Abstract Interpretation III

Semantics and Application to Program Verification

Antoine Miné

École normale supérieure, Paris

year 2015–2016

Course 12

20 May 2016

Overview

- Last week: non-relational abstract domains (intervals)

  abstract each variable independently from the others
  can express important properties (e.g., absence of overflow)
  unable to represent relations between variables

- This week: **relational abstract domains**

  more precise, but more costly

  - the need for relational domains
  - linear equality domain $(\sum_i \alpha_i V_i = \beta_i)$
  - polyhedra domain $(\sum_i \alpha_i V_i \geq \beta_i)$
  - extensions: weakly relational domains, integers, non-linear expressions
  - the Apron library
  - practical exercises: relational analysis with the Apron library

- Next week: selected advanced topics on abstract domains

Motivation

Relational assignments and tests

Example

$X \leftarrow \textbf{rand}(0,10);$
$Y \leftarrow \textbf{rand}(0,10);$
$\textbf{if } X \geq Y \textbf{ then } X \leftarrow Y \textbf{ else skip};$
$D \leftarrow Y - X;$
$\textbf{assert } D \geq 0$

Interval analysis:

- $\mathsf{S}^\sharp[\![\, X \geq Y ?\,]\!]$ is abstracted as the identity

  given $R^\sharp \stackrel{\text{def}}{=} [X \mapsto [0,10], Y \mapsto [0,10]]$

  $\mathsf{S}^\sharp[\![\, \textbf{if } X \geq Y \textbf{ then } \cdots]\!]\, R^\sharp = R^\sharp$

- $D \leftarrow Y - X$ gives $D \in [0,10] -^\sharp [0,10] = [-10,10]$

- the assertion $D \geq 0$ fails

Relational assignments and tests

Example

$X \leftarrow \textbf{rand}(0, 10);$
$Y \leftarrow \textbf{rand}(0, 10);$
$\textbf{if } X \geq Y \textbf{ then } X \leftarrow Y \textbf{ else skip};$
$D \leftarrow Y - X;$
$\textbf{assert } D \geq 0$

Solution: relational domain

- represent explicitly the information $X \leq Y$
- infer that $X \leq Y$ holds after the **if** $\cdots$ **then** $\cdots$ **else** $\cdots$
  $X \leq Y$ both after $X \leftarrow Y$ when $X \geq Y$, and after **skip** when $X < Y$
- use $X \leq Y$ to deduce that $Y - X \in [0, 10]$

Note:
the invariant we seek, $D \geq 0$, can be exactly represented in the interval domain,
but inferring $D \geq 0$ requires a more expressive domain locally

Relational loop invariants

Example

$I \leftarrow 1;\ X \leftarrow 0;$
while $I \leq 1000$ **do**
  $I \leftarrow I + 1;\ X \leftarrow X + 1;$
assert $X \leq 1000$

Interval analysis:

- after iterations with widening, we get in 2 iterations:
  as loop invariant: $I \in [1, +\infty]$ and $X \in [0, +\infty]$
  after the loop: $I \in [1001, +\infty]$ and $X \in [0, +\infty] \implies$ **assert** fails
- using a decreasing iteration after widening, we get:
  as loop invariant: $I \in [1, 1001]$ and $X \in [0, +\infty]$
  after the loop: $I = 1001$ and $X \in [0, +\infty] \implies$ **assert** fails
  (the test $I \leq 1000$ only refines I, but gives no information on X)
- without widening, we get $I = 1001$ and $X = 1000 \implies$ **assert** passes
  but we need 1000 iterations! ($\simeq$ concrete fixpoint computation)

Relational loop invariants

Example

$I \leftarrow 1;\ X \leftarrow 0;$
while $I \leq 1000$ **do**
    $I \leftarrow I + 1;\ X \leftarrow X + 1;$
assert $X \leq 1000$

Solution: relational domain

- infer a relational loop invariant: $I = X + 1 \wedge 1 \leq I \leq 1001$

  $I = X + 1$ holds before entering the loop as $1 = 0 + 1$

  $I = X + 1$ is invariant by the loop body $I \leftarrow I + 1; X \leftarrow X + 1$

  (can be inferred in 2 iterations with widening in the polyhedra domain)

- propagate the loop exit condition $I > 1000$ to get:

  $I = 1001$
  $X = I - 1 = 1000 \implies$ **assert** passes

Note:

the invariant we seek after the loop exit has an interval form: $X \leq 1000$
but we need to infer a more expressive loop invariant to deduce it

Relational procedure analysis

Example: $Z = \mathbf{max}(X, Y, 0)$

$Z \leftarrow X$;
if $Y > Z$ **then** $Z \leftarrow Y$;
if $Z < 0$ **then** $Z \leftarrow 0$

Relational procedure analysis

Example: $Z = \mathbf{max}(X, Y, 0)$

$X' \leftarrow X; Y' \leftarrow Y; Z' \leftarrow Z;$
$Z' \leftarrow X';$
if $Y' > Z'$ **then** $Z' \leftarrow Y';$
if $Z' < 0$ **then** $Z' \leftarrow 0$

- add and rename variables: keep a copy of input values

Relational procedure analysis

Example: $Z = \mathbf{max}(X, Y, 0)$

$X' \leftarrow X; Y' \leftarrow Y; Z' \leftarrow Z;$
$Z' \leftarrow X';$
if $Y' > Z'$ **then** $Z' \leftarrow Y';$
if $Z' < 0$ **then** $Z' \leftarrow 0$
$// \; Z' \geq X \wedge Z' \geq Y \wedge Z' \geq 0 \wedge X' = X \wedge Y' = Y$

- add and rename variables: keep a copy of input values
- infer a relation between input values (X, Y, Z) and current values (X', Y', Z')

Applications: procedure summaries, modular analysis.

Affine Equalities

The affine equality domain

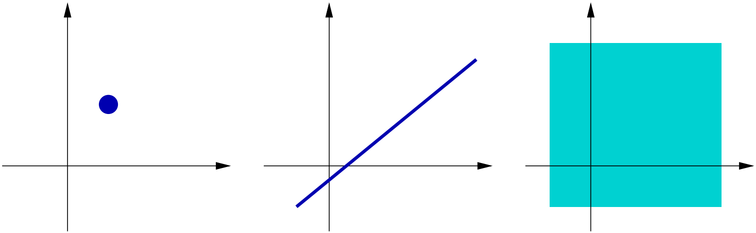
We look for invariants of the form:

$$\wedge_j \left(\sum_{i=1}^{n} \alpha_{ij} V_i = \beta_j\right),\ \alpha_{ij}, \beta_j \in \mathbb{Q}$$

where all the α_{ij} and β_j are inferred automatically

We use a domain of affine spaces proposed by Karr in 1976

$$\mathcal{E}^\sharp \simeq \{\text{ affine subspaces of } \mathbb{V} \to \mathbb{R} \}$$



Notes: we reason in $\mathbb{R}$ to use results from linear algebra
we use coefficients in $\mathbb{Q}$ to be machine representable

Affine equality representation

Machine representation:

$$\mathcal{E}^\sharp \stackrel{\text{def}}{=} \cup_m \ \{\, \langle \mathbf{M}, \vec{C} \rangle \,|\, \mathbf{M} \in \mathbb{Q}^{m\times n}, \vec{C} \in \mathbb{Q}^m \,\} \cup \{\perp\}$$

- either the constant $\perp$
- or a pair $\langle \mathbf{M}, \vec{C} \rangle$ where
  - $\mathbf{M} \in \mathbb{Q}^{m\times n}$ is a $m \times n$ matrix, $n = |\mathbb{V}|$ and $m \leq n$,
  - $\vec{C} \in \mathbb{Q}^m$ is a row-vector with m rows

$\langle \mathbf{M}, \vec{C} \rangle$ represents an equation system, with solutions:

$$\gamma(\langle \mathbf{M}, \vec{C} \rangle) \stackrel{\text{def}}{=} \{\, \vec{V} \in \mathbb{R}^n \,|\, \mathbf{M} \times \vec{V} = \vec{C} \,\}$$

$\mathbf{M}$ should be in row echelon form:

- $\forall i \leq m: \exists k_i: M_{ik_i} = 1$ and
  $\forall c < k_i: M_{ic} = 0,\ \forall l \neq i: M_{lk_i} = 0$,
- if $i < i'$ then $k_i < k_{i'}$ (leading index)

example:

$$\begin{bmatrix} 1 & 0 & 0 & 5 & 0 \\ 0 & 1 & 0 & 6 & 0 \\ 0 & 0 & 1 & 7 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Remarks:

the representation is unique
as $m \leq n = |\mathbb{V}|$, the memory cost is in $\mathcal{O}(n^2)$ at worst
$\top$ is represented as the empty equation system: $m = 0$

