Abstract Interpretation

Semantics and applications to verification

Xavier Rival

École Normale Supérieure

May 5th, 2017

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

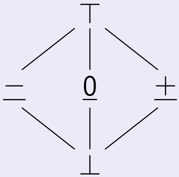
Abstraction example 1: signs

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**:

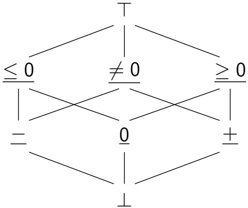
- if $c_0 \vdash_{\mathcal{S}} \underline{+}$ and $c_1 \vdash_{\mathcal{S}} \underline{+}$, then $c_0 \cup c_1 \vdash_{\mathcal{S}} \underline{+}$
- if $c_0 \vdash_{\mathcal{S}} \underline{+}$ and $c_1 \vdash_{\mathcal{S}} \perp$, then $c_0 \cup c_1 \vdash_{\mathcal{S}} \underline{+}$

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 $c_0 \cup c_1 \vdash_{\mathcal{S}} \underline{\geq 0}$

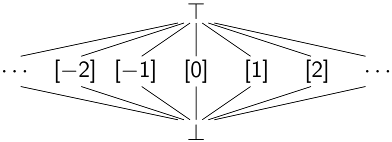


Abstraction example 2: constants

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:** $\bot, \top, (a, b)$ 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:** $\mathcal{P}(X \rightarrow Y)$, inclusion ordering
- **abstract elements:** $X \rightarrow \mathcal{P}(Y)$, 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: $c \vdash a$ 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

1. Abstraction
   - Notion of abstraction
   - Abstraction and concretization functions
   - Galois connections
2. Abstract interpretation
3. Applications of abstract interpretation
4. A basic static analysis
5. A more realistic static analysis
6. Termination of the Static Analysis
7. 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

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

When we have a c (resp., a) and try to map it into a compatible a (resp. a 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 γ.

- $c \vdash a$ if and only if $c \subseteq \gamma(a)$

Concretization function: a few examples

Signs abstraction:

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

Constants abstraction:

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

Non relational abstraction:

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

Parikh vector abstraction: exercise!

Abstraction function

Our **second adjoint function:**

Definition: abstraction function

Abstraction function $\alpha : C \rightarrow A$ (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 α.

Summary on adjoint functions:

- α returns the **most precise abstract predicate** that holds true for its argument
  this is called the **best abstraction**
- γ returns the **most general concrete meaning** of its argument
  hence, is called the **concretization**

Abstraction: a few examples

Constants abstraction:

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

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

1. Abstraction
   - Notion of abstraction
   - Abstraction and concretization functions
   - Galois connections
2. Abstract interpretation
3. Applications of abstract interpretation
4. A basic static analysis
5. A more realistic static analysis
6. Termination of the Static Analysis
7. Conclusion

Definition

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 $\vdash$!

Definition: Galois connection

A **Galois connection** is defined by 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 $D^\sharp_\mathcal{C} = \{\bot, \top\} \uplus \{\underline{n} \mid n \in \mathbb{Z}\}$

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

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 $c = \{n\}$,
  $\alpha_\mathcal{C}(\{n\}) = \underline{n} \sqsubseteq c \iff c = \underline{n} \vee c = \top \iff c = \{n\} \subseteq \gamma_\mathcal{C}(a)$
- 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., $c \subseteq \gamma_\mathcal{C}(a) \iff \alpha_\mathcal{C}(c) = \bot \sqsubseteq a$

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 $c \subseteq \gamma(\alpha(a))$

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

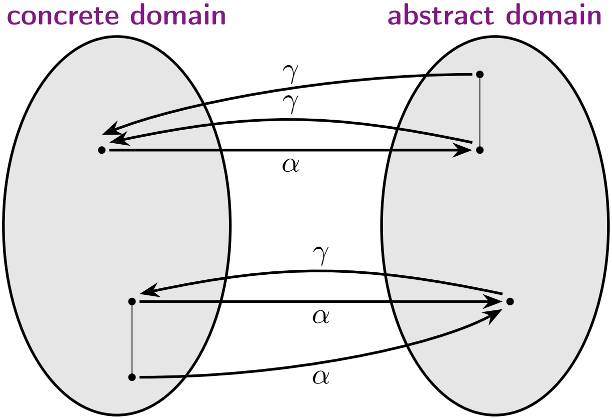
- $\alpha \circ \gamma \circ \alpha = \alpha$
- $\gamma \circ \alpha \circ \gamma = \gamma$
- $\alpha \circ \gamma$ (resp., $\gamma \circ \alpha$) is idempotent, hence a lower (resp., upper) closure operator

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 $a \in A$:

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

Note: when C, A are complete lattices, this extends to families of elements

Galois connection properties

Uniqueness of adjoints

- given $\gamma : C \rightarrow A$, there exists **at most one** $\alpha : A \rightarrow C$ 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 $\alpha : A \rightarrow C$, there exists at most one $\gamma : C \rightarrow A$ 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 $\alpha(a) \sqsubseteq c$ 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, $a_0 \sqsubseteq \alpha(c)$

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 $\gamma : (A, \sqsubseteq) \rightarrow (C, \subseteq)$ 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 $\alpha : C \to A$ 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

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

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

1. Abstraction
2. Abstract interpretation
   - Abstract computation
   - Fixpoint transfer
3. Applications of abstract interpretation
4. A basic static analysis
5. A more realistic static analysis
6. Termination of the Static Analysis
7. 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**

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

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

Abstract transformer

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

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

$$\begin{tikzcd} A \arrow[r, "f^\sharp"] \arrow[d, "\gamma"'] & A \\ C \arrow[r, "f"'] & C \arrow[u, "\alpha"'] \end{tikzcd}$$

- 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** $\alpha \circ f \circ \gamma(a)$:

$$\begin{array}{ll} c \subseteq \gamma(a) & \text{by assumption} \\ f(c) \subseteq f(\gamma(a)) & \text{by monotonicity of } f \\ \alpha(f(c)) \subseteq \alpha(f(\gamma(a))) & \text{by monotonicity of } \alpha \end{array}$$

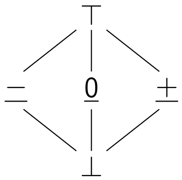
Definition: best and sound abstract transformers

- the **best abstract transformer** approximating f is $f^\sharp = \alpha \circ f \circ \gamma$
- a **sound abstract transformer** approximating f is any operator $f^\sharp : A \to A$, such that $\alpha \circ f \circ \gamma \sqsubseteq f^\sharp$ (or equivalently, $f \circ \gamma \subseteq \gamma \circ f^\sharp$)

Example: lattice of signs

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

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: sound and exact abstract operators

Let $g : C^n \rightarrow C$ 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 $g^\sharp : A^n \rightarrow A$, 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:

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

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** $\cap$:

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

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

Example of loss in precision:

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

Outline

1. Abstraction
2. Abstract interpretation
    - Abstract computation
    - Fixpoint transfer
3. Applications of abstract interpretation
4. A basic static analysis
5. A more realistic static analysis
6. Termination of the Static Analysis
7. 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 $(C, \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \sqsubseteq)$, two functions $f : C \rightarrow C$ and $f^\sharp : A \rightarrow A$ 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)
- $\alpha(\mathbf{lfp}_{c_0}\, f) = \mathbf{lfp}_{a_0}\, f^\sharp$

Fixpoint transfer: proof

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

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

- **To show that $\alpha(\mathbf{lfp}_{c_0} f)$ 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 $\alpha(\mathbf{lfp}_{c_0} f) \sqsubseteq X$, i.e., that $\mathbf{lfp}_{c_0} f \subseteq \gamma(X)$.
As $\mathbf{lfp}_{c_0} f = \bigcup_{n \in \mathbb{N}} f^n(c_0)$ (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** $\alpha(\mathbf{lfp}_{c_0}\, f) = a_\infty$.

Proof: exercise.

Note:

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

Outline

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

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.

Outline

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:

$$\begin{array}{rcll}
P & ::= & \mathtt{x}_i = n & \text{where } n \in \mathbb{Z} \\
 & | & \mathtt{x}_i = \mathtt{x}_j + \mathtt{x}_k & \text{basic, three-addresses arithmetics} \\
 & | & \mathtt{x}_i = \mathtt{x}_j - \mathtt{x}_k & \text{basic, three-addresses arithmetics} \\
 & | & \mathtt{x}_i = \mathtt{x}_j \cdot \mathtt{x}_k & \text{basic, three-addresses arithmetics} \\
 & | & P; P & \text{concatenation} \\
 & | & \textbf{input}(\mathtt{x}_i) & \text{reading of a positive input} \\
 & | & \textbf{if}(\mathtt{x}_i > 0)\ P\ \textbf{else}\ P & \\
 & | & \textbf{while}(\mathtt{x}_i > 0)\ P &
\end{array}$$

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

Concrete semantics

Concrete semantics

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

$$\begin{array}{rcl}
[\![\mathrm{x}_i = n]\!](S) & = & \{\sigma[i \leftarrow n] \mid \sigma \in S\} \\
[\![\mathrm{x}_i = \mathrm{x}_j + \mathrm{x}_k]\!](S) & = & \{\sigma[i \leftarrow \sigma_j + \sigma_k] \mid \sigma \in S\} \\
[\![\mathrm{x}_i = \mathrm{x}_j - \mathrm{x}_k]\!](S) & = & \{\sigma[i \leftarrow \sigma_j - \sigma_k] \mid \sigma \in S\} \\
[\![\mathrm{x}_i = \mathrm{x}_j \cdot \mathrm{x}_k]\!](S) & = & \{\sigma[i \leftarrow \sigma_j \cdot \sigma_k] \mid \sigma \in S\} \\
[\![\mathbf{input}(\mathrm{x}_i)]\!](S) & = & \{\sigma[i \leftarrow n] \mid \sigma \in S \wedge n > 0\} \\
[\![P_0; P_1]\!](S) & = & [\![P_1]\!] \circ [\![P_0]\!](S) \\
[\![\mathbf{if}(\mathrm{x}_i > 0)\, P_0 \ \mathbf{else}\ P_1]\!](S) & = & [\![P_0]\!](\{\sigma \in S \mid \sigma_i > 0\}) \\
 & & \cup [\![P_1]\!](\{\sigma \in S \mid \sigma_i \leq 0\}) \\
[\![\mathbf{while}(\mathrm{x}_i > 0)\, P]\!](S) & = & \{\sigma \in \mathbf{lfp}_S\, f \mid \sigma_i \leq 0\} \text{ where} \\
 & & f : S' \mapsto S' \cup [\![P]\!](\{\sigma \in S' \mid \sigma_i > 0\})
\end{array}$$

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

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:** $(D^\sharp, \sqsubseteq)$, 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

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

Example

Factorial function:

```
input(x0);
x1 = 1;
x2 = 1;
while(x0 > 0){
      x1 = x0 · x1;
      x0 = x0 − x2;
}
```

Abstraction of the semantics:

- abstract pre-condition: $(\underline{0}, \underline{0}, \underline{0})$
- abstract state before the loop: $(\underline{+}, \underline{+}, \underline{+})$
- abstract post-condition (after the loop): $(\underline{0}, \underline{+}, \underline{+})$

Computation of the abstract semantics

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

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

We observe that:

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

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

$$\begin{array}{rl} \multicolumn{2}{l}{\alpha \circ [\![\mathtt{x}_i = n]\!](S)} \\ = & (\alpha_{\mathcal{S}}(\{\sigma_0 \mid \sigma \in \{\sigma[i \leftarrow n] \mid \sigma \in S\}\}), \ldots, \\ & \quad \alpha_{\mathcal{S}}(\{\sigma_{n-1} \mid \sigma \in \{\sigma[i \leftarrow n] \mid \sigma \in S\}\})) \\ = & (\alpha_{\mathcal{S}}(\{\sigma_0 \mid \sigma \in S\}), \ldots, \alpha_{\mathcal{S}}(\{\sigma_{n-1} \mid \sigma \in S\}))[i \leftarrow \alpha_{\mathcal{S}}(\{n\})] \\ = & \alpha(S)[i \leftarrow \alpha_{\mathcal{S}}(\{n\})] \\ = & [\![\mathtt{x}_i = n]\!]^\sharp(\alpha(S)) \end{array}$$

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

Computation of the abstract semantics

Other assignments are treated in a similar manner:

$$\begin{array}{rcl}
[\![\mathtt{x}_i = \mathtt{x}_j + \mathtt{x}_k]\!]^\sharp(S^\sharp) & = & S^\sharp[i \leftarrow S_j^\sharp \oplus^\sharp S_k^\sharp] \\
[\![\mathtt{x}_i = \mathtt{x}_j - \mathtt{x}_k]\!](S^\sharp) & = & S^\sharp[i \leftarrow S_j^\sharp \ominus^\sharp S_k^\sharp] \\
[\![\mathtt{x}_i = \mathtt{x}_j \cdot \mathtt{x}_k]\!]^\sharp(S^\sharp) & = & S^\sharp[i \leftarrow S_j^\sharp \otimes^\sharp S_k^\sharp] \\
[\![\mathbf{input}(\mathtt{x}_i)]\!]^\sharp(S^\sharp) & = & S^\sharp[i \leftarrow \underline{\pm}]
\end{array}$$

Proofs are left as exercises

Computation of the abstract semantics

We now consider the case of **tests:**

$$\begin{array}{rl}
\multicolumn{2}{l}{\alpha \circ [\![\mathbf{if}(\mathtt{x}_i > 0)\, P_0 \,\mathbf{else}\, P_1]\!](S)} \\
= & \alpha([\![P_0]\!](\{\sigma \in S \mid \sigma_i > 0\}) \,\cup\, [\![P_1]\!](\{\sigma \in S \mid \sigma_i \leq 0\})) \\
= & \alpha([\![P_0]\!](\{\sigma \in S \mid \sigma_i > 0\})) \sqcup \alpha([\![P_1]\!](\{\sigma \in S \mid \sigma_i \leq 0\})) \\
 & \quad \text{as } \alpha \text{ preserves least upper bounds} \\
= & [\![P_0]\!]^\sharp(\alpha(\{\sigma \in S \mid \sigma_i > 0\})) \sqcup [\![P_1]\!]^\sharp(\alpha(\{\sigma \in S \mid \sigma_i \leq 0\})) \\
= & [\![P_0]\!]^\sharp(\alpha(S) \sqcap \top[i \leftarrow \underline{+}]) \sqcup [\![P_1]\!]^\sharp(\alpha(S)) \\
= & [\![\mathbf{if}(\mathtt{x}_i > 0)\, P_0 \,\mathbf{else}\, P_1]\!]^\sharp(\alpha(S))
\end{array}$$

where $[\![\mathbf{if}(\mathtt{x}_i > 0)\, P_0 \,\mathbf{else}\, P_1]\!]^\sharp(S^\sharp) = [\![P_0]\!]^\sharp(S^\sharp \sqcap \top[i \leftarrow \underline{+}]) \sqcup [\![P_1]\!]^\sharp(S^\sharp)$

In the case of **loops**:

$$[\![\mathbf{while}(\mathtt{x}_i > 0)\, P]\!]^\sharp(S^\sharp) = \mathbf{lfp}_{S^\sharp}\, f^\sharp$$

where $f^\sharp : S^\sharp \mapsto S^\sharp \sqcup [\![P]\!]^\sharp(S^\sharp \sqcap \top[i \leftarrow \underline{+}])$

Proof: exercise

Abstract semantics

Abstract semantics and soundness

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

$$\begin{array}{rcl}
[\![\mathtt{x}_i = n]\!]^\sharp(S^\sharp) & = & S^\sharp[i \leftarrow \alpha_{\mathcal{S}}(\{n\})] \\
[\![\mathtt{x}_i = \mathtt{x}_j + \mathtt{x}_k]\!]^\sharp(S^\sharp) & = & S^\sharp[i \leftarrow S_j^\sharp \oplus^\sharp S_k^\sharp] \\
[\![\mathtt{x}_i = \mathtt{x}_j - \mathtt{x}_k]\!]^\sharp(S^\sharp) & = & S^\sharp[i \leftarrow S_j^\sharp \ominus^\sharp S_k^\sharp] \\
[\![\mathtt{x}_i = \mathtt{x}_j \cdot \mathtt{x}_k]\!]^\sharp(S^\sharp) & = & S^\sharp[i \leftarrow S_j^\sharp \otimes^\sharp S_k^\sharp] \\
[\![\textbf{input}(\mathtt{x}_i)]\!]^\sharp(S^\sharp) & = & S^\sharp[i \leftarrow \underline{+}] \\
[\![\textbf{if}(\mathtt{x}_i > 0)\, P_0 \textbf{ else } P_1]\!]^\sharp(S^\sharp) & = & [\![P_0]\!]^\sharp(S^\sharp \sqcap \top[i \leftarrow \underline{+}]) \sqcup [\![P_1]\!]^\sharp(S^\sharp) \\
[\![\textbf{while}(\mathtt{x}_i > 0)\, P]\!]^\sharp(S^\sharp) & = & \textbf{lfp}_{S^\sharp}\, f^\sharp \text{ where} \\
 & & f^\sharp : S^\sharp \mapsto S^\sharp \sqcup [\![P]\!]^\sharp(S^\sharp \sqcap \top[i \leftarrow \underline{+}])
\end{array}$$

Furthermore, for all program P: $\alpha \circ [\![P]\!] = [\![P]\!]^\sharp \circ \alpha$

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

Example

Factorial function:

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

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

Iterates on the loop:

iterate	0	1	2
x_0	$\underline{+}$	$\top$	$\top$
x_1	$\underline{+}$	$\underline{+}$	$\underline{+}$
x_2	$\underline{+}$	$\underline{+}$	$\underline{+}$

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

Outline

Soundness of the Analysis

We have designed the abstract semantics so that it satisfies the following theorem:

Theorem: analysis soundness

For any program P, we have:

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

- By the theorem, $[\![P]\!]^\sharp$ **is sound, i.e., always over-approximates the concrete behaviors of the program**
- Moreover, the computation of $[\![P]\!]^\sharp$ is **fully automatic**

However, we have built $[\![.]\!]^\sharp$ under pretty restrictive assumptions

Limitation 1: exact transformers

Assumption 1

For each concrete operation $f : \mathcal{P}(\mathbb{S}) \to \mathcal{P}(\mathbb{S})$, we have built a **abstract transformer** $f^\sharp : D^\sharp \to D^\sharp$ such that: $\alpha \circ f = f^\sharp \circ \alpha$

Can we always build such a function ?

We consider a simple, contrived example, to show that this assumption is **too strong**:

- **Abstract domain:** basic lattice of signs
  (first version we presented, with five elements)
- **Concrete function:** $f : S \mapsto \{\phi(x) \mid x \in S\}$
  where $\phi(x) = 1$ if x is even and $\phi(x) = -1$ if x is odd
- We start from $X = \{2\}$:
  $\alpha \circ f(X) = \alpha(f(\{2\})) = \alpha(\{1\}) = \pm$
  However, $\alpha(X) = \pm$, and $f(\gamma(\pm)) = \{-1, 1\})$, thus, the most precise result we may expect for $f^\sharp(\alpha(X))$ is $\top$...

Limitation 2: Existence of the Best Abstraction

Even worse:

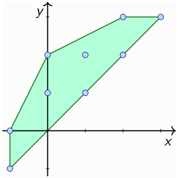
Assumption 2

We assumed the **existence of the best abstraction function**

$$\alpha : \mathcal{P}(\mathbb{S}) \rightarrow D^\sharp$$

Abstract domain of convex polyedra:

- abstract values: **finite conjunctions of linear inequalities**
- **very expressive:** relations between variables



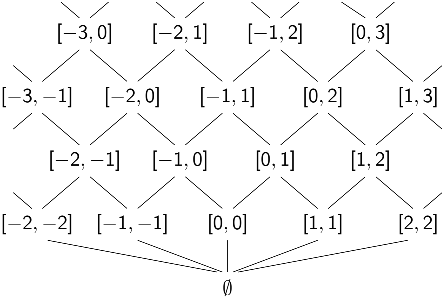
polyedra domain

Without this assumption, **we cannot even use α anymore**

Limitation 3: Finite Height Lattice

Assumption 3

We assumed the **abstract lattice has finite height**



We consider the **lattice of intervals**:

$$
\begin{array}{l}
[0,0] \\
\quad \sqsubset [0,1] \\
\qquad \sqsubset [0,2] \\
\qquad\quad \sqsubset [0,3] \\
\qquad\qquad \sqsubset [0,4] \\
\qquad\qquad\quad \sqsubset \ldots
\end{array}
$$

Without this assumption, **we cannot guarantee the termination of the analysis anymore**

Assumptions

We still assume a **finite height domain**.
We are going to first **relax all assumptions related to α**:

Concretization based framework

We assume:

- an **abstract domain** $D^\sharp$ with an abstract order relation $\sqsubseteq$
- a **monotone concretization function**

$$\gamma : D^\sharp \longrightarrow \mathcal{P}(\mathbb{Z}^n)$$

For a concrete operation $f : \mathcal{P}(\mathbb{Z}^n) \rightarrow \mathcal{P}(\mathbb{Z}^n)$, we set up abstract transfer function $f^\sharp : D^\sharp \rightarrow D^\sharp$ such that:

$$\forall S^\sharp \subseteq D^\sharp,\ f \circ \gamma(S^\sharp) \subseteq \gamma \circ f^\sharp(S^\sharp)$$

Exercise: example function $f : S \mapsto \{\phi(x) \mid x \in S\}$
where $\phi(x) = 1$ if x is even and $\phi(x) = -1$ if x is odd

Comparing Frameworks

We are indeed looking at a **relaxed framework**:

If:

- there exists an abstraction function $\alpha : \mathcal{P}(\mathbb{Z}^n) \rightarrow D^\sharp$
- $\alpha \circ \gamma = \mathbf{Id}$ (or, equivalently α is surjective)
- $f^\sharp : D^\sharp \rightarrow D^\sharp$ is such that $\alpha \circ f = f^\sharp \circ \alpha$

Then:

$$\forall S^\sharp \subseteq D^\sharp,\ f \circ \gamma(S^\sharp) \subseteq \gamma \circ f^\sharp(S^\sharp)$$

- **Proof:** $\alpha \circ f \circ \gamma(S^\sharp) = f^\sharp \circ \alpha \circ \gamma(S^\sharp) = f^\sharp(S^\sharp)$ thus, $\alpha \circ f \circ \gamma(S^\sharp) \sqsubseteq f^\sharp(S^\sharp)$, and $f \circ \gamma(S^\sharp) \subseteq \gamma \circ f^\sharp(S^\sharp)$.
- The condition $\alpha \circ \gamma = \mathbf{Id}$ is not too restrictive (no redundant abstract elements).

  Why ? Is it always satisfied for all lattices seen so far ?

Sound Assignment Operator

Definition

We consider the concrete operation:

$$\mathbf{assign}_{\mathtt{x} \leftarrow \mathsf{e}} : M \longmapsto \{\sigma[\mathtt{x} \leftarrow [\![\mathsf{e}]\!](\sigma)] \mid \sigma \in M\}$$

A sound abstract assignment operator is a function $\mathbf{assign}^\sharp_{\mathtt{x} \leftarrow \mathsf{e}} : D^\sharp \rightarrow D^\sharp$ such that:

$$\forall S^\sharp \in D^\sharp,\ \mathbf{assign}_{\mathtt{x} \leftarrow \mathsf{e}} \circ \gamma(S^\sharp) \subseteq \gamma \circ \mathbf{assign}^\sharp_{\mathtt{x} \leftarrow \mathsf{e}}(S^\sharp)$$

Then, the analysis of all assign statements relies on $\mathbf{assign}^\sharp_{\cdot \leftarrow \cdot}$:

- **Case of $\mathtt{x} = v$:** $[\![\mathtt{x} = v]\!]^\sharp(S^\sharp) = \mathbf{assign}^\sharp_{\mathtt{x} \leftarrow v}(S^\sharp)$
- **Case of $\mathtt{x}_0 = \mathtt{x}_1 \odot \mathtt{x}_2$:** $[\![\mathtt{x}_0 = \mathtt{x}_1 \odot \mathtt{x}_2]\!]^\sharp(S^\sharp) = \mathbf{assign}^\sharp_{\mathtt{x}_0 \leftarrow \mathtt{x}_1 \odot \mathtt{x}_2}(S^\sharp)$
- **Case of input$(\mathtt{x})$:** $[\![\mathbf{input}(\mathtt{x})]\!]^\sharp(S^\sharp) = \mathbf{assign}^\sharp_{\mathtt{x}_0 \leftarrow \mathsf{random}}(S^\sharp)$

Sound Assignment Operator for a Non-Relational Domain

Thus, the real question is **how do we define $\text{assign}^\sharp_{\cdot\leftarrow\cdot}$?**

Case of a non relational domain: $D^\sharp = (D^\sharp_{\mathbb{V}})^n$, where $D^\sharp_{\mathbb{V}} = D^\sharp_{\mathcal{I}}, D^\sharp_{\mathcal{C}}, \ldots$

- **Definition of $[\![\mathrm{e}]\!]^\sharp : D^\sharp \to D^\sharp_{\mathbb{V}}$ by induction over the syntax**:

$$\begin{array}{rcll} [\![v]\!]^\sharp(S^\sharp) & = & v^\sharp & \text{where } v^\sharp \text{ is such that } \{v\} \subseteq \gamma(v^\sharp) \\ [\![\mathtt{x}_k]\!]^\sharp(S^\sharp) & = & S^\sharp_k & \\ [\![\mathrm{e}_0 \odot \mathrm{e}_1]\!]^\sharp(S^\sharp) & = & \odot^\sharp([\![\mathrm{e}_0]\!]^\sharp(S^\sharp), [\![\mathrm{e}_1]\!]^\sharp(S^\sharp)) & \\ [\![\mathrm{random}]\!]^\sharp(S^\sharp) & = & \top & \text{or other approximation of } \mathbb{Z}^\star_+ \end{array}$$

  (very general definition, ways more than that of the expression language)

- **Definition of $\text{assign}^\sharp_{\cdot\leftarrow\cdot}$:**

$$\text{assign}^\sharp_{\mathtt{x}\leftarrow\mathrm{e}}(S^\sharp) = S^\sharp[\mathtt{x} \leftarrow [\![\mathrm{e}]\!]^\sharp(S^\sharp)]$$

Case of relational lattices: **specific algorithms**, next courses

Sound Join Operator

When using α we relied on commutation with union, but with γ this may not work... plus we did not even assume that the abstract domain has a full lattice structure !

Definition

A sound abstract join operator is a binary operator $\sqcup^\sharp : D^\sharp \times D^\sharp \to D^\sharp$ such that:

$$\forall S_0^\sharp, S_1^\sharp \in D^\sharp,\ \gamma(S_0^\sharp) \cup \gamma(S_1^\sharp) \subseteq \gamma(S_0^\sharp \sqcup^\sharp S_1^\sharp)$$

- Non **relational domain:** $D^\sharp = (D_{\mathbb{V}}^\sharp)^n$, where $D_{\mathbb{V}}^\sharp = D_{\mathcal{I}}^\sharp, D_{\mathcal{C}}^\sharp, \ldots$

  We can use the same pointwise operator definition as in the previous lecture (specific case of signs):

  $$S_0^\sharp \sqcup^\sharp S_1^\sharp = k \mapsto (S_0^\sharp(\mathrm{x}_k) \sqcup^\sharp S_1^\sharp(\mathrm{x}_k))$$

- In a **relational abstract domain** (e.g., convex polyedra): convex closure algorithms

Sound Condition Test Operator

To analyze condition tests, we also need a sound abstraction of conditions:

Definition

We consider the concrete operation:

$$\mathbf{test}_{\mathtt{x}\geq 0} : M \longmapsto \{\sigma \in M \mid \sigma(\mathtt{x}) \geq 0\}$$

A sound abstract test operator is a function $\mathbf{test}^{\sharp}_{\mathtt{x}\geq 0} : D^{\sharp} \rightarrow D^{\sharp}$ such that:

$$\mathbf{test}_{\mathtt{x}\geq 0}(\gamma(S^{\sharp})) = \{\sigma \in \gamma(S^{\sharp}) \mid \sigma(\mathtt{x}) \geq 0\} \subseteq \gamma(\mathbf{test}^{\sharp}_{\mathtt{x}\geq 0}(S^{\sharp}))$$

- We observe we get the same definition as in the previous lecture
- Analysis of a condition statement:

$$[\![\mathbf{if}(\mathtt{x} \geq 0)\{P\}]\!]^{\sharp}(S^{\sharp}) = [\![P]\!]^{\sharp}(\mathbf{test}^{\sharp}_{\mathtt{x}\geq 0}(S^{\sharp})) \sqcup^{\sharp} \mathbf{test}^{\sharp}_{\mathtt{x}<0}(S^{\sharp})$$

Approximate Fixpoint Transfer

Transfer theorem

We consider two functions $f : \mathcal{P}(\mathbb{Z}^n) \rightarrow \mathcal{P}(\mathbb{Z}^n)$ and $f^\sharp : D^\sharp \rightarrow D^\sharp$ and two elements $M_0 \in \mathcal{P}(\mathbb{S}), S_0^\sharp \in D^\sharp$ such that:

- f is continuous
- $f^\sharp$ is monotone
- $f \circ \gamma \subseteq \gamma \circ f^\sharp$
- $M_0 \subseteq \gamma(S_0^\sharp)$

Then, **both f and $f^\sharp$ have a least-fixpoint** and $\mathbf{lfp}_{M_0} f \subseteq \gamma(\mathbf{lfp}_{S_0^\sharp} f^\sharp)$

- **Proof:** quite similar to that of the exact case
- **Computation:** we can still compute the abstract lfp by iterative technique, as the abstract lattice is of finite height
- But: this still requires $f^\sharp$ to be monotone...

New static analysis

We can now summarize the definition of a new static analysis:

$$\begin{array}{rcl}
[\![\mathtt{x}_i = n]\!]^\sharp(S^\sharp) & = & \mathbf{assign}^\sharp_{\mathtt{x}_i \leftarrow n}(S^\sharp) \\
[\![\mathtt{x}_i = \mathtt{x}_j + \mathtt{x}_k]\!]^\sharp(S^\sharp) & = & \mathbf{assign}^\sharp_{\mathtt{x}_i \leftarrow \mathtt{x}_j + \mathtt{x}_k}(S^\sharp) \\
[\![\mathtt{x}_i = \mathtt{x}_j - \mathtt{x}_k]\!]^\sharp(S^\sharp) & = & \mathbf{assign}^\sharp_{\mathtt{x}_i \leftarrow \mathtt{x}_j - \mathtt{x}_k}(S^\sharp) \\
[\![\mathtt{x}_i = \mathtt{x}_j \cdot \mathtt{x}_k]\!]^\sharp(S^\sharp) & = & \mathbf{assign}^\sharp_{\mathtt{x}_i \leftarrow \mathtt{x}_j \cdot \mathtt{x}_k}(S^\sharp) \\
[\![\mathbf{input}(\mathtt{x}_i)]\!]^\sharp(S^\sharp) & = & S^\sharp[i \leftarrow \pm] \\
[\![\mathbf{if}(\mathtt{x}_i > 0)\, P_0 \;\mathbf{else}\; P_1]\!]^\sharp(S^\sharp) & = & [\![P_0]\!]^\sharp(\mathbf{test}^\sharp_{\mathtt{x} \geq 0}(S^\sharp)) \sqcup^\sharp [\![P_1]\!]^\sharp(\mathbf{test}^\sharp_{\mathtt{x} < 0}(S^\sharp)) \\
[\![\mathbf{while}(\mathtt{x}_i > 0)\, P]\!]^\sharp(S^\sharp) & = & \mathbf{test}^\sharp_{\mathtt{x}_i \leq 0}(\mathbf{lfp}_{S^\sharp}\, f^\sharp) \text{ where} \\
 & & f^\sharp : S^\sharp \mapsto S^\sharp \sqcup [\![P]\!]^\sharp(\mathbf{test}^\sharp_{\mathtt{x}_i > 0}(S^\sharp))
\end{array}$$

Abstract Semantics Soundness

The analysis function $[\![.]\!]^\sharp$ obtained is sound:

Theorem

For all program P,

$$[\![P]\!](\gamma(S^\sharp)) \subseteq \gamma([\![P]\!]^\sharp(S^\sharp))$$

Soundness means no concrete behavior is left out: the analysis can be used to verify programs...

There are now **more possible origins for imprecisions:**

- the abstraction may not allow to express properties required for the proof
- even if the abstraction expresses the properties required for the proof, specific analysis steps may compute less precise properties

Outline

1. Abstraction
2. Abstract interpretation
3. Applications of abstract interpretation
4. A basic static analysis
5. A more realistic static analysis
6. **Termination of the Static Analysis**
7. Conclusion

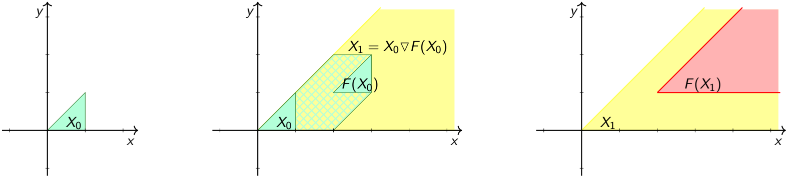
Enforcing Termination with Widening

We now **relax the finite height assumption**, which would cause the current framework **not to guarantee analysis termination**

Computing invariants about infinite executions with widening ∇

- **Widening** ∇ over-approximates $\cup$: **soundness guarantee**
- **Widening** ∇ guarantees the **termination of the analyses**
- Typical choice of ∇: **remove unstable constraints**

Example: iteration of the translation $(2, 1)$, with **octagons**



initial

iteration 1

iteration 2: stable !

Widening Operators

To enforce convergence, we use a widening operator:

Definition

A widening operator is a binary operator $\triangledown$ such that:

1. $\triangledown$ **over-approximates join:**

$$\forall S_0^\sharp, S_1^\sharp \in D^\sharp,\ \gamma(S_0^\sharp) \cup \gamma(S_1^\sharp) \subseteq \gamma(S_0^\sharp \triangledown S_1^\sharp)$$

2. $\triangledown$ **enforces termination:**
   if

$$S_{n+1}^\sharp = S_n^\sharp \triangledown Q_n^\sharp$$

   the sequence $S_n^\sharp$ converges, whatever the values of $Q_n^\sharp$

Widening Operator for the Domain of Intervals

Definition

We let $\triangledown$ be defined by:

$$[a, b_0] \triangledown [a, b_1] = \begin{cases} [a, +\infty[& \text{if } b_0 < b_1 \\ [a, b_0] & \text{if } b_0 \geq b_1 \end{cases}$$

and symmetrically for the left bound of intervals

Intuition:

- an interval is made of 0, 1 or 2 constraints
  (infinite bounds count for no constraint)
- widening may either keep the same number of constraints (stable) or decrease it
- but then, it can decrease it at most twice
  after that, all precision is lost, and the analysis necessarily converges

Fixpoint Approximation

Theorem

We consider two functions $f : \mathcal{P}(\mathbb{Z}^n) \to \mathcal{P}(\mathbb{Z}^n)$ and $f^\sharp : D^\sharp \to D^\sharp$ and two elements $M_0 \in \mathcal{P}(\mathbb{S}), S_0^\sharp \in D^\sharp$ such that:

- ∇ is a widening over $D^\sharp$
- f is continuous
- $f \circ \gamma \subseteq \gamma \circ f^\sharp$
- $M_0 \subseteq \gamma(S_0^\sharp)$
- $S_{n+1}^\sharp = S_n^\sharp \nabla f^\sharp(S_n^\sharp)$

Then, **both f has a least-fixpoint** and $(S_n^\sharp)_{n\in\mathbb{N}}$ converges, with a limit $S_\infty^\sharp$; moreover:

$$\mathop{\mathbf{lfp}}_{M_0} f \subseteq \gamma(S_\infty^\sharp)$$

- No more assumption on the monotonicity of $f^\sharp$!

Fixpoint Approximation: Proof

Proof:

- The existence of **lfp** f follows from Kleene's theorem; moreover:

$$\bigcup_{n\in\mathbb{N}} f^n(M_0)$$

- We can prove by induction over n that:

$$\bigcup_{k=0}^{n} f^k(M_0) \subseteq \gamma(S^\sharp_n)$$

- The definition of the widening entails that the sequence of abstract iterates is stationary, thus defines its limit $S^\sharp_\infty$. We let N be the first rank such that $S^\sharp_N = S^\sharp_\infty$.
- By definition of the least upper bound:

$$\bigcup_{k\in\mathbb{N}}^{n} f^k(M_0) \subseteq \gamma(S^\sharp_\infty)$$

Static Analysis of a Loop with a Widening Operator

- Widening in the combined domain:
  pointwise application of $\triangledown$
- The analysis simply replaces $\sqcup$ with $\triangledown$
- We recover the analysis convergence and soundness theorem:

Static analysis of a loop **while**$(\mathtt{x} \geq 0)\{P\}$

$$\left\{\begin{array}{l} S_0^\sharp \text{ be an approximation of the states before the loop} \\ S_{n+1}^\sharp = S_n^\sharp \triangledown [\![P]\!]^\sharp(\mathbf{test}^\sharp_{\mathtt{x}\geq 0}(S_n^\sharp)) \end{array}\right.$$

Then this sequence of iterates is monotone and eventually converges after finitely many iterates; we let $S_\infty^\sharp = \lim S_n^\sharp$.

Moreover, $S_\infty^\sharp$ is a sound over-approximation of the states at the loop head and:

$$[\![\mathbf{while}(\mathtt{x} \geq 0)\{P\}]\!](\gamma(S^\sharp)) \subseteq \gamma(\mathbf{test}^\sharp_{\mathtt{x}<0}(S_\infty^\sharp))$$

Improving Widening Operators

Widening may lose a lot of precision, if not used carefully

Thus, we generally employ **many tricks**:

- **do not apply widening right away**, i.e., use $\sqcup$ instead, for the first abstract iterations
- **use thresholds**, to give a chance to intermediate bounds to stabilize
- **alternate join and widening iterations**
- **compute post-widening iterations:** keep iterating the abstract function after the widening iteration computes an abstract post-fixpoint

An example analysis with the interval abstract domain

Classical techniques:

- add test values and neighbors as thresholds
- alternate join and widening
- no widening after visiting a new branch

```
int x = 0;
while(TRUE){
    if(x < 10 000){        9999 will be a threshold value at loop head
        x = x + 1;
    } else {
        x = −x;
    }
}
```

An example analysis with the interval abstract domain

Classical techniques:

- add test values and neighbors as thresholds
- alternate join and widening
- no widening after visiting a new branch

```
int x = 0;
            x ∈ [0, 0]
while(TRUE){

    if(x < 10 000){      9999 will be a threshold value at loop head

        x = x + 1;

    } else {

        x = −x;

    }

}
```

An example analysis with the interval abstract domain

Classical techniques:

- add test values and neighbors as thresholds
- alternate join and widening
- no widening after visiting a new branch

```
int x = 0;
            x ∈ [0, 0]
while(TRUE){
            x ∈ [0, 0]
    if(x < 10 000){        9999 will be a threshold value at loop head

        x = x + 1;

    } else {

        x = −x;

    }

}
```

Entering the loop

An example analysis with the interval abstract domain

Classical techniques:

- add test values and neighbors as thresholds
- alternate join and widening
- no widening after visiting a new branch

```
int x = 0;
            x ∈ [0, 0]
while(TRUE){
            x ∈ [0, 0]
    if(x < 10 000){        9999 will be a threshold value at loop head
            x ∈ [0, 0]
        x = x + 1;

    } else {
            x ∈ ∅
        x = −x;

    }

}
```

Only true branch possible

An example analysis with the interval abstract domain

Classical techniques:

- add test values and neighbors as thresholds
- alternate join and widening
- no widening after visiting a new branch

```
int x = 0;
            x ∈ [0, 0]
while(TRUE){
            x ∈ [0, 0]
    if(x < 10 000){      9999 will be a threshold value at loop head
            x ∈ [0, 0]
        x = x + 1;
            x ∈ [1, 1]
    } else {
            x ∈ ∅
        x = −x;
            x ∈ ∅
    }

}
```

Incrementation of interval

An example analysis with the interval abstract domain

Classical techniques:

- add test values and neighbors as thresholds
- alternate join and widening
- no widening after visiting a new branch

```
int x = 0;
                $x \in [0, 0]$
while(TRUE){
                $x \in [0, 0]$
    if(x < 10 000){      9999 will be a threshold value at loop head
                $x \in [0, 0]$
        x = x + 1;
                $x \in [1, 1]$
    } else {
                $x \in \emptyset$
        x = -x;
                $x \in \emptyset$
    }
                $x \in [1, 1]$
}
```

Propagation

An example analysis with the interval abstract domain

Classical techniques:

- add test values and neighbors as thresholds
- alternate join and widening
- no widening after visiting a new branch

```
int x = 0;
            $x \in [0, 0]$
while(TRUE){
            $x \in [0, 1]$
    if(x < 10 000){        9999 will be a threshold value at loop head
            $x \in [0, 0]$
        x = x + 1;
            $x \in [1, 1]$
    } else {
            $x \in \emptyset$
        x = -x;
            $x \in \emptyset$
    }
            $x \in [1, 1]$
}
```

Join at loop head

An example analysis with the interval abstract domain

Classical techniques:

- add test values and neighbors as thresholds
- alternate join and widening
- no widening after visiting a new branch

```
int x = 0;
            x ∈ [0, 0]
while(TRUE){
            x ∈ [0, 1]
    if(x < 10 000){        9999 will be a threshold value at loop head
            x ∈ [0, 1]
        x = x + 1;
            x ∈ [1, 1]
    } else {
            x ∈ ∅
        x = −x;
            x ∈ ∅
    }
            x ∈ [1, 1]
}
```

Still only the true branch is possible

An example analysis with the interval abstract domain

Classical techniques:

- add test values and neighbors as thresholds
- alternate join and widening
- no widening after visiting a new branch

```
int x = 0;
            x ∈ [0, 0]
while(TRUE){
            x ∈ [0, 1]
    if(x < 10 000){      9999 will be a threshold value at loop head
            x ∈ [0, 1]
        x = x + 1;
            x ∈ [1, 2]
    } else {
            x ∈ ∅
        x = −x;
            x ∈ ∅
    }
            x ∈ [1, 1]
}
```

Incrementation of interval

An example analysis with the interval abstract domain

Classical techniques:

- add test values and neighbors as thresholds
- alternate join and widening
- no widening after visiting a new branch

```
int x = 0;
                x ∈ [0, 0]
while(TRUE){
                x ∈ [0, 1]
    if(x < 10 000){        9999 will be a threshold value at loop head
                x ∈ [0, 1]
        x = x + 1;
                x ∈ [1, 2]
    } else {
                x ∈ ∅
        x = −x;
                x ∈ ∅
    }
                x ∈ [1, 2]
}
```

Propagation

An example analysis with the interval abstract domain

Classical techniques:

- add test values and neighbors as thresholds
- alternate join and widening
- no widening after visiting a new branch

```
int x = 0;
          x ∈ [0, 0]
while(TRUE){
          x ∈ [0, 9999]    instead of [0, +∞[
    if(x < 10 000){       9999 will be a threshold value at loop head
          x ∈ [0, 1]
        x = x + 1;
          x ∈ [1, 2]
    } else {
          x ∈ ∅
        x = −x;
          x ∈ ∅
    }
          x ∈ [1, 2]
}
```

Widening at the loop head, + threshold

An example analysis with the interval abstract domain

Classical techniques:

- add test values and neighbors as thresholds
- alternate join and widening
- no widening after visiting a new branch

```
int x = 0;
             x ∈ [0, 0]
while(TRUE){
             x ∈ [0, 9999]   instead of [0, +∞[
     if(x < 10 000){    9999 will be a threshold value at loop head
             x ∈ [0, 9999]
          x = x + 1;
             x ∈ [1, 2]
     } else {
             x ∈ ∅
          x = −x;
             x ∈ ∅
     }
             x ∈ [1, 2]
}
```

Now both branches are possible...

