Abstract Interpretation with Applications to Semantics and Static Analysis

Patrick Cousot
École normale supérieure
45 rue d’Ulm, 75230 Paris cedex 05, France
Patrick.Cousot@ens.fr www.di.ens.fr/~cousot

Visiting IBM T.J. Watson Research Center —— Hawthorne N.Y.

Computer Science Colloquium — NYU
Monday April 9th, 2007
Contents

The problem .................................................... 3
Abstract interpretation ........................................ 7
Theoretical application: semantics of the eager \( \lambda \)-calculus 11
Practical application: static analysis .......................... 41
The ASTRÉE static analyzer .................................. 48
A few research directions ...................................... 76
Conclusion ....................................................... 82
Recent publications ............................................ 85
Other references ............................................... 98
1. The Problem: The Design of Safe and Secure Computer-Based Systems
Software is Everywhere

- Exponential growth of hardware since 1975
  ⇒ exponential growth of software (favored by software engineering methods)
- Mainly manual activity ⇒ bugs are everywhere
Guaranteeing the Reliability and Security of Software-Intensive Systems

- A permanent objective since the origin of computer science
- An industrial requirement, in particular for safety and security critical software (validation can account for up to 60% of software development costs)
Validation/Formal Methods

- **Bug-finding methods**: unit, integration, and system testing, dynamic verification, bounded model-checking, error pattern mining, ...

- **Absence of bug proving methods**: formally prove that the semantics of a program satisfies a specification
  - theorem-proving & proof checking
  - model-checking
  - abstract interpretation

- In practice: complementary methods are used, very difficult to **scale up**
2. Abstract Interpretation
The Theory of Abstract Interpretation

- A theory of sound approximation of mathematical structures, in particular those involved in the behavior of computer systems
- Systematic derivation of sound methods and algorithms for approximating undecidable or highly complex problems in various areas of computer science
- Main practical application is on the safety and security of complex hardware and software computer systems
- Abstraction: extracting information from a system description that is relevant to proving a property
Applications of Abstract Interpretation

- **Static Program Analysis** [54], [59], [55] including Dataflow Analysis; [55], [58], Set-based Analysis [57], Predicate Abstraction [3], ...

- **Grammar Analysis and Parsing** [6];

- **Hierarchies of Semantics and Proof Methods** [56], [5];

- **Typing & Type Inference** [53];

- **(Abstract) Model Checking** [58];

- **Program Transformation** (including program optimization, partial evaluation, etc) [12];
Applications of Abstract Interpretation (Cont’d)

- Software Watermarking [14];
- Bisimulations [71];
- Language-based security [63];
- Semantics-based obfuscated malware detection [70].
- Databases [50, 51, 52]
- Computational biology [60]
- Quantum computing [64, 68]

All these techniques involve sound approximations that can be formalized by abstract interpretation.
3. An Example of Theoretical Application: Semantics of the Eager λ-calculus

Syntax
Syntax of the Eager $\lambda$-calculus

$x, y, z, \ldots \in X$ variables

$c \in C$ constants ($X \cap C = \emptyset$)

c ::= 0 | 1 | \ldots

$v \in V$ values

$v ::= c | \lambda x \cdot a$

$e \in E$ errors

e ::= c a | e a

$a, a', a_1, \ldots, b, \ldots \in T$ terms

$a ::= x | v | a a'$
Trace Semantics
Example I: Finite Computation

\[
\begin{align*}
\text{function} & \quad \text{argument} \\
((\lambda x \cdot x \cdot x) \quad (\lambda y \cdot y)) & \quad ((\lambda z \cdot z) \quad 0) \\
\rightarrow & \quad \text{evaluate function} \\
((\lambda y \cdot y) \quad (\lambda y \cdot y)) & \quad ((\lambda z \cdot z) \quad 0) \\
\rightarrow & \quad \text{evaluate function, cont’d} \\
(\lambda y \cdot y) & \quad ((\lambda z \cdot z) \quad 0) \\
\rightarrow & \quad \text{evaluate argument} \\
(\lambda y \cdot y) \quad 0 & \\
\rightarrow & \quad \text{apply function to argument} \\
0 & \quad \text{a value!}
\end{align*}
\]
Example II: Infinite Computation

\[
\begin{align*}
\text{function} & \quad \text{argument} \\
(\lambda x \cdot x \ x) & \quad (\lambda x \cdot x \ x) \\
\rightarrow & \quad \text{apply function to argument} \\
(\lambda x \cdot x \ x) & \quad (\lambda x \cdot x \ x) \\
\rightarrow & \quad \text{apply function to argument} \\
(\lambda x \cdot x \ x) & \quad (\lambda x \cdot x \ x) \\
\rightarrow & \quad \text{apply function to argument} \\
\ldots & \quad \text{non termination!}
\end{align*}
\]
Example III: Erroneous Computation

