Existentially-quantified LRA

So far, we have considered formulas that may contain free, existentially quantified, and universally quantified variables.

For the special case of conjunction of linear inequations in which *all* variables are existentially quantified, there are more efficient methods available.

Main idea: reduce satisfiability problem to optimization problem.

The Simplex Method

Developed independently by Kantorovich (1939), Dantzig (1948).

Polynomial-time average-case complexity; worst-case time complexity is exponential, though.

Goal:

Solve a linear optimization (also called: linear programming) problem for given numbers $a_{ij}, b_i, c_j \in \mathbb{R}$:

$$\begin{array}{ll} \text{maximize} & \sum_{1 \leq j \leq n} c_j x_j \\ \text{for} & \bigwedge_{1 \leq i \leq m} \sum_{1 \leq j \leq n} a_{ij} x_j \leq b_i \end{array}$$

or in vectorial notation:

$$\begin{array}{ll} \text{maximize} & \vec{c}^{\top} \vec{x} \\ \text{for} & A\vec{x} \leq \vec{b} \end{array}$$

Idea:

$A\vec{x} \leq \vec{b}$ describes a convex polyhedron.

Pick one vertex of the polyhedron,
then follow the edges of the polyhedron towards an optimal solution.

By convexity, the local optimum found in this way is also a global optimum.

Details: see special lecture on optimization

Using an optimization procedure for checking satisfiability:

Goal: Check whether $A\vec{x} \leq \vec{b}$ is satisfiable.

To use the Simplex method, we have to transform the original (possibly empty) polyhedron into another polyhedron that is non-empty and for which we know one initial vertex.

Every real number can be written as the difference of to non-negative real numbers. Use this idea to convert $A\vec{x} \le \vec{b}$ into an equisatisfiable inequation system $\vec{y} \ge \vec{0}$, $B\vec{y} \le \vec{b}$ for new variables $\vec{y}$.

Multiply those inequations of the inequation system $B\vec{y} \le \vec{b}$ in which the number on the right-hand side is negative by -1. We obtain two inequation systems $D_1\vec{y} \le \vec{g}_1$, $D_2\vec{y} \ge \vec{g}_2$, such that $\vec{g}_1 \ge \vec{0}$, $\vec{g}_2 > 0$.

Now solve

$$\begin{array}{ll} \text{maximize} & \vec{1}^\top(D_2\vec{y} - \vec{z}) \\ \text{for} & \vec{y}, \vec{z} \ge \vec{0} \\ & D_1\vec{y} \le \vec{g}_1 \\ & D_2\vec{y} - \vec{z} \le \vec{g}_2 \end{array}$$

where $\vec{z}$ is a vector of new variables with the same size as $\vec{g}_2$.

Observation 1: $\vec{0}$ is a vertex of the polyhedron of this optimization problem.

Observation 2: The maximum is $\vec{1}^\top\vec{g}_2$ if and only if $\vec{y} \ge \vec{0}$, $D_1\vec{y} \le \vec{g}_1$, $D_2\vec{y} \ge \vec{g}_2$ has a solution.
($\Rightarrow$): If $\vec{1}^\top(D_2\vec{y} - \vec{z}) = \vec{1}^\top\vec{g}_2$ for some $\vec{z}$, then $D_2\vec{y} - \vec{z} = \vec{g}_2$, hence $D_2\vec{y} = \vec{g}_2 + \vec{z} \ge \vec{g}_2$.
($\Leftarrow$): $\vec{1}^\top(D_2\vec{y} - \vec{z})$ can never be larger than $\vec{1}^\top\vec{g}_2$. If $\vec{y} \ge \vec{0}$, $D_1\vec{y} \le \vec{g}_1$, $D_2\vec{y} \ge \vec{g}_2$ has a solution, choose $\vec{z} = D_2\vec{y} - \vec{g}_2$; then $\vec{1}^\top(D_2\vec{y} - \vec{z}) = \vec{1}^\top\vec{g}_2$.

1.4 Non-linear Real Arithmetic

Tarski (1951): Quantifier elimination is possible for *non-linear* real arithmetic (or more generally, for real-closed fields). His algorithm had non-elementary complexity, however.

Improved algorithms by Collins and Hong: Cylindrical algebraic decomposition (CAD).

Cylindrical Algebraic Decomposition

Given: First-order formula over atoms of the form $f_i(\vec{x}) \sim 0$, where the f_i are polynomials over variables $\vec{x}$.

Goal: Decompose $\mathbb{R}^n$ into a finite number of regions such that all polynomials have invariant sign on every region X:

$$\begin{array}{ll} \forall i \ (& \forall \vec{x} \in X.\ f_i(\vec{x}) < 0 \\ & \lor\ \forall \vec{x} \in X.\ f_i(\vec{x}) = 0 \\ & \lor\ \forall \vec{x} \in X.\ f_i(\vec{x}) > 0\ ) \end{array}$$

Note: implementation needs exact arithmetic using algebraic numbers (i. e., zeroes of univariate polynomials with integer coefficients).

1.5 Real Arithmetic incl. Transcendental Functions

Real arithmetic with exp/log: decidability unknown.

Real arithmetic with trigonometric functions: undecidable

The following formula holds exactly if $x \in \mathbb{Z}$:

$$\exists y\ (\sin(y) = 0 \wedge 3 < y \wedge y < 4 \wedge \sin(x \cdot y) = 0)$$

(note that necessarily $y = \pi$).

Consequence: Peano arithmetic (which is undecidable) can be encoded in real arithmetic with trigonometric functions.

However, real arithmetic with transcendental functions is decidable for formulas that are *stable under perturbations*, i. e., whose truth value does not change if numeric constants are modified by some sufficiently small ε.

Example:

Stable under perturbations: $\exists x\ x^2 \leq 5$

Not stable under perturbations: $\exists x\ x^2 \leq 0$
(Formula is true, but if we subtract an arbitrarily small $\varepsilon > 0$ from the right-hand side, it becomes false.)

Unsatisfactory from a mathematical point of view, but sufficient for engineering applications (where stability under perturbations is necessary anyhow).

Approach:

Interval arithmetic + interval bisection if necessary (Ratschan).

Sound for general formulas; complete for formulas that are stable under perturbations; may loop forever if the formula is not stable under perturbations.

1.6 Linear Integer Arithmetic

Linear integer arithmetic = Presburger arithmetic

Decidable (Presburger, 1929), but quantifier elimination is only possible if additional divisibility operators are present:

$\exists x\ (y = 2x)$ is equivalent to $\text{divides}(2, y)$ but not to any quantifier-free formula over the base signature.

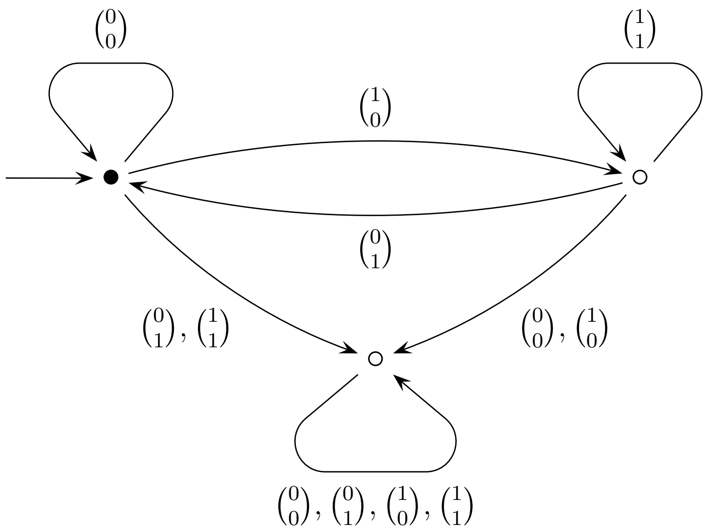
Deciding Presburger Arithmetic using Finite Automata

Decidability of Presburger arithmetic can e. g. be shown by encoding Presburger formulas as finite automata.

Encode tuples of naturals (or integers) as words of tuples of bits (least significant bit first!):

$$\begin{pmatrix}15\\1\\4\end{pmatrix} \mapsto \begin{pmatrix}1&1&1&1\\1&0&0&0\\0&0&1&0\end{pmatrix} \mapsto \begin{pmatrix}1\\1\\0\end{pmatrix}\begin{pmatrix}1\\0\\0\end{pmatrix}\begin{pmatrix}1\\0\\1\end{pmatrix}\begin{pmatrix}1\\0\\0\end{pmatrix}$$

Construct automata for atoms, e. g., for $2x = y$:



Translate logical operators into automata operators:

$\wedge \;\mapsto\; \cap$

$\vee \;\mapsto\; \cup$

$\neg \;\mapsto\;$ complementation

$\exists x \;\mapsto\;$ cylindrification: drop x-component in transition labels

satisfiability of formulas $\mapsto$ non-emptyness test for automata

Finite automata yield an easy proof for the decidability of Presburger arithmetic; in practice, however, methods that are based on decision procedures for linear rational arithmetic are more efficient, at least for the existentially quantified fragment.

The Omega Test

Omega test (Pugh, 1991): variant of Fourier–Motzkin for conjunctions of (in-)equations in linear integer arithmetic.

Idea:

- Perform easy transformations, e. g.:
  $3x + 6y \leq 8 \ \mapsto\ 3x + 6y \leq 6 \ \mapsto\ x + 2y \leq 2$
  $3x + 6y = 8 \ \mapsto\ \bot$
  (since $3x + 6y$ must be divisible by 3).
- Eliminate equations
  (easy, if one coefficient is 1; tricky otherwise).
- If only inequations are left:
  no real solutions $\rightarrow$ unsatisfiable for $\mathbb{Z}$
  "sufficiently many" real solutions $\rightarrow$ satisfiable for $\mathbb{Z}$
  otherwise: branch

What does "sufficiently many" mean?

Consider inequations $ax \leq s$ and $bx \geq t$ with $a, b \in \mathbb{Z}$ and polynomials s, t.

If these inequations have real solutions, the interval of solutions ranges from $\frac{1}{b}t$ to $\frac{1}{a}s$.

The longest possible interval of this kind that does not contain any integer ranges from $i + \frac{1}{b}$ to $i - 1 - \frac{1}{a}$ for some $i \in \mathbb{Z}$; it has the length $1 - \frac{1}{a} - \frac{1}{b}$.

Consequence:

If $\frac{1}{a}s > \frac{1}{b}t + (1 - \frac{1}{a} - \frac{1}{b})$, or equivalently, $bs \geq at + ab - a - b + 1$ is satisfiable, then the original problem must have integer solutions.

It remains to consider the case that $bs \geq at$ is satisfiable (hence there are real solutions) but $bs \geq at + ab - a - b + 1$ is not (hence the interval of real solutions need not contain an integer).

In the latter case, $bs \leq at + ab - a - b$ holds, hence for every solution of the original problem:

$t \leq bx \leq \frac{b}{a}s \leq t + (b - 1 - \frac{b}{a})$

and if x is an integer, $t \leq bx \leq t + \left\lfloor b - 1 - \frac{b}{a} \right\rfloor$

$\Rightarrow$ Branch non-deterministically:
Add one of the equations $bx = t + i$ for $i \in \{0, \ldots, \lfloor b - 1 - \frac{b}{a} \rfloor\}$.

Alternatively, if $b > a$:
Add one of the equations $ax = s - i$ for $i \in \{0, \ldots, \lfloor a - 1 - \frac{a}{b} \rfloor\}$.

Note: Efficiency depends highly on the size of coefficients. In applications from program verification, there is almost always some variable with a very small coefficient. If all coefficients are large, the branching step gets expensive.

Branch-and-Cut

Alternative approach: Reduce satisfiability problem to optimization problem (like Simplex). ILP, MILP: (mixed) integer linear programming.

Two basic approaches:

Branching: If the simplex algorithm finds a solution with $x = 2.7$, add the inequation $x \leq 2$ or the inequation $x \geq 3$.

Cutting planes: Derive an inequation that holds for all real solutions, then round it to obtain an inequation that holds for all integer solutions, but not for the real solution found previously.

Example:

Given:
$$\begin{aligned} 2x - 3y &\leq 1 \\ 2x + 3y &\leq 5 \\ -5x - 4y &\leq -7 \end{aligned}$$

Simplex finds an extremal solution $x = \frac{3}{2}$, $y = \frac{2}{3}$.

From the first two inequations, we see that $4x \leq 6$, hence $x \leq \frac{3}{2}$.

$\Rightarrow$ Add the inequation $x \leq \lfloor \frac{3}{2} \rfloor = 1$, which holds for all integer solutions, but cuts off the solution $(\frac{3}{2}, \frac{2}{3})$.

In practice:

Use both: Alternate between branching and cutting steps.
Better performance than the individual approaches.

1.7 C-Arithmetic

In languages like C: Bounded integer arithmetic (modulo 2^n), in device drivers also combined with bitwise operations.

Bit-Blasting (encode everything as boolean circuits, use DPLL):

Naive encoding: possible, but often too inefficient.

If combined with over-/underapproximation techniques (Bryant, Kroening, et al.): successful.