Oral Exam: SPA Chapter 12

Abstract Interpretation (Full 15 Min Version)

Script & Cheat Sheet

0. Introduction (2 Minutes)

Whiteboard Action

Write the Agenda in the top corner (keep this visible):

1. Collecting Semantics (The Truth)
2. Galois Connections (α, γ)
3. Soundness (The Proof)
4. Optimality & Completeness

Say: Good morning/afternoon. Today I will be presenting Chapter 12 on **Abstract Interpretation**.

Say: In the earlier parts of this course, we learned how to build static analyses—like Sign Analysis or Constant Propagation. We built them intuitively, using lattices and flow functions. But a critical question remains: *How do we know they are right?*

Say: If a tool tells me a variable is "Positive," can I bet my life on it? Or is it possible that, in some rare edge case, it actually becomes negative?

Say: Abstract Interpretation is the mathematical framework that answers this. It bridges the gap between the messy, infinite reality of running code (the concrete world) and the clean, finite world of static analysis (the abstract world).

Say: My goal today is to walk you through the four pillars of this framework: The Ground Truth, The Translation, The Guarantee of Safety, and finally, The Precision.

---

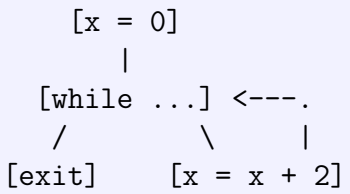
1. Collecting Semantics (4 Minutes)

Say: To prove our analysis is correct, we first need a reference point. We need to define exactly what the program does at runtime, without any simplification. We call this the **Collecting Semantics**.

Say: Imagine we run a program. At every control flow node, we record the state of the memory. If we run it again with different input, we get different states. The Collecting Semantics is the set of **all possible concrete states** that can ever exist at a specific point.

Whiteboard Action

Draw this CFG in the center of the board:



Write next to the loop head:

$$C = \mathcal{P}(\mathbb{Z})$$

$$S = \{0, 2, 4, 6, \dots\}$$

Say: Let's look at this simple loop. We initialize x to 0, and loops increment it by 2. → *[Action: Point to the loop head]* At this point in the code, what are the possible values of x?

Say: It starts at 0. Then 2. Then 4. It is the set of all even positive integers. This set, $\{0, 2, 4, \dots\}$, is the Collecting Semantics for this node.

Say: We define this mathematically as the Least Fixed Point of a concrete transfer function, cf. We are working in the Concrete Domain, C, which is usually the Power Set of all possible states.

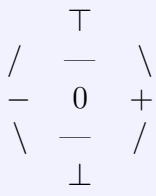
Say: Now, why don't we just use this as our analysis? Because this set is infinite. By Rice's Theorem, calculating non-trivial properties of this set is undecidable. We literally cannot compute it for general programs. So, we must simplify.

2. Abstraction & Concretization (4 Minutes)

Say: To simplify, we move from the Concrete Domain (C) to an **Abstract Domain** (A). For this presentation, I will use **Sign Analysis** as the example, because it's visually clear.

Whiteboard Action

Draw the Sign Lattice clearly:



Say: Here, we have a finite lattice. To relate the infinite sets of integers (C) to these simple symbols (A), we need two translation functions.

Whiteboard Action

Write clearly:

$$\alpha : C \to A \quad \text{(Abstraction)}$$

$$\gamma : A \to C \quad \text{(Concretization)}$$

Say: 1. **Abstraction (α)**: Maps a set of values to the "best" abstract representation. For example, $\alpha(\{2, 4, 100\}) = +$.

Say: 2. **Concretization (γ)**: Maps an abstract value back to the *meaning* it holds. For example, $\gamma(+)$ is the set of all positive integers $\{1, 2, 3, \dots\}$.

Say: These two cannot just be random functions. They must form a **Galois Connection**. This requires two specific properties.

Whiteboard Action

Write the properties:

1. $c \subseteq \gamma(\alpha(c))$ (Extensive)
2. $\alpha(\gamma(a)) \sqsubseteq a$ (Reductive)

Say: The first one—Extensivity—is the most important for safety. *→ [Action: Point to property 1]* It says: If I take a concrete set c, translate it to abstract, and then translate it back... I must get a set that is **larger than or equal to** where I started.

Say: Think of it like taking a low-resolution photo. You lose the fine details (the specific numbers), but the "truth" (the fact that they are positive) is still captured inside the blurry image. We added noise, but we didn't lose the subject. This ensures we never

overlook a possible runtime behavior.

---

3. Soundness (3 Minutes)

Say: Now we can define **Soundness**. An analysis is sound if the abstract result safely over-approximates the collecting semantics.

Whiteboard Action

Write:

$$\alpha(\text{Real}) \sqsubseteq \text{Analysis}$$

Say: To prove this, we don't check the whole program at once. We check it instruction by instruction. We use a diagram to compare the Concrete Step (cf) vs. the Abstract Step (af).

Whiteboard Action

Draw the Soundness Square (Make this big!):

$$\begin{array}{ccc} C & \xrightarrow{cf} & C \\ {\scriptstyle\alpha}\downarrow & & \downarrow{\scriptstyle\alpha} \\ A & \xrightarrow{af} & A \end{array}$$

Write the inequality:

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

Say: Let's trace this. → *[Action: Trace the path: Right then Down]* If we run the code concretely (cf) and then abstract the result... → *[Action: Trace the path: Down then Right]* That result must be **less than or equal to** ($\sqsubseteq$) what we get if we abstract first, and then run the analysis function (af).

Say: In plain English: "The analysis calculation (af) must cover everything that actually happens in the code (cf)." If this holds for every operation (plus, minus, assign), then the Tarski Fixed Point Theorem guarantees the whole analysis is sound.

4. Optimality & Completeness (2 Minutes)

Say: Soundness just means "don't be wrong." But we also want to be precise. This brings us to **Optimality** and **Completeness**.

Say: **Optimality** is about defining the best possible abstract function.

Whiteboard Action

Write:

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

Say: If we define our abstract addition by: concretizing the inputs, adding them effectively, and abstracting the result, we have the optimal definition.

Say: However, even optimal functions can lose information. If we have NO loss of precision, we call it **Complete**.

Whiteboard Action

Write:

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

Say: Notice the equals sign. This rarely happens. Sign analysis is **Incomplete** for addition.

Whiteboard Action

Write:

$$(+1) + (-1) = 0$$

$$\alpha(\text{result}) = \text{"0"}$$

vs

$$(+) \oplus (-) = \top$$

Say: Concrete math gives us 0. Abstract math adds a positive and a negative, which results in "Top" (Unknown). Since Top is less precise than "0", the analysis is incomplete. It's safe, but fuzzy.

---

5. Conclusion (Top of the hour)

Say: To wrap up: Abstract Interpretation allows us to formally verify static analysis. We use **Collecting Semantics** as our ground truth. We use **Galois Connections** to translate that truth into a computable lattice. And we use the **Soundness Theorem** to prove that our tool never misses a bug.

Say: Thank you. I am ready for your questions.

Cheat Sheet: SPA Chapter 12

(Keep this on the table for quick reference)

1. Key Definitions

- **Collecting Semantics:** The ground truth. The set of all concrete states reachable at a program point. $C = \mathcal{P}(States)$.
- **Abstraction (α):** $C \to A$. Maps concrete sets to abstract values.
- **Concretization (γ):** $A \to C$. Maps abstract values to concrete sets.
- **Galois Connection:** The pair (α, γ) such that:
  - $c \subseteq \gamma(\alpha(c))$ (Extensive / Safe)
  - $\alpha(\gamma(a)) \sqsubseteq a$ (Reductive / Precise)

2. The Soundness Proof (The Square)

Diagram:

$$\begin{array}{ccc} C & \xrightarrow{cf} & C \\ {\scriptstyle\alpha}\downarrow & & \downarrow{\scriptstyle\alpha} \\ A & \xrightarrow{af} & A \end{array}$$

Equation: $\alpha(cf(c)) \sqsubseteq af(\alpha(c))$
"Abstract simulation over-approximates concrete execution."

3. Precision

- **Optimality:** The best possible af. Defined as $af = \alpha \circ cf \circ \gamma$.
- **Completeness:** No loss of precision. Defined as $\alpha \circ cf = af \circ \alpha$.
- **Example of Incompleteness:** $(+)+(-) = \top$. We lost the fact that $1+(-1) = 0$.

4. "Emergency" Q&A Answers

Q: Why do we need Fixed Points? A: We are analyzing loops. The state depends on itself. By Tarski's Theorem, because our lattice is complete and functions are monotone, a least fixed point exists.

Q: What is Widening? A: If the lattice has infinite height (like Intervals $[a, b]$), standard iteration might not terminate. Widening (∇) forces convergence by jumping up the lattice (e.g., setting a bound to ∞).

Q: What is Trace Semantics? A: Instead of just collecting states at nodes, we collect entire execution histories (traces). It's a more detailed form of collecting semantics, useful for historical properties.