Abstract Interpretation

Semantics and applications to verification

Xavier Rival

École Normale Supérieure

April 4th, 2025

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 guide

Outline

1. Abstraction
   - Notion of abstraction
   - Abstraction and concretization functions
   - Galois connections
2. Abstract interpretation
3. Application of abstract interpretation
4. Conclusion

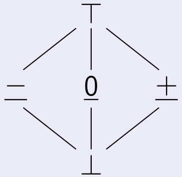
Abstraction example 1: signs

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 $\emptyset$
- $\underline{+}$ denotes any set of positive integers
- $\underline{0}$ denotes any subset of $\{0\}$
- $\underline{-}$ 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: "$\cdot \in \{\underline{+}, \underline{0}, \ldots\}$")
- **abstraction relation:** $c \vdash_{\mathcal{S}} a$ when a describes c

Examples:

- $\{1, 2, 3, 5, 7, 11, 13, 17, 19, 23, \ldots\} \vdash_{\mathcal{S}} \underline{+}$
- $\{1, 2, 3, 5, 7, 11, 13, 17, 19, 23, \ldots\} \vdash_{\mathcal{S}} \top$

We use abstract elements **to reason about operations**:

- if $c_0 \vdash_{\mathcal{S}} \underline{+}$ and $c_1 \vdash_{\mathcal{S}} \underline{+}$, then $\{x_0 + x_1 \mid x_i \in c_i\} \vdash_{\mathcal{S}} \underline{+}$
- if $c_0 \vdash_{\mathcal{S}} \underline{+}$ and $c_1 \vdash_{\mathcal{S}} \underline{+}$, then $\{x_0 \cdot x_1 \mid x_i \in c_i\} \vdash_{\mathcal{S}} \underline{+}$
- if $c_0 \vdash_{\mathcal{S}} \underline{+}$ and $c_1 \vdash_{\mathcal{S}} \underline{0}$, then $\{x_0 \cdot x_1 \mid x_i \in c_i\} \vdash_{\mathcal{S}} \underline{0}$
- if $c_0 \vdash_{\mathcal{S}} \underline{+}$ and $c_1 \vdash_{\mathcal{S}} \bot$, then $\{x_0 \cdot x_1 \mid x_i \in c_i\} \vdash_{\mathcal{S}} \bot$

Abstraction example 1: signs

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

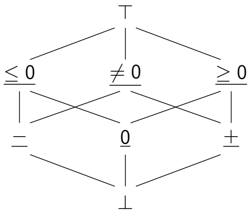
- if $c_0 \vdash_{\mathcal{S}} \underline{+}$ and $c_1 \vdash_{\mathcal{S}} \underline{+}$, then $c_0 \cup c_1 \vdash_{\mathcal{S}} \underline{+}$
- if $c_0 \vdash_{\mathcal{S}} \underline{+}$ and $c_1 \vdash_{\mathcal{S}} \bot$, then $c_0 \cup c_1 \vdash_{\mathcal{S}} \underline{+}$

But, what can we say about $c_0 \cup c_1$, when $c_0 \vdash_{\mathcal{S}} \underline{0}$ **and** $c_1 \vdash_{\mathcal{S}} \underline{+}$?

- clearly, $c_0 \cup c_1 \vdash_{\mathcal{S}} \top$...
- but **no other relation holds**
- in the abstract, **we do not rule out negative values**

We can **extend the initial lattice**:

- $\underline{\geq 0}$ denotes any set of positive or null integers
- $\underline{\leq 0}$ denotes any set of negative or null integers
- $\underline{\neq 0}$ denotes any set of non null integers
- if $c_0 \vdash_{\mathcal{S}} \underline{+}$ and $c_1 \vdash_{\mathcal{S}} \underline{0}$, then $c_0 \cup c_1 \vdash_{\mathcal{S}} \underline{\geq 0}$

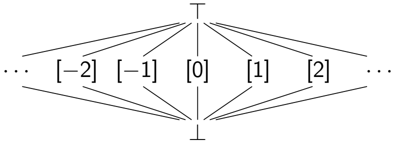


Abstraction example 2: constants

Definition: abstraction based on constants

- **concrete elements:** $\mathcal{P}(\mathbb{Z})$
- **abstract elements:** $\bot, \top, \underline{n}$ where $n \in \mathbb{Z}$
  $(D^\sharp_\mathcal{C} = \{\bot, \top\} \cup \{\underline{n} \mid n \in \mathbb{Z}\})$
- **abstraction relation:** $c \vdash_\mathcal{C} \underline{n} \iff c \subseteq \{n\}$

We obtain a **flat lattice**:



Abstract reasoning:

- if $c_0 \vdash_\mathcal{C} \underline{n_0}$ and $c_1 \vdash_\mathcal{C} \underline{n_1}$, then $\{k_0 + k_1 \mid k_i \in c_i\} \vdash_\mathcal{C} \underline{n_0 + n_1}$

