Abstract Interpretation-Based Static Analysis — Caterina Urban

Lecture 1 - *July 2, 2025*

1 Preliminaries

Abstract Interpretation is the modeling of a program's behavior. Rice's theorem showed that there is no general method to prove non-trivial properties of all programs. As such, Abstract Interpretation provides sound approximations of program properties. It models programs as discrete states and specifies which states are allowed or not. Abstract Interpretation is based on order theory. In this section, we will first introduce partially ordered sets (poset),Galois connections, and Kleene's fixpoint theorem.

There are three parts in Abstract Interpretation.

1. Practical tools
2. Abstract semantics, domains and algorithmic approaches to decide program properties
3. Concrete semantics, which are mathematical models of program behavior

1.1 Order Theory(Points, Chains, Lattices)

A **partially ordered set (poset)** is a pair $(X, \leq)$ where:

- X is a set,
- $\leq$ is a binary relation on X that is
  - reflexive: $\forall x \in X, x \leq x$,
  - antisymmetric: if $x \leq y$ and $y \leq x$, then $x = y$,
  - transitive: if $x \leq y$ and $y \leq z$, then $x \leq z$.

A **totally ordered set** has all the above properties, and the additional property that the binary relation is:

- total: $\forall x, y, z \in X, (x \leq y) \vee (y \leq z)$

For a function $f : X \mapsto X$,

- x is a fixpoint of f if $x = f(x)$
- x is a pre-fixpoint of f if $x \geq f(x)$
- x is a post-fixpoint of f if $x \leq f(x)$

A **chain** of a poset $(X, \leq)$ is a a subset of $S \subseteq X$ that is totally ordered.

- S is a chain of $(X, \leq)$ if $\forall u, v \in S, (u \leq v) \vee (v \leq u)$

A poset is **completely ordered** if every chain in the poset has a supremum.

- Formal representation TBD

A complete poset $(X, \leq, \wedge, \vee, \top, \bot)$ would always be uniquely bounded with a supremum $\top$ and an infimum $\bot$.

- $\vee$ is the join of elements
- $\wedge$ is the meet of elements
- $\forall x \in X, \bot \leq x$
- $\forall x \in X, x \leq \top$

1.2 Galois Connection

A Galois Connection is an isomorphism between two posets.

- Connect two posets $(C, \leq_C)$ and $(A, \leq_A)$ via:
  - Abstraction function $\alpha : C \to A$.
  - Concretization function $\gamma : A \to C$.
- Satisfying:

$$\forall c \in C, a \in A : \alpha(c) \leq_A a \iff c \leq_C \gamma(a).$$

1.3 Kleene's Fixpoint Theorem

Kleene's theorem states that, in a complete partial order (CPO), for a Scott-continuous and monotonic function f, the ascending chain starting from an initial element c—given by the sequence $f^0(c), f^1(c), \ldots$—converges to the least fixpoint of f:

$$\mathrm{lfp}_{\leq c}\, f = \bigsqcup_{i \geq 0} f^i(c),$$

This result forms the theoretical foundation of defining program semantics as fixpoints in abstract interpretation.

- **Prefix Trace Semantics**:
  Defines the set of all finite prefix traces starting from an initial set of states $I \in \mathcal{P}(\Sigma)$:

  $$\mathcal{T}_p(I) = \{s_0, \ldots, s_n \mid n \geq 0, s_0 \in I, \forall i : \langle s_i, s_{i+1} \rangle \in \tau\}.$$

  This can be expressed as the least fixpoint:

  $$\mathcal{T}_p(I) = \mathrm{lfp}_{\subseteq \emptyset} F_p, \quad F_p(T) = I \cup (T; \tau),$$

  where $T; \tau$ denotes the set of traces extended with transitions according to τ.

- **Prefix State Abstraction**:
  Maps prefix traces to their ending states, defined by the Galois connection (α_p, γ_p):

  $$\alpha_p(T) = \{s \in \Sigma \mid \exists \text{ trace ending in } s\},$$

  $$\gamma_p(S) = \{\text{all traces ending in some } s \in S\}.$$

  By Kleene's fixpoint transfer theorem, we have

  $$\alpha_p(\mathcal{T}_p(I)) = \mathcal{R}(I),$$

  where $\mathcal{R}(I)$ denotes the forward reachability semantics.

- **Forward Reachability Semantics**:
  Defines the set of program states reachable from an initial set I:

  $$\mathcal{R}(I) = \{s \mid \exists n \geq 0, s_0 \in I, s = s_n, \forall i : \langle s_i, s_{i+1} \rangle \in \tau\},$$

  which can be written in least fixpoint form:

  $$\mathcal{R}(I) = \mathrm{lfp}_{\subseteq \emptyset} F_r, \quad F_r(S) = I \cup \mathrm{post}(S),$$

  where

  $$\mathrm{post}(S) = \{s' \in \Sigma \mid \exists s \in S, \langle s, s' \rangle \in \tau\}.$$

2 A numerical execution model

We introduce a language saturating **the art of losing precision**, capable of assigning ranges of values to a variable.

2.1 Syntax

- Numeric Expression: $e := c \mid [c, c] \mid x \mid -e \mid e \diamond e$
- Program Syntax: $s := x \leftarrow e \mid s; s \mid \textbf{if } e \textbf{ then } s \textbf{ else } s \mid \textbf{while } e \textbf{ do } s \textbf{ done}$

2.2 Expression Semantics

- $\rho : \mathbb{X} \to \mathbb{Z}$
- $\mathbb{E} : (\mathbb{X} \to \mathbb{Z}) \to \mathcal{P}(\mathbb{Z})$

2.3 Transition Semantics

- Programs modeled as transition systems:
  - Executions modeled as transitions between states.
  - States are environments/memory at specific program points.
  - Transitions represent relations, not necessarily functions.
- For assignments:
  - Transitions map an environment before assignment to one after.
- For control structures:
  - If-statements: evaluate condition, transition to body if true, otherwise skip.
  - While-loops: repeated transitions until condition is false.

2.4 Labeling

Statements syntax:

```
stmt ::= \ell X \leftarrow expr \ell (assignment, \ell \in \mathcal{L})
      | if \ell expr \bowtie 0 then stmt end \ell (conditional, \bowtie \in \{=, \leq, \dots\}
      | while \ell expr \bowtie 0 do stmt done \ell (loop)
      | stmt; stmt (sequence)
```

Transition semantics definitions using labels:

$$\Sigma \stackrel{\text{def}}{=} \mathcal{L} \times (\mathbb{X} \to \mathbb{Z}),$$

$$\tau \subseteq \Sigma \times \Sigma,$$

$$\tau[[\ell_1 X \leftarrow e \ell_2]] \stackrel{\text{def}}{=} \{((\ell_1, \rho), (\ell_2, \rho[X \mapsto v])) \mid \rho \in \mathcal{E}, v \in E[[e]]\rho\},$$

$$\tau[\text{if } \ell_1 e \bowtie 0 \text{ then } \ell_2 s \ell_3 \text{ end } \ell_4] \stackrel{\text{def}}{=} \text{(see slides for complete expression)},$$

$$\tau[\text{while } \ell_1 e \bowtie 0 \text{ do } \ell_2 s \ell_3 \text{ done } \ell_4] \stackrel{\text{def}}{=} \text{(see slides for complete expression)}.$$

3 Concrete Semantics

Concrete semantics:

- State properties: sets of reachable states with desired properties.
- Trace properties: e.g., termination, liveness.

While state semantics capture individual states, trace semantics capture sequences of states as the program executes. State semantics are therefore abstractions of trace semantics, in that you can get from trace semantics to state semantics by removing parts of the sequence in a state. For example, if you remove all but the last state of a trace semantics, you get the forward state semantics, which are the final reachable states of a program.

- Σ — the set of **program states**. Each state typically combines a control location and a store:
$$\Sigma = \mathcal{L} \times (\mathbb{X} \to \mathbb{Z})$$
- $\tau \subseteq \Sigma \times \Sigma$ — the **transition relation** between program states. A pair $(s, s') \in \tau$ means the program can move from state s to state s' in one step.
- Σ^* — the set of all **finite sequences of states**, also known as finite traces. Formally:
$$\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$$
- Σ^ω — the set of all **infinite sequences of states**. These represent non-terminating or infinite executions.
- $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$ — the set of all traces, both finite and infinite.

3.1 Maximal Trace Semantics

The lowest and least abstracted level of trace semantics are maximal trace semantics. Maximal trace semantics capture all possible traces of a program, finite and infinite. More formally we define maximal trace semantics as:

Definition: The set of all finite and infinite traces a program may produce.

