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
**Motivation**

**Relational assignments and tests**

**Example**

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

**Interval analysis:**

- $S^\# [ X \geq Y? ]$ is abstracted as the identity
  - given $R^\# \overset{\text{def}}{=} [X \mapsto [0, 10], Y \mapsto [0, 10]]$
  - $S^\# [ \text{if } X \geq Y \text{ then } \cdots ] R^\# = R^\#$
- $D \leftarrow Y - X$ gives $D \in [0, 10] -^\# [0, 10] = [-10, 10]$
- the assertion $D \geq 0$ fails
Relational assignments and tests

Example

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

Solution: relational domain

- represent explicitly the information \( X \leq Y \)
- infer that \( X \leq Y \) holds after the if \cdots\text{ then} \cdots\text{ 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

\[ I \leftarrow 1; \ X \leftarrow 0; \]
\[ \textbf{while} \ I \leq 1000 \ \textbf{do} \]
\[ I \leftarrow I + 1; \ X \leftarrow X + 1; \]
\[ \textbf{assert} \ X \leq 1000 \]

Interval analysis:

- after iterations with \textit{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 \( X \in [0, +\infty) \) \( \implies \) \textit{assert} fails

- using a \textit{decreasing} iteration after widening, we get:
  - as loop invariant: \( I \in [1, 1001] \) and \( X \in [0, +\infty) \)
  - after the loop: \( I = 1001 \) and \( X \in [0, +\infty) \) \( \implies \) \textit{assert} fails

- without widening, we get \( I = 1001 \) and \( X = 1000 \) \( \implies \) \textit{assert} passes

but we need 1000 iterations!  \( (\simeq \text{concrete fixpoint computation}) \)
Relational loop invariants

Example

\[
\begin{align*}
I & \leftarrow 1; \quad X \leftarrow 0; \\
\textbf{while} \ I \leq 1000 \ \textbf{do} \\
& I \leftarrow I + 1; \quad X \leftarrow X + 1; \\
\textbf{assert} \ X \leq 1000
\end{align*}
\]

Solution: relational domain

- infer a relational loop invariant: \( I = X + 1 \land 1 \leq I \leq 1001 \)
  \( I = X + 1 \) 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:
  \( I = 1001 \)
  \( X = I - 1 = 1000 \implies \textbf{assert} \) 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: \( Z = \max(X, Y, 0) \)

\[
Z \leftarrow X; \\
\text{if } Y > Z \text{ then } Z \leftarrow Y; \\
\text{if } Z < 0 \text{ then } Z \leftarrow 0
\]
Relational procedure analysis

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

\[
\begin{align*}
X' &\leftarrow X; \quad Y' \leftarrow Y; \quad Z' \leftarrow Z; \\
Z' &\leftarrow X'; \\
\text{if } Y' > Z' \text{ then } Z' &\leftarrow Y'; \\
\text{if } Z' < 0 \text{ then } Z' &\leftarrow 0
\end{align*}
\]

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

Example: \( Z = \max(X, Y, 0) \)

\[
\begin{align*}
X' &\leftarrow X; Y' \leftarrow Y; Z' \leftarrow Z; \\
Z' &\leftarrow X'; \\
\text{if } Y' > Z' &\text{ then } Z' \leftarrow Y'; \\
\text{if } Z' < 0 &\text{ then } Z' \leftarrow 0 \\
// & Z' \geq X \land Z' \geq Y \land Z' \geq 0 \land X' = X \land Y' = Y
\end{align*}
\]

- 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:

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

where all the $\alpha_{ij}$ and $\beta_j$ are inferred automatically.

