Computer Science at Kent

Computing Convex Hulls with a Linear Solver

Florence Benoy, Andy King and Fred Mesnard

Technical Report No: 11-02
Date: September 2002

Published by the Computing Laboratory,
University of Kent, Canterbury, Kent CT2 7NF, UK.

Abstract

A programming tactic involving polyhedra is reported that has been widely used in the analysis of logic programs. The method enables the convex hull operation to be coded with linear constraint solving machinery that is available in many Prolog systems.
Keywords: convex hull, polyhedra, abstract interpretation, linear constraints.

1 Introduction

Polyhedra have been widely applied in program analysis [4] particularly for reasoning about logic and constraint logic programs. In this context polyhedra have been used in binding-time analysis [14], cdr-coded list analysis [8], argument-size analysis [1], time-complexity analysis [9], high-precision groundness analysis [2], type analysis [13], termination checking [3] and termination inference [11, 6].

All these techniques use polyhedra to describe relevant properties of the program and manipulate polyhedra using operations that include projection, emptiness checking, inclusion testing for polyhedra, intersection of polyhedra (meet) and convex hull (join). The classic approach to polyhedral analysis uses two representations: (i) frames and rays and (ii) systems of (non-strict) linear inequalities and employs the Chernikova algorithm to convert between them [4]. The rationale for this dual representation is that convex hull can be computed straightforwardly with frames and rays whereas projection and intersection are simpler computed over systems of linear inequalities. However, converting from frames and rays (inequalities) to inequalities (frames and rays) is exponential in the number of frames and rays (inequalities). A simpler tactic that has been widely adopted in the analysis of logic programs [3, 6, 9, 11, 13], is to use only the linear inequality representation and compute convex hull by adapting [1] a relaxation technique [5]. This programming trick allows all the domain operations to be implemented straightforwardly using a standard simplex-based linear constraint solver (see for instance [10]). The next section outlines the method and the following section, an example implementation.

2 Method

Consider two arbitrary polyhedra, P_1 and P_2, represented in standard form:

$$P_1 = \{\vec{x} \in \mathbb{Q}^n \mid A_1\vec{x} \leq \vec{B}_1\} \qquad P_2 = \{\vec{x} \in \mathbb{Q}^n \mid A_2\vec{x} \leq \vec{B}_2\}$$

such that $P_1 \neq \emptyset$ and $P_2 \neq \emptyset$ so that the problem is non-trivial. Note that $A_i\vec{x} \leq \vec{B}_i$ are non-strict and therefore P_1 and P_2 are both closed. The problem in essence is to compute the smallest closed polyhedron that includes P_1 and P_2, that is, the closure of the convex hull of P_1 and P_2. The closure of the convex hull of two polyhedra can always be represented as a system of non-strict linear inequalities. The starting point for our construction is the convex hull of $P_1 \cup P_2$ that is given by:

$$P_H = \left\{ \vec{x} \in \mathbb{Q}^n \;\middle|\; \begin{array}{lllll} \vec{x} = \sigma_1\vec{x}_1 + \sigma_2\vec{x}_2 & \wedge & \sigma_1 + \sigma_2 = 1 & \wedge & 0 \leq \sigma_i \quad \wedge \\ A_1\vec{x}_1 \leq \vec{B}_1, & \wedge & A_2\vec{x}_2 \leq \vec{B}_2 & & \end{array} \right\}$$

Put $\vec{y}_1 = \sigma_1\vec{x}_1$ and $\vec{y}_2 = \sigma_2\vec{x}_2$ so that $\vec{x} = \vec{y}_1 + \vec{y}_2$ and $A_i\vec{y}_i \leq \sigma_i\vec{B}_i$ and define:

$$P_C = \left\{ \vec{x} \in \mathbb{Q}^n \;\middle|\; \begin{array}{lllll} \vec{x} = \vec{y}_1 + \vec{y}_2 & \wedge & \sigma_1 + \sigma_2 = 1 & \wedge & 0 \leq \sigma_i \quad \wedge \\ A_1\vec{y}_1 \leq \sigma_1\vec{B}_1 & \wedge & A_2\vec{y}_2 \leq \sigma_2\vec{B}_2 & & \end{array} \right\}$$

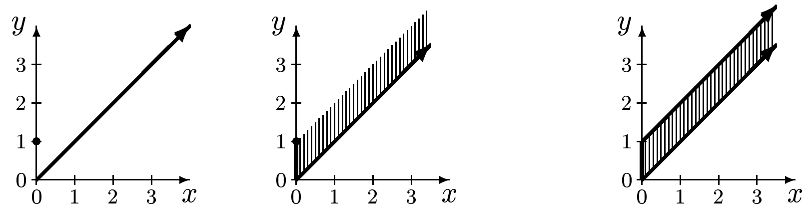
Observe that $P_H \subseteq P_C$. Note too that P_C is closed since the projection of a system of non-strict linear inequalities is closed. In fact the following proposition asserts that P_C coincides with the closure of the convex hull of $P_1 \cup P_2$.

Proposition 2.1 *P_C is the closure of the convex hull of P_1 and P_2.*

Proof 2.1 *Theorem 19.6 of [12] states that the closure of the convex hull of* $P_1 \cup P_2$ *is the set* $(0^+P_1 + P_2) \cup (P_1 + 0^+P_2) \cup (\cup\{\sigma_1 P_1 + \sigma_2 P_2 \mid \sigma_1 + \sigma_2 = 1 \wedge 0 < \sigma_1, \sigma_2\})$ *where* $0^+P_i = \{\vec{x} \in \mathbb{Q}^n \mid \vec{x} + P_i \subseteq P_i\}$. *However* $0^+P_i = \{\vec{y} \in \mathbb{Q}^n \mid A_i\vec{y} \leq \vec{0}\}$ *since* $P_i \neq \emptyset$ *[12][page 62].* P_C *is thus the closure of the convex hull.*

This result leads to an algorithm for computing the closure of the convex hull: construct the systems $A_i\vec{y}_i \leq \sigma_i\vec{B}_i$, add the constraints $\vec{x} = \vec{y}_1 + \vec{y}_2$, $\sigma_1 + \sigma_2 = 1$ and $0 \leq \sigma_i$, then eliminate variables other than $\vec{x}$ using projection to obtain P_C in terms of $\vec{x}$. Hence the closure of the convex hull can be computed without recourse to another representation. This is illustrated below.

Example 2.1 Consider the point $P_1 = \big\{\langle x, y\rangle \in \mathbb{Q}^2 \mid x = 0 \wedge y = 1\big\}$ and the half line $P_2 = \big\{\langle x, y\rangle \in \mathbb{Q}^2 \mid x = y \wedge 0 \leq x\big\}$. Observe that P_1 and P_2 are closed whereas the convex hull of $P_1 \cup P_2$ excludes the points $\big\{\langle x, y\rangle \in \mathbb{Q}^2 \mid x > 0 \wedge y = x + 1\big\}$ and hence is not closed (see the diagram below).



P_1 and P_2 convex hull of $P_1 \cup P_2$ closure of convex hull of $P_1 \cup P_2$

Constructing the systems $A_i\vec{y}_i \leq \sigma_i\vec{B}_i$ (with equalities as pairs of inequalities) and adding $\vec{x} = \vec{y}_1 + \vec{y}_2$, $\sigma_1 + \sigma_2 = 1$ and $0 \leq \sigma_i$ leads to the following system:

$$P_C = \left\{ \langle x, y\rangle \in \mathbb{Q}^2 \;\middle|\; \begin{array}{lllllll} x = x_1 + x_2 & \wedge & y = y_1 + y_2 & \wedge & \sigma_1 + \sigma_2 = 1 & \wedge \\ 0 \leq \sigma_1 & \wedge & 0 \leq \sigma_2 & \wedge & & \\ x_1 \leq 0 & \wedge & -x_1 \leq 0 & \wedge & & \\ y_1 \leq \sigma_1 & \wedge & -y_1 \leq -\sigma_1 & \wedge & & \\ x_2 - y_2 \leq 0 & \wedge & -x_2 + y_2 \leq 0 & \wedge & -x_2 \leq 0 & \end{array} \right\}$$

Eliminating the variables x_i, y_i and σ_i leads to the solution:

$$P_C = \{\langle x, y\rangle \in \mathbb{Q}^2 \mid 0 \leq x \wedge x \leq y \wedge y \leq x + 1\}$$

Note that calculating P_C without the inequalities $0 \leq \sigma_1$ and $0 \leq \sigma_2$ – the relaxation advocated in [5] for computing convex hull – gives $\{\langle x, y\rangle \in \mathbb{Q}^2 \mid 0 \leq x\}$ which is incorrect.

3 Implementation

This section shows how closure of the convex hull can be implemented elegantly using a linear solver in particular the CLP($\mathbb{Q}$) library [7]. Closed polyhedra will be represented by lists (conjunctions) of atomic constraints of the form $c ::= e \leq e \mid e = e \mid e \geq e$ where expressions take the form $e ::= x \mid n \mid n * x \mid -e \mid e + e \mid e - e$ and n is rational and x is a variable. If C is a list of constraints $[c_1, \ldots, c_m]$ and X is a list of variables $[x_1, \ldots, x_n]$, then let $[\![C]\!]_X$ denote the polyhedron $\{\langle y_1, \ldots, y_n\rangle \in \mathbb{Q}^n \mid (\wedge_{i=1}^n x_i = y_i) \models_{\mathbb{Q}} (\wedge_{i=1}^m c_i)\}$. The specification for the main predicate `convex_hull(Xs, Cxs, Ys, Cys, Zs, Czs)`, and then its code, is given below where $vars(t)$ denotes the set of variables occurring in a term t.

preconditions:

1. `Xs` is a closed list with distinct variables as elements and likewise for `Ys`,
2. `Xs` and `Ys` are the same length,
3. `Cxs` and `Cys` are closed lists of linear constraints,
4. `Cxs` and `Cys` are both satisfiable,
5. $vars(\texttt{Cxs}) \subseteq vars(\texttt{Xs})$ and $vars(\texttt{Cys}) \subseteq vars(\texttt{Ys})$.

postconditions:

1. `Zs` is a closed list with distinct variables as elements,
2. $vars(\texttt{Czs}) \subseteq vars(\texttt{Zs})$ and $(vars(\texttt{Xs}) \cup vars(\texttt{Ys})) \cap vars(\texttt{Zs}) = \emptyset$,
3. $[\![\texttt{Czs}]\!]_{\texttt{Zs}}$ is the closure of the convex hull of $[\![\texttt{Cxs}]\!]_{\texttt{Xs}}$ and $[\![\texttt{Cys}]\!]_{\texttt{Ys}}$,
4. Let CS_1 (resp. CS_2) be the numeric constraint store before (resp. after) the call to `convex_hull`. Then $[\![CS_1]\!]_{vars(CS_1)} = [\![CS_2]\!]_{vars(CS_1)}$.

```
:- use_module(library(lists), [append/3]).
:- use_module(library(clpq)).

convex_hull(Xs, Cxs, Ys, Cys, Zs, Czs) :-
    copy_term(Xs-Cxs, Copy_Xs-Copy_Cxs),
    copy_term(Ys-Cys, Copy_Ys-Copy_Cys),
    scale(Copy_Cxs, Sig1, [], C1s), scale(Copy_Cys, Sig2, C1s, C2s),
    add_vect(Copy_Xs, Copy_Ys, Us, C2s, C3s),
    project(Us, [Sig1>=0, Sig2>=0, Sig1+Sig2=1|C3s], Zs, Czs).

scale([], _, Cs, Cs).
scale([C1|C1s], Sig, C2s, C3s) :-
    C1 =.. [RelOp, A1, B1], C2 =.. [RelOp, A2, B2],
    mul_exp(A1, Sig, A2), mul_exp(B1, Sig, B2),
    scale(C1s, Sig, [C2|C2s], C3s).

mul_exp(E1,Sigma,E2)  :- once(mulexp(E1,Sigma,E2)).
mulexp(X, _, X)       :- var(X).
mulexp(N, Sig, N*Sig) :- ground(N).
mulexp(N*X, _, N*X)   :- ground(N), var(X).
mulexp(-X, Sig, -Y)   :- mulexp(X, Sig, Y).
mulexp(A+B, Sig, C+D) :- mulexp(A, Sig, C), mulexp(B, Sig, D).
mulexp(A-B, Sig, C-D) :- mulexp(A, Sig, C), mulexp(B, Sig, D).

add_vect([], [], [], Cs, Cs).
add_vect([X|Xs], [Y|Ys], [Z|Zs], C1s, C2s) :-
    add_vect(Xs, Ys, Zs, [X+Y=Z|C1s], C2s).

project(Xs, Cs, Ys, Es) :-
    tell_cs(Cs), prepare_dump(Xs, Ys, X1s, Y1s, Eqns),
    dump(X1s, Y1s, Ds),
    append(Eqns, Ds, Es).
```

```
prepare_dump([], [], [], [], []).
prepare_dump([X|Xs], [Y|Ys], [X|X1s], [Y|Y1s], Cs) :-
    var(X), prepare_dump(Xs, Ys, X1s, Y1s, Cs).
prepare_dump([X|Xs], [Y|Ys], X1s, Y1s, [Y=X|Cs]) :-
    ground(X), prepare_dump(Xs, Ys, X1s, Y1s, Cs).

tell_cs([]).
tell_cs([C|Cs]) :-  {C}, tell_cs(Cs).
```

Note that `scale` finesses the problem of putting `Copy_Cxs` and `Copy_Cys` into the standard form $A_i\vec{y_i} \leq \vec{B}_i$ before applying scaling. For example, `X + 2 >= 1 + Y` in standard form is `Y - X =< 1` but scaling constants on both sides of the relational operator preserves equivalence in that `X + 2*Sig >= 1*Sig + Y` is equivalent to `Y - X =< 1*Sig`. The calls to `copy_term` generate the lists `Copy_Xs` and `Copy_Ys` which correspond to the vectors of fresh variables $\vec{y_1}$ and $\vec{y_2}$ in the definition of P_C. Renaming also ensures that `[Sig1>=0, Sig2>=0, Sig1+Sig2=1|C3s]` is variable disjoint with the store and hence cannot inadvertently interact with the store. The call `dump(X1s, Y1s, Ds)` will project the constraint store onto the variables `X1s` to obtain the list of atomic constraints `Ds` renamed so that each variable of `X1s` is replaced with a fresh variable in `Y1s`. The pre-processing predicate `prepare_dump` ensures that `dump(X1s, Y1s, Ds)` is always called with `X1s` bound to a list of free variables even when elements of `Xs` are instantiated to numbers.

Example 3.1 *Running this code on the data of Example 2.1 gives:*

```
| ?- convex_hull([X1,Y1],[X1=0,Y1=1],[X2,Y2],[X2>=0,Y2=X2],[X,Y],S).
S = [X>=0,X-Y>= -1,X-Y=<0] ? ;
no
```

Note that by simply changing `library(clpq))` to `library(clpr))`, the closure of the convex hull can be computed with floating point numbers:

```
| ?- convex_hull([X1,Y1],[X1=0,Y1=1],[X2,Y2],[X2>=0,Y2=X2],[X,Y],S).
S = [X>=-0.0,X-Y>= -1.0,X-Y=<-0.0] ? ;
no
```

Finally, we now briefly discuss some properties of the above program. Termination follows from semantics of `once`, termination of the underlying solver, and preconditions 1 and 3. For correctness, let us assume that all the preconditions hold. Postconditions 1, 2, and 4 follow from the use of `copy_term` and `dump`. Postcondition 3 relies on the correctness of the method itself and that of the solver.

4 Conclusions

We have presented a Prolog program for computing convex hulls using linear solver machinery. As Holzbaur's library is also available for CIAO Prolog, ECLiPSe Prolog, XSB Prolog and Yap Prolog, the technique can be easily adapted to these systems. The method is an attractive compromise between conciseness, clarity and efficiency and indeed variants of this program have now been deployed in a variety of analysers [3, 6, 9, 11, 13].

References

[1] F. Benoy and A. King. Inferring Argument Size Relationships with CLP($\mathcal{R}$). In *Logic-based Program Synthesis and Transformation (Selected Papers)*, volume 1207 of *Lecture Notes in Computer Science*, pages 204–223. Springer-Verlag, 1997.

[2] M. Codish, S. Genaim, H. Sondergaard, and P. Stuckey. Higher-Precision Groundness Analysis. In *International Conference on Logic Programming*, volume 2237 of *Lecture Notes in Computer Science*, pages 135–149. Springer-Verlag, 2001.

[3] M. Codish and C. Taboch. A Semantic Basis for the Termination Analysis of Logic Programs. *The Journal of Logic Programming*, 41(1):103–123, 1999.

[4] P. Cousot and N. Halbwachs. Automatic Discovery of Linear Restraints among Variables of a Program. In *Principles of Programming Languages*, pages 84–97. ACM Press, 1978.

[5] B. De Backer and H. Beringer. A CLP language handling disjunctions of linear constraints. In *International Conference on Logic Programming*, pages 550–563. MIT Press, 1993.

[6] S. Genaim and M. Codish. Inferring Termination Conditions for Logic Programs using Backwards Analysis. In *International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 2250 of *Lecture Notes in Artificial Intelligence*, pages 681–690. Springer-Verlag, 2001.

[7] C. Holzbaur. OFAI clp(Q,R) Manual. Technical Report TR-95-09, Austrian Research Institute for Artificial Intelligence (ÖFAI), Schottengasse 3, A-1010 Vienna, Austria, 1995.

[8] R. N. Horspool. Analyzing List Usage in Prolog, March 1990. University of Victoria.

[9] A. King, K. Shen, and F. Benoy. Lower-bound Time-Complexity Analysis of Logic Programs. In *International Symposium on Logic Programming*, pages 261–276. MIT Press, 1997.

[10] K. Marriott and P. Stuckey. *Programming with Constraints*. MIT Press, 1998.

[11] F. Mesnard and U. Neumerkel. Applying Static Analysis Techniques for Inferring Termination Conditions of Logic Programs. In *Static Analysis Symposium*, volume 2126 of *Lecture Notes in Computer Science*, pages 93–110. Springer-Verlag, 2001.

[12] R. T. Rockafellar. *Convex Analysis*. Princeton University Press, 1970.

[13] H. Săglam and J. Gallagher. Constrained Regular Approximation of Logic Programs. In *Logic Programming Synthesis and Transformation (Selected Papers)*, pages 282–299. Springer-Verlag, 1997.

[14] W. Vanhoof and M. Bruynooghe. Binding-Time Annotations Without Binding-Time Analysis. In *International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 2250 of *Lecture Notes in Artificial Intelligence*, pages 707–722. Springer-Verlag, 2001.