Published in Formal Methods in System Design, 2006 ISSN : 0925-9856 copyright Springer-Verlag

Some ways to reduce the space dimension in polyhedra computations *

N. Halbwachs, D. Merchat, and L. Gonnord†
Vérimag‡ Grenoble - France

Abstract

Convex polyhedra are often used to approximate sets of states of programs involving numerical variables. The manipulation of convex polyhedra relies on the so-called *double description*, consisting of viewing a polyhedron both as the set of solutions of a system of linear inequalities, and as the convex hull of a *system of generators*, i.e., a set of vertices and rays. The cost of these manipulations is highly dependent on the number of numerical variables, since the size of each representation can be exponential in the dimension of the space. In this paper, we investigate some ways for reducing the dimension: On one hand, when a polyhedron satisfies *affine equations*, these equations can obviously be used to eliminate some variables. On the other hand, when groups of variables are unrelated with each other, this means that the polyhedron is in fact a *Cartesian product* of polyhedra of lower dimensions. Detecting such Cartesian factoring is not very difficult, but we adapt also the operations to work on Cartesian products. Finally, we extend the applicability of Cartesian factoring by applying suitable *variable change*, in order to maximize the factoring.

1 Introduction

Convex polyhedra, or systems of linear inequalities, are a natural way to represent upper approximations of sets of states of programs involving numerical variables. In particular, *linear relation analysis* [CH78] was defined as an abstract interpretation based on the lattice of polyhedra. The results of such analyzes were used in many contexts, like program verification [HPR97], timed and hybrid system verification [HHWT97], automatic discovery of invariants used in formal proofs [BBC+00, BBM97], compile-time error detection [DRS01], or object code optimization in compilers and automatic program parallelization [IJT91, Fea96]. Convex polyhedra have other applications, e.g., in the design

*This work has been partially supported by the project APRON of the "ACI Sécurité Informatique" of the French Ministry of Research

†Author's email: {Nicolas.Halbwachs,David.Merchat,Laure.Gonnord}@imag.fr

‡Vérimag is a joint laboratory of Université Joseph Fourier, CNRS and INPG associated with IMAG.

Published in Formal Methods in System Design, 2006 ISSN : 0925-9856 copyright Springer-Verlag

of systolic arrays [LMQ91, LM95, QR00], but in this paper we will focus on program analysis.

All these applications require a common set of operations on polyhedra: representation and simplification, intersection and convex hull, affine transformation, test for inclusion and emptiness, and widening. These operations are generally realized thanks to the "double description" of polyhedra [MRTT53], which we recall in Section 2: a polyhedron can be considered as the set of solutions of a system of linear inequalities, or can be characterized by a "system of generators", made of its set of *vertices* and its set of *infinite rays*. In general, each operation is easier on one representation, and the knowledge of both representations allows each of them to be minimized (elimination of irrelevant inequalities, non extremal vertices and rays). Several libraries for polyhedra manipulation are available [Wil93][1] [CL98][2] [HPR97][3] [BRZH02][4].

A well-known problem with the double-description is that the size of each description can grow exponentially with the dimension of the space (number of variables): an n-dimensional hypercube is defined by $2n$ inequalities, but has 2^n vertices; the converse can happen, since the descriptions are completely dual. As a consequence, the dimension of the space is a crucial limitation to polyhedra manipulation. Another, more technical problem that arises with large dimensions, is that the implemented algorithms generally work with *rational numbers* (to avoid precision problems): in such implementations, implementing each coefficient in inequalities and each component of generators as a rational number is very expensive, both in time and memory. This is why, generally, these coefficients and components are generally converted to the same denominator, this denominator being stored only once. Now, when the number n of variables is high, the common denominator of n rational numbers tends to be very large, and one is faced with serious problems of arithmetic overflows.

In program analysis, the number of variables can be reduced by well-known techniques: determining the life range of each variable [Muc97], applying program slicing [Tip95] to discard variables which do not influence the result of the analysis, etc. In this paper, we investigate some complementary approaches, which don't take the analyzed program into account, but work at the level of polyhedra operations. The proposed techniques can be (and have been) implemented as a layer above a polyhedra library.

A first, obvious, idea, is to take advantage of affine equations satisfied by a polyhedron. If we can predict that the result of an operation will satisfy some affine equations, we can use each of these equations to eliminate a variable. Of course, this technique is not likely to induce huge improvements (since it allows only one variable to be eliminated for each equation), and, as a matter of fact, in Section 3, we quickly report on disappointing experiments with this idea.

Another solution consists in detecting that a polyhedron can be factored as a Cartesian product of polyhedra in smaller dimensions. This situation, which occurs very often in real-life examples, means that the set of variables can be partitioned into subsets, such that

[1] see also `http://www.ee.byu.edu/ wilde/polyhedra.html`

[2] see also `http://icps.u-strasbg.fr/PolyLib/`

[3] see also `http://www.irisa.fr/prive/Bertrand.Jeannet/newpolka.html`

[4] see also `http://www.cs.unipr.it/ppl/`

variables belonging to different subsets are independent, i.e., not related by any inequality in the polyhedron. In order to take advantage of such factorings, they must be detected, and operations should be, as far as possible, performed on factored arguments. These topics are addressed in Sections 4 and 5. A preliminary presentation of this idea appeared in [HMPV03].

Now, the success of the factoring method is highly dependent on the choice of the variables. A single variable change in the analyzed program can have dramatic consequences on the performance of the analysis. In order to eliminate this dependence, we propose in Section 6 a way of making the best variable choice, and of performing the basis change which maximizes the factoring.

These techniques have been implemented as a layer above the Parma Polyhedra Library. In Section 7 we give some experimental comparisons on the performance with and without this layer.

2 Convex Polyhedra

Let $\mathcal{N}$ be a numerical field ($\mathbb{R}$ or $\mathbb{Q}$). A convex polyhedron (or "a polyhedron", for short) in $\mathcal{N}^n$ is a subset of $\mathcal{N}^n$ consisting of the intersection of a finite number of half-spaces. The classical "double description" [MRTT53] of a polyhedron P (see Fig.1), consists of characterizing it

- either as the set of solutions of a *system of linear inequalities*:

  $$P = \{X \in \mathcal{N}^n \mid AX \leq B\}$$

  where A is a matrix $m \times n$ and B is an m-vector. When this representation is used, we note P as $ineq(A, B)$.

- or as the convex hull of a *system of generators*, i.e., two finite sets of vectors, $V = \{V_1, \ldots, V_k\}$ ("vertices") and $R = \{R_1, \ldots, R_\ell\}$ ("rays") such that each point of P is the sum of a *convex combination* of vertices, and a *positive combination* of rays:

  $$P = \{\sum_{i=1}^{k} \lambda_i V_i \; + \; \sum_{i=1}^{\ell} \mu_i R_i \mid \lambda_i \geq 0, \mu_i \geq 0, \sum_i \lambda_i = 1\}$$

  When this representation is used, we note P as $gen(V, R)$.

The knowledge of both representations is useful for performing most common operations on polyhedra:

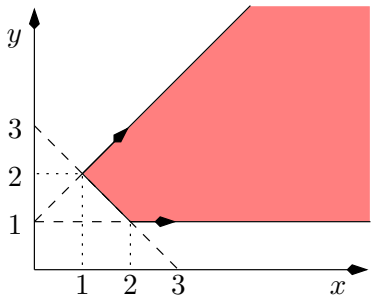
Intersection :

$$ineq(A, B) \cap ineq(A', B') = ineq([A, A'], [B, B'])$$

Convex hull — i.e., the least convex polyhedron containing the union:

$$gen(V, R) \sqcup gen(V', R') = gen(V \cup V', R \cup R')$$

Published in Formal Methods in System Design, 2006 ISSN : 0925-9856 copyright Springer-Verlag



$$\begin{bmatrix} -1 & -1 \\ -1 & 1 \\ & -1 \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \leq \begin{pmatrix} -3 \\ 1 \\ -1 \end{pmatrix}$$

$$V_0 = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, V_1 = \begin{pmatrix} 2 \\ 1 \end{pmatrix}, R_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, R_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Figure 1: Double description of a polyhedron.

Inclusion :

$$gen(V, R) \subseteq ineq(A, B) \text{ iff } AV_i \leq B \ (i = 1..k) \text{ and } AR_i \leq 0 \ (i = 1..\ell)$$

Test for emptiness :

$$gen(V, R) = \emptyset \text{ iff } V = \emptyset$$

Affine transformation — defined from an $n \times n$ matrix C and an n-vector D as $CP + D = \{CX + D \mid X \in P\}$:

$$Cgen(V, R) + D = gen(\{CV_i + D \mid i = 1..k\}, \{CR_i \mid i = 1..\ell\})^5$$

Moreover, the knowledge of both representations allows both of them to be minimized (removing redundant inequalities, vertices and rays), which is essential to avoid an explosion of their size: a constraint, a vertex, or a ray is redundant if removing it does not change the polyhedron. In the example of Fig. 1, $x \geq 0$ would be a redundant constraint, $(2, 2)$ would be a redundant vertex, and $(2, 1)$ would be a redundant ray.

The translation of each representation into the other is not very difficult [Che68, LeV92, Wil93] but can be very expensive (exponential) in the worst case, because each representation can be exponentially larger than the other *as the dimension n of the space increases.*

A last operation that we shall use is the projection, or existential quantification of a variable. If P is a polyhedron, its projection, noted $\exists x.P$, according to some variable x, can be computed using the classical Fourier-Motzkin procedure: all pairs of constraints with non-null and opposite sign coefficients for x are positively combined in order to get a null coefficient for x in the composition.

3 Predicting affine equations

The simplest idea to eliminate variables, is to take advantage of affine equations satisfied by polyhedra. The discovery of invariant affine equations has been studied in Karr's pioneering work [Kar76] a long time ago: Karr's method is an abstract interpretation

[5] In fact, affine transformations can also be performed directly on systems of inequalities.

Published in Formal Methods in System Design, 2006 ISSN : 0925-9856 copyright Springer-Verlag

working of the lattice of *affine varieties*. All the operations for minimizing systems of equations and propagating affine equations over program statement are available, and the iterative computation of affine invariants is guaranteed to converge, since the lattice of linear varieties is of finite depth.

Global affine invariants:

So, it is easy to first apply Karr's method to attach to each program point a system of affine equations invariantly satisfied by the variables, and then to use these equations to eliminate variables in costly polyhedra computations, i.e., in computing the system of generators of a polyhedron, or the intersection of two polyhedra, or the convex hull of two polyhedra. Now, our experiments show that this naive approach gives very poor results: most of the time, the performance is worse, because there are only very few invariant equations, and the benefit of using them is negligible with respect to the cost of finding them.

Temporary affine equations:

Since global affine invariants are rare, a second idea is to take advantage of temporary affine equations: at some step of the analysis, some polyhedra satisfy affine equations that can be used. If systems of inequalities are kept minimal, equations (i.e., pairs of opposite inequalities) are identified in them. So, whenever we have to perform an operation on polyhedra, we know the equations they satisfy. With respect to the previous approach, we are likely to get many more equations, since on one hand we don't require these equations to be global invariants, and on the other hand, linear relation analysis is able to discover equations which are missed by Karr's method (when they result from opposite inequalities). So, knowing the equations satisfied by the arguments of an operation, we can use the corresponding operation on equations to predict the equations satisfied by the result, and use them to eliminate variables.

In particular:

- for computing $P_1 \cap P_2$, one forms the conjunction, say C, of the systems of constraints of P_1 and P_2. Then, to check if the intersection is empty, and to simplify the system of constraints C, one has to compute the system of generators of the intersection. The equations identified in C can be used to eliminate variables before performing the costly computation of the system of generators.

- for computing the convex hull $P_1 \sqcup P_2$, knowing that $P_1 \subset E_1$ and $P_2 \subset E_2$ (where E_1 and E_2 are affine varieties), one first compute the systems of generators of P_1 and P_2. Then, the system of generators of $P_1 \sqcup P_2$ is formed as in §2, and the corresponding system of inequalities must be computed. For that, one can compute first the affine hull of E_1 and E_2, which provides the equations satisfied by $P_1 \sqcup P_2$. These equations can be used to eliminate variables before computing the system of constraints.

Published in Formal Methods in System Design, 2006 ISSN : 0925-9856 copyright Springer-Verlag

Unfortunately, experiments with this idea are very disappointing. They show that, in good cases, improvements are negligible while in bad cases there can be a significant overhead. The reason is that the expensive step is Chernikova's algorithm [Che68], and it appears that, in good implementations of this algorithm, equations are already implicitly exploited.

So, the whole attempt to use affine equations to improve polyhedra operations is a failure, but even failures deserve to be reported!

4 Factoring of Polyhedra

A more promising idea is to detect that a polyhedron is a Cartesian product of several polyhedra of lower dimension. As a matter of fact, such a factoring can result in a logarithmic reduction of the size of the system of generators: for instance, consider the family of hypercubes defined by $H_n = \{0 \leq x_i \leq 1 \mid i = 1, ..n\}$. H_n has 2^n vertices, but if we notice that all variables are independent from each other, we can consider H_n as the product of n intervals in 1-dimensional spaces.

Let I be a subset of $\{1 \dots n\}$. We note $P \downarrow I$ the projection of the polyhedron P on variables with indices in I (i.e., the result, in $\mathcal{N}^{|I|}$ of the existential quantification of all variables with indices outside I). Conversely, if P_I is a polyhedron on variables with indices in I, and if $I \subset J$, we note $P_I \uparrow J$ the extension of P_I to the greater space (i.e., the polyhedron on variables with indices in J, such that $(P_I \uparrow J) \downarrow I = P_I$).

Let $(I_1, I_2, \dots I_\ell)$ be a partition of $\{1 \dots n\}$. We say that a polyhedron P *can be factored according to* $(I_1, I_2, \dots I_\ell)$ if and only if

$$P = P \downarrow I_1 \times P \downarrow I_2 \times \dots P \downarrow I_\ell$$

A matrix A is *block-diagonalizable* according to a partition $(I_1, I_2, \dots I_\ell)$ if for each of its rows A_i there is one $k_i \in \{1..\ell\}$ such that $\{j \mid A_i^j \neq 0\} \subseteq I_{k_i}$.

Some obvious facts:

f1. for any polyhedron P, there is a greatest partition $(I_1, I_2, \dots I_\ell)$ according to which P can be factored (possibly the trivial partition, with $\ell = 1$).

f2. for any matrix A, there is a greatest partition $(I_1, I_2, \dots I_\ell)$ according to which A is block-diagonalizable (possibly the trivial partition, with $\ell = 1$).

f3. if $P = \mathit{ineq}(A, B)$, and if A is block-diagonalizable according to a partition $(I_1, I_2, \dots I_\ell)$, then P can be factored according to $(I_1, I_2, \dots I_\ell)$ (the converse is not true, if the system of constraints is not minimal). This gives an easy way to factor a polyhedron, and to get the constraint descriptions of its factors: each constraint $A_i X \leq B_i$ becomes a constraint of the factor P_{k_i}.

f4. For any pair (P, P') of polyhedra (resp., for any pair (A, A') of matrices) there is a greatest common partition (possibly the trivial partition) according to which both polyhedra can be factored (resp., both matrices are block-diagonalizable).

Published in Formal Methods in System Design, 2006 ISSN : 0925-9856 copyright Springer-Verlag

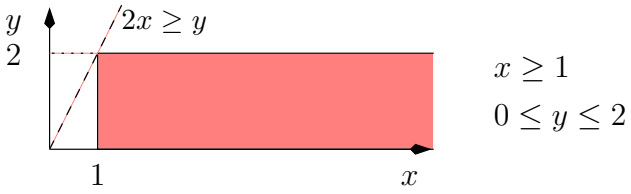


Figure 2: A factored polyhedron.

f5. Conversely, given a description of the factors $P_1, \ldots P_\ell$, one can easily obtain the corresponding description of $P = P_1 \times \ldots \times P_\ell$:

- its system of constraints is just the conjunction of those of the factors;
- its system of generators is obtained by composing together all the ℓ-tuples of vertices (resp., of rays) of the factors. This composition explains the explosion of the size of the systems of generators, since $|V| = \prod_{k=1}^{\ell} |V_k|$ and $|R| = \prod_{k=1}^{\ell} |R_k|$.

A similar treatment works also to obtain a description of P factored according to any partition rougher than $(I_1, I_2, \ldots I_\ell)$.

Example: Fig. 2 shows a factored polyhedron in 2 dimensions. In its minimal system of constraints:

$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \leq \begin{pmatrix} -1 \\ 2 \\ 0 \end{pmatrix}$$

the matrix is block-diagonal. Now, if the redundant constraint $2x \geq y$ is added, the matrix is non longer block-diagonalizable, and the factoring of the polyhedron is hidden.

5 Operations on factored polyhedra

Most of the operations mentioned in Section 2 can be easily applied component-wise to factored polyhedra. The operands need first to be factored in the same way (using *f4* and *f5* above). Moreover, in many cases, the result may be better factored than the operands, which can be done using *f2*. We first consider these easy operations, before dealing with the convex hull, which is more difficult.

5.1 Easy operations

Intersection. If P and P' are factored according to the same partition $(I_1, I_2, \ldots I_\ell)$, then so is $P \cap P' = P_1 \cap P'_1 \times P_2 \cap P'_2 \times \ldots \times P_\ell \cap P'_\ell$. It may be the case that $P \cap P'$ can be further factored (Fig. 3).

Affine transformation. Let $X \mapsto CX + D$ be an affine transformation. If C is block-diagonalizable according to $(I_1, I_2, \ldots I_\ell)$, and P is factored according to the same

(a) same factoring (b) better factoring

Figure 3: Intersection of factored polyhedra.

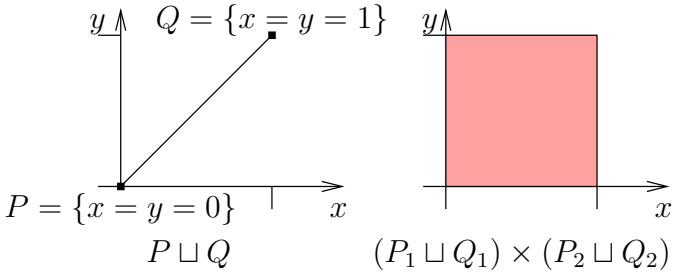


Figure 4: Convex hull vs. component-wise hull

partition, then so is $CP + D = C_{I_1}P_1 + D_{I_1} \times \ldots \times C_{I_\ell}P_\ell + D_{I_\ell}$. If C is not invertible, it can be the case that $CP + D$ can be further factored.

Widening. If P and P' are factored according to the same partition $(I_1, I_2, \ldots I_\ell)$, then so is $P \nabla P' = P_1 \nabla P'_1 \times P_2 \nabla P'_2 \times \ldots \times P_\ell \nabla P'_\ell$. It may be the case (in fact, it happens very often) that $P \nabla P'$ can be further factored

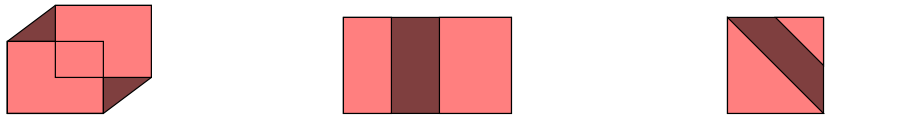
Emptiness and inclusion. Let $P = P_1 \times P_2 \times \ldots \times P_\ell$. Then P is empty if and only if there exists $k \in \{1..\ell\}$ such that P_k is empty. If P and P' are factored according to the same partition $(I_1, I_2, \ldots I_\ell)$, then $P \subseteq P'$ if and only if, for all $k \in \{1..\ell\}$, $P_k \subseteq P'_k$.

5.2 The Convex Hull

The computation of the convex hull is more difficult. Obviously, if $P = P_1 \times \ldots \times P_\ell$ and $Q = Q_1 \times \ldots \times Q_\ell$ are two polyhedra factored according to the same partition, then $P \sqcup Q \subseteq (P_1 \sqcup Q_1) \times \ldots \times (P_\ell \sqcup Q_\ell)$, but the later is generally a rough approximation of the former (see figure 4). Moreover, the convex hull of two factored polyhedra can be either less factored (Fig. 5.a) or as factored (Fig. 5.b), or even more factored (Fig. 5.c) than the operands.

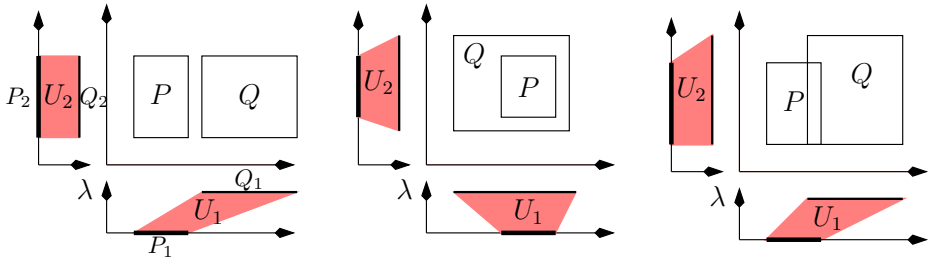
The goal is to get the factored result, when possible, in a decomposed way, and without penalizing the computation when the result is not factored. We consider first the case of partitions (I_1, I_2) of size 2:

Proposition 1. Let $P = P_1 \times P_2$ and $Q = Q_1 \times Q_2$ be two polyhedra factored according to the same partition (I_1, I_2). Let V be the set of variables, $\lambda \notin V$ be an auxiliary



(a) less factoring (b) same factoring (c) better factoring

Figure 5: Convex hull of factored polyhedra.



(a) λ only bounded by constants in U_2: factored result

(b) λ only lower bounded in U_1 and U_2: factored result

(c) λ upper bounded in U_1 and lower bounded in U_2: unfactored result

Figure 6: Convex hull of factored polyhedra

additional variable, and let us note $V' = V \cup \{\lambda\}$, $I'_i = I_i \cup \{\lambda\}, i = 1, 2$. Let us consider

$$\begin{aligned} U_1 &= (P_1{\uparrow}I'_1 \cap \{\lambda = 0\}) \sqcup (Q_1{\uparrow}I'_1 \cap \{\lambda = 1\}) \\ \text{and} \quad U_2 &= (P_2{\uparrow}I'_2 \cap \{\lambda = 0\}) \sqcup (Q_2{\uparrow}I'_2 \cap \{\lambda = 1\}) \end{aligned}$$

Then:

- if λ is upper bounded by a non constant expression in U_1 and lower bounded by a non constant expression in U_2, or conversely, then the convex hull $P \sqcup Q$ is not factored according to (I_1, I_2), and $P \sqcup Q = \exists\lambda.(U_1{\uparrow}V' \cap U_2{\uparrow}V')$.

- otherwise $P \sqcup Q = \exists\lambda.U_1 \times \exists\lambda.U_2$.

So, the factored convex hull algorithm consists in computing the two convex hulls U_1 and U_2 (with an auxiliary variable λ, added to I_1 and I_2), and then to check if the result will be factored; if so, the projections $\exists\lambda.U_1$ and $\exists\lambda.U_2$ (which are computed by Fourier-Motzkin procedure) provide the factors of the results; otherwise, λ must be eliminated (again by Fourier-Motzkin) from the conjunction of the systems of inequalities of U_1 and U_2.

Fig. 6 illustrates the main cases that can occur when applying this algorithm[6].

[6]Of course, in such an example with only two variables, the factored convex hull is of no interest!

Published in Formal Methods in System Design, 2006 ISSN : 0925-9856 copyright Springer-Verlag

Proof of Proposition 1: By definition of the convex hull,

$$\begin{array}{rcl} X \in P \sqcup P' & \Leftrightarrow & \exists Y \in P, Y' \in P', \lambda \in [0,1], \text{ such that } X = \lambda Y + (1-\lambda)Y' \\ & \Leftrightarrow & X = (X_1, X_2) \wedge \exists \lambda \in [0,1] \text{ such that} \\ & & \exists Y_1 \in P_1, Y_1' \in P_1', X_1 = \lambda Y_1 + (1-\lambda)Y_1' \wedge \\ & & \exists Y_2 \in P_2, Y_2' \in P_2', X_2 = \lambda Y_2 + (1-\lambda)Y_2' \\ & \Leftrightarrow & X = (X_1, X_2) \wedge \exists \lambda \in [0,1] \text{ such that} \\ & & (X_1, \lambda) \in Q_1 \wedge (X_2, \lambda) \in Q_2 \end{array}$$

Now, the existential quantification of λ in the last system of constraints can only produce dependencies between previously independent variables in two cases:

- if there is some constraint $E(X_1) \leq \lambda$ in Q_1, and some constraint $\lambda \leq F(X2)$ in Q_2 — which will produce $E(X_1) \leq F(X_2)$;
- or, conversely, if there is some constraint $\lambda \leq E(X_1)$ in Q_1, and some constraint $F(X2) \leq \lambda$ in Q_2 — which will produce $F(X_2) \leq E(X_1)$;

where $E(X_1)$ and $F(X2)$ are non constant expressions. Otherwise, λ can be quantified separately in Q_1 and Q_2.
□

The procedure generalizes directly to polyhedra factored into k factors, still using only one auxiliary variable:

Proposition 2. Let $P = P_1 \times \ldots \times P_\ell$ and $= Q_1 \times \ldots \times Q_\ell$ be two polyhedra factored according to the same partition. Let λ be a fresh variable and let us consider the polyhedra $(U_k)_{k=1..\ell}$ defined by:

$$U_k = (P_k{\uparrow}I_k' \wedge \{\lambda = 0\}) \sqcup (Q_k{\uparrow}I_k' \wedge \{\lambda = 1\}$$

Then, the partition of $P \sqcup Q$ is obtained from $(I_1, \ldots, I_\ell)$ by merging I_k and $I_{k'}$ whenever either λ is lower-bounded by a non constant expression in U_k and upper-bounded by a non constant expression in $U_{k'}$, or conversely. Let $(J_1, \ldots, J_h)$ be the resulting partition, each J_m being a union of some I_ks. Then

$$P \sqcup Q = R_1 \times R_2 \times \ldots \times R_h \text{ where } R_m = \exists \lambda, \bigtimes_{I_k \subseteq J_m} U_k$$

6 More Cartesian factoring

Cartesian factoring often gives good results in practice, but it is highly dependent of the choice of variables in the analyzed program. A simple variable change in the program can have dramatic consequences on the cost of the analysis. To solve this problem, in this section we investigate the idea of performing automatically the most suitable variable change, before applying costly operations. Fig 7 shows the kind of transformation we want to do.

Published in Formal Methods in System Design, 2006 ISSN : 0925-9856 copyright Springer-Verlag

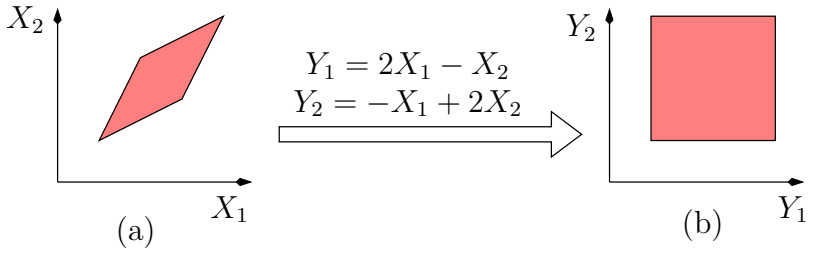


Figure 7: Basis change for factoring

6.1 Best factoring of a polyhedron

As a matter of fact, it is not difficult, given the system of constraints of a polyhedron, to find the variable change which maximizes the factoring: we just have to find a maximal subset of linearly independent constraints — in other terms, to apply the classical Gauss elimination algorithm which extracts a *basis* out of the set of vectors of constraints coefficients —, and to perform a variable change according to these constraints. Intuitively, the basis change deforms the polyhedron so that its faces be maximally orthogonal with each other, and parallel to the axes, just as in Fig. 7.

initial: a system of constraints $AX \geq B$,
where A is an $m \times n$ matrix, and B is an m-vector
Write the system as $AX - Y = B\ ,\ Y \geq 0$ (add "slack variables")
Use Gauss elimination to put the matrix $[A, -I]$ in the form $\begin{bmatrix} I, & -A' \\ & -A'' \end{bmatrix}$;
It results into a system $X = A'Y' + B'\ ,\ Y'' = A''Y' + B''\ ,\ Y' \geq 0\ ,\ Y'' \geq 0$ where $Y = (Y'', Y')$, which can also be written $X = A'Y' + B'\ ,\ A''Y' + B'' \geq 0\ ,\ Y' \geq 0$
return: $X = A'Y' + B'$ as the basis change, and $A''Y' + B'' \geq 0\ ,\ Y' \geq 0$ as the new system of constraints.

The returned system of constraints $A''Y' + B'' \geq 0\ ,\ Y' \geq 0$ is maximally factorizable: of course, the new basis Y' represents a maximal set of linearly independent constraints of the initial system; on the other hand, if a subset Y'_I of variables are related by some constraint in $A''Y' + B'' \geq 0$, these variables correspond to a set of linearly dependent constraints in the initial system. Since basis change does not change linear dependence, this set of constraints would involve related variables in any basis.

Let's illustrate the use of this algorithm on the following example:

$$\begin{bmatrix} x_1 & +x_2 & +x_3 & \geq 0 \\ x_1 & & +x_3 & \geq 0 \\ 3x_1 & -x_2 & +x_3 & \geq 0 \\ -4x_1 & & -2x_3 & \geq -1 \end{bmatrix}$$

Adding the slack variables Y produces

$$\left[\begin{array}{llllllll} x_1 & +x_2 & +x_3 & -y_1 & & & = & 0 \\ x_1 & & +x_3 & & -y_2 & & = & 0 \\ 3x_1 & -x_2 & +x_3 & & & -y_3 & = & 0 \\ -4x_1 & & -2x_3 & & & & -y_4 = & -1 \end{array}\right]$$

The Gauss elimination provides

$$\left[\begin{array}{llllllll} x_1 & & & +y_2 & & +\frac{1}{2}y_4 & = & \frac{1}{2} \\ & x_2 & & +y_2 & +y_3 & +y_4 & = & 1 \\ & & x_3 & -y_2 & & -\frac{1}{2}y_4 & = & \frac{1}{2} \\ & & & y_1 & +y_3 & +y_4 & = & 1 \end{array}\right]$$

which can be read (after elimination of y_1) as

- a system of equations:

$$\left[\begin{array}{llllll} x_1 & = & -y_2 & & -\frac{1}{2}y_4 & +\frac{1}{2} \\ x_2 & = & -y_2 & -y_3 & -y_4 & +1 \\ x_3 & = & 2y_2 & & +\frac{1}{2}y_4 & -\frac{1}{2} \end{array}\right]$$

- and a new system of constraints:

$$\left[\begin{array}{llll} & -y_3 & -y_4 & \geq & -1 \\ y_2 & & & \geq & 0 \\ & y_3 & & \geq & 0 \\ & & y_4 & \geq & 0 \end{array}\right]$$

In the final system of constraints, y_2 is independent of y_3 and y_4, so the system can be partitioned according to $I_1 = \{y_2\}$ and $I_2 = \{y_3, y_4\}$.

6.2 Variable changes in operations

Let us detail how we use the variable change, when dealing with operations on polyhedra.

- A simple case is when one deals with only one polyhedron (computation of generators, test for emptiness). In this case, one first performs a normal factoring, because it is cheaper, as it does not involve a rewriting of constraints. Then, one tries to apply the preceding algorithm on each factor (involving enough variables for the further factoring to be interesting), before performing the operation. If this operation is the computation of generators, the result is obtained by composing the results and applying the inverse variable change.

  In the previous example, one finds at once the generators of the factors:

$$P^{(I_1)} = gen(\{0\}, \{1\}) \quad , \quad P^{(I_2)} = gen(\{(0,0),(1,0),(0,1)\}, \emptyset)$$

Composing these results provides the following system of generators in the basis (y_2, y_3, y_4):

$$V = \{(0,0,0),(0,1,0),(0,0,1)\}, R = \{(1,0,0)\}$$

and the application of the inverse variable change, we get the system of generators in the basis(x_1, x_2, x_3):

$$V = \{(\tfrac{1}{2},1,-\tfrac{1}{2}),(\tfrac{1}{2},0,-\tfrac{1}{2}),(0,0,0)\}, R = \{(-1,-1,2)\}$$

- Now, in general, we want to find a good factoring before applying some binary operation (e.g., a convex hull). So the real problem is to find a good *common* factoring of the arguments of the operation. The above algorithm cannot be simply applied to both arguments, since there is no reason for it to select a common factoring.

  A simple solution consists in merging the systems of constraints of both arguments before applying the variable change algorithm, which provides a maximal common factoring of the arguments.

7 Experimental Results

Two kinds of experiments have been conducted to evaluate the influence of the proposed methods:

- at the operation level, a comparison was made on polyhedra operations, using the Parma Polyhedra Library (PPL) with and without our new layer.

- inside a complete program analyzer, by comparing the analysis performance on several programs, also using the PPL with and without our layer.

7.1 Efficiency of operations

Performing experiments on benchmarks of operations may seem meaningless, since the results are, of course, highly dependent of the chosen benchmark. However, we made two kinds of experiments, at this level:

- some series of systematic experiments on "regular" classes of well-chosen polyhedra, representing good or bad cases, to highlight the variation of performance with increasingly complex problems.

- we benefited of a benchmark of operations [IN04], gathered by another team from their experiments in program verification.

Systematic experiments: We consider two families of polyhedra: easily factored hypercubes $H_n(a,b) = \{a \leq x_i \leq b\}$, and polyhedra $K_n(a,b) = \{\bigwedge_{j-1}^{n}(a \leq \Sigma_{i=1}^{j} x_i \leq b)\}$ which need a basis change to be transformed into a factored hypercube. We give the times for performing convex hulls of such polyhedra, in several situations:

- $H_n(0,2) \sqcup H_n(0,3)$ (Table 1.(a)), where the result is factored;
- $H_n(0,2) \sqcup H_n(1,3)$ (Table 1.(b)), where the result is not factored at all;
- $K_n(0,1) \sqcup K_n(0,2)$ (Table 1.(c)), where a basis change is needed and the result is factored;
- $K_n(0,1) \sqcup K_n(1,2)$ (Table 1.(d)), where a basis change is needed and the result is not factored at all.

Not surprisingly, these results show that, in very good situations, the factoring and basis change involve important improvements. But they show also that (1) the convex hull is always cheaper, even when the result is not factored, and (2) even when it is useless, the basis change only causes a negligible over-cost.

Results on an external benchmark: We got [IN04] a benchmark of convex hulls collected from experiments with the PIPS analyzer [IJT91]. Table 2 shows the results on this benchmark: for each interval of dimensions, it gives the number of convex hulls in this interval of dimensions in the benchmark, and the average ratio between the time taken by the PPL alone and the PPL equipped with our complete factoring layer. Notice that we can compute in quite high dimensions, because, due to the relational bottom-up analysis performed by PIPS, the systems of constraints in this benchmark are very sparse. The results show that the ratio gets better and better as the dimension increases.

7.2 Influence on program analyzes

We also tried our extension of the PPL by comparing the performance of our program analyzer with and without the extensions, on many examples. Here we give the synthetic results concerning

- a series of examples (mostly communication protocols) available on the web page[7] of the FAST tool [BFLP03]. Comparing our performances with those of FAST would be unfair, since we perform approximate analysis while FAST computes the reachable states exactly.
- several versions of the "subway" toy example, presented in [HPR97], where the number of trains can be increased, and where we verify several properties of the system.

[7] `http://www.lsv.ens-cachan.fr/fast/`

Dimension	12	13	14	15	50
PPL	18.99	102.94	516.54	>600	>600
with factoring	0.01	0.02	0.02	0.02	0.03
with factoring and basis change	0.01	0.02	0.02	0.02	0.03

(a) Convex hull of hypercubes H_n, no need for basis change, factored result

Dimension	8	9	10	11	12
PPL	0.70	3.81	11.58	84.96	662.99
with factoring	0.30	1.26	2.50	13.03	77.02
with factoring and basis change	0.33	1.25	3.53	12.86	77.74

(b) Convex hull of hypercubes H_n, no need for basis change, not factored result

Dimension	19	20	21	22	23	24	25
PPL	1.96	10.72	45.56	196.82	>600	>600	>600
with factoring	2.12	11.19	46.77	196.79	>600	>600	>600
with factoring and basis change	0.12	0.28	0.77	3.15	23.78	108.53	442.63

(c) Hypercubes K_n needing basis change, factored result

Dimension	19	20	21	22	23	24	25
PPL	3.49	13.62	60.32	235.67	>600	>600	>600
with factoring	5.51	18.07	65.09	245.04	>600	>600	>600
with factoring and basis change	0.13	0.28	0.77	3.22	23.59	109.28	445.17

(d) Hypercubes K_n needing basis change, not factored result

Table 1: Influence of factoring and basis change on the compuation of simple convex hulls

Dim.	40-49	50-59	60-69	70-79	80-89	90-99	100..9	110..9	120..9	130..9	150..9
Nb hulls	36	7	26	12	13	16	37	6	32	33	3
Ratio	1.8	1.9	2.1	2.1	4.0	4.4	4.7	5.7	6.0	6.0	7.6

Table 2: Results on the PIPS benchmark

Published in Formal Methods in System Design, 2006 ISSN : 0925-9856 copyright Springer-Verlag

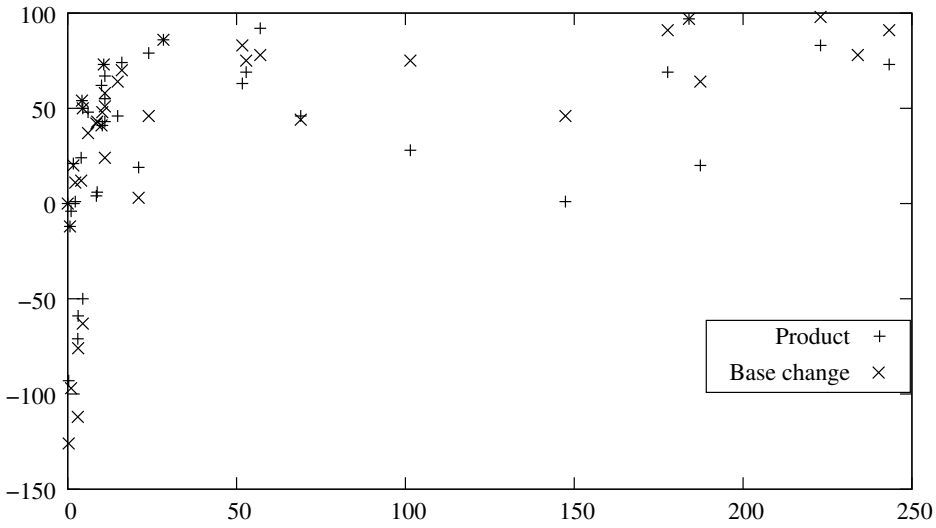


Figure 8: % time gain, in function of the execution time

- an industrial example, which is an avionic fault-tolerant instrument which acquires gyroscopic data (axes: roll, pitch, yaw) from redundant sensors. Here again, the number of sensors and the number of axes may be augmented to increase the complexity of the analysis; moreover, various fault hypotheses can be taken into account.