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

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

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:**

\[ \mathcal{E}^\# \overset{\text{def}}{=} \bigcup_m \{ \langle \mathbf{M}, \mathbf{C} \rangle \mid \mathbf{M} \in \mathbb{Q}^{m \times n}, \mathbf{C} \in \mathbb{Q}^m \} \cup \{\bot\} \]

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

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

\[ \gamma(\langle \mathbf{M}, \mathbf{C} \rangle) \overset{\text{def}}{=} \{ \mathbf{V} \in \mathbb{R}^n \mid \mathbf{M} \times \mathbf{V} = \mathbf{C} \} \]

**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

\textbf{Galois connection:} (actually, a Galois insertion)

between arbitrary subsets and affine subsets

\[(\mathcal{P}(\mathbb{R}^{\mid V\mid}), \subseteq) \underbrace{\leftrightarrow}_{\gamma} (\text{Aff}(\mathbb{R}^{\mid V\mid}), \subseteq) \overbrace{\alpha}^{\gamma}\]

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

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

\[\alpha(X) = \cap \{ Y \in \text{Aff}(\mathbb{R}^{\mid V\mid}) \mid X \subseteq Y \}\]

\text{Aff}(\mathbb{R}^{\mid V\mid}) contains every point in \(\mathbb{R}^{\mid V\mid}\)
we can also construct \(\alpha(X)\) by (abstract) union:

\[\alpha(X) = \cup^\# \{ \{x\} \mid x \in X \}\]

Notes:
- we have assimilated \(V \rightarrow \mathbb{R}\) to \(\mathbb{R}^{\mid V\mid}\)
- we have used \(\text{Aff}(\mathbb{R}^{\mid V\mid})\) instead of the matrix representation \(E^\#\) for simplicity; a Galois connection also exists between \(\mathcal{P}(\mathbb{R}^{\mid V\mid})\) and \(E^\#\)
Normalisation and emptiness testing

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

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

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

**Example:**

\[
\begin{align*}
2X + Y + Z &= 19 \\
2X + Y - Z &= 9 \\
3Z &= 15 \\
\downarrow \\
\begin{cases}
X + 0.5Y &= 7 \\
Z &= 5
\end{cases}
\end{align*}
\]
Affine equalities operators

Abstract operators:

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

$$X^\# \cap^\# Y^\# \overset{\text{def}}{=} \text{Gauss} \left( \langle \left[ \begin{array}{c} M_{X^\#} \\ M_{Y^\#} \end{array} \right], \left[ \begin{array}{c} \vec{C}_{X^\#} \\ \vec{C}_{Y^\#} \end{array} \right] \rangle \right)$$  

(join equations)

$$X^\# =^\# Y^\# \overset{\text{def}}{\iff} M_{X^\#} = M_{Y^\#} \quad \text{and} \quad \vec{C}_{X^\#} = \vec{C}_{Y^\#}$$  

(uniqueness)

$$X^\# \subseteq^\# Y^\# \overset{\text{def}}{\iff} X^\# \cap^\# Y^\# =^\# X^\#$$

$$S^\#[ \sum_j \alpha_j V_j = \beta? ] X^\# \overset{\text{def}}{=} \text{Gauss} \left( \langle \left[ \begin{array}{c} M_{X^\#} \\ \alpha_1 \cdots \alpha_n \end{array} \right], \left[ \begin{array}{c} \vec{C}_{X^\#} \\ \beta \end{array} \right] \rangle \right)$$  

(add equation)

$$S^\#[ e \ncongest e'? ] X^\# \overset{\text{def}}{=} X^\# \quad \text{for other tests}$$

Remark:

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

$$(X^\# \subseteq^\# Y^\# \iff \gamma(X^\#) \subseteq \gamma(Y^\#), \quad \gamma(X^\# \cap^\# Y^\#) = \gamma(X^\#) \cap \gamma(Y^\#), \ldots)$$
Affine equality assignment

**Non-deterministic assignment:** \( S^#[ V_j \leftarrow [-\infty, +\infty] ] \)

**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 \( M \)

- Pick the row \( \langle \tilde{M}_i, C_i \rangle \) 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 \( \implies \) \( M \) stays in row echelon form
- Remove the row \( \langle \tilde{M}_i, C_i \rangle \)

**Example:** forgetting \( Z \)

\[
\begin{align*}
X + Z &= 10 \\
Y + Z &= 7
\end{align*}
\]\( \implies \) \[
\begin{align*}
X - Y &= 3
\end{align*}
\]

