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
  
  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
  - polyhedra domain
  - 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

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

Interval analysis:

- \( S^\#[X \geq Y?] \) is abstracted as the identity
  
  \[
  \begin{align*}
  \text{given } R^\# & \overset{\text{def}}{=} [X \mapsto [0, 10], Y \mapsto [0, 10]] \\
  S^\#[\text{if } X \geq Y \text{ then } \cdots ] R^\# & = R^\#
  \end{align*}
  \]

- \( D \leftarrow Y - X \) 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 \( \cdots \) 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

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

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 \(X \in [0, +\infty]\) \(\Rightarrow\) 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 \(X \in [0, +\infty]\) \(\Rightarrow\) assert fails
  (the test \(I \leq 1000\) only refines \(I\), but gives no information on \(X\))

- without widening, we get \(I = 1001\) and \(X = 1000\) \(\Rightarrow\) assert passes
  but we need 1000 iterations! (\(\approx\) concrete fixpoint computation)
Relational loop invariants

Example

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

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

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

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

$X' \leftarrow X; \ Y' \leftarrow Y; \ Z' \leftarrow Z$

$Z' \leftarrow X'$

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

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

// $Z' \geq X \land Z' \geq Y \land Z' \geq 0 \land X' = X \land Y' = Y$

- 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:

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

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

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

\[E^\# \equiv \{ \text{affine subspaces of } \mathbb{V} \to \mathbb{R} \}\]

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:**

\[ E^\# \overset{\text{def}}{=} \bigcup_m \{ hM, \vec{C}_i \mid M \in \mathbb{Q}^{m \times n}, \vec{C} \in \mathbb{Q}^m \} \cup \{ \bot \} \]

- either the constant \( \bot \)
- or a pair \( hM, \vec{C}_i \) where
  - \( M \in \mathbb{Q}^{m \times n} \) 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

\( hM, \vec{C}_i \) represents an equation system, with solutions:

\[ \gamma(hM, \vec{C}_i) \overset{\text{def}}{=} \{ \vec{V} \in \mathbb{R}^n \mid M \times \vec{V} = \vec{C} \} \]

\( M \) should be in row echelon form:

- \( \forall i \leq m: \exists k_i: M_{ik_i} = 1 \) and
  - \( \forall c < k_i: M_{ic} = 0 \), \( \forall i \neq i': M_{ik_i} = 0 \),
- 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: \( m = 0 \)
Affine Equalities

Galois connection

Galois connection:

between arbitrary subsets and affine subsets

\[
(P(\mathbb{R}^{|V|}, \subseteq), \alpha) \rightleftharpoons (\text{Aff}(\mathbb{R}^{|V|}), \subseteq)
\]

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

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

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

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

we can also construct \(\alpha(X)\) by (abstract) union:

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

Notes:

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

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

The Gaussian reduction $\text{Gauss}(\mathbf{hM}, \tilde{\mathbf{C}}\mathbf{1})$ 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 $E^\#$
- by combining rows linearly to remove variable occurrences

**Example:**

\[
\begin{align*}
2X + Y + Z &= 19 \\
2X + Y - Z &= 9 \\
3Z &= 15
\end{align*}
\]

\[
\downarrow
\]

\[
\begin{align*}
X + 0.5Y &= 7 \\
Z &= 5
\end{align*}
\]
Affine equalities

Abstract operators:

If $X^\#$, $Y^\# \not\perp$, we define:

$X^\# \cap^\# Y^\# \stackrel{\text{def}}{=} \text{Gauss} \left( \langle \begin{bmatrix} M_{X^\#} \\ M_{Y^\#} \end{bmatrix}, \begin{bmatrix} \vec{C}_{X^\#} \\ \vec{C}_{Y^\#} \end{bmatrix} \rangle \right)$ (join equations)

$X^\# =^\# Y^\# \iff M_{X^\#} = M_{Y^\#}$ and $\vec{C}_{X^\#} = \vec{C}_{Y^\#}$ (uniqueness)

$X^\# \subseteq^\# Y^\# \iff X^\# \cap^\# Y^\# =^\# X^\#$

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

$S^\#[e \bowtie e?] X^\# \stackrel{\text{def}}{=} X^\#$ for other tests

Remark:

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

$(X^1 \subseteq^1 Y^1 \iff \gamma(X^1) \subseteq \gamma(Y^1), \gamma(X^1 \cap^1 Y^1) = \gamma(X^1) \cap \gamma(Y^1), \ldots)$
Affine equality assignment

**Non-deterministic assignment:** \( S^\# \begin{bmatrix} V_j \rightarrow [-\infty, +\infty] \end{bmatrix} \)

**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 \( \vec{M}_i, C_i \) 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 \( \vec{M}_i, C_i \)

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

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

The operator is **exact**
Affine equality assignment

Affine assignments: \( S^\#[ V_j \leftarrow \sum_i \alpha_i V_i + \beta ] \)

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

\[
\text{if } \alpha_j \neq 0, \overline{hM}, \overline{Ci} \text{ where } V_j \text{ is replaced with } \frac{1}{\alpha_j} (V_j - \sum_{i \neq j} \alpha_i V_i - \beta)
\]

(variable substitution)

Proof sketch: based on properties in the concrete

non-invertible assignment: \( \alpha_j = 0 \)

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

so \( S[ V_j \leftarrow e ] = S[ V_j = e? ] \circ S[ V_j \leftarrow [-\infty, +\infty] ] \)

invertible assignment: \( \alpha_j \neq 0 \)

\[
S[ V_j \leftarrow e ] \subseteq S[ V_j \leftarrow e ] \circ S[ V_j \leftarrow [-\infty, +\infty] ] \text{ as } e \text{ depends on } V
\]

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

Non-affine assignments: revert to non-deterministic case

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

**Join:** $\mathbf{hM}, \vec{C}i \cup^\# \mathbf{hN}, \vec{D}i$

**Idea:** unify columns 1 to $n$ of $\mathbf{hM}, \vec{C}i$ and $\mathbf{hN}, \vec{D}i$
using row operations

**Example:**

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

$\begin{array}{ll}
R \vec{0} M_1 & R \vec{\beta} N_1 \\
\vec{0} \ 1 M_2 & 0 \vec{0} N_2 \\
0 \vec{0} M_3 & 0 \vec{0} N_3
\end{array}$

$\begin{array}{ll}
R \vec{0} M_1 & R \vec{\beta} N_1 \\
\vec{0} \ 0 M'_1 & \vec{0} \ 0 N_1 \\
\vec{0} \ 1 M_2 & \vec{0} \ 0 N_2 \\
0 \vec{0} M_3 & 0 \vec{0} N_3
\end{array}$

Use the row $\begin{pmatrix} \vec{0} \ 1 \vec{M}_2 \end{pmatrix}$ to create $\vec{\beta}$ in the left argument
Then remove the row $\begin{pmatrix} \vec{0} \ 1 \vec{M}_2 \end{pmatrix}$
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

\[ X \leftarrow 10; \ Y \leftarrow 100; \]
\[ \text{while } X \not= 0 \text{ do} \]
\[ X \leftarrow X - 1; \]
\[ Y \leftarrow Y + 10 \]

Abstract loop iterations:

\[
\lim \lambda X^\# : I^\# \cup S^\#[\text{body}] \ (S^\#[\ X \not= 0? \ ] \ X^\#) \\
\]

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

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

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

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

\[ E^\# = \{ \text{closed convex polyhedra of } V \rightarrow \mathbb{R} \} \]

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

**Constraint representation**

\( \mathbf{hM}, \mathbf{C} \) with \( \mathbf{M} \in \mathbb{Q}^{m \times n} \) and \( \mathbf{C} \in \mathbb{Q}^m \)
represents: \( \gamma(\mathbf{hM}, \mathbf{C}) \overset{\text{def}}{=} \{ \mathbf{V} | \mathbf{M} \times \mathbf{V} \geq \mathbf{C} \} \)

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

\( \gamma(\mathbf{P}, \mathbf{R}) \overset{\text{def}}{=} \{ (\sum_{j=1}^{p} \alpha_j \mathbf{P}_j) + (\sum_{j=1}^{r} \beta_j \mathbf{R}_j) | \forall j, \alpha_j, \beta_j \geq 0: \sum_{j=1}^{p} \alpha_j = 1 \} \)
Double description of polyhedra (cont.)

Generator representation examples:

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

- 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 $\cap^1$, generators for $\cup^1$

$\Rightarrow$ 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

- $2n$ 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) $y + x \geq 0, y - x \geq 0, y \leq 0, y \geq -5$
- (b) $y + x \geq 0, y - x \geq 0, y \leq 0$
- (c) $x \leq 0, x \geq 0, y \leq 0, y \geq 0$

(a) 

(b) 

(c) (non minimal)

(minimal)

(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
  $\Rightarrow$ no Galois connection,
  no best abstraction for arbitrary operators

Example:
a disc has infinitely many polyhedral over-approximations
da no approximation is the best one.
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:

- \( P_0 = \{ (0, \ldots, 0) \} \) (origin)
- \( R_0 = \{ \vec{x}_i, -\vec{x}_i \mid 1 \leq i \leq n \} \) (axes)

For each constraint \( \vec{M}_k \cdot \vec{V} \geq C_k \in \langle \vec{M}, \vec{C} \rangle \), 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 \( P_k \):

\[
\vec{O} \overset{\text{def}}{=} \frac{C_k - \vec{M}_k \cdot \vec{Q}}{\vec{M}_k \cdot \vec{P} - \vec{M}_k \cdot \vec{Q}} \vec{P} - \frac{C_k - \vec{M}_k \cdot \vec{P}}{\vec{M}_k \cdot \vec{P} - \vec{M}_k \cdot \vec{Q}} \vec{Q}
\]
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 $\mathbb{R}^k$: $\vec{O} \overset{\text{def}}{=} (\vec{M}_k \cdot \vec{S})\vec{R} - (\vec{M}_k \cdot \vec{R})\vec{S}$

for any $\vec{P} \in \mathbb{P}^{k-1}$, $\vec{R} \in \mathbb{R}^{k-1}$ 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 $\mathbb{P}^k$: $\vec{O} \overset{\text{def}}{=} \vec{P} + \frac{C_k - \vec{M}_k \cdot \vec{P}}{\vec{M}_k \cdot \vec{R}} \vec{R}$
Example:

\( \mathbf{P}_0 = \{(0,0)\} \quad \mathbf{R}_0 = \{(1,0), (-1,0), (0,1), (0,-1)\} \)
Example:

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

\[ P_0 = \{(0, 0)\} \]
\[ R_0 = \{(1, 0), (-1, 0), (0, 1), (0, -1)\} \]
\[ P_1 = \{(0, 1)\} \]
\[ R_1 = \{(1, 0), (-1, 0), (0, 1)\} \]
\[ P_2 = \{(2, 1)\} \]
\[ R_2 = \{(1, 0), (-1, 1), (0, 1)\} \]
Example:

\[ Y \geq 1 \quad P_0 = \{(0, 0)\} \quad R_0 = \{(1, 0), (-1, 0), (0, 1), (0, -1)\} \]

\[ X + Y \geq 3 \quad P_1 = \{(0, 1)\} \quad R_1 = \{(1, 0), (-1, 0), (0, 1)\} \]

\[ X - Y \leq 1 \quad P_2 = \{(2, 1)\} \quad R_2 = \{(1, 0), (-1, 1), (0, 1)\} \]

\[ P_3 = \{(2, 1), (1, 2)\} \quad R_3 = \{(0, 1), (1, 1)\} \]

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

**Set-theoretic operations:**

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

$$X^\# \subseteq^\# Y^\# \iff \left\{ \begin{array}{l}
\forall \vec{P} \in P_{X^1}: M_{Y^1} \times \vec{P} \geq \vec{C}_{Y^1} \\
\forall \vec{R} \in R_{X^1}: M_{Y^1} \times \vec{R} \geq \vec{0}
\end{array} \right.$$

every generator in $X^1$ must satisfy every constraint in $Y^1$

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

$$X^\# \cap^\# Y^\# \overset{\text{def}}{=} \left\langle \begin{bmatrix} M_{X^1} \\ M_{Y^1} \end{bmatrix}, \begin{bmatrix} \vec{C}_{X^1} \\ \vec{C}_{Y^1} \end{bmatrix} \right\rangle$$
union of constraint sets

