Abstract Interpretation

Chapter 12: Collecting Semantics & Galois Connections

Oral Exam Script

Target: 15 Minutes

1. Introduction (Approx. 2 Minutes)

Good morning. Today I will be presenting Chapter 12 on Abstract Interpretation.

In the previous parts of this course, we have built various static analyses intuitively. However, a critical question remains: *How do we verify that these analyses are actually correct?*

We know from Rice's Theorem that calculating non-trivial properties of runtime behavior perfectly is undecidable. We cannot compute the exact set of all possible states for infinite loops or recursive structures.

To solve this, we use Abstract Interpretation. This is a formal mathematical framework that bridges the gap between the infinite "Concrete" execution of a program and a finite "Abstract" approximation.

My goal is to show how we define the "Ground Truth" (Collecting Semantics) and strictly link it to our analysis using Galois Connections to guarantee Soundness.

Whiteboard Action:

- **The Problem:** Undecidability (Rice's Theorem).
- **The Solution:** Finite Approximation.
- **Concrete Domain (C):** $\mathcal{P}(\mathbb{Z})$ (e.g., $\{0, 2, 4, \dots\}$)
- **Abstract Domain (A):** Signs, Intervals, Parity, etc.

Part 1: The Framework & Galois Connections (4 Minutes)

To perform this analysis formally, we need two domains and a way to translate between them.

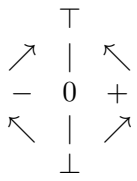
First, we have the **Collecting Semantics**. This represents the ground truth. It is the set of all concrete states reachable at a specific program point. As we established, this set is often infinite.

Therefore, we map it to an **Abstract Domain** (L). To link them, we define a pair of functions: 1. **Abstraction (α):** Maps a set of concrete values to the "best" abstract representation. 2. **Concretization (γ):** Maps an abstract value back to the set of concrete meanings it represents.

Together, these form a **Galois Connection**. This connection relies on two properties to ensure we don't lose safety or precision.

Whiteboard Action:

Draw the Lattice (Sign Analysis):



Write the Galois Properties:

1. $c \subseteq \gamma(\alpha(c))$ **(Extensive / Safe)**
2. $\alpha(\gamma(a)) \sqsubseteq a$ **(Reductive / Precise)**

Property #1 is the most critical. It essentially says: If we abstract a value and then concretize it back, the resulting set must be *larger or equal* to the original. We might add noise (imprecision), but we never lose data (safety).

Property #2 is the **Reductive** property. It ensures our abstraction doesn't drift into useless vagueness. It says: If I take an abstract state (like "Positive"), turn it into numbers, and then re-abstract it, I should get "Positive" back—not "Unknown" ($\top$). If the result were larger than a, it would mean our abstraction function generated noise out of thin air. This property guarantees the abstraction is as precise as possible for the chosen domain.

Part 2: Soundness Constraints (4 Minutes)

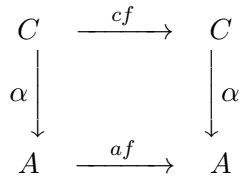
Now that we have the domains, how do we define the analysis itself? We use the **Soundness Condition**.

We do not check the whole program at once. Instead, we verify it instruction by instruction. We compare the Concrete Step (cf) against the Abstract Step (af).

Visually, this forms a commutative diagram.

Whiteboard Action:

Draw the Soundness Square:



Write the Soundness Inequality:

$$\alpha(cf(c)) \sqsubseteq af(\alpha(c))$$

Let's trace this inequality: 1. On the Left Side ($\alpha(cf(c))$): We run the code concretely, then

abstract the result. This is the "Truth". 2. On the Right Side ($af(\alpha(c))$): We abstract the input first, then run the analysis function.

The inequality ($\sqsubseteq$) guarantees that our analysis covers *at least* everything that actually happens in the concrete execution. By Tarski's Fixed Point Theorem, if this holds for every step (local soundness), the entire analysis is sound.

Part 3: Precision & Completeness (4 Minutes)

Soundness means we aren't wrong, but it doesn't mean we are useful. If we always returned "Top" (Unknown), we would be sound, but useless.

We strive for two higher goals: **Optimality** and **Completeness**.

Optimality defines the best possible abstract function. Mathematically, it is defined by inducing the function from the concrete world:

Whiteboard Action:

Write Optimality:

$$af_{opt} = \alpha \circ cf \circ \gamma$$

Completeness is an even stricter requirement. Completeness implies there is NO loss of precision during the analysis.

Whiteboard Action:

Write Completeness:

$$\alpha \circ cf = af \circ \alpha$$

However, in Abstract Interpretation, completeness is rare. Consider the expression $(+1)+(-1)$.

In the concrete world, the result is exactly 0. In the abstract world of signs, adding a Positive to a Negative results in $\top$ (it could be anything). Because $\alpha(0) \sqsubset \top$, the analysis is Incomplete.

Conclusion (¡ 1 Minute)

To summarize:

- We use **Collecting Semantics** to define the concrete ground truth.
- We use **Galois Connections** (α and γ) to formally link the concrete and abstract worlds.
- We rely on the **Soundness Square** to prove that our abstract transfer functions never miss a possible program state.

This framework provides the mathematical proof required to trust static analysis tools. Thank you. I am happy to take any questions.