Abstract Interpretation and Constraint Programming

Charlotte Truchet\textsuperscript{1}, Antoine Miné\textsuperscript{2}

\textsuperscript{1} TASC, LINA (UMR 6241), Université de Nantes, France
\textsuperscript{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

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

\[
P_M\_DERIVE(T\_ALG.E\_BH) := \\
UC\_16S\_EN\_16NS(TDB.T\_ENTIER\_16S((1.0/C_M\_LSB\_BH) * G_M\_INFO\_DERIVE(T\_ALG.E\_BH))); \\
\]

- software exception not caught
  \[\rightarrow\] computer switched off

- all backup computers run the same software
  \[\rightarrow\] all computers switched off, no guidance
  \[\rightarrow\] rocket self-destructs
Abstract interpretation

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

\( (S_0) \)
assume \( X \) in \([0,1000]\);

\( (S_1) \)
\( I := 0; \)

\( (S_2) \)
while \( (S_3) \) I < X do
  \( (S_4) \)
  \( I := I + 2; \)

\( (S_5) \)

\( (S_6) \)
program
Concrete and abstract semantics

\[(S_0)\]
assume \(X\) in \([0,1000]\);
\[(S_1)\]
\(I := 0;\)
\[(S_2)\]
while \((S_3)\) \(I < X\) do
\[(S_4)\]
\(I := I + 2;\)
\[(S_5)\]
\[(S_6)\]
program semantics

Concrete semantics \(S_i \in D = \mathcal{P}(\{I, X\} \rightarrow \mathbb{Z})\):

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

\[S_i \in D = \mathcal{P}(\{I, X\} \rightarrow \mathbb{Z})\]
\[S_0 = \{ (i, x) \mid i, x \in \mathbb{Z} \} = \top\]
\[S_1 = \{ (i, x) \in S_0 \mid x \in [0, 1000] \} = F_1(S_0)\]
\[S_2 = \{ (0, x) \mid \exists i, (i, x) \in S_1 \} = F_2(S_1)\]
\[S_3 = S_2 \cup S_5\]
\[S_4 = \{ (i, x) \in S_3 \mid i < x \} = F_4(S_3)\]
\[S_5 = \{ (i + 2, x) \mid (i, x) \in S_4 \} = F_5(S_4)\]
\[S_6 = \{ (i, x) \in S_3 \mid i \geq x \} = F_6(S_3)\]
Concrete and abstract semantics

\[(S_0)\]
assume \(X\) in \([0,1000]\);
\[(S_1)\]
\(I := 0;\)
\[(S_2)\]
while \((S_3)\) \(I < X\) do
\[(S_4)\]
\(I := I + 2;\)
\[(S_5)\]
program semantics

\[(S_6)\]

Abstract semantics \(S_i^\# \in \mathcal{D}^\#\):

- \(\mathcal{D}^\#\) is a subset of properties of interest with a machine representation (approximation)
- \(F^\# : \mathcal{D}^\# \rightarrow \mathcal{D}^\#\) over-approximates the effect of \(F : \mathcal{D} \rightarrow \mathcal{D}\) in \(\mathcal{D}^\#\) (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 $\mathcal{D}_p^\#$: $6X + 11Y \geq 33 \land \cdots$
Numeric abstract domain examples

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

- Abstract polyhedra $\mathcal{D}_p^\#$:
  $6X + 11Y \geq 33 \land \ldots$

- Abstract octagons $\mathcal{D}_o^\#$:
  $X + Y \geq 3 \land Y \geq 0 \land \ldots$
Numeric abstract domain examples

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

Abstract polyhedra $\mathcal{D}_p^\#: 6X + 11Y \geq 33 \wedge \ldots$

Abstract octagons $\mathcal{D}_o^\#: X + Y \geq 3 \wedge Y \geq 0 \wedge \ldots$

Abstract intervals $\mathcal{D}_i^\#: X \in [0, 12] \wedge Y \in [0, 8]$
Numeric abstract domain examples

Concrete sets $\mathcal{D}$: 
\[ \{(0, 3), (5.5, 0), (12, 7), \ldots \} \]
not computable

Abstract polyhedra $\mathcal{D}_p^\#$: 
\[ 6X + 11Y \geq 33 \land \cdots \]
exponential cost

Abstract octagons $\mathcal{D}_o^\#$: 
\[ X + Y \geq 3 \land Y \geq 0 \land \cdots \]
cubic cost

Abstract intervals $\mathcal{D}_i^\#$: 
\[ X \in [0, 12] \land Y \in [0, 8] \]
linear cost

Trade-off between cost and expressiveness / precision
Correctness proof and false alarms

The program is **correct** \((\text{blue} \cap \text{red} = \emptyset)\).
The program is **correct** \((\text{blue} \cap \text{red} = \emptyset)\).

The polyhedra domain **can prove the correctness** \((\text{cyan} \cap \text{red} = \emptyset)\).
Correctness proof and false alarms

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

1. Introduction
   - AI
   - CP

2. Bringing AI ideas to CP

3. Bringing CP ideas to AI

4. Analyzing Sound Processes with Constraints

5. Conclusion
CP on an example

Definition (CSP)
- $V$: set of variables
- $D$: set of domains
- $C$: set of constraints

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

- list of boxes sols $\leftarrow \emptyset$
- queue of boxes toExplore $\leftarrow \emptyset$
- box e

$e \leftarrow D$

**push e in toExplore**

**while** toExplore $\neq \emptyset$ **do**

- $e \leftarrow \text{pop(toExplore)}$
- $e \leftarrow \text{Propagate(e)}$

**if** $e \neq \emptyset$ **then**

- **if** $\text{maxDim}(e) \leq r$ **or** isSol(e) **then**
  - sols $\leftarrow$ sols $\cup$ 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 \( \text{sols} \leftarrow \emptyset \)
- queue of boxes \( \text{toExplore} \leftarrow \emptyset \)
- box \( e \)

\[ e \leftarrow D \]

**push** \( e \) in \( \text{toExplore} \)

**while** \( \text{toExplore} \neq \emptyset \) **do**

- \( e \leftarrow \text{pop}(\text{toExplore}) \)
- \( e \leftarrow \text{Propagate}(e) \)

**if** \( e \neq \emptyset \) **then**

- **if** \( \text{maxDim}(e) \leq r \) \text{or} \( \text{isSol}(e) \) **then**
  - \( \text{sols} \leftarrow \text{sols} \cup e \)
- **else**
  - split \( e \) in two boxes \( e_1 \) \text{and} \( e_2 \)
  - **push** \( e_1 \) \text{and} \( e_2 \) in \( \text{toExplore} \)
CP on an example

**Parameter:** float \( r \)

list of boxes \( \text{sols} \) ← ∅

queue of boxes toExplore ← ∅

box \( e \)

e ← \text{D}

push \( e \) in toExplore

while toExplore ≠ ∅ do

\( e \) ← \text{pop}(\text{toExplore})

\( e \) ← Propagate(\( e \))

if \( e \) ≠ ∅ then

if \( \text{maxDim}(e) \leq r \) or isSol(\( e \)) then

sols ← sols \( \cup \) \( e \)

else

split \( e \) in two boxes \( e_1 \) and \( e_2 \)

push \( e_1 \) and \( e_2 \) in toExplore
**Parameter:** float \( r \)

<table>
<thead>
<tr>
<th>Parameter:</th>
<th>float ( r )</th>
</tr>
</thead>
<tbody>
<tr>
<td>list of boxes sols</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>queue of boxes toExplore</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>box e</td>
<td>( D )</td>
</tr>
<tr>
<td>e</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>push e in toExplore</td>
<td>( D )</td>
</tr>
<tr>
<td>while toExplore ( \neq \emptyset ) do</td>
<td>( D )</td>
</tr>
<tr>
<td>e</td>
<td>pop(toExplore)</td>
</tr>
<tr>
<td>e</td>
<td>Propagate(e)</td>
</tr>
<tr>
<td>if e ( \neq \emptyset ) then</td>
<td>( D )</td>
</tr>
<tr>
<td>if maxDim(e) ( \leq ) r or isSol(e)</td>
<td>( D )</td>
</tr>
<tr>
<td>then</td>
<td>( D )</td>
</tr>
<tr>
<td>sols</td>
<td>sols ( \cup ) e</td>
</tr>
<tr>
<td>else</td>
<td>( D )</td>
</tr>
<tr>
<td>split e in two boxes e1 and e2</td>
<td>( D )</td>
</tr>
<tr>
<td>push e1 and e2 in toExplore</td>
<td>( D )</td>
</tr>
</tbody>
</table>
Parameter: float $r$

