« 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

1. The Problem: The Design of Safe and Secure Computer-Based Systems

Software is Everywhere

- Exponential growth of hardware since 1975
  $\Rightarrow$ exponential growth of software (favored by *software engineering* methods)
- Mainly *manual* activity $\Rightarrow$ 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];

3. An Example of Theoretical Application: Semantics of the Eager λ-calculus

[1] P. Cousot & R. Cousot. Bi-inductive structural semantics. Februray 15^{th}, 2007. Submitted.

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

Syntax

Syntax of the Eager λ-calculus

$$
\begin{array}{rcll}
x, y, z, \ldots & \in & \mathbb{X} & \text{variables} \\
c & \in & \mathbb{C} & \text{constants } (\mathbb{X} \cap \mathbb{C} = \emptyset) \\
c & ::= & 0 \mid 1 \mid \ldots & \\
v & \in & \mathbb{V} & \text{values} \\
v & ::= & c \mid \lambda x \cdot a & \\
e & \in & \mathbb{E} & \text{errors} \\
e & ::= & c\ a \mid e\ a & \\
a, a', a_1, \ldots, b, , \ldots & \in & \mathbb{T} & \text{terms} \\
a & ::= & x \mid v \mid a\ a' &
\end{array}
$$

Trace Semantics

Example I: Finite Computation

function argument

$((\lambda x \cdot x\ x)\ (\lambda y \cdot y))\ ((\lambda z \cdot z)\ 0)$

$\longrightarrow$ evaluate function

$((\lambda y \cdot y)\ (\lambda y \cdot y))\ ((\lambda z \cdot z)\ 0)$

$\longrightarrow$ evaluate function, cont'd

$(\lambda y \cdot y)\ ((\lambda z \cdot z)\ 0)$

$\longrightarrow$ evaluate argument

$(\lambda y \cdot y)\ 0$

$\longrightarrow$ apply function to argument

0 *a value!*

Example II: Infinite Computation

function argument

$(\lambda x \cdot x\ x)\ (\lambda x \cdot x\ x)$

$\longrightarrow$ apply function to argument

$(\lambda x \cdot x\ x)\ (\lambda x \cdot x\ x)$

$\longrightarrow$ apply function to argument

$(\lambda x \cdot x\ x)\ (\lambda x \cdot x\ x)$

$\longrightarrow$ apply function to argument

$\ldots$ *non termination!*

Example III: Erroneous Computation

function argument

$((\boldsymbol{\lambda}\, x \cdot x\ x)\ ((\boldsymbol{\lambda}\, z \cdot z)\ 0))\ ((\boldsymbol{\lambda}\, y \cdot y)\ 0)$

$\rightarrow$ evaluate argument

$((\boldsymbol{\lambda}\, x \cdot x\ x)\ ((\boldsymbol{\lambda}\, z \cdot z)\ 0))\ 0$

$\rightarrow$ evaluate function

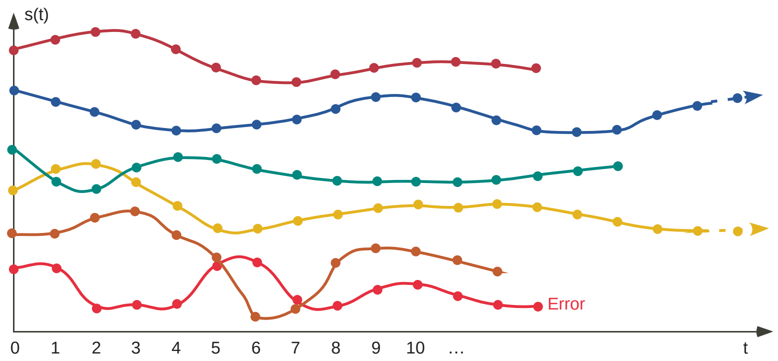
$((\boldsymbol{\lambda}\, x \cdot x\ x)\ 0)\ 0$

$\rightarrow$ evaluate function, cont'd

$(0\ 0)\ 0$

a runtime error!

Finite, Infinite and Erroneous Trace Semantics



Traces

- $\mathbb{T}^\star$ (resp. $\mathbb{T}^+$, $\mathbb{T}^\omega$, $\mathbb{T}^\propto$ and $\mathbb{T}^\infty$) be the set of finite (resp. nonempty finite, infinite, finite or infinite, and nonempty finite or infinite) sequences of terms
- ϵ is the empty sequence $\epsilon \bullet \sigma = \sigma \bullet \epsilon = \sigma$.
- $|\sigma| \in N \cup \{\omega\}$ is the length of $\sigma \in \mathbb{T}^\propto$. $|\epsilon| = 0$.
- If $\sigma \in \mathbb{T}^+$ then $|\sigma| > 0$ and $\sigma = \sigma_0 \bullet \sigma_1 \bullet \ldots \bullet \sigma_{|\sigma|-1}$.
- If $\sigma \in \mathbb{T}^\omega$ then $|\sigma| = \omega$ and $\sigma = \sigma_0 \bullet \ldots \bullet \sigma_n \bullet \ldots$.

Operations on Traces

- For $\mathsf{a} \in \mathbb{T}$ and $\sigma \in \mathbb{T}^\infty$, we define $a@\sigma$ to be $\sigma' \in \mathbb{T}^\infty$ such that $\forall i < |\sigma| : \sigma'_i = a\ \sigma_i$

$$\sigma = \sigma_0\ \sigma_1\ \sigma_2\ \sigma_3\ \ldots\ \sigma_i\ \ldots$$

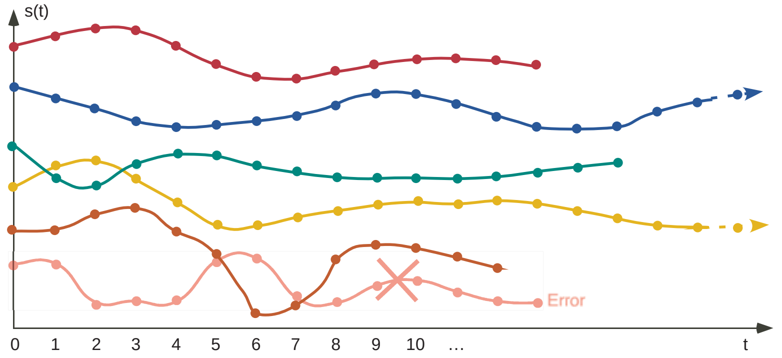
$$a@\sigma = a\ \sigma_0\ \ a\ \sigma_1\ \ a\ \sigma_2\ \ a\ \sigma_3\ \ldots\ a\ \sigma_i\ \ldots$$

- similarly $\sigma@a$ is σ' where $\forall i < |\sigma| : \sigma'_i = \sigma_i\ a$

$$\sigma = \sigma_0\ \sigma_1\ \sigma_2\ \sigma_3\ \ldots\ \sigma_i\ \ldots$$

$$\sigma@a = \sigma_0\ a\ \ \sigma_1\ a\ \ \sigma_2\ a\ \ \sigma_3\ a\ \ldots\ \sigma_i\ a\ \ldots$$