Abstraction example 3: Parikh vector

Definition: Parikh vector abstraction

- **concrete elements:** $\mathcal{P}(\mathcal{A}^\star)$ (sets of words over alphabet $\mathcal{A}$)
- **abstract elements:** $\{\bot, \top\} \cup (\mathcal{A} \to \mathbb{N})$
- **abstraction relation:** $c \vdash_{\mathfrak{P}} \phi : \mathcal{A} \to \mathbb{N}$ 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 $\phi_0, \phi_1 : \mathcal{A} \to \mathbb{N}$ and c_0, c_1 are such that $c_i \vdash_{\mathfrak{P}} \phi_i$,

$$\{w_0 \cdot w_1 \mid w_i \in c_i\} \vdash_{\mathfrak{P}} \phi_0 + \phi_1$$

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:**

$$\begin{array}{l} \emptyset \vdash_{\mathcal{I}} \bot \\ S \vdash_{\mathcal{I}} \top \\ S \vdash_{\mathcal{I}} (a, b) \iff \forall x \in S,\ a \leq x \leq b \end{array}$$

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:** $c \vdash_{\mathcal{NR}} a \iff \forall \phi \in c, \forall x \in X, \phi(x) \in a(x)$

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:

$$\forall \phi \in c, \phi(x_0) = \phi(x_1)$$
$$\forall \phi \in c, \forall x, x' \in X, \phi(x) \neq y_0 \vee \phi(x') \neq y_1$$

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** c **implies that described by** a

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 $a_0 \sqsubseteq a_1$ 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

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

- $\forall c_0, c_1 \in C, \forall a \in A,\ c_0 \subseteq c_1 \wedge c_1 \vdash a \Longrightarrow c_0 \vdash a$
- $\forall c \in C, \forall a_0, a_1 \in A,\ c \vdash a_0 \wedge a_0 \sqsubseteq a_1 \Longrightarrow c \vdash a_1$

When we have a c (resp., a) and try to map it into a compatible a (resp., into a compatible a c), **the abstraction relation is convenient**.

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 $\gamma : A \to C$ (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 γ
- $c \vdash a$ 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 γ before we define an ordering, and let γ define the ordering!

Concretization function: a few examples

Signs abstraction:

$$\begin{array}{rccc} \gamma_{\mathcal{S}} : & \top & \longmapsto & \mathbb{Z} \\ & \underline{+} & \longmapsto & \mathbb{Z}^{\star}_{+} \\ & \underline{0} & \longmapsto & \{0\} \\ & \underline{-} & \longmapsto & \mathbb{Z}^{\star}_{-} \\ & \bot & \longmapsto & \emptyset \end{array}$$

Constants abstraction:

$$\begin{array}{rccc} \gamma_{\mathcal{C}} : & \top & \longmapsto & \mathbb{Z} \\ & \underline{n} & \longmapsto & \{n\} \\ & \bot & \longmapsto & \emptyset \end{array}$$

Non relational abstraction:

$$\begin{array}{rlcl} \gamma_{\mathcal{NR}} : & (X \rightarrow \mathcal{P}(Y)) & \longrightarrow & \mathcal{P}(X \rightarrow Y) \\ & \Phi & \longmapsto & \{\phi : X \rightarrow Y \mid \forall x \in X,\ \phi(x) \in \Phi(x)\} \end{array}$$

Parikh vector abstraction: exercise!

Abstraction function

Our **second adjoint function:**

Definition: abstraction function

An **abstraction function** $\alpha : C \to A$ (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 α
- for the same reason as γ a non monotone α (with respect to logical ordering) would not make sense

Summary on adjoint functions:

- α (**called abstraction**) maps any concrete element **to the most precise abstract predicate** that holds true for it
- γ (**called concretisation**) returns the **most general concrete meaning** of its argument

Abstraction: a few examples

Constants abstraction:

$$\alpha_{\mathcal{C}} : \quad (c \subseteq \mathbb{Z}) \longmapsto \begin{cases} \bot & \text{if } c = \emptyset \\ \underline{n} & \text{if } c = \{n\} \\ \top & \text{otherwise} \end{cases}$$

Non relational abstraction:

$$\begin{array}{rlcl} \alpha_{\mathcal{NR}} : & \mathcal{P}(X \rightarrow Y) & \longrightarrow & X \rightarrow \mathcal{P}(Y) \\ & c & \longmapsto & (x \in X) \mapsto \{\phi(x) \mid \phi \in c\} \end{array}$$

Signs abstraction and **Parikh vector abstraction**: exercises

Outline

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:

$$\begin{array}{l} \forall c \in C,\ \forall a \in A, \\ \quad c \vdash a \\ \qquad \iff c \subseteq \gamma(a) \\ \qquad \iff \alpha(c) \sqsubseteq a \end{array}$$

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** $\alpha : C \to A$
- and a **concretization function** $\gamma : A \to C$

such that:

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

Notation: $$(C, \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \sqsubseteq)$$

Note: in practice, we shall rarely use $\vdash$; we use α, γ instead

Example: constants abstraction and Galois connection

Constants lattice $D^\sharp_\mathcal{C} = \{\bot, \top\} \uplus \{\underline{n} \mid n \in \mathbb{Z}\}$

$$\begin{array}{rcll} \alpha_\mathcal{C}(c) & = & \bot & \text{if } c = \emptyset \\ \alpha_\mathcal{C}(c) & = & \underline{n} & \text{if } c = \{n\} \\ \alpha_\mathcal{C}(c) & = & \top & \text{otherwise} \end{array} \qquad \begin{array}{rcl} \gamma_\mathcal{C}(\top) & \longmapsto & \mathbb{Z} \\ \gamma_\mathcal{C}(\underline{n}) & \longmapsto & \{n\} \\ \gamma_\mathcal{C}(\bot) & \longmapsto & \emptyset \end{array}$$

Thus:

- if $c = \emptyset$, $\forall a,\ c \subseteq \gamma_\mathcal{C}(a)$, i.e., $c \subseteq \gamma_\mathcal{C}(a) \iff \alpha_\mathcal{C}(c) = \bot \sqsubseteq a$
- if $c = \{n\}$,
  $\alpha_\mathcal{C}(\{n\}) = \underline{n} \sqsubseteq a \iff a = \underline{n} \vee a = \top \iff c = \{n\} \subseteq \gamma_\mathcal{C}(a)$
- if c has at least two distinct elements n_0, n_1, $\alpha_\mathcal{C}(c) = \top$ and
  $c \subseteq \gamma_\mathcal{C}(a) \Rightarrow a = \top$, i.e., $c \subseteq \gamma_\mathcal{C}(a) \iff \alpha_\mathcal{C}(c) = \top \sqsubseteq a$

Constant abstraction: Galois connection

$c \subseteq \gamma_\mathcal{C}(a) \iff \alpha_\mathcal{C}(c) \sqsubseteq a$, therefore, $(\mathcal{P}(\mathbb{Z}), \subseteq) \underset{\alpha_\mathcal{C}}{\overset{\gamma_\mathcal{C}}{\leftrightarrows}} (D^\sharp_\mathcal{C}, \sqsubseteq)$

Example: non relational abstraction Galois connection

We have defined:

$$\begin{array}{rlcl}
\alpha_{\mathcal{NR}}: & (c \subseteq (X \to Y)) & \longmapsto & (x \in X) \mapsto \{f(x) \mid f \in c\} \\
\gamma_{\mathcal{NR}}: & (\Phi \in (X \to \mathcal{P}(Y))) & \longmapsto & \{f : X \to Y \mid \forall x \in X,\, f(x) \in \Phi(x)\}
\end{array}$$

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

$$\begin{array}{rcl}
\alpha_{\mathcal{NR}}(c) \sqsubseteq \Phi & \Longleftrightarrow & \forall x \in X,\, \alpha_{\mathcal{NR}}(c)(x) \subseteq \Phi(x) \\
& \Longleftrightarrow & \forall x \in X,\, \{f(x) \mid f \in c\} \subseteq \Phi(x) \\
& \Longleftrightarrow & \forall f \in c,\, \forall x \in X,\, f(x) \in \Phi(x) \\
& \Longleftrightarrow & \forall f \in c,\, f \in \gamma_{\mathcal{NR}}(\Phi) \\
& \Longleftrightarrow & c \subseteq \gamma_{\mathcal{NR}}(\Phi)
\end{array}$$

Non relational abstraction: Galois connection

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

$$(\mathcal{P}(X \to Y), \subseteq) \underset{\alpha_{\mathcal{NR}}}{\overset{\gamma_{\mathcal{NR}}}{\leftrightarrows}} (X \to \mathcal{P}(Y), \sqsubseteq)$$

Galois connection properties

Galois connections have **many useful properties**.

In the next few slides, we consider a Galois connection $(C, \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \sqsubseteq)$ 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 $c \subseteq \gamma(\alpha(c))$

Galois connection properties

Monotonicity of adjoints

- α is **monotone**
- γ is **monotone**

Proof:

- **monotonicity of α:** 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 connnection, $\alpha(c_0) \sqsubseteq \alpha(c_1)$
- **monotonicity of γ:** same principle

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

Duality principle applied for Galois connections

If $(C, \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \sqsubseteq)$, then $(A, \sqsupseteq) \underset{\gamma}{\overset{\alpha}{\leftrightarrows}} (C, \supseteq)$

Galois connection properties

Iteration of adjoints

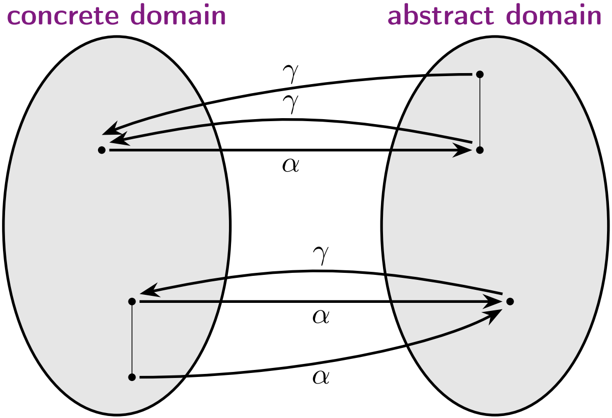
- $\alpha \circ \gamma \circ \alpha = \alpha$
- $\gamma \circ \alpha \circ \gamma = \gamma$
- $\alpha \circ \gamma$ (resp., $\gamma \circ \alpha$) is idempotent, hence a lower (resp., upper) closure operator

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, $\alpha \circ \gamma \circ \alpha(c) \sqsubseteq \alpha(c)$
  moreover, $\gamma \circ \alpha$ is extensive and α monotone, so $\alpha(c) \sqsubseteq \alpha \circ \gamma \circ \alpha(c)$
  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

α 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 $a \in A$:

$$\begin{array}{rcl} \alpha(c_0 \cup c_1) \sqsubseteq a & \Longleftrightarrow & c_0 \cup c_1 \subseteq \gamma(a) \\ & \Longleftrightarrow & c_0 \subseteq \gamma(a) \wedge c_1 \subseteq \gamma(a) \\ & \Longleftrightarrow & \alpha(c_0) \sqsubseteq a \wedge \alpha(c_1) \sqsubseteq a \\ & \Longleftrightarrow & \alpha(c_0) \sqcup \alpha(c_1) \sqsubseteq a \end{array}$$

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 $(C, \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \sqsubseteq)$, 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 $(C, \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \sqsubseteq)$, and it is defined dually

Proof of the first point (the other follows by duality):
we assume that there exists an α 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 when no abstraction can be constructed :-)

Turning an adjoint into a Galois connection

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 α: 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

Galois connection characterization

A characterization of Galois connections

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

- $\alpha \circ \gamma$ is contractive
- $\gamma \circ \alpha$ is extensive

Then, we have a Galois connection

$$(C, \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \sqsubseteq)$$

Proof:

- let $c \in C$ and $a \in A$ such that $\alpha(c) \sqsubseteq a$.
  then: $\gamma(\alpha(c)) \subseteq \gamma(a)$ (as γ 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

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**

$$(C, \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \sqsubseteq)$$

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

We need **several lectures** to cover this.

Towards a notion of abstract transformer...

The problem

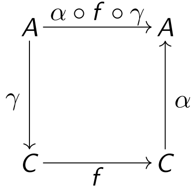
We assume:

- **a monotone concrete function** $f : C \rightarrow C$, known on paper but possibly not computable (intuitively the semantics of a program),
- a **concrete element** $c \in C$ (intuitively an initial state),
- and an **abstract element** $a \in A$ **that abstracts** c

Question: how to derive an abstraction of $f(c)$ **?**

1. $\alpha \circ f(c)$ abstracts the image of c by f
2. $f(c)$ **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 α



Towards a notion of abstract transformer...

We consider a variant of the previous problem:

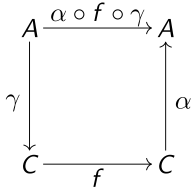
The problem

We assume:

- **a monotone concrete function $f : C \to C$**, known on paper (but not computable) (e.g., the semantics of a program)
- and an **abstract element $a \in A$ that abstracts initial states**

Question: how to derive an abstraction of final states?

1. if c is an initial state, it is abstracted by a, **thus $c \in \gamma(a)$**
2. the same reasoning applies
   so **$f(c)$ is abstracted by $\alpha \circ f \circ \gamma(a)$**



Soundness and completeness

Assumptions:

- a Galois connection, same notation as above
- concrete function $f : C \rightarrow C$

Definition: Best abstract transformer

The **best abstract transformer** approximating f is $f^\sharp = \alpha \circ f \circ \gamma$

In some cases, the best abstract transformer may be *too expensive*, hence we also consider:

Definition: Sound abstract transformers

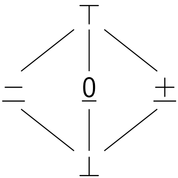
A **sound abstract transformer** approximating f is any operator $f^\sharp : A \rightarrow A$, such that $\alpha \circ f \circ \gamma \sqsubseteq f^\sharp$ (or equivalently, $f \circ \gamma \subseteq \gamma \circ f^\sharp$).