\[
\begin{align*}
\text{function} & \quad \text{argument} \\
((\lambda x \cdot x \cdot x) \ ((\lambda z \cdot z) 0)) & \ ((\lambda y \cdot y) 0) \\
\rightarrow & \\
((\lambda x \cdot x \cdot x) \ ((\lambda z \cdot z) 0)) & \ 0 \\
\rightarrow & \\
((\lambda x \cdot x \cdot x) 0) & \ 0 \\
\rightarrow & \\
(0 0) & \ 0
\end{align*}
\]

* a runtime error! *
Finite, Infinite and Erroneous Trace Semantics

\[ s(t) \]

Error


© P. Cousot
Traces

- $T^*$ (resp. $T^+$, $T^\omega$, $T^\alpha$ and $T^\infty$) be the set of finite (resp. nonempty finite, infinite, finite or infinite, and nonempty finite or infinite) sequences of terms
- $\varepsilon$ is the empty sequence $\varepsilon \cdot \sigma = \sigma \cdot \varepsilon = \sigma$.
- $|\sigma| \in \mathbb{N} \cup \{\omega\}$ is the length of $\sigma \in T^\alpha$. $|\varepsilon| = 0$.
- If $\sigma \in T^+$ then $|\sigma| > 0$ and $\sigma = \sigma_0 \cdot \sigma_1 \cdot \ldots \cdot \sigma |\sigma|_1$.
- If $\sigma \in T^\omega$ then $|\sigma| = \omega$ and $\sigma = \sigma_0 \cdot \ldots \cdot \sigma_n \cdot \ldots$. 
Operations on Traces

- For $a \in T$ and $\sigma \in T^\infty$, we define $a \circ \sigma$ to be $\sigma' \in T^\infty$ such that $\forall i < |\sigma| : \sigma'_i = a \cdot \sigma_i$

\[
\begin{align*}
\sigma &= \sigma_0 \sigma_1 \sigma_2 \sigma_3 \ldots \sigma_i \\
a \circ \sigma &= a \sigma_0 a \sigma_1 a \sigma_2 a \sigma_3 \ldots a \sigma_i \ldots
\end{align*}
\]

- Similarly $\sigma \circ a$ is $\sigma'$ where $\forall i < |\sigma| : \sigma'_i = \sigma_i a$

\[
\begin{align*}
\sigma &= \sigma_0 \sigma_1 \sigma_2 \sigma_3 \ldots \sigma_i \\
\sigma \circ a &= \sigma_0 a \sigma_1 a \sigma_2 a \sigma_3 a \ldots \sigma_i a \ldots
\end{align*}
\]
Finite and Infinite Trace Semantics

s(t)

Error

0 1 2 3 4 5 6 7 8 9 10 ...

t
Bifinitary Trace Semantics $\tilde{S}$ of the Eager $\lambda$-calculus \footnote{Note: $a[x \leftarrow b]$ is the capture-avoiding substitution of $b$ for all free occurrences of $x$ within $a$. We let $\text{FV}(a)$ be the free variables of $a$. We define the call-by-value semantics of closed terms (without free variables) $\bar{T} \triangleq \{a \in T \mid \text{FV}(a) = \emptyset\}$.} [56]

\[
\begin{align*}
\forall v \in \tilde{S}, \ v \in \mathcal{V} & \quad \frac{a[x \leftarrow v] \cdot \sigma \in \tilde{S}}{\varepsilon, \ v \in \mathcal{V}} \\
\sigma \in \tilde{S}^\omega & \quad \frac{(\lambda x \cdot a) v \cdot a[x \leftarrow v] \cdot \sigma \in \tilde{S}}{\varepsilon, \ v \in \mathcal{V}} \\
\sigma \in \tilde{S}^\omega & \quad \frac{\sigma \cdot v \in \tilde{S}^+, \ (a v) \cdot \sigma' \in \tilde{S}}{\varepsilon, \ v, a \in \mathcal{V}} \\
\sigma \in \tilde{S}^\omega & \quad \frac{\sigma \cdot v \in \tilde{S}^+, \ (v b) \cdot \sigma' \in \tilde{S}}{\varepsilon, \ v \in \mathcal{V}} \\
\end{align*}
\]
Non-Standard Meaning of the Rules

The rules

\[
\mathcal{R} = \left\{ \frac{P_i}{C_i} \sqsubseteq \mid i \in \Delta \right\}
\]

define

\[
\text{lfp} \sqsubseteq F[\mathcal{R}]
\]

where the *consequence operator* is

\[
F[\mathcal{R}](T) = \bigsqcup \left\{ C \mid P \sqsubseteq T \land \frac{P}{C} \sqsubseteq \in \mathcal{R} \right\}
\]

and \ldots
The Computational Lattice

Given \( S, T \in \mathcal{P}(\mathbb{T}^\infty) \), we define

- \( S^+ \triangleq S \cap \mathbb{T}^+ \)  \hspace{1cm} finite traces
- \( S^\omega \triangleq S \cap \mathbb{T}^\omega \)  \hspace{1cm} infinite traces
- \( S \sqsubseteq T \triangleq S^+ \subseteq T^+ \land S^\omega \supseteq T^\omega \)  \hspace{1cm} computational order
- \( \langle \mathcal{P}(\mathbb{T}^\infty), \sqsubseteq, \mathbb{T}^\omega, \mathbb{T}^+, \sqcup, \sqcap \rangle \) is a complete lattice
Relational Semantics
Trace Semantics

\[ s(t) \]

\[ t_0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \]

Error
Relational Semantics = $\alpha$(Trace Semantics)
Relational Semantics

s(t)

---

© P. Cousot
Abstraction to the Bifinitary Relational Semantics of
the Eager $\lambda$-calculus

remember the input/output behaviors,
forget about the intermediate computation steps

$$\alpha(T) \overset{\text{def}}{=} \{ \alpha(\sigma) \mid \sigma \in T \}$$

$$\alpha(\sigma_0 \cdot \sigma_1 \cdot \ldots \cdot \sigma_n) \overset{\text{def}}{=} \langle \sigma_0, \sigma_n \rangle$$

$$\alpha(\sigma_0 \cdot \ldots \cdot \sigma_n \cdot \ldots) \overset{\text{def}}{=} \langle \sigma_0, \bot \rangle$$
Bifinitary Relational Semantics of the Eager $\lambda$-calculus

\[
\begin{align*}
  &v \Rightarrow v, \quad v \in \mathbb{V} \\
  &a \Rightarrow \bot \quad \subseteq \\
  &a \ b \Rightarrow \bot \\
  &a[x \leftarrow v] \Rightarrow r \quad \subseteq, \quad v \in \mathbb{V}, \ r \in \mathbb{V} \cup \{\bot\} \\
  &\lambda x \cdot a \ v \Rightarrow r \\
  &a \Rightarrow v, \quad v \ b \Rightarrow r \quad \subseteq, \quad v \in \mathbb{V}, \ r \in \mathbb{V} \cup \{\bot\} \\
  &a \ b \Rightarrow r \\
  &b \Rightarrow v, \quad a \ v \Rightarrow r \quad \subseteq, \quad a \in \mathbb{V}, \ v \in \mathbb{V}, \ r \in \mathbb{V} \cup \{\bot\}. \\
  &a \ b \Rightarrow r
\end{align*}
\]
Natural Semantics
Natural Semantics = \( \alpha (\text{Relational Semantics}) \)
Abstraction to the Natural Big-Step Semantics of the Eager $\lambda$-calculus

remember the finite input/output behaviors, forget about non-termination

\[ \alpha(T) \overset{\text{def}}{=} \bigcup \{ \alpha(\sigma) \mid \sigma \in T \} \]

\[ \alpha(\langle \sigma_0, \sigma_n \rangle) \overset{\text{def}}{=} \{ \langle \sigma_0, \sigma_n \rangle \} \]

\[ \alpha(\langle \sigma_0, \perp \rangle) \overset{\text{def}}{=} \emptyset \]
Natural Big-Step Semantics of the Eager $\lambda$-calculus [65]

\[ \begin{align*}
  v &\Rightarrow v, \quad v \in \mathbb{V} \\
  a[x \leftarrow v] &\Rightarrow r \\
  \frac{a \Rightarrow v}{(\lambda x \cdot a) \Rightarrow r} \quad \subseteq, \quad v \in \mathbb{V}, \ r \in \mathbb{V} \\
  a \Rightarrow v, \quad v \ b &\Rightarrow r \\
  \frac{a \ b \Rightarrow r}{\subseteq, \quad v \in \mathbb{V}, \ r \in \mathbb{V}} \\
  b \Rightarrow v, \quad a \ v \Rightarrow r \\
  \frac{a \ b \Rightarrow r}{\subseteq, \quad a \in \mathbb{V}, \ v \in \mathbb{V}, \ r \in \mathbb{V}} .
\end{align*} \]
Transition Semantics
Transition Semantics = $\alpha$(Trace Semantics)
Abstraction to the Transition Semantics of the Eager $\lambda$-calculus

remember execution steps,
forget about their sequencing

$$\alpha(T') \overset{\text{def}}{=} \bigcup \{ \alpha(\sigma) \mid \sigma \in T \}$$

$$\alpha(\sigma_0 \cdot \sigma_1 \cdot \ldots \cdot \sigma_n) \overset{\text{def}}{=} \{ \langle \sigma_i, \sigma_{i+1} \rangle \mid 0 \leq i \land i < n \}$$

$$\alpha(\sigma_0 \cdot \ldots \cdot \sigma_n \cdot \ldots) \overset{\text{def}}{=} \{ \langle \sigma_i, \sigma_{i+1} \rangle \mid i \geq 0 \}$$
Transition Semantics of the Eager $\lambda$-calculus [69]

$$(\lambda x \cdot a) v \longrightarrow a[x \leftarrow v]$$

\[
\begin{array}{c}
\frac{a_0 \longrightarrow a_1}{a_0 \, b \longrightarrow a_1 \, b} \subseteq \\
\frac{b_0 \longrightarrow b_1}{v \, b_0 \longrightarrow v \, b_1} \subseteq .
\end{array}
\]
Approximation

\[
((\lambda x \cdot x \cdot x) \ ((\lambda z \cdot z) \ 0)) \ (\lambda y \cdot y) \rightarrow ((\lambda x \cdot x \cdot x) \ 0) \ (\lambda y \cdot y) \\
\rightarrow (0 \ 0) \ (\lambda y \cdot y) \quad \text{an error!}
\]
The Abstract Semantics are Correct by Calculational Design

\[ \text{\textbf{Theorem 1}} \quad \text{\textbf{Main Result}} \]

Let \( \mathcal{A} \) be a finite automaton. Then the set of reachable states is given by

\[ \text{lfp}(C) \]

where \( C \) is the closure of the equation

\[ \text{Σ} = \{ x \mid x = \text{glb}(a \cup b) \} \]

for any \( a, b \in \mathcal{A} \).

Proof. By mathematical induction on the number of states of \( \mathcal{A} \).

2.4. Relational Semantics

The relational semantics of a finite automaton \( \mathcal{A} \) is given by

\[ \text{rel}(C) \]

where \( C \) is the closure of the relation

\[ \text{Σ} = \{ x \mid x = \text{glb}(a \cup b) \} \]

for any \( a, b \in \mathcal{A} \).

Proof. By mathematical induction on the number of states of \( \mathcal{A} \).

2.5. observational Abramski semantics

The observational Abramski semantics of a finite automaton \( \mathcal{A} \) is given by

\[ \text{obsv}(C) \]

where \( C \) is the closure of the relation

\[ \text{Σ} = \{ x \mid x = \text{glb}(a \cup b) \} \]

for any \( a, b \in \mathcal{A} \).

Proof. By mathematical induction on the number of states of \( \mathcal{A} \).

2.6. decorated Abramski semantics

The decorated Abramski semantics of a finite automaton \( \mathcal{A} \) is given by

\[ \text{dec}(C) \]

where \( C \) is the closure of the relation

\[ \text{Σ} = \{ x \mid x = \text{glb}(a \cup b) \} \]

for any \( a, b \in \mathcal{A} \).

Proof. By mathematical induction on the number of states of \( \mathcal{A} \).

2.7. Relational Abramski semantics

The relational Abramski semantics of a finite automaton \( \mathcal{A} \) is given by

\[ \text{rel}(C) \]

where \( C \) is the closure of the relation

\[ \text{Σ} = \{ x \mid x = \text{glb}(a \cup b) \} \]

for any \( a, b \in \mathcal{A} \).

Proof. By mathematical induction on the number of states of \( \mathcal{A} \).

2.8. decorated relational Abramski semantics

The decorated relational Abramski semantics of a finite automaton \( \mathcal{A} \) is given by

\[ \text{dec}(C) \]

where \( C \) is the closure of the relation

\[ \text{Σ} = \{ x \mid x = \text{glb}(a \cup b) \} \]

for any \( a, b \in \mathcal{A} \).

Proof. By mathematical induction on the number of states of \( \mathcal{A} \).

2.9. observational decorated Abramski semantics

The observational decorated Abramski semantics of a finite automaton \( \mathcal{A} \) is given by

\[ \text{obsv}(C) \]

where \( C \) is the closure of the relation

\[ \text{Σ} = \{ x \mid x = \text{glb}(a \cup b) \} \]

for any \( a, b \in \mathcal{A} \).

Proof. By mathematical induction on the number of states of \( \mathcal{A} \).
4. Principle of Static Analysis
Principle of Static Analysis
(1) Concrete Semantics

\[ s(t) \]

Error

0 1 2 3 4 5 6 7 8 9 10 ... t
Principle of Static Analysis
(2) Specification
Principle of Static Analysis
(3.1) Abstract Semantics
Principle of Static Analysis

(3.2) Abstract Semantics
Unsoundness
(False Negatives)
Incomplete
(False Positive/Alarms)
5. An Example of Practical Application: The ASTRÉE Static Analyzer
Project Members

Patrick Cousot  Radhia Cousot  Jérôme Feret
Laurent Mauborgne  Antoine Miné  David Monniaux
Xavier Rival
Programs
Programs Analysed by ASTRÉE

- **Application Domain:** large safety critical embedded synchronous software (for real-time non-linear control of very complex control/command systems).

- **C programs:**
  - with
  - basic numeric datatypes, structures and arrays
  - pointers (including on functions),
  - floating point computations
  - tests, loops and function calls
  - limited branching (forward goto, break, continue)
– with (cont’d)
  - union
  - pointer arithmetics & casts
– without
  - dynamic memory allocation
  - recursive function calls
  - unstructured/backward branching
  - conflicting side effects
  - C libraries, system calls (parallelism)

Such limitations are quite common for embedded safety-critical software.
Concrete Semantics
Concrete Trace Semantics

- International **norm of C** (ISO/IEC 9899:1999)
- *restricted by* implementation-specific behaviors depending upon the machine and compiler (e.g. representation and size of integers, IEEE 754-1985 norm for floats and doubles)
- *restricted by* user-defined **programming guidelines** (such as no modular arithmetic for signed integers, even though this might be the hardware choice)
- *restricted by* program specific **user requirements** (e.g. assert)
The Semantics of C is Hard (Ex. 1: Floats)

“Put \( x \) in \([m, M]\) modulo \((M - m)\)”: 

\[
x' = x - (\text{int}) \left( \frac{(x-m)}{(M-m)} \right) (M-m);
\]

- The programmer thinks \( x' \in [m, M] \)
- But with \( M = 4095 \), \( m = -M \), IEEE double precision, and \( x \) is the greatest float strictly less than \( M \), then \( x' = m - \epsilon \) (\( \epsilon \) very small).

Floats are not real.
The Semantics of C is Hard (Ex. 2: Runtime Errors)

What is the effect of out-of-bounds array indexing?

```c
#include <stdio.h>
int main () { int n, T[1];
    n = 2147483647;
    printf("n = %i, T[n] = %i\n", n, T[n]);
}
```

Yields different results on different machines:

- n = 2147483647, T[n] = 2147483647  Macintosh PPC
- n = 2147483647, T[n] = -1208492044  Macintosh Intel
- n = 2147483647, T[n] = -135294988  PC Intel 32 bits
- Bus error  PC Intel 64 bits

Execution stops after a runtime error with unpredictable results\(^2\).

\(^2\) Equivalent semantics if no alarm.
Specification
Implicit Specification: Absence of Runtime Errors

- No violation of the norm of C (e.g. array index out of bounds, division by zero)
- No implementation-specific undefined behaviors (e.g. maximum short integer is 32767, NaN)
- No violation of the programming guidelines (e.g. static variables cannot be assumed to be initialized to 0)
- No violation of the programmer assertions (must all be statically verified).
Example: Dichotomy Search

% cat dichotomy.c
int main () {
    int R[100], X; short lwb, upb, m;
    lwb = 0; upb = 99;
    while (lwb <= upb) {
        m = upb + lwb;
        m = m » 1;
        if (X == R[m]) { upb = m; lwb = m+1; }
        else if (X < R[m]) { upb = m - 1; }
        else { lwb = m + 1; }
    }
    __ASTREE_log_vars((m));
}

% astree -exec-fn main dichotomy.c |& egrep "(WARN)|(m in)"
direct = <integers (intv+cong+bitfield+set): m in [0, 99] \ Top >
%
Example: Dichotomy Search II

