PROGRAMMING PEARL

Computing Convex Hulls with a Linear Solver

FLORENCE BENOY and ANDY KING

Computing Laboratory, University of Kent, UK.
email: {p.m.benoys, a.m.king}@kent.ac.uk

FRED MESNARD
Iremia, Université de La Réunion, France.
email: fred@univ-reunion.fr

Abstract

A programming tactic involving polyhedra is reported that has been widely applied in the polyhedral analysis of (constraint) logic programs. The method enables the computations of convex hulls that are required for polyhedral analysis 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 (Cousot & Halbwachs, 1978) particularly for reasoning about logic and constraint logic programs. In this context polyhedra have been used in binding-time analysis (Vanhoof & Bruynooghe, 2001), cdr-coded list analysis (Horspool, 1990), argument-size analysis (Benoys & King, 1990), time-complexity analysis (King et al., 1997), high-precision groundness analysis (Codish et al., 2001), type analysis (Sağılam & Gallagher, 1997), termination checking (Codish & Taboch, 1999) and termination inference (Mesnard & Neumerkel, 2001; Génaim & Codish, 2001).

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 the convex hull (join). The classic approach to polyhedral analysis (Cousot & Halbwachs, 1978) uses two representations: (i) frames and rays and (ii) systems of (non-strict) linear inequalities and employs the Chernikova algorithm to convert between them (Le Verge, 1992). The rationale for this dual representation is that the convex hull can be computed straightforwardly with frames and rays whereas intersection is more simply computed over systems of linear inequalities. A simpler tactic that has been widely adopted in the analysis of logic programs is to use only the linear inequality representation and compute the convex hull by adapting (Benoy &
a relaxation technique proposed in (De Bocker & Beringer, 1993). The
elegance of this approach is that it enables the convex hull to be computed without
recourse to a dual representation: the problem is recast as a projection problem that
can be subcontracted to standard linear constraint solving machinery with mini-
mal coding effort. Moreover, the performance is acceptable for many applications.
In fact this technique has been widely applied in the analysis of logic programs
(Codish & Taboch, 1999; Genaim & Codish, 2001; King et al., 1997; Mesnard &
Neumerkel, 2001; Saglam & Gallagher, 1997). The next section outlines the method
and the following section, an example implementation. The final section presents
the concluding discussion.

2 Method

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

$$P_1 = \{ \bar{x} \in \mathbb{Q}^n \mid A_1 \bar{x} \leq \bar{b}_1 \} \quad P_2 = \{ \bar{x} \in \mathbb{Q}^n \mid A_2 \bar{x} \leq \bar{b}_2 \}$$

such that $P_1 \neq \emptyset$ and $P_2 \neq \emptyset$ so that the problem is non-trivial. Note that $A_i \bar{x} \leq \bar{b}_i$ are
non-strict and therefore $P_1$ and $P_2$ are both closed. The problem in essence is to
compute the smallest polyhedron that includes $P_1$ and $P_2$. Interestingly, the convex
hull of $P_1 \cup P_2$ is not necessarily closed as is illustrated in the following example.

Example 2.1

Consider the 2-dimensional polyhedra $P_1$ and $P_2$ defined by:

$$P_1 = \left\{ \bar{x} \in \mathbb{Q}^2 \left[ \begin{array}{cc} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{array} \right] \bar{x} \leq \left[ \begin{array}{c} 0 \\ 0 \\ 1 \\ -1 \end{array} \right] \right\} \quad P_2 = \left\{ \bar{x} \in \mathbb{Q}^2 \left[ \begin{array}{cc} -1 & -1 \\ -1 & 0 \end{array} \right] \bar{x} \leq \left[ \begin{array}{c} 0 \\ 0 \end{array} \right] \right\}$$

Observe that $P_1 = \{ (0,1) \}$ is a point whereas $P_2 = \{ (x,y) \in \mathbb{Q}^2 \mid x = y \land 0 \leq x \}$ is
a half-line. Note too that $P_1$ and $P_2$ are closed whereas the convex hull of $P_1 \cup P_2$
excludes the points $\{ (x,y) \in \mathbb{Q}^2 \mid x > 0 \land y = x + 1 \}$ and hence is not closed (see
the diagram below).

