Static Program Analysis using Abstract Interpretation
Introduction
Static Program Analysis

Static program analysis consists of automatically discovering properties of a program that hold for all possible execution paths of the program.

Static program analysis is not

- **Testing**: manually checking a property for some execution paths
- **Model checking**: automatically checking a property for all execution paths
Program Analysis for what?

• Optimizing compilers
• Semantic preprocessing:
  – Model checking
  – Automated test generation
• Program verification
Program Verification

- Check that every operation of a program will never cause an error (division by zero, buffer overrun, deadlock, etc.)

- **Example:**

  ```c
  int a[1000];
  for (i = 0; i < 1000; i++) {
    safe operation  a[i] = ... ;  // 0 <= i <= 999
  }
  buffer overrun   a[i] = ... ;  // i = 1000;
  ```
Incompleteness of Program Analysis

• Discovering a sufficient set of properties for checking every operation of a program is an undecidable problem!
• Every non trivial behavioral property has (at least) NP complexity
• **False positives:** operations that are safe in reality but which cannot be decided safe or unsafe from the properties inferred by static analysis.
Precision versus Efficiency

**Precision:** number of program operations that can be decided safe or unsafe by an analyzer.

- Precision and computational complexity are strongly related
- Tradeoff precision/efficiency: limit in the average precision and scalability of a given analyzer
- Greater precision and scalability is achieved through specialization
Soundness

• What guarantees the soundness of the analyzer results?
• In dataflow analysis and type inference the soundness proof of the resolution algorithm is independent from the analysis specification
• An independent soundness proof precludes the use of test-and-try techniques
• Need for analyzers correct by construction
Abstract Interpretation

• A general methodology for designing static program analyzers that are:
  – Correct by construction
  – Generic
  – Easy to fine-tune

• Scalability is difficult to achieve but the payoff is worth the effort!
Approximation

The core idea of Abstract Interpretation is the formalization of the notion of approximation

• An approximation of memory configurations is first defined
• Then the approximation of all atomic operations
• The approximation is automatically lifted to the whole program structure
Overview of Abstract Interpretation

• Start with a formal specification of the program semantics (the concrete semantics)
• Construct abstract semantic equations w.r.t. a parametric approximation scheme
• Use general fixpoints algorithms to solve the abstract semantic equations
• Try-and-test various instantiations of the approximation scheme in order to find the best fit
The Methodology of Abstract Interpretation
Methodology

Concrete Semantics

Collecting Semantics

Partitioning

Abstract Semantics

Iterative Resolution Algorithms

Tuners

Abstract Domain

Abstract Domain
Lattices and Fixpoints

• A lattice \((L, \sqsubseteq, \bot, \vee, \top, \wedge)\) is a partially ordered set \((L, \sqsubseteq)\) with:
  – Least upper bounds (\(\vee\)) and greatest lower bounds (\(\wedge\)) operators
  – A least element “bottom”: \(\bot\)
  – A greatest element “top”: \(\top\)

• \(L\) is complete if all least upper bounds exist

• A fixpoint \(X\) of \(F: L \rightarrow L\) satisfies \(F(X) = X\)

• We denote by \(\text{lfp} F\) the least fixpoint if it exists
Fixpoint Theorems

- Knaster-Tarski theorem: If $F: L \rightarrow L$ is monotone and $L$ is a complete lattice, the set of fixpoints of $F$ is also a complete lattice.

- Kleene theorem: If $F: L \rightarrow L$ is monotone, $L$ is a complete lattice and $F$ preserves all least upper bounds then $\text{lfp } F$ is the limit of the sequence:

$$
\begin{cases}
F_0 &= \bot \\
F_{n+1} &= F(F_n)
\end{cases}
$$
Concrete Semantics

Small-step operational semantics: \((\Sigma, \rightarrow)\)

\[ s = \langle \text{program point}, \text{env} \rangle \]

Example:

1: \( n = 0; \)
2: \( \text{while } n < 1000 \text{ do} \)
3: \( n = n + 1; \)
4: \( \text{end} \)
5: \( \text{exit} \)

\[ \langle 1, n \Rightarrow \Omega \rangle \rightarrow \langle 2, n \Rightarrow 0 \rangle \rightarrow \langle 3, n \Rightarrow 0 \rangle \rightarrow \langle 4, n \Rightarrow 1 \rangle \rightarrow \langle 2, n \Rightarrow 1 \rangle \rightarrow \ldots \rightarrow \langle 5, n \Rightarrow 1000 \rangle \]