% diff dichotomy.c dichotomy-bug.c
2,3c2,3
<   int R[100], X; short lwb, upb, m;
<   lwb = 0; upb = 99;
--
>   int R[30000], X; short lwb, upb, m;
>   lwb = 0; upb = 29999;
%
% astree -exec-fn main dichotomy-bug.c |& egrep "WARN" | head -n2
dichotomy-bug.c:5.6-19::[call#main@1:loop@4=2::]: WARN: implicit signed int->signed short conversion range [14998, 44999] not included in [-32768, 32767]
dichotomy-bug.c:7.15-19::[call#main@1:loop@4=2::]: WARN: invalid dereference: dereferencing 4 byte(s) at offset(s) [0;4294967295] may overflow the variable R of byte-size 120000 or mis-aligned pointer (1Z+0) may not a multiple of 4%

ASTRÉE finds bugs in programs based on algorithms which have been formally proved correct.
Specification Can Be Tricky

– What is known about the execution environment?
– Warn on integer arithmetic overflows? Including left shifts (to extract bit fields)? Including in initializers?
– Warn on implicit cast/conversion? When they overflow\(^3\)?
– What is an incorrect access to a union field?
– ...

A “reasonable default choice” with analysis parameters for variants

\(^3\) undefined except for unsigned to unsigned.
Abstraction
Abstraction is Extremely Hard

- The analysis must be **automatic** (no user interaction)
- The abstraction must
  - ensure **termination** (and efficiency) of the analysis
  - be **sound** (*ASTRÉE* is a **verifier**, not a bug-finder)
  - scale up (100,000 to 1,000,000 LOCs)
  - be **precise** (no false alarm)

A grand challenge
General-Purpose Abstract Domains: Intervals and Octagons

Intervals:
\[
\begin{align*}
1 & \leq x \leq 9 \\
1 & \leq y \leq 20
\end{align*}
\]

Octagons [66]:
\[
\begin{align*}
1 & \leq x \leq 9 \\
x + y & \leq 77 \\
1 & \leq y \leq 20 \\
x - y & \leq 04
\end{align*}
\]

Difficulties: many global variables, arrays (smashed or not), IEEE 754 floating-point arithmetic (in program and analyzer) [54, 66, 67]
Termination

SLAM uses CEGAR and does not terminate\(^4\) on

```c
% cat slam.c
int main() { int x, y;
  x = 0; y = 0;
  while (x < 2147483647)
    { x = x + 1; y = y + 1; }
  __ASTREE_assert((x == y));
}
```

whereas ASTRéE uses widening/narrowing-based extrapolation techniques to prove the assertion

```c
% astree -exec-fn main slam.c |& egrep "WARN"
%
```

\(^4\) CEGAR cannot generate the invariant \(y = x - 1\) so produces all counter examples \(x = i + 1 \land y = i,\)
\(i = 0, 1, 2, 3, \ldots\)
Boolean Relations for Boolean Control

- Code Sample:

```c
/* boolean.c */
typedef enum {F=0,T=1} BOOL;
BOOL B;
void main () {
    unsigned int X, Y;
    while (1) {
        ...
        B = (X == 0);
        ...
        if (!B) {
            Y = 1 / X;
        }
        ...
    }
}
```

The boolean relation abstract domain is parameterized by the height of the decision tree (an analyzer option) and the abstract domain at the leafs.
2d Order Digital Filter:

Ellipsoid Abstract Domain for Filters

- Computes \( X_n = \begin{cases} \alpha X_{n-1} + \beta X_{n-2} + Y_n \\ I_n \end{cases} \)
- The concrete computation is bounded, which must be proved in the abstract.
- There is no stable interval or octagon.
- The simplest stable surface is an ellipsoid.
typedef enum {FALSE = 0, TRUE = 1} BOOLEAN;  

BOOLEAN INIT;  
float P, X;

void filter () {
    static float E[2], S[2];
    if (INIT) { S[0] = X; P = X; E[0] = X; }
    else { P = ((((((0.5 * X) - (E[0] * 0.7)) + (E[1] * 0.4))
            + (S[0] * 1.5)) - (S[1] * 0.7)); }
    E[1] = E[0]; E[0] = X; S[1] = S[0]; S[0] = P;
    /* S[0], S[1] in [-1327.02698354, 1327.02698354] */
}

void main () {  
    X = 0.2 * X + 5; INIT = TRUE;
    while (1) {
        X = 0.9 * X + 35; /* simulated filter input */
        filter (); INIT = FALSE; }
}
Time Dependent Deviations [62]

```c
void main()
{
  FIRST = TRUE;
  while (TRUE) {
    dev( );
    FIRST = FALSE;
    __ASTREE_wait_for_clock();
  }
}
```

```c
% cat retro.config
__ASTREE_volatile_input((E [-15.0, 15.0]));
__ASTREE_volatile_input((SWITCH [0,1]));
__ASTREE_max_clock((3600000));

|P| <= (15. + 5.87747175411e-39 / 1.19209290217e-07) * (1 + 1.19209290217e-07)^clock - 5.87747175411e-39 / 1.19209290217e-07 <= 23.0393526881
```

```c
% cat retro.c
typedef enum {FALSE=0, TRUE=1} BOOL;
BOOL FIRST;
volatile BOOL SWITCH;
volatile float E;
float P, X, A, B;

void dev( )
{
  X=E;
  if (FIRST) { P = X; }
  else {
    P = (P - (((2.0 * P) - A) - B) * 4.491048e-03));
    B = A;
    if (SWITCH) {A = P;} 
    else {A = X;}
  }
}
```
Incompleteness

