Constraint Solving
Using Numeric Abstract Domains

Marie Pelleau\textsuperscript{2}, Antoine Miné\textsuperscript{1},
Charlotte Truchet\textsuperscript{2}, Frédéric Benhamou\textsuperscript{2}

\textsuperscript{1}CNRS & École normale supérieure, Paris
\textsuperscript{2}Université de Nantes

Seminar “Modélisation et vérification”
LIAFA, Université Paris Diderot
8th April 2013
Introduction

Two fields:
- Constraint programming
- Abstract interpretation
with different goals, tools, communities.

**Observation:** there are similarities
- manipulate symbolic representations of sets of points
- compute fixpoints by iteration

**Goal:** exploit these similarities
to improve constraint programming
by using abstract interpretation techniques
Outline

- Background
  - Constraint programming (CP)
  - Abstract interpretation (AI)

- Links between CP and AI

- Constructing an abstract CP solver

- Experimental results

- Conclusion
Background
Constraint programming
Constraint programming

Introduced by Montanari in 1974:

1. express a problem using constraints
   - conjunctions of first-order numeric formulas
   - declarative programming

2. solve the constraints using generic methods

Many applications in:

- logistics (e.g. job shop scheduling problem)
- biology (e.g. ARN secondary structure)
- verification (e.g. program verification, model verification)
- test generation (e.g. generation of pairwise configuration tests)
- cryptography (e.g. design of cryptographic s-boxes)
- music (e.g. automatic harmonisation)
- etc.
Constraint satisfaction problem

Definition: CSP

- \( V \overset{\text{def}}{=} \{v_1, \ldots, v_n\} \): set of variables
- \( D \overset{\text{def}}{=} D_1 \times \cdots \times D_n \): set of initial domains
  - \( \forall i: D_i \subseteq \mathbb{R} \) and is bounded
- \( C \): set of constraints on \( V \)

Solution

- \( S \): the set of points in \( D \) satisfying all constraints in \( C \)

Notes:
- we can compute
  - all or only one solution(s)
  - the exact set if \( D_i \subseteq \mathbb{Z} \) (discrete case)
  - under- or over-approximations if \( D_i \subseteq \mathbb{R} \) (continuous case)
Constraint satisfaction problem (example)

Continuous problem:

- \( V \stackrel{\text{def}}{=} (v_1, v_2) \)
- \( D_1 \stackrel{\text{def}}{=} [-1, 14], D_2 \stackrel{\text{def}}{=} [-5, 10] \)
- \( C_1 : (v_1 - 9)^2 + v_2^2 \leq 25 \)
- \( C_2 : (v_1 + 1)^2 + (v_2 - 5)^2 \leq 100 \)
Continuous problem:

- \( V \overset{\text{def}}{=} (v_1, v_2) \)
- \( D_1 \overset{\text{def}}{=} [-1, 14], D_2 \overset{\text{def}}{=} [-5, 10] \)
- \( C_1 : (v_1 - 9)^2 + v_2^2 \leq 25 \)
- \( C_2 : (v_1 + 1)^2 + (v_2 - 5)^2 \leq 100 \)
Solving

**Principle:**

1. **Propagation:**
   - use constraints to shrink domains
   - remove domain values that are not part of a solution
   \[ \Rightarrow \text{consistency} \]

2. **Exploration:**
   - split domains
     - halve an interval (continuous case)
     - instantiate a variable (discrete case)
   - finished if the domains contain only solutions / no solution

3. **Backtracking:**
   - iterate until we have only solutions (discrete case)
   - or small enough domains (continuous case)
Parameter: float $r$

list of boxes $\text{sols}$

queue of boxes $\text{toExplore}$

box $e$

$sols \leftarrow \emptyset$

$\text{toExplore} \leftarrow \emptyset$

push $D$ in $\text{toExplore}$

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

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

$e \leftarrow \text{Hull-Consistency}(e)$

if $e \neq \emptyset$ then

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

$sols \leftarrow sols \cup e$

else

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

push $e1$ and $e2$ in $\text{toExplore}$


Parameter: float $r$

list of boxes sols
queue of boxes toExplore
box $e$

$sols \leftarrow \emptyset$
$toExplore \leftarrow \emptyset$
push $D$ in toExplore

while toExplore $\neq \emptyset$ do
  $e \leftarrow \text{pop}(toExplore)$
  $e \leftarrow \text{Hull-Consistency}(e)$
  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 $e_1$ and $e_2$
      push $e_1$ and $e_2$ in toExplore
  end if
end while
Parameter: float r

list of boxes sols
queue of boxes toExplore
box e

sols ← ∅
toExplore ← ∅
push $D$ in toExplore

while toExplore $\neq$ ∅ do
    e ← pop(toExplore)
    e ← Hull-Consistency(e)
    if e $\neq$ ∅ then
        if maxDim(e)$ \leq$ r or isSol(e) then
            sols ← sols $\cup$ e
        else
            split e in two boxes e1 and e2
            push e1 and e2 in toExplore
Parameter: float \( r \)

list of boxes \( \text{sols} \)
queue of boxes \( \text{toExplore} \)
box \( e \)

\( \text{sols} \leftarrow \emptyset \)
\( \text{toExplore} \leftarrow \emptyset \)
push \( D \) in \( \text{toExplore} \)

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

\( e \leftarrow \text{pop} (\text{toExplore}) \)
\( e \leftarrow \text{Hull-Consistency} (e) \)
if \( e \neq \emptyset \) then

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

\( \text{sols} \leftarrow \text{sols} \cup e \)
else

split \( e \) in two boxes \( e_1 \) and \( e_2 \)
push \( e_1 \) and \( e_2 \) in \( \text{toExplore} \)
Parameter: float r

list of boxes sols
queue of boxes toExplore
box e

sols ← ∅
toExplore ← ∅
push D 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
Parameter: float r

list of boxes sols
queue of boxes toExplore
box e

sols ← ∅
toExplore ← ∅
push D 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
Parameter: float r

list of boxes sols
queue of boxes toExplore
box e

sols ← ∅
toExplore ← ∅
push D 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
  else
    break
Solving

Many variants:
- domain representations: sets, intervals
- consistencies: (generalized) arc-consistency, bound-consistency
- propagation algorithms: AC1, AC3, GAC3, AC4, etc.
- exploration algorithms: backtracking, backjumping, etc.
- variable selection heuristics: first-fail, domain, degree, etc.
- value selection heuristics: max solutions, min conflicts, etc.

Limitations:
- either continuous or discrete solving method
- representations limited to Cartesian products
Solving

**Many variants:**
- domain representations: sets, intervals
- consistencies: (generalized) arc-consistency, bound-consistency
- propagation algorithms: AC1, AC3, GAC3, AC4, etc.
- exploration algorithms: backtracking, backjumping, etc.
- variable selection heuristics: first-fail, domain, degree, etc.
- value selection heuristics: max solutions, min conflicts, etc.

**Limitations:**
- either continuous or discrete solving method
  \[\Rightarrow\] design discrete-continuous hybrid methods
- representations limited to Cartesian products
  \[\Rightarrow\] exploit relations between variables
Discrete-continuous problems

State-of-the-art:
- use a discrete solver: discretize continuous constraints
- use a continuous solver: add “integerness” constraints
- integrate two solvers
Abstract interpretation
Abstract interpretation

General theory of the approximation of program semantics introduced by Cousot in 1977.

**Application**

- formalize and compare semantics (operational, denotational, etc.)
- formalize semantic-based, sound static analyses (data-flow, typing, etc.)
- tool to derive new analyses in a systematic way

**Principle**

- choose a set of properties of interest (abstract domain)
- design sound abstract operators
- express semantics as fixpoints, and approximate them
Abstract interpretation example

**Reachability semantics** of numeric programs.

```plaintext
1: int x, y
2: y ← 1
3: x ← random(1, 5)
4: while y<3 and x≤8
   do
   5: x ← x+y
   6: y ← 2*y
   7: x ← x-1
   8: y ← y+1
```
Reachability semantics of numeric programs.

1: int x, y
2: y ← 1
3: x ← random(1, 5)
4: while y < 3 and x ≤ 8
do
5: x ← x + y
6: y ← 2*y
7: x ← x - 1
8: y ← y + 1

Goal: prove that the program never enters the forbidden zone.
Undecidable in general.
Abstract interpretation example

Reachability semantics of numeric programs.

1: int x, y
2: y ← 1
3: x ← random(1, 5)
4: while y < 3 and x ≤ 8
   do
   5: x ← x+y
   6: y ← 2*y
   7: x ← x-1
   8: y ← y+1

Concrete semantics: elements in $X \in D \overset{\text{def}}{=} \mathcal{P}(V \to \mathbb{Z})$. Uncomputable in general.
Reachability semantics of numeric programs.

Abstract semantics: $X^\# \in D^\#$ boxes.
Computable (over-)approximation.
Not precise enough $\Rightarrow$ false alarm.
Reachability semantics of numeric programs.

Abstract semantics: $X^* \in D^*$ octagons.
Computable (over-)approximation.
Proves the absence of error.
Reachability semantics of numeric programs.

1: int x, y
2: y ← 1
3: x ← random(1, 5)
4: while y<3 and x≤8
do
5: x ← x+y
6: y ← 2*y
7: x ← x-1
8: y ← y+1

Abstract semantics: $X^\# \in D^\#$ polyhedra.
Computable (over-)approximation.
Proves the absence of error.
Abstract domains

**Semantic choice:**
- partially ordered semantic set $D^\#$
  - with a monotonic concretization $\gamma : D^\# \to D$
- sound version $F^\#$ of all semantic operators $F$
  - operators: assignments and tests $[\cdot]$, joins $\cup$, etc.
  - soundness: $\forall X^\# \in D^\#: F(\gamma(X^\#)) \sqsubseteq \gamma(F^\#(X^\#))$
- optional abstraction function $\alpha : D \to D^\#$
  - Galois connection: $D \xleftarrow{\gamma} D^\#: \alpha(X) \sqsubseteq Y^\# \iff X \sqsubseteq^\# \gamma(Y^\#)$
  - best abstraction: $F^\# = \alpha \circ F \circ \gamma$

**Algorithmic choice:**
- data-structures for elements in $D^\#$
- algorithms to compute (sound or optimal) $F^\#$

Choosing a domain:
- trade-off between cost and precision / expressiveness.
Fixpoints appear in the semantics of loops
e.g.: $\lceil \neg \text{cond} \rceil (\text{lfp } \lambda X. I \cup (\lceil \text{body} \rceil \circ \lceil \text{cond} \rceil)(X))$

**Approximating $\text{lfp } F$ in $D^\#$:**
- consider a sound abstraction $F^\#$ of $F$
- compute increasing iterates: $X_{n+1}^\# \overset{\text{def}}{=} F^\#(X_n^\#)$ from $X_0^\# = \bot^\#$
- accelerate iterates with a widening $\nabla$
  - compute $X_{n+1}^\# \overset{\text{def}}{=} X_n^\# \nabla F^\#(X_n^\#)$
  - $\nabla$ over-approximates $\cup$ and ensures convergence
    e.g.: put unstable interval bounds to $\infty$
- refine the over-approximation $X_\omega^\#$ with decreasing iterations
  - compute $Y_{n+1}^\# \overset{\text{def}}{=} Y_n^\# \triangle F^\#(Y_n^\#)$ from $Y_0^\# = X_\omega^\#$
  - use a narrowing $\triangle$ to ensure the convergence
    - $\triangle$ over-approximates $\cap \implies$ stay above the (least) fixpoint
Links between abstract interpretation and constraint programming
Solving as fixpoint computation

**Exact solution set:**
Each constraint $C_i$ gives rise to a filter function $\rho_i$:

$$\rho_i(X) \overset{\text{def}}{=} \{ \sigma \in X \mid \sigma \models C_i \} \in D \to D$$

Note $\rho \overset{\text{def}}{=} \rho_1 \circ \cdots \circ \rho_k$, the solution set $S$ is a greatest fixpoint:

$$S \overset{\text{def}}{=} \rho(D) = \text{gfp}_D \rho$$

Interpreting the solving algorithm

- iterative process starting from $D$
- decreasing iterations over-approximating $S$
- termination criterion (small size, singleton)

$\implies$ similar to fixpoint refinement with $\Delta$
Consistencies as abstract domains

Interpreting consistency:

- CP domain $\simeq$ element in an abstract domain $X^\# \in D^\#$
- $X^\#$ is consistent $\iff X^\# = (\alpha \circ \rho \circ \gamma)(X^\#)$

Examples

- **Arc-consistency**: Cartesian abstraction
  - $D^\# \overset{\text{def}}{=} \mathcal{P}(\mathbb{Z})^n$
  - $\alpha(X) \overset{\text{def}}{=} \lambda i.\{x \mid \exists x_1, \ldots, x_n \in X: x = x_i\}$

- **Hull-consistency**: interval abstraction with float bounds
  - $D^\# \overset{\text{def}}{=} (\mathbb{F} \times \mathbb{F})^n$
  - $\alpha(X) \overset{\text{def}}{=} \lambda i.\left[\begin{array}{ll}
   \max \{x \in \mathbb{F} \mid \forall x_1, \ldots, x_n \in X: x_i \geq x\}, \\
   \min \{x \in \mathbb{F} \mid \forall x_1, \ldots, x_n \in X: x_i \leq x\}\end{array}\right]$
Atomic propagation as test

Given a constraint $C_i$:

- $\rho_i$ models a program test in the concrete (filter)
- the consistency is the best abstraction $\alpha \circ \rho_i \circ \gamma$
- a propagator is a sound abstraction $\rho_i^\#$ of $\rho_i$

Example algorithm

Test $expr \leq 0$ with intervals:

- annotate leaves (variables) with current intervals
- evaluate the expression tree bottom-up in interval arithmetic
- intersect the root with $[-\infty, 0]$
- refine top down-using backward interval arithmetic

Known as HC4-revise in CP, independently rediscovered to implement tests in AI.
Interval test example

\[ C : X + Y - Z \leq 0 \text{ with } D_X \mapsto [0, 10], D_Y \mapsto [2, 10], D_Z \mapsto [3, 5] \]
Iterated propagation as local iterations

Propagation of multiple constraints $C_1, \ldots, C_k$.

Idea: approximate $\rho \overset{\text{def}}{=} \rho_1 \circ \cdots \circ \rho_k$ as $\rho^\# \overset{\text{def}}{=} \rho_1^\# \circ \cdots \circ \rho_k^\#$.

Issue: optimal operators do not compose
(even if each $\rho_i^\#$ is optimal, $\rho^\#$ may not be optimal)

Solution: iterate $\rho^\#$

- $\rho^k = \rho$, so $(\rho^\#)^k$ is a sound abstraction of $\rho$
- $(\rho^\#)^k$ is generally more precise than $\rho^\#$
- iterate until a fixpoint, or stop before (narrowing)
- schedule $\rho_i^\#$ based on refined variables (to improve efficiency)

Known as local iterations in AI [Granger], used in CP as well.
Links between AI and CP

Split as disjunctive completion

Solvers abstract solution sets using sets of domains
\[ \implies \text{corresponds to disjunctive completion in AI} \]

\[
\begin{align*}
\mathcal{PD}^\# & \overset{\text{def}}{=} \mathcal{P}_{\text{finite}}(\mathcal{D}^\#) \\
\gamma_{\mathcal{P}}(\mathcal{X}^\#) & \overset{\text{def}}{=} \bigcup \{ \gamma(X^\#) \mid X^\# \in \mathcal{X}^\# \}
\end{align*}
\]

**Split operator** \( \oplus \):

- \( \oplus : \mathcal{PD}^\# \rightarrow \mathcal{PD}^\# \)
- \( \oplus(\mathcal{X}^\#) \) chooses an element in \( \mathcal{X}^\# \in \mathcal{X}^\# \)
  and replaces it with finitely many elements in \( \mathcal{D}^\# \)
- \( \gamma_{\mathcal{P}}(\oplus \mathcal{X}^\#) = \gamma_{\mathcal{P}}(\mathcal{X}^\#) \implies \oplus \) abstracts the identity
- \( \oplus \) decreases for the Smyth order
  \[
  \mathcal{X}^\# \sqsubseteq_{\mathcal{P}} \mathcal{Y}^\# \overset{\text{def}}{\iff} \forall X^\# \in \mathcal{X}^\#: \exists Y^\# \in \mathcal{Y}^\#: X^\# \sqsubseteq Y^\#
  \]
Synthesis

Similarities:
- similar goal: over-approximate a set (environments or solutions)
- similar methods: iterative approximation of fixpoints
- common symbolic representations and algorithms (disjunctions of boxes, propagators)

Differences:
- CP limited to non-relational, homogeneous representations
  AI allows relational and mixed (integer-real) representations
- CP only cares for greatest fixpoints
  AI uses least and greatest fixpoints (loops, local iterations)
- CP uses only decreasing iterations
  AI uses both increasing and decreasing iterations
- CP uses disjunctive completion for dynamic refinement
  more advanced than decreasing iterations than AI (△)
Abstract solving method
Parameter: an abstract domain $D^\#$ equipped with:

- Intervals
- Octagons
- Polyhedra
**Parameter:** an abstract domain $\mathcal{D}^\#$ equipped with:

- test transfer functions $\rho^\#$ for any conjunction of constraints
  (approximate consistency)
Parameter: an abstract domain $D^\#$ equipped with:

- test transfer functions $\rho^\#$ for any conjunction of constraints (approximate consistency)

and additionally:

- a splitting operator $\oplus : P\!D^\# \rightarrow P\!D^\#$ (exploration)
Parameter: an abstract domain $D^\#$ equipped with:

- test transfer functions $\rho^\#$ for any conjunction of constraints (approximate consistency)

and additionally:

- a splitting operator $\bigoplus : \mathcal{PD}^\# \rightarrow \mathcal{PD}^\#$ (exploration)
- a size function $\tau : D^\# \rightarrow \mathbb{R}^+$ (termination criterion)
Abstract solving method

From continuous solving to abstract solving

Continuous solver

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 \( \text{maxDim}(e) \leq r \) 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

From continuous solving to abstract solving

Continuous Abstract solver

Parameter: float r

list of boxes disjunction sols ← ∅
queue of boxes disjunction toExplore ← ∅
box abstract element 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
Soundness and termination

Soundness:

- $D^\# \text{ over-approximates } D \implies D^\# \text{ over-approximates } S$
- $\rho^\# \text{ over-approximates } \lambda X. X \cap S$
- $\oplus \text{ over-approximates the identity}$

$\implies \text{toExplore } \cup \text{sols over-approximates } S$

upon termination, $\text{sols over-approximates } S$

Termination:

Hypothesis on $\oplus$ and $\tau$:

any sequence of consistent splits $\rho^\# \circ \oplus$

ultimately gives elements smaller than $r$ for $\tau$

Then, the algorithm terminates (König’s lemma)
Non-relational split and size functions

**Arc-consistency** for discreet solvers

- Cartesian abstraction: \( D^\# \overset{\text{def}}{=} \mathcal{P}(\mathbb{Z})^n \)
- split on variable \( v_i \):
  \[
  \bigoplus_i (S_1 \times \cdots \times S_i \cdots \times S_n) \overset{\text{def}}{=} \{ S_1 \times \cdots \{s\} \cdots \times S_n \mid s \in S_i \}
  \]
- \( \tau(S_1 \times \cdots \times S_n) \overset{\text{def}}{=} \max_i (|S_i| - 1) \)

**Hull-consistency** for continuous solvers

- float interval abstraction: \( D^\# \overset{\text{def}}{=} (\mathbb{F} \times \mathbb{F})^n \)
- split on variable \( v_i \):
  \[
  \bigoplus_i ([\ell_1, h_1] \times \cdots \times [\ell_i, h_i] \times \cdots \times [\ell_n, h_n]) \overset{\text{def}}{=} \\
  \{ [\ell_1, h_1] \times \cdots \times [\ell_i, m] \times \cdots \times [\ell_n, h_n], \\
  [\ell_1, h_1] \times \cdots \times [m, h_i] \times \cdots \times [\ell_n, h_n] \}
  \]
  where \( m \overset{\text{def}}{=} (\ell_i + h_i)/2 \) (rounded indifferently)
- \( \tau([\ell_1, h_1] \times \cdots \times [\ell_n, h_n]) \overset{\text{def}}{=} \max_i (h_i - \ell_i) \)
Abstract solving method

Octagon domain

\[ D^\# \overset{\text{def}}{=} \{ \alpha v_i + \beta v_j \mid i, j \in [1, n], \alpha, \beta \in \{-1, 1\} \} \rightarrow \mathbb{F} \]

\( D^\# \): associates a (float) bound to each unit binary expression on \( V \)
Octagon domain

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

\[ \tau(X^\#) \overset{\text{def}}{=} \min( \max_{i,j,\beta} (X^\#(v_i + \beta v_j) + X^\#(-v_i - \beta v_j)), \max_i (X^\#(v_i + v_i) + X^\#(-v_i - v_i)) / 2 ) \]

\[ \tau: \text{size of the smallest box containing the octagon} \]
Abstract solving method

Octagon domain

\[ D^\# \overset{\text{def}}{=} \{ \alpha v_i + \beta v_j \mid i, j \in [1, n], \alpha, \beta \in \{-1, 1\} \} \rightarrow \mathbb{F} \]

\[ \tau(X^\#) \overset{\text{def}}{=} \min( \max_{i,j,\beta} (X^\#(v_i + \beta v_j) + X^\#(-v_i - \beta v_j)), \max_i (X^\#(v_i + v_i) + X^\#(-v_i - v_i)) / 2 ) \]

\[ \tau: \text{size of the smallest box containing the octagon} \]
Octagon domain

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

\[ \tau(X^\#) \overset{\text{def}}{=} \min( \max_{i,j,\beta} (X^\#(v_i + \beta v_j) + X^\#(-v_i - \beta v_j)) , \max_i (X^\#(v_i + v_i) + X^\#(-v_i - v_i)) / 2 ) \]

\[ \tau: \text{size of the smallest box containing the octagon} \]
Abstract solving method

**Octagon domain**

\[ D^\# \overset{\text{def}}{=} \{ \alpha v_i + \beta v_j \mid i, j \in [1, n], \alpha, \beta \in \{-1, 1\} \} \rightarrow \mathbb{F} \]

\[ \tau(X^\#) \overset{\text{def}}{=} \min( \max_{i,j,\beta} (X^\#(v_i + \beta v_j) + X^\#(-v_i - \beta v_j)) , \max_{i} (X^\#(v_i + v_i) + X^\#(-v_i - v_i)) / 2 ) \]

\[ \tau: \text{size of the smallest box containing the octagon} \]
Abstract solving method

Octagon domain

\[ D^\# \overset{\text{def}}{=} \{ \alpha v_i + \beta v_j \mid i, j \in [1, n], \alpha, \beta \in \{-1, 1\} \} \rightarrow \mathbb{F} \]

\[ \tau(X^\#) \overset{\text{def}}{=} \min( \max_{i,j,\beta} (X^\#(v_i + \beta v_j) + X^\#(-v_i - \beta v_j)) , \max_i (X^\#(v_i + v_i) + X^\#(-v_i - v_i))/2 ) \]

\[ \oplus(X^\#) \overset{\text{def}}{=} \{ X^\#[(\alpha v_i + \beta v_j) \mapsto m] , X^\#[(-\alpha v_i - \beta v_j) \mapsto -m] \} \]

where \( m \overset{\text{def}}{=} (X^\#(\alpha v_i + \beta v_j) - X^\#(-\alpha v_i - \beta v_j))/2 \)

\( \oplus \): cuts in half perpendicular to the longest side
Abstract solving method

Octagon domain

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

\[ \tau(X^\#) \overset{\text{def}}{=} \min \left( \max_{i,j,\beta} (X^\#(v_i + \beta v_j) + X^\#(-v_i - \beta v_j)), \right. \]

\[ \left. \max_i (X^\#(v_i + v_i) + X^\#(-v_i - v_i)) / 2 \right) \]

\[ \bigoplus(X^\#) \overset{\text{def}}{=} \left\{ X^\#[\alpha v_i + \beta v_j \mapsto m], X^\#[(-\alpha v_i - \beta v_j) \mapsto -m] \right\} \]

where \( m \overset{\text{def}}{=} (X^\#(\alpha v_i + \beta v_j) - X^\#(-\alpha v_i - \beta v_j))/2 \)

\[ \bigoplus: \text{cuts in half perpendicular to the longest side} \]
Octagon domain

\[ D^\# \overset{\text{def}}{=} \{ \alpha v_i + \beta v_j \mid i, j \in [1, n], \alpha, \beta \in \{-1, 1\} \} \rightarrow \mathbb{F} \]

\[ \tau(X^\#) \overset{\text{def}}{=} \min( \max_{i,j,\beta} (X^\#(v_i + \beta v_j) + X^\#(-v_i - \beta v_j)) , \\
\max_i (X^\#(v_i + v_i) + X^\#(-v_i - v_i)) / 2 ) \]

\[ \oplus(X^\#) \overset{\text{def}}{=} \{ X^\# [(\alpha v_i + \beta v_j) \mapsto m], X^\# [(-\alpha v_i - \beta v_j) \mapsto -m] \} \]

where \( m \overset{\text{def}}{=} (X^\#(\alpha v_i + \beta v_j) - X^\#(-\alpha v_i - \beta v_j))/2 \)

\[ \oplus: \text{ cuts in half perpendicular to the longest side} \]
Octagon solving example

Intervals

Octagons
Polyhedra domain

\[ D^\# \overset{\text{def}}{=} \{ \text{convex closed polyhedra in } \mathbb{R}^n \} \]

- constraint representation: \( \{ \sum_i \alpha_i v_i \leq \beta \} \)
- generator representation: vertices \( g_1, \ldots, g_k \in \mathbb{R}^n \)

(domains are bounded, no need for rays)
**Abstract solving method**

**Polyhedra domain**

\[ \mathcal{D}^\# \defeq \{ \text{convex closed polyhedra in } \mathbb{R}^n \} \]

- constraint representation: \( \{ \sum_i \alpha_i v_i \leq \beta \} \)
- generator representation: vertices \( g_1, \ldots, g_k \in \mathbb{R}^n \)  
  (domains are bounded, no need for rays)

\[ \tau(X^\#) \defeq \max_{g_i, g_j \in X^\#} \| g_i - g_j \| \]

\( \tau \): maximal distance between two vertices
Abstract solving method

Polyhedra domain

\[ \mathcal{D}^\# \overset{\text{def}}{=} \{ \text{convex closed polyhedra in } \mathbb{R}^n \} \]

- constraint representation: \( \{ \sum_i \alpha_i v_i \leq \beta \} \)
- generator representation: vertices \( g_1, \ldots g_k \in \mathbb{R}^n \)
  (domains are bounded, no need for rays)

\[ \tau(\mathcal{X}^\#) \overset{\text{def}}{=} \max_{g_i, g_j \in \mathcal{X}^\#} ||g_i - g_j|| \]

\[ \oplus(\mathcal{X}^\#) \overset{\text{def}}{=} \{ \mathcal{X}^\# \cup \{ \sum_i \alpha_i v_i \leq m \} , \mathcal{X}^\# \cup \{ \sum_i \alpha_i v_i \geq m \} \} \]

where \( m \overset{\text{def}}{=} (\min_{v \in \gamma(\mathcal{X}^\#)} (\sum_i \alpha_i v_i) + \max_{v \in \gamma(\mathcal{X}^\#)} (\sum_i \alpha_i v_i)) / 2 \)

\( \oplus \): cuts perpendicular to the longest chord
A note on tests in relational domains

**Issue:** relational domains can represent restricted constraints. How can we model **non-linear** constraints?

**Solution:** linearization [Apron]

- abstract expressions to the form: \([a_0, b_0] + \sum_i [a_i, b_i] v_i\)
  - affine form (easy to manipulate)
  - intervals to abstract complex parts as non-deterministic choices
  - transformation based on a bounding box \(B\) for variables
  
  e.g.: \((v_1 - 9)^2 + v_2^2\) in \([-1, 14] \times [-5, 10]\)
  gives \([-10, 9]v_1 + [-5, 10]v_2 + [-126, 9]\)

- **Polyhedra:** abstract further into \([a'_0, b'_0] + \sum_i c_i v_i\)

- **Octagons:** generate all constraints of the form \(\alpha v_j + \beta v_k \leq m\)
  where \(m \overset{\text{def}}{=} \max_B([a_0, b_0] + \sum_i [a_i, b_i] v_i + \alpha v_j + \beta v_k)\)
  evaluated in interval arithmetic (after simplification)
Abstract domains can also abstract $\mathcal{P}(\mathbb{R}^m \times \mathbb{Z}^n)$

- variables are tagged as real or integer
- the semantics starts as in $\mathcal{P}(\mathbb{R}^{m+n})$
  but each result is intersected with $\mathbb{R}^m \times \mathbb{Z}^n$

Various ways to handle the “integerness” information:

- **ignore** integerness (abstract integers as reals)
  sound but not very precise

- **Intervals**: **truncate** the bounds of integer variables
  sound and optimal

- **Polyhedra**: **truncate** the constant side of constraints

  $\sum \alpha_i v_i \leq \beta$ (subject to conditions on $v_i, \alpha, \beta$)

  sound but not optimal, used in [Apron]
Experimental results
Implementation

Prototype solver: AbSolute

- written in OCaml
- uses the Apron numeric abstract domain library:
  - intervals, octagons, and polyhedra
  - mixed integer-real variables
  - non-linear tests

- propagation loop:
  - propagate all the constraints at each iteration
  - iterate for a fixed number of times

- split: Cartesian split only
- variable choice: round-robin

On-going work: currently AbSolute is based on

- advanced AI features
- but rather naive CP techniques
Experiments: continuous problems

- problems from the COCONUT standard benchmark
- compute all the solutions of the problems
- timeout: 1 hour
- comparison with Ibex
  - standard interval-based Ibex
  - Ibex extended to octagons (from previous works)
  \[\implies\] compare domain independently from solver
## Experimental results

### Results: continuous problems

<table>
<thead>
<tr>
<th>name</th>
<th># vars</th>
<th>ctr type</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>

### Synthesis:

- competitive, despite using naive propagation loop, split, and choice
- octagons may suffer more than intervals from these naive choices
Experimental results

Experiments: mixed problems

**Issue:** difficult to

- find a mixed problem solver to compare to
- find mixed problem benchmarks

**Experiments:**

- optimization problems from the MinLPLib benchmark reformulated as constraint satisfaction problems
  - objective function \( \min f(V) \) with known optimal \( o \)
  - replaced with the constraints \( o - \epsilon \leq f(V) \leq o + \epsilon \)

- compare various domains with AbSolute
- find all solutions
- timeout after 1 hour
Experimental results

Results: mixed problems

<table>
<thead>
<tr>
<th>name</th>
<th># vars</th>
<th>ctr type</th>
<th>Itv</th>
<th>Oct</th>
<th>Poly</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>int</td>
<td>real</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gear4</td>
<td>4</td>
<td>2</td>
<td>=</td>
<td>0.017</td>
<td>0.048</td>
</tr>
<tr>
<td>st_miqp5</td>
<td>2</td>
<td>5</td>
<td>≤</td>
<td>2.636</td>
<td>3.636</td>
</tr>
<tr>
<td>ex1263</td>
<td>72</td>
<td>20</td>
<td>= ≤</td>
<td>473.933</td>
<td>-</td>
</tr>
<tr>
<td>antennes_4_3</td>
<td>6</td>
<td>2</td>
<td>≤</td>
<td>520.766</td>
<td>1562.335</td>
</tr>
</tbody>
</table>

Proof that native handling of mixed constraint is possible.
Conclusion
In this work we:

- uncovered some links between AI and CP
- constructed an abstract solving method parametrized by abstract domains
  - able to simulate existing solvers (consistencies)
  - give rise to new solvers
- showed how to go beyond non-relational representations
- showed how to natively handle mixed constraints using existing techniques from AI
- presented a proof-concept implementation
Abstract solving:
- improve our implementation using existing CP techniques
  - variable selection for split
  - propagation loop
- novel directions to improve our solver
  - split operators and direction selection
  - improved test (consistency) algorithms
  - new domains adapted to specific constraints
    $\Rightarrow$ reduce product in a library of domains

Links between AI and CP:
- exploit CP techniques to improve AI
  - decreasing iterations: alternatives to narrowing
- explore the possible role of widening in CP
- compare CP and AI performance on common problems
  - program analysis and verification