Let Σ be the set of program states, $\tau \subseteq \Sigma \times \Sigma$ be the transition relation, and $\mathcal{B}$ be the set of blocking (final) states:

$$\mathcal{M} \triangleq \{s_0 \dots s_n \in \Sigma^* \mid s_n \in \mathcal{B} \wedge \forall i < n : (s_i, s_{i+1}) \in \tau\} \cup \{s_0 s_1 \dots \in \Sigma^\omega \mid \forall i : (s_i, s_{i+1}) \in \tau\}$$

Fixpoint Formulation:

$$\mathcal{M} = \mathrm{lfp}_{\subseteq}^{\Sigma^\omega} \; F(T) \quad \text{where } F(T) = \mathcal{B} \cup (\tau; T)$$

3.2 Partial Finite Trace Semantics

Partial finite trace semantics are the direct abstraction of maximal trace semantics. Partial finite trace semantics is made of all non-terminating traces of maximal traces. But it does not directly remove all non-terminating traces, it keeps the finite prefixes of infinite traces. Formally, we define:

Definition: The set of all finite valid execution prefixes (not necessarily terminating).

$$\mathcal{T} \triangleq \{s_0 \dots s_n \in \Sigma^* \mid \forall i < n : (s_i, s_{i+1}) \in \tau\}$$

Fixpoint Formulation:

$$\mathcal{T} = \mathrm{lfp}_{\subseteq}^{\emptyset} \; F_p^* \quad \text{where } F_p^*(T) = \Sigma \cup (T; \tau)$$

3.3 Partial Finite Trace Abstraction

Maximal trace semantics are abstracted to partial finite trace semantics by the partial finite trace abstraction. It is made by combining a finite prefix abstraction (which simply returns all finite traces in T) and a suffix abstraction, which returns the prefix of infinite traces. A trace σ is in the prefix abstraction of T if and only if it can be extended by some suffix σ' such that the full trace $\sigma \cdot \sigma'$ is in T.

- **Finite prefix abstraction:** $\alpha^*(T) = T \cap \Sigma^*$
- **Suffix abstraction:** $\alpha_{\preceq}(T) = \{\sigma \in \Sigma^\infty \mid \exists \sigma' : \sigma \cdot \sigma' \in T\}$

3.4 Prefix Trace Semantics

Definition: The set of all finite traces starting from initial states $I \subseteq \Sigma$.

$$\mathcal{T}_p(I) = \{s_0 \dots s_n \in \Sigma^* \mid s_0 \in I \wedge \forall i < n : (s_i, s_{i+1}) \in \tau\}$$

Fixpoint Formulation:

$$\mathcal{T}_p(I) = \mathrm{lfp}^{\emptyset}_{\subseteq} F_p \quad \text{where } F_p(T) = I \cup (T; \tau)$$

3.5 Suffix Trace Semantics

Definition: The set of all finite traces ending in final states $F \subseteq \Sigma$.

$$\mathcal{T}_s(F) = \{s_0 \dots s_n \in \Sigma^* \mid s_n \in F \wedge \forall i < n : (s_i, s_{i+1}) \in \tau\}$$

Fixpoint Formulation:

$$\mathcal{T}_s(F) = \mathrm{lfp}^{\emptyset}_{\subseteq} F_s \quad \text{where } F_s(T) = F \cup (\tau; T)$$

3.6 Prefix / Suffix and Partial Trace Abstractions

Prefix and suffix trace semantics can be obtained from finite partial trace semantics by applying their relevant abstractions.

- **Prefix Abstraction** α_I:
$$\alpha_I(T) = T \cap (I \cdot \Sigma^*)$$
- **Suffix Abstraction** α_F:
$$\alpha_F(T) = T \cap (\Sigma^* \cdot F)$$
- **Finite Prefix Abstraction** α^*:
$$\alpha^*(T) = T \cap \Sigma^*$$

Forward Reachability Semantics

Definition: The set of states reachable from initial states $I \subseteq \Sigma$.

$$\mathcal{R}(I) = \{s \in \Sigma \mid \exists s_0, \dots, s_n : s_0 \in I, s_n = s, \forall i < n : (s_i, s_{i+1}) \in \tau\}$$

Fixpoint Formulation:

$$\mathcal{R}(I) = \mathrm{lfp}^{\emptyset}_{\subseteq} F_r \quad \text{where } F_r(S) = I \cup \mathrm{post}(S)$$

3.7 Backward Reachability Semantics

Definition: The set of states that can reach final states $F \subseteq \Sigma$.

$$\mathcal{C}(F) = \{s \in \Sigma \mid \exists s_0, \dots, s_n : s_0 = s, s_n \in F, \forall i < n : (s_i, s_{i+1}) \in \tau\}$$

Fixpoint Formulation:

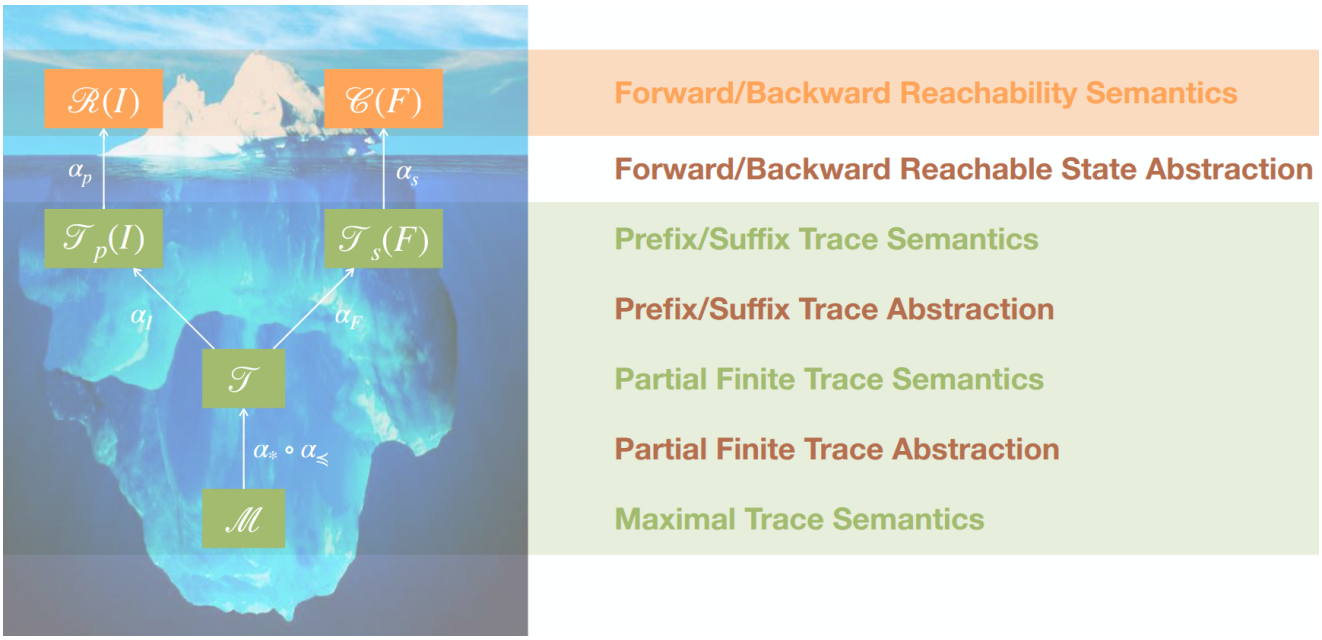
$$\mathcal{C}(F) = \mathrm{lfp}^{\emptyset}_{\subseteq} F_c \quad \text{where } F_c(S) = F \cup \mathrm{pre}(S)$$

3.8 Reachable State Abstractions

Likewise, forward reachable and backward reachable semantics can be obtained by applying the reachable state abstractions.

- **Forward:** $\alpha_p(T) = \{s \in \Sigma \mid \exists s_0, \dots, s \in T\}$
- **Backward:** $\alpha_s(T) = \{s \in \Sigma \mid \exists s, \dots, s_n \in T\}$

4 Abstract Semantics



Abstract Semantics: A sound approximation of the concrete semantics computed within an abstract domain.

Abstract Domain: A lattice or complete partial order $\langle A, \sqsubseteq \rangle$ used to represent and reason about program properties.

- **Abstraction function** $\alpha : C \to A$
- **Concretization function** $\gamma : A \to C$
- Galois connection condition:

$$\forall c \in C, a \in A : \alpha(c) \sqsubseteq a \iff c \in \gamma(a)$$

- Abstract semantics: defined by abstract transformers $f^\sharp : A \to A$
- Correctness: $\alpha(f(c)) \sqsubseteq f^\sharp(\alpha(c))$
  - Overapproximate reachable states.

Abstract domains define a way to specify properties of the program. These are examples of abstract domains:

- Sign Domain
  - Abstracts integers into signs: $\{< 0, = 0, > 0, \leq 0, \geq 0, \neq 0, \top, \bot\}$.
  - Non-relational: tracks each variable individually.
  - Operations like $+, -, \times$ are interpreted over signs using lookup tables.
- Interval Domain
  - Represents values as intervals: $[a, b]$ where $a, b \in \mathbb{Z} \cup \{\pm\infty\}$.
  - Non-relational but more precise than sign domain.
  - Arithmetic operations performed via interval arithmetic.
  - Supports **widening** to ensure convergence of fixpoint computations.
- Polyhedra Domain
  - Tracks linear inequalities over variables (e.g., $x + y \leq 5$).
  - Relational domain: captures relationships between variables.
  - Most precise but computationally expensive.