Numeric abstract domains in static analysis and constraint programming

Antoine Miné

CNRS & École normale supérieure
Paris, France

Journée calcul ensembliste et interprétation abstraite
GT MEA
19 March 2015

Semantic-based static analysis

- analyzes directly the source code (not a model)
- automatic and always terminating
- approximate (for automation and scalability)
- **sound**
  - full coverage of control and data
  - the properties inferred by the analyzer hold on the program
- **incomplete**
  - some properties can be missed by the analyzer
  - false alarms
- traditionally used in low precision settings (e.g., optimization)
- can be made precise enough for validation in some contexts

  (Astrée analyzer: few or no false alarms)
- adaptable to different classes of programs and properties

Outline

- Abstract interpretation
- The Astrée static analyzer

  joint work with P. Cousot's team at ENS
- Abstract constraint programming

  joint work with Marie Pelleau and Charlotte Truchet

Abstract interpretation

Interval analysis example

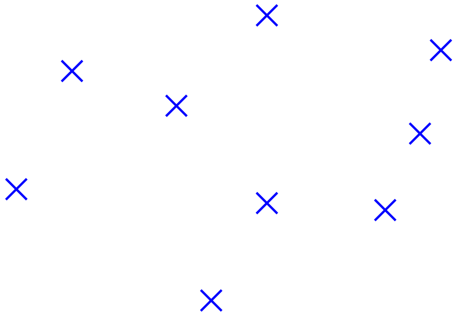
```
int tab[1000];
assume X in [0,1000];

I = 0;

while (I < X) {

    tab[I] = 0;
    I = I + 2;

}
```

Interval analysis example

```
int tab[1000];
assume X in [0,1000];
{X ∈ [0,1000]}
I = 0;
{X ∈ [0,1000], I = 0}
while (I < X) {
    {X ∈ [0,1000], I ∈ [0,999]} •
    tab[I] = 0; // no overflow!
    I = I + 2;
    {X ∈ [0,1000], I ∈ [2,1001]}
}
{X ∈ [0,1000], I ∈ [0,1001]}
```

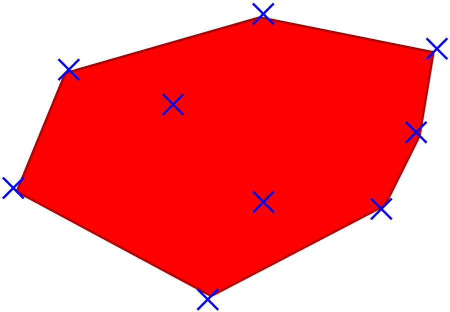
- automatic inference of invariants (including loop invariants)
- sufficient to prove that `tab[I]` has no overflow
- approximate (in fact, at •, $I \in [0,998] \cap 2\mathbb{Z}$)

Numeric abstract domain examples



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

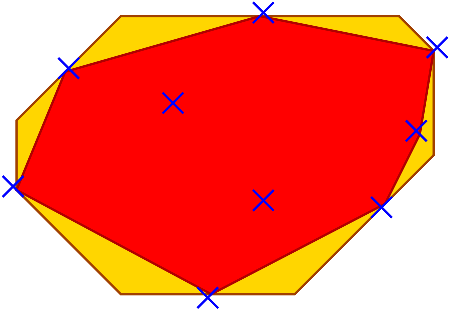
Numeric abstract domain examples



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

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

Numeric abstract domain examples

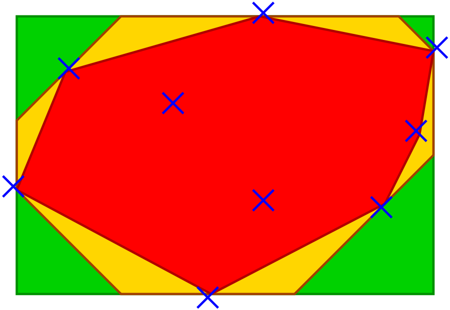


concrete sets $\mathcal{D}$: $\{(0,3),(5.5,0),(12,7),\ldots\}$

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

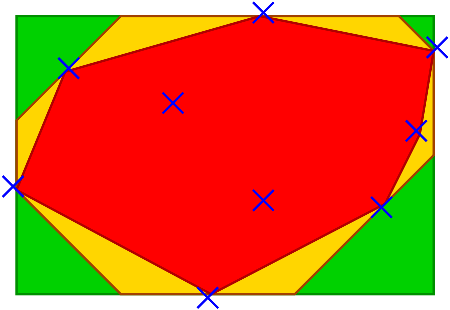
abstract octagons $\mathcal{D}^\sharp_o$: $X + Y \geq 3 \wedge Y \geq 0 \wedge \cdots$

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 $\mathcal{D}^\sharp_i$:	$X \in [0,12] \wedge Y \in [0,8]$

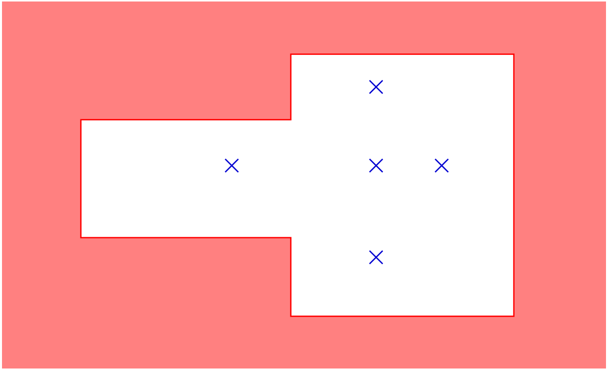
Numeric abstract domain examples



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

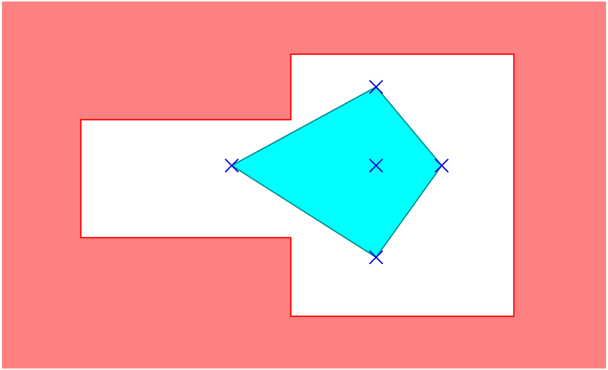
Trade-off between cost and expressiveness / precision

Correctness proof and false alarms



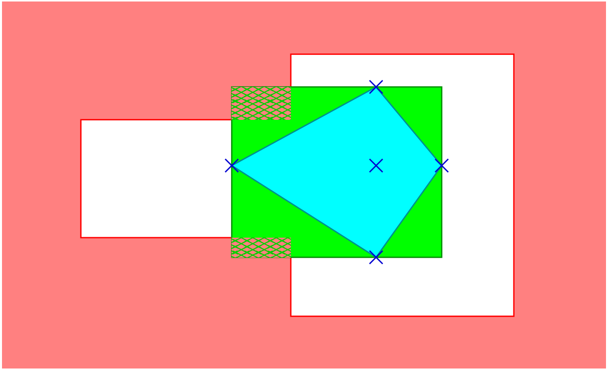
The program is correct (blue $\cap$ red $= \emptyset$).

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

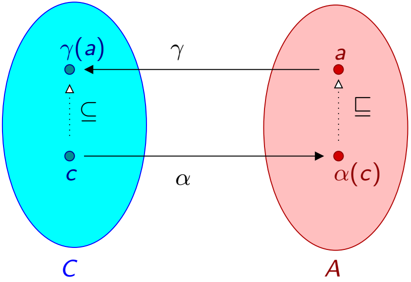


The program is correct (blue $\cap$ red $= \emptyset$).
The polyhedra domain can prove the correctness (cyan $\cap$ red $= \emptyset$).
The interval domain cannot (green $\cap$ red $\neq \emptyset$, false alarm).

Formalization: Galois Connection

Galois Connection $(C, \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \sqsubseteq)$

- $\alpha : C \to A$ (abstraction)
- $\gamma : A \to C$ (concretization)
- $\forall c \in C,\ a \in A :\ \alpha(c) \sqsubseteq a \iff c \subseteq \gamma(a)$ (duality)



$\alpha(c)$ is the best abstraction in A of $c \in C$

$\alpha(c)$ is the smallest a for $\sqsubseteq$ that over-approximates c, i.e., such that $c \subseteq \gamma(a)$

Abstract operators

Abstract interpretation: (literally)

Propagate abstract properties along program execution
(evaluation by induction on the syntax)

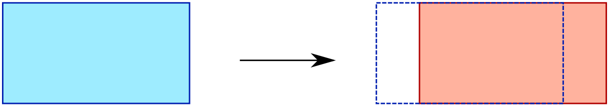
We need a sound abstract version $F^\sharp$ of each concrete operator F!

- soundness: $F \circ \gamma \subseteq \gamma \circ F^\sharp$
- optimality: $F^\sharp = \alpha \circ F \circ \gamma$

Abstract operators: Assignments

Interval assignment: `X = X + 1`

Translation

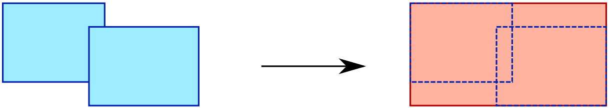


More generally: interval arithmetic

Abstract operators: Control-flow joins

Interval join: **if** $\cdots$ **then** $\cdots$ $\color{blue}\bullet$ **else** $\cdots$ $\color{blue}\bullet$ **fi** $\color{red}\bullet$

Interval hull: $\color{red}\bullet$ $=$ $\color{blue}\bullet$ $\sqcup$ $\color{blue}\bullet$

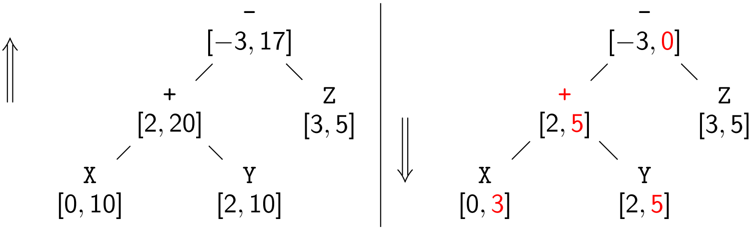


- in general: loss of precision
  (spurious points added)
- in this case, the operator is optimal
  (may not always be the case, for semantic or algorithmic reasons)
- combining optimal operators does not give an optimal operator!
  (imprecisions accumulate along abstract program execution)
  (e.g., **if** $x = 0$ **then** $x \leftarrow 1$ **fi**; **if** $x < 0$ **then** $x \leftarrow -x$ **fi**; **assert** $x > 0$)

$\Longrightarrow$ we may not find the tightest variable bounds

Abstract operators: Tests

C : $\texttt{X} + \texttt{Y} - \texttt{Z} \leq 0$ with $D_{\texttt{X}} \mapsto [0, 10], D_{\texttt{Y}} \mapsto [2, 10], D_{\texttt{Z}} \mapsto [3, 5]$



Algorithm:

- 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
- Similar to HC4-revise!

Abstract operators: Loop iterations

Loop invariant $N \leftarrow [0, 1000];$ **while** $i < N$ **do** $i \leftarrow i + 1$ **od**

Iterate the loop body and accumulate: $X^\sharp_{n+1} = X^\sharp_n \sqcup body(X^\sharp_n)$

$i = 0, N \in [0, 1000]$
$i \in [0, 1], N \in [0, 1000]$
$i \in [0, 2], N \in [0, 1000]$
$\vdots$
$i \in [0, 1000], N \in [0, 1000]$

but converges slowly!

(may even need ω iterations)

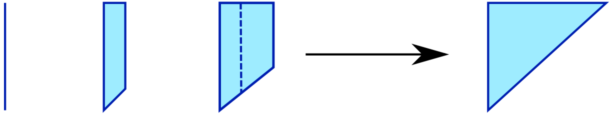
Abstract operators: Octagon widening

Loop invariant $N \leftarrow [0, 1000];$ **while** $i < N$ **do** $i \leftarrow i + 1$ **od**

$\Longrightarrow$ use a widening operator: $X^\sharp_{n+1} = X^\sharp_n \mathbin{\triangledown} body(X^\sharp_n)$

(extrapolate iterates to converge more quickly)

In the octagon domain: remove unstable constraints



$i = 0, N \in [0, 1000], i \leq N \rightarrow i \in [0, 1], i \leq N, N \in [0, 1000] \rightarrow 0 \leq i \leq N, N \in [0, 1000]$
$\Longrightarrow$ after the loop $i \in [0, 1000]$

- widening causes additional loss of precision
- many techniques exist to improve the result (narrowing, etc.)
- relational (affine) loop invariants are sometimes necessary to infer non-relational (interval) information after the loop
  (in the interval domain with widening, we would get $i \in [0, +\infty]$)

Intuitions behind the widening

Inductive reasoning (philosophical logic)

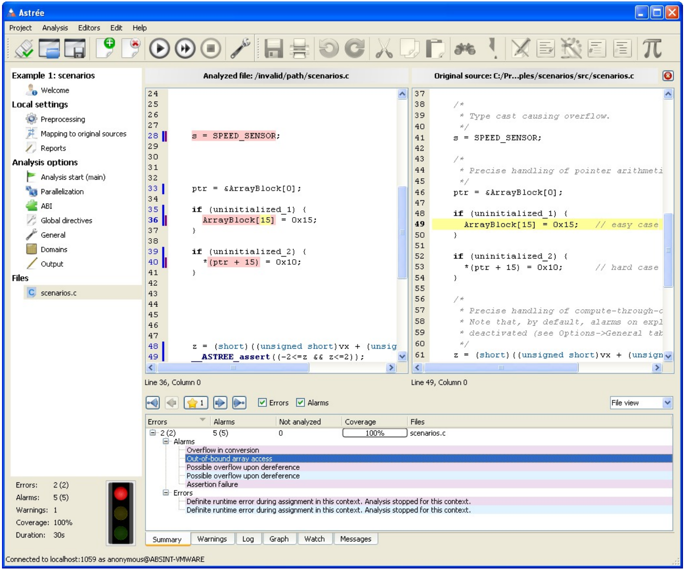
- induction = generalization from a small set of observations
  e.g., if the upper bound is increasing, it is probably unbounded
  major cognitive process
- $\neq$ induction in mathematics, which is deductive by nature
  (apply an induction axiom)
- in philosophy, induction is unreliable (finite observation)
  but in abstract interpretation, widening is always sound!

Inductive invariants

- lfp F defines the most precise invariant (concrete semantics)
- X such that lfp $F \subseteq X$ is a (possibly less precise) invariant
- X such that $F(X) \subseteq X$ is an inductive invariant
  (X is an invariant, and it can be proved to be invariant without computing lfp F)
- $X^\sharp$ such that $F^\sharp(X^\sharp) \sqsubseteq X^\sharp$ is an abstract inductive invariant
  ($\gamma(X^\sharp)$ can be proved to be invariant in the abstract, without computing lfp F)

The Astrée static analyzer

The Astrée static analyzer



The Astrée static analyzer

Analyseur statique de programmes temps-réels embarqués

(static analyzer for real-time embedded software)

- developed at ENS

  B. Blanchet, P. Cousot, R. Cousot, J. Feret,
  L. Mauborgne, D. Monniaux, A. Miné, X. Rival

- industrialized and made commercially available by AbsInt

Astrée
`www.astree.ens.fr`

AbsInt
`www.absint.com`

The Astrée static analyzer

Specialized:

- for the analysis of run-time errors

  (arithmetic overflows, array overflows, divisions by 0, etc.)

- on embedded critical C software

  (no dynamic memory allocation, no recursivity)

- in particular on control / command software

  (reactive programs, intensive floating-point computations)

- intended for validation

  (analysis does not miss any error and tries to minimise false alarms)

The Astrée static analyzer

Specialized:

- for the analysis of run-time errors

  (arithmetic overflows, array overflows, divisions by 0, etc.)
- on embedded critical C software

  (no dynamic memory allocation, no recursivity)
- in particular on control / command software

  (reactive programs, intensive floating-point computations)
- intended for validation

  (analysis does not miss any error and tries to minimise false alarms)

Approximately 40 abstract domains are used at the same time:

- numeric domains (intervals, octagons, ellipsoids, etc.)
- boolean domains
- domains expressing properties on the history of computations

Astrée applications

Airbus A340-300 (2003)

Airbus A380 (2004)

- size: from 70 000 to 860 000 lines of C
- analysis time: from 45mn to $\simeq$40h
- 0 alarm: proof of absence of run-time error

Abstract constraint programming

Constraint satisfaction problem

Definition: Constraint Satisfaction Problem (CSP)

- $\mathcal{V} \stackrel{\text{def}}{=} \{ v_1, \ldots, v_n \}$: set of variables
- $\mathcal{D} \stackrel{\text{def}}{=} D_1 \times \cdots \times D_n$: a set of initial domains
  $\forall i : D_i \subseteq \mathbb{R}$ and D_i is bounded
- $\mathcal{C} \stackrel{\text{def}}{=} \{ C_1, \ldots, C_m \}$ set of constraints on $\mathcal{V}$

CSP solution:

- $\mathcal{S} \stackrel{\text{def}}{=} \{ \vec{x} \in \mathcal{D} \mid \forall i : \vec{x} \models C_i \}$

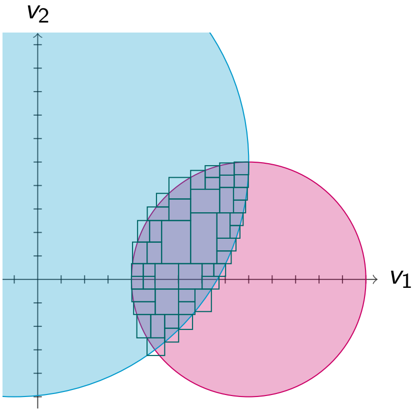
(also possible: look for a single solution instead of all solutions)

Constraint satisfaction problem solution

We would like to enumerate $\mathcal{S} \subseteq \mathbb{R}^2$, but this is impossible!
$\Longrightarrow$ instead, we cover $\mathcal{S}$ tightly with a finite set of boxes

$\mathcal{S}^\sharp$ set of boxes such that:

- $\mathcal{S} \subseteq \cup \mathcal{S}^\sharp$
- $\forall B \in \mathcal{S}^\sharp$:
  - either $B \subseteq \mathcal{S}$
  - or size$(B) \leq \epsilon$ and $B \cap \mathcal{S} \neq \emptyset$



(on discrete problems, solvers eventually enumerate $\mathcal{S}$)

Solving

Principle:

1. **Propagation:**
   use constraints to shrink domains
   remove domain values that are not part of a solution
   $\Longrightarrow$ consistency

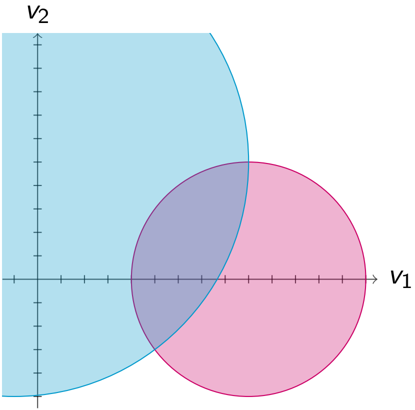
2. **Exploration:**
   split domains
   - halve an interval (continuous case)
   - instantiate a variable (discrete case)

   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)

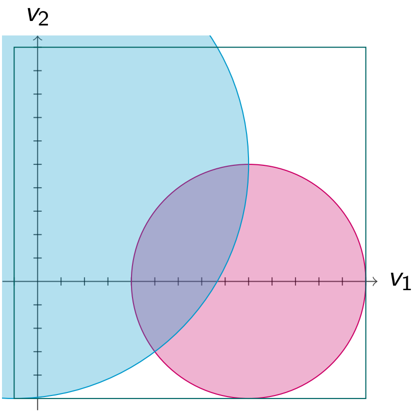
Constraint programming algorithm

- list of boxes toExplore := $\{ D \}$
- while toExplore is not empty



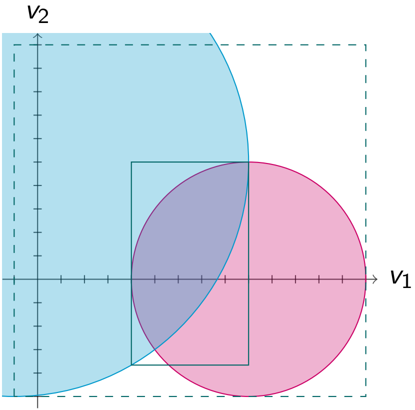
Constraint programming algorithm

- list of boxes toExplore $:= \{ D \}$
- while toExplore is not empty
  - pop a box from toExplore



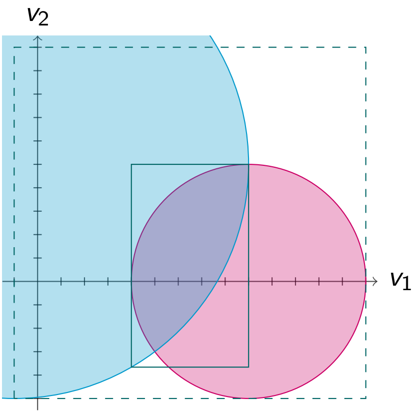
Constraint programming algorithm

- list of boxes toExplore $:= \{ D \}$
- while toExplore is not empty
  - pop a box from toExplore
  - consistency:
    shrink the box using the constraints



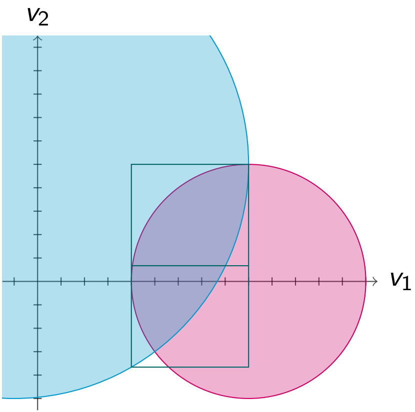
Constraint programming algorithm

- list of boxes toExplore := $\{ D \}$
- while toExplore is not empty
  - pop a box from toExplore
  - consistency:
    shrink the box using the constraints
  - if empty, continue
  - if small or contains only solutions
    shift it the solution list



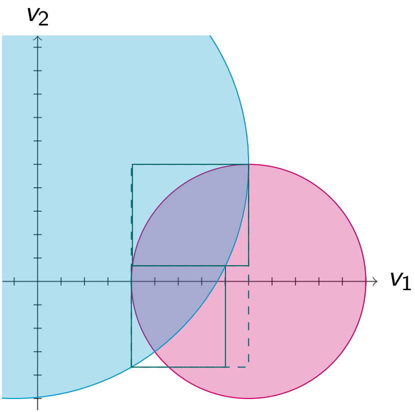
Constraint programming algorithm

- list of boxes toExplore := { D }
- while toExplore is not empty
  - pop a box from toExplore
  - consistency:
    shrink the box using the constraints
  - if empty, continue
  - if small or contains only solutions
    shift it the solution list
  - else
    split the box and
    push the pieces into toExplore



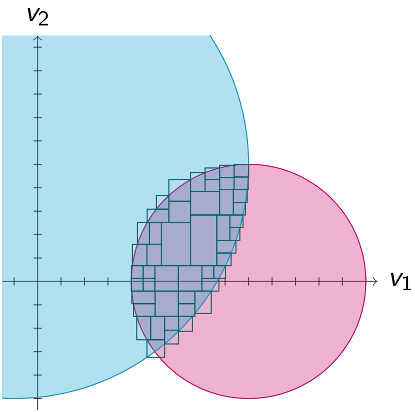
Constraint programming algorithm

- list of boxes toExplore := $\{ D \}$
- while toExplore is not empty
  - pop a box from toExplore
  - consistency:
    shrink the box using the constraints
  - if empty, continue
  - if small or contains only solutions
    shift it the solution list
  - else
    split the box and
    push the pieces into toExplore



Constraint programming algorithm

- list of boxes toExplore := { D }
- while toExplore is not empty
  - pop a box from toExplore
  - consistency:
    shrink the box using the constraints
  - if empty, continue
  - if small or contains only solutions
    shift it the solution list
  - else
    split the box and
    push the pieces into toExplore



Solving as fixpoint computation

Exact solution set:

Each constraint C_i gives rise to a filter function ρ_i:

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

Given $\rho \stackrel{\text{def}}{=} \rho_1 \circ \cdots \circ \rho_k$
the exact solution set S is a (trivial) concrete greatest fixpoint:

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

Interpreting the solving algorithm

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

$\Longrightarrow$ similar to fixpoint refinement with $\triangle$

Consistencies as abstract domains

Interpreting consistency:

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

Example

- Hull-consistency: interval abstraction with float bounds
  - $\mathcal{D}^\sharp \stackrel{\text{def}}{=} (\mathbb{F} \times \mathbb{F})^n$
  - $\alpha(X) \stackrel{\text{def}}{=} \lambda i. [\ \max\{\, x \in \mathbb{F} \mid \forall x_1, \ldots, x_n \in X : \ x_i \geq x \,\}, \ \min\{\, x \in \mathbb{F} \mid \forall x_1, \ldots, x_n \in X : \ x_i \leq x \,\}]$
- Arc-consistency: Cartesian abstraction
  - $\mathcal{D}^\sharp \stackrel{\text{def}}{=} \mathcal{P}(\mathbb{Z})^n$
  - $\alpha(X) \stackrel{\text{def}}{=} \lambda i. \{\, x \mid \exists x_1, \ldots, x_n \in X : \ x = x_i \,\}$

Propagation as test

Given a constraint C_i with filter ρ_i

- the consistency is the best abstraction $\alpha \circ \rho_i \circ \gamma$
- a propagator is a sound abstraction $\rho_i^\sharp$ of ρ_i

Examples:

- generic test $expr \leq 0$ in the interval domain
  - two-step algorithm known as HC4-revise in CP
  - independently rediscovered to implement tests in AI
- to abstract $\rho = \rho_1 \circ \cdots \circ \rho_n$: iterate $\rho_1^\sharp \circ \cdots \circ \rho_n^\sharp$

  Granger's local iterations

Split as disjunctive completion

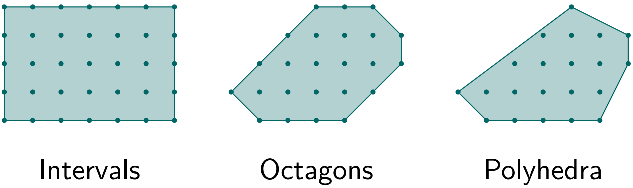
Solvers abstract solution sets using sets of domains
$\Longrightarrow$ corresponds to disjunctive completion in AI

$$\mathcal{PD}^\sharp \stackrel{\text{def}}{=} \mathcal{P}_{finite}(\mathcal{D}^\sharp)$$
$$\gamma_{\mathcal{P}}(\mathcal{X}^\sharp) \stackrel{\text{def}}{=} \bigcup \{ \gamma(X^\sharp) \mid X^\sharp \in \mathcal{X}^\sharp \}$$

Split operator $\oplus$:

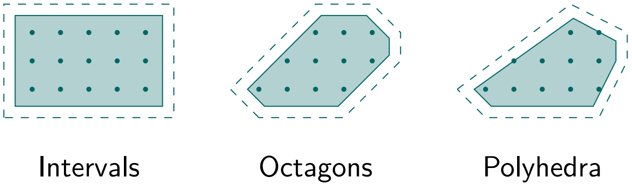
- $\oplus : \mathcal{PD}^\sharp \to \mathcal{PD}^\sharp$
- $\oplus(\mathcal{X}^\sharp)$ chooses an element in $X^\sharp \in \mathcal{X}^\sharp$
  and replaces it with finitely many elements in $\mathcal{D}^\sharp$
- $\gamma_{\mathcal{P}}(\oplus\mathcal{X}^\sharp) = \gamma_{\mathcal{P}}(\mathcal{X}^\sharp) \quad \Longrightarrow \oplus$ abstracts the identity
- $\oplus$ decreases for the Smyth order
  $$\mathcal{X}^\sharp \sqsubseteq^\sharp_{\mathcal{P}} \mathcal{Y}^\sharp \stackrel{\text{def}}{\iff} \forall X^\sharp \in \mathcal{X}^\sharp : \exists Y^\sharp \in \mathcal{Y}^\sharp : X^\sharp \sqsubseteq^\sharp Y^\sharp$$

Parameter



Parameter: an abstract domain $\mathcal{D}^\sharp$ equipped with:

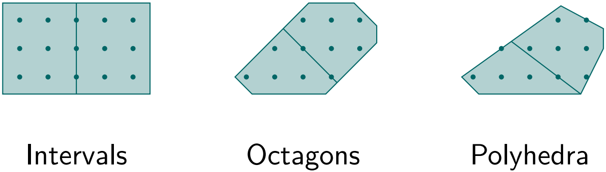
Parameter



Parameter: an abstract domain $\mathcal{D}^\sharp$ equipped with:

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

Parameter



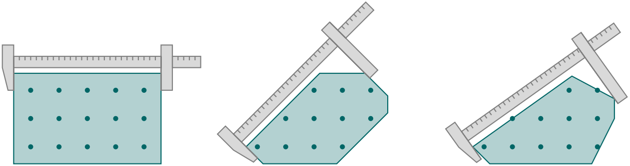
Parameter: an abstract domain $\mathcal{D}^\sharp$ equipped with:

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

and additionally:

- a splitting operator $\oplus : \mathcal{P}\mathcal{D}^\sharp \rightarrow \mathcal{P}\mathcal{D}^\sharp$ (exploration)

Parameter



Intervals | Octagons | Polyhedra

Parameter: an abstract domain $\mathcal{D}^\sharp$ equipped with:

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

and additionally:

- a splitting operator $\oplus : \mathcal{P}\mathcal{D}^\sharp \to \mathcal{P}\mathcal{D}^\sharp$ (exploration)
- a size function $\tau : \mathcal{D}^\sharp \to \mathbb{R}^+$ (termination criterion)

Interval domain

Hull-consistency for continuous solvers

- float interval abstraction: $\mathcal{D}^\sharp \stackrel{\text{def}}{=} (\mathbb{F} \times \mathbb{F})^n$
- split on variable v_i:

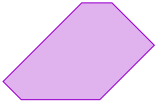
$$\oplus_i([\ell_1, h_1] \times \cdots \times [\ell_i, h_i] \times \cdots \times [\ell_n, h_n]) \stackrel{\text{def}}{=} \{[\ell_1, h_1] \times \cdots \times [\ell_i, m] \times \cdots \times [\ell_n, h_n], [\ell_1, h_1] \times \cdots \times [m, h_i] \times \cdots \times [\ell_n, h_n]\}$$

where $m \stackrel{\text{def}}{=} (\ell_i + h_i)/2$ (rounded indifferently)

- $\tau([\ell_1, h_1] \times \cdots \times [\ell_n, h_n]) \stackrel{\text{def}}{=} \max_i (h_i - \ell_i)$

Octagon domain

$\mathcal{D}^\sharp \stackrel{\text{def}}{=} \{\alpha v_i + \beta v_j \mid i, j \in [1, n], \alpha, \beta \in \{-1, 1\}\} \to \mathbb{F}$

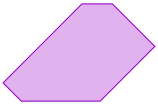


$\mathcal{D}^\sharp$: associates a (float) bound to each unit binary expression on V

Octagon domain

$$\mathcal{D}^\sharp \stackrel{\text{def}}{=} \{\alpha v_i + \beta v_j \mid i,j \in [1,n], \alpha,\beta \in \{-1,1\}\} \to \mathbb{F}$$

$$\tau(X^\sharp) \stackrel{\text{def}}{=} \min\Big(\max_{i,j,\beta} \left(X^\sharp(v_i + \beta v_j) + X^\sharp(-v_i - \beta v_j)\right), \max_i \left(X^\sharp(v_i + v_i) + X^\sharp(-v_i - v_i)\right)/2 \Big)$$

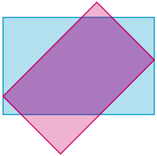


τ: size of the smallest box containing the octagon

Octagon domain

$$\mathcal{D}^\sharp \stackrel{\text{def}}{=} \{\alpha v_i + \beta v_j \mid i,j \in [1,n], \alpha,\beta \in \{-1,1\}\} \to \mathbb{F}$$

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

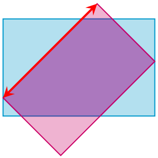


τ: size of the smallest box containing the octagon

Octagon domain

$$\mathcal{D}^\sharp \stackrel{\text{def}}{=} \{\alpha v_i + \beta v_j \mid i,j \in [1,n], \alpha,\beta \in \{-1,1\}\} \to \mathbb{F}$$

$$\tau(X^\sharp) \stackrel{\text{def}}{=} \min(\max_{i,j,\beta} \left(X^\sharp(v_i + \beta v_j) + X^\sharp(-v_i - \beta v_j)\right), \max_i \left(X^\sharp(v_i + v_i) + X^\sharp(-v_i - v_i)\right)/2 \quad)$$

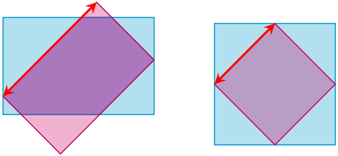


τ: size of the smallest box containing the octagon

Octagon domain

$$\mathcal{D}^\sharp \stackrel{\text{def}}{=} \{\alpha v_i + \beta v_j \mid i,j \in [1,n], \alpha,\beta \in \{-1,1\}\} \to \mathbb{F}$$

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



τ: size of the smallest box containing the octagon

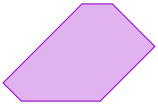
Octagon domain

$$\mathcal{D}^\sharp \stackrel{\text{def}}{=} \{\alpha v_i + \beta v_j \mid i,j \in [1,n], \alpha,\beta \in \{-1,1\}\} \to \mathbb{F}$$

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

$$\oplus(X^\sharp) \stackrel{\text{def}}{=} \left\{X^\sharp\left[(\alpha v_i + \beta v_j) \mapsto m\right], X^\sharp\left[(-\alpha v_i - \beta v_j) \mapsto -m\right]\right\}$$

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



$\oplus$: cuts in half perpendicular to the longest side

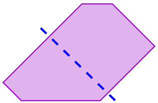
Octagon domain

$$\mathcal{D}^\sharp \stackrel{\text{def}}{=} \{\alpha v_i + \beta v_j \mid i,j \in [1,n], \alpha,\beta \in \{-1,1\}\} \to \mathbb{F}$$

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

$$\oplus(X^\sharp) \stackrel{\text{def}}{=} \left\{ X^\sharp\left[(\alpha v_i + \beta v_j) \mapsto m\right], X^\sharp\left[(-\alpha v_i - \beta v_j) \mapsto -m\right] \right\}$$

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



$\oplus$: cuts in half perpendicular to the longest side

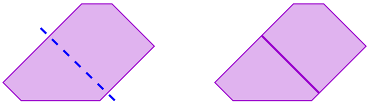
Octagon domain

$$\mathcal{D}^\sharp \stackrel{\text{def}}{=} \{\alpha v_i + \beta v_j \mid i,j \in [1,n], \alpha,\beta \in \{-1,1\}\} \to \mathbb{F}$$

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

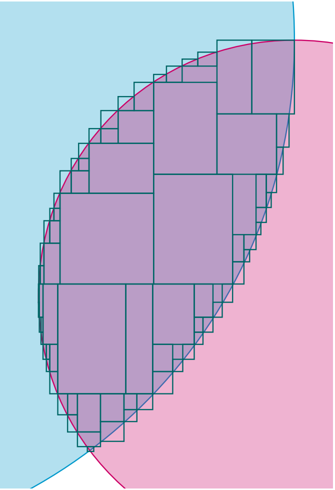
$$\oplus(X^\sharp) \stackrel{\text{def}}{=} \left\{X^\sharp[(\alpha v_i + \beta v_j) \mapsto m],\ X^\sharp[(-\alpha v_i - \beta v_j) \mapsto -m]\right\}$$

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

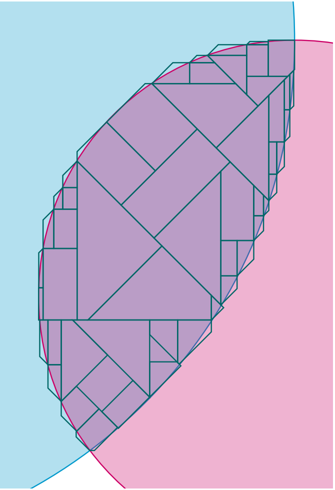


$\oplus$: cuts in half perpendicular to the longest side

Octagon solving example



Intervals



Octagons