list of boxes $\text{sols} \leftarrow \emptyset$

queue of boxes $\text{toExplore} \leftarrow \emptyset$

box $e$

$e \leftarrow D$

**push** $e$ in $\text{toExplore}$

**while** $\text{toExplore} \neq \emptyset$ **do**

$e \leftarrow \text{pop}(\text{toExplore})$

$e \leftarrow \text{Propagate}(e)$

if $e \neq \emptyset$ **then**

if $\text{maxDim}(e) \leq r$ or $\text{isSol}(e)$ **then**

$sols \leftarrow \text{sols} \cup e$

else

split $e$ in two boxes $e1$ and $e2$

**push** $e1$ and $e2$ in $\text{toExplore}$
Parameter: float r

list of boxes sols $\leftarrow \emptyset$
queue of boxes toExplore $\leftarrow \emptyset$
box e

e $\leftarrow$ D
push e in toExplore

while toExplore $\neq \emptyset$ do
  e $\leftarrow$ pop(toExplore)
e $\leftarrow$ Propagate(e)
if e $\neq \emptyset$ then
  if maxDim(e) $\leq$ r or isSol(e) then
    sols $\leftarrow$ sols $\cup$ e
  else
    split e in two boxes e1 and e2
    push e1 and e2 in toExplore
  end if
end if
end while
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 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.

⇒ Efficiently solves many combinatorial problems

What CP does not

- take into account the correlation of the variables
  ⇒ restricted to Cartesian product
- solve mixed discrete-continuous problems in an elegant way (without conversions).
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],

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

1. Introduction
2. Bringing AI ideas to CP
3. Bringing CP ideas to AI
4. Analyzing Sound Processes with Constraints
5. Conclusion
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 $\rho^\#$ (assignment, test, ...)
- meet $\cap^\#$ and join $\cup^\#$
- widening $\vee^\#$ and narrowing $\triangle^\#$

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

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

Parameter: float \( r \)

\[
\begin{align*}
\text{list of boxes disjunction } & \text{sols } \leftarrow \emptyset \\
\text{queue of boxes disjunction } & \text{toExplore } \leftarrow \emptyset \\
\text{box abstract domain } & e \leftarrow D \uparrow \# \\
\end{align*}
\]

\text{push } e \text{ in toExplore}

\text{while } \text{toExplore } \neq \emptyset \text{ do}

\[
\begin{align*}
e & \leftarrow \text{pop(toExplore)} \\
e & \leftarrow \text{Hull-Consistency}(e) \rho^+(e) \\
\text{if } e \neq \emptyset \text{ then}
\end{align*}
\]

\[
\begin{align*}
\text{if } \text{maxDim}(e) \tau(e) \leq r \text{ or } \text{isSol}(e) \text{ then}
\end{align*}
\]

\[
\begin{align*}
sols & \leftarrow sols \cup e \\
\text{else}
\end{align*}
\]

\[
\begin{align*}
\text{split } e \text{ in two boxes } e_1 \text{ and } e_2 \\
\text{push } e_1 \text{ and } e_2 \oplus(e) \text{ in toExplore}
\end{align*}
\]

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
  \[\rightarrow\] 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.

<table>
<thead>
<tr>
<th>name</th>
<th># vars</th>
<th>ctrs</th>
<th>Itv</th>
<th>Oct</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Ibex</td>
<td>AbSolute</td>
</tr>
<tr>
<td>b</td>
<td>4</td>
<td>=</td>
<td>0.02</td>
<td>0.10</td>
</tr>
<tr>
<td>nbody5.1</td>
<td>6</td>
<td>=</td>
<td>95.99</td>
<td>1538.25</td>
</tr>
<tr>
<td>ipp</td>
<td>8</td>
<td>=</td>
<td>38.83</td>
<td>39.24</td>
</tr>
<tr>
<td>brent-10</td>
<td>10</td>
<td>=</td>
<td>21.58</td>
<td>263.86</td>
</tr>
<tr>
<td>KinematicPair</td>
<td>2</td>
<td>≤</td>
<td>59.04</td>
<td>23.14</td>
</tr>
<tr>
<td>biggsc4</td>
<td>4</td>
<td>≤</td>
<td>800.91</td>
<td>414.94</td>
</tr>
<tr>
<td>o32</td>
<td>5</td>
<td>≤</td>
<td>27.36</td>
<td>22.66</td>
</tr>
</tbody>
</table>

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 $\mathcal{O}^\#$

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

$$\mathcal{O}^\# = \{ \alpha v_i + \beta v_j \mid i, j \in [1, n], \alpha, \beta \in \{-1, 1\} \} \rightarrow \mathbb{F}$$
Octagon abstract domain $\mathcal{O}^\#$

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

$$\mathcal{O}^\# = \{ \alpha v_i + \beta v_j | i, j \in [1, n], \alpha, \beta \in \{-1, 1\} \} \rightarrow \mathbb{F}$$

$$\tau_o(X^\#) = \min \left( \max_{i, j, \beta} (X^\#(v_i + \beta v_j) + X^\#(-v_i - \beta v_j)) , \right.$$ 
$$\left. \max_i (X^\#(v_i + v_i) + X^\#(-v_i - v_i)) / 2 \right)$$
Octagon abstract domain $\mathcal{O}^\#$

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

$$
\mathcal{O}^\# = \{ \alpha v_i + \beta v_j \mid i, j \in [1, n], \alpha, \beta \in \{-1, 1\} \} \rightarrow \mathbb{F}
$$

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

$$
\oplus_o(X^\#) = \left\{ X^\# \left[ (\alpha v_i + \beta v_j) \mapsto h \right], X^\# \left[ (-\alpha v_i - \beta v_j) \mapsto -h \right] \right\}
$$
Octagon abstract domain $\mathcal{O}^\#$

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

$$\mathcal{O}^\# = \{ \alpha v_i + \beta v_j | i, j \in [1, n], \alpha, \beta \in \{-1, 1\} \} \rightarrow \mathbb{F}$$

$$\tau_o(X^\#) = \min(\max_{i,j,\beta} (X^\#(v_i + \beta v_j) + X^\#(-v_i - \beta v_j)), \max_i (X^\#(v_i + v_i) + X^\#(-v_i - v_i))/2)$$

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

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

Beautiful slide by courtesy of Marie Pelleau
Experiments

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

<table>
<thead>
<tr>
<th>name</th>
<th>nbvar</th>
<th>ctrs</th>
<th>First solution</th>
<th>All the solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>$\mathbb{I}^n$</td>
<td>Oct</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\mathbb{I}^n$</td>
</tr>
<tr>
<td>h75</td>
<td>5</td>
<td>$\leq$</td>
<td>41.40</td>
<td>0.03</td>
</tr>
<tr>
<td>hs64</td>
<td>3</td>
<td>$\leq$</td>
<td>0.01</td>
<td>0.05</td>
</tr>
<tr>
<td>h84</td>
<td>5</td>
<td>$\leq$</td>
<td>5.47</td>
<td>2.54</td>
</tr>
<tr>
<td>KinematicPair</td>
<td>2</td>
<td>$\leq$</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>pramanik</td>
<td>3</td>
<td>$=$</td>
<td>28.84</td>
<td>0.16</td>
</tr>
<tr>
<td>trigo1</td>
<td>10</td>
<td>$=$</td>
<td>18.93</td>
<td>1.38</td>
</tr>
<tr>
<td>brent-10</td>
<td>10</td>
<td>$=$</td>
<td>6.96</td>
<td>0.54</td>
</tr>
<tr>
<td>h74</td>
<td>5</td>
<td>$=$</td>
<td>305.98</td>
<td>13.70</td>
</tr>
<tr>
<td>fredtest</td>
<td>6</td>
<td>$\leq$</td>
<td>3 146.44</td>
<td>19.33</td>
</tr>
</tbody>
</table>

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 $\mathcal{P}^#$

\[ \tau_p(\mathcal{X}^#) = \max_{g_i, g_j \in \mathcal{X}^#} \| g_i - g_j \| \]

\[ \oplus_p(\mathcal{X}^#) = \left\{ \mathcal{X}^# \cup \left\{ \sum_i \beta_i v_i \leq h \right\}, \mathcal{X}^# \cup \left\{ \sum_i \beta_i v_i \geq h \right\} \right\} \]
Outline

1. Introduction
2. Bringing AI ideas to CP
3. Bringing CP ideas to AI
   - Representing disjunctive information
   - Iterations
4. Analyzing Sound Processes with Constraints
5. Conclusion
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
- program analysis often requires non-convex properties
  such as $X \neq 0$

$\implies$ disjunctive completion: use sets of intervals
Disjunctive analysis: example

Example

if (X ≥ 0 && X < 10) B = 1; else B = 0;
*  
  . .
  if (B == 1)  • A[X] = 0;

we must prove that 0 ≤ X < 10 at •.