Astrée does not know that

$$\forall x, y \in \mathbb{Z} : 7y^2 - 1 \neq x^2$$

so on the following program

```c
void main() { int x, y;
    if ((-4681 < y) && (y < 4681) && (x < 32767) && (-32767 < x) && ((7*y*y - 1) == x*x))
        { y = 1 / x;};
}
```

it produces a false alarm

% astree -exec-fn main false-alarm.c |& egrep "WARN"
false-alarm.c:5.9-14::[call#main@1::]: WARN: integer division by zero ([-32766, 32766]
and {1} / Z)
%
Zero False Alarm Objective

Industrial constraints require ASTRÉE to be extremely precise:

- ASTRÉE is designed for a well-identified family of programs
- The analysis can be tuned using
  - parameters
  - analysis directives (which insertion can be automated)
  - extensions of the analyzer (by the tool designers)
Example of directive

% cat repeat1.c
typedef enum {FALSE=0,TRUE=1} BOOL;
int main () {
    int x = 100; BOOL b = TRUE;

    while (b) {
        x = x - 1;
        b = (x > 0);
    }
}

% astree -exec-fn main repeat1.c |& egrep "WARN"
repeat1.c:5.8-13::[call#main@2:loop@4>=4::]: WARN: signed int arithmetic
range [-2147483649, 2147483646] not included in [-2147483648, 2147483647]
%
% cat repeat2.c
typedef enum {FALSE=0,TRUE=1} BOOL;
int main () {
    int x = 100; BOOL b = TRUE;
    __ASTREE_boolean_pack((b,x));
    while (b) {
        x = x - 1;
        b = (x > 0);
    }
}

% astree -exec-fn main repeat2.c |& egrep "WARN"
%

The insertion of this directive could have been automated.
Industrial Application
Application to Avionics Software

- **Primary flight control software** \(^5\)

- **C** program, automatically generated from a proprietary high-level specification (à la Simulink/SCADE)

- **A340 family**: 200,000 lines \(^6\), **A380**: \(\times 5\)

  *No false alarm, a world première!*

---

\(^5\) “Flight Control and Guidance Unit” (FCGU) running on the “Flight Control Primary Computers” (FCPC). The A340 electrical flight control system is placed between the pilot’s controls (sidesticks, rudder pedals) and the control surfaces of the aircraft, whose movement they control and monitor.

\(^6\) 6 hours on a 2.6 GHz, 16 Gb RAM PC
6. A Few Research Directions
Abstraction of Computations

– **Semantics of concurrency** (anticipated evolution of hardware)

– **Abstract properties and specifications**: safety, liveness, security, probabilistic behaviors, ...

– **Time abstraction**: continuous to discrete, scheduling, performance properties
Abstraction of Computational Paradigms

- Abstraction of data structures
- Abstraction of control structures: imperative, functional, procedural, logical, synchronous, parallel, distributed, and mobile control paradigms
- Abstraction of program structures: procedures, modules, objects, classes, ...
- Abstraction of communication and cooperation structures: synchronous/asynchronous lossy/lossless channels, events, semaphores, mobile communications, exogenous systems, ...
– **Abstraction of hardware structures**: memory caches, pipelines, branch prediction . . . at the assembler level, hardware description languages

– **Abstraction of biological systems**: abstraction of agent-based descriptions of biological systems
Abstraction Validation

- **Abstraction translation**: translation of abstractions while translating models (from mathematical models to programs)
- **Verified abstractions**: beyond toy examples
Abstraction Automatization

- **Imprecision localization**: origin of false alarms
- **Automatic refinement**: automatic design of abstract domains to eliminate false alarms
- **Automatic abstraction**: too precise abstractions are costly
7. Conclusion
Abstract Interpretation

- Abstract interpretation is
  - a theory
  - with effective applications
  - and unprecedented industrial accomplishments.
- Further investigations of the theory are needed (while its scope of application broaden)
- The demand for applications is quasi-illimited
THE END, THANK YOU
8. Recent Publications

Invited Book Chapters


The titles of the publications are clickable references to their web location, whenever available.

Refereed Journal Publications


Invited Conference or Workshop Proceedings Publications


Refereed Conference or Workshop Proceedings Publications


Recent Software


Patents


Invited Conference Lectures and Tutorials


Recent Invited Seminar Presentations


9. Other References


10. Annex
\[ - \, a = (\lambda y \cdot y) \\
- \, \sigma = ((\lambda z \cdot z) \, 0) \cdot 0 \\
- \, a@\sigma = \\
\quad (\lambda y \cdot y)@((\lambda z \cdot z) \, 0) \cdot 0 = \\
\quad ((\lambda y \cdot y) \, ((\lambda z \cdot z) \, 0)) \cdot ((\lambda y \cdot y) \, 0) \]
\[ \sigma = ((\lambda x \cdot x x) (\lambda y \cdot y)) \cdot ((\lambda y \cdot y) (\lambda y \cdot y)) \cdot (\lambda y \cdot y) \]
\[ b = ((\lambda z \cdot z) 0) \]
\[ (\sigma @ b) \]
\[ = \]
\[ (((\lambda x \cdot x x) (\lambda y \cdot y)) \cdot ((\lambda y \cdot y) (\lambda y \cdot y)) \cdot (\lambda y \cdot y) @ ((\lambda z \cdot z) 0)) \]
\[ = \]
\[ (((\lambda x \cdot x x) (\lambda y \cdot y)) ((\lambda z \cdot z) 0)) \cdot (((\lambda y \cdot y) (\lambda y \cdot y)) ((\lambda z \cdot z) 0)) \cdot ((\lambda y \cdot y) ((\lambda z \cdot z) 0)) \]
\[
\sigma \cdot v \in \tilde{S}^+, \quad (a \cdot v) \cdot \sigma' \in \tilde{S}
\]
\[
\subseteq, \quad v, a \in V.
\]

- \( \sigma \cdot v = ((\lambda z \cdot z) \cdot 0) \cdot 0 \in \tilde{S}^+ \)
- \( (a \cdot v) \cdot \sigma' = (\lambda y \cdot y) \cdot 0 \cdot 0 \in \tilde{S} \)
- \( (a \cdot @\sigma) \cdot (a \cdot v) \cdot \sigma' \)
  \[
  = ((\lambda y \cdot y) \cdot (\lambda z \cdot z) \cdot 0) \cdot 0 \cdot 0
  \]
  \[
  = (\lambda y \cdot y) ((\lambda z \cdot z) \cdot 0) \cdot (\lambda y \cdot y) \cdot 0 \cdot 0 \in \tilde{S}
  \]
\[
\begin{align*}
\mathbf{\sigma} \cdot v & \in \mathit{\overline{S}}^{+}, \quad (v \ b) \cdot \mathbf{\sigma}' \in \mathit{\overline{S}} \\
(\mathbf{\sigma} @ b) \cdot (v \ b) \cdot \mathbf{\sigma}' & \in \mathit{\overline{S}} \\
\end{align*}
\]

\[- \mathbf{\sigma} \cdot v = ((\lambda x \cdot x \ x) (\lambda y \cdot y)) \cdot ((\lambda y \cdot y) (\lambda y \cdot y)) \cdot (\lambda y \cdot y) \in \mathit{\overline{S}}^{+} \]

\[- (v \ b) \cdot \mathbf{\sigma}' = (\lambda y \cdot y) ((\lambda z \cdot z) \ 0) \cdot (\lambda y \cdot y) \ 0 \ 0 \in \mathit{\overline{S}} \]

\[- (\mathbf{\sigma} @ b) \cdot (v \ b) \cdot \mathbf{\sigma}' \\
= \\
(((\lambda x \cdot x \ x) (\lambda y \cdot y)) \cdot ((\lambda y \cdot y) (\lambda y \cdot y)) @ ((\lambda z \cdot z) \ 0)) \cdot ((\lambda y \cdot y) ((\lambda z \cdot z) \ 0)) \cdot (\lambda y \cdot y) \ 0 \ 0 \\
= \\
((\lambda x \cdot x \ x) (\lambda y \cdot y)) ((\lambda z \cdot z) \ 0) \cdot ((\lambda y \cdot y) (\lambda y \cdot y)) ((\lambda z \cdot z) \ 0) \cdot (\lambda y \cdot y) ((\lambda z \cdot z) \ 0) \cdot (\lambda y \cdot y) \ 0 \ 0 \in \mathit{\overline{S}} \]