Undefined value
Control Flow Graph

1: \[ n = 0; \]

2: \[ \text{while } n < 1000 \text{ do} \]

3: \[ n = n + 1; \]

4: \[ \text{end} \]

5: \[ \text{exit} \]
Transition Relation

Control flow graph: \( i \xrightarrow{\text{op}} j \)

Operational semantics: \( \langle i, \varepsilon \rangle \rightarrow \langle j, [\text{op}] \varepsilon \rangle \)

Semantics of op
Methodology

Concrete Semantics

Collecting Semantics

Partitioning

Abstract Semantics

Iterative Resolution Algorithms

Abstract Domain

Abstract Domain

Tuners
Collecting Semantics

The collecting semantics is the set of observable behaviours in the operational semantics. It is the starting point of any analysis design.

- The set of all descendants of the initial state
- The set of all descendants of the initial state that can reach a final state
- The set of all finite traces from the initial state
- The set of all finite and infinite traces from the initial state
- etc.
Which Collecting Semantics?

- Buffer overrun, division by zero, arithmetic overflows: state properties
- Deadlocks, un-initialized variables: finite trace properties
- Loop termination: finite and infinite trace properties
State properties

The set of descendants of the initial state $s_0$:

$$S = \{ s \mid s_0 \to \ldots \to s \}$$

Theorem: $F : (\wp(\Sigma), \subseteq) \to (\wp(\Sigma), \subseteq)$

$$F(S) = \{ s_0 \} \cup \{ s' \mid \exists s \in S: s \to s' \}$$

$$S = \text{Ifp } F$$
Example

1:   \[ n = 0; \]
2:   \[ \text{while } n < 1000 \text{ do} \]
3:   \[ n = n + 1; \]
4:   \[ \text{end} \]
5:   \[ \text{exit} \]

\[ S = \{ \langle 1, n \Rightarrow \Omega \rangle, \langle 2, n \Rightarrow 0 \rangle, \langle 3, n \Rightarrow 0 \rangle, \langle 4, n \Rightarrow 1 \rangle, \langle 2, n \Rightarrow 1 \rangle, \ldots, \langle 5, n \Rightarrow 1000 \rangle \} \]
Computation

- \( F_0 = \emptyset \)
- \( F_1 = \{ \langle 1, n \Rightarrow \Omega \rangle \} \)
- \( F_2 = \{ \langle 1, n \Rightarrow \Omega \rangle, \langle 2, n \Rightarrow 0 \rangle \} \)
- \( F_3 = \{ \langle 1, n \Rightarrow \Omega \rangle, \langle 2, n \Rightarrow 0 \rangle, \langle 3, n \Rightarrow 0 \rangle \} \)
- \( F_4 = \{ \langle 1, n \Rightarrow \Omega \rangle, \langle 2, n \Rightarrow 0 \rangle, \langle 3, n \Rightarrow 0 \rangle, \langle 4, n \Rightarrow 1 \rangle \} \)
- ...

Methodology

Concrete Semantics

Collecting Semantics

Partitioning

Abstract Semantics

Abstract Domain

Iterative Resolution Algorithms

Abstract Domain

Tuners
Partitioning

We partition the set $S$ of states w.r.t. program points:

- $\Sigma = \Sigma_1 \oplus \Sigma_2 \oplus \ldots \oplus \Sigma_n$
- $\Sigma_i = \{ \langle k, \varepsilon \rangle \in \Sigma \mid k = i \}$
- $F(S_1, \ldots, S_n)_0 = \{ s_0 \}$
- $F(S_1, \ldots, S_n)_i = \{ s' \in S_i \mid \exists j \exists s \in S_j : s \rightarrow s' \}$

i.e.

$$F(S_1, \ldots, S_n)_i = \{ \langle \text{op}, \varepsilon \rangle \mid \text{CFG (P)} \}$$
Illustration
Semantic Equations

- **Notation**: $E_i = \text{set of environments at program point } i$
- **System of semantic equations**:

$$E_i = \bigcup \{ [\text{op}] E_j | \text{j} \xrightarrow{\text{op}} i \in \text{CFG (P)} \}$$