Since the convex hull of $P_1 \cup P_2$ is not necessarily closed, the convex hull cannot
always be represented by a system of non-strict linear inequalities; in order to
overcome this problem, the closure of the convex hull of $P_1 \cup P_2$ is computed. The
starting point for our construction is the convex hull of $P_1 \cup P_2$ that is given by:

$$P_H = \left\{ \bar{x} \in \mathbb{Q}^n \left| \begin{array}{c} \bar{x} = \sigma_1 \bar{x}_1 + \sigma_2 \bar{x}_2 \\
A_1 \bar{x}_1 \leq \bar{b}_1 \\
A_2 \bar{x}_2 \leq \bar{b}_2 \\
0 \leq \sigma_1 \\
0 \leq \sigma_2 \end{array} \right. \quad \sigma_1 + \sigma_2 = 1 \right\}$$
To avoid the non-linearity $\bar{x} = \sigma_1 \bar{x}_1 + \sigma_2 \bar{x}_2$ the system can be reformulated (relaxed) by putting $\bar{y}_1 = \sigma_1 \bar{y}_1$ and $\bar{y}_2 = \sigma_2 \bar{y}_2$ so that $\bar{x} = \bar{y}_1 + \bar{y}_2$ and $A_1 \bar{y}_1 \leq \sigma_1 \bar{B}_1$ and $A_2 \bar{y}_2 \leq \sigma_2 \bar{B}_2$ to define:

$$P_{CH} = \left\{ \bar{x} \in \mathbb{R}^n \mid \bar{x} = \bar{y}_1 + \bar{y}_2 \wedge \sigma_1 + \sigma_2 = 1 \wedge 0 \leq \sigma_1 \wedge A_1 \bar{y}_1 \leq \sigma_1 \bar{B}_1 \wedge A_2 \bar{y}_2 \leq \sigma_2 \bar{B}_2 \wedge 0 \leq \sigma_2 \right\}$$

Observe that $P_H \subseteq P_{CH}$. Moreover, unlike $P_H$, $P_{CH}$ is expressed in terms of a system of linear inequalities. Note too that $P_{CH}$ is closed since the projection of a system of non-strict linear inequalities is closed. In fact the following proposition asserts that $P_{CH}$ coincides with the closure of the convex hull of $P_1 \cup P_2$.

**Proposition 2.1**

$P_{CH}$ is the closure of the convex hull of $P_1$ and $P_2$.

The proof uses the concept of a recession cone. The recession cone of a polyhedron $P$, denoted $0^+P$, is defined by $0^+P = \{ y \in \mathbb{R}^n \mid \forall \lambda \geq 0, \forall \bar{x} \in P, y + \lambda \bar{x} \in P \}$. The intuition is that $0^+P$ includes a vector $y$ whenever $P$ includes all the half-lines in the direction of $y$ that start in $P$.

**Proof**

Suppose $P_2 = \{ \bar{x} \in \mathbb{R}^n \mid A_2 \bar{x} \leq \bar{B}_2 \}$. Theorem 19.6 of (Rockafellar, 1970) states that the closure of the convex hull of $P_1 \cup P_2$ is the set $(0^+P_1 + P_2) \cup (P_1 + 0^+P_2) \cup (\cup \sigma_1 P_1 + \sigma_2 P_2 \mid \sigma_1 + \sigma_2 = 1 \wedge 0 < \sigma_1, \sigma_2 \}$. Intuitively, $0^+P_1 + P_2$ is $P_2$ extended in the directions of half-lines contained within $P_1$. Let $\bar{x} \in P_1$, then $\bar{y} \in 0^+P_1$ if and only if $A_2(\bar{x} + \lambda \bar{y}) \leq \bar{B}_2$ for all $\lambda \geq 0$ which holds if and only if $A_2 \bar{y} \leq 0$ (Rockafellar, 1970) (pp 62). Therefore $0^+P_1 + P_2 = \{ \bar{x} \in \mathbb{R}^n \mid \bar{x} = \bar{y}_1 + \bar{y}_2 \wedge A_1 \bar{y}_1 \leq \bar{B}_1 \wedge A_2 \bar{y}_2 \leq \bar{B}_2 \}$ and similarly $P_1 + 0^+P_2 = \{ \bar{x} \in \mathbb{R}^n \mid \bar{x} = \bar{y}_1 + \bar{y}_2 \wedge A_1 \bar{y}_1 \leq \bar{B}_1 \wedge A_2 \bar{y}_2 \leq \bar{B}_2 \}$. Furthermore, $\cup \sigma_1 P_1 + \sigma_2 P_2 \mid \sigma_1 + \sigma_2 = 1 \wedge 0 < \sigma_1, \sigma_2 \} = \{ \bar{x} \in \mathbb{R}^n \mid \sigma_1 + \sigma_2 = 1 \wedge 0 < \sigma_1, \sigma_2 \wedge \bar{x} = \bar{y}_1 + \bar{y}_2 \wedge A_1 \bar{y}_1 \leq \sigma_1 \bar{B}_1 \wedge A_2 \bar{y}_2 \leq \sigma_2 \bar{B}_2 \}$. Observe that $\{ \bar{x} \in \mathbb{R}^n \mid \bar{x} = \bar{y}_1 + \bar{y}_2 \wedge A_1 \bar{y}_1 \leq \sigma_1 \bar{B}_1 \wedge A_2 \bar{y}_2 \leq \sigma_2 \bar{B}_2 \}$ coincides with the sets (i) $0^+P_1 + P_2$, (ii) $P_1 + 0^+P_2$, and (iii) $\cup \sigma_1 P_1 + \sigma_2 P_2 \mid \sigma_1 + \sigma_2 = 1 \wedge 0 < \sigma_1, \sigma_2 \}$. Hence the closure of the convex hull can be computed without recourse to another representation. Therefore $P_{CH}$ is the closure of the convex hull. □

