Abstract Interpretation-Based Static Analysis — Caterina Urban

Lecture 4 - *July 4, 2025*

Termination Static Analysis

The goal is to predict a valid ranking function for a given program. A ranking function maps program states to a well-founded domain (e.g., natural numbers or ordinals) such that its value strictly decreases with each step of program execution, and is bounded below. If such a function can be found, the program is guaranteed to terminate.

Piecewise-Defined Ranking Functions

These functions are defined over different regions of the program's state space, with each region having its own ranking function. This allows for more precise analysis of programs with complex control flow.

- **Definite Termination Semantics**: A concrete ranking function $\mathbb{R}_M$ maps states to $\mathbb{R}_M(x)$, where x is a program state.
- **Abstract Definite Termination Semantics**: An abstract ranking function $\gamma(\mathbb{R}^\flat_M)$ approximates $\mathbb{R}_M$. The relationship is defined by:

$$dom(\gamma(\mathbb{R}^\flat_M)) \subseteq dom(\mathbb{R}_M)$$

$$\forall x \in dom(\gamma(\mathbb{R}^\flat_M)) : \mathbb{R}_M(x) \leq \gamma(\mathbb{R}^\flat_M)(x)$$

  This means the abstract domain covers a subset of the concrete domain, and for any state in the abstract domain, the concrete value is less than or equal to the abstract value (sound over-approximation).
- **Piecewise-Defined Function Domain**: Parameterized by an underlying numerical abstract domain $\langle \mathbb{D}, \sqsubseteq_D \rangle$ (e.g., intervals, polyhedra). The domain $\mathfrak{F}$ is defined as:

$$\mathfrak{F} \cong \mathfrak{def}\{\bot_F\} \cup (\mathbb{Z}^{|\mathbb{M}|} \to \mathbb{N}) \cup \{\top_F\}$$

We consider affine functions of the form $f(X_1, ..., X_k) = \sum_{i=1}^{k} m_i \cdot X_i + q$.

- **Approximation Order** $\leq_F [D]$:
  - Between defined leaf nodes:

$$f_1 \leq_F [D] f_2 \stackrel{\triangle def}{=} \forall \rho \in \gamma_D(D) : f_1(..., \rho(X_i), ...) \leq f_2(..., \rho(X_i), ...)$$

  - Otherwise (when one or both leaf nodes are undefined): $\bot_F \leq_F f \leq_F \top_F$.
- **Computational Order** $E_F[D]$: Similar to approximation order, but used for computational purposes within the abstract domain.
- **Piecewise-Defined Functions Domain** $\mathcal{E}$:

$$\mathcal{E} \triangleq \begin{cases} \bot_F & \text{if totally undefined} \\ \top_F & \text{if totally defined} \\ \{c : t_1; t_2\} & \text{if a decision node with condition } c \text{ and subtrees } t_1, t_2 \\ f & \text{if a leaf node with function } f \end{cases}$$

  The concretization function $\gamma_A : \mathcal{E} \to (\top \mathcal{L} \mathcal{E})$ is defined recursively.
- **Abstract Domain Operators**:
  - **Binary Operators**: Rely on a *tree unification* algorithm to find a common refinement for decision trees. Examples include approximation order, computational order, join, meet, and widening.
  - **Unary Operators**: Rely on a *tree pruning* algorithm. Examples include assignment and test.
- **Widening**: A key operator in abstract interpretation to ensure termination of the analysis for infinite-height lattices. It extrapolates the values of the ranking function. The prediction can temporarily be wrong, i.e., it can under-approximate the value or over-approximate the domain.
  - **Domain Widening**: Limits the size of decision trees.
  - **Value Widening**: Widen each (defined) leaf node with respect to adjacent (defined) leaf nodes using an extrapolation operator.

Ordinal-Valued Ranking Functions

For programs that do not terminate with a finite number of steps but still terminate, natural-valued ranking functions are insufficient. Ordinals provide a well-founded domain for such cases.

Need for Ordinals

Consider a program like 'while (x ¿ 0) do x := x - 1 done'. A simple ranking function like x works. However, for more complex nested loops or programs with non-deterministic choices, the number of steps to termination might not be bounded by a natural number. Ordinals extend the natural numbers to include transfinite values like $\omega, \omega+1, \omega\cdot 2, \omega^2$, etc., which are useful for proving termination of programs that exhibit complex termination behavior.