- **Solution of the system**: $S = \text{lfp } F$
Example

\begin{align*}
E_1 &= \{n \Rightarrow \Omega\} \\
E_2 &= \left[ n = 0 \right] E_1 \cup E_4 \\
E_3 &= E_2 \cap \left[ -\infty, 999 \right] \\
E_4 &= \left[ n = n + 1 \right] E_3 \\
E_5 &= E_2 \cap \left[ 1000, +\infty \right]
\end{align*}
Example

$E_1 = \{ n \Rightarrow \Omega \}$

$E_2 = \{ \text{while} n \not\geq 1000 \Rightarrow E_1 \}\{ n = 0; E_3 \}\{ n = n + 1; E_4 \}\{ \text{end} \}$

$E_3 = \{ \text{exit} \}[ -\infty, 999 ]$

$E_4 = \{ n = n + 1 \}[ n \not\geq 1000 ] E_3$

$E_5 = E_2 \cap [ 1000, +\infty[$
Methodology

Concrete Semantics

Collecting Semantics

Partitioning

Abstract Semantics

Abstract Domain

Abstract Domain

Iterative Resolution Algorithms

Tuners
Problem: Compute a sound approximation $S^#$ of $S$

Solution: Galois connections
Galois Connection

$L_1, L_2$ two lattices

$\gamma$  
$\alpha$

$\forall x \forall y : \alpha(x) \leq y \iff x \subseteq \gamma(y)$

$\forall x \forall y : x \subseteq \gamma \circ \alpha(x) \land \alpha \circ \gamma(y) \leq y$
Theorem:
\[ \text{lfp } F \subseteq \gamma \left( \text{lfp } \alpha \circ F \circ \gamma \right) \]
Abstracting the Collecting Semantics

• Find a Galois connection:

\[ (\wp(\Sigma), \subseteq) \leftrightarrow (\Sigma^#, \leq) \]

\[ \gamma \]

\[ \alpha \]

• Find a function: \( \alpha \circ F \circ \gamma \leq F^# \)
Abstract Algebra

- **Notation:** $E$ the set of all environments
- Galois connection:

\[
\begin{align*}
(\emptyset(E), \subseteq) & \quad \xleftarrow{\gamma} \quad (E^\#, \leq) \\
E & \quad \xrightarrow{\alpha} \quad E^\# 
\end{align*}
\]

- $\cup, \cap$ approximated by $\cup^\#, \cap^\#$
- Semantics $\llbracket \text{op} \rrbracket$ approximated by $\llbracket \text{op} \rrbracket^\#$

\[
\alpha \circ \llbracket \text{op} \rrbracket \circ \gamma \subseteq \llbracket \text{op} \rrbracket^\#
\]
Abstract Semantic Equations

1: \( n = 0; \)
2: \( \text{while } n < 1000 \text{ do} \)
3: \( n = n + 1; \)
4: \( \text{end}; \)
5: \( \text{exit}; \)

\[
\begin{align*}
E_1^# &= \alpha \left( \{ n \Rightarrow \Omega \} \right) \\
E_2^# &= \left[ n = 0 \right] \# E_1^# \cup \# E_4^# \\
E_3^# &= E_2^# \cap \# \alpha \left( [\pm \infty, 999] \right) \\
E_4^# &= \left[ n = n + 1 \right] \# E_3^# \\
E_5^# &= E_2^# \cap \# \alpha \left( [1000, +\infty[ \right)
\end{align*}
\]
Methodology

Concrete Semantics

Collecting Semantics

Partitioning

Abstract Semantics

Abstract Domain

Abstract Domain

Iterative Resolution Algorithms

Tuners
Abstract Domains

Various kinds of approximations:

- Signs (non relational)
  \[ x \mapsto +, \ y \mapsto -, \ldots \]

- Intervals (nonrelational):
  \[ x \mapsto [3, 9], \ y \mapsto [-23, 4], \ldots \]

- Polyhedra (relational):
  \[ x + y - 2z \leq 10, \ldots \]

- Difference-bound matrices (weakly relational):
  \[ y - x \leq 5, \ z - y \leq 10, \ldots \]
Example: intervals

1:   n = 0;
2:   while n < 1000 do
3:     n = n + 1;
4:   end
5:   exit

- Iteration 1:  $E_2^# = [0, 0]$
- Iteration 2:  $E_2^# = [0, 1]$
- Iteration 3:  $E_2^# = [0, 2]$
- Iteration 4:  $E_2^# = [0, 3]$
- ...
**Problem**

How to cope with lattices of infinite height?

