Abstract Interpretation and Constraint Programming

Charlotte Truchet[1], Antoine Miné[2]

[1] TASC, LINA (UMR 6241), Université de Nantes, France
[2] Antique, LIENS (UMR 8548), ENS, Paris, France

CPAIOR
May 19th, 2015

Antoine cannot be here in Barcelona, so here he is:

The following is based on joint works with

Marie Pelleau	Frédéric Benhamou	Anicet Bart	Eric Monfroy
			

Outline

1. Introduction
   - AI
   - CP
2. Bringing AI ideas to CP
3. Bringing CP ideas to AI
   - Representing disjunctive information
   - Iterations
4. Analyzing Sound Processes with Constraints
5. Conclusion

NB: in the following, AI means Abstract Interpretation.

Zoom on: Ariane 5, Flight 501

Maiden flight of the Ariane 5 Launcher, 4 June 1996.

Zoom on: Ariane 5, Flight 501

40s after launch...

Zoom on: Ariane 5, Flight 501

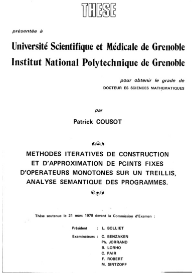
Cause: software error[1]

- arithmetic overflow in unprotected data conversion from 64-bit float to 16-bit integer types[2]

```
P_M_DERIVE(T_ALG.E_BH) :=
  UC_16S_EN_16NS (TDB.T_ENTIER_16S
    ((1.0/C_M_LSB_BH) * G_M_INFO_DERIVE(T_ALG.E_BH)));
```

- software exception not caught
  $\Longrightarrow$ computer switched off
- all backup computers run the same software
  $\Longrightarrow$ all computers switched off, no guidance
  $\Longrightarrow$ rocket self-destructs

Abstract interpretation


THESE

présentée à

Université Scientifique et Médicale de Grenoble
Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR ES SCIENCES MATHEMATIQUES

par

Patrick COUSOT

METHODES ITERATIVES DE CONSTRUCTION
ET D'APPROXIMATION DE PCINTS FIXES
D'OPERATEURS MONOTONES SUR UN TREILLIS,
ANALYSE SEMANTIQUE DES PROGRAMMES.

Thèse soutenue le 21 mars 1978 devant la Commission d'Examen :

Président : L. BOLLIET

Examinateurs : C. BENZAKEN
Ph. JORRAND
B. LORHO
C. PAIR
F. ROBERT
M. SINTZOFF

General theory of the approximation and comparison of program semantics:

- unifies many existing semantics
- allows the definition of new static analyses that are correct by construction

Concrete and abstract semantics

$(\mathcal{S}_0)$
```
assume X in [0,1000];
```
$(\mathcal{S}_1)$
```
I := 0;
```
$(\mathcal{S}_2)$
```
while (S3) I < X do
    (S4)
    I := I + 2;
    (S5)
```
$(\mathcal{S}_6)$

program

Concrete and abstract semantics

```
(S0)
assume X in [0,1000];
(S1)
I := 0;
(S2)
while (S3) I < X do
    (S4)
    I := I + 2;
    (S5)
(S6)
```

program

$\mathcal{S}_i \in \mathcal{D} = \mathcal{P}(\{\mathtt{I},\mathtt{X}\} \to \mathbb{Z})$

$$
\begin{array}{lll}
\mathcal{S}_0 = \{ (i,x) \mid i,x \in \mathbb{Z} \} & & = \top \\
\mathcal{S}_1 = \{ (i,x) \in \mathcal{S}_0 \mid x \in [0,1000] \} & & = F_1(\mathcal{S}_0) \\
\mathcal{S}_2 = \{ (0,x) \mid \exists i,\ (i,x) \in \mathcal{S}_1 \} & & = F_2(\mathcal{S}_1) \\
\mathcal{S}_3 = \mathcal{S}_2 \cup \mathcal{S}_5 & & \\
\mathcal{S}_4 = \{ (i,x) \in \mathcal{S}_3 \mid i < x \} & & = F_4(\mathcal{S}_3) \\
\mathcal{S}_5 = \{ (i+2,x) \mid (i,x) \in \mathcal{S}_4 \} & & = F_5(\mathcal{S}_4) \\
\mathcal{S}_6 = \{ (i,x) \in \mathcal{S}_3 \mid i \geq x \} & & = F_6(\mathcal{S}_3)
\end{array}
$$

semantics

Concrete semantics $\mathcal{S}_i \in \mathcal{D} = \mathcal{P}(\{\mathtt{I},\mathtt{X}\} \to \mathbb{Z})$:

- strongest invariant (and an inductive invariant)
- not computable in general
- smallest solution of a system of equations

Concrete and abstract semantics

$(\mathcal{S}_0)$

```
assume X in [0,1000];
```

$(\mathcal{S}_1)$

```
I := 0;
```

$(\mathcal{S}_2)$

while $(\mathcal{S}_3)$ `I < X do`

  $(\mathcal{S}_4)$

  `I := I + 2;`

  $(\mathcal{S}_5)$

$(\mathcal{S}_6)$

program

$$\mathcal{S}_i^\sharp \in \mathcal{D}^\sharp$$

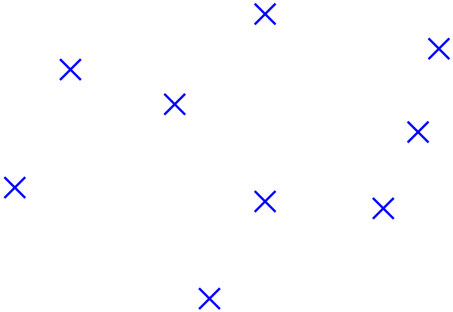
$$\begin{aligned}
\mathcal{S}_0^\sharp &= \top^\sharp \\
\mathcal{S}_1^\sharp &= F_1^\sharp(\mathcal{S}_0^\sharp) \\
\mathcal{S}_2^\sharp &= F_2^\sharp(\mathcal{S}_1^\sharp) \\
\mathcal{S}_3^\sharp &= S_2^\sharp \cup^\sharp \mathcal{S}_5^\sharp \\
\mathcal{S}_4^\sharp &= F_4^\sharp(\mathcal{S}_3^\sharp) \\
\mathcal{S}_5^\sharp &= F_5^\sharp(\mathcal{S}_4^\sharp) \\
\mathcal{S}_6^\sharp &= F_6^\sharp(\mathcal{S}_3^\sharp)
\end{aligned}$$

semantics

Abstract semantics $\mathcal{S}_i^\sharp \in \mathcal{D}^\sharp$:

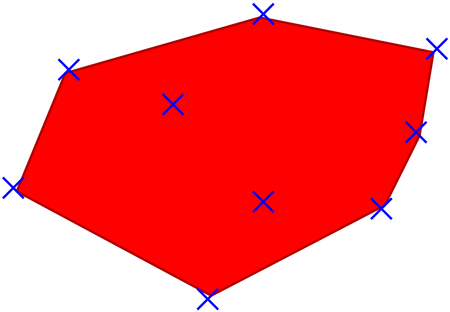
- $\mathcal{D}^\sharp$ is a subset of properties of interest (approximation) with a machine representation
- $F^\sharp : \mathcal{D}^\sharp \to \mathcal{D}^\sharp$ over-approximates the effect of $F : \mathcal{D} \to \mathcal{D}$ in $\mathcal{D}^\sharp$ (with effective algorithms)

Numeric abstract domain examples



concrete sets $\mathcal{D}$: $\{(0,3),(5.5,0),(12,7),\ldots\}$

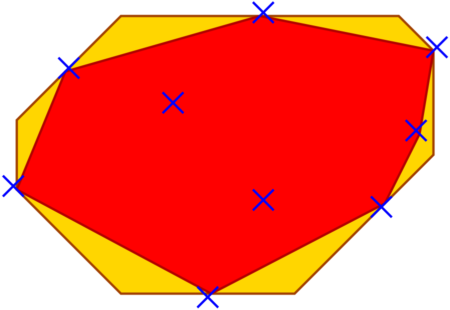
Numeric abstract domain examples



concrete sets $\mathcal{D}$: $\{(0,3),(5.5,0),(12,7),\ldots\}$

abstract polyhedra $\mathcal{D}_p^\sharp$: $6X + 11Y \geq 33 \wedge \cdots$

Numeric abstract domain examples

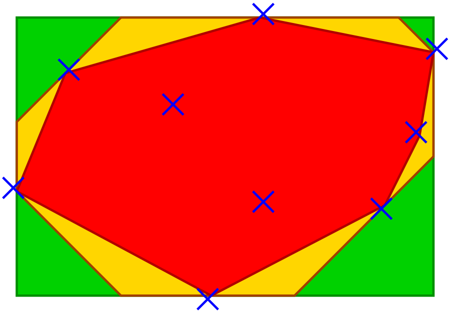


concrete sets $\mathcal{D}$: $\{(0,3),(5.5,0),(12,7),\ldots\}$

abstract polyhedra $\mathcal{D}^\sharp_p$: $6X + 11Y \geq 33 \wedge \cdots$

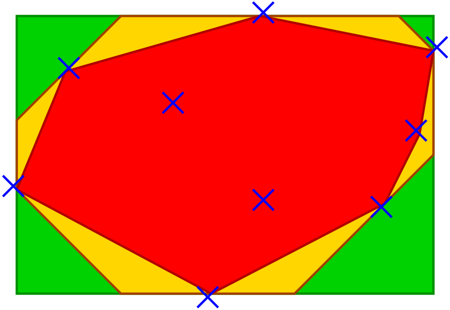
abstract octagons $\mathcal{D}^\sharp_o$: $X + Y \geq 3 \wedge Y \geq 0 \wedge \cdots$

Numeric abstract domain examples



concrete sets $\mathcal{D}$:	$\{(0,3),(5.5,0),(12,7),\ldots\}$
abstract polyhedra $\mathcal{D}^\sharp_p$:	$6X + 11Y \geq 33 \wedge \cdots$
abstract octagons $\mathcal{D}^\sharp_o$:	$X + Y \geq 3 \wedge Y \geq 0 \wedge \cdots$
abstract intervals $\mathcal{D}^\sharp_i$:	$X \in [0,12] \wedge Y \in [0,8]$

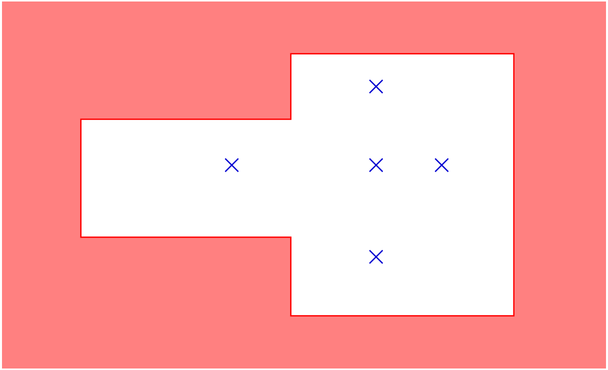
Numeric abstract domain examples



concrete sets $\mathcal{D}$:	$\{(0,3),(5.5,0),(12,7),\dots\}$	not computable
abstract polyhedra $\mathcal{D}_p^\sharp$:	$6X + 11Y \geq 33 \wedge \cdots$	exponential cost
abstract octagons $\mathcal{D}_o^\sharp$:	$X + Y \geq 3 \wedge Y \geq 0 \wedge \cdots$	cubic cost
abstract intervals $\mathcal{D}_i^\sharp$:	$X \in [0,12] \wedge Y \in [0,8]$	linear cost

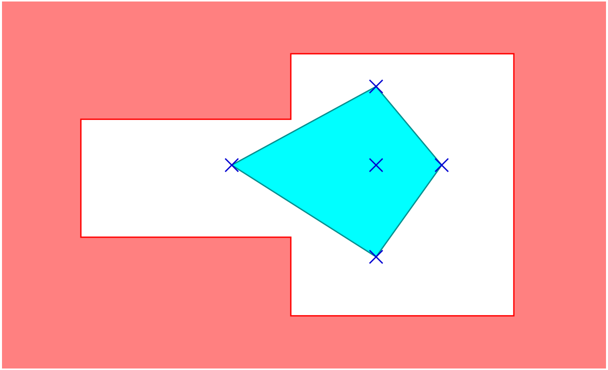
Trade-off between cost and expressiveness / precision

Correctness proof and false alarms



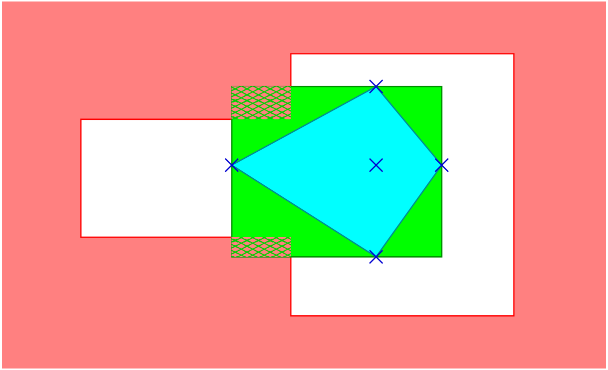
The program is correct (blue $\cap$ red $= \emptyset$).

Correctness proof and false alarms



The program is correct (blue $\cap$ red $= \emptyset$).
The polyhedra domain can prove the correctness (cyan $\cap$ red $= \emptyset$).

Correctness proof and false alarms



The program is correct (blue $\cap$ red $= \emptyset$).
The polyhedra domain can prove the correctness (cyan $\cap$ red $= \emptyset$).
The interval domain cannot (green $\cap$ red $\neq \emptyset$, false alarm).

AI strengths

In the end, AI tools are able to successfully check huge programs for run-time errors:

- primary flight control software of the Airbus A340 (2003), with $132,000$ lines of code,
- electric flight controle code of the Airbus A380 (2004).

What AI does well:

- very fast approximations of the concrete semantics,
- analysis of programs with different types (int, float, bool),
- take into account relations between the variables, with non-cartesian domains,
- have different abstract domains coexist in the same analyzer.

Outline

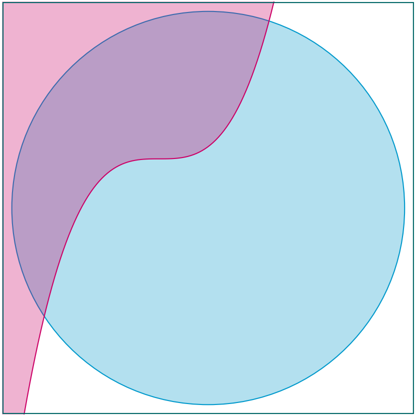
CP on an example

Definition (CSP)

- V: set of variables
- D: set of domains
- C: set of constraints

Example (Continuous)

- $V = (v_1, v_2)$
- $D_1 = [0, 4], D_2 = [0, 4]$
- $C_1 : v_1^2 + v_2^2 \leq 2$
- $C_2 : v_2 > (v_1 + 1)^3 + 0.5$



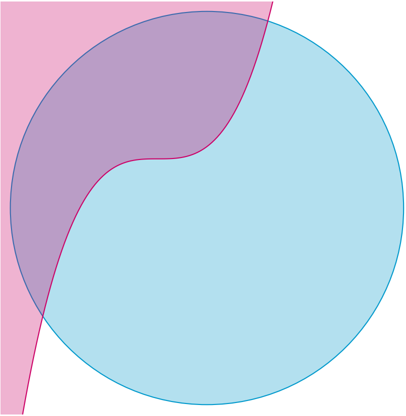
CP on an example

Parameter: float r

```
list of boxes sols ← ∅
queue of boxes toExplore ← ∅
box e

e ← D
push e in toExplore

while toExplore ≠ ∅ do
   e ← pop(toExplore)
   e ← Propagate(e)
   if e ≠ ∅ then
      if maxDim(e) ≤ r or isSol(e)
      then
         sols ← sols U e
      else
         split e in two boxes e1 and
         e2
         push e1 and e2 in toExplore
```



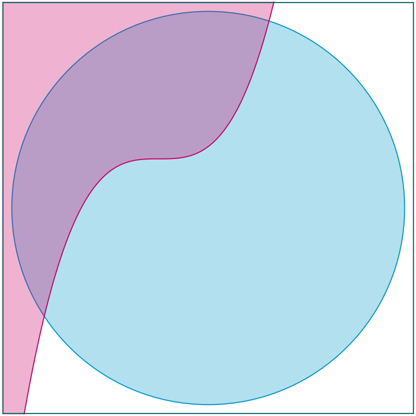
CP on an example

```
Parameter: float r

list of boxes sols ← ∅
queue of boxes toExplore ← ∅
box e

e ← D
push e in toExplore

while toExplore ≠ ∅ do
   e ← pop(toExplore)
   e ← Propagate(e)
   if e ≠ ∅ then
      if maxDim(e) ≤ r or isSol(e)
      then
         sols ← sols U e
      else
         split e in two boxes e1 and
         e2
         push e1 and e2 in toExplore
```



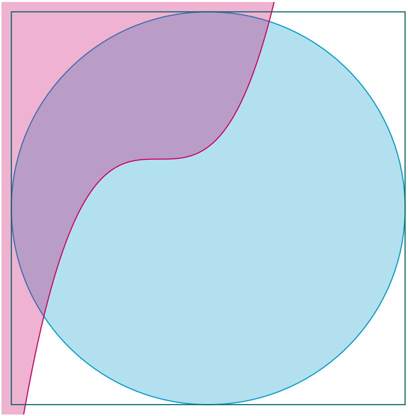
CP on an example

```
Parameter:  float r

list of boxes sols ← ∅
queue of boxes toExplore ← ∅
box e

e ← D
push e in toExplore

while toExplore ≠ ∅ do
  e ← pop(toExplore)
  e ← Propagate(e)
  if e ≠ ∅ then
    if maxDim(e) ≤ r or isSol(e)
    then
      sols ← sols U e
    else
      split e in two boxes e1 and
      e2
      push e1 and e2 in toExplore
```



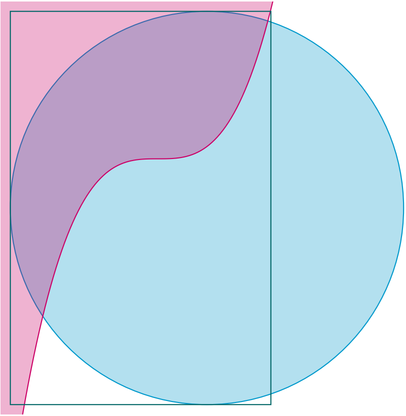
CP on an example

```
Parameter:  float r

list of boxes sols ← ∅
queue of boxes toExplore ← ∅
box e

e ← D
push e in toExplore

while toExplore ≠ ∅ do
   e ← pop(toExplore)
   e ← Propagate(e)
   if e ≠ ∅ then
      if maxDim(e)≤ r or isSol(e)
      then
         sols ← sols U e
      else
         split e in two boxes e1 and
         e2
         push e1 and e2 in toExplore
```



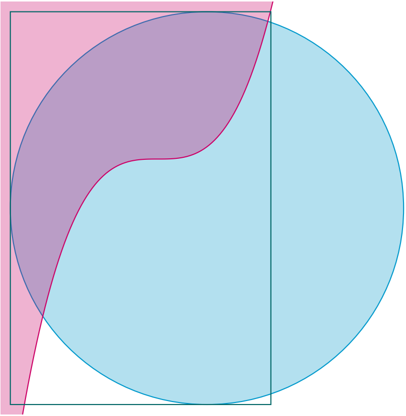
CP on an example

```
Parameter:  float r

list of boxes sols ← ∅
queue of boxes toExplore ← ∅
box e

e ← D
push e in toExplore

while toExplore ≠ ∅ do
  e ← pop(toExplore)
  e ← Propagate(e)
  if e ≠ ∅ then
    if maxDim(e) ≤ r or isSol(e)
    then
      sols ← sols ∪ e
    else
      split e in two boxes e1 and
      e2
      push e1 and e2 in toExplore
```



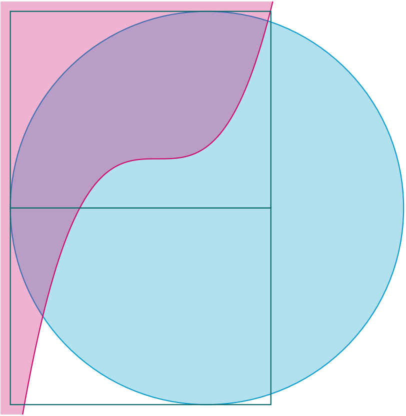
CP on an example

```
Parameter:  float r

list of boxes sols ← ∅
queue of boxes toExplore ← ∅
box e

e ← D
push e in toExplore

while toExplore ≠ ∅ do
   e ← pop(toExplore)
   e ← Propagate(e)
   if e ≠ ∅ then
      if maxDim(e)≤ r or isSol(e)
      then
         sols ← sols U e
      else
         split e in two boxes e1 and e2
         push e1 and e2 in toExplore
```



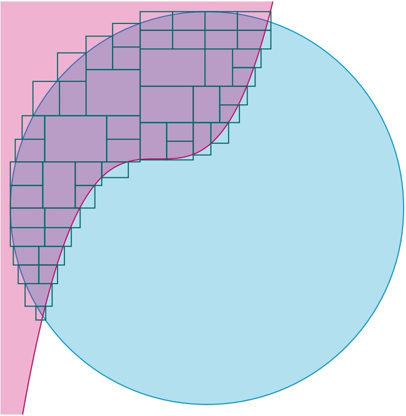
CP on an example

```
Parameter:  float r

list of boxes sols ← ∅
queue of boxes toExplore ← ∅
box e

e ← D
push e in toExplore

while toExplore ≠ ∅ do
   e ← pop(toExplore)
   e ← Propagate(e)
   if e ≠ ∅ then
      if maxDim(e)≤ r or isSol(e)
      then
         sols ← sols U e
      else
         split e in two boxes e1 and
         e2
         push e1 and e2 in toExplore
```



CP strengths and weaknesses

What CP does well

- model many combinatorial problems in a common framework,
- solve problems on either discrete or continuous variables,
- add various heuristics to improve the solving methods.

$\Longrightarrow$ Efficiently solves many combinatorial problems

What CP does not

- take into account the correlation of the variables
  $\Rightarrow$ restricted to Cartesian product
- solve mixed discrete-continuous problems *in an elegant way* (without conversions).

CP and AI ?

Our claim

CP and AI have a lot in common, and the notion of domain is at the core of their connexions.

An example: two algorithms (at least) have been defined on both sides, and called differently:

- HC4 in CP [Benhamou et al., 1999] / bottom-up top-down in AI [Cousot and Cousot, 1977],
- temporal constraints network in CP [Dechter et al., 1989] / improved Floyd-Warshall for octagons in AI [Miné, 2006].

NB: some links between AI and CP have already been highlighted in the literature, for instance on the propagation loop in CP vs the chaotic iterations in AI [Apt, 1999].

Comparison

- Same underlying structure (lattices and fixpoints)
- Same goal: an over-approximation of a desired set
  - Solutions set in CP
  - Sets of program traces in AI
- Different fixpoints and iterative schemes
  - Only decreasing iterations in CP
  - Both decreasing and increasing iterations in AI
- Only the soundness (over-approximation) is guaranteed
- More domains representations in AI than in CP
- AI naturally deals with different domains in the same framework (including many non-numerical domains)

Outline

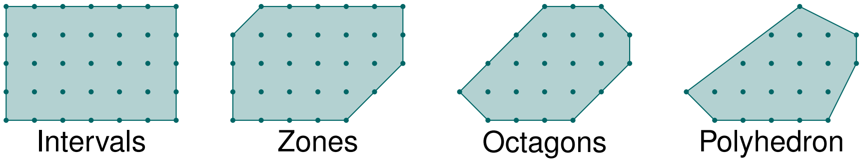
Questions

Can we abstract the notion of domains in CP ?

Will they be the same as AI abstract domains ?

Can we use AI abstract domains in CP ?

What already exist in AI



Abstract domains feature:

- transfer functions $\rho^\sharp$ (assignment, test, ...)
- meet $\cap^\sharp$ and join $\cup^\sharp$
- widening $\nabla^\sharp$ and narrowing $\triangle^\sharp$

We need:

- a consistency
- a choice/splitting operator
- a size function

Abstract Solving Method

We define the resolution as a concrete semantics. Then:

- consistency is defined using transfer function on the constraints,
- propagation loop is defined using local iterations as defined by [Granger, 1992],
- the choice operator is added (in disjunctive completions [Cousot and Cousot, 1992]),
- the size function is added.

Continuous Solving Method

```
Parameter:  float r

list of boxes  sols ← ∅
queue of boxes  toExplore ← ∅
box  e ← D

push e in toExplore

while toExplore ≠ ∅ do
  e ← pop(toExplore)
  e ← Hull-Consistency(e)
  if e ≠ ∅ then
    if maxDim(e)  ≤ r or isSol(e)  then
      sols ← sols ∪ e
    else
      split e in two boxes e1 and e2
      push e1 and e2  in toExplore
```

Abstract Solving Method

```
Parameter:  float r

~~list of boxes~~ disjunction sols ← ∅
~~queue of boxes~~ disjunction toExplore ← ∅
~~box~~ abstract domain e ← ~~D~~ ⊤♯

push e in toExplore

while toExplore ≠ ∅ do
  e ← pop(toExplore)
  e ← ~~Hull-Consistency(e)~~ ρ♯(e)
  if e ≠ ∅ then
    if ~~maxDim(e)~~ τ(e) ≤ r or isSol(e) then
      sols ← sols ∪ e
    else
      ~~split e in two boxes e1 and e2~~
      push ~~e1 and e2~~ ⊕(e) in toExplore
```

Under some conditions on the operators, this abstract solving method terminates, is correct and complete.

Implementation

Prototype with Apron [Jeannet and Miné, 2009], an OCaml library of numerical abstract domains for static analysis

- Consistency: using transfer functions
- Propagation loop: at each iteration, propagate all the constraints
  $\longrightarrow$ Apply all the transfer functions
- Split: only Cartesian split

For the moment, does not feature all of the CP techniques. Still to improve:

- propagation loop,
- abstract splitting operator,
- choice heuristic,

But it **naturally** copes with mixed integer-real problems.

Experiments

Comparison between Absolute and Ibex.

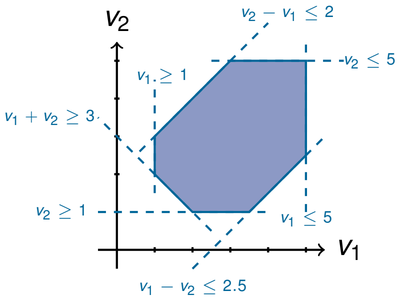
			Itv		Oct	
name	# vars	ctrs	Ibex	AbSolute	Ibex	AbSolute
b	4	$=$	0.02	0.10	0.26	0.14
nbody5.1	6	$=$	95.99	1538.25	27.08	-
ipp	8	$=$	38.83	39.24	279.36	817.86
brent-10	10	$=$	21.58	263.86	330.73	-
KinematicPair	2	$\leq$	59.04	23.14	60.78	31.11
biggsc4	4	$\leq$	800.91	414.94	1772.52	688.56
o32	5	$\leq$	27.36	22.66	40.74	33.17

CPU time in seconds to find all the solutions.
Same solver configuration (octagonal heuristics are unplugged in Absolute).

Octagons

Definition (Octagon [Miné, 2006])

Set of points satisfying a conjunction of constraints of the form $\pm v_i \pm v_j \leq c$, called octagonal constraints

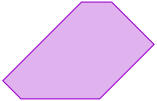


- In dimension n, an octagon has at most $2n^2$ faces
- An octagon can be unbounded
- It can be seen either as a conjunction of octagonal constraints, or as an intersection of boxes.

Octagon abstract domain $\mathcal{O}^\sharp$

Given variables $v_1, \ldots, v_n$, the octagon abstract domain corresponds to

$$\mathcal{O}^\sharp = \left\{ \alpha v_i + \beta v_j \mid i, j \in [\![1, n]\!], \alpha, \beta \in \{-1, 1\} \right\} \to \mathbb{F}$$

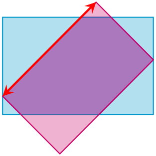


Octagon abstract domain $\mathcal{O}^\sharp$

Given variables $v_1, \ldots, v_n$, the octagon abstract domain corresponds to

$$\mathcal{O}^\sharp = \left\{ \alpha v_i + \beta v_j \,|\, i, j \in [\![1, n]\!], \alpha, \beta \in \{-1, 1\} \right\} \to \mathbb{F}$$

$$\begin{aligned} \tau_o(X^\sharp) = \min(& \max_{i,j,\beta} \left(X^\sharp(v_i + \beta v_j) + X^\sharp(-v_i - \beta v_j) \right), \\ & \max_{i} \left(X^\sharp(v_i + v_i) + X^\sharp(-v_i - v_i) \right) / 2 \quad) \end{aligned}$$



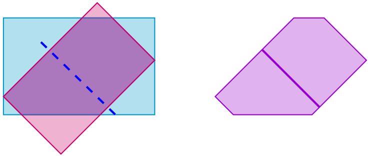
Octagon abstract domain $\mathcal{O}^\sharp$

Given variables $v_1, \ldots, v_n$, the octagon abstract domain corresponds to

$$\mathcal{O}^\sharp = \left\{ \alpha v_i + \beta v_j \mid i,j \in [\![1,n]\!], \alpha,\beta \in \{-1,1\} \right\} \to \mathbb{F}$$

$$\tau_o(X^\sharp) = \min\Big(\max_{i,j,\beta} \left(X^\sharp(v_i + \beta v_j) + X^\sharp(-v_i - \beta v_j)\right), \quad \max_{i} \left(X^\sharp(v_i + v_i) + X^\sharp(-v_i - v_i)\right)/2 \Big)$$

$$\oplus_o(X^\sharp) = \left\{ X^\sharp\left[(\alpha v_i + \beta v_j) \mapsto h\right], X^\sharp\left[(-\alpha v_i - \beta v_j) \mapsto -h\right] \right\}$$