Ordinals

- **Finite Ordinals**: $0, 1, 2, \ldots, n, \ldots$
- **Transfinite Ordinals**: $\omega, \omega+1, \ldots, \omega\cdot 2, \ldots, \omega^2, \ldots$
- **Successor Ordinals**: $\text{succ}(\lambda) \triangleq \lambda \cup \{\lambda\}$
- **Limit Ordinals**: Ordinals that are not successor ordinals (e.g., ω).

Ordinal Arithmetic

- **Addition**:
  - $\lambda + 0 = \lambda$ (zero case)
  - $\lambda + \text{succ}(\beta) = \text{succ}(\lambda + \beta)$ (successor case)
  - $\lambda + \beta = \bigcup_{\gamma<\beta}(\lambda + \gamma)$ (limit case)

  Properties: associative, but **not commutative** (e.g., $1 + \omega = \omega \neq \omega + 1$).
- **Multiplication**:
  - $\lambda \cdot 0 = 0$ (zero case)
  - $\lambda \cdot \text{succ}(\beta) = (\lambda \cdot \beta) + \lambda$ (successor case)
  - $\lambda \cdot \beta = \bigcup_{\gamma<\beta}(\lambda \cdot \gamma)$ (limit case)

  Properties: associative, left distributive, but **not commutative** (e.g., $2 \cdot \omega = \omega \neq \omega \cdot 2$), and **not right distributive** (e.g., $(\omega + 1) \cdot \omega = \omega \cdot \omega \neq \omega \cdot \omega + \omega$).
- **Cantor Normal Form**: Any ordinal $\alpha > 0$ can be uniquely written as:

$$\alpha = \omega^{\beta_1} \cdot n_1 + \omega^{\beta_2} \cdot n_2 + \cdots + \omega^{\beta_k} \cdot n_k$$

  where $\beta_1 > \beta_2 > \cdots > \beta_k \geq 0$ are ordinals and $n_1, \ldots, n_k$ are positive integers.

- **Piecewise-Defined Function Domain for Ordinals**: The set of functions $\mathcal{P}$ is extended to include sums of products of ordinals and functions:

$$\mathcal{P} \triangleq \left\{ \sum_i \omega^i \cdot f_i \mid f_i \in \mathcal{P}_{\text{natural-valued}} \right\} \cup \{\mathcal{C}_W, \longrightarrow_W\}$$

  where $\mathcal{C}_W$ and $\longrightarrow_W$ represent the bottom and top elements for ordinal-valued functions.

- **Approximation Order $\leq_F [D]$ for Ordinals**:
  - Between defined leaf nodes (ordinal-valued functions):

$$\sum_i \omega^i \cdot f_{i1} \leq_W [D] \sum_i \omega^i \cdot f_{i2} \triangleq^{def} \forall \alpha \in \gamma_D(D) : \sum_i \omega^i \cdot f_{i1}(..., \alpha(X_i), ...) \leq \sum_i \omega^i \cdot f_{i2}(..., \alpha(X_i), ...)$$

  - Otherwise: $\bot_W \leq_W f \leq_W \top_W$.

- **Computational Order $E_F[D]$ for Ordinals**: Similar to approximation order.

- **Concretization Function for Ordinals**: The concretization function γ_A for piecewise-defined ordinal-valued functions is defined similarly, mapping abstract elements to concrete ordinal-valued functions.

- **Abstract Domain Operators for Ordinals**: The operators (order, join, meet, widening, assignment, test) are extended to handle ordinal-valued functions, often by applying the operations coefficient-wise in the Cantor Normal Form.
  - **Approximation Join for Ordinals**: Performs join in ascending powers of ω.
  - **Computational Join for Ordinals**: Performs join in ascending powers of ω.
  - **Value Widening for Ordinals**: Extrapolates in ascending powers of ω.
  - **Assignments for Ordinals**: Applied on defined leaf nodes in ascending powers of ω.

Liveness Properties and CTL

Liveness Properties

Liveness properties assert that "something good eventually happens".

