alpha \circ \gamma(x) = x$. □

The property $\alpha \circ \gamma = \lambda y.y$ means intuitively that the concrete properties of L_1 are more precise than the abstract properties of L_2 (every abstract property has a corresponding concrete property which has exactly the same meaning, whereas in general one can only find an approximate abstract property $\alpha(x)$ corresponding to a concrete property x). This property can be enforced by replacing L_2 by $\alpha(L_1)$.

Remark Let $\sigma(y) = \sqcap\{y' \mid \gamma(y') = \gamma(y)\}$. Then $\alpha(L_1) = \sigma(L_2)$.

Proof First, for all $y'' \in \sigma(L_2)$, $\alpha \circ \gamma(y'') = y''$. Indeed, let $y'' = \sigma(y)$. We always have $\alpha \circ \gamma(y'') \leq y''$. Conversely, $\gamma \circ \alpha \circ \gamma(y'') \geq \gamma(y'')$, since $\gamma \circ \alpha(x) \geq x$, and $\gamma \circ \alpha \circ \gamma(y'') \leq \gamma(y'')$

since $\alpha \circ \gamma(y'') = y''$. So $\gamma \circ \alpha \circ \gamma(y'') = \gamma(y'') = \gamma(y)$. So $\alpha \circ \gamma(y'') \in \{y' \mid \gamma(y') = \gamma(y)\}$, so $\alpha \circ \gamma(y'') \geq \sigma(y) = y''$.

Take $y'' \in \sigma(L_2)$. $\alpha \circ \gamma(y'') = y''$, so $y'' \in \alpha(L_1)$.

Second, for all $y'' \in \alpha(L_1)$, $\sigma(y'') = y''$. Indeed, let $y'' = \alpha(x)$. We always have $\sigma(y'') \leq y''$. Conversely, if $\gamma(y') = \gamma(y'')$, then $\gamma(y') = \gamma \circ \alpha(x) \geq x$, so $y' \geq \alpha(x) = y''$. Then $\sigma(y'') = \sqcap\{y' \mid \gamma(y') = \gamma(y'')\} \geq y''$. So $\sigma(y'') = y''$.

Take $y'' \in \alpha(L_1)$. $\sigma(y'') = y''$, so $y'' \in \sigma(L_2)$. □

The design of an analysis is then defining α or γ and computing for each construct of the analyzed language the image of its semantics by the abstraction α.

3.7 Computation of the abstract semantics

The abstract semantics can be systematically computed from the concrete semantics and the Galois connection. (This an important advantage of abstract interpretation.)

Let us consider for instance the small language defined at the beginning of the course. We start from the semantics of this language, lift it to sets to obtain the *collecting semantics*, and abstract the collecting semantics (lifting to sets ordering by inclusion is important to have lattices).

	Semantics	Collecting semantics	Abstraction	Abstract semantics
Values	$\mathbb{Z}$	$\mathcal{P}(\mathbb{Z}), \subseteq$	$\underset{\alpha}{\overset{\gamma}{\leftrightarrows}}$	$L, \leq$
Operators	$+ : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$	$+ : \mathcal{P}(\mathbb{Z}) \times \mathcal{P}(\mathbb{Z}) \to \mathcal{P}(\mathbb{Z})$		$a^\sharp +^\sharp b^\sharp = \alpha(\gamma(a^\sharp) + \gamma(b^\sharp))$

The correctness of the abstraction $x^\sharp$ of a value x is $\alpha(x) \leq x^\sharp$ (which is equivalent to $x \leq \gamma(x^\sharp)$ by definition of the Galois connection).

The correctness of the abstraction of an operator, such a $+$, is defined by taking the above correctness as an invariant. That is, $+^\sharp$ is a correct abstraction of $+$ if and only if, if $\alpha(a) \leq a^\sharp$ and $\alpha(b) \leq b^\sharp$, then $\alpha(a + b) \leq a^\sharp +^\sharp b^\sharp$. We can show that $+^\sharp$ defined by $a^\sharp +^\sharp b^\sharp = \alpha(\gamma(a^\sharp) + \gamma(b^\sharp))$ satisfies this condition. It is in fact the best possible abstraction of $+$ (the smallest that satisfies this condition). All operators can be abstracted in this way by a systematic computation. (See also Proposition 10 below.)

	Semantics	Collecting semantics	Abstraction	Abstract semantics
Environments	$\rho : \mathit{Var} \to \mathbb{Z}$	$R : \mathcal{P}(\mathit{Var} \to \mathbb{Z}), \subseteq$	$\underset{\alpha_R}{\overset{\gamma_R}{\leftrightarrows}}$	$R^\sharp : \mathit{Var} \to L, \leq$

The abstraction of environments can be defined in two steps:

- First we abstract sets of environments to mappings from variables to sets of integers:

$$\mathcal{P}(\mathit{Var} \to \mathbb{Z}) \underset{\alpha_{R1}}{\overset{\gamma_{R1}}{\leftrightarrows}} \mathit{Var} \to \mathcal{P}(\mathbb{Z})$$

  The abstraction is defined by $\alpha_{R1}(R_1) = \lambda x.\{\rho(x) \mid \rho \in R_1\}$.

  The concretization is defined by $\gamma_{R1}(R_1^\sharp) = \{\lambda x.y \mid y \in R_1^\sharp(x)\}$.

This abstraction corresponds to losing relations between variables. Only the set of possible values of each variable is kept. (This is not the only possible way to do an analysis, but the simplest one. Some analyses keep relations between variables. For these analyses, we cannot use this abstraction; we must define another abstraction of environments into an abstract lattice, which is then not of the form $Var \to L$.)

- In the obtained mapping, we then abstract each result into L by (α, γ).

$$Var \to \mathcal{P}(\mathbb{Z}) \underset{\alpha_{R2}}{\overset{\gamma_{R2}}{\leftrightarrows}} Var \to L$$

The Galois connection $(\alpha_{R2}, \gamma_{R2})$ is obtained by lifting (α, γ) to functions (see Proposition 8 below).

The abstraction is defined by $\alpha_{R2}(R_2) = \lambda x.\alpha(R_2(x))$.

The concretization is defined by $\gamma_{R1}(R_2^\sharp) = \lambda x.\gamma(R_2^\sharp(x))$.

The Galois connection

$$\mathcal{P}(\, Var \to \mathbb{Z}) \underset{\alpha_R}{\overset{\gamma_R}{\leftrightarrows}} Var \to L$$

is then obtained by composing the two Galois connections: $\alpha_R = \alpha_{R2} \circ \alpha_{R1}$ and $\gamma_R = \gamma_{R1} \circ \gamma_{R2}$. (see Proposition 12).

We can now abstract expressions:

Semantics	Collecting semantics	Abstr.	Abstract semantics
$[\![E]\!]\rho \in \mathbb{Z}$	$[\![E]\!]R \in \mathcal{P}(\mathbb{Z})$		$[\![E]\!]^\sharp R^\sharp \in L$
$[\![x]\!]\rho = \rho(x)$			$[\![x]\!]^\sharp R^\sharp = R^\sharp(x)$
$[\![E_1 + E_2]\!]\rho = [\![E_1]\!]\rho + [\![E_2]\!]\rho$			$[\![E_1 + E_2]\!]^\sharp R^\sharp = [\![E_1]\!]^\sharp R^\sharp +^\sharp [\![E_2]\!]^\sharp R^\sharp$

$[\![E]\!]R$ is the set of evaluation of E for all environments in R, that is $[\![E]\!]R = \{[\![E]\!]\rho \mid \rho \in R\}$.

The correctness of the abstract semantics of expressions is expressed by: if the environment is correctly abstracted, then the result of the expression is correctly abstracted. Formally: if $\alpha_R(R) \leq R^\sharp$ then $\alpha([\![E]\!]R) \leq [\![E]\!]^\sharp R^\sharp$.

Proof of correctness of the abstract semantics of expressions: This proof is by induction on expressions. We handle the cases variable and $+$.

- $\alpha([\![x]\!]R) = \alpha(\{\rho(x) \mid x \in R\}) = \alpha_R(R)(x) \leq R^\sharp(x)$ by correctness of $R^\sharp$.
- We have $[\![E_1 + E_2]\!]R = \{[\![E_1]\!]\rho + [\![E_2]\!]\rho \mid \rho \in R\} \subseteq [\![E_1]\!]R + [\![E_2]\!]R$. (We have only an inclusion, because on the left of the inclusion, both E_1 and E_2 must be evaluated in the same environment, whereas on the right they can be evaluated in different environments that both belong to R.)

  Then $\alpha([\![E_1 + E_2]\!]R) \geq \alpha([\![E_1]\!]R + [\![E_1]\!]R) \geq \alpha(\gamma([\![E_1]\!]^\sharp R^\sharp) + \gamma([\![E_2]\!]^\sharp R^\sharp))$ (by correctness on E_1 and E_2 by induction hypothesis). So $\alpha([\![E_1 + E_2]\!]R) \geq [\![E_1]\!]^\sharp R^\sharp +^\sharp [\![E_2]\!]^\sharp R^\sharp$.

One can notice that the abstract semantics of expressions is simply obtained by replacing each concrete operator with its corresponding abstract operator (for example, $+$ is replaced with $+^\sharp$).

The computation of the abstract trace semantics follows.
If $Prog(pc) = x := E,\ pc, R^\sharp \to pc+1, R^\sharp[x \mapsto [\![E]\!]^\sharp R^\sharp]$
If $Prog(pc) = input\ x,\ pc, R^\sharp \to pc+1, R^\sharp[x \mapsto \top]$.
If $Prog(pc) = print\ E,\ pc, R^\sharp \to pc+1, R^\sharp$
If $Prog(pc) = if\ E\ goto\ n$,
$pc, R^\sharp \to n, R^\sharp$.
$pc, R^\sharp \to pc+1, R^\sharp$.

We have a non-deterministic branch for the test. For some (simple) conditions, we can have a better analysis of tests. For instance, considering interval analysis:
If $Prog(pc) = if\ x > 0\ goto\ n$,
$pc, R^\sharp \to n, R^\sharp[x \mapsto R^\sharp(x) \sqcap [1, +\infty[\ ]$.
$pc, R^\sharp \to pc+1, R^\sharp[x \mapsto R^\sharp(x) \sqcap]-\infty, 0]\ ]$.

Indeed, when the test is true, we are sure that x is positive, and when the test is false, we are sure that x is negative or 0. This idea can be generalized to more complex expressions than just $x > 0$, by considering a backward analysis (that is, from the result of the test, we determine possible values for the variables contained in E). We are not going to detail this point here.

The correctness of the abstract trace semantics is that if $pc_1, \rho_1 \to pc_2, \rho_2$, and $\alpha_R(\{\rho_1\}) \leq R_1^\sharp$, then there exists $R_2^\sharp$, such that $pc_1, R_1^\sharp \to pc_2, R_2^\sharp$ and $\alpha_R(\{\rho_2\}) \leq R_2^\sharp$. That is intuitively, if a state is correctly abstracted, then the next state is also correctly abstracted.

We now abstract sets of states.

Semantics	Collecting semantics	Abstr.	Abstract semantics
$s : PC \times (Var \to \mathbb{Z})$	$S : \mathcal{P}(PC \times (Var \to \mathbb{Z}))$	$\underset{\alpha_S}{\overset{\gamma_S}{\leftrightarrows}}$	$S^\sharp : PC \to (\{\bot\} \cup (Var \to L))$

The abstraction is defined by $\alpha_S(S) = \lambda pc.\alpha'_R(\{\rho \mid (pc, \rho) \in S\})$ where $\alpha'_R(R) = \alpha_R(R)$ when R is not empty, and $\alpha'_R(\emptyset) = \bot$. Defining $\gamma'_R(\bot) = \emptyset$ and $\gamma'_R(R^\sharp) = \gamma_R(R^\sharp)$ when $R^\sharp \neq \bot$, we in fact get another Galois connection for sets of environments. (We could also consider abstract states as partial maps, that are not defined at pc when no concrete state exists at that pc. It is perhaps clearer to have an explicit value $\bot$ for this situation.)

The concretization is defined by $\gamma_S(S^\sharp) = \{(pc, \rho) \mid \rho \in \gamma'_R(S^\sharp(pc))\}$.

The abstract semantics has one abstract environment for each program point, giving the possible values of variables at that program point.

We define the function $post$ that, from a set of states, returns the set of next states. We have $post(S) = \{s' \mid s \in S, s \to s'\}$. We can now compute the corresponding abstract function $post^\sharp$ such that $post^\sharp \geq \alpha_S \circ post \circ \gamma_S$, that transforms an abstract state into the next abstract state, that represents the possible next states.

$$\alpha_S \circ post \circ \gamma_S(S^\sharp) = \lambda pc_2.\alpha'_R(\{\rho_2 \mid \rho_1 \in \gamma'_R(S^\sharp(pc_1)), (pc_1, \rho_1) \to (pc_2, \rho_2)\})$$

We have $(pc_1, \rho_1) \rightarrow (pc_2, \rho_2)$ and $\alpha_R(\{\rho_1\}) \leq S^\sharp(pc_1)$, so by correctness of the abstract transition relation, there exists $R_2^\sharp$ such that $(pc_1, S^\sharp(pc_1)) \rightarrow (pc_2, R_2^\sharp)$ and $\alpha_R(\{\rho_2\}) \leq R_2^\sharp$. So

$$\alpha_S \circ post \circ \gamma_S(S^\sharp) \leq \lambda pc_2. \sqcup \{R_2^\sharp \mid \exists pc_1, (pc_1, S^\sharp(pc_1)) \rightarrow (pc_2, R_2^\sharp)\}$$

We take

$$post^\sharp(S^\sharp) = \lambda pc_2. \sqcup \{R_2^\sharp \mid \exists pc_1, (pc_1, S^\sharp(pc_1)) \rightarrow (pc_2, R_2^\sharp)\}$$

The upper bound corresponds to the case when we reach the same program point pc_2 from several program points pc_1. In this case, we have to compute an abstract environment at pc_2 that is correct with respect to (that is, greater than, in the precision ordering) all abstract environments coming from the various program points pc_1. The upper bound does exactly that.

Section 5 explain how we can compute a correct abstraction of the set of reachable states from the function $post^\sharp$.

Example: if we consider the example of interval analysis, we have

- $[a, b] +^\sharp [a', b'] = [a + a', b + b']$, $\emptyset +^\sharp x^\sharp = x^\sharp +^\sharp \emptyset = \emptyset$.
- $[a, b] -^\sharp [a', b'] = [a - b', a' - b]$, $\emptyset -^\sharp x^\sharp = x^\sharp -^\sharp \emptyset = \emptyset$.

The computation of other abstract operations is left to the reader. (For product and division, note that it is a bit more complicated because of signs.) We also leave to the reader the special cases when one bound of the interval is infinite.

For the example of program

```
1: input x
2: if x <= 0 goto 4
3: print 10/x
4: input y
5: print y*y*x
```

the abstract transition relation is

$$\begin{aligned}
&1, R^\sharp \rightarrow 2, R^\sharp[x \mapsto]-\infty, +\infty[\ ] \\
&2, R^\sharp \rightarrow 4, R^\sharp[x \mapsto R^\sharp(x) \sqcap]-\infty, 0]\ ] \\
&2, R^\sharp \rightarrow 3, R^\sharp[x \mapsto R^\sharp(x) \sqcap [1, +\infty[\ ] \\
&3, R^\sharp \rightarrow 4, R^\sharp \\
&4, R^\sharp \rightarrow 5, R^\sharp[y \mapsto]-\infty, +\infty[\ ] \\
&5, R^\sharp \rightarrow 6, R^\sharp
\end{aligned}$$

4 Combinations of abstractions

4.1 Complex domains

4.1.1 Cardinal product

Proposition 6 *Let L_i, L'_i $(i \in I)$ be complete lattices. If (α_i, γ_i) is a Galois connection,*

$$(L_i, \leq_i) \underset{\alpha_i}{\overset{\gamma_i}{\rightleftarrows}} (L'_i, \leq'_i)$$

then we have a Galois connection:

$$(\times_{i \in I} L_i, \leq) \underset{\alpha}{\overset{\gamma}{\rightleftarrows}} (\times_{i \in I} L'_i, \leq')$$

where $a \leq b$ if and only if for all $i \in I$, $a_i \leq_i b_i$ (similarly for $\leq'$), $\alpha((x_i)_{i \in I}) = (\alpha_i(x_i))_{i \in I}$, and $\gamma((y_i)_{i \in I}) = (\gamma(y_i))_{i \in I}$.

4.1.2 Sets

Proposition 7 *Let L be a complete lattice, and S a set. Let $f : S \to L$ be a function. Then we have a Galois connection:*

$$(\mathcal{P}(S), \subseteq) \underset{\alpha}{\overset{\gamma}{\rightleftarrows}} (L, \leq)$$

where $\alpha(x) = \sqcup\{f(x') \mid x' \in x\}$ and $\gamma(y) = \{x' \mid f(x') \leq y\}$.

Proof $\alpha(x) \leq y$ if and only if for all $x' \in x$, $f(x') \leq y$ if and only if $x \subseteq \gamma(y)$. □

This is what we have done when lifting the abstraction to the collecting semantics.
Example: for signs, $f(x) = -$ if $x < 0$, $f(x) = 0$ if $x = 0$ and $f(x) = +$ if $x > 0$.

4.1.3 Functions with fixed input

Proposition 8 *Let L and L' be complete lattices, and S a set. If (α_i, γ_i) is a Galois connection,*

$$(L, \leq) \underset{\alpha}{\overset{\gamma}{\rightleftarrows}} (L', \leq')$$

then we have a Galois connection:

$$(S \to L, \leq) \underset{\alpha'}{\overset{\gamma'}{\rightleftarrows}} (S \to L', \leq')$$

where the function lattices are ordered pointwise: $f \leq f'$ if and only if $\forall x \in S, f(x) \leq f'(x)$, $\alpha'(f) = \lambda x.\alpha(f(x))$, and $\gamma'(f) = \lambda x.\gamma(f(x))$.

This is a particular case of the cardinal product. This is what we have done for the environment in our small language.

Proposition 9 *Let L and L' be complete lattices, and S a set. If (α_i, γ_i) is a Galois connection,*

$$(L, \leq) \underset{\alpha}{\overset{\gamma}{\rightleftarrows}} (L', \leq')$$

then we have a Galois connection:

$$(S \to L, \leq) \underset{\alpha'}{\overset{\gamma'}{\rightleftarrows}} (L', \leq')$$

where the function lattices are ordered pointwise: $f \leq f'$ if and only if $\forall x \in S, f(x) \leq f'(x)$, $\alpha'(f) = \sqcup\{\alpha(f(x)) \mid x \in S\}$, and $\gamma'(y) = \{f \mid \forall x \in S, f(x) \leq \gamma(y)\}$.

This is a much coarser approximation.

Application to arrays: An array corresponds to a function that associates to each index the element of the array at that index. The elements of an array can be analyzed either with one abstract value for each element of the array (this is the first abstraction), or with one abstract element for the whole array (this is the second abstraction).

4.1.4 Functions with abstracted input; higher-order abstract interpretation

Proposition 10 *1. Let $L_1(\leq_1, \bot_1, \top_1, \sqcup_1, \sqcap_1)$ and $L_2(\leq_2, \bot_2, \top_2, \sqcup_2, \sqcap_2)$ be complete lattices. The set of monotone functions $(L_1 \overset{\text{m}}{\to} L_2)(\leq_\text{f}, \bot_\text{f}, \top_\text{f}, \sqcup_\text{f}, \sqcap_\text{f})$ ordered pointwise is a complete lattice.*

$$\begin{array}{ll} \phi_1 \leq_\text{f} \phi_2 \Leftrightarrow \forall x \in X, \phi_1(x) \leq \phi_2(x), & \\ \bot_\text{f} = \lambda x.\bot, & \top_\text{f} = \lambda x.\top, \\ \sqcup_\text{f} F = \lambda x. \sqcup \{f(x) \mid f \in F\}, & \sqcap_\text{f} F = \lambda x. \sqcap \{f(x) \mid f \in F\}. \end{array}$$

*2. Let $L_1(\leq_1, \bot_1, \top_1, \sqcup_1, \sqcap_1)$ and $L_2(\leq_2, \bot_2, \top_2, \sqcup_2, \sqcap_2)$ be complete lattices. A function $f : L_1 \to L_2$ is **additive** if and only if for all $S \subseteq L_1$, $f(\sqcup S) = \sqcup\{f(x) \mid x \in S\}$. The set of additive functions $(L_1 \overset{\text{a}}{\to} L_2)(\leq_\text{f}, \bot_\text{f}, \top_\text{f}, \sqcup_\text{f}, \sqcap_\text{f})$ ordered pointwise is a complete lattice.*

$$\begin{array}{ll} \phi_1 \leq_\text{f} \phi_2 \Leftrightarrow \forall x \in X, \phi_1(x) \leq \phi_2(x), & \\ \bot_\text{f} = \lambda x.\bot, & \top_\text{f} = \lambda x.\top, \\ \sqcup_\text{f} F = \lambda x. \sqcup \{f(x) \mid f \in F\}, & \sqcap_\text{f} F = \sqcup_\text{f}\{g \mid \forall f \in F, g \leq_\text{f} f\}. \end{array}$$

Proposition 11 *[11, (17)] Let L_1, L'_1, L_2, L'_2 be complete lattices. If (α, γ) is a Galois connection,*

$$(L_1, \leq_1) \underset{\alpha}{\overset{\gamma}{\rightleftarrows}} (L_2, \leq_2)$$

and (α', γ') is a Galois connection,

$$(L'_1, \leq'_1) \underset{\alpha'}{\overset{\gamma'}{\rightleftarrows}} (L'_2, \leq'_2)$$

then we have a Galois connection:

$$(L_1 \stackrel{\mathrm{m}}{\to} L'_1, \leq''_1) \underset{\alpha''}{\overset{\gamma''}{\leftrightarrows}} (L_2 \stackrel{\mathrm{m}}{\to} L'_2, \leq''_2)$$

If γ and γ' are additive,

$$(L_1 \stackrel{\mathrm{a}}{\to} L'_1, \leq''_1) \underset{\alpha''}{\overset{\gamma''}{\leftrightarrows}} (L_2 \stackrel{\mathrm{a}}{\to} L'_2, \leq''_2)$$

where the function lattices are ordered pointwise, $\alpha''(f) = \alpha' \circ f \circ \gamma$ and $\gamma''(f^\sharp) = \gamma' \circ f^\sharp \circ \alpha$.

If $\alpha \circ \gamma = \mathrm{Id}$ and $\alpha' \circ \gamma' = \mathrm{Id}$ then $\alpha'' \circ \gamma'' = \mathrm{Id}$.

Proof that this is a Galois connection.

- Assume that $\alpha''(f) \leq f^\sharp$. Then for all $x^\sharp$, $\alpha' \circ f \circ \gamma(x^\sharp) \leq f^\sharp(x^\sharp)$. So, by the definition of Galois connections, $f \circ \gamma(x^\sharp) \leq \gamma' \circ f^\sharp(x^\sharp)$.

  Then, for all x, $\gamma''(f^\sharp)(x) = \gamma' \circ f^\sharp \circ \alpha(x) \geq f \circ \gamma \circ \alpha(x)$ by the above inequality applied to $x^\sharp = \alpha(x)$. Therefore, $\gamma''(f^\sharp)(x) \geq f(x)$ since $\gamma \circ \alpha \geq \mathrm{Id}$ and f is monotone.

- Conversely, assume that $\gamma''(f^\sharp) \geq f$. Then for all x, $\gamma' \circ f^\sharp \circ \alpha(x) \geq f(x)$. So, by the definition of Galois connections, $f^\sharp \circ \alpha(x) \geq \alpha' \circ f(x)$.

  Then, for all $x^\sharp$, $\alpha''(f)(x^\sharp) = \alpha' \circ f \circ \gamma(x^\sharp) \leq f^\sharp \circ \alpha \circ \gamma(x^\sharp)$ by the above inequality applied to $x = \gamma(x^\sharp)$. Therefore, $\alpha''(f)(x^\sharp) \leq f^\sharp(x^\sharp)$ since $\alpha \circ \gamma \leq \mathrm{Id}$ and $f^\sharp$ is monotone.

Note that α'' maps really monotone functions to monotone functions (if f is monotone, $\alpha' \circ f \circ \gamma$ is monotone, since α' and γ are monotone). We have the same property for γ''. We also have the same properties for additivity, when γ and γ' are additive. (Remember that abstraction functions are always additive.)

We leave the proof that $\alpha'' \circ \gamma'' = \mathrm{Id}$ to the reader. □

This is a key result to abstract semantic operators. It can naturally be extended to functions with several parameters.

Example: abstraction of + for the sign lattice.

4.2 Composition of abstractions; hierarchy of analyses

Proposition 12 *Let L_1, L_2, L_3 be complete lattices. If (α, γ) is a Galois connection,*

$$(L_1, \leq_1) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (L_2, \leq_2)$$

and (α', γ') is a Galois connection,

$$(L_2, \leq_2) \underset{\alpha'}{\overset{\gamma'}{\leftrightarrows}} (L_3, \leq_3)$$

then we have a Galois connection:

$$(L_1, \leq_1) \underset{\alpha' \circ \alpha}{\overset{\gamma \circ \gamma'}{\leftrightarrows}} (L_3, \leq_3)$$

If $\alpha \circ \gamma = \mathrm{Id}$ and $\alpha' \circ \gamma' = \mathrm{Id}$ then $(\alpha' \circ \alpha) \circ (\gamma \circ \gamma') = \mathrm{Id}$.

Example: signs as abstraction of intervals.

4.3 Composition of analyses

4.3.1 Reduced cardinal product

You have two analyses of the same variable/program, and you want to do both of them. In some cases, doing both analyses together can result in a precise information than doing each analysis separately.

Example: intervals and congruences. The congruence information can reduce the interval.

This is formalized by the reduced cardinal product of the analyses.

Definition 5 Let

$$(L, \leq) \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} (L_1, \leq_1)$$

and

$$(L, \leq) \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} (L_2, \leq_2)$$

be two Galois connections. Their ***reduced cardinal product*** is

$$(L, \leq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\sigma(L_1 \times L_2), \leq)$$

where $(a_1, a_2) \leq (b_1, b_2)$ if and only if $a_1 \leq_1 b_1$ and $a_2 \leq_2 b_2$, $\sigma : L_1 \times L_2 \to L_1 \times L_2$ is defined by $\alpha(x) = (\alpha_1(x), \alpha_2(x))$, $\gamma(a_1, a_2) = \gamma_1(a_1) \sqcap \gamma_2(a_2)$, and $\sigma(a_1, a_2) = \sqcap\{(b_1, b_2) \mid \gamma(a_1, a_2) = \gamma(b_1, b_2)\}$.

Proof that the reduced cardinal product is a Galois connection.

$x \leq \gamma(a_1, a_2)$ if and only if $x \leq \gamma_1(a_1)$ and $x \leq \gamma_2(a_2)$, if and only if $\alpha_1(x) \leq a_1$ and $\alpha_2(x) \leq a_2$ if and only if $\alpha(x) \leq (a_1, a_2)$. □

Remark: σ is the ***reduction***: it combines the informations of both analyses. It is used to enforce the third property of 3, since it is in general not true after the cardinal product has been computed.

When we compute the transformers for the product, we use $F^\sharp = \alpha \circ F \circ \gamma$, which is more precise than the independent computation $\lambda(a_1, a_2).(\alpha_1 \circ F \circ \gamma_1(a_1), \alpha_2 \circ F \circ \gamma_2(a_2))$. Another way of improving the precision in a combination of analyses is to compute the analyses independently, and apply the reduction σ after each computation. Then, we also work in the lattice $\sigma(L_1 \times L_2)$, but the best precision is not always achieved (but the design is simpler: we can use the transfer functions for the independent analyses instead of recomputing transfer functions for the combined analyses).

Example: see [10] for a program in which the reduced product of signs and parity is more precise than the independent analysis.

4.3.2 Reduced cardinal power

The reduced cardinal power takes into account relations between two analyses.

Definition 6 Let

$$(L, \leq) \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} (L_1, \leq_1)$$

and

$$(L, \leq) \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} (L_2, \leq_2)$$

be two Galois connections. Their ***reduced cardinal power*** is

$$(L, \leq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\sigma(L_1 \overset{\text{m}}{\to} L_2), \leq)$$

where $f \leq f'$ if and only if $\forall y_1 \in L_1, f(y_1) \leq f'(y_1)$, $\alpha(x) = \lambda y_1.\alpha_2(x \sqcap \gamma_1(y_1))$, $\gamma(f) = \sqcup\{x \in L \mid \forall y_1 \in L_1, \gamma_1(y_1) \sqcap x \leq \gamma_2(f(y_1))\}$, and $\sigma(f) = \sqcap\{f' \mid \gamma(f) = \gamma(f')\}$.

Proof that the reduced cardinal power is a Galois connection.

α is a complete join-morphism (since α_2 and $\lambda x.x \sqcap \gamma_1(y_1)$ are), and $\gamma(y) = \sqcup\{x \in L \mid \alpha(x) \leq y\}$, so by Proposition 4, (α, γ) is a Galois connection. □

Example: see [10].

5 Loops

Example of program with loop (see end of semantics section). Try to analyze it, and show that we need a fixpoint.

5.1 Collecting semantics; fixpoints

A fixpoint of a function f is an element x such that $f(x) = x$. The least fixpoint of f is denoted by $\text{lfp}(f)$ and its greatest fixpoint by $\text{gfp}(f)$. The collecting semantics can also be defined by fixpoints.

The set of reachable states can be computed from the function *post*. Formally, let $F_s(S) = S_0 \cup post(S)$, where S_0 is the set of initial states (often only one state). The set of reachable states is then $S_r = \text{lfp}(F_s) = post^*(S_0) = \cup\{post^n(S_0) \mid n \in \mathbb{N}\}$

We can also compute the set of possible traces as a fixpoint. Let $F_t(T) = \{s \mid s \in S_0\} \cup \{t \to s' \mid t \in T, T \text{ ends in } s, s \to s'\}$. Then the set of possible traces is $T_r = \text{lfp}(F_t)$.

Theorem 1 (Tarski) *The set of fixpoints of a monotone operator f on a complete lattice $L(\leq, \bot, \top, \sqcup, \sqcap)$ is a non-empty complete lattice, ordered by $\leq$. The least fixpoint of f is*

$$\text{lfp}(f) = \sqcap\{x \in L \mid f(x) \leq x\}$$

and its greatest fixpoint is

$$\text{gfp}(f) = \sqcup\{x \in L \mid x \leq f(x)\}.$$

Proof I only prove that $a = \sqcap\{x \in L \mid f(x) \leq x\}$ is the least fixpoint of f.

If $f(x) \leq x$, $f(a) \leq f(x) \leq x$, so $f(a) \leq a$.

Then $f(a) \in \{x \in L \mid f(x) \leq x\}$, so $a \leq f(a)$.

Then $a = f(a)$, so a is a fixpoint of f.

Consider another fixpoint x: $f(x) = x$, so $x \in \{x \in L \mid f(x) \leq x\}$, so $a \leq x$. Then a is the least fixpoint of f. □

The next few lines are for those who have heard about ordinals. The others can ignore. The least fixpoint could in fact be obtained by a transfinite iteration of f from $\bot$. If $f^\lambda(\bot)$, where λ is an ordinal, is defined by

$$\begin{aligned} f^0(\bot) &= \bot \\ f^{\lambda+1}(\bot) &= f(f^\lambda(\bot)) \\ f^\lambda(\bot) &= \sqcup\{f^\beta(\bot) \mid \beta < \lambda\} \text{ if } \lambda \text{ is a limit ordinal,} \end{aligned}$$

this transfinite sequence eventually stabilizes, and its limit is $\mathrm{lfp}(f)$ (this is a particular case of [3, Th. 2.5.2.0.2]).

Definition 7 (CPO) Let $S(\leq)$ be a partially ordered set (poset). A ***chain*** $C = (x_n)_{n\in\mathbb{N}}$ is a monotone sequence of elements of S: $x_0 \leq x_1 \leq ... \leq x_n \leq x_{n+1} \leq ....$

A ***complete partial order*** (CPO) is a poset $S(\leq)$ such that S has a least element $\bot$ and every chain C has a least upper bound $\sqcup C$.

Let $S(\leq)$ and $S'(\leq')$ be two CPOs. An operator $f : S \to S'$ is ***continuous*** if and only if for all chains $C \subseteq S$, $f(\sqcup C) = \sqcup' f(C)$.

Theorem 2 (Kleene) *Let $S(\leq)$ be a CPO. Let $f : S \to S$ be a continuous operator. Then f has a least fixpoint:*

$$\mathrm{lfp}(f) = \sqcup\{f^i(\bot) \mid i \in \mathbb{N}\}.$$

A complete lattice is also a CPO, then Kleene's theorem can be applied: it shows that the above sequence stops at ordinal ω if f is continuous.

5.2 Abstraction of fixpoints

Proposition 13 *Let F be a monotone operator on a complete lattice $F : L_1 \to L_1$, and (α, γ) be a Galois connection between L_1 and L_2. Then $\gamma(\mathrm{lfp}(\alpha \circ F \circ \gamma)) \geq \mathrm{lfp}(F)$, that is, $\mathrm{lfp}(\alpha \circ F \circ \gamma) \geq \alpha(\mathrm{lfp}(F))$.*

This is a very important result in abstract interpretation: it is the key to the analysis of loops.

Proof

$$\begin{aligned} F(\gamma(\mathrm{lfp}(\alpha \circ F \circ \gamma))) &\leq (\gamma \circ \alpha \circ F \circ \gamma)(\mathrm{lfp}(\alpha \circ F \circ \gamma)) \\ &\leq \gamma(\mathrm{lfp}(\alpha \circ F \circ \gamma)) \end{aligned}$$

so $a = \gamma(\mathrm{lfp}(\alpha \circ F \circ \gamma))$ is a post-fixpoint of F ($F(a) \leq a$). Knowing that $\mathrm{lfp}(F) = \sqcap\{x \in L \mid f(x) \leq x\}$, $\mathrm{lfp}(F) \leq a$. □

Example with signs (take again the same program of the end of the semantics section).

Then, we compute the fixpoint of $F_S^\sharp = \alpha_S \circ F_S \circ \gamma_S$. We have $F_S^\sharp(S^\sharp) = \alpha_S(S_0 \cup post(\gamma_S(S^\sharp))) = S_0^\sharp \sqcup post^\sharp(S^\sharp)$ with $S_0^\sharp = \alpha_S(S_0)$ and $post^\sharp = \alpha_S \circ post \circ \gamma_S$.

Alternatively, we can also start from traces, to obtain the same result. The abstraction of traces is:

- Let T be a set of traces. $\alpha_T(T) = \alpha_S(\{s \mid \exists t \in T, s \in t\})$
- Let $T^\sharp$ be an abstract trace. $\gamma_T(T^\sharp) = \{t \mid \forall s \in t, s \in \gamma_S(T^\sharp)\}$

The operator $F_T^\sharp$ of which we compute the fixpoint is $F^\sharp = \alpha_T \circ F_T \circ \gamma_T$, and

$$\begin{aligned}
F_T^\sharp(T^\sharp) &= \alpha_T \circ F_T \circ \gamma_T(T^\sharp) \\
&= \alpha_T \circ F_T(\{t \mid \forall s \in t, s \in \gamma_S(T^\sharp)\}) \\
&= \alpha_T(\{s \text{ (one state trace)} \mid s \in S_0\} \cup \{t \to s' \mid \forall s'' \in t, s'' \in \gamma_S(T^\sharp), t \text{ ends in } s, s \to s'\}) \\
&= \alpha_S(\{s' \mid s' \in S_0 \text{ or } \exists s \text{ such that } s \in \gamma_S(T^\sharp), s \to s'\}) \\
&= S_0^\sharp \sqcup post^\sharp(T^\sharp)
\end{aligned}$$

so we have in fact $F_S^\sharp = F_T^\sharp$. The interest of using traces appears for more precise analyses, that aim to determine properties depending not only on the set of reachable states, but also on the possible traces.

Example: we can run the fixpoint iteration on the examples of programs of the semantics section, for the interval analysis.

		x	y	x	y	x	y	x	y	x	y	x	y
1:	`input x`	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
2:	`if x <= 0 goto 4`			$I_\mathbb{Z}$	$\emptyset$	$I_\mathbb{Z}$	$\emptyset$	$I_\mathbb{Z}$	$\emptyset$	$I_\mathbb{Z}$	$\emptyset$	$I_\mathbb{Z}$	$\emptyset$
3:	`print 10/x`					I_+	$\emptyset$	I_+	$\emptyset$	I_+	$\emptyset$	I_+	$\emptyset$
4:	`input y`					I_-	$\emptyset$	$I_\mathbb{Z}$	$\emptyset$	$I_\mathbb{Z}$	$\emptyset$	$I_\mathbb{Z}$	$\emptyset$
5:	`print y*y*x`							I_-	$I_\mathbb{Z}$	$I_\mathbb{Z}$	$I_\mathbb{Z}$	$I_\mathbb{Z}$	$I_\mathbb{Z}$
6:										I_-	$I_\mathbb{Z}$	$I_\mathbb{Z}$	$I_\mathbb{Z}$

where $I_\mathbb{Z} =]-\infty, +\infty[$, $I_+ = [1, +\infty[$, $I_- =]-\infty, 0]$.

On each line, we give the abstract environment at the corresponding program point (that is, *before* the execution of the corresponding instruction). This abstract environment contains two abstract values, one for `x`, one for `y`.

The first column (after the program itself) is the initial state: the abstract environment at the first program point is empty: all variables are mapped to $\emptyset$. Other program points are not yet found reachable. The next columns represent the successive iterations of $F_S^\sharp$.

In practice, we have one abstract environment at each program point, and we iterate. Let $R_i^\sharp$ the information at $pc = i$. We project the iteration on each program point:

- Initial state: $R_i^\sharp \geq S_0^\sharp(i)$

- Iteration: $R^\sharp_{pc'} \geq post^\sharp(\lambda pc.R^\sharp_{pc})(pc')$, or simply: if $pc, R^\sharp_{pc} \to pc', R^\sharp_2$ then $R^\sharp_{pc'} \geq R^\sharp_2$.

For the last example of program of section semantics, the equations are:

$$\begin{aligned}
R^\sharp_1 &\geq \{x \mapsto \emptyset, y \mapsto \mathbb{Z}\} \\
R^\sharp_2 &\geq R^\sharp_1[x \mapsto [0,0]] \\
R^\sharp_3 &\geq R^\sharp_2 \\
R^\sharp_5 &\geq R^\sharp_2 \\
R^\sharp_4 &\geq R^\sharp_3[x \mapsto [a+1, b+1]] \text{ if } R^\sharp_3(x) = [a,b] \\
R^\sharp_2 &\geq R^\sharp_4
\end{aligned}$$

The variable `y` is considered as initialized with any value. (We could also have added `input y` at the beginning of the program.) Note that the test `if x >= y` does not give any information on the interval of `x` since `y` can have any value.

The equations can be iterated in any order, provided that, when a parameter of an equation is modified, then the equation is executed again at some point. This is called *chaotic* iterations.

5.3 Widening [12]

Example with intervals.

```
   input n;
   x := 0;
2: x++;
   print x;
   if x < n goto 2
```

The analysis does not terminate, because the interval becomes larger and large without stabilizing. This can happen when the height of the lattice is not finite. We need to "accelerate the convergence" of the iterations. This can be done by a widening.

Definition 8 (Widening) A ***widening*** operator is a function $\triangledown : L \times L \to L$ such that for all x, y in L, $x \leq x \triangledown y$, $y \leq x \triangledown y$, and for all increasing chains $x_0 \leq x_1 \leq \ldots$, the increasing chain $y_0 = x_0, \ldots, y_{i+1} = y_i \triangledown x_{i+1}, \ldots$ is not strictly increasing.

Proposition 14 *Assuming F monotone and $\triangledown$ a widening, the sequence*

$$\begin{aligned}
X_0 &= \bot \\
X_{i+1} &= X_i && \text{if } F(X_i) \leq X_i \\
&= X_i \triangledown F(X_i) && \text{otherwise}
\end{aligned}$$

is ultimately stationary and its limit is greater than (or equal to) $\mathrm{lfp}(F)$.

Proof The sequence X_i is increasing. Indeed, $X_{i+1} \geq F(X_i)$. This is obvious when $X_{i+1} = X_i$. Otherwise, $X_{i+1} = X_i \triangledown F(X_i) \geq X_i$. Define $x_0 = \bot$, $x_{i+1} = F(X_i)$. This sequence is also increasing since F is monotone. Taking $y_0 = x_0 = \bot$, $y_{i+1} = y_i \triangledown x_{i+1} = y_i \triangledown F(X_i)$, $y_i = X_i$ by induction. Moreover, y_i is not strictly increasing, so there exists i such that $X_{i+1} = X_i$. That is, either $X_i \triangledown F(X_i) = X_i$, so $F(X_i) \leq X_i \triangledown F(X_i) = X_i$, so this case is impossible, or $F(X_i) \leq X_i$. Now, the sequence is stationary, since $F(X_{i+1}) = F(X_i) \leq X_i = X_{i+1}$ so the first case applies forever.

When the sequence stabilizes, $F(X_i) \leq X_i$ as shown above. Since the least fixpoint is the greatest lower bound of the post-fixpoints of F, $X_i \geq \mathrm{lfp}(F)$. □

Numerous variants are possible. We could use a different widening for each iterate, or a widening that depends on the history of all previous iterates.

Example: widening for intervals $\emptyset \triangledown X = X \triangledown \emptyset = X$, $[a, b] \triangledown [a', b'] = [\text{if } a' < a \text{ then } -\infty \text{ else } a, \text{if } b' > b \text{ then } +\infty \text{ else } b]$. We will run this widening on the example in the course.

5.4 Narrowing

However, the widening may lead to an important loss of precision. Example with intervals.

```
   x := 0;
2: x++;
   print x;
   if x < 5 goto 2
```

We obtain $[0, \infty[$ as interval for x, while in reality $x \in [0, 5]$. We can regain some precision by performing decreasing iterations after the widening sequence has stabilized.

Definition 9 (Narrowing) A ***narrowing*** operator is a function $\Delta : L \times L \to L$ such that for all x, y in L, $y \leq x$ implies $y \leq x \Delta y \leq x$, and for all decreasing chains $x_0 \geq x_1 \geq \ldots$, the decreasing chain defined by $y_0 = x_0, \ldots, y_{i+1} = y_i \Delta x_{i+1}, \ldots$ is not strictly decreasing.

Proposition 15 *Assume that F is monotone, and Δ is a narrowing. If $X_0 \geq \mathrm{lfp}(F)$ and $F(X_0) \leq X_0$, then the sequence $X_{i+1} = X_i \Delta F(X_i)$ is decreasing, ultimately stationary and its limit is greater than (or equal to)* $\mathrm{lfp}(F)$.

Proof First, for all i, $X_i \geq \mathrm{lfp}(F)$ and $F(X_i) \leq X_i$. True for X_0. If it is true for i, $X_{i+1} = X_i \Delta F(X_i)$, then $F(X_i) \leq X_i \Delta F(X_i) = X_{i+1} \leq X_i$, so $X_{i+1} \geq F(X_i) \geq F(\mathrm{lfp}(F)) = \mathrm{lfp}(F)$ (since $X_i \geq \mathrm{lfp}(F)$ and F is monotone). $F(X_{i+1}) \leq F(X_i)$ since $X_{i+1} \leq X_i$, so $F(X_{i+1}) \leq X_{i+1}$.

We have also seen that $X_{i+1} \leq X_i$ so the sequence is decreasing.

Take $x_{i+1} = F(X_i)$. The sequence x_i is decreasing. Take $y_0 = x_0, \ldots, y_{i+1} = y_i \Delta x_{i+1}$. Then $y_i = X_i$, so X_i is not strictly decreasing. When $X_{i+1} = X_i$, the sequence stabilizes. □

If $F(X_i) = X_i$ then $X_{i+1} = X_i$, so if X_0 is a fixpoint of F, it cannot be improved by narrowing.

The iterations with narrowing can be stopped at any point, since all iterates are greater than the fixpoint. Waiting until the sequence stabilizes yields the best precision. Note that

it is also possible to iterate simply F after computing the sequence with widening, but then, we have to set an arbitrary limit on the number of iterations, because we have no guarantee that the decreasing iterations of F will terminate.

Example with intervals: $\emptyset \Delta X = X \Delta \emptyset = \emptyset$, $[a, b] \Delta [a', b'] = [\text{if } a = \infty \text{ then } a' \text{ else } a, \text{if } b = +\infty \text{ then } b' \text{ else } b]$. This narrowing improves infinite bounds only. In the previous example of program, we now find the good bound for x. (We will show that in the course.)

One interesting question is whether all analyses that can be done with widening and narrowing can be done with lattices of finite height. The answer is no, see [12]. In contrast, all analyses done with an abstraction can be done with a widening/narrowing in the concrete lattice.

5.5 Iteration strategies [1, 18]

At least when we have no widening/narrowing, the result of the computation does not depend on the order in which we iterate the equations, so we can iterate them in any order (chaotic iterations).

However, choosing a good iteration ordering can reduce the computation time dramatically. The main idea is that, when x depends on y, we try to compute y first, then x. Of course, this is not possible when there are cycles in the dependencies (that's why we need to iterate).

Here, we give a good iteration algorithm, taken from [18].

1. Build the dependency graph of the equations: nodes n are unknowns, edges are $n \to n'$ when n' depends on n.

2. Split the dependency graph into strongly connected components (see Tarjan's scc algorithm [22] and below). In an oriented graph, a strongly connected component is a subset S of the nodes of the graph, such that for all nodes n and n' in S, there exists a path from n to n' and from n' to n.

3. Consider the strongly connected components in topological order. In an acyclic oriented graph, the topological order is defined by $n \leq n'$ if and only if there exists a path from n to n'. After merging all nodes of the strongly connected components in the dependency graph, the remaining graph is acyclic, so we can compute the topological order on it.

   Inside each strongly connected component, iterate in the order given by the scc building (reverse order of `nodelist` below, since `nodelist` contains the node in reverse dependency order, and it is better iterating in dependency order: if the value at n' depends on the value at n, try to compute n first). At each iteration, apply all equations of the scc, and repeat until the scc stabilizes.

In the presence of widening/narrowing, it is enough to apply them at least once per cycle, for example at the end of back edges of the dependency graph.

A theoretically better algorithm (that is, better in the worst case) has been given in [1]. In practice, the one above is quite good, probably better than [1], and certainly simpler.

Here is Tarjan's algo. We do not detail how this algorithm works, since it is a bit far away from abstract interpretation strictly speaking. For general information on graph algorithms (including another algorithm for splitting into strongly connected components), see [2].

```
age := 0
nodelist := []
for all nodes n, n.age := -1

depth n =
  if n.age = -1 then
  begin
    age := age + 1;
    nodelist := n :: nodelist;
    n.age := age;
    age_sons := min { depth n' | n -> n' is an edge }
    if age_sons >= n.age then
    begin
      (* top node of a scc *)
      (* An scc is the set of nodes in nodelist until n (n included)
         Do the desired treatment on that scc here *)
      nodelist := the tail of nodelist, after n
      return age_sons
    end
    else
      return age_sons
  end
  else
    return n.age
```

6 Abstract domains

6.1 Numerical abstract domains

6.1.1 Constants

$\bot \leq c \leq \top$

6.1.2 Intervals

$\emptyset \leq [a, b]$, with $a \in \mathbb{Z} \cup \{-\infty\}$, $b \in \mathbb{Z} \cup \{+\infty\}$, $a \leq b$.

6.1.3 Linear equalities [19]

$a_1 x_1 + \ldots + a_n x_n = c$

6.1.4 Polyhedra [14]

$a_1x_1 + \ldots + a_nx_n \leq c$

6.1.5 Octagons [20]

$+/-x +/-y \leq c$.