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

**Examples:**

- two bounded polyhedra
- a point and line

$\cup^\#$ is **optimal** in $\mathcal{P}(\mathbb{V} \rightarrow \mathbb{R})$

$\alpha$ is not always defined, but $\alpha(\gamma(X^1) \cup \gamma(Y^1))$ always exists

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

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

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

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

\[ S^\# \llbracket V_j \leftarrow \text{rand}(-\infty, +\infty) \rrbracket X^\# \overset{\text{def}}{=} \left[ P_{X^1}, \left[ R_{X^1} \bar{x}_j (-\bar{x}_j) \right] \right] \]

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

- in the abstract:
  add two rays parallel to the “forgotten” variable

- exact operator in \( \mathcal{P}(\mathbb{V} \rightarrow \mathbb{R}) \)
Affine assignment:

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

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

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

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: \( S^\#[ V \leftarrow e ] \overset{\text{def}}{=} S^\#[ V \leftarrow [-\infty, +\infty] ] \)
  (sound but not optimal)
Naive widening on polyhedra

$E^\#$ has strictly increasing infinite chains $\implies$ we need a widening

**Definition:**
$$X^\# \bigcup D Y^\# \overset{\text{def}}{=} \{ c \in X^\# \mid Y^\# \subseteq^\# \{c\} \}$$

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

**Example:**
$$\{X \geq 1, Y \geq 1, Y \leq 1\} \bigcup D \{X \geq 1, Y \geq 1, Y \leq 2, X \geq Y\} = \{X \geq 1, Y \geq 1\}$$
Better widenings on polyhedra

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

\[
X^\# \cdot O \ Y^\# \overset{\text{def}}{=} \{ c \in X^\# \mid Y^\# \subseteq \{ c \} \} \\
\cup \{ c \in Y^\# \mid \exists c' \in X^\# : X^\# \equiv^\# (X^\# \setminus c') \cup \{ c \} \}
\]

also keeps the constraints from \( Y^1 \) that are equivalent to a constraint from \( X^1 \)

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

Widening with thresholds

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

\[
X^\# \cdot O \ Y^\# \overset{\text{def}}{=} \{ c \in X^\# \mid Y^\# \subseteq \{ c \} \} \\
\cup \{ c \in T \mid X^\# \subseteq \{ c \} \land Y^\# \subseteq \{ c \} \}
\]

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

Example

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

Loop invariant:

increasing iteration with widening

\[\begin{align*}
X_1^1 &= \{X = 2, I = 0\} \\
X_2^1 &= \{X = 2, I = 0\} \cup \{X \in [-1, 4], I = 1\} \\
&= \{X = 2, I = 0\} \cup \{I \in [0,1], 2 - 3I \leq X \leq 2I + 2\} \\
&= \{I \geq 0, 2 - 3I \leq X \leq 2I + 2\}
\end{align*}\]

decreasing iteration: to get \( I \leq 10 \)

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

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

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

\[ X^\#_1 = \{ X = 2, I = 0 \} \]
\[ X^\#_2 = \{ X = 2, I = 0 \} \cup \{ X \in [-1, 4], I = 1 \} \]
\[ X^\#_3 = \{ X = 2, I = 0 \} \cup \{ I \in [1, 10], 2 - 3I \leq X \leq 2I + 2 \} \]

\[ X^\#_1 \]
\[ F^\#(X^\#_1) \]
\[ X^\#_2 \]
\[ X^\#_3 \]
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>

- 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: $x + y \leq c \land -y + z \leq d \implies x + z \leq c + d$
    - 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:** $\mathbf{M} \times \vec{V} \geq \vec{C}$ for a fixed $\mathbf{M}$
Integers

**Issue:**

In relational domains we used implicitly real-valued environments $\forall \rightarrow \mathbb{R}$; our concrete semantics is based on integer-valued environments $\forall \rightarrow \mathbb{Z}$.

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

$$\gamma_\mathbb{Z}(X^\#) \overset{\text{def}}{=} \gamma(X^\#) \cap \mathbb{Z}^{\mathcal{V}}$$

(keep only integer points)

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

- $\subseteq^\#$ and $\equiv^\#$ are no longer exact
  
  e.g., $\gamma(2X = 1) \neq \gamma(\bot)$, but $\gamma_\mathbb{Z}(2X = 1) = \gamma(\bot) = \emptyset$

- $\cap^\#$ and affine tests are still exact

- Affine and non-deterministic assignments are no longer exact
  
  e.g., $R^1 = (Y = 2X)$, $S^1[X \leftarrow [-\infty, +\infty]] R^1 = \top$, but $S[X \leftarrow [-\infty, +\infty]] (\gamma_\mathbb{Z}(R^1)) = \mathbb{Z} \times (2\mathbb{Z})$

- All the operators are **still sound**

$$\mathbb{Z}^{\mathcal{V}} \subseteq \mathbb{R}^{\mathcal{V}}$$, so $\forall X^1 : \gamma_\mathbb{Z}(X^1) \subseteq \gamma(X^1)$

(in general, soundness, exactness, optimality depend on the definition of $\gamma$)
Integers (cont.)

**Possible solutions:**

- **enrich** the domain (add exact representations for operation results)
  - congruence equalities: \( \land_i \sum_j \alpha_{ij} V_j \equiv \beta_i [\gamma_i] \) (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 \( \gamma_Z(X^1) \)
    - emptiness testing: integer programming
      - NP-hard, while linear programming is P

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

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

**Soundness property:**
If $\gamma(X^\#) \models e \equiv e'$ then:

- $S[V \leftarrow e] \gamma(X^\#) \subseteq \gamma(S[V \leftarrow e'] X^\#)$
- $S[e \triangleright 0?] \gamma(X^\#) \subseteq \gamma(S[e' \triangleright 0?] X^\#)$

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

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

**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 \(V_k\)
Operations on affine interval forms

- adding and subtracting two forms
- multiplying and dividing a form by an interval

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

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

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

\[
\ldots
\]

**Projection**  \( \pi_k : E\# \to expr_\ell \)

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

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

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 \( \ell : (\text{expr} \times E^\#) \to \text{expr}_\ell \)

Defined by induction on the syntax of expressions:

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

Linearization application

Property  soundness of the linearization:

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

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

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

$\begin{align*}
Y & \leftarrow \text{rand}(0, 1000); \\
T & \leftarrow \text{rand}(-1, 1); \\
X & \leftarrow T \times Y
\end{align*}$

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

Apron library

Underlying libraries & abstract domains
- box
- intervals
- octagons
- 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: \( A \rightarrow \wp(\mathbb{Z}^n \times \mathbb{R}^m) \)
dimensions and space dimensionality

Variables and Environments

Semantics: \( A \rightarrow \wp(V \rightarrow \mathbb{Z} \cup \mathbb{R}) \)

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 $\mathbb{V} \to \mathbb{R}$.

$\mathbb{V}$ 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 $[1, n]$
### Concrete expression trees

```ocaml
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**

```ocaml
type Texpr1.unop = Neg | ···
```

**binary operators**

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

**numeric type:**

(we only use integers, but reals and floats are also possible)

```ocaml
type Texpr1.typ = Int | ···
```

**rounding direction:**

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

```ocaml
type Texpr1.round = Zero | ···
```
Internal expression form: type \(\text{Texpr1.t}\)

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

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

Coefficients: type \(\text{Coeff.t}\)

Can be either a scalar \(\{c\}\) or an interval \([a, b]\)

We can use the \text{Mpqf} module to convert from strings to arbitrary precision integers, before converting them into \(\text{Coeff.t}\):

- for scalars \(\{c\}\):
  \(\text{Coeff.s_of_mpqf (Mpqf.of_string c)}\)

- for intervals \([a, b]\):
  \(\text{Coeff.i_of_mpqf (Mpqf.of_string a) (Mpqf.of_string b)}\)
Using the Apron Library

**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:

```plaintext```
let c = Tcons1.make texpr1 typ in
let ar = Tcons1.arrayMake env 1 in
Tcons1.arraySet ar 0 c
```
```plaintext```
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 $\circ$

- **Abstract1.is_leq**: `Manager.t -> t -> t -> bool`  
  $\subseteq^1$: 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 ` Manager.t` and ` 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 (`∪`, `∩`, `⊓`, `⊆`) are defined on the **same environment**

Failure to ensure this results in a `Manager.Error` exception
Using the Apron Library

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