Program and Model Analysis
(Graduiertenkolleg Programm- Und Modell-Analyse)

Abstraction and applications

Patrick Cousot

pcousot@cs.nyu.edu
http://cs.nyu.edu/~pcousot

cousot@ens.fr
http://www.di.ens.fr/~cousot

Technische Universität München
Fakultät für Informatik

May 29, 2009

Next time ?

- More formal on foundations ?, or
- More on the notion of abstraction ?, or
- More practical on applications (like ASTRÉE)

Your choice ?

⇒ a little on the notion of abstraction

⇒ a little on applications (like ASTRÉE)

⇒ nothing formal on foundations!

Abstraction, explained by examples

Trace semantics

- (Infinite) set of maximal (finite/infinite) traces (sequences of states)

$$P \triangleq {}^{1}\texttt{x} := 1 \texttt{ ; while } {}^{2}\texttt{true do } {}^{3}\texttt{x} := (\texttt{x} + 1)\texttt{; od}^{4}.$$

- Trace semantics:

$$\{\langle {}^{1}, z\rangle\langle {}^{2}, 1\rangle\langle {}^{3}, 1\rangle\langle {}^{2}, 2\rangle\langle {}^{3}, 2\rangle \ldots \langle {}^{2}, i\rangle\langle {}^{3}, i\rangle\langle {}^{2}, i+1\rangle \ldots \mid z \in \mathbb{Z}\}$$

Safety semantics

- Set of prefixes of the maximal traces
- Termination (liveness) properties are lost

$$P \triangleq {}^{1}\texttt{x} := 1 \ ; \ \texttt{while}\ {}^{2}\texttt{true}\ \texttt{do}\ {}^{3}\texttt{x} := (\texttt{x} + 1); \ \texttt{od}^{4}.$$

- Safety semantics:

$$\{\langle^1, z\rangle\langle^2, 1\rangle\langle^3, 1\rangle\langle^2, 2\rangle\langle^3, 2\rangle \ldots \langle^2, i\rangle \mid i \in \mathbb{N} \wedge z \in \mathbb{Z}\} \cup$$
$$\{\langle^1, z\rangle\langle^2, 1\rangle\langle^3, 1\rangle\langle^2, 2\rangle\langle^3, 2\rangle \ldots \langle^2, i\rangle\langle^3, i\rangle \mid i \in \mathbb{N} \wedge z \in \mathbb{Z}\}$$

Safety abstraction

$$\alpha(S) = \{\sigma \mid \exists\ \sigma' : \sigma\sigma' \in S\}$$

Invariance semantics

- Set of states along prefix traces
- The order of appearance of states during execution is lost

$$P \triangleq {}^{1}\texttt{x} := 1 \ ; \ \texttt{while}\ {}^{2}\texttt{true}\ \texttt{do}\ {}^{3}\texttt{x} := (\texttt{x} + 1); \ \texttt{od}^{4}.$$

- Invariance semantics:

$$\{<1,z> \mid z \in \mathbb{Z}\} \ \cup \ \{<2,z> \mid z>0\} \ \cup \ \{<3,z> \mid z>0\}$$

Invariance abstraction

$$\alpha(S) = \{s \mid \exists\ \sigma,\sigma' : \sigma s \sigma' \in S\}$$

Local invariance semantics

- Set of memory states attached to each program point
- Isomorphism (no lost of information)

$P \triangleq {}^{1}\texttt{x := 1 ; while }{}^{2}\texttt{true do }{}^{3}\texttt{x := (x + 1); od}{}^{4}.$

- Local invariance semantics:

$$\begin{aligned} I_1 &= \mathbb{Z} \\ I_2 &= \{z \in \mathbb{Z} \mid z > 0\} \\ I_3 &= \{z \in \mathbb{Z} \mid z > 0\} \\ I_4 &= \emptyset \end{aligned}$$

Local invariance abstraction

$$\alpha(S) = \lambda i \bullet \{m \mid \langle i,m \rangle \in S\}$$

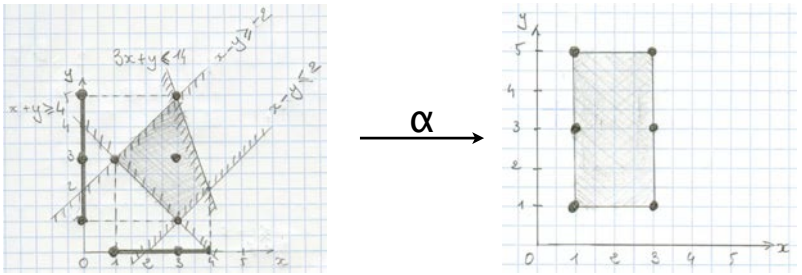
$$\in \quad (\mathbb{L} \mapsto \wp(\mathcal{E}))$$

$\mathbb{L}$: set of program points

$m \in \mathcal{E} \triangleq \mathbb{V} \longrightarrow \mathcal{V}$ maps variables to values

Cartesian abstract semantics

- Set of possible values of each variable attached to each program point
- Relations between variables are lost (if ≥ 2 variables!)



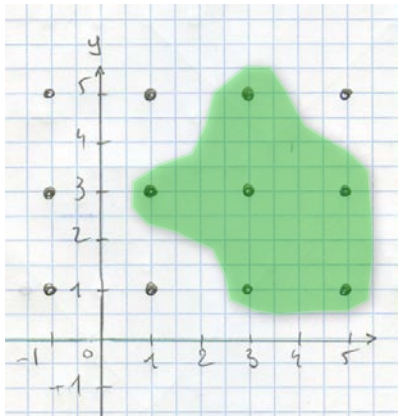
Cartesian abstraction

$$\alpha(S) = \lambda x \bullet \{m(x) \mid m \in S(i)\}$$

$$\in \quad (\mathbb{L} \mapsto \wp(\mathcal{E})) \mapsto (\mathbb{L} \mapsto (\mathbb{V} \mapsto \wp(\mathcal{V})))$$

Parity abstract semantics

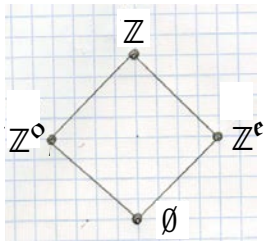
- Parity of each variable attached to each program point
- Possible values of variables are lost



Parity abstraction

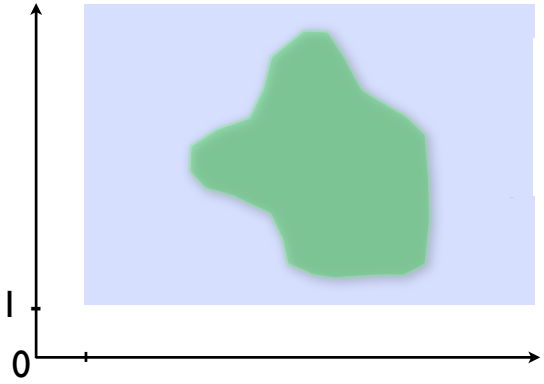
$$
\begin{array}{rcl}
\mathfrak{L}^p & \triangleq & \{\cup S \mid S \subseteq \{\mathbb{Z}^o, \mathbb{Z}^e\}\} \\
 & = & \{\emptyset, \mathbb{Z}^o, \mathbb{Z}^e, \mathbb{Z}\} \\
\alpha^p & \in & (\mathbb{L} \mapsto (\mathbb{V} \mapsto \wp(\mathbb{Z}))) \mapsto (\mathbb{L} \mapsto (\mathbb{V} \mapsto \mathfrak{L}^p)) \\
\alpha^p(C)(\mathrm{x}) & \triangleq & \{\mathbb{Z}^o \mid C(x) \cap \mathbb{Z}^o \neq \emptyset\} \cup \{\mathbb{Z}^e \mid C(x) \cap \mathbb{Z}^e \neq \emptyset\}
\end{array}
$$

$\mathfrak{L}^p \triangleq$



Sign abstract semantics

- Sign of each variable attached to each program point
- Possible values of variables are lost



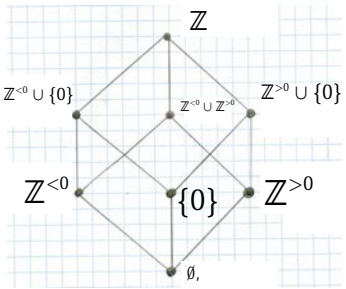

```
while [1](1 < x) do
  [2]x := x - 1
od[3].
```

$\{\langle 1, \langle \mathrm{x}, \mathbb{Z} \rangle\rangle\langle 2, \langle \mathrm{x}, \mathbb{Z}^{>0} \rangle\rangle\langle 3, \langle \mathrm{x}, \mathbb{Z} \rangle\rangle\}$

Sign abstraction

$$
\begin{array}{rcl}
\mathfrak{L}^s & \triangleq & \{\cup S \mid S \subseteq \{\mathbb{Z}^{<0}, \{0\}, \mathbb{Z}^{>0}\}\} \\
 & = & \{\emptyset, \mathbb{Z}^{<0}, \mathbb{Z}^{>0}, \mathbb{Z}^{<0} \cup \{0\}, \mathbb{Z}^{<0} \cup \mathbb{Z}^{>0}, \mathbb{Z}^{>0} \cup \{0\}, \mathbb{Z}\} \\
\alpha^s & \in & (\mathbb{L} \mapsto (\mathbb{V} \mapsto \wp(\mathbb{Z}))) \mapsto (\mathbb{L} \mapsto (\mathbb{V} \mapsto \mathfrak{L}^s)) \\
\alpha^s[C](\ell)\mathrm{x} & \triangleq & \{\mathbb{Z}^{<0} \mid C(\ell)\mathrm{x} \cap \mathbb{Z}^{<0} \neq \emptyset\} \cup \{\{0\} \mid 0 \in C(\ell)\mathrm{x}\} \cup \\
 & & \{\mathbb{Z}^{>0} \mid \mathbb{Z}^{>0} \cap C(\ell)\mathrm{x} \neq \emptyset\}
\end{array}
$$

$\mathfrak{L}^s \triangleq$

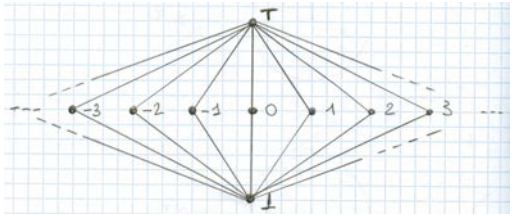


Constant abstract semantics

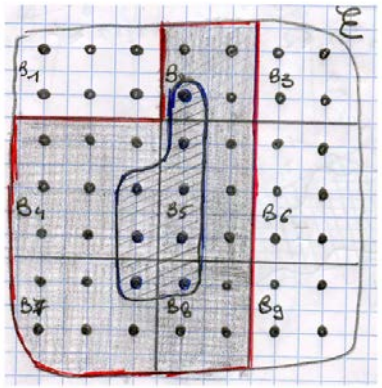
- Record values of variables attached to each program point if and when only one
- Possible values of variables are lost when not constant

Constant abstraction

$$
\begin{array}{rcll}
\mathfrak{L}^c & \triangleq & \{\emptyset, \mathcal{V}\} \cup \{\{v\} \mid v \in \mathcal{V}\} & \\
\alpha^c & \in & (\mathbb{L} \mapsto (\mathbb{V} \mapsto \wp(\mathcal{V}))) \mapsto (\mathbb{L} \mapsto (\mathbb{V} \mapsto \mathfrak{L}^c)) & \\
\alpha^c[C](\ell)\mathrm{x} & \triangleq & \emptyset & \text{if } C(\ell)\mathrm{x} = \emptyset \\
\alpha^c[C](\ell)\mathrm{x} & \triangleq & \{v\} & \text{if } C(\ell)\mathrm{x} = \{v\}, \text{ where } v \in \mathcal{V} \\
\alpha^c[C](\ell)\mathrm{x} & \triangleq & \mathcal{V} & \text{otherwise.}
\end{array}
$$



Partitionning abstraction



- Examples: sign, parity, etc
- Used in model-checking
- Not general enough (cannot do constant propagation!)

Partitionning abstraction (cont'd)

Given a partition $\mathfrak{C} \in \wp(\wp(\mathcal{S}))$

$\bigcup \mathfrak{C} = \mathcal{S}$, The blocks of the partition cover $\mathcal{S}$

$\forall B_1, B_2 \in \mathfrak{C} : (B_1 \neq B_2) \Rightarrow (B_1 \cup B_2 = \emptyset$, the blocks of the partition are disjoint

define the partitioning abstraction as:

$$
\begin{array}{rcl}
\mathfrak{L}^{\mathfrak{C}} & \triangleq & \{\cup S \mid S \subseteq \mathfrak{C}\} \\
\alpha^{\mathfrak{C}} & \in & \wp(\mathcal{S}) \mapsto \mathfrak{L}^{\mathfrak{C}} \\
\alpha^{\mathfrak{C}}(C) & \triangleq & \bigcup \{B \in \mathfrak{C} \mid C \cap B \neq \emptyset\}
\end{array}
$$

Finite unparameterized predicate abstraction

Given a set $\mathcal{S}$ of states and a finite set $\mathfrak{P} \in \wp(\wp(\mathcal{S}))$ of properties of elements of $\mathcal{S}$, the predicate abstraction $\alpha^{\mathfrak{P}}$ relative to $\mathfrak{P}$ is

$$\begin{array}{rcl} \alpha^{\mathfrak{P}} & \in & \wp(\mathcal{S}) \mapsto \wp(\mathcal{S}) \\ \alpha^{\mathfrak{P}}(C) & \triangleq & \bigcap \{P \in \mathfrak{P} \mid C \subseteq P\} \end{array}$$

- Examples: sign, parity, etc
- Any finite abstraction is a predicate abstraction
- Used in model-checking
- Not general enough (cannot do constant propagation (the set of predicates is infinite)!)

Maximum abstraction

- Record maximum of values of variables attached to each program point

$$\begin{array}{rcl} \alpha^{M} & \in & (\mathbb{L} \mapsto (\mathbb{V} \mapsto \wp(\mathbb{Z}))) \mapsto (\mathbb{L} \mapsto (\mathbb{V} \mapsto \mathbb{Z}^\infty)) \\ \alpha^{M}[C](\ell)\mathtt{x} & \triangleq & \max C(\ell)(x) \end{array}$$

where $\min \mathbb{Z} \triangleq -\infty$, $\max \mathbb{Z} \triangleq = +\infty$, $\mathbb{Z}^\infty \triangleq \mathbb{Z} \cup \{-\infty, +\infty\}$.

Minimum abstraction

$$\begin{array}{rcl} \alpha^{m} & \in & (\mathbb{L} \mapsto (\mathbb{V} \mapsto \wp(\mathbb{Z}))) \mapsto (\mathbb{L} \mapsto (\mathbb{V} \mapsto \mathbb{Z}^\infty)) \\ \alpha^{m}[C](\ell)\mathtt{x} & \triangleq & \min C(\ell)(x) \end{array}$$

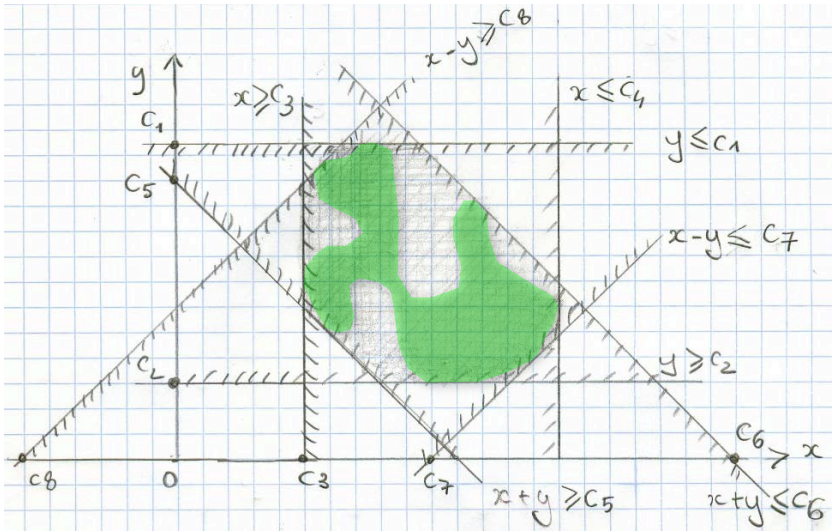
Interval abstraction

- Record both minimum and maximum of values of variables attached to each program point

$$\begin{array}{rcl} \alpha^{i} & \in & (\mathbb{L} \mapsto (\mathbb{V} \mapsto \wp(\mathbb{Z}))) \mapsto (\mathbb{L} \mapsto (\mathbb{V} \mapsto (\mathbb{Z}^\infty \times \mathbb{Z}^\infty))) \\ \alpha^{i}[C](\ell)\mathtt{x} & \triangleq & [\min C(\ell)(x),\ \max C(\ell)(x)] \end{array}$$

where a pair $\langle l, h \rangle \in \mathbb{Z}^\infty \times \mathbb{Z}^\infty$ is written $[l, h]$ with the convention that $[l, h] = \emptyset$ whenever $h < l$.

Octogon abstraction



Octogon abstraction (cont'd)

$$\alpha^o \in (\mathbb{L} \mapsto \wp(\mathcal{E})) \mapsto (\mathbb{L} \mapsto \wp(\mathcal{E})), \qquad \mathcal{E} \triangleq \mathbb{V} \mapsto \mathbb{Z}$$

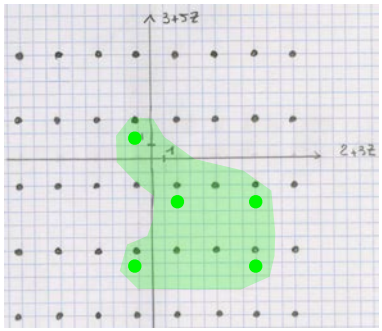
$$\begin{aligned}\alpha^o[I](\ell) \triangleq \bigcap_{x,y \in \mathbb{V}} & \{\rho \mid \max\{c \mid \{\rho' \mid c \leqslant \rho'(x) \pm \rho'(y)\} \subseteq I(\ell)\} \\ & \qquad \leqslant \rho(x) \pm \rho(y) \leqslant \\ & \qquad \min\{c' \mid \{\rho' \mid \rho'(x) \pm \rho'(y) \leqslant c'\} \subseteq I(\ell)\}\} \\ & \cap \{\rho \mid \max\{c \mid \{\rho' \mid c \leqslant \rho'(x)\} \subseteq I(\ell)\} \\ & \qquad \leqslant \rho(x) \leqslant \\ & \qquad \min\{c' \mid \{\rho' \mid \rho'(x) \leqslant c'\} \subseteq I(\ell)\}\}\end{aligned}$$

The coefficients have to be determined by the analysis among infinitely many possibilities.

Congruence abstract semantics

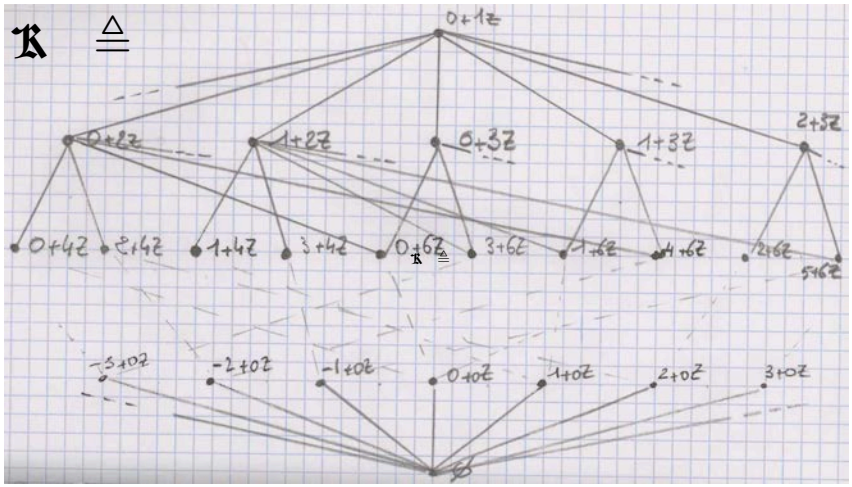
- Record the smallest congruence class of values of variables attached to each program point

  congruence class : $a + b\mathbb{Z} \triangleq \{x \mid x \equiv a \bmod b\} = \{a + k \times b \mid k \in \mathbb{Z}\}$



- Generalizes parity & constant propagation

Congruence abstract semantics (cont'd)



Congruence abstraction

$$\mathfrak{K} \triangleq \{a + b\mathbb{Z} \mid 0 \leqslant a < b\} \cup \{a + 0\mathbb{Z} \mid a \in \mathbb{Z}\} \cup \{\emptyset\}$$

$$\alpha^{\equiv} \in (\mathbb{L} \mapsto (\mathbb{V} \mapsto \wp(\mathcal{V}))) \mapsto (\mathbb{L} \mapsto (\mathbb{V} \mapsto \mathfrak{K}))$$

$$\alpha^{\equiv}[I](\ell)x \triangleq \bigcap\nolimits^{\equiv} \{a + b\mathbb{Z} \in \mathfrak{K} \mid I(\ell)x \subseteq a + b\mathbb{Z}\}$$

Bounded parametric abstraction

- Abstract to parametric properties of the form:

$$\bigcap_{i\in\Delta} \bigcap_{\langle a_i^1,\ldots,a_i^{k_i}\rangle\in\Delta_i} \{\sigma \mid P_i(a_i^1,\ldots,a_i^{k_i},\sigma)\},\quad \Delta\in\wp([1,n]),\quad \Delta_i\in\wp([1,n_i]^{k_i})$$

- Examples: predicate, congruence, interval abstraction:

at program point ℓ the value of variable $\mathtt{x}$ belongs to the interval $[a_{\ell,\mathtt{x}},\ b_{\ell,\mathtt{x}}]$ where the lower bound $a_{\ell,\mathtt{x}}$ and the upper bound $b_{\ell,\mathtt{x}}$ are integer constants to be determined by the analysis.

$$\bigcap_{\langle \ell,\mathtt{x}\rangle\in\mathbb{L}\times\mathbb{V}} \bigcap_{\langle a,b\rangle\in\{\langle a_{\ell,\mathtt{x}},\,b_{\ell,\mathtt{x}}\rangle\}} \{\sigma \mid P_{\ell,\mathtt{x}}(a,b,\sigma)\}$$

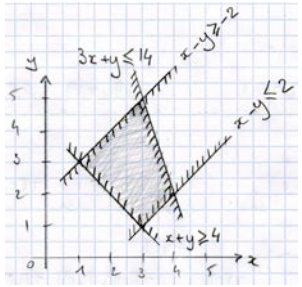
with $P_{\ell,\mathtt{x}}(a,b,\sigma) \triangleq (\sigma = \langle \ell,\ \rho\rangle) \Rightarrow (a \leqslant \rho(\mathtt{x}) \leqslant b)$.

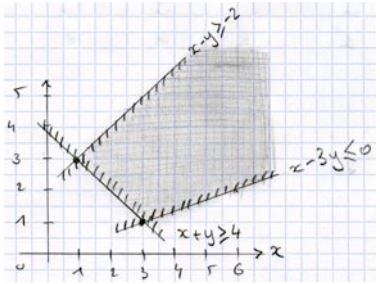
Polyhedral abstract properties

- Abstract to properties of the form:

$$\bigcap_{\langle a_1,\ldots,a_{n+1}\rangle\in\Delta} \{\langle x_1,\ \ldots,\ x_n\rangle \mid a_1x_1+\ldots+a_nx_n \leqslant a_{n+1}\}$$

where $\Delta\in\wp(\mathbb{R}^{n+1})$.

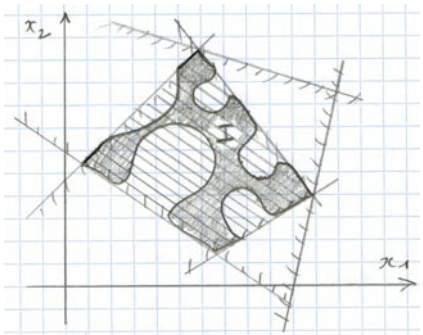




□

Polyhedral abstraction

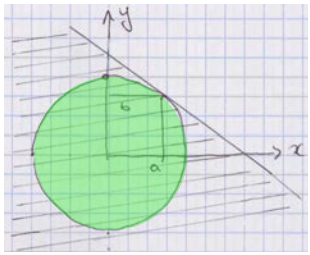
$$\begin{array}{rcl}\alpha^P & \in & (\mathbb{L}\mapsto\wp(\mathcal{E}))\mapsto(\mathbb{L}\mapsto\wp(\mathbb{R}^n))\\ \alpha^P(I)\ell & \triangleq & \bigcap_{\langle a_1,\ldots,a_n,a_{n+1}\rangle\in\mathbb{R}^{n+1}} \{\langle x_1,\ \ldots,\ x_n\rangle \mid a_1x_1+\ldots+a_nx_n\leqslant a_{n+1}\wedge\\ & & \quad \forall\rho\in I(\ell): a_1\rho(\mathtt{X}_1)+\ldots+a_n\rho(\mathtt{X}_n)\leqslant a_{n+1}\}\end{array}$$



Unbounded Parametric Properties

$$\bigcap_{i\in\Delta} \bigcap_{\langle a_i^1,\ldots,a_i^{k_i}\rangle\in\Delta_i} \{\sigma \mid P_i(a_i^1,\ldots,a_i^{k_i},\sigma)\},\qquad \Delta\in\wp(\mathbb{N}),\quad \Delta_i\in\wp(\mathbb{R}^{k_i})$$

- Example: $\bigcap_{\langle a,b\rangle:a^2+b^2=1} \{\langle x,\ y\rangle \mid ax+by-1\leqslant 0\}$



- In practice: must be limited to finitely many constraints!

[Dis]advantages of finite abstractions

The advantage of finite abstractions is that finite abstract domains have simple universal computer representations (e.g. using bit vectors or binary decision diagrams). However the restriction to finite abstractions has severe limitations, in particular if the same abstraction is to be used for infinitely many programs. In this case, it can be proved, that infinitary abstractions are strictly more powerful than finitary abstractions.

Example of infinitary abstraction

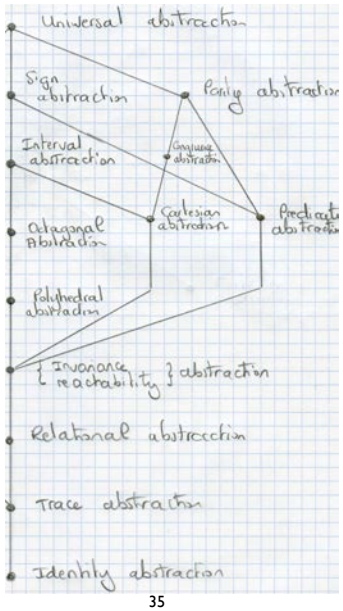
- ^{1}`x := 0`
  `while` ^{2}`(x < N) do`
  ^{3}`x := x + 1`
  `od`^{4}.

  $N \in \mathfrak{N}$ is a constant natural number 0, 1, ...

- Interval analysis: the value of x is in $[0, N-1]$ on loop entry.
- Achievable by finite analysis for a given N
- Infinitary analysis not satisfying ACC necessary for all N:

Any static analysis of this infinite family of programs using a finite abstract domain can only represent finitely many such intervals in $\{[0, N-1] \mid N \in \mathfrak{N}\}$ hence will necessarily be strictly less precise on infinitely many programs in this family.□

Hierarchy of abstractions



A few words on the formalization of abstraction

Example : Sintzoff (1972)

"$a \times a + b \times b$ yields always the object "pos" when a and b are the objects "pos" or "neg", and when the valuation is defined as follows :

pos+pos	=	pos	pos × pos	=	pos
pos+neg	=	pos,neg	pos × neg	=	neg
neg+pos	=	pos,neg	neq × pos	=	neg
neg+neg	=	neg	neg × neg	=	pos
V(p+q)	=	V(p)+V(q)	V(p × q)	=	V(p) × V(q)
V(0)	=	V(1) = ... =	pos		
V(-1)	=	V(-2) = ... =	neg		

The valuation of $a \times a + b \times b$ yields "pos" by the following computation :

V(a)	=	pos,neg	V(b)	=	pos,neg
V($a \times a$)	=	pos × pos, neg × neg	V($b \times b$)	=	pos × pos, neg × neg
	=	pos,pos = pos		=	pos,pos = pos
V($a \times a + b \times b$)	=	V($a \times a$)+V($b \times b$) =	pos+pos	=	pos"

Example : Sintzoff (1972), cont'd

What is wrong about it?

Example : Sintzoff (1972), cont'd

What is wrong about it?

We have "pos × neq = neq" with "V(0) = pos" and "V(-1) = V(-2) = ... = neg" so V(0 × -1) = V(0) × V(-1) = pos × neq = neq, proving that $0 = 0 \times -1 < 0$!

The rule of signs first appears in the work "Brahmasphutasiddhanta" (The Opening of the Universe) of the Indian mathematician and astronomer Brahmagupta (598–668) who got the correct rules (with "fortune" for positive and "debt" for negative but for "zero divided by zero is zero")

The theory of abstract interpretation

- Just to avoid such mistakes !
- Given a semantics and an abstraction, how to get an abstract semantics? ... a static analyzer? ... a verification method, etc.

Properties

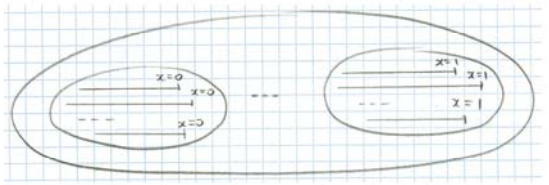
A *property* of elements of a set may be defined by a logical formula. For example, the property "x is an even natural number", written "$\mathbf{even}(x)$", may be defined as $\mathbf{even}(x) \triangleq (x \in \mathfrak{N} \wedge \exists k \in \mathfrak{N} : x = 2k)$.