Finite and Infinite Trace Semantics



Bifinitary Trace Semantics $\vec{\mathbb{S}}$ of the Eager λ-calculus [1] [56]

$$\mathsf{v} \in \vec{\mathbb{S}},\ \mathsf{v} \in \mathbb{V} \qquad \frac{\mathsf{a}[\mathsf{x} \leftarrow \mathsf{v}] \bullet \sigma \in \vec{\mathbb{S}}}{(\boldsymbol{\lambda}\,\mathsf{x} \cdot \mathsf{a})\ \mathsf{v} \bullet \mathsf{a}[\mathsf{x} \leftarrow \mathsf{v}] \bullet \sigma \in \vec{\mathbb{S}}}\sqsubseteq,\ \mathsf{v} \in \mathbb{V}$$

$$\frac{\sigma \in \vec{\mathbb{S}}^{\omega}}{\mathsf{a}@\sigma \in \vec{\mathbb{S}}}\sqsubseteq,\ \mathsf{a} \in \mathbb{V} \qquad \frac{\sigma \bullet \mathsf{v} \in \vec{\mathbb{S}}^{+},\ (\mathsf{a}\ \mathsf{v}) \bullet \sigma' \in \vec{\mathbb{S}}}{(\mathsf{a}@\sigma) \bullet (\mathsf{a}\ \mathsf{v}) \bullet \sigma' \in \vec{\mathbb{S}}}\sqsubseteq,\ \mathsf{v}, \mathsf{a} \in \mathbb{V}$$

$$\frac{\sigma \in \vec{\mathbb{S}}^{\omega}}{\sigma@\mathsf{b} \in \vec{\mathbb{S}}}\sqsubseteq \qquad \frac{\sigma \bullet \mathsf{v} \in \vec{\mathbb{S}}^{+},\ (\mathsf{v}\ \mathsf{b}) \bullet \sigma' \in \vec{\mathbb{S}}}{(\sigma@\mathsf{b}) \bullet (\mathsf{v}\ \mathsf{b}) \bullet \sigma' \in \vec{\mathbb{S}}}\sqsubseteq,\ \mathsf{v} \in \mathbb{V}$$

[1] Note: $\mathsf{a}[\mathsf{x} \leftarrow \mathsf{b}]$ is the capture-avoiding substitution of b for all free occurences of x within a. We let $\mathrm{FV}(\mathsf{a})$ be the free variables of a. We define the call-by-value semantics of closed terms (without free variables) $\overline{\mathbb{T}} \triangleq \{\mathsf{a} \in \mathbb{T} \mid \mathrm{FV}(\mathsf{a}) = \varnothing\}$.

Non-Standard Meaning of the Rules

The rules

$$\mathcal{R} = \left\{ \frac{P_i}{C_i}\sqsubseteq \;\middle|\; i \in \Delta \right\}$$

define

$$\mathsf{lfp}^{\sqsubseteq}\, F[\![\mathcal{R}]\!]$$

where the *consequence operator* is

$$F[\![\mathcal{R}]\!](T) = \bigsqcup \left\{ C \;\middle|\; P \sqsubseteq T \wedge \frac{P}{C}\sqsubseteq \in \mathcal{R} \right\}$$

and ...

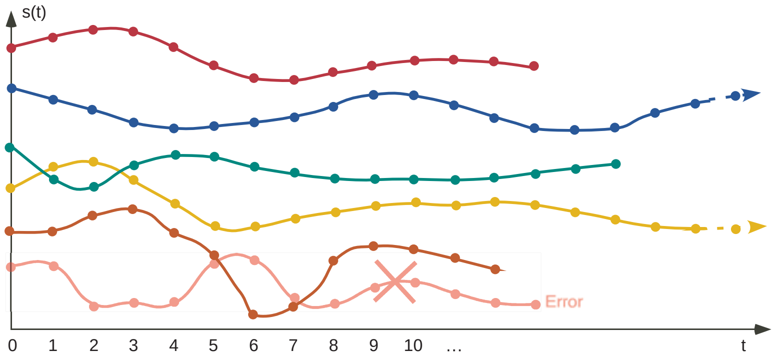
The Computational Lattice

Given $S, T \in \wp(\mathbb{T}^{\infty})$, we define

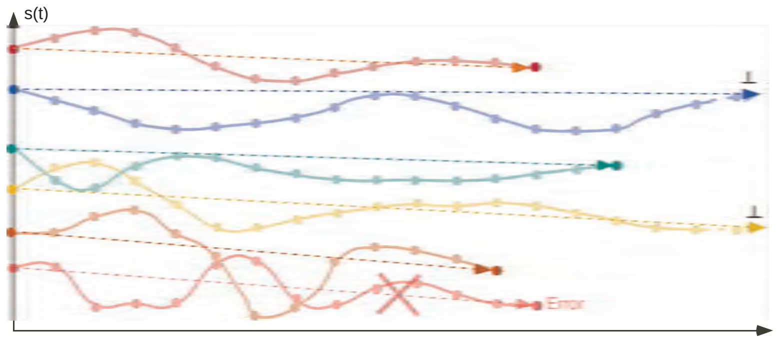
- $S^{+} \triangleq S \cap \mathbb{T}^{+}$ finite traces
- $S^{\omega} \triangleq S \cap \mathbb{T}^{\omega}$ infinite traces
- $S \sqsubseteq T \triangleq S^{+} \subseteq T^{+} \wedge S^{\omega} \supseteq T^{\omega}$ computational order
- $\langle \wp(\mathbb{T}^{\infty}),\ \sqsubseteq,\ \mathbb{T}^{\omega},\ \mathbb{T}^{+},\ \sqcup,\ \sqcap \rangle$ is a complete lattice

Relational Semantics

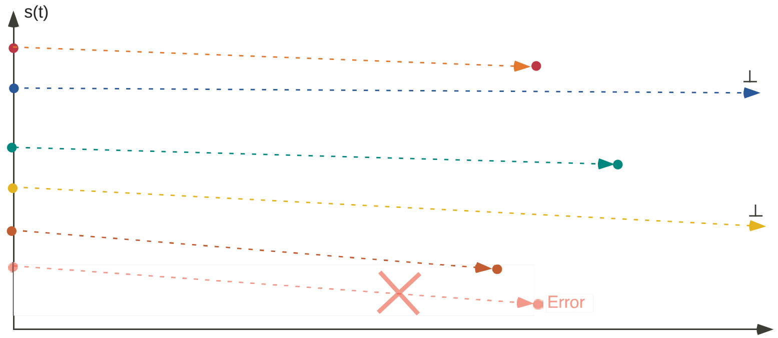
Trace Semantics



Relational Semantics = α(Trace Semantics)



Relational Semantics



Abstraction to the Bifinitary Relational Semantics of the Eager λ-calculus

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

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

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

$$\alpha(\sigma_0 \bullet \ldots \bullet \sigma_n \bullet \ldots) \stackrel{\text{def}}{=} \langle \sigma_0,\ \bot \rangle$$

Bifinitary Relational Semantics of the Eager λ-calculus

$$\mathsf{v} \Longrightarrow \mathsf{v}, \quad \mathsf{v} \in \mathbb{V}$$

$$\frac{\mathsf{a} \Longrightarrow \bot}{\mathsf{a}\ \mathsf{b} \Longrightarrow \bot} \sqsubseteq \qquad\qquad \frac{\mathsf{b} \Longrightarrow \bot}{\mathsf{a}\ \mathsf{b} \Longrightarrow \bot} \sqsubseteq, \quad \mathsf{a} \in \mathbb{V}$$

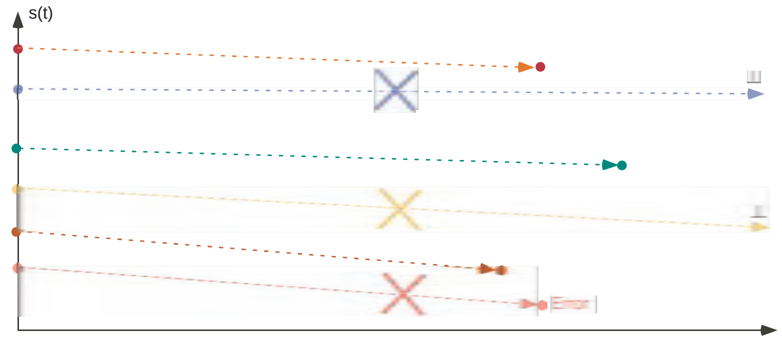
$$\frac{\mathsf{a}[\mathsf{x} \leftarrow \mathsf{v}] \Longrightarrow r}{(\boldsymbol{\lambda}\,\mathsf{x} \cdot \mathsf{a}) \quad \mathsf{v} \Longrightarrow r} \sqsubseteq, \quad \mathsf{v} \in \mathbb{V},\ r \in \mathbb{V} \cup \{\bot\}$$

$$\frac{\mathsf{a} \Longrightarrow \mathsf{v}, \quad \mathsf{v}\ \mathsf{b} \Longrightarrow r}{\mathsf{a}\ \mathsf{b} \Longrightarrow r} \sqsubseteq, \quad \mathsf{v} \in \mathbb{V},\ r \in \mathbb{V} \cup \{\bot\}$$

$$\frac{\mathsf{b} \Longrightarrow \mathsf{v}, \quad \mathsf{a}\ \mathsf{v} \Longrightarrow r}{\mathsf{a}\ \mathsf{b} \Longrightarrow r} \sqsubseteq, \quad \mathsf{a} \in \mathbb{V},\ \mathsf{v} \in \mathbb{V},\ r \in \mathbb{V} \cup \{\bot\}\,.$$

Natural Semantics

Natural Semantics = α(Relational Semantics)



Abstraction to the Natural Big-Step Semantics of the Eager λ-calculus

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

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

$$\alpha(\langle\sigma_0,\ \sigma_n\rangle) \stackrel{\text{def}}{=} \{\langle\sigma_0,\ \sigma_n\rangle\}$$

$$\alpha(\langle\sigma_0,\ \bot\rangle) \stackrel{\text{def}}{=} \varnothing$$

Natural Big-Step Semantics of the Eager λ-calculus [65]

$$\mathsf{v} \Longrightarrow \mathsf{v}, \quad \mathsf{v} \in \mathbb{V}$$

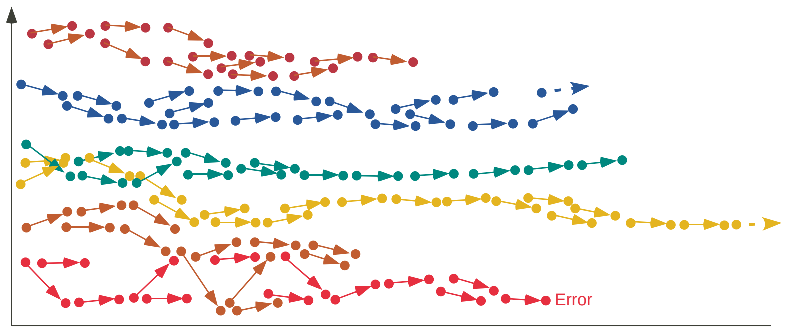
$$\frac{\mathsf{a}[\mathsf{x} \leftarrow \mathsf{v}] \Longrightarrow r}{(\boldsymbol{\lambda}\,\mathsf{x}\cdot\mathsf{a})\quad \mathsf{v} \Longrightarrow r} \subseteq, \quad \mathsf{v} \in \mathbb{V},\ r \in \mathbb{V}$$

$$\frac{\mathsf{a} \Longrightarrow \mathsf{v}, \quad \mathsf{v}\ \mathsf{b} \Longrightarrow r}{\mathsf{a}\ \mathsf{b} \Longrightarrow r} \subseteq, \quad \mathsf{v} \in \mathbb{V},\ r \in \mathbb{V}$$

$$\frac{\mathsf{b} \Longrightarrow \mathsf{v}, \quad \mathsf{a}\ \mathsf{v} \Longrightarrow r}{\mathsf{a}\ \mathsf{b} \Longrightarrow r} \subseteq, \quad \mathsf{a} \in \mathbb{V},\ \mathsf{v} \in \mathbb{V},\ r \in \mathbb{V}\ .$$

Transition Semantics

Transition Semantics = α(Trace Semantics)



Abstraction to the Transition Semantics of the Eager λ-calculus

remember execution steps,
forget about their sequencing

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

$$\alpha(\sigma_0 \bullet \sigma_1 \bullet \ldots \bullet \sigma_n) \stackrel{\text{def}}{=} \{\langle\sigma_i,\ \sigma_{i+1}\rangle \mid 0 \leqslant i \wedge i < n\}$$

$$\alpha(\sigma_0 \bullet \ldots \bullet \sigma_n \bullet \ldots) \stackrel{\text{def}}{=} \{\langle\sigma_i,\ \sigma_{i+1}\rangle \mid i \geqslant 0\}$$

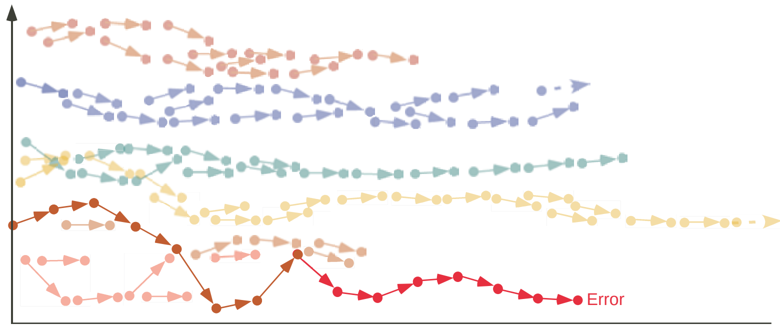
Transition Semantics of the Eager λ-calculus [69]

$$((\boldsymbol{\lambda}\,\mathsf{x} \cdot \mathsf{a})\ \mathsf{v}) \longrightarrow \mathsf{a}[\mathsf{x} \leftarrow \mathsf{v}]$$

$$\frac{\mathsf{a}_0 \longrightarrow \mathsf{a}_1}{\mathsf{a}_0\ \mathsf{b} \longrightarrow \mathsf{a}_1\ \mathsf{b}} \subseteq$$

