OREGON PROGRAMMING LANGUAGES SUMMER SCHOOL

Static Analysis of Liveness Properties — Caterina Urban

Lecture 3 - *July 4, 2025*

1. Motivation: Static Termination Failures

Static analysis is critical to verify termination, a fundamental liveness property.

Zune Bug (Dec 31, 2008): A leap year bug caused all devices to freeze in an infinite loop.

Other real-world failures:

- *Apache HTTP Server* (pre-2.3.3): DoS via unhandled loop.
- *Azure Storage* (2014): Infinite retries led to downtime.

2. Liveness Properties and Termination

Liveness properties assert that "something good eventually happens".

- **Guarantee properties**: Something happens at least once.
- **Recurrence properties**: Something happens infinitely often.

Termination is a guarantee property. Defined over trace sets:

$$\text{Termination: } T = \Sigma^*, \quad T \subseteq \Sigma^\infty$$

3. Potential vs. Definite Termination

For a program $\mathcal{M}$ and the set of all finite (terminating) traces Σ^*

- **Potential termination**: Some executions terminate if $\mathcal{M} \cap \Sigma^* \neq \emptyset$
- **Definite termination**: All executions terminate if $\mathcal{M} \subseteq \Sigma^*$

In deterministic programs, the two coincide.

4. Ranking Functions for Proving Termination

To define termination, we need to define the concept of a set of a ranking function such that we can put program states Σ into a well ordered set who's value strictly decreases through transtitions between states. A ranking function $f : \Sigma \to \mathcal{W}$ maps program states to a well-ordered set $(\mathcal{W}, \leq)$, and satisfies:

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

Example:

```
1: x := [-infinity, +infinity]
2: while (1 - x < 0) do
3:     x := x - 1
4: done
```

For this line of code, we can see that it will eventually terminate as x decreases. We can create a ranking function that maps various points of the program to a value. We define the ranking function by partitioning it with respect to the program control points:

$$f : \Sigma \to \mathbb{O} \quad \text{is defined as} \quad f : \mathcal{L} \to (\mathcal{E} \to \mathbb{O})$$

That is, for each control point $\ell \in \mathcal{L}$, we define a function $f(\ell)$ mapping environments ρ to ordinals.

Ranking function (piecewise):

$$f(\rho) = \begin{cases} 2\rho(x) - 1 & \text{if } 1 - \rho(x) < 0 \\ 2 & \text{otherwise} \end{cases}$$

$$\begin{aligned}
f(4) &= \lambda\rho.\ 0 \\
f(2) &= \lambda\rho.\ \begin{cases} 1 & \text{if } 1-\rho(x) \not< 0 \\ 2\rho(x)-1 & \text{if } 1-\rho(x) < 0 \end{cases} \\
f(3) &= \lambda\rho.\ \begin{cases} 2 & \text{if } 2-\rho(x) \not< 0 \\ 2\rho(x)-2 & \text{if } 2-\rho(x) < 0 \end{cases} \\
f(1) &= \lambda\rho.\ \omega
\end{aligned}$$

5. 3-Step Static Termination Analysis

1. **Concrete semantics**: Full trace semantics of execution.
2. **Abstract semantics**: Approximate behavior with abstract domains.
3. **Practical tools**: Implement analysis (e.g., AProVE, Termite).

6. Hierarchy of Semantics

$\mathcal{M}$	Maximal trace semantics
$\mathcal{T}_M$	Termination trace semantics
$\mathcal{R}_M$	Termination semantics (ranking abstraction)

Abstractions relate them:

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

7. Definite Termination Trace Semantics

Defined using the abstraction α^*:

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

Where nhdb (non-harmless debug, or prefix-overlapping) traces. This function removes finite traces that share a prefix with infinite traces.

8. Tarskian Fixpoint Transfer

To reason about abstract fixpoints:

$$\alpha(\text{lfp } f) = \text{lfp } f^{\#} \quad \text{under suitable conditions}$$

Used to connect:

$$\text{Concrete } \mathcal{M} = \text{lfp } F \Rightarrow \mathcal{T}_M = \text{lfp } F^*$$

9. Ranking Function Abstraction

Define α_M:

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

Backward step counting:

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

10. Least Fixpoint for Termination Semantics

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

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

Theorem: $I \subseteq \text{dom}(\mathcal{R}_M) \Rightarrow$ Program terminates from I.

11. Denotational Semantics

Transformers for statements:

$$\mathcal{R}_M[[s_1; s_2]]f = \mathcal{R}_M[[s_1]](\mathcal{R}_M[[s_2]]f)$$

$$\mathcal{R}_M[[x \leftarrow e]]f(\rho) = \sup\{f(\rho[x \mapsto v]) + 1\}$$

Conditionals and loops use case distinctions and fixpoint iteration.

12. Piecewise-Defined Ranking Abstractions

Abstract semantics $\mathcal{R}_M^\#$:

$$\gamma(\mathcal{R}_M^\#)(x) \supseteq \mathcal{R}_M(x)$$

Use auxiliary domains:

- **Linear Constraints Domain**: Canonical inequalities
- **Natural-Valued Affine Functions**: $f(x) = m_1 x_1 + \cdots + m_k x_k + q$

Example: Domain splits by guards like $x \geq 0$, $x - 3 \geq 0$, etc.

Conclusion

We saw how termination can be analyzed using:

- Concrete and abstract semantics
- Ranking functions and fixpoint abstractions
- Numerical and logical abstractions for precision