Abstract Interpretation
Semantics and applications to verification

Xavier Rival
École Normale Supérieure

April 15th, 2016
Program of this lecture

Studied so far:

- **semantics**: behaviors of programs
- **properties**: safety, liveness, security...
- **approaches to verification**: typing, use of proof assistants, model checking

Today’s lecture: introduction to abstract interpretation

a general framework for comparing semantics introduced by Patrick Cousot and Radhia Cousot (1977)

- **abstraction**: use of a lattice of predicates
- **computing abstract over-approximations**, while preserving soundness
- **computing abstract over-approximations for loops**
Outline

1. Abstraction
   - Notion of abstraction
   - Abstraction and concretization functions
   - Galois connections

2. Abstract interpretation

3. Application of abstract interpretation

4. Conclusion
Abstraction example 1: signs

Abstraction: defined by a family of properties to use in proofs

Example:
- objects under study: sets of mathematical integers
- abstract elements: signs

Lattice of signs

- \( \bot \) denotes only \( \emptyset \)
- \( \pm \) denotes any set of positive integers
- \( 0 \) denotes any subset of \( \{0\} \)
- \( - \) denotes any set of negative integers
- \( \top \) denotes any set of integers

Note: the order in the abstract lattice corresponds to inclusion...
Abstraction example 1: signs

**Definition: abstraction relation**

- **Concrete elements**: elements of the original lattice \( c \in \mathcal{P}(\mathbb{Z}) \)
- **Abstract elements**: predicate \( a: \cdot \in \{\pm, 0, \ldots\} \)
- **Abstraction relation**: \( c \vdash_s a \) when \( a \) describes \( c \)

**Examples**:

- \( \{1, 2, 3, 5, 7, 11, 13, 17, 19, 23, \ldots\} \vdash_s \pm \)
- \( \{1, 2, 3, 5, 7, 11, 13, 17, 19, 23, \ldots\} \vdash_s \top \)

We use abstract elements to reason about operations:

- if \( c_0 \vdash_s \pm \) and \( c_1 \vdash_s \pm \), then \( \{x_0 + x_1 \mid x_i \in c_i\} \vdash_s \pm \)
- if \( c_0 \vdash_s \pm \) and \( c_1 \vdash_s \pm \), then \( \{x_0 \cdot x_1 \mid x_i \in c_i\} \vdash_s \pm \)
- if \( c_0 \vdash_s \pm \) and \( c_1 \vdash_s 0 \), then \( \{x_0 \cdot x_1 \mid x_i \in c_i\} \vdash_s 0 \)
- if \( c_0 \vdash_s \pm \) and \( c_1 \vdash_s \bot \), then \( \{x_0 \cdot x_1 \mid x_i \in c_i\} \vdash_s \bot \)
Abstraction example 1: signs

We can also consider the **union operation**:

- if \(c_0 \vdash_S \pm\) and \(c_1 \vdash_S \pm\), then \(c_0 \cup c_1 \vdash_S \pm\)
- if \(c_0 \vdash_S \pm\) and \(c_1 \vdash_S \bot\), then \(c_0 \cup c_1 \vdash_S \pm\)

But, what can we say about \(c_0 \cup c_1\), when \(c_0 \vdash_S 0\) and \(c_1 \vdash_S \pm\) ?

- clearly, \(c_0 \cup c_1 \vdash_S \top\)...
- but no other relation holds
- in the abstract, we do not rule out negative values

We can extend the initial lattice:

- \(\geq 0\) denotes any set of positive or null integers
- \(\leq 0\) denotes any set of negative or null integers
- \(\neq 0\) denotes any set of non null integers
- if \(c_0 \vdash_S \pm\) and \(c_1 \vdash_S 0\), then \(c_0 \cup c_1 \vdash_S \geq 0\)
Abstraction example 2: constants

**Definition: abstraction based on constants**

- **concrete elements:** $\mathcal{P}(\mathbb{Z})$
- **abstract elements:** $\bot, \top, n$ where $n \in \mathbb{Z}$
  \[ D_C^\# = \{ \bot, \top \} \cup \{ n \mid n \in \mathbb{Z} \} \]
- **abstraction relation:** $c \vdash c n \iff c \subseteq \{ n \}$

We obtain a flat lattice:

\[
\begin{array}{ccccccc}
\vdash & \vdash & \vdash & \vdash & \vdash & \vdash \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
\vdash & \vdash & \vdash & \vdash & \vdash & \vdash \\
\end{array}
\]

**Abstract reasoning:**

- if $c_0 \vdash c n_0$ and $c_1 \vdash c n_1$, then \( \{ k_0 + k_1 \mid k_i \in c_i \} \vdash c n_0 + n_1 \)
Abstraction example 3: Parikh vector

Definition: Parikh vector abstraction

- **concrete elements**: \( \mathcal{P}(\mathcal{A}^*) \) (sets of words over alphabet \( \mathcal{A} \))
- **abstract elements**: \( \{\bot, \top\} \cup (\mathcal{A} \rightarrow \mathbb{N}) \)
- **abstraction relation**: \( c \vdash \phi : \mathcal{A} \rightarrow \mathbb{N} \) if and only if:

\[
\forall w \in c, \forall a \in \mathcal{A}, \ a \text{ appears } \phi(a) \text{ times in } w
\]

Abstract reasoning:

- **concatenation**: If \( \phi_0, \phi_1 : \mathcal{A} \rightarrow \mathbb{N} \) and \( c_0, c_1 \) are such that \( c_i \vdash \phi_i \),

\[
\{ w_0 \cdot w_1 | w_i \in c_i \} \vdash \phi_0 + \phi_1
\]

Information preserved, information deleted:

- **very precise** information about the number of occurrences
- the order of letters is totally abstracted away (lost)
Abstraction example 4: interval abstraction