This logical formula defines the set of naturals which are even that is $\{x \in \mathfrak{N} \mid \exists k \in \mathfrak{N} : x = 2k\} = \{0, 2, 4, \ldots, 2k, \ldots\}$.

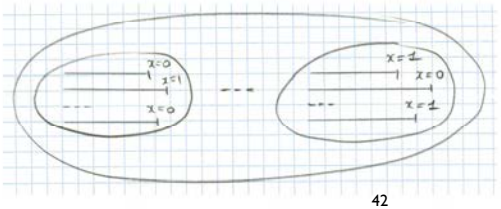
So instead of defining a property by a logical formula characterizing elements of a set, we can equivalently define a property as the set of elements of the set that have this property.

Example: trace properties

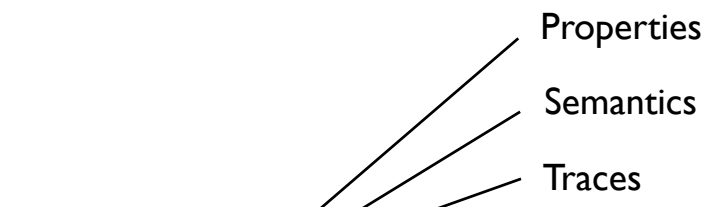
(I) If execution of program P terminates then the final value of variable x is always 0 or is always 1. In pictures



(II) If execution of program P terminates then the final value of variable x is always 0 or 1. In pictures



Trace abstraction



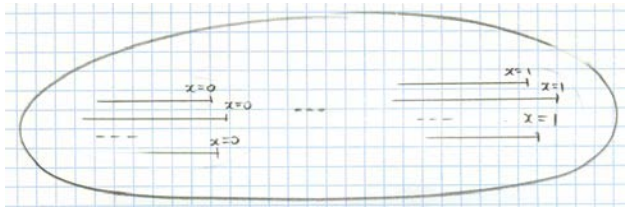
$$\alpha^{\pi} \in \wp(\wp(\mathbb{T})) \mapsto \wp(\mathbb{T})$$

$$\alpha^{\pi}(P) \triangleq \bigcup P$$

$$= \bigcup \{S \mid S \in P\}$$

Example of trace abstraction

- Both properties (I) and (II) have the same trace abstraction:



- Note: the safety/liveness distinction only applies to trace properties!

Best abstractions

- There is always a most precise way to abstract a concrete property
- Formalization: by a Galois connection

A Galois connections between posets $\langle C, \leqslant \rangle$ *and* $\langle A, \sqsubseteq \rangle$ *is a pair of functions* $\alpha \in C \mapsto A$ *and* $\gamma \in A \mapsto C$ *such that*

1. *the abstraction* $\alpha \in C \mapsto A$ *is increasing,*
2. *the concretization* $\gamma \in A \mapsto C$ *is increasing,*
3. $\gamma \circ \alpha$ *is extensive,*
4. $\alpha \circ \gamma$ *is reductive,*

which is written $\langle C, \leqslant \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle A, \sqsubseteq \rangle$.

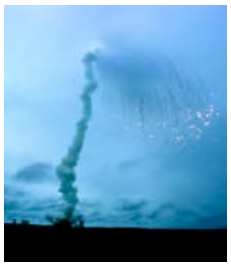
- Consequence: there is a best way to do the static analysis (but for the widening/narrowing)

Good abstractions

- There may be no best way to abstract a concrete property
- Formalization: by a concretization function
  - $\gamma \in A \mapsto \dot{C}$
  - *the concretization* $\gamma \in A \mapsto C$ *is increasing*
- Consequence: there is no best way to do the static analysis (arbitrary choices have to be made)

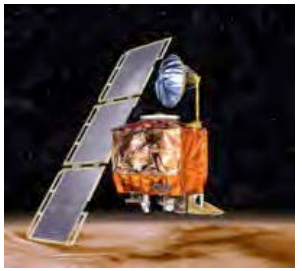
Abstract interpretation in action: a few words on ASTRÉE

All Computer Scientists Have Experienced Bugs



Ariane 5.01 failure
(overflow)

Patriot failure
(float rounding)



Mars orbiter loss
(unit error)

Introduction to the hard life of the miserable programmer

1) Integers
2) Reals
3) Arrays
4) ...

Integers

The factorial program (fact.c)

```
#include <stdio.h>
int fact (int n ) {                    ← fact(n) = 2 × 3 × ··· × n
  int r, i;
  r = 1;
  for (i=2; i<=n; i++) {
    r = r*i;
  }
  return r;
}
int main() { int n;
  scanf("%d",&n);
  printf("%d!=%d\n",n,fact(n));
}
```

← read n (typed on keyboard)
← write $n! = \text{fact}(n)$

Execution of the factorial program (fact.c)

```
#include <stdio.h>
int fact (int n ) {
  int r, i;
  r = 1;
  for (i=2; i<=n; i++) {
    r = r*i;
  }
  return r;
}
int main() { int n;
  scanf("%d",&n);
  printf("%d!=%d\n",n,fact(n));
}
```

```
% gcc fact.c -o fact.exec
% ./fact.exec
3
3!  = 6
% ./fact.exec
4
4!  = 24
%
```

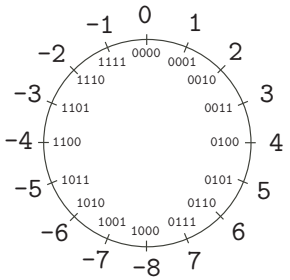
Execution of the factorial program (fact.c)

```
#include <stdio.h>
int fact (int n ) {
  int r, i;
  r = 1;
  for (i=2; i<=n; i++) {
    r = r*i;
  }
  return r;
}
int main() { int n;
  scanf("%d",&n);
  printf("%d!=%d\n",n,fact(n));
}
```

```
% gcc fact.c -o fact.exec
% ./fact.exec
3
3!  = 6
% ./fact.exec
4
4!  = 24
% ./fact.exec
100
100!  = 0
% ./fact.exec
20
20!  = -2102132736
```

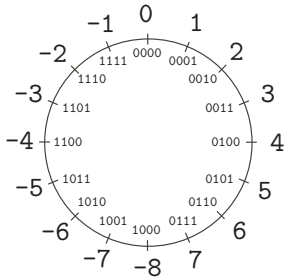

Bug hunt

- Computers use integer modular arithmetics on n bits (where $n = 16$, 32, 64, etc)
- Example of an integer representation on 4 bits (in *complement to two*) :



Bug hunt

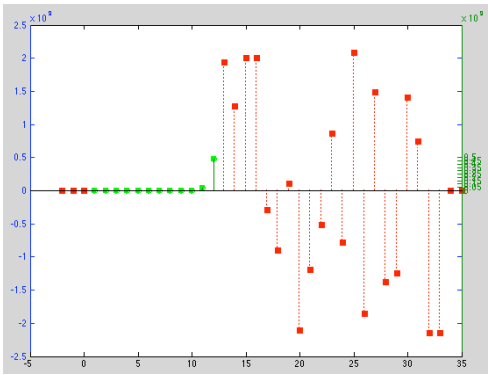
- Computers use integer modular arithmetics on n bits (where $n = 16$, 32, 64, etc)
- Example of an integer representation on 4 bits (in *complement to two*) :



- Only integers between -8 and 7 can be represented on 4 bits
- We get $7 + 2 = -7$
  $7 + 9 = 0$

The bug is a failure of the programmer

In the computer, the function `fact(n)` coincide with $n! = 2 \times 3 \times \ldots \times n$ on the integers only for $1 \leqslant n \leqslant 12$:



And in OCaml the result is different!

```
let rec fact n = if (n = 1) then 1 else n * fact(n-1);;
```

fact(n)	C	OCaml
fact(1)	1	1
...	...	...
fact(12)	479001600	479001600
fact(13)	1932053504	−215430144
fact(14)	1278945280	−868538368
fact(15)	2004310016	−143173632
fact(16)	2004189184	−143294464
fact(17)	−288522240	−288522240
fact(18)	−898433024	−898433024
fact(19)	109641728	109641728
fact(20)	−2102132736	45350912
fact(21)	−1195114496	952369152

fact(22)	−522715136	−522715136
fact(23)	862453760	862453760
fact(24)	−775946240	−775946240
fact(25)	2076180480	−71303168
fact(26)	−1853882368	293601280
fact(27)	1484783616	−662700032
fact(28)	−1375731712	771751936
fact(29)	−1241513984	905969664
fact(30)	1409286144	−738197504
fact(31)	738197504	738197504
fact(32)	−2147483648	0
fact(33)	−2147483648	0
fact(34)	0	0

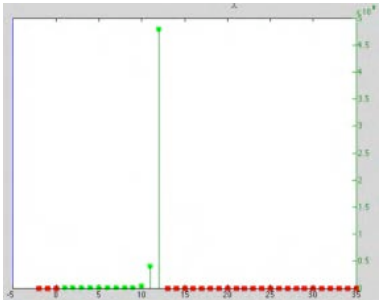
Absence of runtime error

```
int fact (int n ) {
  int r, i;
  r = 1;
  for (i=2; i<=n; i++) {     ← no overflow of i++
    r = r*i;                 ← no overflow of r*i
  }
  return r;
}
```


Proof of absence of runtime error by static analysis

```
% cat -n fact_lim.c
 1 int MAXINT = 2147483647;
 2 int fact (int n) {
 3    int r, i;
 4    if (n < 1) || (n = MAXINT) {
 5        r = 0;
 6    } else {
 7        r = 1;
 8        for (i = 2; i<=n; i++) {
 9            if (r <= (MAXINT / i)) {
10                r = r * i;
11            } else {
12                r = 0;
13            }
14        }
15    }
16    return r;
17 }
18
19 int main() {
20    int n, f;
21    f = fact(n);
22 }
% astree -exec-fn main fact_lim.c |& grep WARN
%
```

→ No alarm!



