Static Program Analysis using Abstract Interpretation

Arnaud Venet  
venet@email.arc.nasa.gov

Guillaume Brat  
brat@email.arc.nasa.gov

Kestrel Technology  
NASA Ames Research Center  
Moffett Field, CA 94035

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
- Program understanding
- 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!
- **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*

Specialization

- Tailoring the program analyzer algorithms for a specific class of programs (flight control commands, digital signal processing, etc.)
- Precision and scalability is guaranteed for this class of programs only
- Requires a lot of try-and-test to fine-tune the algorithms
- **Need for an open architecture**
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
- The approximation is generally a scheme that depends on some other parameter approximations

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 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

Abstract Domain

Abstract Domain

Tuners
Lattices and Fixpoints

- A lattice \((L, \sqsubseteq, \bot, \top, \sqcup, \sqcap)\) is a partially ordered set \((L, \sqsubseteq)\) with:
  - Least upper bounds \((\sqcup)\) and greatest lower bounds \((\sqcap)\) 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:

\[
\begin{align*}
F_0 &= \bot \\
F_{n+1} &= F(F_n)
\end{align*}
\]
Methodology

Concrete Semantics

Collecting Semantics

Partitioning

Iterative Resolution Algorithms

Tuners

Abstract Domain

Abstract Domain

Concrete Semantics

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

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

Example:

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

Transition Relation

Control flow graph: \( 1 \xrightarrow{\text{op}} 1 \)

Operational semantics: \( \langle 1, \epsilon \rangle \rightarrow \langle 1, [\text{op}] \epsilon \rangle \)

Semantics of op
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 $s_0$:

\[ S = \{ s \mid s_0 \rightarrow \ldots \rightarrow s \} \]

Theorem: \( F : (\varnothing(\Sigma), \subseteq) \rightarrow (\varnothing(\Sigma), \subseteq) \)

\[ F(S) = \{s_0\} \cup \{s' \mid \exists s \in S: s \rightarrow s'\} \]

\[ S = \text{lfp } F \]
Example

1: \ n = 0;
2: \ \textbf{while} \ n < 1000 \ \textbf{do}
3: \ \ n = n + 1;
4: \ \textbf{end}
5: \ \textbf{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 \} \\
- \ \ldots \]
Methodology

Partitioning

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

- $\Sigma = \Sigma_1 \oplus \Sigma_2 \oplus \ldots \oplus \Sigma_n$
- $\Sigma_i = \{ \langle k, \varepsilon \rangle \in \Sigma \mid k = i \}$
- $F(S_1, \ldots, S_n)_i = \{ s' \in S_i \mid \exists j \exists s \in S_j \colon s \rightarrow s' \}$
- $F(S_1, \ldots, S_n)_i = \{ \langle i, \left[ \text{op} \right] \varepsilon \rangle \mid \text{op} \overset{\text{CFG} (P)}{\rightarrow} \}$$
- $F(S_1, \ldots, S_n)_0 = \{ s_0 \}$
Illustration

Semantic Equations

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

$$E_i = \bigcup \{ [\text{op}] E_j \mid \text{op} \in \text{CFG} (P) \}$$

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

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

\[
E_1 = \{n \Rightarrow \Omega\}
\]
\[
E_2 = [n = 0] E_1 \cup E_4
\]
\[
E_3 = E_2 \cap ]-\infty, 999]\n\]
\[
E_4 = [n = n + 1] E_3
\]
\[
E_5 = E_2 \cap [1000, +\infty[\n\]
Other Kinds of Partitioning

- In the case of collecting semantics of traces:
  - Partitioning w.r.t. procedure calls: **context sensitivity**
  - Partitioning w.r.t. executions paths in a procedure: **path sensitivity**
  - Dynamic partitioning (Bourdoncle)

Methodology

```
Concrete Semantics

Collecting Semantics

Partitioning

Abstract Semantics

Iterative Resolution Algorithms
```

Tuners

Abstract Domain

Abstract Domain
Approximation

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

$S \subseteq S^\#$

Solution: Galois connections

Galois Connection

$L_1$, $L_2$ two lattices

\[ (L_1, \subseteq) \leftrightarrow (L_2, \leq) \]

\[ \alpha \quad \gamma \]

Abstract domain

- $\forall x \forall y : \alpha(x) \leq y \iff x \subseteq \gamma(y)$
- $\forall x \forall y : x \subseteq \gamma \circ \alpha(x) \land \alpha \circ \gamma(y) \leq y$
Fixpoint Approximation

\[ L_2 \xrightarrow{\alpha \circ F \circ \gamma} L_2 \]
\[ L_1 \xrightarrow{\gamma} L_1 \]
\[ L_1 \xrightarrow{F} L_1 \]

Theorem:
\[ \text{Ifp } F \subseteq \gamma (\text{Ifp } \alpha \circ F \circ \gamma) \]

Abstracting the Collecting Semantics

- Find a Galois connection:
\[ (\wp(\Sigma), \subseteq) \xleftrightarrow{\gamma} (\Sigma^#, \leq) \]
\[ \xrightarrow{\alpha} \]

- Find a function: \[ \alpha \circ F \circ \gamma \leq F^# \]

Partitioning \(\Rightarrow\) Abstract sets of environments
Abstract Algebra

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

\[
(\varnothing(E), \subseteq) \leftrightarrow (E^\#, \leq) \quad \gamma \quad \alpha
\]