Definition: abstraction based on intervals

- **Concrete elements:** $\mathcal{P}(\mathbb{Z})$
- **Abstract elements:** $\perp, \top, (a, b)$ where $a \in \{-\infty\} \cup \mathbb{Z}$, $b \in \mathbb{Z} \cup \{+\infty\}$ and $a \leq b$
- **Abstraction relation:**

  $\emptyset \vdash_{\mathcal{I}} \perp$
  $S \vdash_{\mathcal{I}} \top$
  $S \vdash_{\mathcal{I}} (a, b) \iff \forall x \in S, a \leq x \leq b$

Operations: TD
Abstraction example 5: non relational abstraction

Definition: non relational abstraction

- **concrete elements**: $\mathcal{P}(X \rightarrow Y)$, inclusion ordering
- **abstract elements**: $X \rightarrow \mathcal{P}(Y)$, pointwise inclusion ordering
- **abstraction relation**: $c \vdash_{\mathcal{N}_R} a \iff \forall \phi \in c, \forall x \in X, \phi(x) \in a(x)$

Information preserved, information deleted:

- **very precise** information about the **image** of the functions in $c$
- **relations** such as (for given $x_0, x_1 \in X, y_0, y_1 \in Y$) the following are lost:
  
  $\forall \phi \in c, \phi(x_0) = \phi(x_1)$
  
  $\forall \phi \in c, \forall x, x' \in X, \phi(x) \neq y_0 \lor \phi(x') \neq y_1$
Notion of abstraction relation

**Concrete order:** so far, always inclusion
- the tighter the concrete set, the fewer behaviors
- smaller concrete sets correspond to more precise properties

**Abstraction relation:** $c \models a$ when $c$ satisfies $a$
- if $c_0 \subseteq c_1$ and $c_1$ satisfies $a$, in all our examples, $c_0$ also satisfies $a$

**Abstract order:** in all our examples,
- it matches the abstraction relation as well:
  if $a_0 \sqsubseteq a_1$ and $c$ satisfies $a_0$, then $c$ also satisfies $a_1$
- great advantage: we can reason about implication in the abstract, without looking back at the concrete properties

We will now formalize this in detail...
Outline

1 Abstraction
   - Notion of abstraction
   - Abstraction and concretization functions
   - Galois connections

2 Abstract interpretation

3 Application of abstract interpretation

4 Conclusion
Towards adjoint functions

We consider a **concrete lattice** \((C, \subseteq)\) and an **abstract lattice** \((A, \sqsubseteq)\).

So far, we used **abstraction relations**, that are consistent with orderings:

**Abstraction relation compatibility**

- \(\forall c_0, c_1 \in C, \forall a \in A, c_0 \subseteq c_1 \land c_1 \vdash a \implies c_0 \vdash a\)
- \(\forall c \in C, \forall a_0, a_1 \in A, c \vdash a_0 \land a_0 \sqsubseteq a_1 \implies c \vdash a_1\)

When we have a \(c\) (resp., \(a\)) and try to map it into a compatible \(a\) (resp. \(c\)), the abstraction relation is not a convenient tool.

Hence, we shall use **adjoint functions** between \(C\) and \(A\).

- from concrete to abstract: **abstraction**
- from abstract to concrete: **concretization**
Concretization function

Our **first adjoint function**: 

**Definition: concretization function**

Concretization function \( \gamma : A \rightarrow C \) (if it exists) maps abstract \( a \) into the weakest (i.e., most general) concrete \( c \) that satisfies \( a \) (i.e., \( c \vdash a \)).

Note: in common cases, there exists a \( \gamma \).

- \( c \vdash a \) if and only if \( c \subseteq \gamma(a) \)
Concretization function: a few examples

Signs abstraction:

\[ \gamma_S : \begin{array}{c}
\top & \mapsto \mathbb{Z} \\
\pm & \mapsto \mathbb{Z}^+ \\
0 & \mapsto \{0\} \\
\mp & \mapsto \mathbb{Z}^- \\
\bot & \mapsto \emptyset
\end{array} \]

Constants abstraction:

\[ \gamma_C : \begin{array}{c}
\top & \mapsto \mathbb{Z} \\
n & \mapsto \{n\} \\
\bot & \mapsto \emptyset
\end{array} \]

Non relational abstraction:

\[ \gamma_{NR} : (X \to \mathcal{P}(Y)) \mapsto \mathcal{P}(X \to Y) \]

\[ \Phi \mapsto \{\phi : X \to Y \mid \forall x \in X, \phi(x) \in \Phi(x)\} \]

Parikh vector abstraction: exercise!
Abstraction function

Our second adjoint function:

**Definition: abstraction function**

**Abstraction function** $\alpha : C \to A$ (if it exists) maps concrete $c$ into the most precise abstract $a$ that soundly describes $c$ (i.e., $c \vdash a$).

Note: in quite a few cases (including some in this course), there is no $\alpha$.

**Summary on adjoint functions:**

- $\alpha$ returns the **most precise abstract predicate** that holds true for its argument
  this is called the **best abstraction**
- $\gamma$ returns the **most general concrete meaning** of its argument
  hence, is called the **concretization**
Abstraction: a few examples

Constants abstraction:

$$\alpha_C : (c \subseteq \mathbb{Z}) \mapsto \begin{cases} \bot & \text{if } c = \emptyset \\ n & \text{if } c = \{n\} \\ \top & \text{otherwise} \end{cases}$$

Non relational abstraction:

$$\alpha_{NR} : (c \subseteq (X \rightarrow Y)) \mapsto (x \in X) \mapsto \{\phi(x) \mid \phi \in c\}$$

Signs abstraction and Parikh vector abstraction: exercises
Outline

1 Abstraction
   - Notion of abstraction
   - Abstraction and concretization functions
   - Galois connections

2 Abstract interpretation

3 Application of abstract interpretation

4 Conclusion
So far, we have:

- **abstraction** $\alpha : C \rightarrow A$
- **concretization** $\gamma : A \rightarrow C$

How to tie them together?

**They should agree on a same abstraction relation $\vdash$ !**

**Definition: Galois connection**

A **Galois connection** is defined by a concrete lattice $(C, \subseteq)$, an abstract lattice $(A, \sqsubseteq)$, an abstraction function $\alpha : C \rightarrow A$ and a concretization function $\gamma : A \rightarrow C$ such that:

$$\forall c \in C, \forall a \in A, \alpha(c) \sqsubseteq a \iff c \subseteq \gamma(a) \quad (\iff c \vdash a)$$

**Notation:**

$$(C, \subseteq) \xleftarrow{\alpha} \xrightarrow{\gamma} (A, \sqsubseteq)$$

Note: in practice, we shall rarely use $\vdash$; we use $\alpha, \gamma$ instead
Example: constants abstraction and Galois connection

**Constants lattice** $D^\#_C = \{\bot, \top\} \uplus \{n \mid n \in \mathbb{Z}\}$

\[
\begin{align*}
\alpha_C(c) &= \bot \quad \text{if } c = \emptyset \\
\alpha_C(c) &= n \quad \text{if } c = \{n\} \\
\alpha_C(c) &= \top \quad \text{otherwise}
\end{align*}
\]

\[
\begin{align*}
\gamma_C(\top) &\mapsto \mathbb{Z} \\
\gamma_C(n) &\mapsto \{n\} \\
\gamma_C(\bot) &\mapsto \emptyset
\end{align*}
\]

Thus:

- if $c = \emptyset$, $\forall a$, $c \subseteq \gamma_C(a)$, i.e., $c \subseteq \gamma_C(a) \iff \alpha_C(c) = \bot \subseteq a$
- if $c = \{n\}$,
  \[
  \alpha_C(\{n\}) = n \subseteq c \iff c = n \vee c = \top \iff c = \{n\} \subseteq \gamma_C(a)
  \]
- if $c$ has at least two distinct elements $n_0, n_1$, $\alpha_C(c) = \top$ and $c \subseteq \gamma_C(a) \Rightarrow a = \top$, i.e., $c \subseteq \gamma_C(a) \iff \alpha_C(c) = \bot \subseteq a$

**Constant abstraction: Galois connection**

\[
c \subseteq \gamma_C(a) \iff \alpha_C(c) \subseteq a, \text{ therefore, } (\mathcal{P}(\mathbb{Z}), \subseteq) \xleftarrow{\alpha_C} (D^\#_C, \subseteq)
\]
Example: non relational abstraction Galois connection

We have defined:
\[ \alpha_{NR} : (c \subseteq (X \rightarrow Y)) \rightarrow \mapsto (x \in X) \mapsto \{ f(x) | f \in c \} \]
\[ \gamma_{NR} : (\Phi \in (X \rightarrow \mathcal{P}(Y))) \rightarrow \mapsto \{ f : X \rightarrow Y | \forall x \in X, f(x) \in \Phi(x) \} \]

Let \( c \in \mathcal{P}(X \rightarrow Y) \) and \( \Phi \in (X \rightarrow \mathcal{P}(Y)) \); then:
\[ \alpha_{NR}(c) \subseteq \Phi \iff \forall x \in X, \alpha_{NR}(c)(x) \subseteq \Phi(x) \]
\[ \iff \forall x \in X, \{ f(x) | f \in c \} \subseteq \Phi(x) \]
\[ \iff \forall f \in c, \forall x \in X, f(x) \in \Phi(x) \]
\[ \iff \forall f \in c, f \in \gamma_{NR}(\Phi) \]
\[ \iff c \subseteq \gamma_{NR}(\Phi) \]

Non relational abstraction: Galois connection
\[ c \subseteq \gamma_{NR}(a) \iff \alpha_{NR}(c) \subseteq a, \text{ therefore,} \]
\[ (\mathcal{P}(X \rightarrow Y), \subseteq) \xrightarrow{\alpha_{NR}} (X \rightarrow \mathcal{P}(Y), \subseteq) \xrightarrow{\gamma_{NR}} \]
Galois connection properties

Galois connections have **many useful properties**.

In the next few slides, we consider a Galois connection \((C, \subseteq) \leftarrow \gamma \rightarrow (A, \subseteq)\) and establish a few interesting properties.

**Extensivity, contractivity**

- \(\alpha \circ \gamma\) is **contractive**: \(\forall a \in A, \alpha \circ \gamma(a) \subseteq a\)
- \(\gamma \circ \alpha\) is **extensive**: \(\forall c \in C, c \subseteq \gamma \circ \alpha(c)\)

**Proof:**

- let \(a \in A\); then, \(\gamma(a) \subseteq \gamma(a)\), thus \(\alpha(\gamma(a)) \subseteq a\)
- let \(c \in C\); then, \(\alpha(c) \subseteq \alpha(c)\), thus \(c \subseteq \gamma(\alpha(a))\)
Galois connection properties

Monotonicity of adjoints
- $\alpha$ is monotone
- $\gamma$ is monotone

Proof:
- **monotonicity of $\alpha$:** let $c_0, c_1 \in C$ such that $c_0 \subseteq c_1$; by extensivity of $\gamma \circ \alpha$, $c_1 \subseteq \gamma(\alpha(c_1))$, so by transitivity, $c_0 \subseteq \gamma(\alpha(c_1))$ by definition of the Galois connection, $\alpha(c_0) \subseteq \alpha(c_1)$
- **monotonicity of $\gamma$:** same principle

**Note:** many proofs can be derived by duality

Duality principle applied for Galois connections

If $(C, \subseteq) \xleftarrow{\gamma} (A, \sqsubseteq)$, then $(A, \sqsupseteq) \xleftarrow{\alpha} (C, \supseteq)$
Galois connection properties

**Iteration of adjoints**

- \( \alpha \circ \gamma \circ \alpha = \alpha \)
- \( \gamma \circ \alpha \circ \gamma = \gamma \)
- \( \alpha \circ \gamma \) (resp., \( \gamma \circ \alpha \)) is idempotent, hence a lower (resp., upper) closure operator

**Proof:**

- \( \alpha \circ \gamma \circ \alpha = \alpha \):
  - let \( c \in C \), then \( \gamma \circ \alpha(c) \subseteq \gamma \circ \alpha(c) \)
  - hence, by the Galois connection property, \( \alpha \circ \gamma \circ \alpha(c) \subseteq \alpha(c) \)
  - moreover, \( \gamma \circ \alpha \) is extensive and \( \alpha \) monotone, so \( \alpha(c) \subseteq \alpha \circ \gamma \circ \alpha(c) \)
  - thus, \( \alpha \circ \gamma \circ \alpha(c) = \alpha(c) \)

- the second point can be proved similarly (duality); the others follow
Galois connection properties

Properties on iterations of adjoint functions:

![Diagram showing Galois connections between concrete and abstract domains with labels α and γ.]
Galois connection properties

\( \alpha \) preserves least upper bounds

\[ \forall c_0, c_1 \in C, \; \alpha(c_0 \cup c_1) = \alpha(c_0) \sqcup \alpha(c_1) \]

By duality:

\[ \forall a_0, a_1 \in A, \; \gamma(c_0 \cap c_1) = \gamma(c_0) \sqcap \gamma(c_1) \]

**Proof:**
First, we observe that \( \alpha(c_0) \sqcup \alpha(c_1) \sqsubseteq \alpha(c_0 \cup c_1) \), i.e. \( \alpha(c_0 \cup c_1) \) is an upper bound of \( \{ \alpha(c_0), \alpha(c_1) \} \).

We now prove it is the least upper bound. For all \( a \in A \):

\[ \alpha(c_0 \cup c_1) \sqsubseteq a \iff c_0 \cup c_1 \sqsubseteq \gamma(a) \]
\[ \iff c_0 \sqsubseteq \gamma(a) \wedge c_1 \sqsubseteq \gamma(a) \]
\[ \iff \alpha(c_0) \sqsubseteq a \wedge \alpha(c_1) \sqsubseteq a \]
\[ \iff \alpha(c_0) \sqcup \alpha(c_1) \sqsubseteq a \]

**Note:** when \( C, A \) are complete lattices, this extends to families of elements.
Galois connection properties

Uniqueness of adjoints

- given $\gamma : C \to A$, there exists at most one $\alpha : A \to C$ such that $(C, \subseteq) \xrightarrow{\gamma} (A, \subseteq)$, and, if it exists, $\alpha(c) = \cap \{ a \in A \mid c \subseteq \gamma(a) \}$

- similarly, given $\alpha : A \to C$, there exists at most one $\gamma : C \to A$ such that $(C, \subseteq) \xleftarrow{\alpha} (A, \subseteq)$, and it is defined dually

Proof of the first point (the other follows by duality):
we assume that there exists an $\alpha$ so that we have a Galois connection and prove that, $\alpha(c) = \cap \{ a \in A \mid c \subseteq \gamma(a) \}$ for a given $c \in C$.

- if $a \in A$ is such that $c \subseteq \gamma(a)$, then $\alpha(a) \subseteq c$ thus, $\alpha(a)$ is a lower bound of $\{ a \in A \mid c \subseteq \gamma(a) \}$.

- let $a_0 \in A$ be a lower bound of $\{ a \in A \mid c \subseteq \gamma(a) \}$.
  since $\gamma \circ \alpha$ is extensive, $c \subseteq \gamma(\alpha(c))$ and $\alpha(c) \in \{ a \in A \mid c \subseteq \gamma(a) \}$.
  hence, $a_0 \subseteq \alpha(c)$

Thus, $\alpha(c)$ is the least upper bound of $\{ a \in A \mid c \subseteq \gamma(a) \}$
Construction of adjoint functions

The adjoint uniqueness property is actually a very strong property:
- it allows to construct an abstraction from a concretization
- ... or to understand why no abstraction can be constructed :-)

Turning an adjoint into a Galois connection (1)

Let $(C, \subseteq)$ and $(A, \sqsubseteq)$ be two lattices, such that any subset of $A$ as a greatest lower bound and let $\gamma : (A, \sqsubseteq) \to (C, \subseteq)$ be a monotone function.

Then, the function below defines a Galois connection:

$$\alpha(c) = \sqcap \{ a \in A \mid c \subseteq \gamma(a) \}$$

Example of abstraction with no $\alpha$: when $\sqcap$ is not defined on all families, e.g., lattice of convex polyhedra, abstracting sets of points in $\mathbb{R}^2$.