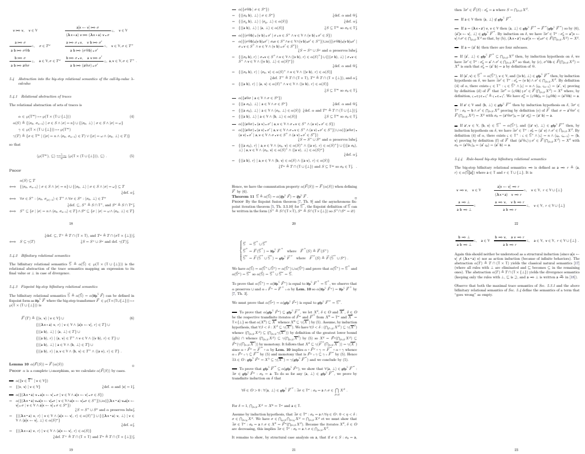
$$\frac{\mathsf{b}_0 \longrightarrow \mathsf{b}_1}{\mathsf{v}\ \mathsf{b}_0 \longrightarrow \mathsf{v}\ \mathsf{b}_1} \subseteq .$$

Approximation



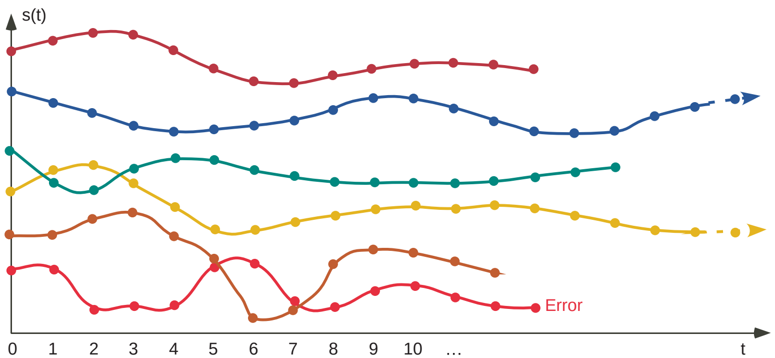
$((\boldsymbol{\lambda}\,\mathsf{x} \cdot \mathsf{x}\ \mathsf{x})\ ((\boldsymbol{\lambda}\,\mathsf{z} \cdot \mathsf{z})\ 0))\ (\boldsymbol{\lambda}\,\mathsf{y} \cdot \mathsf{y}) \to ((\boldsymbol{\lambda}\,\mathsf{x} \cdot \mathsf{x}\ \mathsf{x})\ 0)\ (\boldsymbol{\lambda}\,\mathsf{y} \cdot \mathsf{y})$
$\to (0\ 0)\ (\boldsymbol{\lambda}\,\mathsf{y} \cdot \mathsf{y})$ *an error!*

The Abstract Semantics are Correct by Calculational Design

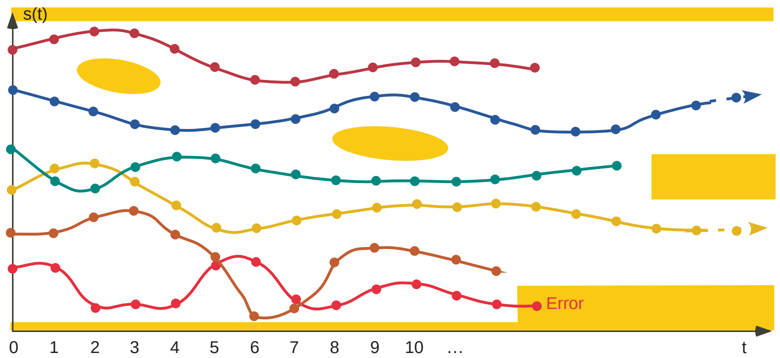


4. Principle of Static Analysis

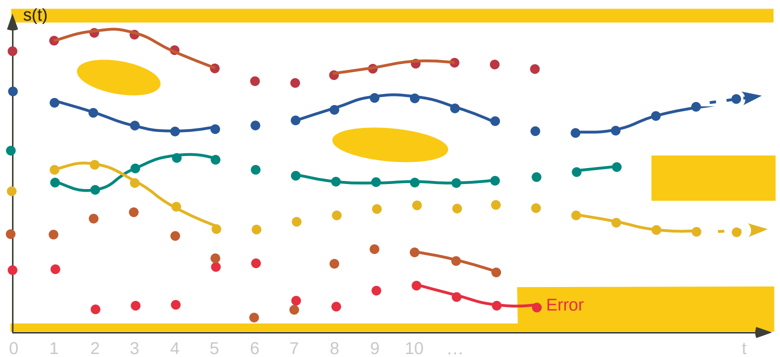
Principle of Static Analysis
(1) Concrete Semantics



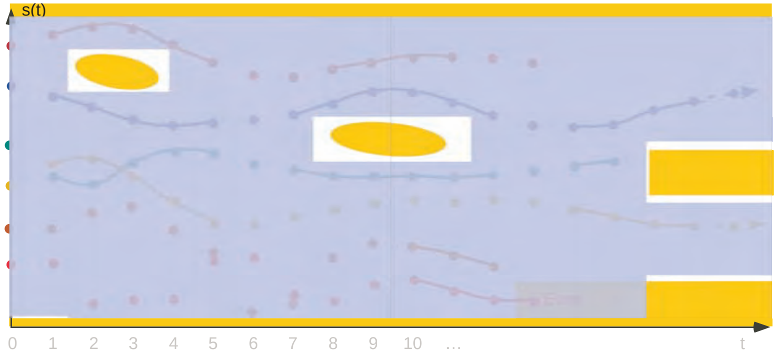
Principle of Static Analysis
(2) Specification



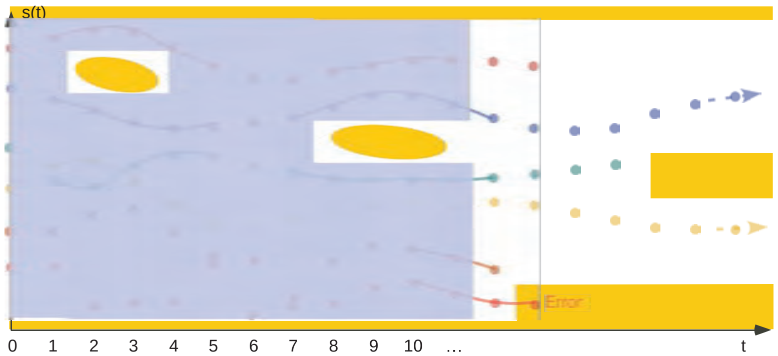
Principle of Static Analysis
(3.1) Abstract Semantics



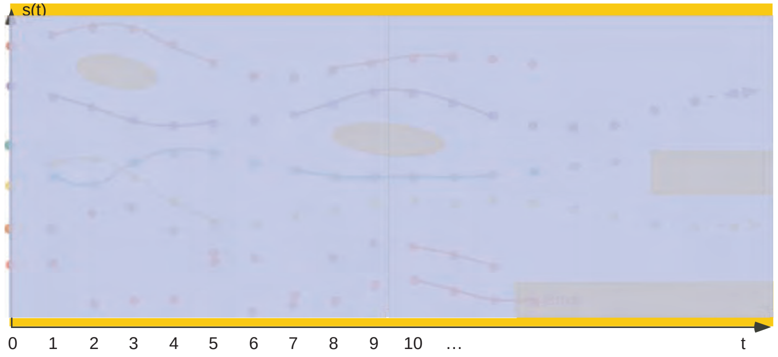
Principle of Static Analysis (3.2) Abstract Semantics



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 - (int) ((x-m)/(M-m))*(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 $- \epsilon$ (ϵ 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?

```
% cat unpredictable.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 I

```
% 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.

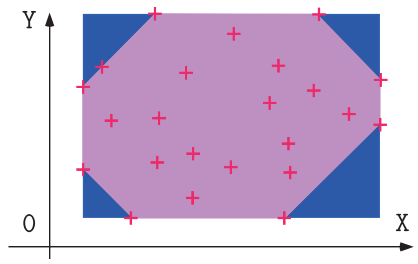
Abstraction

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:
$$\begin{cases} 1 \leq x \leq 9 \\ 1 \leq y \leq 20 \end{cases}$$

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

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

Termination

SLAM uses CEGAR and does not terminate[4] on

```
% 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

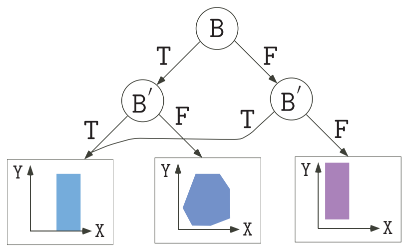
```
% astree -exec-fn main slam.c |& egrep "WARN"
%
```

[4] CEGAR cannot generate the invariant `y = x - 1` so produces all counter examples $x = i + 1 \wedge y = i$, $i = 0, 1, 2, 3, \ldots$

Boolean Relations for Boolean Control

– Code Sample:

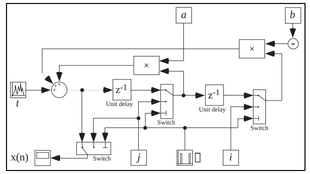
```
/* boolean.c */
typedef enum {F=0,T=1} BOOL;
BOOL B;
void main () {
  unsigned int X, Y;
  while (1) {
    ...
    B = (X == 0);
    ...
    if (!B) {
      Y = 1 / X;
    }
    ...
  }
}
```



The boolean relation abstract domain is parameterized by the height of the decision tree (an analyzer option) and the abstract domain at the leafs

Ellipsoid Abstract Domain for Filters

2^d Order Digital Filter:

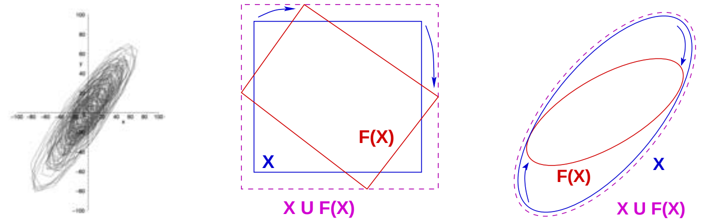


– Computes $X_n = \begin{cases} \alpha X_{n-1} + \beta X_{n-2} + Y_n \\ I_n \end{cases}$

– 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.



execution trace | unstable interval | stable ellipsoid

Filter Example [61]

```
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]

```
% 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;}
}
```

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


Incompleteness

ASTRÉE does not know that

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

so on the following program

```
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]
%
```

Example of directive (Cont'd)

```
% 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.

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: × 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

Industrial Application

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

Publications between 2002 and 2006 [7].

Invited Book Chapters

[2] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux and X. Rival. – Design and Implementation of a Special-Purpose Static Program Analyzer for Safety-Critical Real-Time Embedded Software, invited chapter. *In: The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, edited by T. Mogensen, D. Schmidt and I. Sudborough, pp. 85–108. – Springer, Berlin, Germany, 2002, *Lecture Notes in Computer Science 2566*.

[3] P. Cousot. – Verification by Abstract Interpretation, invited chapter. *In: Proceedings of the International Symposium on Verification – Theory & Practice – Honoring Zohar Manna's 64th Birthday*, edited by N. Dershowitz, pp. 243–268. – Taormina, Italy, Lecture Notes in Computer Science 2772, Springer, Berlin, Germany, 29 June – 4 July 2003.

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

[4] P. Cousot and R. Cousot. – Basic Concepts of Abstract Interpretation, invited chapter. *In: Building the Information Society*, edited by P. Jacquart, Chapter 4, pp. 359–366. – Kluwer Academic Publishers, Dordrecht, The Netherlands, 2004.

Refereed Journal Publications

[5] P. Cousot. – Constructive Design of a Hierarchy of Semantics of a Transition System by Abstract Interpretation. *Theoretical Computer Science*, Vol. 277, nº 1—2, 2002, pp. 47–103.

[6] P. Cousot and R. Cousot. – Parsing as Abstract Interpretation of Grammar Semantics. *Theoretical Computer Science*, Vol. 290, nº 1, January 2003, pp. 531–544.

Invited Conference or Workshop Proceedings Publications

[7] P. Cousot and R. Cousot. – Modular Static Program Analysis, invited paper. *In: Proceedings of the Eleventh International Conference on Compiler Construction, CC '2002*, edited by R. Horspool, Grenoble, France, 6–14 April 2002. pp. 159–178. – Lecture Notes in Computer Science 2304, Springer, Berlin, Germany.

[8] P. Cousot and R. Cousot. – On Abstraction in Software Verification, invited paper. *In: Proceedings of the Fourteenth International Conference on Computer Aided Verification, CAV '2002*, edited by E. Brinksma and K. Larsen. *Copenhagen, Denmark, Lecture Notes in Computer Science 2404*, pp. 37–56. – Springer, Berlin, Germany, 27–31 July 2002.

[9] P. Cousot. – Proving Program Invariance and Termination by Parametric Abstraction, Lagrangian Relaxation and Semidefinite Programming, invited paper. *In: Proceedings of the Sixth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI 2005)*, edited by R. Cousot, Paris, France, 17–19 January 2005. pp. 1–24. – Lecture Notes in Computer Science 3385, Springer, Berlin, Germany.

[10] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux and X. Rival. – Combination of Abstractions in the ASTRÉE Static Analyzer, invited paper. *In: Eleventh Annual Asian Computing Science Conference, ASIAN 06*, edited by M. Okada and I. Satoh, Tokyo, Japan, 6–8 December 2006. – Lecture Notes in Computer Science , Springer, Berlin, Germany. To appear.

[11] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux and X. Rival. – Varieties of Static Analyzers: A Comparison with ASTRÉE, invited paper. *In : First IEEE & IFIP International Symposium on Theoretical Aspects of Software Engineering, TASE '07*, edited by M. Hinchey and H. J. J. Sanders, Shanghai, China, 6–8 June 2007. – IEEE Computer Society Press, Los Alamitos, California, United States. To appear.

Refereed Conference or Workshop Proceedings Publications

[12] P. Cousot and R. Cousot. – Systematic Design of Program Transformation Frameworks by Abstract Interrpetation. *In : Conference Record of the Twentyninth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Portland, Oregon, United States, January 2002. pp. 178–190. – ACM Press, New York, New York, United States.

[13] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux and X. Rival. – A Static Analyzer for Large Safety-Critical Software. *In : Proceedings of the ACM SIGPLAN '2003 Conference on Programming Language Design and Implementation (PLDI)*, San Diego, California, United States, 7–14 June 2003. pp. 196–207. – ACM Press, New York, New York, United States.

[14] P. Cousot and R. Cousot. – An Abstract Interpretation-Based Framework for Software Watermarking. *In : Conference Record of the Thirtyfirst Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Venice, Italy, 14–16 January 2004. pp. 173–185. – ACM Press, New York, New York, United States.

[15] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux and X. Rival. – The ASTRÉE analyser. *In : Proceedings of the Fourteenth European Symposium on Programming Languages and Systems, ESOP '2005, Edinburg, Scotland*, edited by M. Sagiv, pp. 21–30. – Springer, Berlin, Germany, 2–10 April 2005, *Lecture Notes in Computer Science*, Vol. 3444.

Recent Software

[16] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux and X. Rival. – The ASTRÉE Static Analyzer. – http://www.astree.ens.fr/.

[17] P. Cousot. – ANAA: The abstract interpretation-based software watermarker, June 2003.

Patents

[18] P. Cousot, M. Riguidel and A. Venet. – Dispositif et procédé pour la signature, le marquage et l'authentification de programmes d'ordinateur (in French). – November 2003. Reference WO 02/091141.

Invited Conference Lectures and Tutorials

[19] P. Cousot. – Abstract Interpretation Software Technologies, invited talk. *In : Workshop on Software Technologies, Embedded Systems and Distributed Systems in the sixth Framework Programme, TESSS*, European Commission, Brussels, Belgium, 2 May 2002.

[20] P. Cousot. – Abstract Interpretation: Theory and Practice, invited speaker. *In : Proceedings of the Ninth International Workshop on Model Checking of Software, SPIN '2002*, edited by D. Bosnacki and S. Leue, Copenhagen, Denmark, 27–31 July 2002. *Lecture Notes in Computer Science 2318*, pp. 2–5. – Springer, Berlin, Germany.

[21] P. Cousot. – Abstract Interpretation: Theory and Practice, invited speaker. *In : European Joint Conferences on Theory and Practice of Software (ETAPS'02)*, Grenoble, France, 8–12 April 2002.

[22] P. Cousot. – On Abstraction in Software Verification, invited tutorial. *In : Fourteenth International Conference on Computer Aided Verification, CAV '2002*, Copenhagen, Denmark, 27–31 July 2002.

[23] P. Cousot and R. Cousot. – Abstract Interpretation: A Theory of Approximation, invited talk. *In : Special session on Abstract Interpretation, Eightteenth Workshop on the Mathematical Foundations of Programming Semantics (MFPS'02), Tulane University*, New Orleans, Louisiana, United States, 23–26 March 2002.

[24] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux and X. Rival. – ASTRÉE: A Static Analyzer for Large Safety-Critical Software. *In : Schloß Dagstuhl Seminar 3451 on "Applied Deductive Verification"*, Schloß Dagstuhl, Wadern, Germany, 2–7 November 2003.

[25] P. Cousot. – Automatic Verification by Abstract Interpretation, invited tutorial. *In : Proceedings of the Fourth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI 2003)*, edited by L. Zuck, P. Attie, A. Cortesi and S. Mukhopadhyay, Courant Institute, NYU, New York, New York, United States, 9–11 January 2003. pp. 20–24. – Lecture Notes in Computer Science 2575, Springer, Berlin, Germany.

[26] P. Cousot. – A Static Analyzer for Large Safety-Critical Software, invited talk. *In : Italian CoVer (Constraint-based Verification of Reactive systems) project meeting*, Florence, Italy, 25–26 september 2003.

[27] P. Cousot. – Abstract Interpretation of Computations. *In : Workshop on Robustness, Abstractions and Computations*, University of Pennsylvania, Philadelphia, United States, 28 March 2004.

[28] P. Cousot. – Automated Verification of Infinite-State Systems by Abstract Interpretation, invited talk. *In : Third International Workshop on Automated Verification of Infinite-State Systems (AVIS'04)*, Barcelona, Spain, 3–4 April 2004.

[29] P. Cousot. – Grand Challenges for Abstract Interpretation. *In : Second Workshop on Dependable Systems Evolution*, T. Hoare, P. O'Hearn, . Thimbleby & J. Woodcock (Organizers), Gresham College, London, United Kingdom, 18 March 2004.

[30] P. Cousot. – A Lagrangian relaxation and mathematical programming framework for static analysis and verification, invited talk. *In : International Symposium on Static Analysis, SAS'04 & on Logic Program Synthesis and Transformation, LOPSTR'04*, Verona, Italy, 28 August 2004.

[31] P. Cousot. – Software Verification by Abstract Interpretation: Current Trends and Perspectives, invited talk. *In : IV Jornadas de Programación y Lenguajes*, Málaga, Spain, 11–12 November 2004.

[32] P. Cousot. – Abstract Interpretation-based Formal Verification of Complex Computer Systems. *In : Minta Martin Lecture*, Department of Aeronautics and Astronautics, MIT, Cambridge, Massachusetts, United States, 13 May 2005.

[33] P. Cousot. – Automatic Verification of Embedded Control Software with ASTRÉE. *In : Workshop on Critical Research Areas in Aerospace Software*, MIT, Cambridge, Massachusetts, United States, 9 August 2005.

[34] P. Cousot. – Challenges in Abstract Interpretation for Software Safety. *In : French-Japanese symposium on computer security*, Keio University, Mita Campus, Global Security Research Institute, Tokyo, Japan, 5–7 september 2005.

[35] P. Cousot. – Integrating Physical Systems in the Static Analysis of Embedded Control Software, invited paper. *In : Proceedings of the Third Asian Symposium on Programming Languages and Systems, APLAS'2005*, Tsukuba, Japan, 3–5 November 2005. pp. 135–138. – Lecture Notes in Computer Science 3780, Springer, Berlin, Germany.

[36] P. Cousot. – Parametric Abstraction. *In : First International Workshop on Numerical & Symbolic Abstract Domains, NSAD'05*, Maison Des Polytechniciens, Paris, France, 21 January 2005.

[37] P. Cousot. – A Tutorial on Abstract Interpretation. *In : Industrial day on Automatic Tools for Program Verification, International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI 2005)*, Maison Des Polytechniciens, Paris, France, 20 January 2005.

[38] P. Cousot. – The Verification Grand Challenge and Abstract Interpretation. *In : Verified Software: Theories, Tools, Experiments (VSTTE)*, ETH Zürich, Switzerland, 10–13 October 2005.

[39] P. Cousot. – Formalizations of Abstraction in the Abstract Interpretation Theory. *In : The Challenge of Software Verification*, Dagstuhl Seminar 6281, Schloß Dagstuhl, Wadern, Germany, 9–13 July 2006.

[40] P. Cousot. – Program Verification by Parametric Abstraction and Semi-definite Programming, invited talk. *In : Logic and Algorithms Workshop "Constraints and Verification"*, Isaac Newton Institute for Mathematical Sciences, Cambridge, United Kingdom, 8–12 May 2006.

[41] P. Cousot. – The Scientific Work of Reinhard Wilhelm. *In : Special event to honour the 60th birthday of Prof. Reinhard Wilhelm*, Universität Saarbrücken, Germany, 10 June 2006.

[42] P. Cousot. – Verification of Large Complex Software by Abstract Interpretation, invited talk. *In : Eleventh Annual Asian Computing Science Conference, ASIAN 06*, National Center of Sciences, Tokyo, Japan, 6–8 December 2006.

[43] P. Cousot and R. Cousot. – Grammar Abstract Interpretation. *In : Seminar in Honor of Reinhard Wilhelm's 60th Birthday*, Dagstuhl Seminar 6232, Schloß Dagstuhl, Wadern, Germany, 9–10 June 2006.

Recent Invited Seminar Presentations

[44] P. Cousot. – Abstract Interpretation & Applications. *In :* *AA & EECS Seminar*, MIT, Cambridge, Massachusetts, United States, 3 April 2006.

[45] P. Cousot. – Application of Abstract Interpretation to the Static Verification of Safety Critical Code. *In :* *Seminar, IBM Thomas J. Watson Research Center*, Hawthorne, New York, United States, 20 January 2006.

[46] P. Cousot. – Interprétation abstraite : application aux logiciels de l'A380. *In :* *Exposé sur des questions d'actualité*, Académie des Sciences, Paris, France, 6 June 2006.

[47] P. Cousot. – Program Termination Proofs by Parametric Abstraction, Lagrangian Relaxation and Semi-Definite Programming. *In :* *Specialised Talk, Seminar Series*, Department of Computing and Information Sciences, Kansas State University, Manhattan, Kansas, United States, 6 september 2006.

[48] P. Cousot. – Static Verification of Safety Critical Code by Abstract Interpretation. *In :* *Distinguished Lecturer Series*, Department of Computing and Information Sciences, Kansas State University, Manhattan, Kansas, United States, 5 september 2006.

[49] P. Cousot and R. Cousot. – Abstract interpretation and a range of applications. *In :* *Seminario del Dipartimento di Informatica*, Università Ca' Foscari Venezia, Mestre, Italy, 23 October 2006.

9. Other References

[50] G. Amato, F. Giannotti and G. Mainetto. – Data sharing analysis for a database programming language via abstract interpretation. *In : Proceedings of the Ninthteenth International Conference on Very Large Data Bases*, edited by R. Agrawal, S. Baker and D.A.Bell, Dublin, Ireland, 24–27 August 1993. pp. 405–415. – MORGANKAUFMANN.

[51] J. Bailey, A. Poulovassilis and C. Courtenage. – Optimising active database rules by partial evaluation and abstract interpretation. *In : Proceedings of the Eight International Workshop on Database Programming Languages*, Frascati, Italy, 8–10 september 2001. *Lecture Notes in Computer Science 2397*, pp. 300–317. – Springer, Berlin, Germany.

[52] V. Benzaken and X. Schaefer. – Static Integrity Constraint Management in Object-Oriented Database Programming Languages via Predicate Transformers. *In : Proceedings of the Eleventh European Conference on Object-Oriented Programming, ECOOP '97*, edited by M. Aksit and S. Matsuoka. – Jyväskylä, Finland, Springer, Berlin, Germany, 9–13 June 1997, *Lecture Notes in Computer Science 1241*.

[53] P. Cousot. – Types as Abstract Interpretations, invited paper. *In : Conference Record of the Twentyfourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Paris, France, January 1997. pp. 316–331. – ACM Press, New York, New York, United States.

[54] P. Cousot and R. Cousot. – Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. *In : Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Los Angeles, California, 1977. pp. 238–252. – ACM Press, New York, New York, United States.

[55] P. Cousot and R. Cousot. – Systematic design of program analysis frameworks. *In : Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, San Antonio, Texas, 1979. pp. 269–282. – ACM Press, New York, New York, United States.

[56] P. Cousot and R. Cousot. – Inductive Definitions, Semantics and Abstract Interpretation. *In : Conference Record of the Ninthteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Albuquerque, New Mexico, United States, 1992. pp. 83–94. – ACM Press, New York, New York, United States.

[57] P. Cousot and R. Cousot. – Formal Language, Grammar and Set-Constraint-Based Program Analysis by Abstract Interpretation. *In : Proceedings of the Seventh ACM Conference on Functional Programming Languages and Computer Architecture*, La Jolla, California, United States, 25–28 June 1995. pp. 170–181. – ACM Press, New York, New York, United States.

[58] P. Cousot and R. Cousot. – Temporal Abstract Interpretation. *In :* *Conference Record of the Twentyseventh Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Boston, Massachusetts, United States, January 2000. pp. 12–25. – ACM Press, New York, New York, United States.

[59] P. Cousot and N. Halbwachs. – Automatic discovery of linear restraints among variables of a program. *In :* *Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Tucson, Arizona, 1978. pp. 84–97. – ACM Press, New York, New York, United States.

[60] V. Danos. – Abstract views on biological signalling. *In :* *Mathematical Foundations of Programming Semantics, Twentythird Annual Conference (MFPS XXIII).* – 2007.

[61] J. Feret. – Static Analysis of Digital Filters. *In :* *Proceedings of the Thirteenth European Symposium on Programming Languages and Systems, ESOP '2004, Barcelona, Spain*, edited by D. Schmidt. *Lecture Notes in Computer Science*, Vol. 2986, pp. 33–48. – Springer, Berlin, Germany, March 27 – April 4, 2004.

[62] J. Feret. – The Arithmetic-Geometric Progression Abstract Domain. *In :* *Proceedings of the Sixth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI 2005)*, edited by R. Cousot, Paris, France, 17–19 January 2005. pp. 42–58. – Lecture Notes in Computer Science 3385, Springer, Berlin, Germany.

[63] R. Giacobazzi and I. Mastroeni. – Abstract non-interference: Parameterizing non-interference by abstract interpretation. *In :* *Conference Record of the Thirtyfirst Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Venice, Italy, 2004. pp. 186–197. – ACM Press, New York, New York, United States.

[64] P. Jorrand and S. Perdrix. – Towards a quantum calculus. *In :* *Proceedings of the Fourth International Workshop on Quantum Programming Languages, ENTCS.* – 2006.

[65] G. Kahn. – Natural semantics. *In :* *Programming of Future Generation Computers*, edited by K. Fuchi and M. Nivat, pp. 237–258. – Elsevier Science Publishers B.V., Amsterdam, The Netherlands, 1988.

[66] A. Miné. – A New Numerical Abstract Domain Based on Difference-Bound Matrices. *In :* *Proceedings of the Second Symposium PADO '2001, Programs as Data Objects*, edited by . Danvy and A. Filinski. *Århus, Denmark, 21–23 May 2001, Lecture Notes in Computer Science 2053*, pp. 155–172. – Springer, Berlin, Germany, 2001.

[67] A. Miné. – Relational Abstract Domains for the Detection of Floating-Point Run-Time Errors. *In :* *Proceedings of the Thirteenth European Symposium on Programming Languages and Systems, ESOP '2004, Barcelona, Spain*, edited by D. Schmidt. *Lecture Notes in Computer Science*, Vol. 2986, pp. 3–17. – Springer, Berlin, Germany, March 27 – April 4, 2004.

[68] S. Perdrix. – *Modèles formels du calcul quantique : ressources, machines abstraites et calcul par mesure.* – ThËse, Institut National Polytechnique de Grenoble, Laboratoire Leibniz, 2006.

[69] G. Plotkin. – *A structural Approach to Operational Semantics.* – Technical Report nº DAIMI FN-19, Aarhus University, Denmark, september 1981.

[70] M. D. Preda, M. Christodorescu, S. Jha and S. Debray. – A Semantics-Based Approach to Malware Detection. *In :* *Conference Record of the Thirtyfourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Nice, France, 2007. – ACM Press, New York, New York, United States.

[71] F. Ranzato and F. Tapparo. – Strong Preservation as Completeness in Abstract Interpretation. *In :* *Proceedings of the Thirteenth European Symposium on Programming Languages and Systems, ESOP '04*, edited by D. Schmidt, Barcelona, Spain, March 29 – April 2 2004. *Lecture Notes in Computer Science*, Vol. 2986, pp. 18–32. – Springer, Berlin, Germany.

10. Annex

- $a = (\boldsymbol{\lambda} y \cdot y)$
- $\sigma = ((\boldsymbol{\lambda} z \cdot z)\ 0) \bullet 0$
- $a@\sigma =$

  $(\boldsymbol{\lambda} y \cdot y)@((\boldsymbol{\lambda} z \cdot z)\ 0) \bullet 0 =$

  $((\boldsymbol{\lambda} y \cdot y)\ ((\boldsymbol{\lambda} z \cdot z)\ 0)) \bullet ((\boldsymbol{\lambda} y \cdot y)\ 0)$

- $\sigma = ((\boldsymbol{\lambda} x \cdot x\ x)\ (\boldsymbol{\lambda} y \cdot y)) \bullet ((\boldsymbol{\lambda} y \cdot y)\ (\boldsymbol{\lambda} y \cdot y)) \bullet (\boldsymbol{\lambda} y \cdot y)$
- $b = ((\boldsymbol{\lambda} z \cdot z)\ 0)$
- $(\sigma@b)$

  $=$

  $(((\boldsymbol{\lambda} x \cdot x\ x)\ (\boldsymbol{\lambda} y \cdot y)) \bullet ((\boldsymbol{\lambda} y \cdot y)\ (\boldsymbol{\lambda} y \cdot y)) \bullet (\boldsymbol{\lambda} y \cdot y)@((\boldsymbol{\lambda} z \cdot z)\ 0))$

  $=$

  $((((\boldsymbol{\lambda} x \cdot x\ x)\ (\boldsymbol{\lambda} y \cdot y))\ ((\boldsymbol{\lambda} z \cdot z)\ 0)) \bullet (((\boldsymbol{\lambda} y \cdot y)\ (\boldsymbol{\lambda} y \cdot y))\ ((\boldsymbol{\lambda} z \cdot z)\ 0)) \bullet ((\boldsymbol{\lambda} y \cdot y)\ ((\boldsymbol{\lambda} z \cdot z)\ 0))$

$$\boxed{\frac{\sigma \bullet v \in \vec{\mathbb{S}}^+,\ (a\ v) \bullet \sigma' \in \vec{\mathbb{S}}}{(a@\sigma) \bullet (a\ v) \bullet \sigma' \in \vec{\mathbb{S}}} \sqsubseteq, \quad v, a \in \mathbb{V}\ .}$$

- $\sigma \bullet v = ((\boldsymbol{\lambda} z \cdot z)\ 0) \bullet 0 \in \in \vec{\mathbb{S}}^+$
- $(a\ v) \bullet \sigma' = (\boldsymbol{\lambda} y \cdot y)\ 0 \bullet 0 \in \vec{\mathbb{S}}$
- $(a@\sigma) \bullet (a\ v) \bullet \sigma'$

  $=$

  $((\boldsymbol{\lambda} y \cdot y)@((\boldsymbol{\lambda} z \cdot z)\ 0) \bullet\ 0) \bullet 0$

  $=$

  $(\boldsymbol{\lambda} y \cdot y)\ ((\boldsymbol{\lambda} z \cdot z)\ 0) \bullet (\boldsymbol{\lambda} y \cdot y)\ 0 \bullet 0 \in \vec{\mathbb{S}}$

$$\boxed{\frac{\sigma \bullet v \in \vec{\mathbb{S}}^+,\ (v\ b) \bullet \sigma' \in \vec{\mathbb{S}}}{(\sigma@b) \bullet (v\ b) \bullet \sigma' \in \vec{\mathbb{S}}} \sqsubseteq, \quad v \in \mathbb{V}}$$

- $\sigma \bullet v = ((\boldsymbol{\lambda} x \cdot x\ x)\ (\boldsymbol{\lambda} y \cdot y)) \bullet ((\boldsymbol{\lambda} y \cdot y)\ (\boldsymbol{\lambda} y \cdot y)) \bullet (\boldsymbol{\lambda} y \cdot y) \in \vec{\mathbb{S}}^+$
- $(v\ b) \bullet \sigma' = (\boldsymbol{\lambda} y \cdot y)\ ((\boldsymbol{\lambda} z \cdot z)\ 0) \bullet (\boldsymbol{\lambda} y \cdot y)\ 0 \bullet 0 \in \vec{\mathbb{S}}$
- $(\sigma@b) \bullet (v\ b) \bullet \sigma'$

  $=$

  $(((\boldsymbol{\lambda} x \cdot x\ x)\ (\boldsymbol{\lambda} y \cdot y)) \bullet ((\boldsymbol{\lambda} y \cdot y)\ (\boldsymbol{\lambda} y \cdot y))@((\boldsymbol{\lambda} z \cdot z)\ 0)) \bullet$
  $((\boldsymbol{\lambda} y \cdot y)\ ((\boldsymbol{\lambda} z \cdot z)\ 0)) \bullet (\boldsymbol{\lambda} y \cdot y)\ 0 \bullet 0$

  $=$

  $((\boldsymbol{\lambda} x \cdot x\ x)\ (\boldsymbol{\lambda} y \cdot y))\ ((\boldsymbol{\lambda} z \cdot z)\ 0) \bullet ((\boldsymbol{\lambda} y \cdot y)\ (\boldsymbol{\lambda} y \cdot y))\ ((\boldsymbol{\lambda} z \cdot z)\ 0)$
  $\bullet\ (\boldsymbol{\lambda} y \cdot y)\ ((\boldsymbol{\lambda} z \cdot z)\ 0) \bullet (\boldsymbol{\lambda} y \cdot y)\ 0 \bullet 0 \in \vec{\mathbb{S}}$