Plain interval analysis: one box at each program point
at ∗ we must join (B ∈ [1, 1], X ∈ [0, 9]) and (B ∈ [0, 0], X ∈ [−∞, +∞])
to get (B ∈ [0, 1], X ∈ [−∞, +∞])
⇒ at •, 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 ∗
⇒ at •, B == 1 recovers X ∈ [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

Truchet-Miné
Control-free programs

What happens when there is no explicit control flow?

Example: digital filter

```
while true do
  r = 1.5 \times s0 - 0.7 \times 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
\(\implies\) 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

\[ \implies \text{the analysis with boxes will fail} \]

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

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

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

**Example**

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

**Interval analysis:**

searching for an interval loop invariant $i$ for $x$

\[ i = \text{lfp } F \]

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

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

\(\Rightarrow\) we must over-approximate least fixpoints

classic technique:

- increasing iterations: from \(\emptyset\), iterate $F$
  use extrapolation \(\nabla\) to finish in finite time
  \(\Rightarrow\) we obtain a rough over-approximation

- decreasing iterations to refine the approximation

(explained in the following slides)
Increasing iterations in AI

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

the least fixpoint is: \( \text{lfp } F = [0, 101] \)

the iterates are: \( \emptyset, [0, 0], [0, 2], [0, 4], \ldots, [0, 98], [0, 100], [0, 101] \)
Increasing iterations with extrapolation in AI

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

the least fixpoint is: \( \text{lfp } F = [0, 101] \)

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

unstable bounds are set to \( +\infty \)

\( \implies \) over-approximates \([0, 101]\), but coarse
Bringing CP ideas to AI

Iterations

Decreasing iterations in AI

\[ F(x) = [0, 0] \sqcup ((x \cap [\infty, 99]) \oplus [2, 2]) \]

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)
  \[\Rightarrow\] 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

1. Introduction
2. Bringing AI ideas to CP
3. Bringing CP ideas to AI
4. Analyzing Sound Processes with Constraints
5. Conclusion
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](http://faust.grame.fr)
Faust

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

\[
y[n] = \frac{x[n]}{2147483647.0} \times l[n]
\]

\[
x[n] = 12345 + x[n-1] \times 1103515245
\]

\[
l[n] \in [0..1] \text{ UI 'level' slider}
\]
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 (\textit{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.

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

<table>
<thead>
<tr>
<th>program</th>
<th># blocks (# fby)</th>
<th>time</th>
<th>distance</th>
<th># blocks evaluations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(# bly)</td>
<td>avg</td>
<td>avg</td>
<td>min</td>
</tr>
<tr>
<td>SIMPLE-ECHO</td>
<td>4 (1)</td>
<td>1ms</td>
<td>&lt; 0.001</td>
<td>4</td>
</tr>
<tr>
<td>SIMPLE-COUNTER</td>
<td>3 (2)</td>
<td>11ms</td>
<td>0</td>
<td>3604</td>
</tr>
<tr>
<td>SIMPLE-SINUS</td>
<td>4 (2)</td>
<td>10ms</td>
<td>0</td>
<td>3595</td>
</tr>
<tr>
<td>PAPER-EXAMPLE</td>
<td>4 (2)</td>
<td>16ms</td>
<td>&lt; 0.001</td>
<td>3169</td>
</tr>
<tr>
<td>FAUST-NOISE</td>
<td>6 (2)</td>
<td>1ms</td>
<td>0</td>
<td>115</td>
</tr>
<tr>
<td>FAUST-VOLUME</td>
<td>8 (2)</td>
<td>21ms</td>
<td>0</td>
<td>4153</td>
</tr>
<tr>
<td>FAUST-ECHO</td>
<td>16 (2)</td>
<td>14ms</td>
<td>0</td>
<td>3156</td>
</tr>
<tr>
<td>FAUST-OSC</td>
<td>28 (7)</td>
<td>31ms</td>
<td>&lt; 0.001</td>
<td>7791</td>
</tr>
<tr>
<td>FAUST-FREEVERB</td>
<td>237 (104)</td>
<td>0.51s</td>
<td>0</td>
<td>48348</td>
</tr>
<tr>
<td>FAUST-KARPLUS32</td>
<td>530 (133)</td>
<td>0.69s</td>
<td>0</td>
<td>102813</td>
</tr>
</tbody>
</table>
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!
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!


Bibliography

Languages, pages 238–252, Los Angeles, California. ACM Press, New York, NY.