Exercise: state the dual property and apply the same principle to the concretization.
Galois connection characterization

A characterization of Galois connections

Let \((C, \subseteq)\) and \((A, \sqsubseteq)\) be two lattices, and \(\alpha : C \to A\) and \(\gamma : A \to C\) be two monotone functions, such that:

- \(\alpha \circ \gamma\) is contractive
- \(\gamma \circ \alpha\) is extensive

Then, we have a Galois connection

\[(C, \subseteq) \xleftrightarrow{\gamma} (A, \sqsubseteq)\]

Proof:

- let \(c \in C\) and \(a \in A\) such that \(\alpha(c) \sqsubseteq a\).
  then: \(\gamma(\alpha(c)) \subseteq \gamma(a)\) (as \(\gamma\) is monotone)
  \(c \subseteq \gamma(\alpha(c))\) (as \(\gamma \circ \alpha\) is extensive)
  thus, \(c \subseteq \gamma(a)\), by transitivity
- the other implication can be proved by duality
Outline

1 Abstraction

2 Abstract interpretation
   - Abstract computation
   - Fixpoint transfer

3 Application of abstract interpretation

4 Conclusion
Constructing a static analysis

We have set up a notion of abstraction:

- it describes sound approximations of concrete properties with abstract predicates
- there are several ways to formalize it (abstraction, concretization...)
- we now wish to compute sound abstract predicates

In the following, we assume

- a Galois connection

\[(C, \subseteq) \xleftrightarrow{\gamma}{\alpha} (A, \sqsubseteq)\]

- a concrete semantics \([.\.], with a constructive definition
  i.e., \([P]\) is defined by constructive equations (\([P] = f(\ldots)\)), least fixpoint formula (\([P] = \text{lfp}_\emptyset f\)...)
Abstract transformer

A fixed concrete element $c_0$ can be abstracted by $\alpha(c_0)$.

We now consider a monotone concrete function $f : C \rightarrow C$

- given $c \in C$, $\alpha \circ f(c)$ abstracts the image of $c$ by $f$
- if $c \in C$ is abstracted by $a \in A$, then $f(c)$ is abstracted by $\alpha \circ f \circ \gamma(a)$:

  - $c \subseteq \gamma(a)$ by assumption
  - $f(c) \subseteq f(\gamma(a))$ by monotonicity of $f$
  - $\alpha(f(c)) \subseteq \alpha(f(\gamma(a)))$ by monotonicity of $\alpha$

\[
\begin{align*}
A & \xrightarrow{f^\#} A \\
\gamma & \downarrow \\
\tilde{C} & \xrightarrow{f} C \\
\alpha & \uparrow
\end{align*}
\]

Definition: best and sound abstract transformers

- the best abstract transformer approximating $f$ is $f^\# = \alpha \circ f \circ \gamma$
- a sound abstract transformer approximating $f$ is any operator $f^\# : A \rightarrow A$, such that $\alpha \circ f \circ \gamma \sqsubseteq f^\#$ (or equivalently, $f \circ \gamma \subseteq \gamma \circ f^\#$)
Example: lattice of signs

- \( f : D_C^\# \rightarrow D_C^\#, c \mapsto \{-n \mid n \in c\} \)
- \( f^\# = \alpha \circ f \circ \gamma \)

Lattice of signs:

Abstract negation operator:

<table>
<thead>
<tr>
<th>( a )</th>
<th>( \ominus^#(a) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \perp )</td>
<td>( \perp )</td>
</tr>
<tr>
<td>( \equiv )</td>
<td>( \equiv )</td>
</tr>
<tr>
<td>( 0 )</td>
<td>( 0 )</td>
</tr>
<tr>
<td>( \equiv )</td>
<td>( \equiv )</td>
</tr>
<tr>
<td>( \top )</td>
<td>( \top )</td>
</tr>
</tbody>
</table>

- here, the best abstract transformer is very easy to compute
- no need to use an approximate one
Abstract interpretation

Abstract computation

Abstract $n$-ary operators

We can generalize this to $n$-ary operators, such as boolean operators and arithmetic operators.

Definition: sound and exact abstract operators

Let $g : C^n \to C$ be a monotone $n$-ary operator. Then:

- the **best abstract operator** approximating $g$ is defined by:
  \[
  g^\#: \ A^n \to A \\
  (a_0, \ldots, a_{n-1}) \mapsto \alpha \circ g\left(\gamma(a_0), \ldots, \gamma(a_{n-1})\right)
  \]

- a **sound abstract transformer** approximating $g$ is any operator $g^\#: A^n \to A$, such that
  \[
  \forall (a_0, \ldots, a_{n-1}) \in A^n, \quad \alpha \circ g\left(\gamma(a_0), \ldots, \gamma(a_{n-1})\right) \sqsubseteq g^\#(a_0, \ldots, a_{n-1})
  \]
Example: lattice of signs arithmetic operators

Application:
• ⊕ : \( C^2 \rightarrow C, (c_0, c_1) \mapsto \{ n_0 + n_1 \mid n_i \in c_i \} \)
• ⊗ : \( C^2 \rightarrow C, (c_0, c_1) \mapsto \{ n_0 \cdot n_1 \mid n_i \in c_i \} \)

Best abstract operators:

<table>
<thead>
<tr>
<th>⊕#</th>
<th>⊥</th>
<th>=</th>
<th>0</th>
<th>±</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
</tr>
<tr>
<td>=</td>
<td>⊥</td>
<td>=</td>
<td>=</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>0</td>
<td>⊥</td>
<td>=</td>
<td>0</td>
<td>±</td>
<td>T</td>
</tr>
<tr>
<td>±</td>
<td>⊥</td>
<td>T</td>
<td>±</td>
<td>±</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>⊥</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>⊗#</th>
<th>⊥</th>
<th>=</th>
<th>0</th>
<th>±</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
</tr>
<tr>
<td>=</td>
<td>⊥</td>
<td>±</td>
<td>0</td>
<td>⊥</td>
<td>T</td>
</tr>
<tr>
<td>0</td>
<td>⊥</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>±</td>
<td>⊥</td>
<td>=</td>
<td>0</td>
<td>+</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>⊥</td>
<td>T</td>
<td>0</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

Example of loss in precision:
• \( \{8\} \in \gamma_S(\pm) \) and \( \{-2\} \in \gamma_S(\mp) \)
• \( \oplus^\#(\mp, \mp) = T \) is a lot worse than \( \alpha_S(\oplus(\{8\}, \{-2\})) = \mp \)
Example: lattice of signs set operators

Best abstract operators approximating \( \cup \) and \( \cap \):

\[
\begin{array}{c|cccc}
\text{\(\cup^\#\)} & \bot & \neg & 0 & \pm & T \\
\hline
\bot & \bot & \neg & 0 & \pm & T \\
\neg & \neg & \neg & T & T & T \\
0 & 0 & T & 0 & T & T \\
\pm & \pm & T & T & \pm & T \\
T & T & T & T & T & T \\
\end{array}
\]

\[
\begin{array}{c|cccc}
\text{\(\cap^\#\)} & \bot & \neg & 0 & \pm & T \\
\hline
\bot & \bot & \bot & \bot & \bot & \bot \\
\neg & \neg & \neg & \bot & \bot & \bot \\
0 & \bot & \bot & 0 & \bot & 0 \\
\pm & \bot & \bot & \bot & \pm & \pm \\
T & \bot & \bot & \bot & \pm & T \\
\end{array}
\]

Example of loss in precision:

\[\gamma(\neg) \cup \gamma(\pm) = \{n \in \mathbb{Z} \mid n \neq 0\} \subset \gamma(T)\]
Outline

1 Abstraction

2 Abstract interpretation
   - Abstract computation
   - Fixpoint transfer

3 Application of abstract interpretation

4 Conclusion
Fixpoint transfer

What about loops? Semantic functions defined by fixpoints?

**Theorem: exact fixpoint transfer**

We consider a Galois connection $\langle C, \subseteq \rangle \leftrightarrow (A, \subseteq)$, two functions $f : C \to C$ and $f^\# : A \to A$ and two elements $c_0 \in C$, $a_0 \in A$ such that:

- $f$ is continuous
- $f^\#$ is monotone
- $\alpha \circ f = f^\# \circ \alpha$
- $\alpha(c_0) = a_0$

Then:

- both $f$ and $f^\#$ have a least-fixpoint (Tarski’s fixpoint theorem)
- $\alpha(\text{lfp}_{c_0} f) = \text{lfp}_{a_0} f^\#$
Fixpoint transfer: proof

- \( \alpha(lfp_{c_0} f) \) is a fixpoint of \( f^{\#} \) since:
  \[
  f^{\#}(\alpha(lfp_{c_0} f)) = \alpha(f(lfp_{c_0} f)) = \alpha(lfp_{c_0} f)
  \]
  since \( \alpha \circ f = f^{\#} \circ \alpha \) by definition of the fixpoints

- To show that \( \alpha(lfp_{c_0} f) \) is the least-fixpoint of \( f^{\#} \),
  we assume that \( X \) is another fixpoint of \( f^{\#} \) greater than \( a_0 \) and we show that \( \alpha(lfp_{c_0} f) \subseteq X \), i.e., that \( lfp_{c_0} f \subseteq \gamma(X) \).
  As \( lfp_{c_0} f = \bigcup_{n \in \mathbb{N}} f_0^n(c_0) \),
  it amounts to proving that
  \( \forall n \in \mathbb{N}, f_0^n(c_0) \subseteq \gamma(X) \).
  By induction over \( n \):
  - \( f^0(c_0) = c_0 \), thus \( \alpha(f^0(c_0)) = a_0 \subseteq X \); thus, \( f^0(c_0) \subseteq \gamma(X) \).
  - let us assume that \( f^n(c_0) \subseteq \gamma(X) \), and let us show that
    \( f^{n+1}(c_0) \subseteq \gamma(X) \), i.e. that \( \alpha(f^{n+1}(c_0)) \subseteq X \):
    \[
    \alpha(f^{n+1}(c_0)) = \alpha \circ f(f^n(c_0)) = f^{\#} \circ \alpha(f^n(c_0)) \subseteq f^{\#}(X) = X
    \]
    as \( \alpha(f^n(c_0)) \subseteq X \) and \( f^{\#} \) is monotone.
Abstract interpretation
Fixpoint transfer

Constructive analysis of loops

How to get a constructive fixpoint transfer theorem?

Theorem: fixpoint abstraction

Under the assumptions of the previous theorem, and with the following additional hypothesis:
- lattice $A$ is of finite height

We compute the sequence $(a_n)_{n \in \mathbb{N}}$ defined by $a_{n+1} = a_n \sqcup f^\#(a_n)$.

Then, $(a_n)_{n \in \mathbb{N}}$ converges and its limit $a_\infty$ is such that $\alpha(\text{lfp}_{c_0} f) = a_\infty$.

Proof: exercise.

Note:
- the assumptions we have made are too restrictive in practice
- more general fixpoint abstraction methods in the next lectures
Outline

1. Abstraction
2. Abstract interpretation
3. Application of abstract interpretation
4. Conclusion
Comparing existing semantics

1. A concrete semantics $\llbracket P \rrbracket$ is given: e.g., big steps operational semantics

2. An abstract semantics $\llbracket P \rrbracket^\#$ is given: e.g., denotational semantics

3. Search for an abstraction relation between them
   e.g., $\llbracket P \rrbracket^\# = \alpha(\llbracket P \rrbracket)$, or $\llbracket P \rrbracket \subseteq \gamma(\llbracket P \rrbracket^\#$

Examples:
- finite traces semantics as an abstraction of bi-finitary trace semantics
- denotational semantics as an abstraction of trace semantics
- types as an abstraction of denotational semantics
- ...

Payoff:
- better understanding of ties across semantics
- chance to generalize existing definitions
Derivation of a static analysis

1. Start from a **concrete semantics** $[P]$
2. **Choose an abstraction** defined by a Galois connection or a concretization function (usually)
3. **Derive an abstract semantics** $[P]^\#$ such that $[P] \subseteq \gamma([P]^\#)$

**Examples:**
- derivation of an analysis with a numerical lattice (constants, intervals...)
- construction of an analysis for a complex programming language

**Payoff:**
- the derivation of the abstract semantics is quite systematic
- this process offers good opportunities for a modular analysis design

There are many ways to apply abstract interpretation.
A very simple language and its semantics

We now apply this to a very simple language, and derive a static analysis step by step, from a concrete semantics and an abstraction.