Octagon abstract domain $\mathcal{O}^\sharp$

Given variables $v_1, \ldots, v_n$, the octagon abstract domain corresponds to

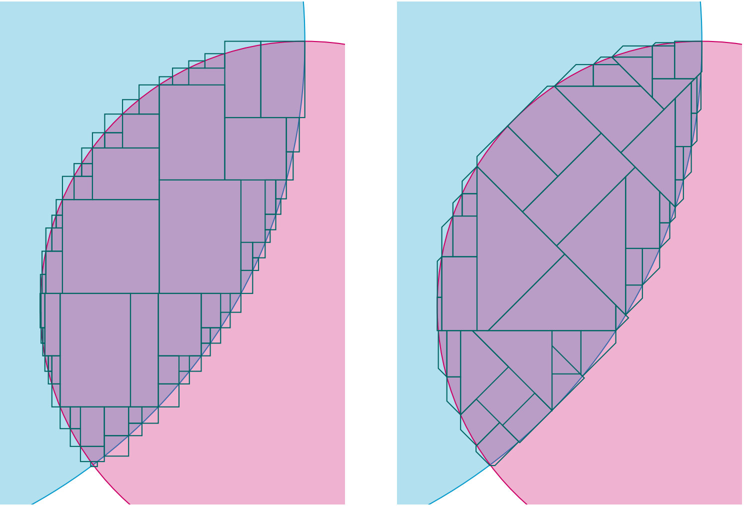
$$\mathcal{O}^\sharp = \left\{ \alpha v_i + \beta v_j \,|\, i, j \in [\![1, n]\!], \alpha, \beta \in \{-1, 1\} \right\} \to \mathbb{F}$$

$$\begin{aligned}\tau_o(X^\sharp) = \min(\; & \max_{i,j,\beta} \left(X^\sharp(v_i + \beta v_j) + X^\sharp(-v_i - \beta v_j) \right), \\ & \max_{i} \left(X^\sharp(v_i + v_i) + X^\sharp(-v_i - v_i) \right) /2 \quad)\end{aligned}$$

$$\oplus_o(X^\sharp) = \left\{ X^\sharp \left[(\alpha v_i + \beta v_j) \mapsto h \right], X^\sharp \left[(-\alpha v_i - \beta v_j) \mapsto -h \right] \right\}$$

In practice, consistency is computed by interleaving Floyd-Warshall (for the octagonal constraints) and the usual constraint propagation on all the rotated boxes.

Output



Same problem with the same time limit.

Beautiful slide by courtesy of Marie Pelleau

Experiments

Comparison of an ad-hoc implentation of the same solving algorithm, using either the octagon abstract domain or the intervals.

			First solution		All the solutions	
name	nbvar	ctrs	$\mathbb{I}^n$	Oct	$\mathbb{I}^n$	Oct
h75	5	$\leq$	41.40	0.03	-	-
hs64	3	$\leq$	0.01	0.05	-	-
h84	5	$\leq$	5.47	2.54	-	7238.74
KinematicPair	2	$\leq$	0.00	0.00	53.09	16.56
pramanik	3	$=$	28.84	0.16	193.14	543.46
trigo1	10	$=$	18.93	1.38	20.27	28.84
brent-10	10	$=$	6.96	0.54	17.72	105.02
h74	5	$=$ $\leq$	305.98	13.70	1 304.23	566.31
fredtest	6	$=$ $\leq$	3 146.44	19.33	-	-

Solver: Ibex [Chabert and Jaulin, 2009].
Problems from the COCONUT benchmark.
CPU time in seconds, TO 3 hours.

Why octagons work ?

From a CP point of view, octagons allow us to infer constraints, in a restricted, reasonably tractable language ($O(n^3)$).
For more details see Marie Pelleau's papers at CP2011, VMCAI 2013 or in Constraints.

Could it be generalized ?

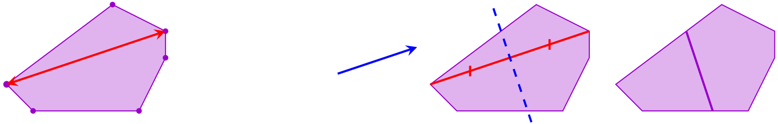
Other abstract domains

Work in progress...

Polyhedra abstract domain $\mathcal{P}^\sharp$

$$\tau_p(X^\sharp) = \max_{g_i, g_j \in X^\sharp} ||g_i - g_j||$$

$$\oplus_p(X^\sharp) = \left\{ X^\sharp \cup \left\{ \sum_i \beta_i v_i \leq h \right\}, X^\sharp \cup \left\{ \sum_i \beta_i v_i \geq h \right\} \right\}$$



Outline

Disjunctive properties

Both AI and CP construct complex properties
by disjunctions of simpler ones

In CP:

- complex shapes are tightly covered with boxes

In AI:

- abstract domains can generally express only convex sets
  conjunctions of constraints, such as intervals or polyhedra
- program analysis often requires non-convex properties
  such as $X \neq 0$

$\Longrightarrow$ disjunctive completion: use sets of intervals

Disjunctive analysis: example

Example

if (X$\geq$ 0 && X $<$ 10) B = 1; else B = 0;
$\star$
...
if (B == 1) $\bullet$ A[X] = 0;

we must prove that $0 \leq X < 10$ at $\bullet$.

Plain interval analysis: one box at each program point

at $\star$ we must join $(B \in [1,1], X \in [0,9])$ and $(B \in [0,0], X \in [-\infty, +\infty])$
to get $(B \in [0,1], X \in [-\infty, +\infty])$
$\Longrightarrow$ at $\bullet$, $B == 1$ gives no information an X!

With disjunctive completion: keep several boxes at each control point

by avoiding (or delaying) the abstract join at $\star$
$\Longrightarrow$ at $\bullet$, $B == 1$ recovers $X \in [0,9]$

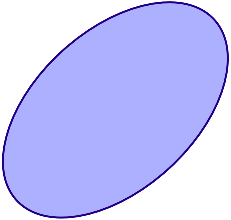
This works well because the disjunction can be guided by the control flow:
each disjunct corresponds to a branch of the first if

Control-free programs

What happens when there is no explicit control flow?

