Verification of Real-Time Systems
Foundations of Abstract Interpretation

Jan Reineke

Advanced Lecture, Summer 2015
Applications of Abstract Interpretation

- Compilers:
  - Is it safe to perform a certain program transformation?

- Verification:
  - Can there be runtime errors? E.g. array out-of-bounds access, division-by-zero, dereferencing of null pointers
  - Security analysis: does information about secrets leak?
  - Timing analysis: loop bounds, cache analysis,…
Recap: Value Analysis

Determines **invariants on values of registers** at different program points. Invariants are often in the form of **enclosing intervals** of all possible values.

Where is this information used?

- **Microarchitectural Analysis**
  - Pipeline Analysis
  - Cache Analysis
- **Control-Flow Analysis**
  - Detect infeasible paths
  - Derive loop bounds
Value Analysis
Intuition of Interval Analysis

\[ R_1 = [-\infty, +\infty] \]
\[ R_2 = [-\infty, +\infty] \]

Can be formalized as Abstract Interpretation.
\[ \Rightarrow \text{Yields soundness and termination guarantees.} \]
Abstract Interpretation

- Semantics-based approach to program analysis
- Framework to develop provably correct and terminating analyses

Ingredients:
- **Concrete semantics**: Formalizes meaning of a program
- **Abstract semantics**
- Both semantics defined as fixpoints of monotone functions over some domain
- Relation between the two semantics establishing correctness
Representing Programs by Control-Flow Graphs

$CFG \ G = (V, E, start, \text{labeling})$

$\text{start} \in V$

$E \subseteq V \times V$

$\text{labeling}: E \rightarrow \text{Statement}$

$\text{labeling}(3, 4) = a = M[x]$
Four Kinds of Statements

1. Assignment: \( R = e \)
2. Load: \( R = M[e] \)
3. Store: \( M[e_1] = e_2 \)
4. Test: Pos(e) or Neg(e)

\( e = x + y + z \)
States consists of variables and memory:

\[ s = (\rho, \mu) \in \text{States} \]

- \( \rho : \text{Vars} \rightarrow \text{int} \) \hspace{1cm} \text{Values of Variables}
- \( \mu : \mathbb{N} \rightarrow \text{int} \) \hspace{1cm} \text{Contents of Memory}

Execution of a statement transforms states:

\[
\begin{align*}
\llbracket \text{statement} \rrbracket & \subseteq \text{States} \times \text{States} \\
\llbracket R = e \rrbracket & := \{((\rho, \mu), (\rho[R \mapsto \llbracket e \rrbracket_\rho], \mu)) \mid (\rho, \mu) \in \text{States}\} \\
\llbracket R = M[e] \rrbracket & := \{((\rho, \mu), (\rho[R \mapsto \mu(\llbracket e \rrbracket_\rho)], \mu)) \mid (\rho, \mu) \in \text{States}\} \\
\llbracket M[e_1] = e_2 \rrbracket & := \{((\rho, \mu), (\rho, \mu[[e_1]_\rho \mapsto \llbracket e_2 \rrbracket_\rho])) \mid (\rho, \mu) \in \text{States}\} \\
\llbracket \text{Pos}(e) \rrbracket & := \{((\rho, \mu), (\rho, \mu)) \mid (\rho, \mu) \in \text{States} \land \llbracket e \rrbracket_\rho \neq 0\} \\
\llbracket \text{Neg}(e) \rrbracket & := \{((\rho, \mu), (\rho, \mu)) \mid (\rho, \mu) \in \text{States} \land \llbracket e \rrbracket_\rho = 0\}
\end{align*}
\]
Meaning of Expressions

Evaluation of expressions is as expected:

\[
\begin{align*}
\llbracket a \rrbracket \rho & := \rho(a) \quad \text{if} \ a \in Vars \\
\llbracket e_1 \otimes e_2 \rrbracket \rho & := \llbracket e_1 \rrbracket \rho \otimes \llbracket e_2 \rrbracket \rho \\
\llbracket a < b \rrbracket \rho & := \begin{cases} 
1 & : \llbracket a \rrbracket \rho < \llbracket b \rrbracket \rho \\
0 & : \text{otherwise}
\end{cases}
\end{align*}
\]
Concrete Semantics

Different **semantics** are required for different properties:

- “Is there an execution in which the value of x alternates between 3 and 5?” ➔ **Trace Semantics**
- “Is the final value of x always the same as the initial value of x?” ➔ “Input/Output” Semantics
- “May x ever assume the value 45 at program point 7?” ➔ **Reachability Semantics**
Concrete Semantics

- **Trace Semantics**: Captures set of traces of states that the program may execute.
- **Input/Output Semantics**: Captures the pairs of initial and final states of execution traces.
  - Abstraction of Trace Semantics
- **Reachability Semantics**: Captures the set of reachable states at each program point
  - Abstraction of Trace Semantics
Reachability Semantics

Captures the set of reachable states at each program point. Formally: $\text{Reach} : V \rightarrow \mathcal{P}(\text{States})$