- Clearly, the best abstract transformer is sound
- Other transformers give up completeness

Example: lattice of signs

- $f : D^\sharp_{\mathcal{C}} \to D^\sharp_{\mathcal{C}}, c \mapsto \{-n \mid n \in c\}$
- $f^\sharp = \alpha \circ f \circ \gamma$

Lattice of signs:



Abstract negation operator:

a	$\ominus^\sharp(a)$
$\bot$	$\bot$
$\underline{-}$	$\underline{+}$
$\underline{0}$	$\underline{0}$
$\underline{+}$	$\underline{-}$
$\top$	$\top$

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

Abstract n-ary operators

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

Definition: best and sound 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:

$$\begin{array}{rlcl} g^\sharp : & A^n & \longmapsto & A \\ & (a_0, \ldots, a_{n-1}) & \longmapsto & \alpha \circ g(\gamma(a_0), \ldots, \gamma(a_{n-1})) \end{array}$$

- a **sound abstract transformer** approximating g is any operator $g^\sharp : A^n \to A$, such that

$$\forall (a_0, \ldots, a_{n-1}) \in A^n,\ \alpha \circ g(\gamma(a_0), \ldots, \gamma(a_{n-1})) \sqsubseteq g^\sharp(a_0, \ldots, a_{n-1})$$

(i.e., equivalently, $g(\gamma(a_0), \ldots, \gamma(a_{n-1})) \subseteq \gamma \circ g^\sharp(a_0, \ldots, a_{n-1})$)

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:

$\oplus^\sharp$	$\bot$	$\underline{-}$	$\underline{0}$	$\underline{+}$	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
$\underline{-}$	$\bot$	$\underline{-}$	$\underline{-}$	$\top$	$\top$
$\underline{0}$	$\bot$	$\underline{-}$	$\underline{0}$	$\underline{+}$	$\top$
$\underline{+}$	$\bot$	$\top$	$\underline{+}$	$\underline{+}$	$\top$
$\top$	$\bot$	$\top$	$\top$	$\top$	$\top$

$\otimes^\sharp$	$\bot$	$\underline{-}$	$\underline{0}$	$\underline{+}$	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
$\underline{-}$	$\bot$	$\underline{+}$	$\underline{0}$	$\underline{-}$	$\top$
$\underline{0}$	$\bot$	$\underline{0}$	$\underline{0}$	$\underline{0}$	$\underline{0}$
$\underline{+}$	$\bot$	$\underline{-}$	$\underline{0}$	$\underline{+}$	$\top$
$\top$	$\bot$	$\top$	$\underline{0}$	$\top$	$\top$

Example of loss in precision:

- $\{8\} \in \gamma_{\mathcal{S}}(\underline{+})$ and $\{-2\} \in \gamma_{\mathcal{S}}(\underline{-})$
- $\oplus^\sharp(\underline{+}, \underline{-}) = \top$ is **a lot worse than** $\alpha_{\mathcal{S}}(\oplus(\{8\}, \{-2\})) = \underline{+}$

Example: lattice of signs set operators

Best abstract operators approximating $\cup$ **and** $\cap$ defined over pairs of sets (thus, as binary operators):

$\cup^\sharp$	$\bot$	$\underline{-}$	$\underline{0}$	$\underline{+}$	$\top$
$\bot$	$\bot$	$\underline{-}$	$\underline{0}$	$\underline{+}$	$\top$
$\underline{-}$	$\underline{-}$	$\underline{-}$	$\top$	$\top$	$\top$
$\underline{0}$	$\underline{0}$	$\top$	$\underline{0}$	$\top$	$\top$
$\underline{+}$	$\underline{+}$	$\top$	$\top$	$\underline{+}$	$\top$
$\top$	$\top$	$\top$	$\top$	$\top$	$\top$

$\cap^\sharp$	$\bot$	$\underline{-}$	$\underline{0}$	$\underline{+}$	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
$\underline{-}$	$\bot$	$\underline{-}$	$\bot$	$\bot$	$\underline{-}$
$\underline{0}$	$\bot$	$\bot$	$\underline{0}$	$\bot$	$\underline{0}$
$\underline{+}$	$\bot$	$\bot$	$\bot$	$\underline{+}$	$\underline{+}$
$\top$	$\bot$	$\underline{-}$	$\underline{0}$	$\underline{+}$	$\top$

Soundness: $\left\{ \begin{array}{lcl} \gamma(a_0) \cup \gamma(a_1) & \subseteq & \gamma(c_0 \cup^\sharp c_1) \\ \gamma(a_0) \cap \gamma(a_1) & \subseteq & \gamma(c_0 \cap^\sharp c_1) \end{array} \right.$

Example of loss in precision:

- $\gamma(\underline{-}) \cup \gamma(\underline{+}) = \{n \in \mathbb{Z} \mid n \neq 0\} \subset \gamma(\top)$

Outline

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 $(C, \subseteq) \underset{\alpha}{\overset{\gamma}{\longleftrightarrow}} (A, \sqsubseteq)$, two functions $f : C \to C$ and $f^\sharp : A \to A$ and two elements $c_0 \in C, a_0 \in A$ such that:

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

Then:

- **both f and $f^\sharp$ have a least-fixpoint** (by Tarski's fixpoint theorem)
- $\alpha(\mathrm{lfp}_{c_0} f) = \mathrm{lfp}_{a_0} f^\sharp$

Fixpoint transfer: proof

- **$\alpha(\mathsf{lfp}_{c_0} f)$ is a fixpoint of $f^\sharp$** since:

$$\begin{array}{rcll} f^\sharp(\alpha(\mathsf{lfp}_{c_0} f)) & = & \alpha(f(\mathsf{lfp}_{c_0} f)) & \text{since } \alpha \circ f = f^\sharp \circ \alpha \\ & = & \alpha(\mathsf{lfp}_{c_0} f) & \text{by definition of the fixpoints} \end{array}$$

- **To show that $\alpha(\mathsf{lfp}_{c_0} f)$ is the least-fixpoint of $f^\sharp$,** we assume that X is another fixpoint of $f^\sharp$ greater than a_0 and we show that $\alpha(\mathsf{lfp}_{c_0} f) \sqsubseteq X$, i.e., that $\mathsf{lfp}_{c_0} f \subseteq \gamma(X)$.
  As $\mathsf{lfp}_{c_0} f = \bigcup_{n \in \mathbb{N}} f^n(c_0)$ (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 $\alpha(f^0(c_0)) = a_0 \sqsubseteq X$; 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 $\alpha(f^{n+1}(c_0)) \sqsubseteq X$:

$$\alpha(f^{n+1}(c_0)) = \alpha \circ f(f^n(c_0)) = f^\sharp \circ \alpha(f^n(c_0)) \sqsubseteq f^\sharp(X) = X$$

    as $\alpha(f^n(c_0)) \sqsubseteq X$ and $f^\sharp$ 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 $a_{n+1} = a_n \sqcup f^\sharp(a_n)$.

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

Proof: exercise.

Note:

- the assumptions we have made are **too restrictive** in practice
- more general fixpoint abstraction methods in the next lecture

Outline

Comparing existing semantics

1. A **concrete semantics** $[\![P]\!]$ is given: e.g., big steps operational semantics
2. An **abstract semantics** $[\![P]\!]^\sharp$ is given: e.g., denotational semantics
3. **Search for an abstraction relation between them**
   e.g., $[\![P]\!]^\sharp = \alpha([\![P]\!])$, or $[\![P]\!] \subseteq \gamma([\![P]\!]^\sharp)$

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: connection 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** $[\![P]\!]^\sharp$ such that $[\![P]\!] \subseteq \gamma([\![P]\!]^\sharp)$

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** $\mathtt{x}_0, \ldots, \mathtt{x}_{n-1}$
- we consider the language defined by the grammar below:

$$
\begin{array}{rcll}
P & ::= & \mathtt{x}_i = n & \text{where } n \in \mathbb{Z} \\
& | & \mathtt{x}_i = \mathtt{x}_j + \mathtt{x}_k & \text{basic, three-addresses arithmetics} \\
& | & \mathtt{x}_i = \mathtt{x}_j - \mathtt{x}_k & \text{basic, three-addresses arithmetics} \\
& | & \mathtt{x}_i = \mathtt{x}_j \cdot \mathtt{x}_k & \text{basic, three-addresses arithmetics} \\
& | & P; P & \text{concatenation} \\
& | & \text{while}()\ P & \text{loop, non-deterministic iteration count}
\end{array}
$$

- 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

Concrete semantics

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

$$
\begin{array}{rcl}
[\![\mathrm{x}_i = n]\!](\mathcal{M}) & = & \{\sigma[i \leftarrow n] \mid \sigma \in \mathcal{M}\} \\
[\![\mathrm{x}_i = \mathrm{x}_j + \mathrm{x}_k]\!](\mathcal{M}) & = & \{\sigma[i \leftarrow \sigma_j + \sigma_k] \mid \sigma \in \mathcal{M}\} \\
[\![\mathrm{x}_i = \mathrm{x}_j - \mathrm{x}_k]\!](\mathcal{M}) & = & \{\sigma[i \leftarrow \sigma_j - \sigma_k] \mid \sigma \in \mathcal{M}\} \\
[\![\mathrm{x}_i = \mathrm{x}_j * \mathrm{x}_k]\!](\mathcal{M}) & = & \{\sigma[i \leftarrow \sigma_j * \sigma_k] \mid \sigma \in \mathcal{M}\} \\
[\![P_0; P_1]\!](\mathcal{M}) & = & [\![P_1]\!] \circ [\![P_0]\!](\mathcal{M}) \\
[\![\mathrm{while}()\ P]\!](\mathcal{M}) & = & \mathrm{lfp}\, f \\
\multicolumn{3}{c}{f : \mathcal{M}' \mapsto \mathcal{M} \cup \mathcal{M}' \cup [\![P]\!](\mathcal{M}')}
\end{array}
$$

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

Example

A couple of contrived examples

enough to show the behavior of the analysis...