Example: digital filter

```
while true do
   r = 1.5 × s0 - 0.7 × s1 + input [-0.1,0.1];
   s1 = s0;    s0 = r;
done
```



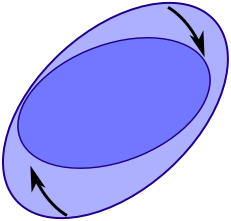
In this example, the reachable states (s0,s1) form an ellipsoid.

no if-then-else, no join operation to create additional boxes
$\Longrightarrow$ even with disjunctive completion, AI will use a single box

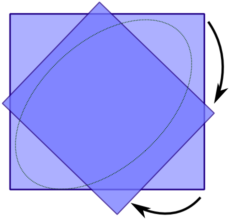
This will not work (see next slide)

Control-free programs: limitations of boxes

When searching for a valid approximation,
AI searches for an inductive invariant:
i.e., a shape X that is stable by a loop iteration $F(X) \subseteq X$



There is a stable ellipsoid



No single box is stable

$\Longrightarrow$ the analysis with boxes will fail

Standard AI solution:
abandon boxes
make a new abstract domain representing directly ellipsoids
(hard work, that must be redone for every shape)

Towards more powerful disjunctive representations

CP knows naturally how to approximate an ellipsoid with a set of boxes to an arbitrary precision criterion

idea: can we use CP to avoid designing an ellipsoid domain?

Challenges:

- new precision criterion:
  the boxes must be tight enough to form an inductive set
- no control-flow to guide the disjunction

Outline

Fixpoint computations

In AI, the semantic problem is expressed as a fixpoint
(generally, a least fixpoint)

Example

```
x = 0;
while x < 100 do
   x += 2;
done
```

Interval analysis:

searching for an interval loop invariant i for x

$i = \mathrm{lfp}\, F$
$F(x) = [0,0] \sqcup ((x \sqcap [-\infty, 99]) \oplus [2,2])$

$\sqcup$, $\sqcap$, $\oplus$ are $\cup$, $\cap$, $+$ in the interval domain

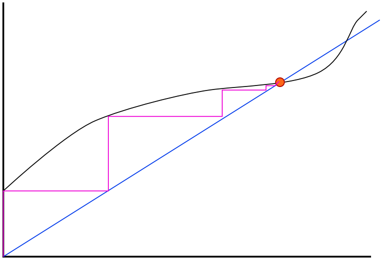
$\Longrightarrow$ we must over-approximate least fixpoints

classic technique:

- increasing iterations: from $\emptyset$, iterate F
  use extrapolation $\triangledown$ to finish in finite time
  $\Longrightarrow$ we obtain a rough over-approximation
- decreasing iterations to refine the approximation

(explained in the following slides)

Increasing iterations in AI

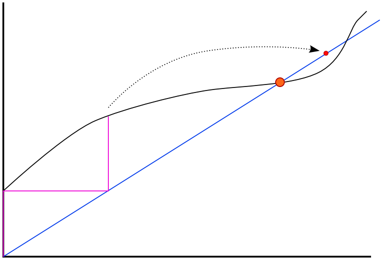


$F(x) = [0,0] \sqcup ((x \sqcap [-\infty, 99]) \oplus [2,2])$

the least fixpoint is: $\mathrm{lfp}\, F = [0, 101]$

the iterates are: $\emptyset$, $[0,0]$, $[0,2]$, $[0,4]$, $\ldots$, $[0,98]$, $[0,100]$, $[0,101]$

Increasing iterations with extrapolation in AI



$F(x) = [0,0] \sqcup ((x \sqcap [-\infty, 99]) \oplus [2,2])$

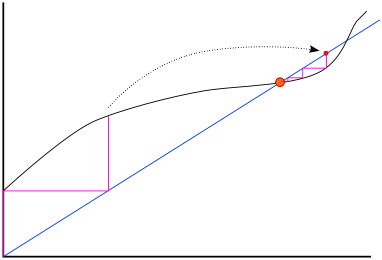
the least fixpoint is: $\operatorname{lfp} F = [0, 101]$

the iterates with extrapolation are: $\emptyset$, $[0,0]$, $[0,2]$, $[0,+\infty]$

unstable bounds are set to $+\infty$

$\Longrightarrow$ over-approximates $[0, 101]$, but coarse

Decreasing iterations in AI



$$F(x) = [0,0] \sqcup ((x \sqcap [-\infty, 99]) \oplus [2,2])$$

the least fixpoint approximation is: $[0, +\infty]$

the gain precision, we continue iterating
the iterates are now decreasing towards the fixpoint

the decreasing iterates are: $[0, +\infty]$, $[0, 101]$

Decreasing iterations in AI

Possible issues:

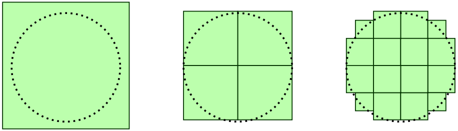
- decreasing sequence may be too slow (or non-terminating)
  $\Longrightarrow$ stop it short
- AI has narrowing operators to extrapolate decreasing sequences but they often fail
- if the extrapolation during increasing iteration is too coarse we may never be able to recover enough precision

  if we jump above a non-least fixpoint, we will stay above it
  and never reach the least fixpoint

How CP might help AI iterations

CP solving can be seen as an iteration sequence

- decreasing iterations
- can approach the fixpoint form above with arbitrary precision



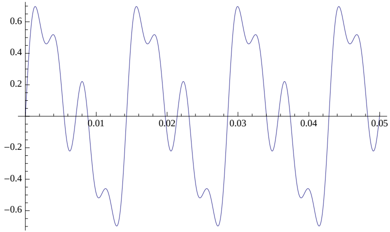
Could we use CP to:

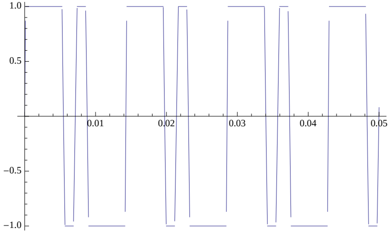
- make more precise decreasing iterations?
- in particular, split during the iteration
- adapt it to the increasing iteration as well?

Outline

Sound processes (ongoing work !)

Our goal: prove that sound processes do not produce saturated sounds.