Example:
Reachability Semantics

Can be captured as the least solution of:

\[
Reach(\text{start}) = \bigcup_{v \in V, (v, v') \in E} \left[ \text{labeling}(v, v') \right](Reach(v))
\]

\[
\forall v' \in V \setminus \{\text{start}\} : Reach(v') = \bigcup_{v \in V, (v, v') \in E} \left[ \text{labeling}(v, v') \right](Reach(v))
\]

\[
Reach(1) = [\text{labeling}(\text{start}, 1)](Reach(\text{start})) \cup [\text{labeling}(2, 1)](Reach(2))
\]

\[
Reach(2) = [\text{labeling}(1, 2)](Reach(1))
\]

\[
Reach(3) = [\text{labeling}(1, 3)](Reach(1))
\]

\[
Reach(1) = [x = 0](Reach(\text{start})) \cup [x = x + 1](Reach(2))
\]

\[
Reach(2) = [\text{Pos}(x < 100)](Reach(1))
\]

\[
Reach(3) = [\text{Neg}(x < 100)](Reach(1))
\]

\[
Reach(1) = \{0\} \cup \{v + 1 \mid v \in Reach(2)\}
\]

\[
Reach(2) = Reach(1) \cap \{\ldots, 98, 99\}
\]

\[
Reach(3) = Reach(1) \cap \{100, 101, \ldots \}
\]
Questions

- Why the **least solution**?
- Is there more than one solution?
- Is there a **unique** least solution?
- Can we systematically compute it?
Answers

- Is there more than one solution? Yes!
- Is there a unique least solution? Yes!
- Can we systematically compute it? Yes and No.
Why? Knaster-Tarski Fixpoint Theorem!

**Theorem 1** (Knaster-Tarski, 1955).
Assume $(D, \leq)$ is a complete lattice. Then every monotonic function $f : D \to D$ has a least fixed point $d_0 \in D$.

Raises more questions:
- What is a **complete lattice**?
- What is a **monotonic function**?
- What is a **fixed point**?
Monotone Functions

\[ f : A \to \beta \]

Let \((D, \leq)\) be partially-ordered set.
For example: \(D = \mathbb{N}\) and \(\leq\) the order on natural numbers.

Function \(f : D \to D\) is monotone (order-preserving) iff
for all \(d_1, d_2 \in D: d_1 \leq d_2 \Rightarrow f(d_1) \leq f(d_2)\).

Examples:
\[
\begin{align*}
f(x) &= x & \checkmark & \quad d_1 \leq d_2 \Rightarrow d_1 \leq d_2 \\
g(x) &= -x & \times \\
h(x) &= x - 1 & \checkmark \\
\end{align*}
\]

\((\mathcal{P}(\mathbb{N}), \subseteq)\)
\[
\begin{align*}
F(X) &= \{f(x) \mid x \in X\} \\
G(X) &= \{y \mid x \in X \land (x, y) \in R\}
\end{align*}
\]

Need to know what the order is.
Partial Orders

A binary relation $\leq: D \times D$ is a partial order, iff for all $a, b, c \in D$, we have that:

- $a \leq a$ (reflexivity),
- if $a \leq b$ and $b \leq a$ then $a = b$ (antisymmetry),
- if $a \leq b$ and $b \leq c$ then $a \leq c$ (transitivity).

A set with a partial order is called a partially-ordered set.
Partial Orders: Examples I

The natural numbers ordered by the standard less-than-or-equal relation: \((\mathbb{N}, \leq)\). \((\mathbb{N}, \geq)\)

The set of subsets of a given set (its powerset) ordered by the subset relation: \((\mathcal{P}(A), \subseteq)\).

The set of subsets of a given set (its powerset) ordered by the subset relation: \((\mathcal{P}(A), \supseteq)\).

The natural numbers ordered by \textit{divisibility}: \((\mathbb{N}, |)\).
Partial Orders: Examples II

The vertex set $V$ of a directed acyclic graph $G = (V, E)$ ordered by reachability (reflexive, transitive closure of edge relation).

The vertex set $V$ of an arbitrary graph $G = (V, E)$ ordered by reachability.

For a set $X$ and a partially-ordered set $P$, the function space $F : X \rightarrow P$, where $f \leq g$ if and only if $f(x) \leq g(x)$ for all $x$ in $X$.

What about $\overline{\text{Reach}} : V \rightarrow \mathcal{P}(\text{States})$?

$f \leq g : \iff \forall v \in V. f(v) \leq g(v)$
A partially-ordered set \((L, \leq)\) is a complete lattice if every subset \(A\) of \(L\) has both a least upper bound (denoted \(\bigvee A\)) and a greatest lower bound (denoted \(\bigwedge A\)).

**What is an upper bound of a set \(A\)?**

An element \(x\) is an upper bound of a set \(A\) if \(x\) if for every element \(a\) of \(A\), we have \(a \leq x\).

**What is the least upper bound (also: join, supremum) of a set \(A\)?**

\(x\) is the least upper bound of \(A\), denoted \(\bigvee A\), if

1. \(x\) is an upper bound of \(A\),
2. for every upper bound \(y\) of \(A\), we have \(x \leq y\).