Galois connection

Galois connection: (actually, a Galois insertion)

between arbitrary subsets and affine subsets

$$(\mathcal{P}(\mathbb{R}^{|\mathbb{V}|}), \subseteq) \xleftarrow[\alpha]{\gamma} (\mathit{Aff}(\mathbb{R}^{|\mathbb{V}|}), \subseteq)$$

- $\gamma(X) \stackrel{\text{def}}{=} X$ (identity)
- $\alpha(X) \stackrel{\text{def}}{=}$ smallest affine subset containing X

  $\mathit{Aff}(\mathbb{R}^{|\mathbb{V}|})$ is closed under arbitrary intersections, so we have:

  $\alpha(X) = \cap \{ Y \in \mathit{Aff}(\mathbb{R}^{|\mathbb{V}|}) \mid X \subseteq Y \}$

  $\mathit{Aff}(\mathbb{R}^{|\mathbb{V}|})$ contains every point in $\mathbb{R}^{|\mathbb{V}|}$

  we can also construct $\alpha(X)$ by (abstract) union:
  $\alpha(X) = \cup^\sharp \{ \{x\} \mid x \in X \}$

Notes:

- we have assimilated $\mathbb{V} \to \mathbb{R}$ to $\mathbb{R}^{|\mathbb{V}|}$
- we have used $\mathit{Aff}(\mathbb{R}^{|\mathbb{V}|})$ instead of the matrix representation $\mathcal{E}^\sharp$ for simplicity; a Galois connection also exists between $\mathcal{P}(\mathbb{R}^{|\mathbb{V}|})$ and $\mathcal{E}^\sharp$

Normalisation and emptiness testing

Let $\mathbf{M} \times \vec{V} = \vec{C}$ be a system, not necessarily in normal form

The Gaussian reduction $\mathit{Gauss}(\langle \mathbf{M}, \vec{C} \rangle)$ with $\mathcal{O}(n^3)$ time:

- tells whether the system is satisfiable
- gives an equivalent system in normal form
  i.e., it returns an element in $\mathcal{E}^\sharp$
- by combining rows linearly to remove variable occurrences

Example:

$$\left\{ \begin{array}{rcrcrcr} 2X & + & Y & + & Z & = & 19 \\ 2X & + & Y & - & Z & = & 9 \\ & & & & 3Z & = & 15 \end{array} \right.$$

$$\Downarrow$$

$$\left\{ \begin{array}{rcrcrcr} X & + & 0.5Y & & & = & 7 \\ & & & & Z & = & 5 \end{array} \right.$$

Affine equality operators

Abstract operators:

If $X^\sharp, Y^\sharp \neq \bot$, we define:

$$X^\sharp \cap^\sharp Y^\sharp \stackrel{\text{def}}{=} \mathit{Gauss}\left(\left\langle \begin{bmatrix} \mathbf{M}_{X^\sharp} \\ \mathbf{M}_{Y^\sharp} \end{bmatrix}, \begin{bmatrix} \vec{C}_{X^\sharp} \\ \vec{C}_{Y^\sharp} \end{bmatrix} \right\rangle\right) \qquad \textit{(join equations)}$$

$$X^\sharp =^\sharp Y^\sharp \stackrel{\text{def}}{\iff} \mathbf{M}_{X^\sharp} = \mathbf{M}_{Y^\sharp} \text{ and } \vec{C}_{X^\sharp} = \vec{C}_{Y^\sharp} \qquad \textit{(uniqueness)}$$

$$X^\sharp \subseteq^\sharp Y^\sharp \stackrel{\text{def}}{\iff} X^\sharp \cap^\sharp Y^\sharp =^\sharp X^\sharp$$

$$S^\sharp[\![\textstyle\sum_j \alpha_j V_j = \beta?]\!] X^\sharp \stackrel{\text{def}}{=} \mathit{Gauss}\left(\left\langle \begin{bmatrix} \mathbf{M}_{X^\sharp} \\ \alpha_1 \cdots \alpha_n \end{bmatrix}, \begin{bmatrix} \vec{C}_{X^\sharp} \\ \beta \end{bmatrix} \right\rangle\right) \qquad \textit{(add equation)}$$

$$S^\sharp[\![e \bowtie e'?]\!] X^\sharp \stackrel{\text{def}}{=} X^\sharp \qquad \text{for other tests}$$

Remark:

$\subseteq^\sharp$, $=^\sharp$, $\cap^\sharp$, $=^\sharp$ and $S^\sharp[\![\sum_j \alpha_j V_j - \beta = 0?]\!]$ are exact:

$(X^\sharp \subseteq^\sharp Y^\sharp \iff \gamma(X^\sharp) \subseteq \gamma(Y^\sharp), \quad \gamma(X^\sharp \cap^\sharp Y^\sharp) = \gamma(X^\sharp) \cap \gamma(Y^\sharp), \ldots)$

Affine equality assignment

Non-deterministic assignment: $S^\sharp[\![V_j \leftarrow [-\infty, +\infty]]\!]$

Principle: remove all the occurrences of V_j
but reduce the number of equations by only one
(add a single degree of freedom)

Algorithm: assuming V_j occurs in $\mathbf{M}$

- Pick the row $\langle \vec{M}_i, C_i \rangle$ such that $M_{ij} \neq 0$ and i maximal
- Use it to eliminate all the occurrences of V_j in lines before i
  (i maximal $\implies$ $\mathbf{M}$ stays in row echelon form)
- Remove the row $\langle \vec{M}_i, C_i \rangle$

Example: forgetting Z

$$\left\{ \begin{array}{rl} X \quad + Z &= 10 \\ Y + Z &= 7 \end{array} \right. \implies \left\{ \, X - Y = 3 \right.$$

The operator is exact

Affine equality assignment

Affine assignments: $S^\sharp[\![V_j \leftarrow \sum_i \alpha_i V_i + \beta]\!]$

$S^\sharp[\![V_j \leftarrow \sum_i \alpha_i V_i + \beta]\!] X^\sharp \stackrel{\text{def}}{=}$

if $\alpha_j = 0$, $(S^\sharp[\![V_j = \sum_i \alpha_i V_i + \beta?]\!] \circ S^\sharp[\![V_j \leftarrow [-\infty, +\infty]]\!])X^\sharp$

if $\alpha_j \neq 0$, $\langle \mathbf{M}, \vec{C} \rangle$ where V_j is replaced with $\frac{1}{\alpha_j}(V_j - \sum_{i \neq j} \alpha_i V_i - \beta)$

(variable substitution)

Proof sketch: based on properties in the concrete

non-invertible assignment: $\alpha_j = 0$

$S[\![V_j \leftarrow e]\!] = S[\![V_j \leftarrow e]\!] \circ S[\![V_j \leftarrow [-\infty, +\infty]]\!]$ as the value of V is not used in e

so $S[\![V_j \leftarrow e]\!] = S[\![V_j = e?]\!] \circ S[\![V_j \leftarrow [-\infty, +\infty]]\!]$

invertible assignment: $\alpha_j \neq 0$

$S[\![V_j \leftarrow e]\!] \subsetneq S[\![V_j \leftarrow e]\!] \circ S[\![V_j \leftarrow [-\infty, +\infty]]\!]$ as e depends on V

$$\begin{aligned}
\rho \in S[\![V_j \leftarrow e]\!] R &\iff \exists \rho' \in R\colon\ \rho = \rho'[V_j \mapsto \textstyle\sum_i \alpha_i \rho'(V_i) + \beta] \\
&\iff \exists \rho' \in R\colon\ \rho[V_j \mapsto (\rho(V_j) - \textstyle\sum_{i \neq j} \alpha_i \rho'(V_i) - \beta)/\alpha_j] = \rho' \\
&\iff \qquad\qquad \rho[V_j \mapsto (\rho(V_j) - \textstyle\sum_{i \neq j} \alpha_i \rho(V_i) - \beta)/\alpha_j] \in R
\end{aligned}$$

Non-affine assignments: revert to non-deterministic case

$S^\sharp[\![V_j \leftarrow e]\!] X^\sharp \stackrel{\text{def}}{=} S^\sharp[\![V_j \leftarrow [-\infty, +\infty]]\!] X^\sharp$ (imprecise but sound)

Affine equality join

Join: $\langle \mathbf{M}, \vec{C} \rangle \cup^\sharp \langle \mathbf{N}, \vec{D} \rangle$

Idea: unify columns 1 to n of $\langle \mathbf{M}, \vec{C} \rangle$ and $\langle \mathbf{N}, \vec{D} \rangle$ using row operations

Example:

Assume that we have unified columns 1 to k to get $\begin{pmatrix} \mathbf{R} \\ \mathbf{0} \end{pmatrix}$, arguments are in row echelon form, and we have to unify at column $k+1$: ${}^t(\vec{0}\ 1\ \vec{0})$ with ${}^t(\vec{\beta}\ 0\ \vec{0})$

$$\begin{pmatrix} \mathbf{R} & \vec{0} & \mathbf{M_1} \\ \vec{0} & 1 & \vec{M_2} \\ \mathbf{0} & \vec{0} & \mathbf{M_3} \end{pmatrix}, \begin{pmatrix} \mathbf{R} & \vec{\beta} & \mathbf{N_1} \\ \vec{0} & 0 & \vec{N_2} \\ \mathbf{0} & \vec{0} & \mathbf{N_3} \end{pmatrix} \Longrightarrow \begin{pmatrix} \mathbf{R} & \vec{\beta} & \mathbf{M'_1} \\ \vec{0} & 0 & \vec{0} \\ \mathbf{0} & \vec{0} & \mathbf{M_3} \end{pmatrix}, \begin{pmatrix} \mathbf{R} & \vec{\beta} & \mathbf{N_1} \\ \vec{0} & 0 & \vec{N_2} \\ \mathbf{0} & \vec{0} & \mathbf{N_3} \end{pmatrix}$$

Use the row $(\vec{0}\ 1\ \vec{M_2})$ to create $\vec{\beta}$ in the left argument
Then remove the row $(\vec{0}\ 1\ \vec{M_2})$
The right argument is unchanged
$\Longrightarrow$ we have now unified columns 1 to $k+1$

Unifying ${}^t(\vec{\alpha}\ 0\ \vec{0})$ and ${}^t(\vec{0}\ 1\ \vec{0})$ is similar
Unifying ${}^t(\vec{\alpha}\ 0\ \vec{0})$ and ${}^t(\vec{\beta}\ 0\ \vec{0})$ is a bit more complicated...
No other case possible as we are in row echelon form

Analysis example

No infinite increasing chain: we can iterate without widening!

Example

$X \leftarrow 10$; $Y \leftarrow 100$;
while $X \neq 0$ **do**
  $X \leftarrow X - 1$;
  $Y \leftarrow Y + 10$

Abstract loop iterations: $\lim \lambda X^\sharp . I^\sharp \cup^\sharp \mathsf{S}^\sharp[\![\, body \,]\!] (\mathsf{S}^\sharp[\![\, X \neq 0? \,]\!]\, X^\sharp)$

- loop entry: $I^\sharp = (X = 10 \wedge Y = 100)$
- after one loop body iteration: $F^\sharp(I^\sharp) = (X = 9 \wedge Y = 110)$
- $\Longrightarrow X^\sharp \stackrel{\text{def}}{=} I^\sharp \cup^\sharp F^\sharp(I^\sharp) = (10X + Y = 200)$
- $X^\sharp$ is stable

at loop exit, we get $\mathsf{S}^\sharp[\![\, X = 0? \,]\!]\, (10X + Y = 200) = (X = 0 \wedge Y = 200)$

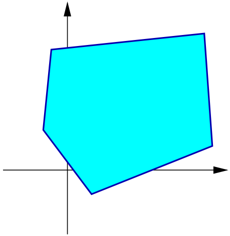
Polyhedra

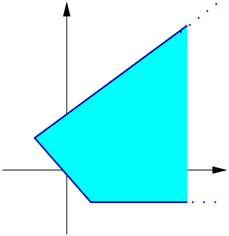
The polyhedra domain

We look for invariants of the form: $\wedge_j \left(\sum_{i=1}^{n} \alpha_{ij} V_i \geq \beta_j\right)$

We use the polyhedra domain by Cousot and Halbwachs (1978)

$\mathcal{E}^\sharp \simeq \{$ closed convex polyhedra of $\mathbb{V} \to \mathbb{R} \}$





Notes:
- polyhedra need not be bounded ($\neq$ polytopes)
- we keep reasoning in $\mathbb{R}$, to use affine theory

Double description of polyhedra

Polyhedra have dual representations (Weyl–Minkowski Theorem)

Constraint representation

$\langle \mathbf{M}, \vec{C} \rangle$ with $\mathbf{M} \in \mathbb{Q}^{m \times n}$ and $\vec{C} \in \mathbb{Q}^m$
represents: $\gamma(\langle \mathbf{M}, \vec{C} \rangle) \stackrel{\text{def}}{=} \{ \vec{V} \mid \mathbf{M} \times \vec{V} \geq \vec{C} \}$

We will also often use a constraint set notation: $\{ \sum_i \alpha_{ij} V_i \geq \beta_j \}$

Generator representation

$[\mathbf{P}, \mathbf{R}]$ where

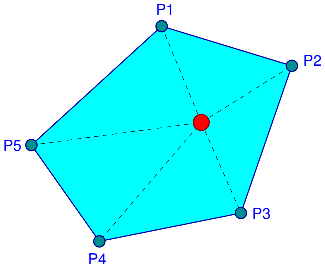
- $\mathbf{P} \in \mathbb{Q}^{n \times p}$ is a set of p points: $\vec{P}_1, \ldots, \vec{P}_p$
- $\mathbf{R} \in \mathbb{Q}^{n \times r}$ is a set of r rays: $\vec{R}_1, \ldots, \vec{R}_r$

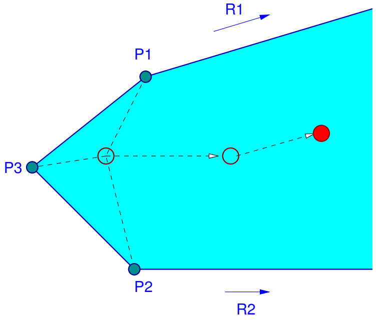
$\gamma([\mathbf{P}, \mathbf{R}]) \stackrel{\text{def}}{=} \{ (\sum_{j=1}^{p} \alpha_j \vec{P}_j) + (\sum_{j=1}^{r} \beta_j \vec{R}_j) \mid \forall j, \alpha_j, \beta_j \geq 0\text{: } \sum_{j=1}^{p} \alpha_j = 1 \}$

Double description of polyhedra (cont.)

Generator representation examples:

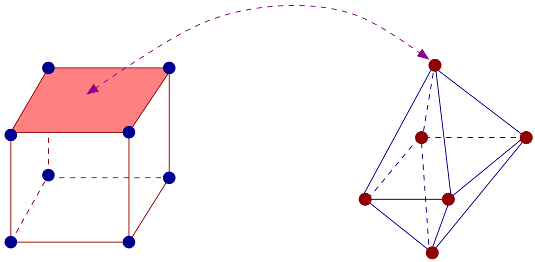
$$\gamma([\mathbf{P},\mathbf{R}]) \stackrel{\text{def}}{=} \{\, (\sum_{j=1}^{p} \alpha_j \vec{P}_j) + (\sum_{j=1}^{r} \beta_j \vec{R}_j) \mid \forall j, \alpha_j, \beta_j \geq 0 \colon \sum_{j=1}^{p} \alpha_j = 1 \,\}$$





- the points define a bounded convex hull
- the rays allow unbounded polyhedra

Duality in polyhedra



$$P \qquad\qquad\qquad P^*$$

$$0x + 0y + 1z \leq 1 \qquad \Longleftrightarrow \qquad (0, 0, 1)$$

Duality: P^* is the dual of P, so that:

- the generators of P^* are the constraints of P
- the constraints of P^* are the generators of P
- $P^{**} = P$

Double description: pros and cons

Pros:

Abstract operations are generally easy on one of the representations

which representation is best depends on the operation

e.g., constraints for $\cap^\sharp$, generators for $\cup^\sharp$

$\implies$ polyhedra operations are reduced to a single complex algorithm: changing one representation into the other

Cons:

Changing the representation can be costly
and cause a combinatorial explosion in the size of the representation!

Example: a hypercube in $\mathbb{R}^n$ with axis-aligned faces

- $2n$ contraints
- but 2^n generators (vertices of the hypercube)
- yet, hypercubes occur frequently in program analysis!

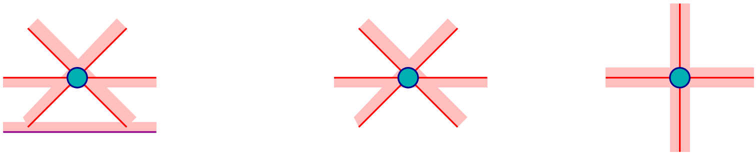
We are not free to choose the most compact representation
but have to use the representation required by our operation...

Uniqueness, minimality

Minimal representations

- A constraint / generator system is minimal if no constraint / generator can be omitted without changing the concretization
- Minimal representations are not unique

Example: three different constraint representations for a point

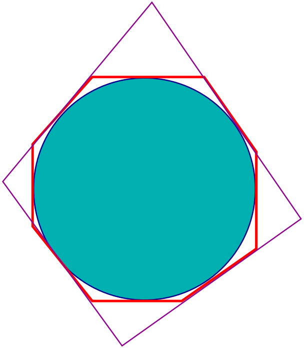


(a) (b) (c)

- (a) $y + x \geq 0, y - x \geq 0, y \leq 0, y \geq -5$ (non mimimal)
- (b) $y + x \geq 0, y - x \geq 0, y \leq 0$ (minimal)
- (c) $x \leq 0, x \geq 0, y \leq 0, y \geq 0$ (minimal)

Bound on polyhedra

- There is no bound on the size of the representation of polyhedra
  even for minimal representations
- There is no abstraction operator α
  no optimal abstraction as polyhedra for some sets of points
  $\Longrightarrow$ no Galois connection,
  no best abstraction for arbitrary operators



Example:

a disc has infinitely many polyhedral over-approximations

no approximation is the best one

Representation change: Chernikova's algorithm

Chernikova's algorithm (1968), improved by LeVerge (1992):

- changes a constraint system into an equivalent generator system
- by duality, also changes a generator system into an equivalent constraint system
- also minimizes the representation

Intuition: incremental algorithm

- start from a generator representation of $\mathbb{R}^n$
- add constraints one by one
- filter generators to keep only those that satisfy the new constraint
- move generators to force them to satisfy the new constraint
  i.e., they must *saturate* the constraint

Chernikova's algorithm

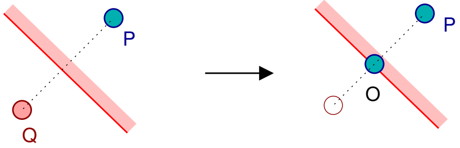
Algorithm: incrementally add constraints one by one

Start with: $\begin{cases} \mathbf{P}_0 = \{\ (0,\ldots,0)\ \} & \text{(origin)} \\ \mathbf{R}_0 = \{\ \vec{x}_i,\ -\vec{x}_i \mid 1 \leq i \leq n\ \} & \text{(axes)} \end{cases}$

For each constraint $\vec{M}_k \cdot \vec{V} \geq C_k \in \langle \mathbf{M}, \vec{C} \rangle$, update $[\mathbf{P}_{k-1}, \mathbf{R}_{k-1}]$ to $[\mathbf{P}_k, \mathbf{R}_k]$.

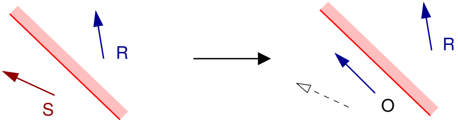
Start with $\mathbf{P}_k = \mathbf{R}_k = \emptyset$,

- for any $\vec{P} \in \mathbf{P}_{k-1}$ s.t. $\vec{M}_k \cdot \vec{P} \geq C_k$, add $\vec{P}$ to $\mathbf{P}_k$
- for any $\vec{R} \in \mathbf{R}_{k-1}$ s.t. $\vec{M}_k \cdot \vec{R} \geq 0$, add $\vec{R}$ to $\mathbf{R}_k$
- for any $\vec{P}, \vec{Q} \in \mathbf{P}_{k-1}$ s.t. $\vec{M}_k \cdot \vec{P} > C_k$ and $\vec{M}_k \cdot \vec{Q} < C_k$, add to $\mathbf{P}_k$:
  $\vec{O} \stackrel{\text{def}}{=} \frac{C_k - \vec{M}_k \cdot \vec{Q}}{\vec{M}_k \cdot \vec{P} - \vec{M}_k \cdot \vec{Q}} \vec{P} - \frac{C_k - \vec{M}_k \cdot \vec{P}}{\vec{M}_k \cdot \vec{P} - \vec{M}_k \cdot \vec{Q}} \vec{Q}$

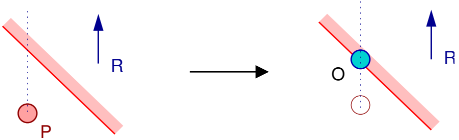


Chernikova's algorithm (cont.)

- for any $\vec{R}, \vec{S} \in \mathbf{R}_{k-1}$ s.t. $\vec{M}_k \cdot \vec{R} > 0$ and $\vec{M}_k \cdot \vec{S} < 0$, add to $\mathbf{R}_k$:
  $\vec{O} \stackrel{\text{def}}{=} (\vec{M}_k \cdot \vec{S})\vec{R} - (\vec{M}_k \cdot \vec{R})\vec{S}$

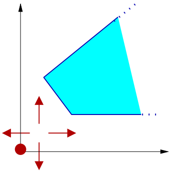


- for any $\vec{P} \in \mathbf{P}_{k-1}$, $\vec{R} \in \mathbf{R}_{k-1}$ s.t.
  either $\vec{M}_k \cdot \vec{P} > C_k$ and $\vec{M}_k \cdot \vec{R} < 0$, or $\vec{M}_k \cdot \vec{P} < C_k$ and $\vec{M}_k \cdot \vec{R} > 0$
  add to $\mathbf{P}_k$: $\vec{O} \stackrel{\text{def}}{=} \vec{P} + \frac{C_k - \vec{M}_k \cdot \vec{P}}{\vec{M}_k \cdot \vec{R}} \vec{R}$



Chernikova's algorithm example

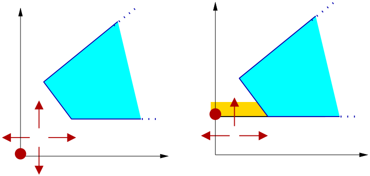
Example:



$\mathbf{P}_0 = \{(0,0)\}$ $\qquad$ $\mathbf{R}_0 = \{(1,0),\ (-1,0),\ (0,1),\ (0,-1)\}$

Chernikova's algorithm example

Example:



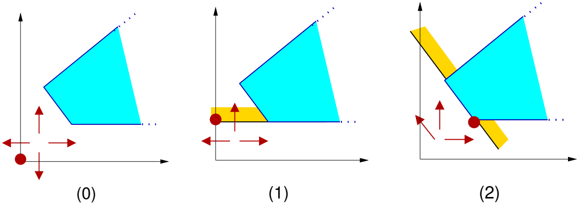
(0) (1)

$$Y \geq 1 \qquad \mathbf{P}_0 = \{(0,0)\} \qquad \mathbf{R}_0 = \{(1,0),\ (-1,0),\ (0,1),\ (0,-1)\}$$

$$\mathbf{P}_1 = \{(0,1)\} \qquad \mathbf{R}_1 = \{(1,0),\ (-1,0),\ (0,1)\}$$

Chernikova's algorithm example

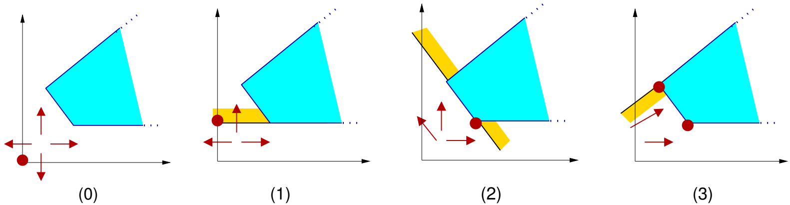
Example:



$$\begin{array}{lll}
 & \mathbf{P}_0 = \{(0,0)\} & \mathbf{R}_0 = \{(1,0),\ (-1,0),\ (0,1),\ (0,-1)\} \\
Y \geq 1 & \mathbf{P}_1 = \{\color{blue}{(0,1)}\} & \mathbf{R}_1 = \{(1,0),\ (-1,0),\ (0,1)\} \\
X + Y \geq 3 & \mathbf{P}_2 = \{\color{blue}{(2,1)}\} & \mathbf{R}_2 = \{(1,0),\ \color{blue}{(-1,1)},\ (0,1)\}
\end{array}$$

Chernikova's algorithm example

Example:



	$\mathbf{P}_0 = \{(0,0)\}$	$\mathbf{R}_0 = \{(1,0),\ (-1,0),\ (0,1),\ (0,-1)\}$
$Y \geq 1$	$\mathbf{P}_1 = \{(0,1)\}$	$\mathbf{R}_1 = \{(1,0),\ (-1,0),\ (0,1)\}$
$X + Y \geq 3$	$\mathbf{P}_2 = \{(2,1)\}$	$\mathbf{R}_2 = \{(1,0),\ (-1,1),\ (0,1)\}$
$X - Y \leq 1$	$\mathbf{P}_3 = \{(2,1),\ (1,2)\}$	$\mathbf{R}_3 = \{(0,1),\ (1,1)\}$

we omit redundant generators; they are removed by the full version of the algorithm

Polyhedral abstract operators

Set-theoretic operations:

Assuming $X^\sharp, Y^\sharp \neq \bot$, we define:

$$X^\sharp \subseteq^\sharp Y^\sharp \quad \stackrel{\text{def}}{\iff} \quad \begin{cases} \forall \vec{P} \in \mathbf{P}_{X^\sharp} : \mathbf{M}_{Y^\sharp} \times \vec{P} \geq \vec{C}_{Y^\sharp} \\ \forall \vec{R} \in \mathbf{R}_{X^\sharp} : \mathbf{M}_{Y^\sharp} \times \vec{R} \geq \vec{0} \end{cases}$$

every generator in $X^\sharp$ must satisfy every constraint in $Y^\sharp$

$$X^\sharp =^\sharp Y^\sharp \quad \stackrel{\text{def}}{\iff} \quad X^\sharp \subseteq^\sharp Y^\sharp \text{ et } Y^\sharp \subseteq^\sharp X^\sharp$$

both inclusion

$$X^\sharp \cap^\sharp Y^\sharp \quad \stackrel{\text{def}}{=} \quad \left\langle \begin{bmatrix} \mathbf{M}_{X^\sharp} \\ \mathbf{M}_{Y^\sharp} \end{bmatrix}, \begin{bmatrix} \vec{C}_{X^\sharp} \\ \vec{C}_{Y^\sharp} \end{bmatrix} \right\rangle$$

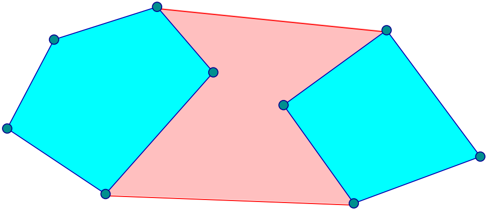
union of constraint sets

$\subseteq^\sharp$, $=^\sharp$ and $\cap^\sharp$ are exact in $\mathcal{P}(\mathbb{V} \to \mathbb{R})$

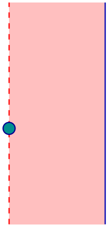
Polyhedral abstract operators (cont.)

Union: $X^\sharp \cup^\sharp Y^\sharp \stackrel{\text{def}}{=} [\,[\mathbf{P}_{X^\sharp}\ \mathbf{P}_{Y^\sharp}],\ [\mathbf{R}_{X^\sharp}\ \mathbf{R}_{Y^\sharp}]\,]$ union of generator sets

Examples:



two bounded polyhedra



a point and line

$\cup^\sharp$ is optimal in $\mathcal{P}(\mathbb{V} \to \mathbb{R})$

α is not always defined, but $\alpha(\gamma(X^\sharp) \cup \gamma(Y^\sharp))$ always exists

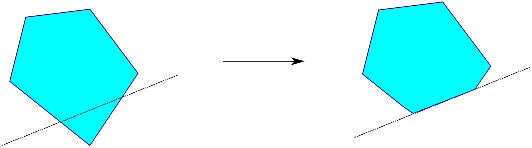
$\Longrightarrow$ topological closure of the convex hull of of $\gamma(X^\sharp) \cup \gamma(Y^\sharp)$

Polyhedral abstract operators (cont.)

Affine test :

$$S^\sharp[\![\, \sum_i \alpha_i V_i \geq \beta ?\,]\!]\, X^\sharp \stackrel{\text{def}}{=} \left\langle \begin{bmatrix} \mathbf{M}_{X^\sharp} \\ \alpha_1 \cdots \alpha_n \end{bmatrix}, \begin{bmatrix} \vec{C}_{X^\sharp} \\ \beta \end{bmatrix} \right\rangle$$

$$S^\sharp[\![\, \sum_i \alpha_i V_i = \beta ?\,]\!]\, X^\sharp \stackrel{\text{def}}{=} S^\sharp[\![\, \sum_i \alpha_i V_i \geq -\beta ?\,]\!]\, (S^\sharp[\![\, \sum_i (-\alpha_i) V_i \geq \beta ?\,]\!]\, X^\sharp)$$



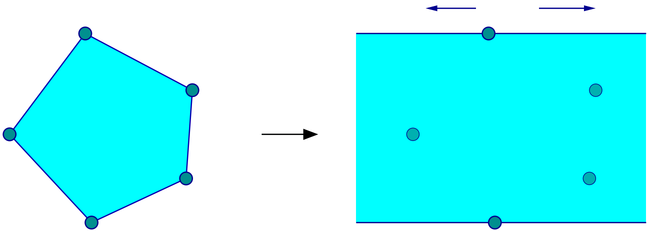
- simply adds a constraint to the constraint set
- the operators are exact
- the other tests can be abstracted as $S^\sharp[\![\, c\,]\!]\, X^\sharp \stackrel{\text{def}}{=} X^\sharp$

  sound but very imprecise

Polyhedral abstract operators (cont.)

Non-deterministic assignment:

$$\mathsf{S}^\sharp[\![\, V_j \leftarrow \textbf{rand}(-\infty,+\infty)\,]\!]\, X^\sharp \stackrel{\text{def}}{=} [\, \mathbf{P}_{X^\sharp},\ [\, \mathbf{R}_{X^\sharp}\ \ \vec{x}_j\ \ (-\vec{x}_j)\,]\,]$$



- in the concrete:
  $\mathsf{S}[\![\, V_j \leftarrow \textbf{rand}(-\infty,+\infty)\,]\!]\, R = \{\, \rho[V_j \mapsto v] \mid \rho \in R,\ v \in \mathbb{R}\,\}$
- in the abstract:
  add two rays parallel to the "forgotten" variable
- exact operator in $\mathcal{P}(\mathbb{V} \to \mathbb{R})$

Operators on polyhedra (cont.)

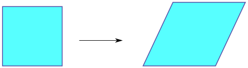
Affine assignment:

$\mathsf{S}^\sharp[\![V_j \leftarrow \sum_i \alpha_i V_i + \beta]\!] X^\sharp \stackrel{\text{def}}{=}$

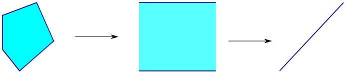
$$\text{if } \alpha_j \neq 0, \langle \mathbf{M}, \vec{C} \rangle \text{ where } V_j \text{ is replaced with } \frac{1}{\alpha_j}(V_j - \textstyle\sum_{i \neq j} \alpha_i V_i - \beta)$$

$$\text{if } \alpha_j = 0, (\mathsf{S}^\sharp[\![\textstyle\sum_i \alpha_i V_i = V_j - \beta ?]\!] \circ \mathsf{S}^\sharp[\![V_j \leftarrow [-\infty, +\infty]]\!])X^\sharp$$

Examples : $X \leftarrow X + Y$



$X \leftarrow Y$



- similar to the assignment in the equality domain
- the assignment is exact (in $\mathcal{P}(\mathbb{V} \to \mathbb{R})$)
- assignments can also be defined on the generator system
- for non-affine assignments: $\mathsf{S}^\sharp[\![V \leftarrow e]\!] \stackrel{\text{def}}{=} \mathsf{S}^\sharp[\![V \leftarrow [-\infty, +\infty]]\!]$ (sound but not optimal)

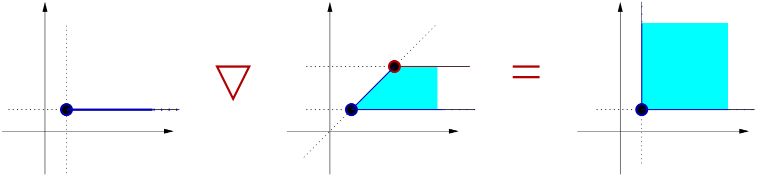
Naive widening on polyhedra

$\mathcal{E}^\sharp$ has strictly increasing infinite chains $\implies$ we need a widening

Ddfinition: $X^\sharp \triangledown Y^\sharp \stackrel{\text{def}}{=} \{\, c \in X^\sharp \mid Y^\sharp \subseteq^\sharp \{c\} \,\}$

- keep the constraints from $X^\sharp$ satisfied by $Y^\sharp$
- unlike $\cup^\sharp$, no new constraint is created
- $\triangledown$ reduces the set of constraints
  $\implies$ ensures termination

Example:



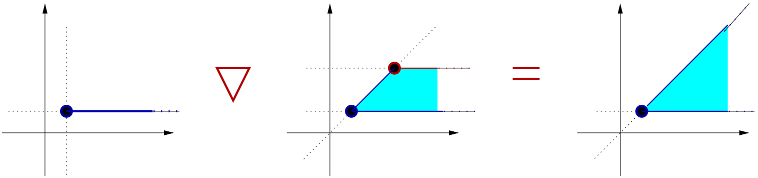
$\{X \geq 1, Y \geq 1, Y \leq 1\} \triangledown \{X \geq 1, Y \geq 1, Y \leq 2, X \geq Y\} = \{X \geq 1, Y \geq 1\}$

Better widenings on polyhedra

Taking into account constraints from $Y^\sharp$

$$X^\sharp \triangledown Y^\sharp \stackrel{\text{def}}{=} \{\, c \in X^\sharp \mid Y^\sharp \subseteq^\sharp \{c\} \,\} \cup \{\, c \in Y^\sharp \mid \exists c' \in X^\sharp \colon X^\sharp =^\sharp (X^\sharp \setminus c') \cup \{c\} \,\}$$

also keeps the constraints from $Y^\sharp$ that are equivalent to a constraint from $X^\sharp$



$\{X \geq 1, Y \geq 1, Y \leq 1\} \triangledown \{X \geq 1, Y \geq 1, Y \leq 2, X \geq Y\} = \{X \geq 1, X \geq Y\}$

Widening with thresholds

parameterized by a finite set of constraints T

$$X^\sharp \triangledown Y^\sharp \stackrel{\text{def}}{=} \{\, c \in X^\sharp \mid Y^\sharp \subseteq^\sharp \{c\} \,\} \cup \{\, c \in T \mid X^\sharp \subseteq^\sharp \{c\} \wedge Y^\sharp \subseteq^\sharp \{c\} \,\}$$

adds constraints from T when stable, similar to the widening on intervals...

Example analysis with polyhedra

Example

$X \leftarrow 2$; $I \leftarrow 0$;
while $I < 10$ **do**
  if rand$(0,1) = 0$ **then** $X \leftarrow X + 2$ **else** $X \leftarrow X - 3$;
  $I \leftarrow I + 1$
done

Loop invariant :

increasing iteration with widening

$$\begin{array}{rcl}
X_1^\sharp & = & \{X = 2, I = 0\} \\
X_2^\sharp & = & \{X = 2, I = 0\} \triangledown (\{X = 2, I = 0\} \cup^\sharp \{X \in [-1,4],\ I = 1\}) \\
& = & \{X = 2, I = 0\} \triangledown \{\, I \in [0,1],\ 2 - 3I \leq X \leq 2I + 2\,\} \\
& = & \{I \geq 0,\ 2 - 3I \leq X \leq 2I + 2\}
\end{array}$$

decreasing iteration: to get $I \leq 10$

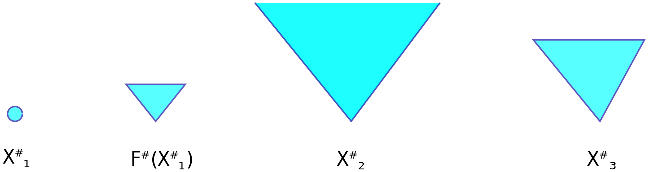
$$\begin{array}{rcl}
X_3^\sharp & = & \{X = 2, I = 0\} \cup^\sharp \{\, I \in [1,10],\ 2 - 3I \leq X \leq 2I + 2\,\} \\
& = & \{I \in [0,10],\ 2 - 3I \leq X \leq 2I + 2\}
\end{array}$$

at the end of the loop, we get: $I = 10 \wedge X \in [-28, 22]$

Example analysis with polyhedra (illustration)

Example

$X \leftarrow 2;\ I \leftarrow 0;$
while $I < 10$ **do**
  if $\mathbf{rand}(0,1) = 0$ **then** $X \leftarrow X + 2$ **else** $X \leftarrow X - 3;$
  $I \leftarrow I + 1$
done



$$
\begin{array}{lcl}
X_1^\sharp & = & \{X = 2, I = 0\} \\
X_2^\sharp & = & \{X = 2, I = 0\} \triangledown (\{X = 2, I = 0\} \cup^\sharp \{X \in [-1,4],\ I = 1\}) \\
 & = & \{I \geq 0,\ 2 - 3I \leq X \leq 2I + 2\} \\
X_3^\sharp & = & \{X = 2, I = 0\} \cup^\sharp \{\, I \in [1,10],\ 2 - 3I \leq X \leq 2I + 2\,\} \\
 & = & \{I \in [0,10],\ 2 - 3I \leq X \leq 2I + 2\}
\end{array}
$$

Summary of numeric abstract domains

Cost vs. precision:

Domain	Invariants	Memory cost	Time cost (per op.)
intervals	$V \in [\ell, h]$	$\mathcal{O}(\lvert\mathbb{V}\rvert)$	$\mathcal{O}(\lvert\mathbb{V}\rvert)$
affine equalities	$\sum_i \alpha_i V_i = \beta_i$	$\mathcal{O}(\lvert\mathbb{V}\rvert^2)$	$\mathcal{O}(\lvert\mathbb{V}\rvert^3)$
polyhedra	$\sum_i \alpha_i V_i \geq \beta_i$	unbounded, exponential in practice	

- domains provide a tradeoff between precision and cost
- relational invariants are sometimes necessary
  even to prove non-relational properties
- an abstract domain is defined by
  - a choice of abstract properties and operators (semantic aspect)
  - data-structures and algorithms (algorithmic aspect)
- an abstract domain mixes two kinds of approximations:
  - static approximations (choice of abstract properties)
  - dynamic approximations (widening)

Extensions

Weakly relational domains

Principle: restrict the expressiveness of polyhedra
to be more efficient at the cost of precision

Example domains:

- Based on constraint propagation: (closure algorithms)
  - Octagons: $\pm X \pm Y \leq c$
    shortest path closure: $x + y \leq c \wedge -y + z \leq d \implies x + z \leq c + d$
    quadratic memory cost, cubic time cost
  - Two-variables per inequality: $\alpha x + \beta y \leq c$
    slightly more complex closure algorithm, by Nelson
  - Octahedra: $\sum \alpha_i V_i \leq c$, $\alpha_i \in \{ -1, 0, 1 \}$
    incomplete propagation, to avoid exponential cost
  - Pentagons: $X - Y \leq 0$
    restriction of octagons
    incomplete propagation, aims at linear cost
- Based on linear programming:
  - Template polyhedra: $\mathbf{M} \times \vec{V} \geq \vec{C}$ for a fixed $\mathbf{M}$

Integers

Issue:

in relational domains we used implicitly real-valued environments $\mathbb{V} \to \mathbb{R}$
our concrete semantics is based on integer-valued environments $\mathbb{V} \to \mathbb{Z}$

In fact, an abstract element $X^\sharp$ does not represent $\gamma(X^\sharp) \subseteq \mathbb{R}^{|\mathbb{V}|}$, but:

$$\gamma_{\mathbb{Z}}(X^\sharp) \stackrel{\text{def}}{=} \gamma(X^\sharp) \cap \mathbb{Z}^{|\mathbb{V}|}$$ (keep only integer points)

Soundness and exactness for $\gamma_{\mathbb{Z}}$

- $\subseteq^\sharp$ and $=^\sharp$ are is no longer exact
  e.g., $\gamma(2X = 1) \neq \gamma(\perp)$, but $\gamma_{\mathbb{Z}}(2X = 1) = \gamma(\perp) = \emptyset$
- $\cap^\sharp$ and affine tests are still exact
- affine and non-deterministic assignments are no longer exact
  e.g., $R^\sharp = (Y = 2X)$, $S^\sharp[\![X \leftarrow [-\infty, +\infty]]\!] R^\sharp = \top$,
  but $S[\![X \leftarrow [-\infty, +\infty]]\!] (\gamma_{\mathbb{Z}}(R^\sharp)) = \mathbb{Z} \times (2\mathbb{Z})$
- all the operators are **still sound**
  $\mathbb{Z}^{|\mathbb{V}|} \subseteq \mathbb{R}^{|\mathbb{V}|}$, so $\forall X^\sharp: \gamma_{\mathbb{Z}}(X^\sharp) \subseteq \gamma(X^\sharp)$

(in general, soundness, exactness, optimality depend on the definition of γ)

Integers (cont.)

Possible solutions:

- enrich the domain (add exact representations for operation results)
  - congruence equalities: $\wedge_i \sum_j \alpha_{ij} V_j \equiv \beta_i \; [\gamma_i]$ (Granger 1991)
  - Pressburger arithmetic (first order logic with 0, 1, $+$)
    decidable, but with very costly algorithms
- design optimal (non-exact) operators

  also based on costly algorithms, e.g.:
  - normalization: integer hull
    smallest polyhedra containing $\gamma_Z(X^\sharp)$
  - emptiness testing: integer programming
    NP-hard, while linear programming is P
- pragmatic solution (efficient, non-optimal)

  use regular operators for $\mathbb{R}^{|\mathbb{V}|}$, then tighten each constraint to remove as many non-integer points as possible

  e.g.: $2X + 6Y \geq 3 \rightarrow X + 3Y \geq 2$

Note: we abstract integers as reals!

Non-linear expressions

Issue:

Our relational domains can only deal with linear expressions
How can we abstract non-linear assignments such as $X \leftarrow Y \times Z$?

Idea: replace $Y \times Z$ with a sound linear approximation

Framework:

We define an approximation preorder $\preceq$ on expressions:

$$R \models e_1 \preceq e_2 \stackrel{\text{def}}{\iff} \forall \rho \in R,\ \mathsf{E}[\![\, e_1 \,]\!]\, \rho \subseteq \mathsf{E}[\![\, e_2 \,]\!]\, \rho$$

Soundness property:

if $\gamma(X^\sharp) \models e \preceq e'$ then:

- $\mathsf{S}[\![\, V \leftarrow e \,]\!]\, \gamma(X^\sharp) \subseteq \gamma(\mathsf{S}[\![\, V \leftarrow e' \,]\!]\, X^\sharp)$
- $\mathsf{S}[\![\, e \bowtie 0? \,]\!]\, \gamma(X^\sharp) \subseteq \gamma(\mathsf{S}^\sharp[\![\, e' \bowtie 0? \,]\!]\, X^\sharp)$

(we can now use e' in the abstract instead of e!)

Linearization

In practice, we put expressions into affine interval form:

$$expr_\ell : [a_0, b_0] + \sum_k [a_k, b_k] V_k$$

Benefits:

- affine expressions are easy to manipulate
- interval coefficients allow non-determinism in expressions, hence, the opportunity for abstraction
- we can easily construct generalized abstract operators to handle affine interval expressions in our domains
  possibly by first abstracting further expressions into $[a_0, b_0] + \sum_k c_k V_k$ using the bounds on each V_k

Linearization (cont.)

Operations on affine interval forms

- adding $\boxplus$ and subtracting $\boxminus$ two forms
- multiplying $\boxtimes$ and dividing $\boxslash$ a form by an interval

Noting i_k the interval $[a_k, b_k]$ and using interval operations $+^\sharp$, $-^\sharp$, $\times^\sharp$, $/^\sharp$ (e.g., $[a,b] +^\sharp [c,d] = [a+c, b+d]$):

- $(i_0 + \sum_k i_k \times V_k) \boxplus (i'_0 + \sum_k i'_k \times V_k) \stackrel{\text{def}}{=} (i_0 +^\sharp i'_0) + \sum_k (i_k +^\sharp i'_k) \times V_k$
- $i \boxtimes (i_0 + \sum_k i_k \times V_k) \stackrel{\text{def}}{=} (i \times^\sharp i_0) + \sum_k (i \times^\sharp i_k) \times V_k$
- ...

Projection $\pi_k : \mathcal{E}^\sharp \to expr_\ell$

We suppose we are given an abstract interval projection operator π_k such that:

$$\pi_k(X^\sharp) = [a,b] \text{ where } [a,b] \supseteq \{\, \rho(V_k) \mid \rho \in \gamma(X^\sharp) \,\}$$

Linearization (cont.)

Intervalization $\quad \iota : (expr_\ell \times \mathcal{E}^\sharp) \to expr_\ell$

Flattens the expression into a single interval:

$$\iota(i_0 + \sum_k (i_k \times V_k),\ X^\sharp) \stackrel{\text{def}}{=} i_0 \ +^\sharp\ \sum\nolimits^\sharp_{b,\,k} \ (i_k \times^\sharp \pi_k(X^\sharp)).$$

Linearization $\quad \ell : (expr \times \mathcal{E}^\sharp) \to expr_\ell$

Defined by induction on the syntax of expressions:

- $\ell(V, X^\sharp) \stackrel{\text{def}}{=} [1,1] \times V$
- $\ell(\textbf{rand}(a,b), X^\sharp) \stackrel{\text{def}}{=} [a,b]$
- $\ell(e_1 + e_2, X^\sharp) \stackrel{\text{def}}{=} \ell(e_1, X^\sharp) \boxplus \ell(e_2, X^\sharp)$
- $\ell(e_1 - e_2, X^\sharp) \stackrel{\text{def}}{=} \ell(e_1, X^\sharp) \boxminus \ell(e_2, X^\sharp)$
- $\ell(e_1 / e_2, X^\sharp) \stackrel{\text{def}}{=} \ell(e_1, X^\sharp) \boxslash \iota(\ell(e_2, X^\sharp), X^\sharp)$
- $\ell(e_1 \times e_2, X^\sharp) \stackrel{\text{def}}{=}$ can be $\begin{cases} \text{either} & \iota(\ell(e_1, X^\sharp), X^\sharp) \boxtimes \ell(e_2, X^\sharp) \\ \text{or} & \iota(\ell(e_2, X^\sharp), X^\sharp) \boxtimes \ell(e_1, X^\sharp) \end{cases}$

Linearization application

Property soundness of the linearization:

For any abstract domain $\mathcal{E}^\sharp$, any $X^\sharp \in \mathcal{E}^\sharp$ and $e \in \mathit{expr}$, we have:

$$\gamma(X^\sharp) \models e \preceq \ell(e, X^\sharp)$$

Remarks:

ℓ results in a loss of precision
ℓ is not monotonic for $\preceq$
(e.g., $\ell(V/V, V \mapsto [1, +\infty]) = [0, 1] \times V \not\preceq 1$)

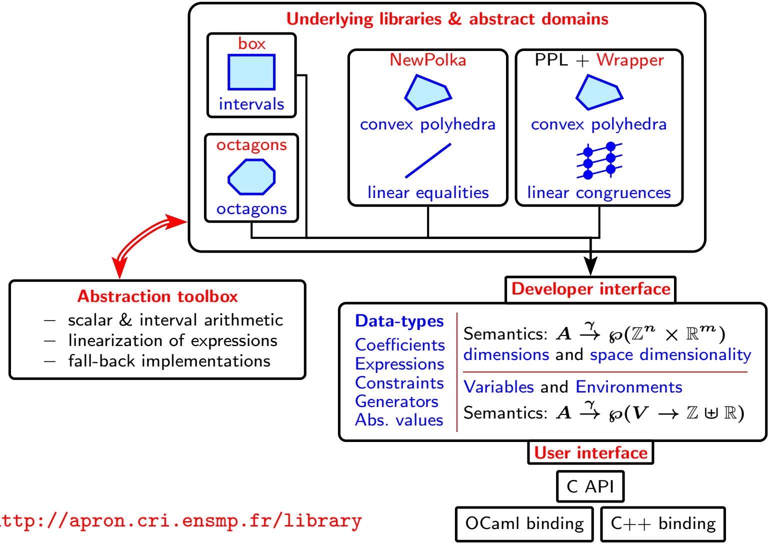
Example: analysis with polyhedra

$$\boxed{\begin{array}{l} Y \leftarrow \textbf{rand}(0, 1000); \\ T \leftarrow \textbf{rand}(-1, 1); \\ X \leftarrow T \times Y \end{array}}$$

- $T \times Y$ is linearized as $[-1, 1] \times Y$
- we can prove that $X \leq Y$

Using the Apron Library

Apron library



http://apron.cri.ensmp.fr/library

`opam install apron`

Apron modules

The `Apron` module contains sub-modules:

- `Abstract1`
  abstract elements
- `Manager`
  abstract domains (arguments to all `Abstract1` operations)
- `Polka`
  creates a manager for polyhedra abstract elements
- `Var`
  integer or real program variables (denoted as a string)
- `Environment`
  sets of integer and real program variables
- `Texpr1`
  arithmetic expression trees
- `Tcons1`
  arithmetic constraints (based on `Texpr1`)
- `Coeff`
  numeric coefficients (appear in `Texpr1`, `Tcons1`)

Variables and environments

Variables: `type Var.t`

variables are denoted by their name, as a string:

(assumes implicitly that no two program variables have the same name)

- `Var.of_string: string -> Var.t`

Environments: `type Environment.t`

an abstract element abstracts a set of mappings in $\mathbb{V} \to \mathbb{R}$

$\mathbb{V}$ is the environment; it contains integer-valued and real-valued variables

- `Environment.make: Var.t array -> Var.t array -> t`
  `make ivars rvars` creates an environment with `ivars` integer variables and `rvars` real variables;
  `make [||] [||]` is the empty environment
- `Environment.add: Environment.t -> Var.t array -> Var.t array -> t`
  `add env ivars rvars` adds some integer or real variables to env
- `Environment.remove: t -> Var.t array -> t`

internally, an abstract element abstracts a set of points in $\mathbb{R}^n$;
the environment maintains the mapping from variable names to dimensions in $[1, n]$

Expressions

Concrete expression trees: `type Texpr1.expr`

```
type expr =  | Cst of Coeff.t                                 (constants)
             | Var of Var.t                                   (variables)
             | Unop of unop * expr * typ * round              (unary op.)
             | Binop of binop * expr * expr * typ * round     (binary op.)
```

- unary operators

  `type Texpr1.unop = Neg | ···`

- binary operators

  `type Texpr1.binop = Add | Sub | Mul | Div | ···`

- numeric type:
  (we only use integers, but reals and floats are also possible)

  `type Texpr1.typ = Int | ···`

- rounding direction:
  (only useful for the division on integers; we use rounding to zero, i.e., truncation)

  `type Texpr1.round = Zero | ···`

Expressions (cont.)

Internal expression form: `type Texpr1.t`

concrete expression trees must be converted to an internal form to be used in abstract operations

- `Texpr1.of_expr: Environment.t -> Texpr1.expr -> Texpr1.t`
  (the environment is used to convert variable names to dimensions in $\mathbb{R}^n$)

Coefficients: `type Coeff.t`

can be either a scalar $\{c\}$ or an interval $[a, b]$

we can use the `Mpqf` module to convert from strings to arbitrary precision integers, before converting them into `Coeff.t`:

- for scalars $\{c\}$:
  ```
  Coeff.s_of_mpqf (Mpqf.of_string c)
  ```
- for intervals $[a, b]$:
  ```
  Coeff.i_of_mpqf (Mpqf.of_string a) (Mpqf.of_string b)
  ```

Constraints

Constraints: type `Tcons1.t`

constructor *expr* $\bowtie$ 0:

- `Tcons1.make: Texpr1.t -> TCons1.typ -> Tcons1.t`

  where:

  `type Tcons1.typ =` `SUPEQ` ($\geq$) | `SUP` ($>$) | `EQ` ($=$) | `DISEQ` ($\neq$) | $\cdots$

Note: avoid using `DISEQ` directly, which is not very precise;
but use a disjunction of two `SUP` constraints instead

Constraint arrays: type `Tcons1.earray`

abstract operators do not use constraints, but constraint arrays instead

Example: constructing an array `ar` containing a single constraint:

```
let c = Tcons1.make texpr1 typ in
let ar = Tcons1.array_make env 1 in
Tcons1.array_set ar 0 c
```

Abstract operators

Abstract elements: type `Abstract1.t`

- `Abstract1.top: Manager.t -> Environment.t -> t`
  create an abstract element where variables have any value
- `Abstract1.env: t -> Environment.t`
  recover the environment on which the abstract element is defined
- `Abstract1.change_environment: Manager.t -> t -> Environment.t -> bool -> t`
  set the new environment, adding or removing variables if necessary
  the `bool` argument should be set to `false`: variables are not initialized
- `Abstract1.assign_texpr: Manager.t -> t -> Var.t -> Texpr1.t -> t option -> t`
  abstract assignment; the `option` argument should be set to `None`
- `Abstract1.forget_array: Manager.t -> t -> Var.t array -> bool -> t`
  non-deterministic assignment: forget the value of variables (when `bool` is `false`)
- `Abstract1.meet_tcons_array: Manager.t -> t -> Tcons1.earray -> t`
  abstract test: add one or several constraint(s)

Abstract operators (cont.)

- `Abstract1.join`: `Manager.t -> t -> t -> t`
  abstract union $\cup^\sharp$
- `Abstract1.meet`: `Manager.t -> t -> t -> t`
  abstract intersection $\cap^\sharp$
- `Abstract1.widen`: `Manager.t -> t -> t -> t`
  widening $\triangledown$
- `Abstract1.is_leq`: `Manager.t -> t -> t -> bool`
  $\subseteq^\sharp$: return `true` if the first argument is included in the second
- `Abstract1.is_bottom`: `Manager.t -> t -> t bool`
  whether the abstract element represents $\emptyset$
- `Abstract1.print`: `Format.formatter -> t -> unit`
  print the abstract element

Contract:

- operators return a new, immutable abstract element (functional style)
- operators return over-approximations
  (not always optimal; e.g.: for non-linear expressions)
- predicates return `true` (definitely true) or `false` (don't know)

Managers

Managers: `type Manager.t`

The manager denotes a choice of abstract domain
To use the polyhedra domain, construct the manager with:

- `let manager = Polka.manager_alloc_loose ()`

the same `manager` variable is passed to all `Abstract1` function

to choose another domain, you only need to change the line defining `manager`

Other libraries:

- `Polka.manager_alloc_equalities` (affine equalities)
- `Polka.manager_alloc_strict` ($\geq$ and $>$ affine inequalities over $\mathbb{R}$)
- `Box.manager_alloc` (intervals)
- `Oct.manager_alloc` (octagons)
- `Ppl.manager_alloc_grid` (affine congruences)
- `PolkaGrid.manager_alloc` (affine inequalities and congruences)

Errors

Argument compatibility: ensure that:

- the same manager is used when creating and using an abstract element

  the type system checks for the compatibility between `'a Manager.t` and `'a Abstract1.t`

- expressions and abstract elements have the same environment
- assigned variables exist in the environment of the abstract element
- both abstract elements of binary operators ($\cup$, $\cap$, $\triangledown$, $\subseteq$) are defined on the same environment

Failure to ensure this results in a `Manager.Error` exception

Abstract domain skeleton using Apron

```
open Apron

module RelationalDomain = (struct
  (* manager *)
  type man = Polka.loose Polka.t
  let manager = Polka.manager_alloc_loose ()

  (* abstract elements *)
  type t = man Abstract1.t

  (* utilities *)
  val expr_to_texpr:  expr -> Texpr1.expr

  (* implementation *)
  ...

end:  ENVIRONMENT_DOMAIN)
```

To compile: add to the `Makefile`:

```
OCAMLINC = ··· -I +zarith -I +apron -I +gmp
CMA = bigarray.cma gmp.cma apron.cma polkaMPQ.cma
```

Fall-back assignments and tests

```
let rec expr_to_texpr = function
| AST_binary (op, e1, e2) ->
  match op with
    | AST_PLUS -> Texpr1.Binop ···
    | ···
    | _ -> raise Top

let assign env var expr =
  try
    let e = expr_to_texpr expr in
    Abstract1.assign_texpr ···
  with Top -> Abstract1.forget_array ···

let compare abs e1 e2 =
  try
    ···
    Abstract1.meet_tcons_array ···
  with Top -> abs
```

Idea:

raise Top to abort a computation
catch it to fall-back to sound coarse assignments and tests