**Solution:** automatic extrapolation operators
Methodology

Concrete Semantics

Collecting Semantics

Partitioning

Abstract Semantics

Abstract Domain

Abstract Domain

Tuners

Iterative Resolution Algorithms
Widening operator

Lattice \((L, \leq)\): \(\nabla : L \times L \to L\)

- Abstract union operator:
  \[ \forall x \forall y : x \leq x \nabla y \quad \& \quad y \leq x \nabla y \]

- Enforces convergence:
  \( (x_n)_{n \geq 0} \)

\[
\begin{cases}
  y_0 &= x_0 \\
  y_{n+1} &= y_n \nabla x_{n+1}
\end{cases}
\]

\((y_n)_{n \geq 0}\) is ultimately stationary
Widening of intervals

\[ [a, b] \triangledown [a', b'] \]

- If \( a \leq a' \) then \( a \) else \(-\infty\)
- If \( b' \leq b \) then \( b \) else \(+\infty\)

\(\rightarrow\) Open unstable bounds (jump over the fixpoint)
Widening and Fixpoint
Iteration with widening

1:  \( n = 0; \)
2:  while \( n < 1000 \) do
3:    \( n = n + 1; \)
4:  end
5:  exit

\[
(E_2^#)_{n+1} = (E_2^#)_n \lor (\llbracket n = 0 \rrbracket \# (E_1^#)_n \cup \# (E_4^#)_n)
\]

Iteration 1 (union): \( E_2^# = [0, 0] \)
Iteration 2 (union): \( E_2^# = [0, 1] \)
Iteration 3 (widening): \( E_2^# = [0, +\infty] \Rightarrow \text{stable} \)
Imprecision at loop exit

1: n = 0;
2: while n < 1000 do
3: n = n + 1;
4: end
5: exit; t[n] = 0; // t has 1500 elements

False positive!!!
Narrowing operator

Lattice \((L, \leq)\): \(\Delta : L \times L \to L\)

- Abstract intersection operator:
  \[\forall x \forall y : x \cap y \leq x \Delta y\]

- Enforces convergence: \((x_n)_{n \geq 0}\)

\[
\begin{cases}
y_0 = x_0 \\
y_{n+1} = y_n \Delta x_{n+1}
\end{cases}
\]

\((y_n)_{n \geq 0}\) is ultimately stationary
Narrowing of intervals

\[ [a, b] \triangle [a', b'] \]

- If \( a = -\infty \) then \( a' \) else \( a \)
- If \( b = +\infty \) then \( b' \) else \( b \)

➡️ Refines open bounds
Narrowing and Fixpoint

- Narrowing
- Fixpoint
- Widening
Iteration with narrowing

1: n = 0;
2: while n < 1000 do
3:   n = n + 1;
4: end
5: t[n] = 0;

\((E_2^\#)_{n+1} = (E_2^\#)_n \Delta \left( [n = 0] \# (E_1^\#)_n \mathbin{\cup} \# (E_4^\#)_n \right)\)

Beginning of iteration: \(E_2^\# = [0, +\infty[\)

Iteration 1: \(E_2^\# = [0, 1000] \Rightarrow \text{stable}\)

Consequence: \(E_5^\# = [1000, 1000]\)
Methodology

Concrete Semantics

Collecting Semantics

Partitioning

Abstract Semantics

Iterative Resolution Algorithms

Tuners

Abstract Domain

Abstract Domain
Tuning the abstract domains

1: n = 0;
2: k = 0;
3: while n < 1000 do
4:    n = n + 1;
5:    k = k + 1;
6:  end
7:  exit

- Intervals:
  \[ E_4^\# = \langle n \Rightarrow [0, 1000], k \Rightarrow [0, +\infty[ \rangle \]

- Convex polyhedra:
  \[ E_4^\# = \langle 0 \leq n \leq 1000, 0 \leq k \leq 1000, n - k = 0 \rangle \]
Annotated Bibliography
References

• The historic paper:

• Accessible introductions to the theory:

• Beyond Galois connections, a presentation of relaxed frameworks:

• A thorough description of a static analyzer with all the proofs (difficult to read):
• The abstract domain of intervals:

• The abstract domain of convex polyhedra:

• Weakly relational abstract domains:

• Classical data flow analysis: