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
  - Project: relational analysis with the Apron library

- **Next week:** selected advanced topics on abstract domains
  - (Not needed for the project)
Motivation
Relational assignments and tests

Example

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

Interval analysis:

- \( S^\#[ X > Y? ] \) is abstracted as the identity

\[ S^\#[ \text{if } X > Y \text{ then } \cdots ] R^\# = R^\# \]

- \( D \leftarrow Y - X \) gives \( D \in [0, 10] -^\# [0, 10] = [-10, 10] \)

- the assertion \( D \geq 0 \) fails
Motivation

Relational assignments and tests

Example

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

Solution: relational domain

- represent explicitly the information \( X \leq Y \)
- infer that \( X \leq Y \) holds after the \text{if} \cdots \text{then} \cdots \text{else} \cdots \\
  \( X \leq Y \) both after \( X \leftarrow Y \) when \( X > Y \), and after \text{skip} when \( X \leq Y \)
- 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
Motivation

Relational loop invariants

Example

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

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 \( X \in [0, +\infty) \implies \text{assert fails} \)

• 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 \( X \in [0, +\infty) \implies \text{assert fails} \)
  (the test \( I \leq 1000 \) only refines \( I \), but gives no information on \( X \))

• without widening, we get \( I = 1001 \) and \( X = 1000 \implies \text{assert passes} \)
  but we need 1000 iterations! (\( \simeq \) concrete fixpoint computation)
Relational loop invariants

Example

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

**Solution:** relational domain

- infer a \textbf{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 \textbf{expressive loop invariant} to deduce it
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

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

(with a suitable machine representation)
Affine equality representation

**Machine representation:**

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

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

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

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

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

- \( \forall i \leq m : \exists k_i : M_{ik_i} = 1 \) and
  \( \forall c < k_i : M_{ic} = 0, \forall i \neq i : M_{lk_i} = 0 \),
- 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 \( m \leq n = |V| \), the memory cost is in \( \mathcal{O}(n^2) \) at worst
- \( \bot \) is represented as the empty equation system: \( m = 0 \)
**Galois connection:**

between arbitrary subsets and affine subsets

\[(\mathcal{P}(\mathbb{R}^{\mathbb{V}}), \subseteq) \leftrightarrow (\text{Aff}(\mathbb{R}^{\mathbb{V}}), \subseteq)\]

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

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

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

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

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

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

Notes:

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

Let $\mathbf{M} \times \mathbf{V} = \mathbf{C}$ be a system, not necessarily in normal form. The Gaussian reduction tells in $O(n^3)$ time:

- whether the system is satisfiable, and in that case
- gives an equivalent system in normal form

i.e.: it returns an element in $\mathcal{E}^\#$

Example:

$$\begin{cases} 2X + Y + Z = 19 \\ 2X + Y - Z = 9 \\ 3Z = 15 \end{cases}$$

$\Downarrow$

$$\begin{cases} X + 0.5Y = 7 \\ Z = 5 \end{cases}$$
Gaussian reduction algorithm: \( Gauss(\langle M, \vec{C} \rangle) \)

\[
\begin{align*}
\text{Gauss}(\langle M, \vec{C} \rangle) & \quad \text{for } c \text{ from 1 to } n \quad (\text{column } c) \\
& \quad \text{if } \exists \ell > r : M_{\ell c} \neq 0 \quad (\text{pivot } \ell) \\
& \quad \quad r \leftarrow r + 1 \\
& \quad \quad \text{swap } \langle \vec{M}_\ell, C_\ell \rangle \text{ and } \langle \vec{M}_r, C_r \rangle \\
& \quad \quad \text{divide } \langle \vec{M}_r, C_r \rangle \text{ by } M_{rc} \\
& \quad \quad \text{for } j \text{ from 1 to } n, j \neq r \\
& \quad \quad \quad \text{replace } \langle \vec{M}_j, C_j \rangle \text{ with } \langle \vec{M}_j, C_j \rangle - M_{jc} \langle \vec{M}_r, C_r \rangle \\
& \quad \quad \text{if } \exists \ell : \langle \vec{M}_\ell, C_\ell \rangle = \langle 0, \ldots, 0, c \rangle, c \neq 0 \\
& \quad \quad \quad \text{then return } \bot \\
& \quad \text{remove all rows } \langle \vec{M}_\ell, C_\ell \rangle \text{ that equal } \langle 0, \ldots, 0, 0 \rangle
\end{align*}
\]
Affine equality operators

Abstract operators:

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

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

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

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

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

$S^\#[e \otimes e'] X^\# \overset{\text{def}}{=} X^\#$ 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

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

\[
\Rightarrow \quad \begin{cases} 
X - Y = 3 
\end{cases}
\]

The operator is **exact**
Affine equality assignment

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

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

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

if \( \alpha_j \neq 0 \), \( \langle M, \vec{C} \rangle \) 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:** \( \alpha_j = 0 \)

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

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

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

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

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

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

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

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

\[
S^\#[V_j \leftarrow e] \times^\# \triangleq S^\#[V_j \leftarrow [-\infty, +\infty]] \times^\#
\]

(imprecise but sound)
Affine equality join

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

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

**Example:**

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

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

Use the row \( \begin{pmatrix} \vec{0} & 1 & M_2 \end{pmatrix} \) to create \( \vec{\beta} \) in the left argument

Then remove the row \( \begin{pmatrix} \vec{0} & 1 & M_2 \end{pmatrix} \)
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

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

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: \( F^\#(I^\#) = (X = 9 \land Y = 110) \)
- \( X^\# \overset{\text{def}}{=} I^\# \cup F^\#(I^\#) = (10X + Y = 200) \)
- \( X^\# \) is stable

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

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

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

$E^\# \simeq \{ \text{closed convex polyhedra of } \forall \rightarrow \mathbb{R} \}$

Note: polyhedra need not be bounded (≠ polytopes)
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**

\[ [P, R] \text{ where} \]

- \( P \in \mathbb{Q}^{n \times p} \) 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:

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

\[
0x + 0y + 1z \leq 1 \iff (0, 0, 1)
\]
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) \( x \leq 0, x \geq 0, y \leq 0, y \geq 0 \)
Polyhedra representations (cont.)

- No bound on the size of representations (even minimal ones)
- No best abstraction $\alpha$

Example: a disc has infinitely many polyhedral over-approximations, but no best one
Chernikova’s algorithm

Algorithm by Chernikova (1968), improved by LeVerge (1992) to switch from a constraint system to an equivalent generator system

**Motivation:** most operators are easier on one representation

- By **duality**, we can use the same algorithm to switch from generators to constraints.
- The minimal generator system can be **exponential** in the original constraint system (e.g., hypercube: \(2n\) constraints, \(2^n\) vertices).
- **Equality** constraints and lines (pairs of opposed rays) may be handled separately and more efficiently.
- Chernikova’s algorithm minimizes the representation on-the-fly (not presented here).

**Algorithm:** incrementally add constraints one by one

Start with:

\[
\begin{align*}
P_0 &= \{ (0, \ldots, 0) \} & \text{(origin)} \\
R_0 &= \{ \bar{x}_i, -\bar{x}_i \mid 1 \leq i \leq n \} & \text{(axes)}
\end{align*}
\]
Chernikova’s algorithm (cont.)

Update \([P_{k-1}, R_{k-1}]\) to \([P_k, R_k]\)
by adding one constraint \(\vec{M}_k \cdot \vec{V} \geq C_k \in \langle \mathbf{M}, \vec{C} \rangle\):

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 \(\vec{O}\) to \(P_k\):

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

- 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 $\mathbb{P}_k$:
  $$\vec{O} \triangleq \vec{P} + \frac{C_k - \vec{M}_k \cdot \vec{P}}{\vec{M}_k \cdot \vec{R}} \vec{R}$$
Chernikova’s algorithm example

Example:

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

Example:

\[ Y \geq 1 \]

\[ P_0 = \{(0, 0)\} \]
\[ P_1 = \{(0, 1)\} \]

\[ R_0 = \{(1, 0), (-1, 0), (0, 1), (0, -1)\} \]
\[ R_1 = \{(1, 0), (-1, 0), (0, 1)\} \]
Chernikova’s algorithm example

Example:

\[ Y \geq 1 \]
\[ X + Y \geq 3 \]

\[ \mathbf{P}_0 = \{(0, 0)\} \]
\[ \mathbf{P}_1 = \{(0, 1)\} \]
\[ \mathbf{P}_2 = \{(2, 1)\} \]

\[ \mathbf{R}_0 = \{(1, 0), (-1, 0), (0, 1), (0, -1)\} \]
\[ \mathbf{R}_1 = \{(1, 0), (-1, 0), (0, 1)\} \]
\[ \mathbf{R}_2 = \{(1, 0), (-1, 1), (0, 1)\} \]
Chernikova’s algorithm example

Example:

\[ P_0 = \{(0, 0)\} \]
\[ R_0 = \{(1, 0), (-1, 0), (0, 1), (0, -1)\} \]
\[ P_1 = \{(0, 1)\} \]
\[ R_1 = \{(1, 0), (-1, 0), (0, 1)\} \]
\[ P_2 = \{(2, 1)\} \]
\[ R_2 = \{(1, 0), (-1, 1), (0, 1)\} \]
\[ P_3 = \{(2, 1), (1, 2)\} \]
\[ R_3 = \{(0, 1), (1, 1)\} \]
Operators on polyhedra

Abstract operators:

Given \( X^\#, Y^\# \neq \bot \), we define:

\[
X^\# \subseteq^\# Y^\# \iff \\
\forall \vec{P} \in P_{X^\#} : M_{Y^\#} \times \vec{P} \geq \vec{C}_{Y^\#} \\
\forall \vec{R} \in R_{X^\#} : M_{Y^\#} \times \vec{R} \geq \vec{0}
\]

\[
X^\# \equiv^\# Y^\# \iff X^\# \subseteq^\# Y^\# \text{ and } Y^\# \subseteq^\# X^\#
\]

\[
X^\# \cap^\# Y^\# \defeq \langle \left[ \begin{array}{c} M_{X^\#} \\ M_{Y^\#} \end{array} \right], \left[ \begin{array}{c} \vec{C}_{X^\#} \\ \vec{C}_{Y^\#} \end{array} \right] \rangle \quad \text{(join constraint sets)}
\]

\( \subseteq^\#, =^\# \text{ and } \cap^\# \) are exact (in \( \mathcal{P}(\forall \to \mathbb{R}) \))
**Join:** \[ X^\# \cup^\# Y^\# \overset{\text{def}}{=} \begin{bmatrix} \{ P_{X^\#}, P_{Y^\#} \} \quad [ R_{X^\#}, R_{Y^\#} \} \end{bmatrix} \] (join generator sets)

**Examples:**

- two polytopes
- a point and a line

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

we get the topological closure of the convex hull of \(\gamma(X^\#) \cup \gamma(Y^\#)\)
Operators on polyhedra (cont.)

**Affine tests:**

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

**Non-deterministic assignment:**

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

- these operators are exact (in \( P(\forall \rightarrow \mathbb{R}) \))
- other tests can be abstracted as \( S^\#[c?] X^\# \overset{\text{def}}{=} X^\# \) (sound but not optimal)
Affine assignment:

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

if \( \alpha_j = 0 \),

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

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

- similar to the assignment in the equality domain
- the assignment is exact (in \( P(\mathbb{V} \to \mathbb{R}) \))
- 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)
Polyhedra widening

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

**Definition:**

Take \( X^\# \) and \( Y^\# \) in minimal constraint-set form

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

We suppress any unstable constraint \( c \in X^\# \), i.e., \( Y^\# \not\subseteq^\# \{ c \} \)

**Example:**

![Diagram](image)
Polyhedra widening

\( \mathcal{E}' \) has strictly increasing infinite chains \( \implies \) we need a widening

**Definition:**

Take \( X' \) and \( Y' \) in minimal constraint-set form

\[
X' \triangledown Y' \overset{\text{def}}{=} \{ c \in X' \mid Y' \subseteq \{ c \} \} \\
\cup \{ c \in Y' \mid \exists c' \in X' : X' = (X' \setminus c') \cup \{ c \} \}
\]

We suppress any unstable constraint \( c \in X' \), i.e., \( Y' \not\subseteq \{ c \} \)

We also keep constraints \( c \in Y' \) equivalent to those in \( X' \), i.e., when \( \exists c' \in X' : X' = (X' \setminus c') \cup \{ c \} \)

**Example:**

![Diagram showing polyhedra widening](image)
Example

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

Loop invariant:

increasing iterations with widening:

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

decreasing iteration: (recover \( I \leq 10 \))

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

at the loop exit, we find eventually: \( I = 10 \land X \in [-28, 22] \)
Partial conclusion

**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)
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: \( M \times \vec{V} \geq \vec{C} \) for a fixed \( M \)
**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 $\gamma(X^\#) \subseteq \mathbb{R}^{\forall}$, but:

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

(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., $R^\# = (Y = 2X)$, $S^\#[X \leftarrow [-\infty, +\infty]] R^\# = \top$, but $S[X \leftarrow [-\infty, +\infty]] (\gamma_{\mathbb{Z}}(R^\#)) = \mathbb{Z} \times (2\mathbb{Z})$

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

(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: $\bigwedge_i \sum_j \alpha_{ij} V_j \equiv \beta_i \lfloor \gamma_i \rfloor$ (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 $\gamma \mathbb{Z}(X^\#)$
    - 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!
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

Developer interface

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

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

Semantics:
- $A \xrightarrow{\gamma} \wp(\mathbb{Z}^n \times \mathbb{R}^m)$
- dimensions and space dimensionality

Variables and Environments
- Semantics: $A \xrightarrow{\gamma} \wp(V \rightarrow \mathbb{Z} \cup \mathbb{R})$

http://apron.cri.ensmp.fr/library
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)
Using the Apron Library

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 $\mathbb{V} \rightarrow \mathbb{R}$
$\mathbb{V}$ 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: type \texttt{Texpr1.expr}

\begin{align*}
\text{type } \texttt{expr} &= \ | \ \text{Cst of Coeff.t} & \text{(constants)} \\
& \quad \ | \ \text{Var of Var.t} & \text{(variables)} \\
& \quad \ | \ \text{Unop of unop * expr * typ * round} & \text{(unary op.)} \\
& \quad \ | \ \text{Binop of binop * expr * expr * typ * round} & \text{(binary op.)}
\end{align*}

- **unary operators**
  \begin{align*}
  \text{type } \texttt{Texpr1.unop} &= \text{Neg} \ | \ldots
  \end{align*}

- **binary operators**
  \begin{align*}
  \text{type } \texttt{Texpr1.binop} &= \text{Add} \ | \ \text{Sub} \ | \ \text{Mul} \ | \ \text{Div} \ | \ldots
  \end{align*}

- **numeric type:**
  \begin{align*}
  \text{(we only use integers, but reals and floats are also possible)}
  \text{type } \texttt{Texpr1.typ} &= \text{Int} \ | \ldots
  \end{align*}

- **rounding direction:**
  \begin{align*}
  \text{(only useful for the division on integers; we use rounding to zero, i.e., truncation)}
  \text{type } \texttt{Texpr1.round} &= \text{Zero} \ | \ldots
  \end{align*}
**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`

Coefficients 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_mpfq (Mpfq.of_string c)
  ```

- For intervals $[a, b]$:
  
  ```
  Coeff.i_of_mpfq (Mpfq.of_string a) (Mpfq.of_string b)
  ```
**Constraints:** type \( T\text{cons1} \).t  

constructor \( expr \bowtie 0 \):  

- \( T\text{cons1}.\text{make} \): \( T\text{expr1} \).t \( \rightarrow \) \( T\text{Cons1}.\text{typ} \) \( \rightarrow \) \( T\text{cons1} \).t  

where:  

\[
\text{type } T\text{cons1}.\text{typ} = \begin{array}{c}
\text{SUPEQ} \quad | \quad \text{SUP} \quad | \quad \text{EQ} \quad | \quad \text{DISEQ} \quad | \quad \ldots \\
\geq \quad | \quad > \quad | \quad = \quad | \quad \neq
\end{array}
\]

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

**Constraint arrays:** type \( T\text{cons1}.\text{earray} \)  

abstract operators do not use constraints, but constraint arrays instead  

Example: constructing an array \( ar \) containing a single constraint:  

```plaintext  
let c = T\text{cons1}.\text{make} \ \text{texpr1} \ \text{typ} \ \text{in}  
let ar = T\text{cons1}.\text{array_make} \ \text{env} \ 1 \ \text{in}  
T\text{cons1}.\text{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 $\nabla$

- **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:

```ml
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 (∪, ∩, ⊓, ⊆) 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:

    OCAMLLINC = ⋯ -I +zarith -I +apron -I +gmp
    CMA = bigarray.cma gmp.cma apron.cma polkaMPQ.cma
Fall-back assignments and tests

```ocaml
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