This result leads to an algorithm for computing the closure of the convex hull:

- Construct the systems $A_1 \bar{y}_1 \leq \sigma_1 \bar{B}_1$ by scaling the constant vectors $\bar{B}_1$; by $\sigma_1$, add the constraints $\bar{x} = \bar{y}_1 + \bar{y}_2$, $\sigma_1 + \sigma_2 = 1$ and $0 \leq \sigma_1$, then eliminate variables other than $\bar{x}$ using projection to obtain $P_{CH}$ in terms of $\bar{x}$. Hence the closure of the convex hull can be computed without recourse to another representation. This is illustrated below.

**Example 2.2**

Returning to example 2.1, consider the systems $A_i \bar{x} \leq \bar{B}_i$:

$$P_1 = \left\{ (x, y) \in \mathbb{R}^2 \mid x \leq 0 \wedge -x \leq 0 \wedge y \leq 1 \wedge -y \leq -1 \right\} \quad P_2 = \left\{ (x, y) \in \mathbb{R}^2 \mid x - y \leq 0 \wedge -x + y \leq 0 \wedge -x \leq 0 \right\}$$
Adding \( \bar{x} = y_1 + y_2, \sigma_1 + \sigma_2 = 1 \) and \( 0 \leq \sigma_i \) leads to the following system:

\[
P_{CH} = \{ \langle x, y \rangle \in \mathbb{Q}^2 \mid x = x_1 + x_2 \land y = y_1 + y_2 \land 0 \leq \sigma_1 \land 0 \leq \sigma_2 \land x_1 \leq 0 \land -x_1 \leq 0 \land y_1 \leq \sigma_1 \land -y_1 \leq -\sigma_1 \land x_2 - y_2 \leq 0 \land -x_2 + y_2 \leq 0 \land -x_2 \leq 0 \}
\]

Eliminating the variables \( x_i, y_i \) and \( \sigma_i \) leads to the solution:

\[
P_{CH} = \{ \langle x, y \rangle \in \mathbb{Q}^2 \mid 0 \leq x \land x \leq y \land y \leq x + 1 \}
\]

Theorem 19.6 of (Rockafellar, 1970), which is used in the proof, asserts that \( P_{CH} \) includes \( P_1 + \alpha P_2 = P_1 + P_2 = \{ \langle x, y \rangle \in \mathbb{Q}^2 \mid x \geq 0 \land y = x + 1 \} \) and therefore includes the points \( \{ \langle x, y \rangle \in \mathbb{Q}^2 \mid x > 0 \land y = x + 1 \} \), and hence ensures closure. Note that calculating \( P_{CH} \) without the inequalities \( 0 \leq \sigma_1 \) and \( 0 \leq \sigma_2 \) – the relaxation advocated in (De Baecker & Beringer, 1993) 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 (Holzbaumer, 1995). The behaviour of a predicate is described with the aid of modes, that is, + indicates an argument that should be instantiated to a non-variable term when the predicate is called; - indicates an argument that should be uninstantiated; and ? indicates an argument that may or may not be instantiated (De Raes et al., 1996).

3.1 Closed Polyhedra

Closed polyhedra will be represented by lists (conjunctions) of linear constraints of the form \( c ::= e \leq e \mid e = e \mid e \geq e \) where expressions take the form \( e ::= x \mid n \mid n \times x \mid -e \mid e + e \mid e - e \) and \( n \) is a rational number and \( x \) is a variable. A convenient representation for a closed polyhedron is a (non-ground) list of constraints. This representation is interpreted with respect to a totally ordered (finite) set of variables. The ordering governs the mapping of each variable to its specific dimension. In practice, the ordering on variables is itself represented by the position of each variable within a list. Specifically, if \( C \) is a list of linear constraints \( [c_1, \ldots, c_m] \) and \( X \) is a list of variables \( [x_1, \ldots, x_n] \), then the represented polyhedron is \( P_{C,X} = \{ \langle y_1, \ldots, y_n \rangle \in \mathbb{Q}^n \mid (\land_{i=1}^m c_i = y_i) \land (\land_{j=1}^n x_i = y_i) \land (\land_{j=1}^n c_j) \} \). Note that although the order of variables in \( X \) is significant, the order of the constraints in \( C \) is not. Finally, let \( \text{vars}(o) \) denote the set of variables occurring in the syntactic object \( o \).

Example 3.1

The polyhedron \( P_1 \) from example 2.2 can be represented by the lists \( C_1 = [x = 0, y = 1] \) and \( X = [x, y] \), that is, \( P_1 = P_{C_1,X} \). Moreover, \( P_2 = P_{C_2,X} \) where \( C_2 = [x = y, x \geq 0] \) or alternatively \( C_2 = [y + z \geq x, x \geq y + 2 \times z, y \geq 0, z \geq 0] \).
Hence the dimension of $P_{C,X}$ is defined by the length of the list $X$ rather than the number of variables in $C$.

### 3.2 Projection

Projection is central to computing the convex hull. The desire, therefore, is to construct a predicate $\text{project}(+\text{Xs}, +\text{Cxs}, -\text{ProjectCxs})$ that is true when for a given list of dimensions $\text{Xs}$ and a given list of constraints $\text{Cxs}$, $\text{ProjectCxs}$ is the projection of $\text{Cxs}$ onto $\text{Xs}$. The specification of such a predicate is given below.

**preconditions:**
- $\text{Xs}$ is a closed list with distinct variables as elements,
- $\text{Cxs}$ is a closed list of linear constraints,
- $\text{Cxs}$ is satisfiable.

**postconditions:**
- $\text{Xs}$ is a closed list with distinct variables as elements,
- $\text{ProjectCxs}$ is a closed list of linear constraints,
- $\text{vars}(\text{ProjectCxs}) \subseteq \text{vars}(\text{Xs}),$
- $P_{\text{Cxs, Xs}} = P_{\text{ProjectCxs, Xs}}.$

Such a predicate can be constructed by adding the given constraints to the store and then invoking the projection facility provided in the CLP(Q) library, that is, the predicate $\text{dump}(+\text{Target}, -\text{NewVars}, -\text{CodedAnswer})$ (Holzbaur, 1995). Quoting from the manual: "[dump] reflects the constraints on the target variables into a term, where Target and NewVars are lists of variables of equal length and CodedAnswer is the term representation of the projection of constraints onto the target variables where the target variables are replaced by the corresponding variables from NewVars". This leads to the following implementation of $\text{project}$:

```prolog
:- use_module(library(clpq)).

project(Xs, Cxs, ProjectCxs) :-
  tell_cs(Cxs),
  dump(Xs, Vs, ProjectCxs), Xs = Vs.

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

**Example 3.2**

For example, the query $\text{project}([X, Z], [X < Y, Y < Z], \text{ProjectCxs})$ will correctly bind $\text{Cxs}$ to $[X-Z<0]$. However, correctness of this predicate is compromised by existing constraints in the store. For instance, the compound query $\{X = Z + 1\}$, $\text{project}([X, Z], [X < Y, Y < Z], \text{ProjectCxs})$ will fail because constraints posted within $\text{tell_cs}$ interact with those already in the store.

To insulate the constraints posted in $\text{tell_cs}$, both the variables $\text{Xs}$ and the constraints $\text{Cxs}$ need to be renamed. Renaming is trivial with the built-in $\text{copy_term}$
but care must be taken to ensure that Xs and Cxs are renamed consistently, that
is that variable sharing in Xs and Cxs is preserved in the copies. However, in SIC-
Stus Prolog copy_term(Term, Cpy) copies any constraints in the store that in-
volve variables in Term. For example, the query \{X = Y\}, copy_term(X = Y + 1, Cpy)
will bind Cpy to \_A = \_B + 1 where \_A and \_B are fresh variables. It will also copy
the constraint \(X = Y\) by posting the new constraint \_A = \_B to the store. To nu-
lify this effect, copy_term is called within the scope of call_residue. The call
call_residue(copy_term(X = Y + 1, Cpy), Residues) residuates any new constraint
into Residues instead of posting it to the store, thereby copying the term without
copying any constraint. Whether residuation is required depends on the particular
Prolog system. This leads to the following (SICStus Prolog specific) revision:

\[
\text{project}(Xs, Cxs, ProjectCxs) :-
\text{call_residue}(\text{copy_term}(Xs = Cxs, CpyXs = CpyCxs), \_),
\text{tell_cs}(CpyCxs),
\text{dump}(CpyXs, Vs, ProjectCxs), Xs = Vs.
\]

\textit{Example 3.3}

Using this revision, the query \{X = Z + 1\}, project([X, Z], [X < Y, Y < Z],
ProjectCxs) will succeed binding ProjectCxs to \([X<Z\)]). However, adding \(Z = 5\)
to the list of constraints induces an error. The problem is that posting the constraints
binds \(Z\) to \(5\) so that \text{dump} is called with its first argument instantiated to a list that
contains a non-variable term.

A pre-processing predicate \text{prepare_dump} is therefore introduced to ensure that
\text{dump} is called correctly. The following revision to \text{project}, in effect, extends the
facility provided by \text{dump} to capture constraints over both uninstantiated and in-
stantiated variables:

\[
\text{project}(Xs, Cxs, ProjectCxs) :-
\text{call_residue}(\text{copy_term}(Xs = Cxs, CpyXs = CpyCxs), \_),
\text{tell_cs}(CpyCxs),
\text{prepare_dump}(CpyXs, Xs, Zs, DumpCxs, ProjectCxs),
\text{dump}(Zs, Vs, DumpCxs), Xs = Vs.
\]

\text{prepare_dump}([], [], [], Cs, Cs).
\text{prepare_dump}([X | Xs], YsIn, ZsOut, CsIn, CsOut) :-
(ground(X) ->
  YsIn = [Y | Ys],
  ZsOut = [I | Zs],
  CsOut = [Y = X | Cs]
);
  YsIn = [I | Ys],
  ZsOut = [X | Zs],
  CsOut = Cs
),
\text{prepare_dump}(Xs, Ys, Zs, CsIn, Cs).
The literal `prepare_dumps(+Xs, +Ys, -Zs, ?CsIn, -CsOut)` is true for a given list Xs which contains either variables or numbers (or a mixture of the two) and a given list Ys which contains only variables, if

- Zs is the list obtained by substituting the non-variable terms of Xs with fresh variables and
- CsOut is an open ended list of equality constraints with CsIn at its end that contains one equality constraint for each number in Xs. Each constraint equates a numeric element of Xs with the element of Ys that is in the same list position.

The call `prepare_dumps([X1, 1, X3, 2], [A, B, C, D], Zs, CsIn, CsOut),` for instance, will bind Zs to `[X1, A, X3, B]` and CsOut to `[B=1, D=2|CsIn]`. The predicate ensures that `dumps` is called with its first argument bound to a list of free variables even when the list Xs includes numbers. In the CLP(Q) library, numbers coincide with rationals which are represented as compound (ground) terms of the form `rat(n, d)` where n and d are integers. The `ground(X)` test effectively checks whether X is instantiated to a number; the test `number(X)` is inappropriate since it would always fail.

Example 3.4
Consider again example 3.1. The second representation of \( P_2 \) can be simplified by using projection as follows:

\[
\begin{align*}
| \text{?- Cs} &= [Y+Z=X, X=>Y+2*Z, Y=>0, Z=>0], \text{project}([X, Y], \text{Cs}, \text{ProjectCs}). \\
\text{ProjectCs} &= [Y=>0, X=Y] \ ? ; \\
\text{no}
\end{align*}
\]

The system Cs is expressed over 3 variables and therefore defines a 3 dimensional space. Intuitively, the projection onto \([X, Y]\) is the shadow cast by \( P_{Cs\{x,y,z\}} \) onto the 2 dimensional space over X and Y. The projection ProjectCs in fact defines a half-line confined to the first quadrant since, by rearranging Cs, it follows that \( P_{Cs\{x,y,z\}} = \{(x,y,z) \in \mathbb{Q}^{3} \mid x = y \land 0 \leq y \land z = 0\} \).

### 3.3 Convex Hull

The specification for the main predicate `convex_hull(+Xs, +Cxs, +Ys, +Cys, -Zs, -Czs),` and then its code, is given below.

**preconditions:**
- Xs is a closed list with distinct variables as elements and likewise for Ys,
- Xs and Ys have the same length,
- \( \text{vars}(Xs) \cap \text{vars}(Ys) = \emptyset \),
- Cxs and Cys are closed lists of linear constraints,
- Cxs and Cys are both satisfiable,
- \( \text{vars}(Cxs) \subseteq \text{vars}(Xs) \) and \( \text{vars}(Cys) \subseteq \text{vars}(Ys) \).
postconditions:
- \(Xs, Ys\) and \(Zs\) are closed lists with distinct variables as elements,
- \(Zs\) is the same length as both \(Xs\) and \(Ys\),
- \(Czs\) is a closed list of linear constraints,
- \(\text{vars}(Czs) \subseteq \text{vars}(Zs)\) and \((\text{vars}(Xs) \cup \text{vars}(Ys)) \cap \text{vars}(Zs) = \emptyset\)
- \(P_{\text{convex hull}}\) is the closure of the convex hull of \(P_{\text{convex hull}}\).

\[
\text{convex hull}(Xs, Czs, Ys, Cys, Zs, Czs) :- \\
\text{scale}(Czs, Sig1, [], C1s), \\
\text{scale}(Cys, Sig2, C1s, C2s), \\
\text{add_vect}(Xs, Ys, Zs, C2s, C3s), \\
\text{project}(Zs, [\text{Sig}1 > 0, \text{Sig}2 > 0, \text{Sig}1 + \text{Sig}2 = 1|C3s], Czs).
\]

\[
\text{scale}([], _, Cs, Cs).
\]

\[
\text{scale}([C1|C1s], Sig, C2s, C3s) :- \\
\text{C1} = [\text{RelOp}, A1, B1], \\
\text{C2} = [\text{RelOp}, A2, B2], \\
\text{mul_exp}(A1, Sig, A2), \\
\text{mul_exp}(B1, Sig, B2), \\
\text{scale}(C1s, Sig, [\text{C2}|C2s], C3s).
\]

\[
\text{mul_exp}(E1, Sigma, E2) :- \text{once}(\text{mul_exp}(E1, Sigma, E2)).
\]

\[
\text{mul_exp}(X, _, X) :- \text{var}(X).
\]

\[
\text{mul_exp}(N*X, _, N*X) :- \text{ground}(N), \text{var}(X).
\]

\[
\text{mul_exp}(-X, Sig, -Y) :- \text{mul_exp}(X, Sig, Y).
\]

\[
\text{mul_exp}(A+B, Sig, C+D) :- \text{mul_exp}(A, Sig, C), \text{mul_exp}(B, Sig, D).
\]

\[
\text{mul_exp}(A-B, Sig, C-D) :- \text{mul_exp}(A, Sig, C), \text{mul_exp}(B, Sig, D).
\]

\[
\text{mul_exp}(N, Sig, N*Sig) :- \text{ground}(N).
\]

\[
\text{add_vect}([], [], [], Cs, Cs).
\]

\[
\text{add_vect}([U|Us], [V|Vs], [W|Ws], C1s, C2s) :- \\
\text{add_vect}(Us, Vs, Ws, W = U+V|C1s], C2s).
\]

The predicate \text{mul_exp}(E1, Sigma, E2) scales the numeric constants that occur within \(E1\) by the variable \(Sigma\), providing they are not coefficients of variables, to obtain the expression \(E2\). Note that \(Sigma\) is a variable and the expression \(E1\) may be a variable, hence both \(E1\) and \(Sigma\) have mode \(?\) rather than \(\_\). Since a non-ground representation is employed for expressions, the test \(\text{var}(X)\) is used to determine whether the expression is a variable. As before, the test \(\text{ground}(N)\) detects numeric constants—rational numbers—which are the only type of subexpressions that are ground. Observe that \text{mul_exp} can return more than one solution, for example, \text{mul_exp}(X, Sig, E2) generates \(E2 = X; X = -(A); E2 = -(A); X = -(-(A))\), \(E2 = -(-(A))\) etc as solutions. Thus the pruning operator \text{once} is applied within \text{mul_exp}(E1, Sigma, E2) to prevent erroneous solutions.
The predicate \texttt{scale(+C1s, ?Sigma, ?C2s, -C3s)} scales each constraint within the list \texttt{C1s} by the variable \texttt{Sigma}. Each constraint consists of a binary operator and two expressions, and scaling is applied to the numeric constants in each expression as specified by \texttt{mul.exp}. For example, \texttt{scale([X+2 >= 1+Y, Y = 2], Sigma, Tail, ScaledCs)} binds \texttt{ScaledCs} to \([Y = Z, X+2*Sigma >= 1*Sigma+Y | Tail] \). Note that \texttt{scale} finesse the problem of putting \texttt{Cxs} and \texttt{Cys} into the standard form \(A_i y_i \leq B_i\) before applying scaling. In standard form, \(X+2 \geq 1+Y\) is \(Y-X \leq 1\) but scaling constants on both sides of the relational operator preserves equivalence in that \(X+2*Sig \geq 1*Sig+Y\) is equivalent to \(Y-X \leq 1*Sig\). The use of a difference list avoids an unnecessary call to append in the body of \texttt{convex hull}.

The predicate \texttt{add_vec(+Us, +Vs, -Ws, ?C1s, -C2s)} operates on the lists \(Us = [U_1, \ldots, U_n]\) and \(Vs = [V_1, \ldots, V_n]\) which correspond to the vectors \(g_1\) and \(g_2\) (as introduced in section 2). The argument \(Ws\) is instantiated to another list of variables \(W_1, \ldots, W_n\), which corresponds with \(\bar{x}\). The predicate creates the system of equalities \(W_1 = U_1+V_1, \ldots, W_n = U_n+V_n\) corresponding to the system \(\bar{x} = g_1 + g_2\). The scaled constants output by the two calls to scale are passed to \texttt{add_vec} via its accumulator and thereby combined with the system of equalities.

For example, the call \texttt{add_vec([X1,Y1], [X2, Y2], Ws, Tail, Cs)} returns the bindings \(Cs = [A=Y1+Y2, B=X1+X2|Tail]\) and \(Ws = [B,A]\).

The predicate \texttt{convex hull(Xs, Cxs, Ys, Cys, Ze, Czs)} takes, as input, two lists of constraints (Cxs and Cys) and their corresponding lists of variables (Xs and Ys) and produces as output a single list of constraints Czs over the variables Zs that represents the closure of the convex hull of the two input polyhedra. If Xs and Ys are not variable disjoint, then the pre-requisite can be satisfied by appropriately renaming variables. Specifically, the variables Xs and constraints Cxs can be renamed with \texttt{copy_term(Xs-Cxs, CpyXs-CpyCxs)} and the call \texttt{convex hull(Xs, Cxs, Ys, Cys, Ze, Czs)} replaced with \texttt{convex hull(CpyXs, CpyCxs, Ys, Cys, Ze, Czs)}. Since the integrity of the constraint store is preserved by \texttt{project} and since \texttt{project} is the only source of interaction with the store, then it follows that \texttt{convex hull} also does not side-effect any existing constraints. The following is an illustrative example.

\textbf{Example 3.5}

Running this code on the data of Example 2.2 gives:

\begin{verbatim}
| ?- convex hull([X1,Y1], [X1=0,Y1=1], [X2,Y2], [X2>=0,Y2=X2], V, S).
S = [[A>=0, A-B=<=-1, A-B=<=0],
  V = [[A,-B] ? ;
no
\end{verbatim}

\section{Discussion}

This section discusses the method proposed in the paper, comparing it with related techniques. The Chernikova method is exponential in the worst-case (Le Verge, 1992) and the Fourier-Motzkin method, like all projection techniques over linear
inequalities (Chandru et al., 2000), is also exponential. The exponential behaviour of both methods stems from the same source: the possibly exponential relationship between the number of vertices and the number of half-spaces that define a polyhedron. In fact the problem of calculating the closure of the convex hull of two polyhedra is also exponential even for bounded polyhedra (polytopes). This can be demonstrated by considering the so-called cross polytope in $n$-dimensions which is the polyhedron with the vertex set $\{ (\pm 1,0,\ldots,0), (0,\pm 1,\ldots,0), \ldots, (0,0,\ldots,\pm 1) \}$. The cross polytope can be defined by no less than $2^n$ inequalities yet can arise as the convex hull of two polyhedra both of which can be defined with $O(n)$ inequalities. Specifically consider the $n$-dimensional polyhedra

$$P_1 = \{ (x_1,\ldots,x_n) \in \mathbb{Q}^n \mid (\sum_{i=1}^{n} -x_i \leq 1) \land (\wedge_{j=1}^{n} x_j \leq 0) \}$$

$$P_2 = \{ (x_1,\ldots,x_n) \in \mathbb{Q}^n \mid (\sum_{i=1}^{n} x_i \leq 1) \land (\wedge_{j=1}^{n} -x_j \leq 0) \}$$

Because $P_1$ and $P_2$ are polytopes, they can be expressed in terms of their vertices:

$$P_1 = \text{conv}(\{ (0,0,\ldots,0), (-1,0,\ldots,0), (0,-1,\ldots,0), \ldots, (0,0,\ldots,-1) \})$$

$$P_2 = \text{conv}(\{ (0,0,\ldots,0), (1,0,\ldots,0), (0,1,\ldots,0), \ldots, (0,0,\ldots,1) \})$$

Since $(0,0,\ldots,0)$ is convexly spanned by $(1,0,\ldots,0)$ and $(-1,0,\ldots,0)$, it follows that $d(\text{conv}(P_1 \cup P_2)) = \text{conv}(P_1 \cup P_2) = \text{conv}(\{ (\pm 1,0,\ldots,0), (0,\pm 1,\ldots,0), \ldots, (0,0,\ldots,\pm 1) \})$ which is the $n$-dimensional cross polytope. The 2 and 3 dimensional cases are denoted in Figure 1 by (i) $P_1$ and $P_2$ and (ii) $Q_1$ and $Q_2$ respectively for which the cross polytopes are a solid square and an octahedron. Hence the problem of calculating the closure of the convex hull is intrinsically exponential irrespective of the algorithm employed.
Example 4.1
The following query illustrates how the hull algorithm yields an exponential number of inequalities for the 4 dimensional case.

?- Xs = [X1, X2, X3, X4], Ys = [Y1, Y2, Y3, Y4],
    Cxs = [X1 <= 0, X2 <= 0, X3 <= 0, X4 <= 0],
    Cys = [Y1+Y2+Y3+Y4 <= 1, 0 <= Y1, 0 <= Y2, 0 <= Y3, 0 <= Y4],
    convex_hull(Xs, Cxs, Ys, Cys, Zs, Czs),
    Zs = [A, B, C, D].

Czs = [A-B+C+D>=1, A+B+C-D<1, A+B+C+D>=1, A-B-C-D<1,
       A-B+C-D>=1, A+B-C-D<1, A+B+C-D>=1, A-B+C-D<1,
       A-B-C-D>=1, A+B+C-D<1, A+B-C-D>=1, A-B+C-D<1] ? ;

no

However, it would be wrong to conclude from these examples that the frame and ray representation is preferable – inequalities are unavoidable since they are required for other polyhedral operations.

Despite the scaling problems that are inherent to any convex hull algorithm, in practice the technique proposed in this paper has been widely applied in logic programming (Codish & Taboch, 1999; Genaim & Codish, 2001; King et al., 1997; Mesnard & Neumerkel, 2001; Saglam & Gallagher, 1997), mostly to satisfaction. For example, in the context of inferring termination conditions for logic programs this method is feasible since it accounts for 42% of this first pass of the analysis and the first pass itself constitutes only 23% of the total analysis time (Mesnard & Neumerkel, 2001). Whether the approach presented in this paper is applicable depends on the application context. When only standard domain operations are required and performance is not an issue, this method has much to commend it. However, when the application has to additionally reason, say, about integral points (Auncourt, 1991; Quinton et al., 1997) or parameterised polyhedra (Loechner & Wilde, 1997) then specialised polyhedral libraries are required. Further, if performance is important, then recourse should be made to a polyhedral library, since a state-of-the-art implementation employing the Chernikova algorithm (Bagnara et al., 2002), will outperform the approach presented here.

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, XSB and Yap Prolog, the technique can be easily adapted to these systems. The method is a reasonable compromise between conciseness, clarity and efficiency and variants of this program have now been widely deployed.

Acknowledgements Thanks are due to Mats Carlsson, Bart Demoen, Pat Hill, Joachim Schimpf and Raimund Seidel and, of course, the anonymous referees.
References


