Abstract Interpretation
Semantics and applications to verification

Xavier Rival

École Normale Supérieure

March 30th, 2018
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**, using fixpoints as a basis
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, \( c_0 \cup c_1 \vdash_S T \)...
- 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:** \(\perp, \top, n\) where \(n \in \mathbb{Z}\)
  \((D_\mathcal{C}^\# = \{\perp, \top\} \cup \{n \mid n \in \mathbb{Z}\})\)
- **abstraction relation:** \(c \vdash c\ n \iff c \subseteq \{n\}\)

We obtain a flat lattice:

```
[-2]  [-1]  [0]  [1]  [2]  ...
```

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**: \( P(\mathcal{A}^*) \) (sets of words over alphabet \( \mathcal{A} \))
- **Abstract elements**: \( \{\perp, \top\} \cup (\mathcal{A} \rightarrow \mathbb{N}) \)
- **Abstraction relation**: \( c \vdash_{\mathcal{P}} \phi : \mathcal{A} \rightarrow \mathbb{N} \) if and only if:

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

**Abstract reasoning:**

- **Concatenation:**

\[
\text{if } \phi_0, \phi_1 : \mathcal{A} \rightarrow \mathbb{N} \text{ and } c_0, c_1 \text{ are such that } c_i \vdash_{\mathcal{P}} \phi_i,
\]

\[
\{w_0 \cdot w_1 \mid w_i \in c_i\} \vdash_{\mathcal{P}} \phi_0 + \phi_1
\]

**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:** $\bot, (a, b)$ where $a \in \{-\infty\} \cup \mathbb{Z}$, $b \in \mathbb{Z} \cup \{+\infty\}$ and $a \leq b$
- **abstraction relation:**

  \[
  \emptyset \vdash_{\mathcal{I}} \bot \\
  S \vdash_{\mathcal{I}} \top \\
  S \vdash_{\mathcal{I}} (a, b) \iff \forall x \in S, \ a \leq x \leq b
  \]

Operations: **TD**
Abstraction example 5: non relational abstraction

**Definition: non relational abstraction**

- **concrete elements:** $\mathcal{P}(X \to Y)$, inclusion ordering
- **abstract elements:** $X \to \mathcal{P}(Y)$, 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**
Intuitively, the abstraction relation also describes implication:
\( c \vdash a \) effectively means “the property described by \( a \) implies that described by \( c \)

**Advantage on static analysis** (hint about the following lectures):
- abstract predicates are a lot easier to manipulate than sets of concrete states or logical formulas
- we can still derive concrete facts from abstract predicates
Abstraction relation and monotonicity

Order relations, abstraction relation and monotonicity
- both orders and the abstraction relation describe ordering
- we derive from transitivity there monotonicity properties
  i.e., chains of implications compose

Abstraction relation: \( c \vdash a \) 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 \( a_0 \subseteq a_1 \) 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:

\[
\begin{align*}
\forall c_0, c_1 \in C, \forall a \in A, \ & c_0 \subseteq c_1 \land c_1 \vdash a \implies c_0 \vdash a \\
\forall c \in C, \forall a_0, a_1 \in A, \ & c \vdash a_0 \land a_0 \sqsubseteq a_1 \implies c \vdash a_1
\end{align*}
\]

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) is a monotone function that maps abstract $a$ into the weakest (i.e., most general) concrete $c$ that satisfies $a$ (i.e., $c \vdash a$).

**Notes:**

- in common cases, there exists a $\gamma$
- $c \vdash a$ if and only if $c \subseteq \gamma(a)$
- a concretization that is not monotone with respect to the “logical ordering” would not make sense
- in fact, in some cases, we will even define $\gamma$ before we define an ordering, and let $\gamma$ define the ordering!
Concretization function: a few examples

**Signs abstraction:**

\[
\gamma_S : \begin{array}{c}
\top \mapsto \mathbb{Z} \\
\pm \mapsto \mathbb{Z}^* \\
0 \mapsto \{0\} \\
\neg \mapsto \mathbb{Z}^* \\
\bot \mapsto \emptyset
\end{array}
\]

**Constants abstraction:**

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

**Non relational abstraction:**

\[
\gamma_{NR} : (X \rightarrow \mathcal{P}(Y)) \mapsto \mathcal{P}(X \rightarrow Y) \\
\Phi \mapsto \{\phi : X \rightarrow Y \mid \forall x \in X, \phi(x) \in \Phi(x)\}
\]

**Parikh vector abstraction:** exercise!
Abstraction function

Our second adjoint function:

**Definition: abstraction function**

An **abstraction function** \( \alpha : C \rightarrow A \) (if it exists) is a monotone function that 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 \( \alpha \)
- for the same reason as \( \gamma \) a non monotone \( \alpha \) (with respect to logical ordering) would not make sense

**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
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:**

\[ \alpha_{\mathcal{N}R} : \ \mathcal{P}(X \rightarrow Y) \mapsto X \rightarrow \mathcal{P}(Y) \]

\[ c \mapsto (x \in X) \mapsto \{\phi(x) \mid \phi \in c\} \]

**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
Tying definitions of abstraction relation

So far, we have:

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

How to tie them together?

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

This means:

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

This observation is at the basis of the definition of **Galois connections**
Galois connection

Definition: Galois connection

A Galois connection is defined by a:
- 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: \((C, \subseteq) \xleftarrow{\alpha} (A, \sqsubseteq)\)

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

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

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

\[
\gamma_C(t) \mapsto \mathbb{Z} \\
\gamma_C(n) \mapsto \{n\} \\
\gamma_C(\bot) \mapsto \emptyset
\]

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 \( c = \{n\} \), \( \alpha_C(\{n\}) = n \subseteq c \iff c = n \lor c = \top \iff c = \{n\} \subseteq \gamma_C(a) \)
- 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**

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

We have defined:

\[ \alpha_{NR} : (c \subseteq (X \to Y)) \mapsto (x \in X) \mapsto \{ f(x) \mid f \in c \} \]
\[ \gamma_{NR} : (\Phi \in (X \to P(Y))) \mapsto \{ f : X \to Y \mid \forall x \in X, f(x) \in \Phi(x) \} \]

Let \( c \in \mathcal{P}(X \to Y) \) and \( \Phi \in (X \to \mathcal{P}(Y)) \); then:

\[ \alpha_{NR}(c) \subseteq \Phi \iff \forall x \in X, \alpha_{NR}(c)(x) \subseteq \Phi(x) \]
\[ \iff \forall x \in X, \{ f(x) \mid f \in c \} \subseteq \Phi(x) \]
\[ \iff \forall f \in c, \forall x \in X, f(x) \in \Phi(x) \]
\[ \iff \forall f \in c, \ f \in \gamma_{NR}(\Phi) \]
\[ \iff c \subseteq \gamma_{NR}(\Phi) \]

Non relational abstraction: Galois connection

\( c \subseteq \gamma_{NR}(a) \iff \alpha_{NR}(c) \subseteq a \), therefore,

\[ (\mathcal{P}(X \to Y), \subseteq) \xrightarrow{\gamma_{NR}} (X \to \mathcal{P}(Y), \subseteq) \xleftarrow{\alpha_{NR}} (X \to \mathcal{P}(Y), \subseteq) \]
Galois connection properties

Galois connections have many useful properties.

In the next few slides, we consider a Galois connection \((C, \subseteq) \xleftarrow{\gamma} (A, \sqsubseteq)\) and establish a few interesting properties.

**Extensivity, contractivity**

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

**Proof:**

- let \(a \in A\); then, \(\gamma(a) \subseteq \gamma(a)\), thus \(\alpha(\gamma(a)) \sqsubseteq a\)
- let \(c \in C\); then, \(\alpha(c) \sqsubseteq \alpha(c)\), thus \(c \subseteq \gamma(\alpha(c))\)
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 $(A, \sqsupseteq) \xrightarrow{\alpha} (C, \supseteq)$
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:
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:

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

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 \( a \in A \):

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

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

**Uniqueness of adjoints**
- given $\gamma : A \to C$, there exists at most one $\alpha : C \to A$ such that $(C, \subseteq) \xleftrightarrow{\gamma} (A, \sqsubseteq)$, and, if it exists, $\alpha(c) = \sqcap\{a \in A \mid c \subseteq \gamma(a)\}$
- similarly, given $\alpha : C \to A$, there exists at most one $\gamma : A \to C$ such that $(C, \subseteq) \xleftrightarrow{\alpha} (A, \sqsubseteq)$, 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, $\alpha(c) = \sqcap\{a \in A \mid c \subseteq \gamma(a)\}$ for a given $c \in C$.

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

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 polyedra, abstracting sets of points in \(\mathbb{R}^2\).

Exercise: state the dual property and apply the same principle to the concretization
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

\[
(C, \subseteq) \leftrightarrow (A, \sqsubseteq)
\]

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**
    
    $$(C, \subseteq) \xleftarrow{\gamma} (A, \sqsubseteq)$$
  
  - a **concrete semantics $\llbracket . \rrbracket$**, with a **constructive definition**
    
    i.e., $\llbracket P \rrbracket$ is defined by constructive equations ($\llbracket P \rrbracket = f(\ldots)$), least fixpoint formula ($\llbracket P \rrbracket = \text{lfp}_{\emptyset} f$)
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 $\alpha$

\[\begin{array}{c}
A \xrightarrow{f^\#} A \\
\gamma \downarrow \quad \alpha \\
\tilde{C} \xrightarrow{f} C
\end{array}\]

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, $f \circ \gamma \sqsubseteq \gamma \circ f^\#$)
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>( \pm )</td>
</tr>
<tr>
<td>( 0 )</td>
<td>( 0 )</td>
</tr>
<tr>
<td>( \pm )</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 an \( n \)-ary operator, monotone in each component. Then:

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

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

Application:

- $\oplus : C^2 \to C, (c_0, c_1) \mapsto \{n_0 + n_1 \mid n_i \in c_i\}$
- $\otimes : C^2 \to C, (c_0, c_1) \mapsto \{n_0 \cdot n_1 \mid n_i \in c_i\}$

Best abstract operators:

<table>
<thead>
<tr>
<th>$\oplus^#$</th>
<th>$\bot$</th>
<th>$-$</th>
<th>$0$</th>
<th>$\pm$</th>
<th>$T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
</tr>
<tr>
<td>$-$</td>
<td>$\bot$</td>
<td>$-$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
</tr>
<tr>
<td>$0$</td>
<td>$\bot$</td>
<td>$-$</td>
<td>$0$</td>
<td>$\pm$</td>
<td>$T$</td>
</tr>
<tr>
<td>$\pm$</td>
<td>$\bot$</td>
<td>$T$</td>
<td>$\pm$</td>
<td>$\pm$</td>
<td>$T$</td>
</tr>
<tr>
<td>$T$</td>
<td>$\bot$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\otimes^#$</th>
<th>$\bot$</th>
<th>$-$</th>
<th>$0$</th>
<th>$\pm$</th>
<th>$T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
</tr>
<tr>
<td>$-$</td>
<td>$\bot$</td>
<td>$+$</td>
<td>$0$</td>
<td>$-$</td>
<td>$T$</td>
</tr>
<tr>
<td>$0$</td>
<td>$\bot$</td>
<td>$0$</td>
<td>$0$</td>
<td>$0$</td>
<td>$0$</td>
</tr>
<tr>
<td>$\pm$</td>
<td>$\bot$</td>
<td>$-$</td>
<td>$0$</td>
<td>$+$</td>
<td>$T$</td>
</tr>
<tr>
<td>$T$</td>
<td>$\bot$</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 $\{-2\} \in \gamma_S(-)$
- $\oplus^\#(+, -) = T$ is a lot worse than $\alpha_S(\oplus(\{8\}, \{-2\})) = \pm$
Example: lattice of signs set operators

Best abstract operators approximating $\cup$ and $\cap$:

$$\begin{array}{|c|c|c|c|c|}
\hline
\cup^\# & \bot & \neg & 0 & \pm & \top \\
\hline
\bot & \bot & \neg & 0 & \pm & \top \\
\neg & \neg & \neg & \top & \top & \top \\
0 & 0 & \top & 0 & \top & \top \\
\pm & \pm & \top & \top & \pm & \top \\
\top & \top & \top & \top & \top & \top \\
\hline
\end{array}$$

$$\begin{array}{|c|c|c|c|c|}
\hline
\cap^\# & \bot & \neg & 0 & \pm & \top \\
\hline
\bot & \bot & \bot & \bot & \bot & \bot \\
\neg & \neg & \neg & \bot & \bot & \bot \\
0 & 0 & \bot & 0 & \bot & \bot \\
\pm & \bot & \bot & \bot & \pm & \pm \\
\top & \bot & \neg & 0 & \pm & \top \\
\hline
\end{array}$$

Example of loss in precision:

- $\gamma(\neg) \cup \gamma(\pm) = \{n \in \mathbb{Z} \mid n \neq 0\} \subset \gamma(\top)$
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 assume \((C, \subseteq)\) and \((A, \sqsubseteq)\) are complete lattices. We consider a Galois connection \((C, \subseteq) \xleftrightarrow{\gamma} (A, \sqsubseteq)\), two functions \(f : C \rightarrow C\) and \(f^\# : A \rightarrow A\) 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** (by Tarski’s fixpoint theorem)
- \(\alpha(\text{lfp}_{c_0} f) = \text{lfp}_{a_0} f^\#\)
Fixpoint transfer: proof

- $\alpha(\text{lfp}_{c_0} f)$ is a fixpoint of $f^\#$ since:

$$f^\#(\alpha(\text{lfp}_{c_0} f)) = \alpha(f(\text{lfp}_{c_0} f))$$

since $\alpha \circ f = f^\# \circ \alpha$

$$= \alpha(\text{lfp}_{c_0} f)$$

by definition of the fixpoints

- To show that $\alpha(\text{lfp}_{c_0} f)$ is the least-fixpoint of $f^\#$,
we assume that $X$ is another fixpoint of $f^\#$ greater than $a_0$ and we show that $\alpha(\text{lfp}_{c_0} f) \subseteq X$, i.e., that $\text{lfp}_{c_0} f \subseteq \gamma(X)$. As $\text{lfp}_{c_0} f = \bigcup_{n \in \mathbb{N}} f^n(c_0)$ (by Kleene’s fixpoint theorem), it amounts to proving that $\forall n \in \mathbb{N}, f^n(c_0) \subseteq \gamma(X)$. 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.
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 $\alpha(lfp_{c_0} f) = a_\infty$.

**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** $[P]$ is given: e.g., big steps operational semantics

2. An **abstract semantics** $[P]^\#$ is given: e.g., denotational semantics

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

**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

Example: we have seen the tie between reachable states and denotational semantics.
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.
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:

\[
P ::= \begin{align*}
x_i &= n \\
x_i &= x_j + x_k \\
x_i &= x_j - x_k \\
x_i &= x_j \cdot x_k \\
\text{P; P} \\
\text{input}(x_i) \\
\text{if}(x_i > 0) \ P \ 	ext{else} \ P \\
\text{while}(x_i > 0) \ P
\end{align*}
\]

where \( n \in \mathbb{Z} \)

- **basic, three-addresses arithmetics**
- **concatenation**
- **reading of a positive input**

- 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:

\[
\begin{align*}
\llbracket x_i = n \rrbracket (\mathcal{M}) &= \{ \sigma[i \leftarrow n] \mid \sigma \in \mathcal{M} \} \\
\llbracket x_i = x_j + x_k \rrbracket (\mathcal{M}) &= \{ \sigma[i \leftarrow \sigma_j + \sigma_k] \mid \sigma \in \mathcal{M} \} \\
\llbracket x_i = x_j - x_k \rrbracket (\mathcal{M}) &= \{ \sigma[i \leftarrow \sigma_j - \sigma_k] \mid \sigma \in \mathcal{M} \} \\
\llbracket x_i = x_j \times x_k \rrbracket (\mathcal{M}) &= \{ \sigma[i \leftarrow \sigma_j \times \sigma_k] \mid \sigma \in \mathcal{M} \} \\
\llbracket \text{input}(x_i) \rrbracket (\mathcal{M}) &= \{ \sigma[i \leftarrow n] \mid \sigma \in \mathcal{M} \land n > 0 \} \\
\llbracket P_0; P_1 \rrbracket (\mathcal{M}) &= \llbracket P_1 \rrbracket \circ \llbracket P_0 \rrbracket (\mathcal{M}) \\
\llbracket \text{if}(x_i > 0) P_0 \text{ else } P_1 \rrbracket (\mathcal{M}) &= \llbracket P_0 \rrbracket (\{ \sigma \in \mathcal{M} \mid \sigma_i > 0 \}) \\
&\quad \cup \llbracket P_1 \rrbracket (\{ \sigma \in \mathcal{M} \mid \sigma_i \leq 0 \}) \\
\llbracket \text{while}(x_i > 0) P \rrbracket (\mathcal{M}) &= \{ \sigma \in \text{lfp } f \mid \sigma_i \leq 0 \} \text{ where } \\
f : \mathcal{M}' \mapsto \mathcal{M} \cup \mathcal{M}' \cup \llbracket P \rrbracket (\{ \sigma \in \mathcal{M}' \mid \sigma_i > 0 \})
\end{align*}
\]