6.1.6 Congruences [17]

$a_1x_1 + \ldots + a_nx_n \equiv c(d)$

6.2 Other abstract domains

6.2.1 Alias analysis [15, 21, 23]

6.2.2 Escape analysis [16]

7 Abstract interpretation frameworks [11]

7.1 Upper closure operators

Definition 10 An ***upper closure operator*** ρ is a function $\rho : L \to L$, where L is a complete lattice, such that ρ is monotone, $\rho(x) \geq x$, and $\rho \circ \rho = \rho$.

Proposition 16
1. *If (α, γ) is a Galois connection from L_1 to L_2, then $\gamma \circ \alpha$ is an upper closure operator on L_1.*
2. *If ρ is an upper closure operator on L_1, then (ρ, Id) is a Galois connection from L_1 to $\rho(L_1)$.*

So we have equivalence between the formalism of Galois connections and the one of upper closure operators.

7.2 Concretization

When there is no best approximation (so we cannot define an abstraction α), we can work with only a concretization relation γ. y is a correct approximation of x if and only if $x \leq \gamma(y)$.

With such a framework, we cannot systematically compute the analysis, but we can prove the correctness of an analysis.

7.3 Correctness relation

The most general framework, but also the weakest, is to use neither an abstraction nor a concretization, but only a correctness relation σ. y is a correct approximation of x if and only if $\sigma(x, y)$.

If we work with only one lattice (like in the upper closure operator framework), the correctness relation is simply $x \leq y$.

8 Abstract interpretation as a "thinking tool"

Abstract interpretation can be used to formalize many concepts in the area of programming languages. Here are some examples with references to the corresponding papers. (Not very easy to read!)

8.1 Hierarchy of semantics of programming languages [4]

Semantics are abstractions of other semantics (the trace semantics being the most concrete one). For instance, the set of states considered above is an abstraction of the set of traces.

8.2 Types as abstract interpretation [5]

8.3 Program transformations [13]

9 Some introductory papers

The following papers can be a good starting point to study abstract interpretation: [6–8]. The papers [7] and [8] give some intuition without too much formalism. [6] is much more formal. If you skip the more complex formulae (or if you are very courageous to decrypt them!), it can be a good example of how to formally design an analysis by abstract interpretation.

Then, [11] and [12] can be good papers to continue with. The first papers, [9] and [10] are also interesting, but rather difficult to read.

Papers by Patrick Cousot are available at `http://www.di.ens.fr/~cousot/COUSOTpapers.shtml`.

References

[1] François Bourdoncle. Efficient chaotic iteration strategies with widenings. In *Proc. of the International Conference on Formal Methods in Programming and their Applications*, volume 735 of *Lecture Notes on Computer Science*, pages 128–141. Springer Verlag, 1993.

[2] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to algorithms*. MIT Press, 1990.

[3] Patrick Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique des programmes*. PhD thesis, Université Scientifique et Médicale de Grenoble, 21 March 1978.

[4] Patrick Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Electronic Notes in Theoretical Computer Science*, 6, 1997. URL: `http://www.elsevier.nl/locate/entcs/volume6.html`, 25 pages.

[5] Patrick Cousot. Types as Abstract Interpretations. In *24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'97)*, pages 316–331, Paris, France, January 1997.

[6] Patrick Cousot. The Calculational Design of a Generic Abstract Interpreter. In M. Broy and R. Steinbrüggen, editors, *Calculational System Design*. NATO ASI Series F. IOS Press, Amsterdam, 1999.

[7] Patrick Cousot. Abstract Interpretation: Achievements and Perspectives. In *Proceedings of the SSGRR 2000 Computer & eBusiness International Conference*, Compact disk paper 224 and electronic proceedings `http://www.ssgrr.it/en/ssgrr2000/proceedings.htm`, L'Aquila, Italy, July 31 – August 6 2000. Scuola Superiore G. Reiss Romoli.

[8] Patrick Cousot. Abstract Interpretation Based Formal Methods and Future Challenges, invited paper. In R. Wilhelm, editor, *" Informatics — 10 Years Back, 10 Years Ahead "*, volume 2000 of *Lecture Notes on Computer Science*, pages 138–156. Springer Verlag, 2001.

[9] Patrick Cousot and Radhia Cousot. Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Conference Record of the 4th Annual ACM Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, January 1977.

[10] Patrick Cousot and Radhia Cousot. Systematic Design of Program Analysis Frameworks. In *Conference Record of the Sixth Annual ACM Symposium on Principles of Programming Languages*, pages 269–282, San Antonio, Texas, 29-31 January 1979.

[11] Patrick Cousot and Radhia Cousot. Abstract Interpretation Frameworks. *Journal of Logic and Computation*, 2(4):511–547, August 1992.

[12] Patrick Cousot and Radhia Cousot. Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation. In Maurice Bruynooghe and Martin Wirsing, editors, *Proceedings of the fourth international symposium PLILP'92 (Programming Language Implementation and Logic Programming)*, Lecture Notes on Computer Science, pages 269–295. Springer Verlag, August 1992.

[13] Patrick Cousot and Radhia Cousot. Systematic design of program transformation frameworks by abstract interpretation. In *Conference Record of the Twentyninth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 178–190, Portland, Oregon, January 2002. ACM Press.

[14] Patrick Cousot and Nicolas Halbwachs. Automatic Discovery of Linear Restraints Among Variables of a Program. In *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages*, pages 84–96, January 1978.

[15] Alain Deutsch. Interprocedural May-Alias Analysis for Pointers: Beyond k-limiting. In *Proceedings of the SIGPLAN '94 Conference on Programming Language Design and Implementation*, pages 230–241, Orlando, Florida, 20-24 June 1994. ACM Press.

[16] David Gay and Bjarne Steensgaard. Fast Escape Analysis and Stack Allocation for Object-Based Programs. In David A. Watt, editor, *Compiler Construction, 9th International Conference, CC'2000*, volume 1781 of *Lecture Notes on Computer Science*, pages 82–93. Springer Verlag, March 2000.

[17] Philippe Granger. Static Analysis of Linear Congruence Equalities among Variables of a Program. In *Proceedings of the International Joint Conference on Theory and Practice of Software Development (TAPSOFT'91)*, volume 1, pages 169–192, April 1991.

[18] Susan Horwitz, Alan Demers, and Tim Teitelbaum. An Efficient General Iterative Algorithm for Dataflow Analysis. *Acta Informatica*, 24(6):679–694, 1987.

[19] Michael Karr. Affine Relationships Among Variables of a Program. *Acta Informatica*, 6:133–151, 1976.

[20] Antoine Miné. The octagon abstract domain. In *Analysis, Slicing, and Transformation (AST 2001) in WCRE 2001*, pages 310–319. IEEE Computer Society Press, October 2001.

[21] Bjarne Steensgaard. Points-to analysis in almost linear time. In *Proceedings of the Twentythird Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 32–41, January 1996.

[22] Robert E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[23] Arnaud Venet. Nonuniform Alias Analysis of Recursive Data Structures and Arrays. In Manuel V. Hermenegildo and German Puebla, editors, *Static Analysis, 9th International Symposium, SAS 2002*, volume 2477 of *Lecture Notes on Computer Science*, pages 36–51, Madrid, Spain, September 2002. Springer Verlag.