Reals

Mathematical models and their implementation on computers

- Mathematical models of physical systems use real numbers
- Computer modeling languages (like SCADE) use real numbers
- Real numbers are hard to represent in a computer (π has an infinite number of decimals)
- Computer programming languages (like C or OCAML) use floating point numbers

Floats

- *Floating point numbers* are a finite subset of the *rationals*
- For example one can represent 32 floats on 6 bits, the 16 positive normalized floats spread as follows on the line:

- When real computations do not spot on a float, one must *round* the result to a close float

Example of rounding error (1)

$$(x + a) - (x - a) \neq 2a$$

```
#include <stdio.h>
int main() {
  double x, a; float y, z;
  x = 1125899973951488.0;
  a = 1.0;
  y = (x+a);
  z = (x-a);
  printf("%f\n", y-z);
}
```

```
% gcc arrondi1.c -o arrondi1.exec
% ./arrondi1.exec
134217728.000000
%
```


Example of rounding error (2)

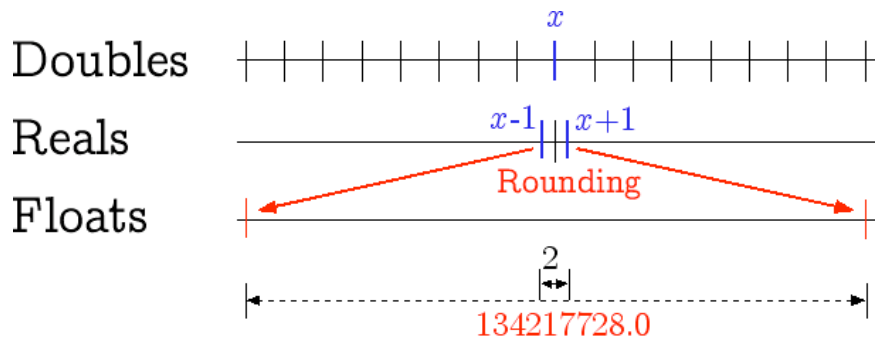
$$(x + a) - (x - a) \neq 2a$$

```
#include <stdio.h>
int main() {
  double x, a; float y, z;
  x = 1125899973951487.0;
  a = 1.0;
  y = (x+a);
  z = (x-a);
  printf("%f\n", y-z);
}
```

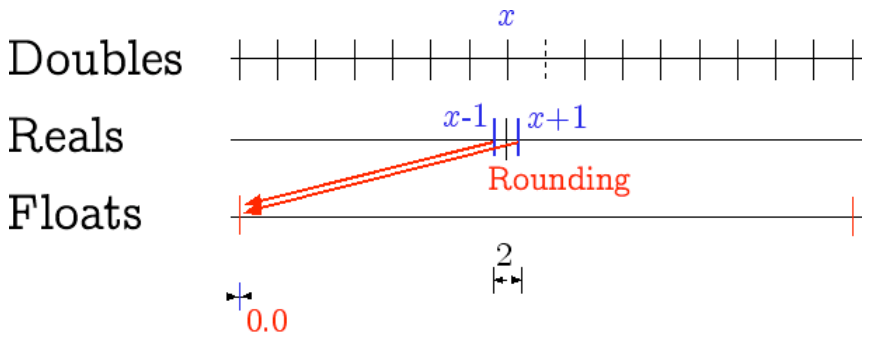
```
% gcc arrondi2.c -o arrondi2.exec
% ./arrondi2.exec
0.000000
%
```

only 1 digit changed

Bug hunt (1)



Bug hunt (2)

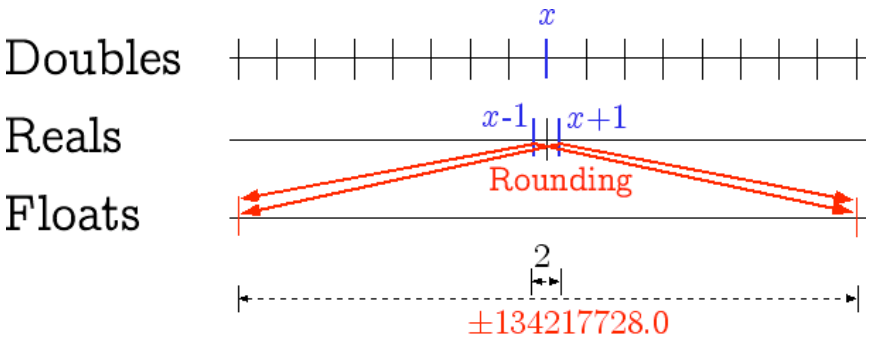


Proof of absence of runtime error by static analysis

```
% cat -n arrondi3.c
     1 int main() {
     2     double x; float y, z, r;;
     3     x = 1125899973951488.0;
     4     y = x + 1;
     5     z = x - 1;
     6     r = y - z;
     7     __ASTREE_log_vars((r));
     8 }
% astree -exec-fn main -print-float-digits 10 arrondi3.c \
  |& grep "r in "
direct = <float-interval:  r in [-134217728, 134217728] >
```
(1)

(1) Astrée considers the worst rounding case (towards $+\infty$, $-\infty$, 0 or to the nearest) whence the possibili
to obtain -134217728.

The verification is done in the worst case



A programming exercices...that goes wrong

"*Put* x *in* [m, M] *modulo* (M − m)":

```
y = x - (int) ((x-m)/(M-m))*(M-m);
```

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

```
% cat -n modulo.c
#include <stdio.h>
#include <math.h>
int main () {
   float m, M, x, y; M = 4095.0; m = -M;
   x = 4094.9997558593750; /* largest float strictly less than M */
   y = x - (int) ((x-m)/(M-m))*(M-m);
printf("%.20f\n",y);
}
% gcc modulo.c; ./a.out
-4095.00024414062500000000
%
```


Analysis by ASTRÉE

```
% cat modulo-a.c
int main () {
   float m, M, x, y;
M = 4095.0; m = -M;
x = 4094.9997558593750; /* largest float strictly less than M */
   y = x - (int) ((x-m)/(M-m))*(M-m);
__ASTREE_log_vars((y));
}
% astree -exec-fn main -print-float-digits 25 modulo-a.c |& grep "y in"
direct = <float-interval: y in [-4095.000244140625, 4094.999755859375] >
%
```


Examples of bugs due to rounding errors

- The patriot missile bug missing Scuds in 1991 because of a software clock incremented by $\frac{1}{10}$th of a seconde ($(0,1)_{10} = (0,0001100110011001100\ldots)_2$ in binary)
- The Exel 2007 bug : 77.1 × 850 gives 65,535 but displays as 100,000! (2)

2	65535-2^(-37)	100000	65536-2^(-37)	100001
3	65535-2^(-36)	100000	65536-2^(-36)	100001
4	65535-2^(-35)	100000	65536-2^(-35)	100001
5	65535-2^(-34)	65535	65536-2^(-34)	65536
6	65535-2^(-36)-2^(-37)	100000	65536-2^(-36)-2^(-37)	100001
7	65535-2^(-35)-2^(-37)	100000	65536-2^(-35)-2^(-37)	100001
8	65535-2^(-35)-2^(-36)	100000	65536-2^(-35)-2^(-36)	100001
9	65535-2^(-35)-2^(-36)-2^(-37)	65535	65536-2^(-35)-2^(-36)-2^(-37)	65536

(2) Incorrect float rounding which leads to an alignment error in the conversion table while translating 64 bits IEEE 754 floats into a Unicode character string. The bug appears exactly for six numbers between 65534.99999999995 and 65535 and six between 65535.99999999995 and 65536.

Arrays

Buffer overflow

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
```


Analysis by ASTRÉE

```
% cat -n unpredictable-a.c
     1  const int false = 0;
     2  int main () { int n, T[1], x;
     3  n = 1;
     4  x = T[n];
     5  __ASTREE_assert((false));
     6  }
% astree -exec-fn main unpredictable-a.c |& grep "WARN"
unpredictable-a.c:4.4-8::[call#main@2:]: WARN: invalid dereference: dereferencing
4 byte(s) at offset(s) [4;4] may overflow the variable T of byte-size 4
%
```

No alarm on `assert(false)` because execution is assumed to stop after a definite runtime error with unpredictable results [4].

[4] Equivalent semantics if no alarm.

ASTRÉE

Implicit specification

Different Classes of Run-time Errors

1. Errors terminating the execution [5]. ASTRÉE warns and continues by taking into account only the executions that did not trigger the error.
2. Errors not terminating the execution with predictable outcome [6]. ASTRÉE warns and continues with worst-case assumptions.
3. Errors not terminating the execution with unpredictable outcome [7]. ASTRÉE warns and continues by taking into account only the executions that did not trigger the error.

$\Rightarrow$ ASTRÉE is sound with respect to C standard, unsound with respect to C implementation, unless no false alarm of type 3.

[5] floating-point exceptions e.g. (invalid operations, overflows, etc.) when traps are activated
[6] e.g. overflows over signed integers resulting in some signed integer.
[7] e.g. memory corruptionss.

Undecidability and complexity

- The mathematical proof problem is undecidable [3]
- Even assuming finite states, the complexity is much too high for combinatorial exploration to succeed
- Example: 1.000.000 lines $\times$ 50.000 variables $\times$ 64 bits $\simeq 10^{27}$ states
- Exploring 10^{15} states per seconde, one would need 10^{12} s $>$ 300 centuries (and a lot of memory)!

[3] there are infinitely many programs for which a computer cannot solve them in finite time even with an infinite memory.

The difficulty of scaling up

- The abstraction must be coarse enough to be effectively computable with reasonable resources
- The abstraction must be precise enough to avoid false alarms
- Abstractions to *infinite domains with widenings* are more expressive than abstractions to *finite domains* (when considering the analysis of a programming language) [CC92a]
- Abstractions are ultimately incomplete (even intrinsically for some semantics and specifications [CC00])

Required Precision

- Coverity Prevent™ Static Analyzer has "an average FP rate of about 15%, with some users reporting FP rates of as low as 5%" [`www.coverity.com/html/prevent-for-c-features.html`]
- Consider a 1.000.000 LOCS control/command safety critical program, with 1 potential error per line (often much more)
- 5% FP = 5.000 false positives
- In safety critical software, false alarms must be justified for certification
- False/true alarms can take hours to days to be solved $\Longrightarrow$ the cost is several man $\times$ years!

Modular refinable abstraction

The abstract semantics is decomposed into:

- A structural fixpoint iterator (by composition on the program syntax)
- A collection of parametric abstract domains with:
  - parameters to adjust the expressivity of the abstraction
  - parametric convergence acceleration (parameters to adjust the frequence and precision of widenings/narrowings)
  - analysis directives (to locally adjust the choice of abstractions)
- A reduction performing the conjunction of the abstractions

$\Rightarrow$ Easily refinable by parameter/directive adjustment and extendable by addition of new abstract domains!

Abstraction/refinement by tuning the cost/precision ratio in Astrée

- Approximate reduced product of a choice of coarsenable/refinable abstractions
- Tune their precision/cost ratio by
  - Globally by parametrization
  - Locally by (automatic) analysis directives

  so that the overall abstraction is not uniform.

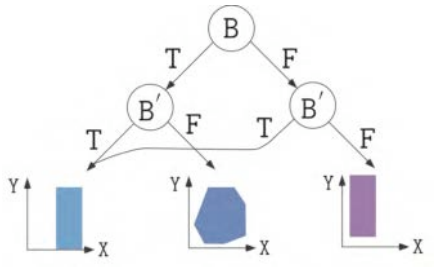
Example of abstract domain choice in Astrée

```
/* Launching the forward abstract interpreter */
/* Domains:  Guard domain, and Boolean packs (based on Absolute
value equality relations, and Symbolic constant propagation
(max_depth=20), and Linearization, and Integer intervals, and
congruences, and bitfields, and finite integer sets, and Float
intervals), and Octagons, and High_passband_domain(10), and
Second_order_filter_domain (with real roots)(10), and
Second_order_filter_domain (with complex roots)(10), and
Arithmetico-geometric series, and new clock, and Dependencies
(static), and Equality relations, and Modulo relations, and
Symbolic constant propagation (max_depth=20), and Linearization,
and Integer intervals, and congruences, and bitfields, and
finite integer sets, and Float intervals.  */
```


Example of abstract domain functor in Astrée: decision trees

- 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

Reduction [CC79, CCF+08]

Example: reduction of intervals by simple congruences

```
% cat -n congruence.c
     1 /* congruence.c */
     2 int main()
     3 { int X;
     4   X = 0;
     5   while (X <= 128)
     6     { X = X + 4; };
     7   __ASTREE_log_vars((X));
     8 }
% astree congruence.c -no-relational -exec-fn main |& egrep "(WARN)|(X in)"
direct = <integers (intv+cong+bitfield+set): X in {132} >
```

Intervals : $X \in [129, 132]$ + congruences : $X = 0 \mod 4 \Longrightarrow X \in \{132\}$.

Abstraction by floating-point linearization [Min04a, Min04b]

- Approximate arbitrary expressions in the form
  $[a_0, b_0] + \sum_k ([a_k, b_k] \times V_k)$
- Example:
  `Z = X - (0.25 * X)` is linearized as
  $Z = ([0.749\cdots, 0.750\cdots] \times X) + (2.35\cdots 10^{-38} \times [-1, 1])$
- Allows simplification even in the interval domain
  if X ∈ [-1,1], we get $|Z| \leq 0.750\cdots$ instead of $|Z| \leq 1.25\cdots$
- Allows using a relational abstract domain (octagons)
- Example of good compromize between cost and precision

Parameterized abstractions

- Parameterize the cost / precision ratio of abstractions in the static analyzer
- Examples:
  - array smashing: `--smash-threshold` n (400 by default)
    → smash elements of arrays of size $> n$, otherwise individualize array elements (each handled as a simple variable).
  - packing in octogons: (to determine which groups of variables are related by octagons and where)
    · `--fewer-oct`: no packs at the function level,
    · `--max-array-size-in-octagons` n: unsmashed array elements of size $> n$ don't go to octagons packs

Parameterized widenings

- Parameterize the rate and level of precision of widenings in the static analyzer
- Examples:
  - delayed widenings: `--forced-union-iterations-at-beginning` n (2 by default)
  - thresholds for widening (e.g. for integers):

```
let widening_sequence =
 [ of_int 0; of_int 1; of_int 2; of_int 3; of_int 4; of_int 5;
   of_int 32767; of_int 32768; of_int 65535; of_int 65536;
   of_string "2147483647"; of_string "2147483648";
   of_string "4294967295" ]
```


Analysis directives

- Require a local refinement of an abstract domain
- Example:

```
% 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 be automated in Astrée (if the considered family of programs has "repeat" loops).

Automatic analysis directives

- The directives can be inserted automatically by static analysis
- Example:

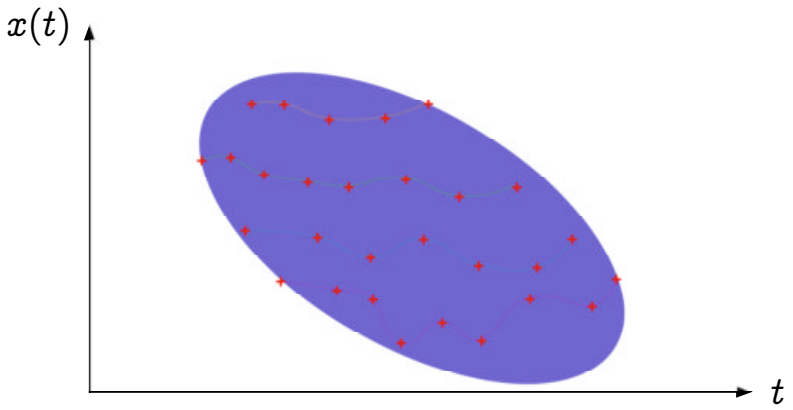
```
% cat p.c
int clip(int x, int max, int min) {
 if (max >= min) {
  if (x <= max) {
   max = x;
  }
  if (x < min) {
   max = min;
  }
 }
 return max;
}
void main() {
 int m = 0; int M = 512; int x, y;
 y = clip(x, M, m);
  __ASTREE_assert(((m<=y) && (y<=M)));
}
% astree -exec-fn main p.c |& grep WARN
%
```

```
% astree -exec-fn main p.c -dump-partition
...
int (clip)(int x, int max, int min)
{
  if ((max >= min))
  {  __ASTREE_partition_control((0))
    if ((x <= max))
    {
      max = x;
    }
    if ((x < min))
    {
      max = min;
    }
    __ASTREE_partition_merge_last(());
  }
  return max;
}
...
%
```


Adding new abstract domains

- The weakest invariant to prove the specification may not be expressible with the current refined abstractions ⇒ false alarms cannot be solved
- No solution, but adding a new abstract domain:
  - representation of the abstract properties
  - abstract property transformers for language primitives
  - widening
  - reduction with other abstractions
- Examples : ellipsoids for filters, exponentials for accumulation of small rounding errors, quaternions, ...

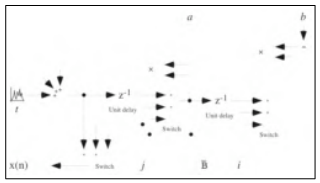
Abstraction by ellipsoid for filters



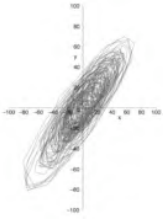
Ellipsoids $(x-a)^2+(y-b)^2 \leq c$

Example of Refinement: Ellipsoid Abstract Domain for Filters

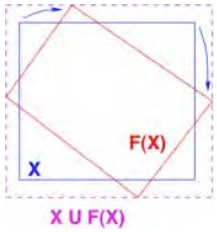
2^d Order Digital Filter:



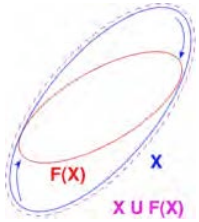
- Computes $X_n = \begin{cases} \alpha X_{n-1} + \beta X_{n-2} + Y_n \\ I_n \end{cases}$
- The concrete computation is bounded, which must be proved in the abstract.
- There is no stable interval or octagon.
- The simplest stable surface is an ellipsoid.



execution trace



unstable interval



stable ellipsoid

Example of analysis by ASTRÉE

```
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; }
}
```


... Analysis with CBMC

```
Script started on Tue Jul 29 23:44:00 2008
% time ./cbmc filter.c
...
Starting Bounded Model Checking
Unwinding loop 1 iteration 1
Unwinding loop 1 iteration 2
...
Unwinding loop 1 iteration 95479
cbmc(34799) malloc: *** mmap(size=2097152) failed (error code=12)
*** error: can't allocate region
*** set a breakpoint in malloc_error_break to debug
terminate called after throwing an instance of 'std::bad_alloc'
  what():  St9bad_alloc
...
Abort
29668.051u 101.916s 8:20:41.88 99.0% 0+0k 1+10io 2680pf+0w
% ^Dexit
```


A common believe on static analyzers

"The properties that can be proved by static analyzers are often simple" [2]

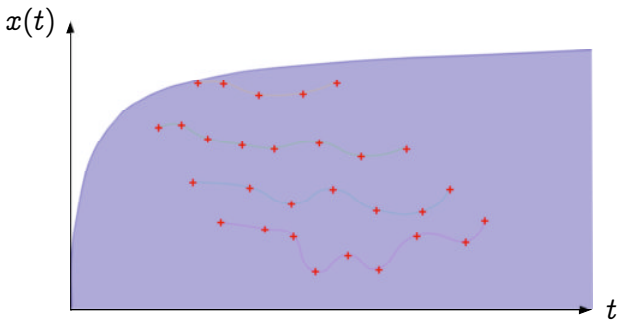
Like in mathematics:

- May be simple to state (no overflow)
- But harder to discover ($P \in [-1325.4522, 1325.4522]$)
- And difficult to prove (since it requires finding a non trivial non-linear invariant for second order filters with complex roots [Fer04], which can hardly be found by exhaustive enumeration)

Reference

[2] Vijay D'Silva, Daniel Kroening, and Georg Weissenbacher. A Survey of Automated Techniques for Formal Software Verification. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 27, No. 7, July 2008.