An example analysis with the interval abstract domain

Classical techniques:

- add test values and neighbors as thresholds
- alternate join and widening
- no widening after visiting a new branch

```
int x = 0;
            x ∈ [0, 0]
while(TRUE){
            x ∈ [0, 9999]    instead of [0, +∞[
    if(x < 10 000){    9999 will be a threshold value at loop head
            x ∈ [0, 9999]
        x = x + 1;
            x ∈ [1, 10000]
    } else {
            x ∈ ∅
        x = −x;
            x ∈ ∅
    }
            x ∈ [1, 2]
}
```

Numerical assignments

An example analysis with the interval abstract domain

Classical techniques:

- add test values and neighbors as thresholds
- alternate join and widening
- no widening after visiting a new branch

```
int x = 0;
            $x \in [0, 0]$
while(TRUE){
            $x \in [0, 9999]$    instead of $[0, +\infty[$
    if($x < 10\,000$){     9999 will be a threshold value at loop head
            $x \in [0, 9999]$
        x = x + 1;
            $x \in [1, 10000]$
    } else {
            $x \in \emptyset$
        x = -x;
            $x \in \emptyset$
    }
            $x \in [1, 10000]$
}
```

Join at the end of the loop

An example analysis with the interval abstract domain

Classical techniques:

- add test values and neighbors as thresholds
- alternate join and widening
- no widening after visiting a new branch

```
int x = 0;
            $x \in [0, 0]$
while(TRUE){
            $x \in [0, 10000]$   instead of $[-\infty, +\infty[$
    if($x < 10\,000$){      9999 will be a threshold value at loop head
            $x \in [0, 9999]$
        x = x + 1;
            $x \in [1, 10000]$
    } else {
            $x \in \emptyset$
        x = -x;
            $x \in \emptyset$
    }
            $x \in [1, 10000]$
}
```

Join after widening

An example analysis with the interval abstract domain

Classical techniques:

- add test values and neighbors as thresholds
- alternate join and widening
- no widening after visiting a new branch

```
int x = 0;
            x ∈ [0, 0]
while(TRUE){
            x ∈ [0, 10000]    instead of [−∞, +∞[
     if(x < 10 000){     9999 will be a threshold value at loop head
            x ∈ [0, 9999]
          x = x + 1;
            x ∈ [1, 10000]
     } else {
            x ∈ [10000, 10000]    instead of [10000, +∞[
          x = −x;
            x ∈ ∅
     }
            x ∈ [1, 10000]
}
```

True branch stable, false branch visited for the first time

An example analysis with the interval abstract domain

Classical techniques:

- add test values and neighbors as thresholds
- alternate join and widening
- no widening after visiting a new branch

```
int x = 0;
            x ∈ [0, 0]
while(TRUE){
            x ∈ [0, 10000]     instead of [−∞, +∞[
    if(x < 10 000){      9999 will be a threshold value at loop head
            x ∈ [0, 9999]
        x = x + 1;
            x ∈ [1, 10000]
    } else {
            x ∈ [10000, 10000]     instead of [10000, +∞[
        x = −x;
            x ∈ [−10000, −10000]
    }
            x ∈ [1, 10000]
}
```

True branch stable, false branch visited for the first time

An example analysis with the interval abstract domain

Classical techniques:

- add test values and neighbors as thresholds
- alternate join and widening
- no widening after visiting a new branch

```
int x = 0;
                x ∈ [0, 0]
while(TRUE){
                x ∈ [0, 10000]     instead of [−∞, +∞[
      if(x < 10 000){       9999 will be a threshold value at loop head
                x ∈ [0, 9999]
            x = x + 1;
                x ∈ [1, 10000]
      } else {
                x ∈ [10000, 10000]     instead of [10000, +∞[
            x = −x;
                x ∈ [−10000, −10000]
      }
                x ∈ [−10000, 10000]
}
```

Join at the end of the loop

An example analysis with the interval abstract domain

Classical techniques:

- add test values and neighbors as thresholds
- alternate join and widening
- no widening after visiting a new branch

```
int x = 0;
                x ∈ [0, 0]
while(TRUE){
                x ∈ [−10000, 10000]     instead of [−∞, +∞[
      if(x < 10 000){      9999 will be a threshold value at loop head
                x ∈ [0, 9999]
            x = x + 1;
                x ∈ [1, 10000]
      } else {
                x ∈ [10000, 10000]     instead of [10000, +∞[
            x = −x;
                x ∈ [−10000, −10000]
      }
                x ∈ [−10000, 10000]
}
```

Join again: no widening after visiting a new branch

An example analysis with the interval abstract domain

Classical techniques:

- add test values and neighbors as thresholds
- alternate join and widening
- no widening after visiting a new branch

```
int x = 0;
            $x \in [0, 0]$
while(TRUE){
            $x \in [-10000, 10000]$    instead of $[-\infty, +\infty[$
     if(x < 10 000){       9999 will be a threshold value at loop head
            $x \in [-10000, 9999]$
          x = x + 1;
            $x \in [1, 10000]$
     } else {
            $x \in [10000, 10000]$    instead of $[10000, +\infty[$
          x = -x;
            $x \in [-10000, -10000]$
     }
            $x \in [-10000, 10000]$
}
```

Branches

An example analysis with the interval abstract domain

Classical techniques:

- add test values and neighbors as thresholds
- alternate join and widening
- no widening after visiting a new branch

```
int x = 0;
                $x \in [0, 0]$
while(TRUE){
                $x \in [-10000, 10000]$    instead of $[-\infty, +\infty[$
    if($x < 10\,000$){     9999 will be a threshold value at loop head
                $x \in [-10000, 9999]$
        $x = x + 1$;
                $x \in [-9999, 10000]$
    } else {
                $x \in [10000, 10000]$    instead of $[10000, +\infty[$
        $x = -x$;
                $x \in [-10000, -10000]$
    }
                $x \in [-10000, 10000]$
}
```

Incrementation of interval in true branch; false branch stable

An example analysis with the interval abstract domain

Classical techniques:

- add test values and neighbors as thresholds
- alternate join and widening
- no widening after visiting a new branch

```
int x = 0;
                x ∈ [0, 0]
while(TRUE){
                x ∈ [−10000, 10000]    instead of [−∞, +∞[
    if(x < 10 000){         9999 will be a threshold value at loop head
                x ∈ [−10000, 9999]
        x = x + 1;
                x ∈ [−9999, 10000]
    } else {
                x ∈ [10000, 10000]    instead of [10000, +∞[
        x = −x;
                x ∈ [−10000, −10000]
    }
                x ∈ [−10000, 10000]
}
```

Everything is stable; exact ranges inferred

Outline

1. Abstraction
2. Abstract interpretation
3. Applications of abstract interpretation
4. A basic static analysis
5. A more realistic static analysis
6. Termination of the Static Analysis
7. Conclusion

Summary

The last two lectures:

- **abstraction** and its formalization
- **computation of an abstract semantics** in a very simplified case
- **more realistic static analysis**

Next lectures:

- **construction** of a few **non trivial abstractions**
- **more advanced static analysis techniques**