Abstract Interpretation and Applications in Security and ML — Caterina Urban

Lecture 2 - *July 3, 2025*

1. Motivation: Why Prove Termination?

Termination is a liveness property: it ensures a program eventually completes its execution. It cannot be verified by finite testing—non-termination requires exploring infinite behavior.

Real-world examples of failure due to non-termination:

- **Zune Bug (2008)**: A leap-year date parsing bug caused infinite loops in every Zune device on December 31, 2008.
- **Apache HTTP Server (pre-2.3.3)**: Vulnerable to denial-of-service via infinite loop behavior.
- **Azure Storage Outage (2014)**: Transient errors triggered retry loops that never exited.

These examples motivate static analysis of termination — i.e., verifying at compile-time that programs always terminate under all conditions.

2. Liveness Properties: Trace-Based Semantics

Liveness properties can be formalized using trace semantics. A trace $t \in \Sigma^\infty$ is a (possibly infinite) sequence of program states.

Types of liveness properties:

- **Guarantee**: "Something good eventually happens." (e.g., termination)

- **Recurrence**: "Something good happens infinitely often." (e.g., starvation freedom)

Key Concept: Liveness cannot be falsified with finite traces — counterexamples must be infinite.

3. Termination Semantics: Potential vs Definite

We distinguish:

- **Potential termination**: At least one execution path terminates.

$$\mathcal{M} \cap \Sigma^* \neq \emptyset$$

- **Definite termination**: All execution paths terminate.

$$\mathcal{M} \subseteq \Sigma^*$$

In non-deterministic programs, these differ. In deterministic settings, they coincide.

Example:

```
while (*) {
  if (random() % 2 == 0) break;
}
```

This loop may terminate (potential), but not always (not definite).

4. Ranking Functions: Core Termination Proof Technique

To prove termination, we use a **ranking function** $f : \Sigma \to \mathcal{W}$ where $(\mathcal{W}, \leq)$ is a well-founded set (no infinite descending chains).

Definition: For a transition system (Σ, τ):

$$(\sigma, \sigma') \in \tau \Rightarrow f(\sigma') < f(\sigma)$$

Common well-founded sets:

- Natural numbers $\mathbb{N}$
- Ordinals (e.g., ω)

Concrete Example:

```
while (x > 0) { x = x - 1; }
```

Here, $f(x) = x$. Each iteration decreases x, proving termination.

Formalization in Control Points:

$$f : \Sigma \to \mathbb{N} \quad \text{can be viewed as } f : \mathcal{L} \to (\mathcal{E} \to \mathbb{N})$$

where $\mathcal{L}$ is the control point (line) and $\mathcal{E}$ is the environment (state variables).

5. The 3-Step Termination Analysis Recipe

1. **Concrete Semantics:** Capture all actual program behaviors.
2. **Abstract Semantics:** Use finite representations (abstract domains).
3. **Practical Tools:** Build analyzers to check termination properties.

Each step approximates or encodes the previous to ensure tractability.

6. Hierarchy of Termination Semantics

We define several layers of semantics:

- $\mathcal{M}$: Maximal trace semantics (all executions)
- $\mathcal{T}_M$: Terminating traces only
- $\mathcal{R}_M$: Termination via ranking abstraction

These are linked by abstraction:

$$\mathcal{M} \xrightarrow{\alpha^*} \mathcal{T}_M \xrightarrow{\alpha_M} \mathcal{R}_M$$

7. Definite Termination Trace Semantics

To eliminate ambiguity due to prefix overlap between finite and infinite traces, we define:

$$\alpha^*(T) = \{t \in T \cap \Sigma^* \mid \text{nhdb}(t, T \cap \Sigma^\omega) = \emptyset\}$$

where nhdb = "non-harmless debug prefixes."

Goal: Exclude finite traces with infinite extensions.

8. Fixpoint Transfer: From Concrete to Abstract

Using Tarski's fixpoint theorem, we transfer least fixpoints to abstract domains:

$$\alpha(\text{lfp } f) = \text{lfp } f^\# \quad \text{if } \alpha \text{ is a complete morphism}$$

Used to derive:

$$\mathcal{T}_M = \text{lfp}_\subseteq F^* \quad \text{where } F^* \text{ is the abstract transformer}$$

9. Ranking Function Abstraction

We approximate the set of state transitions with a relation $r \subseteq \Sigma \times \Sigma$, and define:

$$\alpha_V(r)(\sigma) = \begin{cases} 0 & \text{if no successor} \\ \sup\{\alpha_V(r)(\sigma') + 1\} & \text{otherwise} \end{cases}$$

Then:

$$\alpha_M(T) = \alpha_V(\rightarrow \alpha(T))$$

10. Termination Semantics as Fixpoint

We define:

$$\mathcal{R}_M = \text{lfp}_\preceq F_M$$

$$F_M(f)(\sigma) = \begin{cases} 0 & \sigma \in \mathcal{B} \\ \sup\{f(\sigma') + 1 \mid (\sigma, \sigma') \in \tau\} & \text{otherwise} \end{cases}$$

If $I \subseteq \text{dom}(\mathcal{R}_M)$, the program terminates from initial states I.

11. Denotational Semantics of Termination

Statements have corresponding transformers:

$$\mathcal{R}_M[[x := e]](f)(\rho) = \sup\{f(\rho[x \mapsto v]) + 1\}$$
$$\mathcal{R}_M[[if\ e\ then\ s]](f) = \text{case-based}$$
$$\mathcal{R}_M[[while\ e\ do\ s]](f) = \text{lfp of nested transformer}$$

This supports a compositional analysis.

12. Piecewise-Defined Ranking Abstractions

Ranking functions can be defined piecewise across different constraints:

$$\mathcal{R}_M^\# : \mathcal{L} \to \mathcal{A} \quad \text{with } \mathcal{A} = \text{piecewise function domain}$$

Auxiliary Abstract Domains:

- **Linear Constraints:** Intervals, polyhedra over variables.
- **Affine Functions:** $f(x) = m_1 x_1 + \cdots + m_k x_k + q$

Example:

```
while (x >= 0) {
  x = x - 2 * x + 10;
}
```

Different ranking functions apply depending on the value of x.

13. Summary

Termination analysis using abstract interpretation relies on:

- Rich semantics for execution traces
- Sound abstraction of behavior

- Ranking functions over well-founded domains
- Piecewise abstraction and numeric reasoning

Such analyses are powerful tools to ensure program reliability and safety.