Factorial function:

$$
\begin{array}{l}
\mathtt{x}_0 = 0; \\
\mathtt{x}_1 = 1; \\
\mathtt{x}_2 = 1; \\
\text{while}()\{ \\
\qquad \mathtt{x}_1 = \mathtt{x}_0 * \mathtt{x}_1; \\
\qquad \mathtt{x}_0 = \mathtt{x}_0 + \mathtt{x}_2; \\
\}
\end{array}
$$

- loops exit at some (non deterministic) point
- at the end $\mathtt{x}_1$ is equal to $\mathtt{x}_0$!
- outputs $\mathtt{x}_0, \mathtt{x}_1, \mathtt{x}_2$ 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:** $(D^\sharp, \sqsubseteq)$, where $D^\sharp = (D^\sharp_{\mathcal{S}})^n$ and $\sqsubseteq$ is the pointwise ordering
- **Galois connection** $(\mathcal{P}(\mathbb{Z}), \subseteq) \xrightleftharpoons[\alpha]{\gamma} (D^\sharp, \sqsubseteq)$, defined by

$$\begin{array}{rcll} \alpha : & S & \longmapsto & (\alpha_{\mathcal{S}}(\{\sigma_0 \mid \sigma \in S\}), \ldots, \alpha_{\mathcal{S}}(\{\sigma_{n-1} \mid \sigma \in S\})) \\ \gamma : & M^\sharp & \longmapsto & \{\sigma \in \mathbb{Z}^n \mid \forall i,\ \sigma_i \in \gamma_{\mathcal{S}}(M^\sharp_i)\} \end{array}$$

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**

Abstract semantics: sequences

We search **for an abstract semantics** $[\![P]\!]^\sharp : D^\sharp \to D^\sharp$ such that:

$$\alpha \circ [\![P]\!] \sqsubseteq [\![P]\!]^\sharp \circ \alpha$$

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:

- $\alpha \circ [\![P_0]\!] \sqsubseteq [\![P_0]\!]^\sharp \circ \alpha$
- $\alpha \circ [\![P_1]\!] \sqsubseteq [\![P_1]\!]^\sharp \circ \alpha$

Since $[\![P_0; P_1]\!] = [\![P_1]\!] \circ [\![P_0]\!]$, we expect $[\![P_0; P_1]\!]^\sharp = [\![P_1]\!]^\sharp \circ [\![P_0]\!]^\sharp$:

$$\begin{array}{rcll} \alpha \circ [\![P_1]\!] \circ [\![P_0]\!] & \sqsubseteq & [\![P_1]\!]^\sharp \circ \alpha \circ [\![P_0]\!] & \text{(by induction)} \\ & \sqsubseteq & [\![P_1]\!]^\sharp \circ [\![P_0]\!]^\sharp \circ \alpha & \text{by induction...} \\ & & & \text{and if } [\![P_1]\!]^\sharp \text{ monotone)!} \end{array}$$

Big additional constraint (only today): $[\![P]\!]^\sharp$ monotone

Abstract semantics: assignment command

We now consider the analysis of **assignment statements**

We observe that:

$$\begin{array}{rcl} \alpha(\mathcal{M}) & = & (\alpha_{\mathcal{S}}(\{\sigma_0 \mid \sigma \in \mathcal{M}\}), \ldots, \alpha_{\mathcal{S}}(\{\sigma_{n-1} \mid \sigma \in \mathcal{M}\})) \\ \alpha \circ [\![P]\!](\mathcal{M}) & = & (\alpha_{\mathcal{S}}(\{\sigma_0 \mid \sigma \in [\![P]\!](\mathcal{M})\}), \ldots, \alpha_{\mathcal{S}}(\{\sigma_{n-1} \mid \sigma \in [\![P]\!](\mathcal{M})\})) \end{array}$$

We start with $\mathtt{x}_i = n$:

$$\begin{array}{l} \alpha \circ [\![\mathtt{x}_i = n]\!](\mathcal{M}) \\ \quad = \quad (\alpha_{\mathcal{S}}(\{\sigma_0 \mid \sigma \in [\![P]\!](\{\sigma[i \leftarrow n] \mid \sigma \in \mathcal{M}\})\}), \ldots, \\ \qquad\qquad \alpha_{\mathcal{S}}(\{\sigma_{n-1} \mid \sigma \in [\![P]\!](\{\sigma[i \leftarrow n] \mid \sigma \in S\})\})) \\ \quad = \quad (\alpha_{\mathcal{S}}(\{\sigma_0 \mid \sigma \in \mathcal{M}\}), \ldots, \alpha_{\mathcal{S}}(\{\sigma_{n-1} \mid \sigma \in \mathcal{M}\}))[i \leftarrow \alpha_{\mathcal{S}}(\{n\})] \\ \quad = \quad \alpha(\mathcal{M})[i \leftarrow \alpha_{\mathcal{S}}(\{n\})] \\ \quad = \quad [\![\mathtt{x}_i = n]\!]^\sharp(\alpha(\mathcal{M})) \end{array}$$

where:

$$[\![\mathtt{x}_i = n]\!]^\sharp(M^\sharp) = M^\sharp[i \leftarrow \alpha_{\mathcal{S}}(\{n\})]$$

Computation of the abstract semantics

Other assignments are treated in a similar manner:

$$\begin{array}{rcl}
[\![\mathrm{x}_i = n]\!]^\sharp(M^\sharp) & = & M^\sharp[i \leftarrow \alpha_{\mathcal{S}}(\{n\})] \\
[\![\mathrm{x}_i = \mathrm{x}_j + \mathrm{x}_k]\!]^\sharp(M^\sharp) & = & M^\sharp[i \leftarrow M^\sharp_j \oplus^\sharp M^\sharp_k] \\
[\![\mathrm{x}_i = \mathrm{x}_j - \mathrm{x}_k]\!](M^\sharp) & = & M^\sharp[i \leftarrow M^\sharp_j \ominus^\sharp M^\sharp_k] \\
[\![\mathrm{x}_i = \mathrm{x}_j * \mathrm{x}_k]\!]^\sharp(M^\sharp) & = & M^\sharp[i \leftarrow M^\sharp_j \otimes^\sharp M^\sharp_k]
\end{array}$$

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

Analysis of a loop

We have seen that:

$$\begin{array}{rcl} [\![\text{while}()\ P]\!](\mathcal{M}) & = & \text{lfp}\, f \\ \text{where } f(\mathcal{M}') & = & \mathcal{M} \cup \mathcal{M}' \cup [\![P]\!](\mathcal{M}') \end{array}$$

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
- $f^\sharp$ is monotone
- $\alpha \circ f \sqsubseteq f^\sharp \circ \alpha$
- $\alpha(\emptyset) = \bot$

Then, $\alpha(\text{lfp}\, f) \sqsubseteq \text{lfp}\, f^\sharp$

Analysis of a loop

Application:

- we consider the analysis of the loop with pre-condition $M^\sharp$
- we take
$$f^\sharp(M_0^\sharp) = M^\sharp \cup M_0^\sharp \cup [\![P]\!]^\sharp(M_0^\sharp)$$
- then, $\alpha \circ f \sqsubseteq f^\sharp \circ \alpha$
- we can apply **the new fixpoint transfer theorem**...

$$\begin{array}{rcl} [\![\mathrm{while}()\ P]\!]^\sharp(M^\sharp) & = & \mathrm{lfp}_{M^\sharp}\, f^\sharp \\ \textbf{where } f^\sharp(M_0^\sharp) & = & M^\sharp \cup M_0^\sharp \cup [\![P]\!]^\sharp(M_0^\sharp) \end{array}$$

One more thing:

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

Abstract semantics

Abstract semantics and soundness

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

$$\begin{array}{rcl}
[\![\mathrm{x}_i = n]\!]^\sharp(M^\sharp) & = & M^\sharp[i \leftarrow \alpha_{\mathcal{S}}(\{n\})] \\
[\![\mathrm{x}_i = \mathrm{x}_j + \mathrm{x}_k]\!]^\sharp(M^\sharp) & = & M^\sharp[i \leftarrow M^\sharp_j \oplus^\sharp M^\sharp_k] \\
[\![\mathrm{x}_i = \mathrm{x}_j - \mathrm{x}_k]\!]^\sharp(M^\sharp) & = & M^\sharp[i \leftarrow M^\sharp_j \ominus^\sharp M^\sharp_k] \\
[\![\mathrm{x}_i = \mathrm{x}_j \cdot \mathrm{x}_k]\!]^\sharp(M^\sharp) & = & M^\sharp[i \leftarrow M^\sharp_j \otimes^\sharp M^\sharp_k] \\
[\![\mathrm{while}()\ P]\!]^\sharp(M^\sharp) & = & \mathrm{lfp}_{M^\sharp}\, f^\sharp \text{ where} \\
 & & f^\sharp : M^\sharp \mapsto M^\sharp \sqcup [\![P]\!]^\sharp(M^\sharp)
\end{array}$$

Furthermore, for all program P: $\alpha \circ [\![P]\!] \sqsubseteq [\![P]\!]^\sharp \circ \alpha$
Last, $[\![P]\!]^\sharp$ is **monotone**

An **over-approximation of the final states** is computed by $[\![P]\!]^\sharp(\top)$.

Example

Factorial function:

$x_0 = 1;$
$x_1 = 1;$
$x_2 = 1;$
while(){
    $x_1 = x_0 * x_1;$
    $x_0 = x_0 + x_2;$
}

Abstract state **before the loop:**
$(\underline{+}, \underline{+}, \underline{+})$

Iterates on the loop:

iterate	0	1
x_0	$\underline{+}$	$\underline{+}$
x_1	$\underline{+}$	$\underline{+}$
x_2	$\underline{+}$	$\underline{+}$

Abstract state **after the loop:** $(\underline{+}, \underline{+}, \underline{+})$

Outline

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**