Abstraction by exponentials for accumulation of small rounding errors



Exponentials $a^x \leq y$

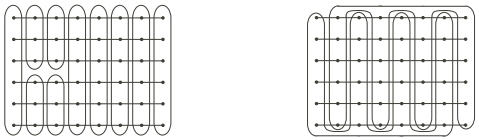
Example of analysis by ASTRÉE

```
% 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)
                * 5.0e-03)); };
  B = A;
  if (SWITCH) {A = P;}
  else {A = X;}
}
```

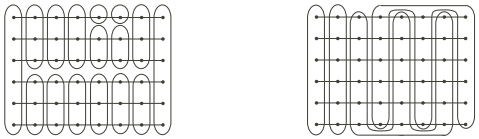
```
void main()
{ FIRST = TRUE;
  while (TRUE) {
    dev( );
    FIRST = FALSE;
    __ASTREE_wait_for_clock(());
  }}
% cat retro.config
__ASTREE_volatile_input((E [-15.0, 15.0]));
__ASTREE_volatile_input((SWITCH [0,1]));
__ASTREE_max_clock((3600000));

astree -exec-fn main -config-sem retro.config
retro.c |& grep "|P|" | tail -n 1
|P| <=1.0000002*((15.  +
5.8774718e-39/(1.0000002-1))*(1.0000002)ĉlock -
5.8774718e-39/(1.0000002-1)) + 5.8774718e-39 <=
23.039353
```


Trace partitioning

- State partitionning by program points [9]



- Trace partitionning [10]



- Trace partitionning abstract interpretation combines the effects of case analysis and symbolic execution [MR05, RM07]

(9) all reachable states corresponding to a given program point are over-approximated by a local invariant on memory states reachable at that program points

(10) portions of traces starting at a given program point for given memory values and finishing at a given program point are analyzed by an overapproximating abstract execution

Example of trace partitioning

Principle:

- Semantic equivalence:

```
if (B) { C1 } else { C2 }; C3
```

⇓

```
if (B) { C1; C3 } else { C2; C3 };
```

- More precise in the abstract: concrete execution paths are merged later.

Application:

```
if (B)
  { X=0; Y=1; }
else
  { X=1; Y=0; }
R = 1 / (X-Y);
```

cannot result in a division by zero

Scalability of trace partitioning

```
% cat -n explode.c
     1 void main() {
     2   /* uninitialized local variables */
     3   unsigned int a1, a2, a3, a4, a5, a6; int r;
     4   while(a1<20) { a1++; }
     5   while(a2<20) { a2++; }
     6   while(a3<20) { a3++; }
     7   while(a4<20) { a4++; }
     8   while(a5<20) { a5++; }
     9 }
% (time astree -exec-fn main -partition-all explode.c) |& egrep
  "error:|WARN|pf\+"
*** error: can't allocate region
Fatal error: out of memory.
5072.204u 44.410s 1:29:31.01 95.2% 0+0k 0+0io 45661pf+0w
%
```

Trace partitionning must be performed locally (thanks to *analysis directives*)

– CBMC cannot make it either:

```
% time ./cbmc explode.c
...
Unwinding loop 1 iteration 1
Unwinding loop 1 iteration 2
...
Unwinding loop 1 iteration 8731
...
Abort
```

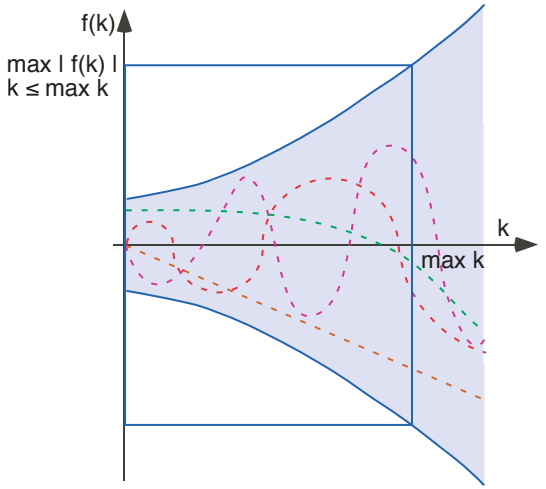
– But ASTRÉE succeeds (without partitionning `-partition-all`)

```
% (time astree -exec-fn main explode.c) |& egrep "error:|WARN|pf"
0.402u 0.064s 0:00.48 95.8% 0+0k 0+0io 0pf+0w
%
```


Example of abstract domain: exponential

Basic idea: bound evolution over time

Overapproximation with an arithmetico-geometric series:



Arithmetico-geometric series [14] [Fer05a]

– Abstract domain: $(R^+)^5$

– Concretization:

$$\gamma \in (R^+)^5 \longmapsto \wp(N \mapsto R)$$

$$\gamma(M, a, b, a', b') =$$
$$\{f \mid \forall k \in N : |f(k)| \leq \left(\boldsymbol{\lambda}\, x \cdot ax + b \circ (\boldsymbol{\lambda}\, x \cdot a'x + b')^k\right)(M)\}$$

i.e. any function bounded by the arithmetic-geometric progression.

Reference

[4] J. Feret. The arithmetic-geometric progression abstract domain. In *VMCAI'05*, Paris, LNCS 3385, pp. 42–58, Springer, 2005.

[14] here in R but must be implemented in the floats by appropriate roundings!

Example

```
% cat count.c
typedef enum {FALSE = 0, TRUE = 1} BOOLEAN;
volatile BOOLEAN I; int R; BOOLEAN T;
void main() {
  R = 0;
  while (TRUE) {
    __ASTREE_log_vars((R));
    if (I) { R = R + 1; }
    else { R = 0; }
    T = (R >= 100);
    __ASTREE_wait_for_clock(());
  }}
```

← potential overflow!

```
% cat count.config
__ASTREE_volatile_input((I [0,1]));
__ASTREE_max_clock((3600000));
% astree -exec-fn main -config-sem count.config count.c|grep '|R|'
|R| <= 0. + clock *1. <= 3600001.
```

More precise than the *clock domain* (intervals for X, $X + \texttt{clock}$, $X - \texttt{clock}$) which could therefore be suppressed!

Applications

Success stories of ASTRÉE

- In November 2003, Astrée proved the absence of any real-time errors in the primary flight-control software of one of Airbus' models. The analysis was performed completely automatically. The system's 132,000 lines of C code were analyzed in only 80 minutes on a 2.8GHz 32-bit PC using 300MB of memory (and in only 50 minutes on an AMD Athlon 64 using 580MB of memory). In January 2004, Astrée was extended to analyze the electric flight-control codes for another Airbus series.

© 2007 Xavier Rival

- In April 2008, Astrée was able to prove the absence of any RTE in a C version of the automatic docking software of the Jules Vernes Automated Transfer Vehicle (ATV), enabling ESA to transport payloads to the International Space Station. The analysis was performed completely automatically.

Courtesy of NASA

The Airbus A340/600 & A380 flight control software

Application at Airbus France

- Automatic proofs of absence of runtime errors in Electric Flight Control Software:
  - A340/600: 132.000 lines of C, 40mn on a PC 2.8 GHz, 300 Mb (Nov. 2003)
  - A380: 1.000.000 lines of C, 34h, 8 Gb (Nov. 2005)

  no false alarm, **World premières !**

The ATV control software

The ASTRIUM ST case study MSU SW

- The MSU SW contains mainly:
  1. Navigation and control algorithms
  2. A (very simplified) mission management
- Single task cyclic synchronous software
- Initially in ADA → Scade 5/6 exact model → C (38K LOCS)

" ⇒ **The C code of the case study may contain errors!** " [:-)]

esa ceo ÉCOLE POLYTECHNIQUE EADS ASTRIUM Space

Preparatory work

- Definition of stubs for the library (reusable)
- Choice of code generation options for SCADE
- Definition of a few environment properties (a few input ranges)
- Setting ASTRÉE parameters
- Analysis takes < 4mn

Alarms raised by ASTRÉE

- Complex control flow
- Quaternion computation (normalisation)
- Runge Kutta integration (4th order integration scheme)
- Kalman filters (8th order linear filter)
- Controller estimation

} False alarms

- Bugs

} True alarms

Analysis of the alarms

- Correct bugs (in Scade compiler V6 and in the analyzed program)
- Add numerical protections on environment (forgotten hypotheses)
- Add numerical protections in computations (in a first phase)
- Design and implement a few abstract domains (quaternions, Runge Kutta integration, Kalman filters) avoiding the unecessary protections (in a second phase)

➔ 0 false alarms

Very efficient tool compared to Polyspace Verifier

Resolution of the alarms

- Complex control flow

} Improvement of the model

- Quaternion computation
- Runge Kutta integration
- Kalman filters
- Controller estimation

} ≈ 30 numerical protections (first phase)

- Bugs

} Corrected

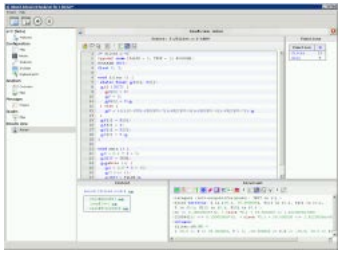
➔ 0 false alarm

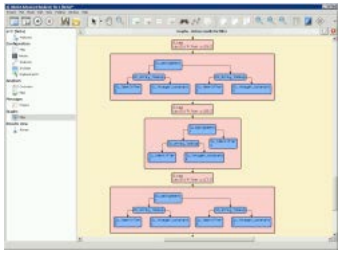
Commercialisation

Commertialisation: http://www.absint.de/astree/

AbsInt Angewandte Informatik — Astrée Run-Time Error Analyzer

Astrée is a static program analyzer that proves the absence of run-time errors (RTE) in safety-critical embedded applications written or automatically generated in C.





Astrée analyzes structured C programs with complex memory usages and recursion but without dynamic memory allocation. This targets embedded applications as found in earth transportation, nuclear energy, medical instrumentation, aeronautics and space flight, in particular synchronous control/command such as electric flight control.

Conclusion

Abstract interpretation is great!

- Abstract interpretation ranges from esoteric theory to practical applications
- Does scale up !
- A great research subject !

A wide range of applications

- Static Program Analysis [CC77], [CH78], [CC79] including Dataflow Analysis; [CC79], [CC00], Set-based Analysis [CC95], Predicate Abstraction [Cou03], ...
- Grammar Analysis and Parsing [CC03];
- Hierarchies of Semantics and Proof Methods [CC92b], [Cou02];
- Typing & Type Inference [Cou97];
- (Abstract) Model Checking [CC00];
- Program Transformation (including program optimization, partial evaluation, etc) [CC02];

- Software Watermarking [CC04];
- Bisimulations [RT04, RT06];
- Language-based security [GM04];
- Semantics-based obfuscated malware detection [PCJD07].
- Databases [AGM93, BPC01, BS97]
- Computational biology [Dan07]
- Quantum computing [JP06, Per06]

All these techniques involve sound approximations that can be formalized by abstract interpretation

Challenges in Abstract Interpretation

Short term : automatic help on the diagnosis of the origin of alarms

Midterm : Parallelism

Long term : Liveness for infinite state systems

Bibliography

Short bibliography

[AGM93] G. Amato, F. Giannotti, and G. Mainetto. Data sharing analysis for a database programming language via abstract interpretation. In R. Agrawal, S. Baker, and D.A.Bell, editors, *Proc. 19th Int. Conf. on Very Large Data Bases*, pages 405–415, Dublin, IE, 24–27 Aug. 1993. MORGANKAUFMANN.

[BCC+03] 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 *Proc. ACM SIGPLAN '2003 Conf. PLDI*, pages 196–207, San Diego, CA, US, 7–14 June 2003. ACM Press.

[BPC01] J. Bailey, A. Poulovassilis, and C. Courtenage. Optimising active database rules by partial evaluation and abstract interpretation. In *Proc. 8th Int. Work. on Database Programming Languages*, LNCS 2397, pages 300–317, Frascati, IT, 8–10 Sep. 2001. Springer.

[BS97] V. Benzaken and X. Schaefer. Static integrity constraint management in object-oriented database programming languages via predicate transformers. In M. Aksit and S. Matsuoka, editors, *Proc. 11th European Conf. on Object-Oriented Programming, ECOOP '97*, LNCS 1241. Springer, Jyväskylä, FI, 9–13 June 1997.

[CC76] P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proc. 2nd Int. Symp. on Programming*, pages 106–130, Paris, FR, 1976. Dunod.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th POPL*, pages 238–252, Los Angeles, CA, 1977. ACM Press.

[CC79] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *6th POPL*, pages 269–282, San Antonio, TX, 1979. ACM Press.

[CC92a] P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation, invited paper. In M. Bruynooghe and M. Wirsing, editors, *Proc. 4th Int. Symp. on PLILP '92*, Leuven, BE, 26–28 Aug. 1992, LNCS 631, pages 269–295. Springer, 1992.

[CC92b] P. Cousot and R. Cousot. Inductive definitions, semantics and abstract interpretation. In *19th POPL*, pages 83–94, Albuquerque, NM, US, 1992. ACM Press.

[CC95] P. Cousot and R. Cousot. Formal language, grammar and set-constraint-based program analysis by abstract interpretation. In *Proc. 7th FPCA*, pages 170–181, La Jolla, CA, US, 25–28 June 1995. ACM Press.

[CC00] P. Cousot and R. Cousot. Temporal abstract interpretation. In *27th POPL*, pages 12–25, Boston, MA, US, Jan. 2000. ACM Press.

[CC02] P. Cousot and R. Cousot. Systematic design of program transformation frameworks by abstract interpretation. In *29th POPL*, pages 178–190, Portland, OR, US, Jan. 2002. ACM Press.

[CC03] P. Cousot and R. Cousot. Parsing as abstract interpretation of grammar semantics. *Theoret. Comput. Sci.*, 290(1):531–544, Jan. 2003.

[CC04] P. Cousot and R. Cousot. An abstract interpretation-based framework for software watermarking. In *31st POPL*, pages 173–185, Venice, IT, 14–16 Jan. 2004. ACM Press.

[CCF+07] 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 M. Hinchey, He Jifeng, and J. Sanders, editors, *Proc. 1st TASE '07*, pages 3–17, Shanghai, CN, 6–8 June 2007. IEEE Comp. Soc. Press.

[CCF+08] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. Combination of abstractions in the Astrée static analyzer. In M. Okada and I. Satoh, editors, *11th ASIAN 06*, pages 272–300, Tokyo, JP, 6–8 Dec. 2006, 2008. LNCS 4435, Springer.

[CGJ+00] E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In E.A. Emerson and A.P. Sistla, editors, *Proc. 12th Int. Conf. CAV '00*, Chicago, IL, US, LNCS 1855, pages 154–169. Springer, 15–19 Jul. 2000.

[CGR07] P. Cousot, P. Ganty, and J.-F. Raskin. Fixpoint-guided abstraction refinements. In G. Filé and H. Riis-Nielson, editors, *Proc. 14th Int. Symp. SAS '07*, Kongens Lyngby, DK, LNCS 4634, pages 333–348. Springer, 22–24 Aug. 2007.

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *5th POPL*, pages 84–97, Tucson, AZ, 1978. ACM Press.

[CMC08] L. Chen, A. Miné, and P. Cousot. A sound floating-point polyhedra abstract domain. To appear in The Sixth ASIAN SYMP on Programming Languages and Systems, APLAS 2008, Bangalore, India, 9–11 December, 2008, 2008.

[Cou97] P. Cousot. Types as abstract interpretations, invited paper. In *24th POPL*, pages 316–331, Paris, FR, Jan. 1997. ACM Press.

[Cou00] P. Cousot. Partial completeness of abstract fixpoint checking, invited paper. In B.Y. Choueiry and T. Walsh, editors, *Proc. 4th Int. Symp. SARA '2000*, Horseshoe Bay, TX, US, LNAI 1864, pages 1–25. Springer, 26–29 Jul. 2000.

[Cou02] P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theoret. Comput. Sci.*, 277(1—2):47–103, 2002.

[Cou03] P. Cousot. Verification by abstract interpretation, invited chapter. In N. Dershowitz, editor, *Proc. Int. Symp. on Verification – Theory & Practice – Honoring Zohar Manna's 64th Birthday*, pages 243–268. LNCS 2772, Springer, Taormina, IT, 29 June – 4 Jul. 2003.

[Dan07] V. Danos. Abstract views on biological signaling. In *Mathematical Foundations of Programming Semantics, 23rd Annual Conf. (MFPS XXIII)*, 2007.

[DS07] D. Delmas and J. Souyris. Astrée: from research to industry. In G. Filé and H. Riis-Nielson, editors, *Proc. 14th Int. Symp. SAS '07*, Kongens Lyngby, DK, LNCS 4634, pages 437–451. Springer, 22–24 Aug. 2007.

[Fer04] J. Feret. Static analysis of digital filters. In D. Schmidt, editor, *Proc. 30th ESOP '2004, Barcelona, ES*, volume 2986 of *LNCS*, pages 33–48. Springer, Mar. 27 – Apr. 4, 2004.

[Fer05a] J. Feret. The arithmetic-geometric progression abstract domain. In R. Cousot, editor, *Proc. 6th Int. Conf. VMCAI 2005*, pages 42–58, Paris, FR, 17–19 Jan. 2005. LNCS 3385, Springer.

[Fer05b] J. Feret. Numerical abstract domains for digital filters. In *1st Int. Work. on Numerical & Symbolic Abstract Domains, NSAD"05*, Maison Des Polytechniciens, Paris, FR, 21 Jan. 2005.

[GM04] R. Giacobazzi and I. Mastroeni. Abstract non-interference: Parameterizing non-interference by abstract interpretation. In *31st POPL*, pages 186–197, Venice, IT, 2004. ACM Press.

[Gra89] P. Granger. Static analysis of arithmetical congruences. *Int. J. Comput. Math.*, 30:165–190, 1989.

[GRS00] R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract interpretations complete. *J. ACM*, 47(2):361–416, 2000.

[JP06] Ph. Jorrand and S. Perdrix. Towards a quantum calculus. In *Proc. 4th Int. Work. on Quantum Programming Languages, ENTCS*, 2006.

[Min04a] A. Miné. Relational abstract domains for the detection of floating-point run-time errors. In D. Schmidt, editor, *Proc. 30th ESOP '2004, Barcelona, ES*, volume 2986 of *LNCS*, pages 3–17. Springer, Mar. 27 – Apr. 4, 2004.

[Min04b] A. Miné. *Weakly Relational Numerical Abstract Domains.* Thèse de doctorat en informatique, École polytechnique, Palaiseau, FR, 6 Dec. 2004.

[Min06] A. Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19:31–100, 2006.

[Mon08] D. Monniaux. The pitfalls of verifying floating-point computations. *TOPLAS*, 30(3):Article No. 12, may 2008.

[MR05] L. Mauborgne and X. Rival. Trace partitioning in abstract interpretation based static analyzer. In M. Sagiv, editor, *Proc. 14th ESOP '2005, Edinburg, UK*, volume 3444 of *LNCS*, pages 5–20. Springer, Apr. 2Ñ-10, 2005.

[PCJD07] M. Dalla Preda, M. Christodorescu, S. Jha, and S. Debray. Semantics-based approach to malware detection. In *34th POPL*, pages 238–252, Nice, France, 17–19 Jan. 2007. ACM Press.

[Per06] S. Perdrix. *Modèles formels du calcul quantique : ressources, machines abstraites et calcul par mesure.* PhD thesis, Institut National Polytechnique de Grenoble, Laboratoire Leibniz, 2006.

[RM07] X. Rival and L. Mauborgne. The trace partitioning abstract domain. *TOPLAS*, 29(5), Aug. 2007.

[RT04] F. Ranzato and F. Tapparo. Strong preservation as completeness in abstract interpretation. In D. Schmidt, editor, *Proc. 30th ESOP '04*, volume 2986 of *LNCS*, pages 18–32, Barcelona, ES, Mar. 29 – Apr. 2 2004. Springer.

[RT06] F. Ranzato and F. Tapparo. Strong preservation of temporal fixpoint-based operators by abstract interpretation. In A.E. Emerson and K.S. Namjoshi, editors, *Proc. 7th Int. Conf. VMCAI 2006*, pages 332–347, Charleston, SC, US, 8–10 Jan. 2006. LNCS 3855 , Springer.