- we assume a fixed set of $n$ integer variables $x_0, \ldots, x_{n-1}$
- we consider the language defined by the grammar below:

$$
P ::= x_i = n \quad \text{where } n \in \mathbb{Z}
| x_i = x_j + x_k \quad \text{basic, three-addresses arithmetics}
| x_i = x_j - x_k \quad \text{basic, three-addresses arithmetics}
| x_i = x_j \cdot x_k \quad \text{basic, three-addresses arithmetics}
| P; P \quad \text{concatenation}
| \text{input}(x_i) \quad \text{reading of a positive input}
| \text{if}(x_i > 0) P \text{ else } P
| \text{while}(x_i > 0) P
$$

- a state is a vector $\sigma = (\sigma_0, \ldots, \sigma_{n-1}) \in \mathbb{Z}^n$
- a single initial state $\sigma_{\text{init}} = (0, \ldots, 0)$
Concrete semantics

We let $[P] : \mathcal{P}(\mathbb{Z}^n) \rightarrow \mathcal{P}(\mathbb{Z}^n)$ be defined by:

- $[x_i = n](S) = \{ \sigma[i \leftarrow n] | \sigma \in S \}$
- $[x_i = x_j + x_k](S) = \{ \sigma[i \leftarrow \sigma_j + \sigma_k] | \sigma \in S \}$
- $[x_i = x_j - x_k](S) = \{ \sigma[i \leftarrow \sigma_j - \sigma_k] | \sigma \in S \}$
- $[x_i = x_j \cdot x_k](S) = \{ \sigma[i \leftarrow \sigma_j \cdot \sigma_k] | \sigma \in S \}$
- $[\text{input}(x_i)](S) = \{ \sigma[i \leftarrow n] | \sigma \in S \land n > 0 \}$
- $[[\text{if}(x_i > 0) P_0 \text{ else } P_1]](S) = \lbrack P_0 \rbrack(\{ \sigma \in S | \sigma_i > 0 \}) \cup \lbrack P_1 \rbrack(\{ \sigma \in S | \sigma_i \leq 0 \})$
- $[[\text{while}(x_i > 0) P]](S) = \{ \sigma \in \text{lfp}_S f | \sigma_i \leq 0 \}$ where $f : S' \mapsto S' \cup \lbrack P \rbrack(\{ \sigma \in S' | \sigma_i > 0 \})$

- given a complete program $P$, the **reachable states** are defined by $\lbrack P \rbrack(\{ \sigma_{\text{init}} \})$
Abstraction

We compose two abstractions:

- **non relational abstraction**: the values a variable may take is abstracted separately from the other variables
- **sign abstraction**: the set of values observed for each variable is abstracted into the lattice of signs

**Abstraction**

- **concrete domain**: \((\mathcal{P}(\mathbb{Z}^n), \subseteq)\)
- **abstract domain**: \((D^#, \sqsubseteq)\), where \(D^\# = (D^\#_S)^n\) and \(\sqsubseteq\) is the pointwise ordering
- **Galois connection** \((\mathcal{P}(\mathbb{Z}), \subseteq) \xleftarrow{\gamma} (D^#, \sqsubseteq)\), defined by

\[
\begin{align*}
\alpha : & \quad S \quad \mapsto \quad (\alpha_S(\{\sigma_0 \mid \sigma \in S\}), \ldots, \alpha_S(\{\sigma_{n-1} \mid \sigma \in S\})) \\
\gamma : & \quad S^\# \quad \mapsto \quad \{\sigma \in \mathbb{Z}^n \mid \forall i, \sigma_i \in \gamma_S(S^\#_i)\}
\end{align*}
\]
Example

Factorial function:

\[
\text{input}(x_0);
\]
\[
x_1 = 1;
\]
\[
x_2 = 1;
\]
\[
\text{while}(x_0 > 0)\{
\]
\[
x_1 = x_0 \cdot x_1;
\]
\[
x_0 = x_0 - x_2;
\]
\[
\}
\]

Abstraction of the semantics:

- abstract pre-condition: \((\top, \top, \top)\)
- abstract state before the loop: \((\pm, \pm, \pm)\)
- abstract post-condition (after the loop): \((\top, \pm, \pm)\)
Computation of the abstract semantics

We search **for an abstract semantics** $\llbracket P \rrbracket^\#: D^\# \to D^\#$ such that:

\[
\alpha \circ \llbracket P \rrbracket = \llbracket P \rrbracket^\# \circ \alpha
\]

We observe that:

\[
\alpha(S) = (\alpha_S(\{\sigma_0 \mid \sigma \in S\}), \ldots, \alpha_S(\{\sigma_{n-1} \mid \sigma \in S\}))
\]

\[
\alpha \circ \llbracket P \rrbracket(S) = (\alpha_S(\{\sigma_0 \mid \sigma \in \llbracket P \rrbracket(S)\}), \ldots, \alpha_S(\{\sigma_{n-1} \mid \sigma \in \llbracket P \rrbracket(S)\}))
\]

We start with $x_i = n$:

\[
\alpha \circ \llbracket x_i = n \rrbracket(S) = (\alpha_S(\{\sigma_0 \mid \sigma \in \llbracket P \rrbracket(\{\sigma[i \leftarrow n] \mid \sigma \in S\})\}), \ldots,
\alpha_S(\{\sigma_{n-1} \mid \sigma \in \llbracket P \rrbracket(\{\sigma[i \leftarrow n] \mid \sigma \in S\})\}))
\]

\[
= (\alpha_S(\{\sigma_0 \mid \sigma \in S\}), \ldots, \alpha_S(\{\sigma_{n-1} \mid \sigma \in S\}))[i \leftarrow \alpha_S(\{n\})]
\]

\[
= \alpha(S)[i \leftarrow \alpha_S(\{n\})]
\]

\[
= \llbracket x_i = n \rrbracket^\#(\alpha(S))
\]

where \[
\llbracket x_i = n \rrbracket^\#(S^\#) = S^\#[i \leftarrow \alpha_S(\{n\})]
\]
Computation of the abstract semantics

