Fourier-Motzkin extension to Multivariate Polynomial Integer Constraints

Diogo Sampaio

December 12, 2017

2017 - 12 - 07	Diogo Sampaio	First version

Contents

1. Introduction

Quantifier elimination is the process of removing existential variables of a given first-order formula, obtaining one that is simpler in the number of variables, and that implies the original formula.

A very well known algorithm is the Fourier-Motzkin elimination process, that given a system (or formula) of inequalities removes one quantified variable by combining all of it's upper and lower bounds.

For example, let the system:

$$\exists x \mid 0 < y \wedge z < 1 \wedge y < x \wedge x < z$$

In such formula, x is a existential quantified variable, and y, z are parameters. Using FME to remove variable x from this formula requires to find it's upper and lower bounds. In this case we have the upper bound

$$x < z$$

and the lower bound

$$y < x.$$

For last x is eliminated by isolating x of each formula and combining lower with upper bounds such as:

$$y < x < z \Rightarrow y < z$$

generating the new system:

$$0 < y \wedge z < 1 \wedge y < z.$$

This algorithm is designed for linear systems, where all coefficients of the variable being eliminated are numeric values, and the inequality can be classified as either a upper or lower bound.

When dealing with polynomials, variable coefficients might be symbolic expressions. In such case, all possible signs of the coefficient (positive, negative, or zero) must be explored. e.g. imagine the system:

$$\exists x \mid a - 1 > 0 \wedge -b > 0 \wedge ax > 0 \wedge bx > 0$$

when eliminating the variable x it is required to classify the terms $ax > 0$ and $bx > 0$ as either upper or lower bounds of x. Not knowing the sign of the coefficients (a, b) requires to evaluate nine different combinations:

$$a > 0 \wedge b > 0$$

$$a > 0 \wedge b = 0$$

$$a > 0 \wedge b < 0$$

$$a = 0 \wedge b > 0$$

$$a = 0 \wedge b = 0$$

$$a = 0 \wedge b < 0$$

$$a < 0 \wedge b > 0$$

$$a < 0 \wedge b = 0$$

$$a < 0 \wedge b < 0$$

To avoid this branching we use an implementation of a theorem used in the positiveness test algorithm, proposed by Markus Schweighofer (https://doi.org/10.1016/S0022-4049(01)00041-X), to retrieve symbolic coefficient signs. Using such algorithm it is possible to infer that only the red assumption holds correct.

The same positiveness test algorithm is of major importance when resolving system over integer variables, instead of reals. It is used in many other techniques required to preserve the precision of the simplified formula, such as extending the normalization (https://doi.org/10.1145/125826.125848) technique to symbolic expressions, perform convex hull detection and remove redundant constraints.

Our C++ implementation uses GiNaC (https://www.ginac.de/) to manipulate symbolic expressions and GLPK to implement the positiveness test (https://www.gnu.org/software/glpk).

For further details please refer to *Profile Guided Hybrid Compilation* §2.7 and §5. (https://hal.archives-ouvertes.fr/tel-01428425)

2. Access to it

2.1 Code access and description

The code is publicly available for download at:
https://gitlab.inria.fr/nunessam/pghc.git
Source-code folders contents:

- doc: this document.
- FM: the FME and a stand-alone positiveness tester.
- converters: distinct converters to generate input to other quantifier-elimination tools, such as
  - QEPCAD: https://github.com/PetterS/qepcad
  - reduce/redlog: https://sourceforge.net/projects/reduce-algebra/

  as well as input to barvinok http://barvinok.gforge.inria.fr/ where it tries to discover if it is possible to prove if a system of constraints define a empty / absurd space. For such task it computes maximum and minimum values of polynomials inside a convex polyhedron space (see https://icps.u-strasbg.fr/upload/icps-2006-173.pdf).
- packages: Dockerfile for a painless process of install.

2.2 Compiling requirements

- C++11 compatible compiler
- GNU make, GiNaC, GLPK

OBS: These tools where tested solely in **Linux**.

3. How to use

3.1 Fourier-Motzkin elimination

3.1.1 Inputs

The application can either read a input text file or it can read from the standard input stream. Example files can be seen with extension `.in` inside the folder FM. The binary file it self is called `Simplifier`.

3.1.2 Input format

The input formula must be given in a disjunction of systems, where each system is a conjunction of terms. In a bottom up manner we have that:

- A *term* is a INTEGER VALUED equality or inequality, such as:

```
4*x^2 - 5*y < 0
x > 0
y <= s
N*h - 5*z == 0
```

- A system of terms declares all quantified variables, parameter such as the example in the file `FM/POL.in`:

```
[n]->[jj,j,k,kk,ii,i]: -kk^2+j+j^2-kk-2*ii+2*i = 0 &
k >= 0 & -1-k+n >= 0 &
i >= 0 & -1+k-i >= 0 &
-1+j-k >= 0 & -1-j+n >= 0 &
kk >= 0 & -1-kk+n >= 0 &
ii >= 0 & -1+kk-ii >= 0 &
-1+jj-kk >= 0 & -1-jj+n >= 0 &
-1-k+kk >= 0 & -1-ii+i >= 0 &
n >= 0;
```

The distinct parts of the system are:

  - A system starts with a list, comma separated, of parameters, e.g.: `[n]`, `[nn,n, z, b]`, `[]`.
  - An arrow `->` describing "mapping from".
  - A list of the quantified variables to be eliminated, comma separated e.g.: `[jj,j,k,kk,ii,i]`, `[a, bc,c]`, `[]`.
  - A list of polynomial terms, initialized by a colon(`:`) and separated by the logic and symbol (`&`) and terminated by a semi-colon (`;`).

- Multiple systems can be represented in a single input file, being interpreted as an `logic or` of each one of the systems:

```
[]->[x]: x < 1 & x > 0;
[x]->[a]: a < x & a > 3*x-4;
```

  is interpreted as $\{\exists x \in \mathbb{Z} | x < 1 \wedge x > 0\} \vee \{\exists a \in \mathbb{Z} | a < x \wedge 3x - 4 < a\}$.

3.2 Schweighofer Tester

In the same FM folder, the binary file `st` allows to interactively play with a single system of constraints. It can read from input files or iteratively add expressions to the system. It allows the user to:

- [A]ppend or [D]elete a constraint.
- [T]est if an expression is implied by the system.
- Obtain the [R]oot expressions of the system.
- [S]how the generated expression of the Schweighofer Tester.
- Show the Schweighofer Tester [m]onomials generated.
- [C]hange the degree (number of multiplications) the Schweighofer Tester multiply all system constraints against all others.