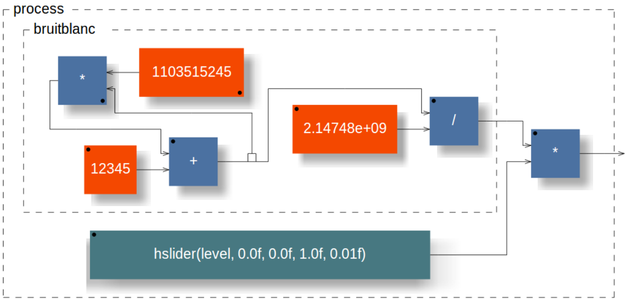


Faust

- Faust is a Domain-Specific Language for real-time signal processing and synthesis (like Csound, Max/MSP, Supercollider, Puredata,. . .).
- Faust is used on stage for concerts and artistic productions, for education and research, for open-source projects and commercial applications.
- `http://faust.grame.fr`

Faust

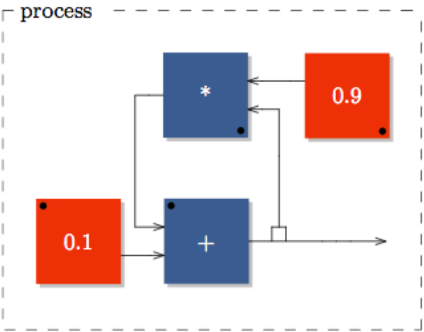
```
process= bruitblanc * hslider("level",0,0,1,0.01);
bruitblanc = +(12345) ~ *(1103515245) :  /(2147483647.0);
```



$$y[n] = x[n]/2147483647.0 * l[n]$$
$$x[n] = 12345 + x[n-1] * 1103515245$$
$$l[n] \in [0..1] \quad UI\ 'level'\ slider$$

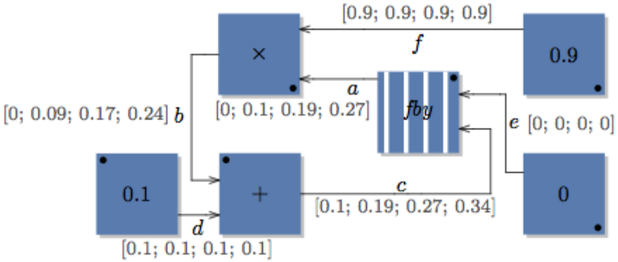
Analyzing sound processes

Faust comes with a formal semantic based on block-diagram. All the variables are infinite streams over the reals.



Analyzing sound processes

We first rewrite this BD in order to identify non-functional dependencies on the streams (*fby* instructions / temporal dependencies).



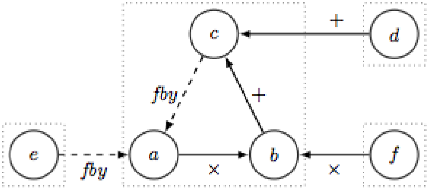
Analyzing sound processes

We abstract time, replacing the streams by an envelope of their possible values, and generate a constraint problem on real intervals.

$$\begin{array}{ll}
a := [\underline{\text{fby}}](e,c) & d := [0.1] \\
b := [\times](a,f) & e := [0] \\
c := [+](b,d) & f := [0.9]
\end{array}$$

Analyzing sound processes

The we build the graph of these dependencies, which is used as a basis to propagate the constraints.



Analyzing sound processes

Finally, the system is solved by an *ad hoc* algorithm that:

- propagates the functional dependencies,
- randomly, but cleverly, jumps over the fixpoints in order to approximate the least fixpoint for loops.

See Anicet Bart's paper at JFPC 2015 for more details.

Tests

program	♯ blocks (♯ fby)	time avg	distance avg	♯ blocks evaluations min	♯ blocks evaluations avg	♯ blocks evaluations max
SIMPLE-ECHO	4 (1)	1ms	$< 0,001$	4	4	4
SIMPLE-COUNTER	3 (2)	11ms	0	3604	3620	3635
SIMPLE-SINUS	4 (2)	10ms	0	3595	3619	3637
PAPER-EXAMPLE	4 (2)	16ms	$< 0,001$	3169	3214	3260
FAUST-NOISE	6 (2)	1ms	0	115	115	115
FAUST-VOLUME	8 (2)	21ms	0	4153	4249	4318
FAUST-ECHO	16 (2)	14ms	0	3156	3170	3182
FAUST-OSC	28 (7)	31ms	$< 0,001$	7791	7871	7916
FAUST-FREEVERB	237 (104)	0,51s	0	48348	48356	48360
FAUST-KARPLUS32	530 (133)	0,69s	0	102813	102828	102842

Conclusion

Conclusion

By relying on the common notion of domains, we can combine the strengths of both AI and CP:

- CP can be precise,
- AI can have different types and adapt the domains to the problems.

Further research

- improve the CP features of Absolute: global constraints, heuristics,
- adapt the abstract domains to the constraints,
- ...

There is a lot to be done !

Play with us !

To try AI numerical domains, try the Interproc toy language, which uses Apron:
`http://pop-art.inrialpes.fr/interproc/interprocweb.cgi`

There is no webpage for Absolute for now, but we would be happy to share the code. Just send us an email !

Apt, K. R. (1999).
The essence of constraint propagation.
Theoretical Computer Science, 221.

Benhamou, F., Goualard, F., Granvilliers, L., and Puget, J.-F. (1999).
Revisiting hull and box consistency.
In Proceedings of the 16th International Conference on Logic Programming, pages 230–244.

Chabert, G. and Jaulin, L. (2009).
Contractor programming.
Artificial Intelligence, 173:1079–1100.

Cousot, P. and Cousot, R. (1977).
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, pages 238–252, Los Angeles, California. ACM Press, New York, NY.

Cousot, P. and Cousot, R. (1992).
Abstract interpretation frameworks.
Journal of Logic and Computation, 2(4):511–547.

Dechter, R., Meiri, I., and Pearl, J. (1989).
Temporal constraint networks.
In Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning.

Granger, P. (1992).
Improving the results of static analyses of programs by local decreasing iterations.
In Proceedings of the 12th Conference on Foundations of Software Technology and Theoretical Computer Science.

Jeannet, B. and Miné, A. (2009).
Apron: A library of numerical abstract domains for static analysis.

In Proceedings of the 21th International Conference Computer Aided Verification (CAV 2009), volume 5643 of Lecture Notes in Computer Science, pages 661–667. Springer.

Miné, A. (2006).
The octagon abstract domain.
Higher-Order and Symbolic Computation, 19(1):31–100.