Other assignments are treated in a similar manner:

\[
\begin{align*}
\llbracket x_i &= x_j + x_k \rrbracket (S^\#) &= S^\#[i \leftarrow S_j^\# \oplus^\# S_k^#] \\
\llbracket x_i &= x_j - x_k \rrbracket (S^\#) &= S^\#[i \leftarrow S_j^\# \ominus^\# S_k^#] \\
\llbracket x_i &= x_j \cdot x_k \rrbracket (S^\#) &= S^\#[i \leftarrow S_j^\# \otimes^\# S_k^#] \\
\llbracket \text{input}(x_i) \rrbracket (S^\#) &= S^\#[i \leftarrow \pm]
\end{align*}
\]

Proofs are left as exercises
Computation of the abstract semantics

We now consider the case of tests:

\[
\alpha \circ \llbracket \text{if} (x_i > 0) \ P_0 \ \text{else} \ P_1 \rrbracket (S)
= \alpha(\llbracket P_0 \rrbracket (\{ \sigma \in S \mid \sigma_i > 0 \})) \cup \llbracket P_1 \rrbracket (\{ \sigma \in S \mid \sigma_i \leq 0 \})
= \alpha(\llbracket P_0 \rrbracket (\{ \sigma \in S \mid \sigma_i > 0 \})) \sqcup \alpha(\llbracket P_1 \rrbracket (\{ \sigma \in S \mid \sigma_i \leq 0 \}))
\]

as \( \alpha \) preserves least upper bounds

\[
= \llbracket P_0 \rrbracket ^\# (\alpha (\{ \sigma \in S \mid \sigma_i > 0 \})) \sqcup \llbracket P_1 \rrbracket ^\# (\alpha (\{ \sigma \in S \mid \sigma_i \leq 0 \}))
= \llbracket P_0 \rrbracket ^\# (\alpha (S) \cap \top [i \leftarrow \pm]) \sqcup \llbracket P_1 \rrbracket ^\# (\alpha (S))
= \llbracket \text{if} (x_i > 0) \ P_0 \ \text{else} \ P_1 \rrbracket ^\# (\alpha (S))
\]

where

\[
\llbracket \text{if} (x_i > 0) \ P_0 \ \text{else} \ P_1 \rrbracket ^\# (S^\#) = \llbracket P_0 \rrbracket ^\# (S^\# \cap \top [i \leftarrow \pm]) \sqcup \llbracket P_1 \rrbracket ^\# (S^\#)
\]

In the case of loops:

\[
\llbracket \text{while} (x_i > 0) \ P \rrbracket ^\# (S^\#) = \text{lfp}_{S^\#} f^\#
\]

where

\[
f^\#: S^\# \mapsto S^\# \sqcup \llbracket P \rrbracket ^\# (S^\# \cap \top [i \leftarrow \pm])
\]

Proof: exercise
Abstract semantics

Abstract semantics and soundness

We have derived the following definition of $[P]#$:

\[
\begin{align*}
[x_i = n]#(S#) &= S#[i \leftarrow \alpha_S(\{n\})] \\
[x_i = x_j + x_k]#(S#) &= S#[i \leftarrow S_j \oplus# S_k] \\
[x_i = x_j - x_k]#(S#) &= S#[i \leftarrow S_j \ominus# S_k] \\
[x_i = x_j \cdot x_k]#(S#) &= S#[i \leftarrow S_j \otimes# S_k] \\
\text{input}(x_i)]#(S#) &= S#[i \leftarrow +] \\
\text{if}(x_i > 0)\ P_0\ \text{else}\ P_1]#(S#) &= [P_0]#(S# \sqcap \top[i \leftarrow +]) \sqcup [P_1]#(S#) \\
\text{while}(x_i > 0)\ P]#(S#) &= \text{lfp}_S f\# \text{ where } f\# : S# \mapsto S# \sqcup [P]#(S# \sqcap \top[i \leftarrow +])
\end{align*}
\]

Furthermore, for all program $P$: $\alpha \circ [P] = [P]# \circ \alpha$

An over-approximation of the final states is computed by $[P]#(\top)$. 
Example

**Factorial function:**

```
input(x_0);
\text{let } x_1 = 1;
\text{let } x_2 = 1;
\text{while} (x_0 > 0)\
  \{ 
  \quad x_1 = x_0 \cdot x_1;
  \quad x_0 = x_0 - x_2;
  \}
```

**Abstract state before the loop:**

\((\bot, \bot, \bot)\)

**Iterates on the loop:**

```
<table>
<thead>
<tr>
<th>iterate</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_0</td>
<td>\pm</td>
<td>\top</td>
<td>\top</td>
</tr>
<tr>
<td>x_1</td>
<td>\pm</td>
<td>\pm</td>
<td>\pm</td>
</tr>
<tr>
<td>x_2</td>
<td>\pm</td>
<td>\pm</td>
<td>\pm</td>
</tr>
</tbody>
</table>
```

**Abstract state after the loop:**

\((\top, \pm, \pm)\)
Outline

1. Abstraction
2. Abstract interpretation
3. Application of abstract interpretation
4. Conclusion
Summary

This lecture:
- abstraction and its formalization
- computation of an abstract semantics in a very simplified case

Next lectures:
- construction of a few non trivial abstractions
- more general ways to compute sound abstract properties

Update on projects...