The operator is **exact**
Affine equality assignment

**Affine assignments:**

\[ S^\#[ V_j \leftarrow \sum_i \alpha_i V_i + \beta ] \]

\[ S^\#[ V_j \leftarrow \sum_i \alpha_i V_i + \beta ] X^\# \overset{\text{def}}{=} \]

if \( \alpha_j = 0 \), \((S^\#[ V_j = \sum_i \alpha_i V_i + \beta ] \circ S^\#[ V_j \leftarrow [\infty, +\infty] ]) X^\# \)

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:** \( \alpha_j \neq 0 \)

\( S[ V_j \leftarrow e ] \subset S[ V_j \leftarrow e ] \circ S[ V_j \leftarrow [\infty, +\infty] ] \) as \( e \) depends on \( V \)

\( \rho \in S[ V_j \leftarrow e ] R \iff \exists \rho' \in R: \rho = \rho'[V_j \mapsto \sum_i \alpha_i \rho'(V_i) + \beta] \)

\( \iff \exists \rho' \in R: \rho[V_j \mapsto (\rho(V_j) - \sum_{i \neq j} \alpha_i \rho'(V_i) - \beta)/\alpha_j] = \rho' \)

\( \iff \rho[V_j \mapsto (\rho(V_j) - \sum_{i \neq j} \alpha_i \rho(V_i) - \beta)/\alpha_j] \in R \)

**Non-affine assignments:** revert to non-deterministic case

\( S^\#[ V_j \leftarrow e ] X^\# \overset{\text{def}}{=} S^\#[ V_j \leftarrow [\infty, +\infty] ] X^\# \) (imprecise but sound)
Affine equality join

**Join:**  \( \langle M, \tilde{C} \rangle \cup^\# \langle N, \tilde{D} \rangle \)

**Idea:** unify columns 1 to \( n \) of \( \langle M, \tilde{C} \rangle \) and \( \langle N, \tilde{D} \rangle \) using row operations

**Example:**

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

\[
\begin{pmatrix} R \ 0 \ M_1 \\ 0 \ 1 \ M_2 \\ 0 \ 0 \ M_3 \end{pmatrix}, \begin{pmatrix} R \ \vec{\beta} \ N_1 \\ 0 \ 0 \ N_2 \\ 0 \ 0 \ N_3 \end{pmatrix} \implies \begin{pmatrix} R \ \vec{\beta} \ M'_1 \\ 0 \ 0 \ 0 \\ 0 \ 0 \ M_3 \end{pmatrix}, \begin{pmatrix} R \ \vec{\beta} \ N_1 \\ 0 \ 0 \ N_2 \\ 0 \ 0 \ N_3 \end{pmatrix}
\]

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

\( \implies \) 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

\[
X \leftarrow 10; \ Y \leftarrow 100; \\
\textbf{while } X \neq 0 \textbf{ do} \\
\quad X \leftarrow X - 1; \\
\quad Y \leftarrow Y + 10
\]

Abstract loop iterations:

\[
\lim \lambda X^\#.I^\# \cup^\# S^\#[ \text{body}] (S^\#[ X \neq 0?] X^#)
\]

- loop entry: \( I^\# = (X = 10 \land Y = 100) \)
- after one loop body iteration: \( F^\#(I^#) = (X = 9 \land Y = 110) \)
- \( \implies X^# \overset{\text{def}}{=} I^# \cup^# F^\#(I^#) = (10X + Y = 200) \)
- \( X^# \) is stable

at loop exit, we get \( S^\#[ X = 0?] (10X + Y = 200) = (X = 0 \land Y = 200) \)
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)

\[ \mathcal{E}^\# \simeq \{ \text{closed convex polyhedra of } \mathbb{V} \to \mathbb{R} \} \]

Notes:
- polyhedra need not be bounded \((\neq \text{polytopes})\)
- we keep reasoning in \(\mathbb{R}\), to use affine theory
Polyhedra have dual representations (Weyl–Minkowski Theorem)

**Constraint representation**

\[ \langle M, \vec{C} \rangle \text{ with } M \in \mathbb{Q}^{m \times n} \text{ and } \vec{C} \in \mathbb{Q}^{m} \]

represents: \( \gamma(\langle M, \vec{C} \rangle) \overset{\text{def}}{=} \{ \vec{V} \mid M \times \vec{V} \geq \vec{C} \} \)

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

**Generator representation**

\([P, R]\) where

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

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

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

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

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 \( P \)
- \( P^{**} = P \)

\[
0x + 0y + 1z \leq 1 \iff (0, 0, 1)
\]
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 $\cap\#$, generators for $\cup\#

$\Rightarrow$ 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

- $2n$ constraints
- 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...
### 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) \( y + x \geq 0, y - x \geq 0, y \leq 0, y \geq -5 \) (non minimal)
- (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** $\alpha$
  - no optimal abstraction as polyhedra for some sets of points
  - $\implies$ 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:
\[
\begin{align*}
P_0 &= \{ (0, \ldots, 0) \} \quad \text{(origin)} \\
R_0 &= \{ \vec{x}_i, -\vec{x}_i \mid 1 \leq i \leq n \} \quad \text{(axes)}
\end{align*}
\]

For each constraint \( \vec{M}_k \cdot \vec{V} \geq C_k \in \langle M, \vec{C} \rangle \), update \([P_{k-1}, R_{k-1}]\) to \([P_k, R_k]\).

Start with \( P_k = R_k = \emptyset \),

- for any \( \vec{P} \in P_{k-1} \) s.t. \( \vec{M}_k \cdot \vec{P} \geq C_k \), add \( \vec{P} \) to \( P_k \)
- for any \( \vec{R} \in R_{k-1} \) s.t. \( \vec{M}_k \cdot \vec{R} \geq 0 \), add \( \vec{R} \) to \( R_k \)
- for any \( \vec{P}, \vec{Q} \in P_{k-1} \) s.t. \( \vec{M}_k \cdot \vec{P} > C_k \) and \( \vec{M}_k \cdot \vec{Q} < C_k \), add to \( P_k \):
  \[
  \vec{O} \overset{\text{def}}{=} \frac{C_k - \vec{M}_k \cdot \vec{Q}}{\vec{M}_k \cdot \vec{P} - \vec{M}_k \cdot \vec{Q}} \vec{P} \quad \text{and} \quad \vec{O} \overset{\text{def}}{=} \frac{C_k - \vec{M}_k \cdot \vec{P}}{\vec{M}_k \cdot \vec{P} - \vec{M}_k \cdot \vec{Q}} \vec{Q}
  \]

![Diagram](image)
Chernikova’s algorithm (cont.)

For any $\vec{R}, \vec{S} \in \mathbb{R}_{k-1}$ s.t. $\vec{M}_k \cdot \vec{R} > 0$ and $\vec{M}_k \cdot \vec{S} < 0$, add to $\mathbf{R}_k$:

$$\vec{O} \overset{\text{def}}{=} (\vec{M}_k \cdot \vec{S}) \vec{R} - (\vec{M}_k \cdot \vec{R}) \vec{S}$$

For any $\vec{P} \in \mathbf{P}_{k-1}$, $\vec{R} \in \mathbb{R}_{k-1}$ 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 $\mathbf{P}_k$:

$$\vec{O} \overset{\text{def}}{=} \vec{P} + \frac{C_k - \vec{M}_k \cdot \vec{P}}{\vec{M}_k \cdot \vec{R}} \vec{R}$$
Example:

\[ P_0 = \{(0, 0)\} \quad R_0 = \{(1, 0), (-1, 0), (0, 1), (0, -1)\} \]
Chernikova’s algorithm example

**Example:**

\[ P_0 = \{(0, 0)\} \]
\[ R_0 = \{(1, 0), (-1, 0), (0, 1), (0, -1)\} \]
\[ P_1 = \{(0, 1)\} \]
\[ R_1 = \{(1, 0), (-1, 0), (0, 1)\} \]
**Example:**

\[ P_0 = \{(0,0)\} \]
\[ R_0 = \{(1,0), (-1,0), (0,1), (0,-1)\} \]
\[ Y \geq 1 \]
\[ P_1 = \{(0,1)\} \]
\[ R_1 = \{(1,0), (-1,0), (0,1)\} \]
\[ X + Y \geq 3 \]
\[ P_2 = \{(2,1)\} \]
\[ R_2 = \{(1,0), (-1,1), (0,1)\} \]
Chernikova’s algorithm example

Example:

\[ P_0 = \{(0, 0)\} \]
\[ R_0 = \{(1, 0), (-1, 0), (0, 1), (0, -1)\} \]
\[ Y \geq 1 \]
\[ P_1 = \{(0, 1)\} \]
\[ R_1 = \{(1, 0), (-1, 0), (0, 1)\} \]
\[ X + Y \geq 3 \]
\[ P_2 = \{(2, 1)\} \]
\[ R_2 = \{(1, 0), (-1, 1), (0, 1)\} \]
\[ X - Y \leq 1 \]
\[ P_3 = \{(2, 1), (1, 2)\} \]
\[ R_3 = \{(0, 1), (1, 1)\} \]

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

Set-theoretic operations:

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

$X^\# \subseteq^\# Y^\# \iff \forall \vec{P} \in P_{X^\#}: M_{Y^\#} \times \vec{P} \geq \vec{C}_{Y^\#}$

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

$X^\# =^\# Y^\# \iff X^\# \subseteq^\# Y^\# \text{ et } Y^\# \subseteq^\# X^\#$

both inclusion

$X^\# \cap^\# Y^\# \stackrel{\text{def}}{=} \langle \left[ \begin{array}{c|c} M_{X^\#} & \vec{C}_{X^\#} \\ \hline M_{Y^\#} & \vec{C}_{Y^\#} \end{array} \right] \rangle$

union of constraint sets

$\subseteq^\#, =^\# \text{ and } \cap^\#$ are exact in $\mathcal{P}(\forall \to \mathbb{R})$
**Union:** \( X^\# \cup^\# Y^\# \overset{\text{def}}{=} [ [P_{X^\#} P_{Y^\#}], [R_{X^\#} R_{Y^\#}] ] \)  
union of generator sets

**Examples:**

- two bounded polyhedra
- a point and line

\( \cup^\# \) is optimal in \( \mathcal{P}(\forall \rightarrow \mathbb{R}) \)
\( \alpha \) is not always defined, but \( \alpha(\gamma(X^\#) \cup \gamma(Y^\#)) \) always exists

\( \Rightarrow \) topological closure of the convex hull of \( \gamma(X^\#) \cup \gamma(Y^\#) \)
Affine test:

\[ S^\#\left[ \sum_i \alpha_i V_i \geq \beta ? \right] X^\# \overset{\text{def}}{=} \left\langle \begin{bmatrix} M_{X^\#} \\ \alpha_1 \cdots \alpha_n \end{bmatrix}, \begin{bmatrix} \vec{C}_{X^\#} \\ \beta \end{bmatrix} \right\rangle \]

\[ S^\#\left[ \sum_i \alpha_i V_i = \beta ? \right] X^\# \overset{\text{def}}{=} S^\#\left[ \sum_i \alpha_i V_i \geq -\beta ? \right] (S^\#\left[ \sum_i (-\alpha_i) V_i \geq \beta ? \right] X^\#) \]

- simply adds a constraint to the constraint set
- the operators are exact
- the other tests can be abstracted as
  \[ S^\#\left[ c \right] X^\# \overset{\text{def}}{=} X^\# \]
  sound but very imprecise
Non-deterministic assignment:

\[ S^\# \llbracket V_j \leftarrow \text{rand}(-\infty, +\infty) \rrbracket X^\# \overset{\text{def}}{=} \left[ P_{X^\#}, \left[ R_{X^\#} \vec{x}_j (-\vec{x}_j) \right] \right] \]

- in the concrete:
  \[ S\llbracket V_j \leftarrow \text{rand}(-\infty, +\infty) \rrbracket R = \{ \rho[V_j \mapsto v] \mid \rho \in R, \ v \in \mathbb{R} \} \]

- 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:**

\[
S^\#[ V_j \leftarrow \sum_i \alpha_i V_i + \beta ] \quad \text{def} =
\]

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

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

**Examples:**

\( X \leftarrow X + Y \)

\( X \leftarrow Y \)

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

\[ \mathcal{E}^\# \text{ has strictly increasing infinite chains } \implies \text{ we need a widening} \]

**Definition:**

\[ X^\# \nabla Y^\# \overset{\text{def}}{=} \{ \ c \in X^\# \mid Y^\# \subseteq^\# \{c\} \} \]

- keep the constraints from \( X^\# \) satisfied by \( Y^\# \)
- unlike \( \cup^\# \), no new constraint is created
- \( \nabla \) reduces the set of constraints
  \( \implies \) ensures termination

**Example:**

\[ \{X \geq 1, Y \geq 1, Y \leq 1\} \nabla \{X \geq 1, Y \geq 1, Y \leq 2, X \geq Y\} = \{X \geq 1, Y \geq 1\} \]
Better widenings on polyhedra

Taking into account constraints from \( Y^\# \)

\[
\begin{align*}
X^\# \triangledown Y^\# & \overset{\text{def}}{=} \{ c \in X^\# \mid Y^\# \subseteq^\# \{ c \} \} \\
& \cup \{ c \in Y^\# \mid \exists c' \in X^\#: X^\# =^\# (X^\# \setminus c') \cup \{ c \} \}
\end{align*}
\]

also keeps the constraints from \( Y^\# \) that are equivalent to a constraint from \( X^\# \)

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

Widening with thresholds

parameterized by a finite set of constraints \( T \)

\[
\begin{align*}
X^\# \triangledown Y^\# & \overset{\text{def}}{=} \{ c \in X^\# \mid Y^\# \subseteq^\# \{ c \} \} \\
& \cup \{ c \in T \mid X^\# \subseteq^\# \{ c \} \land Y^\# \subseteq^\# \{ c \} \}
\end{align*}
\]

adds constraints from \( T \) when stable, similar to the widening on intervals
Example analysis with polyhedra

Example

\( X \leftarrow 2; I \leftarrow 0; \)
\[ \text{while } I \leq 10 \text{ do} \]
\[ \quad \text{if rand}(0, 1) = 0 \text{ then } X \leftarrow X + 2 \text{ else } X \leftarrow X - 3; \]
\[ \quad I \leftarrow I + 1 \]
\[ \text{done} \]

Loop invariant:

increasing iteration with widening

\( X_1^\# = \{ X = 2, I = 0 \} \)
\( X_2^\# = \{ X = 2, I = 0 \} \nabla \{ X \in [-1, 4], I = 1 \} \)
\( = \{ X = 2, I = 0 \} \nabla \{ I \in [0, 1], 2 - 3I \leq X \leq 2I + 2 \} \)
\( = \{ I \geq 0, 2 - 3I \leq X \leq 2I + 2 \} \)

decreasing iteration:

to get \( I \leq 10 \)

\( X_3^\# = \{ X = 2, I = 0 \} \cup \{ I \in [1, 10], 2 - 3I \leq X \leq 2I + 2 \} \)
\( = \{ I \in [0, 10], 2 - 3I \leq X \leq 2I + 2 \} \)

at the end of the loop, we get: \( I = 10 \land X \in [-28, 22] \)
Example analysis with polyhedra (illustration)

Example

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

\[
X^\#_1 = \{X = 2, \ I = 0\}
\]
\[
X^\#_2 = \{X = 2, \ I = 0\} \, \nabla \, (\{X = 2, \ I = 0\} \, \cup^\# \, \{X \in [-1, 4], \ I = 1\})
\]
\[= \{I \geq 0, \ 2 - 3I \leq X \leq 2I + 2\}\]
\[
X^\#_3 = \{X = 2, \ I = 0\} \, \cup^\# \, \{I \in [1, 10], \ 2 - 3I \leq X \leq 2I + 2\}
\]
\[= \{I \in [0, 10], \ 2 - 3I \leq X \leq 2I + 2\}\]
Summary of numeric abstract domains

**Cost vs. precision:**

<table>
<thead>
<tr>
<th>Domain</th>
<th>Invariants</th>
<th>Memory cost</th>
<th>Time cost (per op.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>intervals</td>
<td>$V \in [\ell, h]$</td>
<td>$O(</td>
<td>V</td>
</tr>
<tr>
<td>affine equalities</td>
<td>$\sum_i \alpha_i V_i = \beta_i$</td>
<td>$O(</td>
<td>V</td>
</tr>
<tr>
<td>polyhedra</td>
<td>$\sum_i \alpha_i V_i \geq \beta_i$</td>
<td>unbounded, exponential in practice</td>
<td></td>
</tr>
</tbody>
</table>
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: $x + y \leq c \land -y + z \leq d \implies x + z \leq c + d$
    - 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: $M \times \vec{V} \succeq \vec{C}$ for a fixed $M$
Integers

**Issue:**
in relational domains we used implicitly real-valued environments \( V \rightarrow \mathbb{R} \) our concrete semantics is based on integer-valued environments \( V \rightarrow \mathbb{Z} \)

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

\[
\gamma_Z(X^\#) \overset{\text{def}}{=} \gamma(X^\#) \cap \mathbb{Z}^{\mid V\mid} \quad \text{(keep only integer points)}
\]

**Soundness and exactness** for \( \gamma_Z \)

- \( \subseteq^\# \) and \( =^\# \) are is no longer exact
  - e.g., \( \gamma(2X = 1) \neq \gamma(\bot) \), but \( \gamma_Z(2X = 1) = \gamma(\bot) = \emptyset \)

- \( \cap^\# \) and affine tests are still exact

- affine and non-deterministic assignments are no longer exact
  - e.g., \( R^\# = (Y = 2X) \), \( S^\#[X \leftarrow [-\infty, +\infty]] R^\# = T \),
    but \( S[X \leftarrow [-\infty, +\infty]] (\gamma_Z(R^\#)) = \mathbb{Z} \times (2\mathbb{Z}) \)

- all the operators are **still sound**
  - \( \mathbb{Z}^{\mid V\mid} \subseteq \mathbb{R}^{\mid V\mid} \), so \( \forall X^\#: \gamma_Z(X^\#) \subseteq \gamma(X^\#) \)

(in general, soundness, exactness, optimality depend on the definition of \( \gamma \))
Possible solutions:

- **enrich** the domain (add exact representations for operation results)
  - congruence equalities: \( \bigwedge_i \sum_j \alpha_{ij} V_j \equiv \beta_i [\gamma_i] \) (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 \( \gamma_Z(X^\#) \)
    - emptiness testing: integer programming
      - NP-hard, while linear programming is P

- **pragmatic solution** (efficient, non-optimal)
  - use regular operators for \( \mathbb{R}^{|V|} \), 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:

\[
R \models e_1 \preceq e_2 \iff \forall \rho \in R, E[e_1] \rho \subseteq E[e_2] \rho
\]

**Soundness property:**
if \( \gamma(X^\#) \models e \preceq e' \) then:

1. \( S[V \leftarrow e] \gamma(X^\#) \subseteq \gamma(S[V \leftarrow e'] X^\#) \)
2. \( S[e \triangleright 0?] \gamma(X^\#) \subseteq \gamma(S[e' \triangleright 0?] X^\#) \)

(we can now use \( e' \) in the abstract instead of \( e \))
In practice, we put expressions into affine interval form:

$$\text{expr}_\ell : [a_0, b_0] + \sum_k [a_k, b_k] V_k$$

**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 \(+\) and subtracting \(-\) two forms
- multiplying \(\times\) and dividing \(\div\) a form by an interval

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

\[
(i_0 + \sum_k i_k \times V_k) + (i'_0 + \sum_k i'_k \times V_k) \overset{\text{def}}{=} (i_0 + i'_0) + \sum_k (i_k + i'_k) \times V_k
\]

\[
i \times (i_0 + \sum_k i_k \times V_k) \overset{\text{def}}{=} (i \times i_0) + \sum_k (i \times i_k) \times V_k
\]

...  

Projection \(\pi_k : E^\# \rightarrow expr_\ell\)

We suppose we are given an abstract interval projection operator \(\pi_k\) such that:

\(\pi_k(X^\#) = [a, b] \text{ where } [a, b] \supseteq \{ \rho(V_k) | \rho \in \gamma(X^\#) \}\)
Intervalization \( \iota : (\text{expr}_\ell \times \mathcal{E}^\#) \rightarrow \text{expr}_\ell \)

Flattens the expression into a single interval:

\[
\iota(i_0 + \sum_k (i_k \times V_k), X^\#) \overset{\text{def}}{=} i_0 + \# \sum_{b, k} (i_k \times \# \pi_k(X^\#)).
\]

Linearization \( \ell : (\text{expr} \times \mathcal{E}^\#) \rightarrow \text{expr}_\ell \)

Defined by induction on the syntax of expressions:

- \( \ell(V, X^\#) \overset{\text{def}}{=} [1, 1] \times V \)
- \( \ell(\text{rand}(a, b), X^\#) \overset{\text{def}}{=} [a, b] \)
- \( \ell(e_1 + e_2, X^\#) \overset{\text{def}}{=} \ell(e_1, X^\#) \oplus \ell(e_2, X^\#) \)
- \( \ell(e_1 - e_2, X^\#) \overset{\text{def}}{=} \ell(e_1, X^\#) \ominus \ell(e_2, X^\#) \)
- \( \ell(e_1 / e_2, X^\#) \overset{\text{def}}{=} \ell(e_1, X^\#) \oslash \iota(\ell(e_2, X^\#), X^\#) \)
- \( \ell(e_1 \times e_2, X^\#) \overset{\text{def}}{=} \text{can be } \left\{ \begin{array}{ll}
\text{either } & \ell(\ell(e_1, X^\#), X^\#) \boxtimes \ell(e_2, X^\#) \\
\text{or } & \ell(\ell(e_2, X^\#), X^\#) \boxtimes \ell(e_1, X^\#)
\end{array} \right. \)
Linearization application

**Property**  soundness of the linearization:

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

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

Remarks:

- $\ell$ results in a loss of precision
- $\ell$ 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

```
Y ← rand(0, 1000);
T ← rand(-1, 1);
X ← T × Y
```

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

Apron library

Underlying libraries & abstract domains
- box
- intervals
- octagons
- NewPolka: convex polyhedra
- linear equalities
- PPL + Wrapper: convex polyhedra
- linear congruences

Abstraction toolbox
- scalar & interval arithmetic
- linearization of expressions
- fall-back implementations

Data-types
- Coefficients
- Expressions
- Constraints
- Generators
- Abs. values

Semantics:
- $A \rightarrow \wp(Z^n \times R^m)$ for dimensions and space dimensionality
- $A \rightarrow \wp(V \rightarrow Z \cup R)$ for Variables and Environments

Developer interface
User interface
- C API
- OCaml binding
- C++ binding

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 \( V \rightarrow \mathbb{R} \)
\( V \) 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 | ...
**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 \[a, b\]:
Constraints: type Tcons1.t

constructor `expr ⋈ 0`:

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

where:

- `type Tcons1.typ = SUPEQ | SUP | EQ | DISEQ | ...`
- `≥ | > | = | ≠`

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 $\cup$

- **Abstract1.meet**: `Manager.t -> t -> t -> t`  
  abstract intersection $\cap$

- **Abstract1.widen**: `Manager.t -> t -> t -> t`  
  widening $\forall$

- **Abstract1.is_leq**: `Manager.t -> t -> t -> bool`  
  $\subseteq$: 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: 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 (≥ 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 ($\cup$, $\cap$, $\nabla$, $\subseteq$) are defined on the **same environment**

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