- $\cup$, $\cap$ approximated by $\cup^\#$, $\cap^\#
- Semantics $\lbrack op \rbrack$ approximated by $\lbrack op \rbrack^\#

\[
\alpha \circ \lbrack op \rbrack \circ \gamma \subseteq \lbrack op \rbrack^\#
\]

Abstract Semantic Equations

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

\[
E_1^\# = \alpha (\{n \Rightarrow \Omega\})
\]
\[
E_2^\# = [n = 0] \# E_1^\# \cup^\# E_4^#
\]
\[
E_3^\# = E_2^\# \cap^\# \alpha ([-\infty, 999])
\]
\[
E_4^\# = [n = n + 1] \# E_3^#
\]
\[
E_5^\# = E_2^\# \cap^\# \alpha ([1000, +\infty[)
\]
Abstract Domains

Environment: \( x \Rightarrow v, \ y \Rightarrow w, \ldots \)

Various kinds of approximations:

- **Intervals** (nonrelational):
  \( x \Rightarrow [a, b], \ y \Rightarrow [a', b'], \ldots \)

- **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

Iterative Resolution Algorithms

Tuners

Abstract Domain

Abstract Domain

Widening operator

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

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

- 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

\[ [a, b] \lor [a', b'] \]

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

⇒ Open unstable bounds (jump over the fixpoint)

Widening and Fixpoint

![Diagram showing widening and fixpoint](image)
Iteration with widening

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

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

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

Imprecision at loop exit

1: \( n = 0; \)
2: \[ \text{while } n < 1000 \text{ do} \]
3: \( n = n + 1; \)
4: \text{end}
5: \text{exit, } t[n] = 0; // t has 1500 elements

\[ \text{False positive!!!} \]

- \( E_5^# = [1000, +\infty[ \)
- The information is present in the equations
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: \((x_n)_{n \geq 0}\)
  \[
  \begin{align*}
  y_0 &= x_0 \\
  y_{n+1} &= y_n \Delta x_{n+1}
  \end{align*}
  \]

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

Narrowing of intervals

\([a, b] \Delta [a', b']\)

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

\(\Rightarrow\) Refine open bounds
Narrowing and Fixpoint

Iteration with narrowing

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

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

Beginning of iteration: \( E_2^\# = [0, +\infty[ \)
Iteration 1: \( E_2^\# = [0, 1000] \Rightarrow \text{stable} \)
Consequence: \( E_5^\# = [1000, 1000] \)
Methodology

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 or DBMs:
  \[ E_4^\# = \langle 0 \leq n \leq 1000, \ 0 \leq k \leq 1000, \ n - k = 0 \rangle \]
Comparison with Data Flow Analysis

Data Flow Framework

- Forward Data Flow Equations
  \[ \text{in}(B) = \begin{cases} \text{Init} & , B = \text{entry} \\ \bigcap_{P \in \text{Pred}(B)} F_B \text{(in}(B)) & , \text{otherwise} \end{cases} \]

- L is a lattice
- \text{in}(B) \in L is the data-flow information on entry to B
- \text{Init} is the appropriate initial value on entry to the program
- \( F_B \) is the transformation of the data-flow information upon executing block B
- \( \cap \) models the effect of combining the data-flow information on the edges entering a block
Data-Flow Solutions

• Solving the data-flow equations computes the meet-over-all-paths (MOP) solution

\[
\text{MOP}(B) = \bigcap_{p \in \text{Path}(B)} \text{F}_p(\text{Init}) \text{ for } B = \text{entry}, B_1, \ldots, B_n, \text{exit}
\]

• If \( F_B \) is monotone, i.e.,

\[
F_B(x \cap y) \subseteq F_B(x) \cap F_B(y)
\]

• then \( \text{MOP} \leq \text{MFP} \) (maximum fixpoint)

• If \( F_B \) is distributive, i.e.,

\[
F_B(x \cap y) = F_B(x) \cap F_B(y)
\]

• then \( \text{MOP} = \text{MFP} \)

Typical Data-Flow Analyses

• Reaching Definitions
• Available Expressions
• Live Variables
• Upwards-Exposed Uses
• Copy-Propagation Analysis
• Constant-Propagation Analysis
• Partial-redundancy Analysis
Reaching Definitions

- Data-flow equations:
  \[ \forall i: \text{RCHin}(i) = \bigcup \ (\text{GEN}(j) \cup (\text{RCHin}(j) \cap \text{PRSV}(j))) \]
  where
  - PRSV are the definitions preserved by the block
  - GEN are the definitions generated by the blocks

- This is an iterative forward bit-vector problem
  - Iterative: it is solved by iteration from a set of initial values
  - Forward: information flows in direction of execution
  - Bit-vector: each definition is represented as a 1 (may reach given point) or a 0 (it does not reach this point)

AI versus classical DFA

- Classical DFA is stated in terms of properties whereas AI is usually stated in terms of models, whence the duality in the formulation.
- In classical DFA the proof of soundness must be made separately whereas it comes from the construction of the analysis in AI.
- Added benefits of AI:
  - Approximation of fixpoints in AI
    - Widening operators
    - Narrowing operators
  - Abstraction is explicit in AI
    - Galois connections
    - Can build a complex analysis as combination of basic, already-proved-correct, analyses
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):
References

- The abstract domain of intervals:

- The abstract domain of convex polyhedra:

- Weakly relational abstract domains:

- Classical data flow analysis: