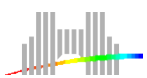
The Care and Feeding of Polyhedra

Paul Feautrier

ENS de Lyon, LIP, Compsys, INRIA, UCBL
Paul.Feautrier@ens-lyon.fr

May 13, 2013



Outline of the Polyhedral Model

In the polyhedral model, a program is not a collection of functions but a set of instances of statements or *operations*.

```
for(i=0; i<n; i++){
  S: a[i] = 0.;
}
```

The set of operations is $\{< S, i > | 0 \leq i < n\}$.
Generalizes to the iteration vector and the iteration domain.

Important Concepts

The Happens Before relation:

$$< S, i > \prec < S, i' > \equiv i < i'$$

generalizes to lexicographic order on the iteration vectors.

Dependences:
An instance u depends on v if $v \prec u$ and if both access the same memory cell, one of the accesses being a write.

Representing Large Sets

Iteration domains are large, indefinite or even infinite sets:

- A 1 GFlops processor generates 10^9 operations per second
- The number of dependences may be up to the square of the number of operations
- Enumerating the set elements is impossible

All these sets must be represented as the solutions of a system of constraints:

$$S = \{x \in U | Q(x) = \textbf{true}\}$$

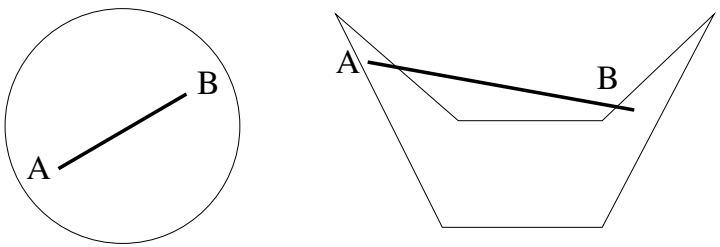
where U is some *universe*.

Loop Programs

For regular loop programs, the constraints are derived from the loop bounds which are usually constants or affine functions of parameters and surrounding loop counters ....
.... and the variables are integers.
Hence the sets are Z-polyhedra or unions of Z-polyhedra.

An Important Concept: Convexity

A set in R^n is *convex* iff with two point A, B it contains the segment AB.



convex non convex

In analytical terms:

$$A, B \in S \Rightarrow \forall \lambda : 0 \leq \lambda \leq 1 : \lambda A + (1 - \lambda) B \ inS$$

The convex hull of a set S: the smallest convex set that includes S:

$$\{\lambda x + (1 - \lambda) y | x, y \in S, 0 \leq \lambda \leq 1\}$$

A Word on Complexity

One usually distinguish:

- Polynomial problems, for which an algorithm running in $O(n^d)$ steps is known
- Non Deterministic Polynomial problems, for which, if given a solution, it can be checked in time polynomial
- Among those, NP-complete problems, a set of problems which can be reduced to each other in time polynomial. Example: the Boolean satisfaction problem.
- Undecidable problems, for which no algorithm is know, or for which the impossibility of an algorithm is proved.

Beware of Complexity Results

- Some NP-complete problems are *usualy* easy: examples: the boolean statisfaction problem, the Integer Linear Programming problem
- Some exponential algorithms are *usualy* faster than polynomial ones: the Simplex and the Ellipsoïd algorithm
- Some undecidable problems have good heuristics or partial solutions
- Complexity results are usually asymptotic: always check that you are really on the asymptote.

Definition: Polyhedron

- A polyhedron is the set of solutions of a system of affine constraints:

$$P = \{x | Ax + b \geq 0\}$$

  A an integral matrix, b an integral vector.

- or: a polyhedron is the convex hull of a finite set of rays and vertices (Minkowsky):

$$P = \{\sum_{i=1}^{n} \lambda_i v_i + \sum_{j=1}^{m} \mu_j r_j | \lambda_i, \mu_j \geq 0, \sum_{i=1}^{n} \lambda_i = 1\}$$

- Theorem: the two representations define the same objects.

One can move from one representation to the other by Chernikova's algorithm. See Miné presentation.

Lattices

A lattice (not to be confused with lattices in order theory) is a subgroup of Z^d. Given a matrix A of size $d \times n$, the lattice generated by A is:

$$\Lambda(A) = \{\sum_{i=1}^{n} \lambda_i a_i | \lambda_i \in Z\}$$

where a_i is the $i-$th column vector of A.
Another representation:

$$L(A) = \{x | Ax \equiv 0 \mod D\}$$

Hermite Normal Form

A matrix of full row rank is in Hermite Normal Form (HNF) if $A = [B\ 0]$ where:

- B is non negative and lower triangular,
- B is non singular,
- In each column, the dominant element is on the main diagonal.

There exists a polynomial algorithm for computing the HNF of any matrix A.
If $[B\ 0]$ is the HNF of A, there exists U ($\det U = \pm 1$) such that $A = [B\ 0]U$.
Two matrices generate the same lattice if they have the same HNF.

Algorithms I: Solving a Linear System in Integers

Find the integer solution of $Ax = b$ (if it exists).

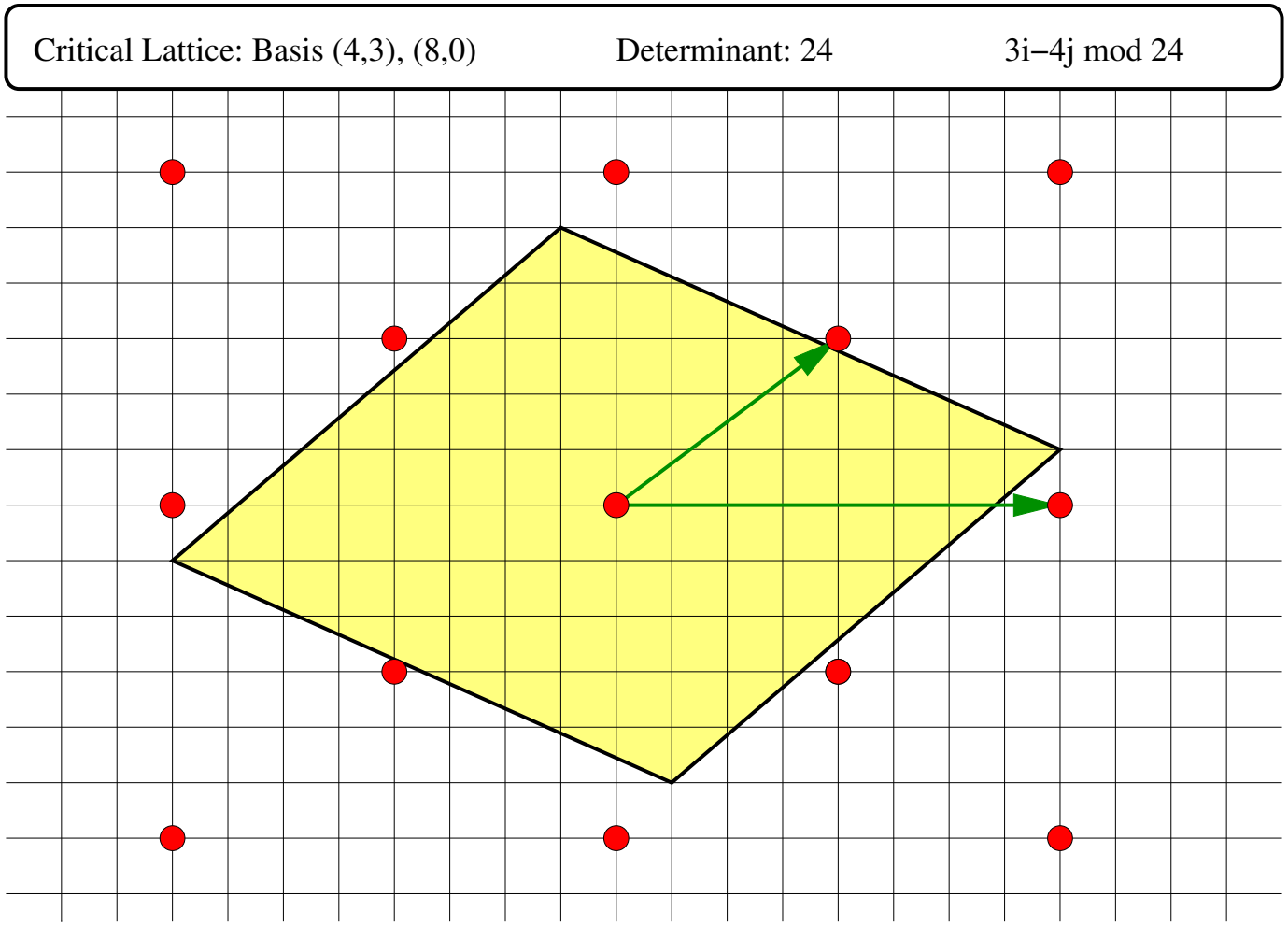
- Compute the HNF $[B\ 0]$ of A and the associated unimodular U.

$$[B0]Ux = b$$

- Set $y = Ux$. The last $n - d$ components of y are arbitrary.
- Solve $By = b$ by backward substitution, checkind divisibility at each step
- $x = U^{-1}y$.

This algorithm generalizes the well known gcd test.

Algorithm II: The Critical Lattice



Z-polyhedra

The intersection of a lattice and a polyhedron ...
The set of points with integer coordinates in a polyhedron.
Linearly bounded lattices (LBL):

$$\{y = Ax + b | \exists x : Cx + d \geq 0\}$$

Problems for Z-polyhedra are usually more difficult that for polyhedra, because convexity no longer applies.
The integer hull of S : the convex hull of the set of integer points in S.

Operations on Polyhedra

- Intersection: trivial in the constraint representation: just collect all the constraints.
- Union: the union of two polyhedra may not be a polyhedron. One usually over approximate by taking the convex hull of the union. Trivial in the Minkowsky representation.
- Image and pre-image by an affine function: trivial in both representation.
- The image of a Z-polyhedron is an LBL.

Emptyness Test, I

The most important operation on polyhedra. Application: the dependence calculation.

The Fourier-Motzkin Algorithm Outline:

$$\begin{aligned} ax \geq b, a > 0 &\Rightarrow b/a \leq x \\ a'x \geq b', a' < 0 &\Rightarrow x \leq b'/a' \\ &\Rightarrow b/a \leq b'/a' \end{aligned}$$

and x has been eliminated. If this is done systematically, one find constraints of the form $n \geq 0$, which can be checked numerically.

Two problems:

- The complexity is super-exponential
- The integer extension is complex (the Omega test, Bill Pugh).

Emptyness Test, II

The Simplex Algorithm

- Find a solution to $Ax \geq b$ or prove that none exists. A is an $m \times n$ matrix, $m \geq n$, x is an n vector and b is an m vector
- The vectors x are ranked, either by lexicographic order, or by a linear objective function
- If there is a row in A whose elements are all negative or null while b is positive, the problem has no solution
- Otherwise, select an $n \times n$ submatrix of A, A', solve $A'x = b'$ and check whether the solution satisfies the constraints
- If not, change the selection.

Details of the algorithms:

- enumerate the tentative solution in increasing or decreasing order (thus avoiding cycles)
- incremental solution of $A'x + b'$

Comparison

- Fourier-Motzkin is simpler, but has super exponential complexity, and is difficult to extend to integers
- Programming the Simplex is complex (but there are many libraries), the worst case complexity is exponential, but the algorithm is "probably" polynomial (in fact, the same complexity as Gaussian elimination).

Use Fourier-Motzkin for small and/or sparse problems, the Simplex for large ones.

Extensions

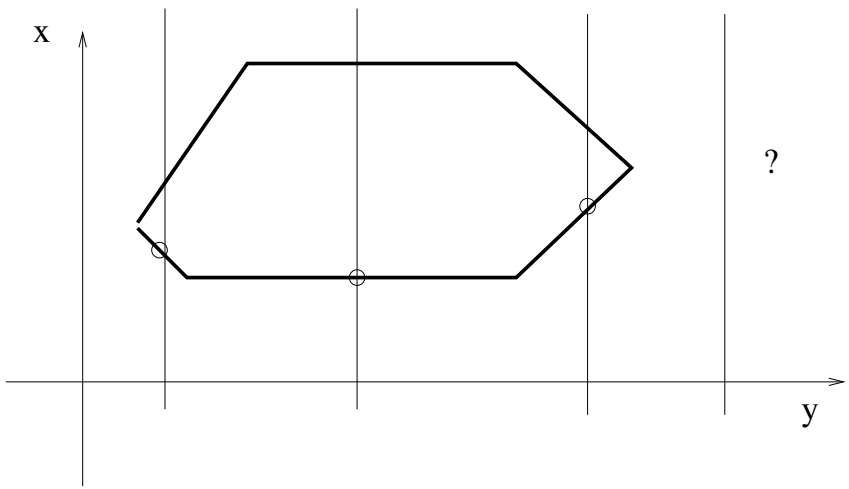
The Simplex has two extensions:

- An integer extension: find only integer solutions : Gomory cuts
- A parametric extension: find a point x in:

$$\{x | Ax + By \geq c, Dy \geq e\}$$

as a function of y.

The result is a piecewise affine function or *quast*:



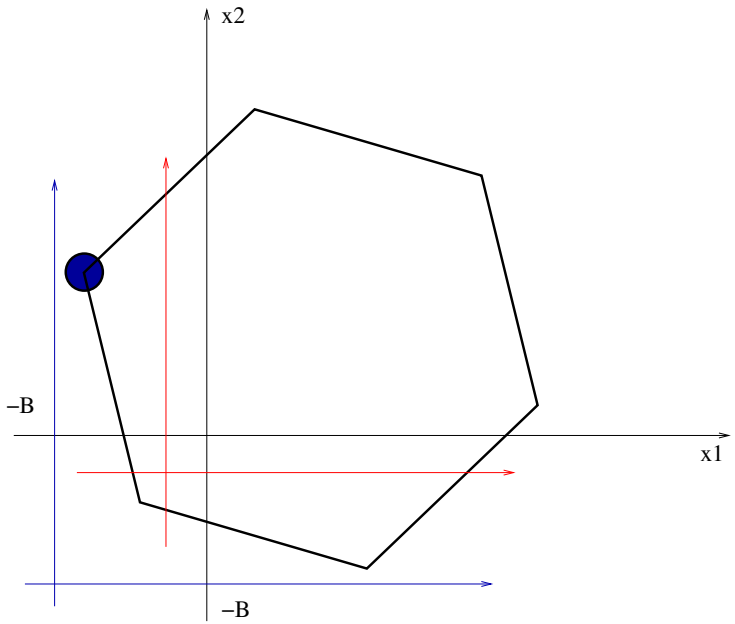
The simplex has many implementations, both academic (PIPlib, isl, Parma Polylib) and industrial (CPLEX).

Tricks or PIP

PIP has many limitations – positive variables, positive integer parameters, non-strict inequalities, but these can be side-stepped by several tricks:

- the bigparm trick: if a parameter is "large", the sign of a linear form is the sign of its coefficient
- replace signed x by $y - B, y \geq 0$ and declare B a big parameter
- to handle rational parameters, go homogeneous: instead of $Ax + By \geq c$, solve $Ax + By \geq cz$, z a new parameter, and set $z = 1$ in the solution
- strict inequalities: replace $Ax + By > c$ by $Ax + By \geq c + \epsilon$ and have $\epsilon \to 0$ in the solution. Notice that the result may not be in the original open polyhedron.

The Bigparm Trick



When B is large enough, and if the polyhedron is bounded from below, the minimum no longer move with B.

Scanning Polyhedra

Visit each integer point in a polyhedron once and only once in some order (usually, lexicographic order)
The basic method is a variant of the Fourier-Motzkin algorithm:

- select the innermost variable x and collect its lower and upper bounds
- write a loop `for(x = max(LB); x <= min(UB); x++)`
- eliminate x and recurse on the next variable

Example

Scan:

$$0 \leq x \leq n-1, 0 \leq y \leq x-1$$

with x innermost.

- Collect x constraints:

$$0 \leq x, x \leq n-1, y+1 \leq x$$

```
for(x=max(0,y+1); x <= n-1; x++)
```

- Eliminate x:

$$0 \leq y, y \leq n-2$$

```
for(y=0; y <= n-2; y++)
```

- One can still eliminate y, giving $n \geq 2$, the condition for the loop to make at least one iteration
- One can simplify the lower bound of the x loop.

The state-of-the-art implementation: CLooG.

Scanning Polyhedra Without DO Loops

Let $P(n)$ be the polyhedron to be scanned in lexicographic order. P may depend on parameters n.

- find the lexicographic minimum of P: $\mathrm{first}(n) = \min_{\ll} P$.
- given a point $x \in P$, find the point to be visited next:

$$\mathrm{next}(x) = \min_{\ll}\{y \in P(n) | x \ll y\}$$

- the problem can be solved by PIP and extends to union of Z-polyhedra by a combination of PIP and rewrite rules
- If the answer is $\perp$, the polyhedron has been completely visited, the program terminates.

```
x := first(n);
while(w != _|_){
  S;
  x := next(x,n);
}
```

Scanning Polyhedra WithoutDO Loops, II

Example: scan $0 \leq x \leq n-1, 0 \leq y \leq x-1$ with x innermost.

$$
\begin{aligned}
\text{first}(n) &= \min_{\ll}\{\begin{pmatrix} v \\ u \end{pmatrix} | 0 \leq u \leq n-1, 0 \leq v \leq u-1\} \\
&= \text{if } n \geq 2 \text{ then } \begin{pmatrix} 0 \\ 1 \end{pmatrix} \text{ else } \bot \\
\text{next}(n, \begin{pmatrix} y \\ x \end{pmatrix}) &= \{\begin{pmatrix} v \\ u \end{pmatrix} \left| \begin{array}{l} 0 \leq u \leq n-1, \\ 0 \leq v \leq u-1 \end{array} \right. , \begin{pmatrix} y \\ x \end{pmatrix} \ll \begin{pmatrix} v \\ u \end{pmatrix}\} \\
&= \text{if } i+2 > n \wedge j+3 \leq n \text{ then } \begin{pmatrix} y+1 \\ y+2 \end{pmatrix} \\
&\quad \text{else if } x+2 \leq n \text{ then } \begin{pmatrix} y \\ x+1 \end{pmatrix} \\
&\quad \text{else } \bot
\end{aligned}
$$

And the resulting program is ...

```
enum {start,stop,A} state;
state = start;
while(1){
  switch(state){
    case start:
      if((n >= 2)){
        x = 1; y = 0;
        state = A; break;
      }
      if((2 > n)){
        state = stop; break;
      }
    case stop:
      return;
```

```
    case A:
      if(( (x+2 > n) && (n >= y+3) )
        x = y+2; y = y+1;
        state = A; break;
      }
      if((n >= x+2)){
        x = x+1;
        state = A; break;
      }
      if((x+2 > n)){
        state = stop; break;
      }
  }
}
```

Counting Integer Points: How and Why

As Z-polyhedra represent objects in a program, counting integer points gives measures of complexity:

- Counting points in the iteration space: time complexity
- Counting values to be sent on a network: communication overhead
- Counting modified array cells: memory footprint, locality
- Counting barriers: synchronization

Most of the time, the polyhedra to be counted depend on one or more parameter: the counting must be symbolic, not numerical.

Eugène Ehrhart, 1906–2000

Count the integer points in $P(n) = \{x | Ax \geq nb\}$ as a function of n: A and b have integer coefficients and $P(n)$ is bounded.

- $H(n) = \mathrm{Card}\ P(n)$ is a *periodic polynomial* in n whose degree is the dimension d of $P(n)$.
- The coefficients of $H(n)$ are periodic numbers:

$$[a_0, \ldots, a_{m-1}](n) = a_{n \bmod m}$$

- The period, m is the lcm of the denominators of the coordinates of the vertices of $P(n)$
- Exception: the coefficient of highest degree is constant

Implementation

$H(n)$ depends linearly on its coefficients, and the number of coefficients is bounded by $m \times d$.
Select md values of n, for each value count the number of integer points using scanning methods, and solve a system of nd equations in nd unknowns.
The trick: the shape of $P(n)$ and hence the number and position of its vertices may depend on n. In each *chamber*, $H(n)$ may be a different polynomial.

Quantifier Elimination and SMT Solvers

- *Quantifier Elimination*: given a logical formula $\phi(x, y)$, find a formula $\psi(y) \equiv \exists x \phi(x, y)$. The x are the variables and y the parameters.
- *Satisfaction Modulo a Theory*: Given a formula $\phi(x)$, decide if there exists x_0 such that $\phi(x_0)$ or not. If so, answer SAT; x_0 is the witness; if not, answer UNSAT.

Both tools have similar uses and similar techniques, except that SMT solvers usually give much less information.
See also *Constraint Satisfaction Programming*: similar to SMT solvers, except that most variables have finite domain:

- more general types of constraints
- allow brute force search

Quantifier Elimination, How and Why

Quantifier Elimination has many other names: projection, variable elimination, etc...
Quantifier Elimination has many uses. Example: find the *footprint* of a array access in a loop. Let D be the iteration domain of the loop and f be the subscript function: the footprint is the set:

$$\{i | \exists x \in D : i = f(x)\}$$

Methods:

- many algorithms (Gaussian Elimination, Fourier-Motzkin, PIP and even Chernikova) already are QE algorithms for conjuctions
- since $\exists$ distributes over $\vee$, convert the given formula to Disjunctive Normal Form and do the elimination independently in each clause.

Feasibility Results

Quantifier elimination is not always possible. It depends on:

- the domain of x (integer, reals, finite, infinite)
- the "logic" of ϕ
- the "logic" of ψ
- if ϕ is affine in x or Boolean-affine:
  - if x is real or rational, always possible, easy, ψ is boolean-affine
  - if x is integral, always possible, difficult, ψ may need integer division or the modulo operator
- if ϕ is a polynomial
  - if x is real, possible (Tarski), difficult (Cylindrical Algebraic Decomposition)
  - if x is integer or rational, impossible (Hilbert 10th problem) but one can use heuristics
- if x is Boolean (or belongs to a finite set), trivial:

$$\exists \phi(x, y) = \phi(\text{true}, y) \vee \phi(\text{false}, y)$$

A Special Case: Farkas Lemma

Given A and b solve :

$$\exists c, d : \forall x \in \mathbb{R} : Ax + b \geq 0 \Rightarrow cx + d \geq 0$$

- doubly quantified
- non linear due to the terme c.x

Farkas says: If $Ax + b \geq 0$ is feasible, the the problem is equivalent to:

$$\exists \lambda \geq 0 : c = \lambda A \wedge d \geq \lambda b$$

If $Ax + b \geq 0$ is unfeasible, the problem is trivial.
If x is integral, one must replace the polyhedron $Ax + b \geq 0$ by its integer hull (the convex hull of its integer points) or some conservative approximation (cutting planes).

SMT Solvers

SAT Solvers:

- given a Boolean formula, find an assignments to its variables that gives it the value true, or prove that none exists
- no known polynomial algorithm, but the set of difficult problems is relatively small.

SMT Solvers:

- A theory is a set of formulas, closed by negation, and with a decision procedure for satisfiability. Example: the set of affine inequalities in the reals with the simplex.
- given a boolean formula whose atoms are in the underlying theory, find an assignment to its variable that gives it the value true, or prove that none exists
- Basic algorithm: convert the formula to DNF, then apply the theory decision procedure to each clause.

Symbolic Algorithms, How To

Many problems in compilation can be cast as optimization programs:

$$\begin{array}{ll} \min & f(x) \\ x \in & D \end{array}$$

This is difficult if the objective function, f, is complex, and almost impossible if f is given by algorithm.
Hence the importance of transforming a numerical algorithm in a symbolic one, which return a closed form instead of a number.
PIP is such a symbolic algorithm, its arguments are limited to linear parameters.

Symbolic Execution

- Select a symbolic representation for the values in the program
- This representation must be closed under the operations of the program and of bounded size
  - Example: linear form have a finite reprsentation and are closed under addition, multiplication by a constant and composition.
  - Polynomials are also closed under the same operations plus multiplication, but their size is not bounded, hence the complexity of the algorithm must be bounded.

The Problem of Tests

It is usually impossible to decide a test when the predicate is symbolic.

- Construct a problem tree, which branches whenever a test is encountered
- Record the outcome of the test on each branch
- Check that the tests of a branch are consistent, or close the branch
  - Update the predicate of the test after each assignment
  - Use an SMT solver

Hope for or prove convergence or enforce convergence by approximation.

And Now, Where To?

- Approximations: mating the polyhedral model and abstract interpretation.
- Dynamism: can one use the polyhedral model at run time?
- Artifical Intelligence: can one imitate Computer Algebra Systems, which store a huge amount of mathetical knowledge and use it on a pattern matching basis?
- Domain Specific Languages

The truth, as always, will be far stranger
A. C. Clarke

The End

THE END

Further Reading

Books

- A. Schrijver: Theory of Linear and Integer Programming, Wiley, 1982
- A. Barvinok: A Course in Convexity, Amer. Math. Soc. 2002
- M. Newman: Integer Matrices, Academic Press, 1972
- R. Smullyan: First Order Logic, Dover, 1995

Libraries

- *piplib* `www.piplib.org`
- *polylib* `icps.u-strasbg.fr/polylib`
- *isl* `repo.or.cz/w/isl.git`
- *qepcad* `www.usna.edu/CS/~qepcad/B/QEPCAD.html`
- *Z3* `z3.codeplex.com/`
- *Yices* `yices.csl.sri.com/`
- *CLooG* `www.cloog.org/`

A Non-Polyhedral Example

Given:

- An alphabet A,
- A grammar G (regular, context-free, ...) on A as terminals

The language generated by G is a subset of A^*:

$$\mathcal{L}(G) = \{w \in A^* | K \text{ accepts } w\}$$

This kind of sets is suitable for discussing recursive programs.