- **Guarantee Properties**: "something good eventually happens at least once" (e.g., Program Termination).
- **Recurrence Properties**: "something good eventually happens infinitely often" (e.g., Starvation Freedom).

Computation Tree Logic (CTL)

CTL is a branching temporal logic used to express properties about computation trees. Formulas ϕ are built from atomic propositions (a), logical connectives ($\neg, \wedge, \vee$), and temporal operators with path quantifiers:

- **Path Quantifiers**:
  - $\mathbb{A}$: for all paths
  - $\mathbb{E}$: for some path
- **Temporal Operators**:
  - $\mathbb{X}$: next state
  - $\mathbb{U}$: until
  - $\mathbb{F}$: eventually (future) - $\mathbb{F}\phi \equiv \text{true}\mathbb{U}\phi$
  - $\mathbb{G}$: globally (always) - $\mathbb{G}\phi \equiv \neg\mathbb{F}\neg\phi$

Common combinations: $\mathbb{AF}\phi, \mathbb{EF}\phi, \mathbb{AG}\phi, \mathbb{EG}\phi, \mathbb{AU}\phi, \mathbb{EU}\phi$.

Static Guarantee Analysis

This analysis aims to prove guarantee properties.

- **Program Guarantee Semantics**: Defines the concrete semantics for guarantee properties. For a property ϕ, the semantics $\subseteq^G_\phi$ is defined as a greatest fixed point (gfp) or least fixed point (lfp) depending on the property.
- **Abstract Program Guarantee Semantics**: An abstract interpretation framework using piecewise-defined ranking functions to approximate the concrete guarantee semantics.
- **Soundness**: If the abstract analysis proves a property, then the concrete program satisfies it.

Static Recurrence Analysis

This analysis aims to prove recurrence properties.

- **Program Recurrence Semantics**: Defines the concrete semantics for recurrence properties, often building upon guarantee semantics.

- **Abstract Program Recurrence Semantics**: An abstract interpretation framework for recurrence properties. This often employs a **dual widening** operator.
- **Dual Widening**: For a poset $\langle D, \sqsubseteq \rangle$, a dual widening $\overline{\nabla} : D \times D \to D$ satisfies:
  (i) For all $x, y \in D$, we have $x \sqsubseteq x \overline{\nabla} y$ and $y \sqsubseteq x \overline{\nabla} y$.
  (ii) For all decreasing chains $x_0 \sqsupseteq x_1 \sqsupseteq \cdots \sqsupseteq x_n \sqsupseteq \ldots$, the chain $y_0 \triangleq x_0$, $y_{n+1} \triangleq y_n \overline{\nabla} x_{n+1}$ is ultimately stationary.

  Dual widening is used to stabilize decreasing chains in the abstract domain, which is crucial for analyzing recurrence properties.

Termination Resilience

Termination Resilience addresses the question of whether a program terminates under certain non-deterministic choices (angelic non-determinism) or diverges under others (demonic non-determinism).

Robust Non-Termination

A program robustly non-terminates if for all inputs, there exists a non-deterministic choice that leads to divergence. This corresponds to demonic non-determinism.

Termination Resilience

A program exhibits termination resilience if for all inputs, there exists a non-deterministic choice that leads to termination. This corresponds to angelic non-determinism. The analysis counts execution steps backwards.

Static Termination Resilience Analysis

This involves a 3-step recipe similar to other static analyses:

1. Concrete Semantics (mathematical models of program behavior).
2. Abstract Semantics, Abstract Domains (algorithmic approaches to decide program properties).
3. Practical Tools (targeting specific programs).

The abstract domain for termination resilience also uses piecewise-defined ranking functions, and the operators (like join) are adapted for this context, sometimes referred to as "resilience join".

Experimental Evaluation

The techniques discussed have been experimentally evaluated on benchmarks like SV-COMP 2024, Raad et al. @ OOPSLA 2024, and Shi et al. @ FSE 2022.

- **Multivariate Constraints**: Show good performance for termination and termination resilience.
- **Univariate Constraints**: Can also be effective, and in some cases, univariate constraints are more precise than multivariate, or vice-versa, depending on the specific property.