- given a complete program \(P\), the **reachable states** are defined by \([P](\{ \sigma_{\text{init}} \})\)

---

Application of abstract interpretation

Concrete semantics

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

\[
\begin{align*}
\llbracket x_i = n \rrbracket (\mathcal{M}) &= \{ \sigma[i \leftarrow n] \mid \sigma \in \mathcal{M} \} \\
\llbracket x_i = x_j + x_k \rrbracket (\mathcal{M}) &= \{ \sigma[i \leftarrow \sigma_j + \sigma_k] \mid \sigma \in \mathcal{M} \} \\
\llbracket x_i = x_j - x_k \rrbracket (\mathcal{M}) &= \{ \sigma[i \leftarrow \sigma_j - \sigma_k] \mid \sigma \in \mathcal{M} \} \\
\llbracket x_i = x_j \times x_k \rrbracket (\mathcal{M}) &= \{ \sigma[i \leftarrow \sigma_j \times \sigma_k] \mid \sigma \in \mathcal{M} \} \\
\llbracket \text{input}(x_i) \rrbracket (\mathcal{M}) &= \{ \sigma[i \leftarrow n] \mid \sigma \in \mathcal{M} \land n > 0 \} \\
\llbracket P_0; P_1 \rrbracket (\mathcal{M}) &= \llbracket P_1 \rrbracket \circ \llbracket P_0 \rrbracket (\mathcal{M}) \\
\llbracket \text{if}(x_i > 0) P_0 \text{ else } P_1 \rrbracket (\mathcal{M}) &= \llbracket P_0 \rrbracket (\{ \sigma \in \mathcal{M} \mid \sigma_i > 0 \}) \\
&\quad \cup \llbracket P_1 \rrbracket (\{ \sigma \in \mathcal{M} \mid \sigma_i \leq 0 \}) \\
\llbracket \text{while}(x_i > 0) P \rrbracket (\mathcal{M}) &= \{ \sigma \in \text{lfp } f \mid \sigma_i \leq 0 \} \text{ where } \\
f : \mathcal{M}' \mapsto \mathcal{M} \cup \mathcal{M}' \cup \llbracket P \rrbracket (\{ \sigma \in \mathcal{M}' \mid \sigma_i > 0 \})
\end{align*}
\]

- given a complete program \(P\), the **reachable states** are defined by \([P](\{ \sigma_{\text{init}} \})\)
Examples

A couple of contrived examples
enough to show the behavior of the analysis...

Absolute value function:

```plaintext
if (x_0 > 0) { 
    x_1 = x_0;
} else { 
    x_2 = 0;
    x_1 = x_2 - x_0;
}
```

- input unknowns
- output \( x_1 \) should be positive

Factorial function:

```plaintext
input(x_0);
    x_1 = 1;
    x_2 = 1;
while (x_0 > 0) { 
    x_1 = x_0 * x_1;
    x_0 = x_0 - x_2;
}
```

- input unknowns
- output \( x_0 \) should be null
- outputs \( x_1, x_2 \) should be positive
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**: $(D^\#, \subseteq)$, where $D^\# = (D^\#_S)^n$ and $\subseteq$ is the pointwise ordering
- **Galois connection** $(\mathcal{P}(\mathbb{Z}), \subseteq) \xleftarrow{\gamma} (D^\#, \subseteq)$, defined by

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

  \[
  \gamma : M^\# \mapsto \{\sigma \in \mathbb{Z}^n \mid \forall i, \sigma_i \in \gamma_S(M^\#_i)\}\]
Towards an abstraction for our small language

Basic intuitions for our abstraction:

1. a memory state is a vector of scalars

2. the concrete semantics is a function, that maps a concrete pre-condition to an abstract post-condition

3. sign lattice abstract elements abstract sets of values

4. an abstract state should thus consist of a vector of abstract values

5. moreover, the abstract semantics should consist of a function that maps an abstract pre-condition into an abstract post-condition
Examples

Absolute value function:

```java
if (x0 > 0) {
    x1 = x0;
} else {
    x2 = 0;
    x1 = x2 - x0;
}
```

- abstract pre-condition: $(\top, \top)$
- abstract post-condition: $(\top, \pm)$

Factorial function:

```java
input(x0);
```

```java
x1 = 1;
x2 = 1;
while (x0 > 0) {
    x1 = x0 * x1;
    x0 = x0 - x2;
}
```

- abstract pre-condition: $(\top, \top, \top)$
- abstract state before the loop: $(\pm, \pm, \pm)$
- abstract post-condition (after the loop): $(0, \pm, \pm)$
We search for an abstract semantics \([P]^{\#} : D^{\#} \rightarrow D^{\#}\) such that:

\[\alpha \circ [P] \sqsubseteq [P]^{\#} \circ \alpha\]

We aim for a proof by induction over the syntax of programs.

So, let us start with sequences / composition, under the assumption that the property holds for \(P_0, P_1\):

\[\alpha \circ [P_0] \sqsubseteq [P_0]^{\#} \circ \alpha\]
\[\alpha \circ [P_1] \sqsubseteq [P_1]^{\#} \circ \alpha\]

Since \([P_0; P_1] = [P_1] \circ [P_0]\), we expect \([P_0; P_1]^{\#} = [P_1]^{\#} \circ [P_0]^{\#}\):

\[\alpha \circ [P_1] \circ [P_0] \sqsubseteq [P_1]^{\#} \circ \alpha \circ [P_0] \quad \text{by induction}\]
\[\sqsubseteq [P_1]^{\#} \circ [P_0]^{\#} \circ \alpha \quad \text{by induction}\ldots\]
and if \([P_1]^{\#}\) monotone!

Big additional constraint (only today): \([P]^{\#}\) monotone
Analysis of assignment

We now consider the analysis of assignment statements.

We observe that:

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

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

We start with \( x_i = n \):

\[ \alpha \circ [x_i = n](M) \]

\[ = (\alpha_S(\{\sigma_0 \mid \sigma \in [P](\{\sigma[i \leftarrow n] \mid \sigma \in M\})\}), \ldots, \]

\[ \quad \alpha_S(\{\sigma_{n-1} \mid \sigma \in [P](\{\sigma[i \leftarrow n] \mid \sigma \in S\})\})) \]

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

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

\[ = [x_i = n](\alpha(M)) \]

where:

\[ [x_i = n](\alpha(M)) = \alpha(M)[i \leftarrow \alpha_S(\{n\})] \]
Computation of the abstract semantics

Other assignments are treated in a similar manner:

\[
\begin{align*}
\llbracket x_i = n \rrbracket^\#(M^\#) &= M^\#[i \leftarrow \alpha_S\{n\}] \\
\llbracket x_i = x_j + x_k \rrbracket^\#(M^\#) &= M^\#[i \leftarrow M_j^\# \oplus M_k^\#] \\
\llbracket x_i = x_j - x_k \rrbracket^\#(M^\#) &= M^\#[i \leftarrow M_j^\# \ominus M_k^\#] \\
\llbracket x_i = x_j \times x_k \rrbracket^\#(M^\#) &= M^\#[i \leftarrow M_j^\# \otimes M_k^\#] \\
\llbracket \text{input}(x_i) \rrbracket^\#(M^\#) &= M^\#[i \leftarrow \pm]
\end{align*}
\]

- Proofs are left as exercises
- As remarked before, we only get \( \alpha \circ [P] \subseteq [P]^\# \circ \alpha \)
  i.e., equality is too hard to derive
- On the other hand, monotonicity is good so far (exercise)
Computation of the abstract semantics

We now consider the case of tests:

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

as \( \alpha \) preserves least upper bounds

\[ \sqsubseteq \llbracket P_0 \rrbracket^\#(\alpha(\{\sigma \in M \mid \sigma_i > 0\})) \cup \llbracket P_1 \rrbracket^\#(\alpha(\{\sigma \in M \mid \sigma_i \leq 0\})) \]
by induction and as \( \cup \) is monotone

\[ \sqsubseteq \llbracket P_0 \rrbracket^\#(\alpha(M) \cap \top[i \leftarrow +]) \cup \llbracket P_1 \rrbracket^\#(\alpha(M) \cap \top[i \leftarrow \leq 0]) \]
\[ \sqsubseteq \llbracket \text{if}(x_i > 0) \ P_0 \ \text{else} \ P_1 \rrbracket^\#(\alpha(M)) \]

where:

\[ \llbracket \text{if}(x_i > 0) \ P_0 \ \text{else} \ P_1 \rrbracket^\#(M^\#) = \]
\[ \llbracket P_0 \rrbracket^\#(M^\# \cap \top[i \leftarrow +]) \cup \llbracket P_1 \rrbracket^\#(M^\# \cap \top[i \leftarrow \leq 0]) \]

- Monotonicity: by induction...
Application of abstract interpretation

An example with basic condition test

Absolute value function:

```latex
\text{if}(x_0 > 0)\{
    x_1 = x_0;
\}
\text{else}\{
    x_2 = 0;
    x_1 = x_2 - x_0;
\}
```

Analysis steps:

1. entry point: $(\top, \top)$
2. after entry in true branch: $(\pm, \top)$
3. exit of true branch: $(\pm, -)$
4. after entry in false branch: $(\leq 0, \top)$
5. exit of false branch: $(\leq 0, \geq 0)$
6. exit: $(\top, \geq 0)$
Analysis of a loop

We have seen that:

\[
\llbracket \text{while}(x_i > 0) \ P \rrbracket (\mathcal{M}) = \{ \sigma \in \text{lfp}\ f \mid \sigma_i \leq 0 \}
\]

where \( f(\mathcal{M}') = \mathcal{M} \cup \mathcal{M}' \cup \llbracket P \rrbracket (\{ \sigma \in \mathcal{M}' \mid \sigma_i > 0 \}) \).

Thus, we look for a fixpoint transfer, but our fixpoint transfer theorem requires equality, so it does not apply...

We will use a variant of the previous theorem:

If:

- \( f \) is continuous
- \( f^\# \) is monotone
- \( \alpha \circ f \subseteq f^\# \circ \alpha \)
- \( \alpha(\emptyset) = \bot \)

Then, \( \alpha(\text{lfp}\ f) \subseteq \text{lfp}\ f^\# \)
Analysis of a loop

Application:

- we consider the analysis of the loop with pre-condition $M^\#$
- we take
  
  \[ f^\#(M^\#_0) = M^\# \cup M^\#_0 \cup [P]^\#(M^\#_0 \cap T[i \leftarrow \pm]) \]

- then, $\alpha \circ f \subseteq f^\# \circ \alpha$
- we can apply the new fixpoint transfer theorem...

\[
\begin{align*}
\llbracket \text{while}(x_i > 0) \ P \rrbracket^\#(M^\#) &= T[i \leftarrow \leq 0] \cap \text{lfp}_{\mathcal{M}^\#} f^\#
\end{align*}
\]
\[
\text{where } f^\#(M^\#_0) = M^\# \cup M^\#_0 \cup [P]^\#(M^\#_0 \cap T[i \leftarrow \pm])
\]

One more thing:

- we need to prove monotonicity of the fixpoint image since the whole abstract semantics soundness relies on it!
Abstract semantics and soundness

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

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

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:

\[
\text{input}(x_0);
x_1 = 1;
x_2 = 1;
\text{while}(x_0 > 0)\
\quad\begin{align*}
x_1 &= x_0 \cdot x_1; \\
x_0 &= x_0 - x_2;
\end{align*}
\]

Abstract state before the loop: \((\pm, \pm, \pm)\)

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...