We subsume the experimental results by giving the gain, in percentage, between the analysis times with PPL alone and with one or the other of our improvements. Fig. 8 gives these gains in function of the execution time with PPL alone. It shows that both methods are effective for big examples. Fig. 9 and 10 show these gains in function of the number of variables, with Cartesian factoring alone and with basis change, respectively. Here, we see that the benefits become apparent from some dimension, between 8 and 12. This means, of course, that our techniques should not be applied to small examples; it means also that the partitioning should not be refined beyond some limit, in terms of size of the classes.

These examples show that, not only the Cartesian factoring can significantly improve the performance of Linear Relation Analysis, but also that further improvements can be obtained using basis change, without introducing a significant overhead. In our opinion, these benefits due to basis change are especially apparent for systems dealing with counters, thresholds, etc., where the analyzed properties often depend on differences of variables, or even arbitrary affine combinations of variables.

8 Related works and Conclusion

We presented two attempts for reducing the number of variables in polyhedra operations. The first one, using affine equations, was found uninteresting. In contrast, the use of Cartesian factoring and its extension by changing the space basis, show real improve-

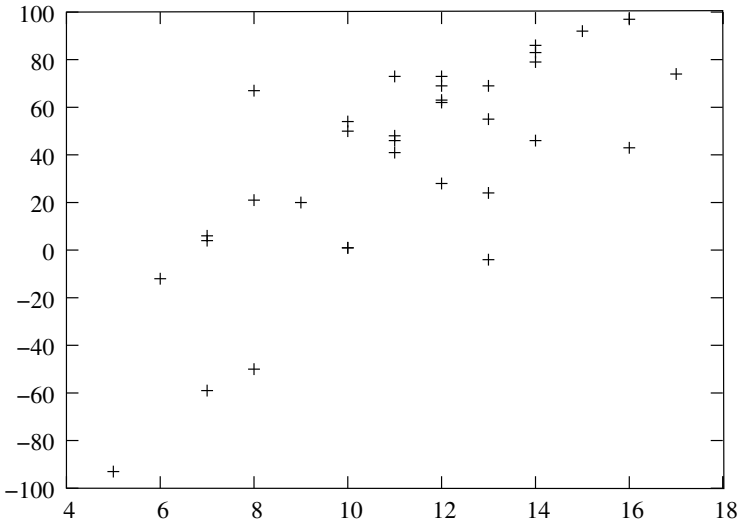


Figure 9: % time gain with factorization, in function of the dimension

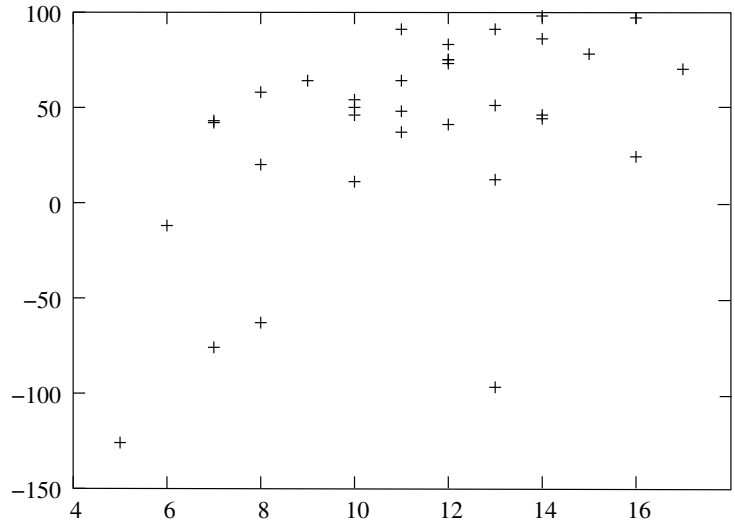


Figure 10: % time gain with basis change, in function of the dimension

ments in many cases, and negligible overhead in bad cases. So, the Cartesian factoring technique extended with basis change constitutes a layer which should be inserted above the polyhedra libraries. We did not find, in the literature, other attempts to reduce the dimension at the operation level, and *without losing information*. In Astrée [BCC+03], a so-called "packing" technique is applied to reduce the size of relations in relational lattices: the set of variables is split into subsets, and only relations concerning variables in the same subset are considered. The choice of the subset is made statically (i.e., before the analysis), according to some heuristic, and, of course, the "packing" of relations loses information. In [GDD+04], a technique is presented for dealing with unbounded dimensions, by folding the space using "summary dimensions". Here again, of course, the folding does not preserve the information of the initial relation. Our approach tries to reduce the dimension without losing information. Only when further reduction is necessary, should losing information reductions be considered.

References

[BBC+00] N. Bjorner, A. Browne, M. Colon, B. Finkbeiner, Z. Manna, H. Sipma, and T. Uribe. Verifying temporal properties of reactive systems: A STeP tutorial. *Formal Methods in System Design*, 16:227–270, 2000.

[BBM97] N. Bjorner, I. Anca Browne, and Z. Manna. Automatic generation of invariants and intermediate assertions. *Theoretical Computer Science*, 173(1):49–87, February 1997.

[BCC+03] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. In *PLDI 2003, ACM SIGPLAN SIGSOFT Conference on Programming Language Design and Implementation*, pages 196–207, San Diego (Ca.), June 2003.

[BFLP03] S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. Fast: Fast acceleration of symbolic transition systems. In *CAV'03*, pages 118–121, Boulder (Colorado), July 2003. LNCS 2725, Springer-Verlag.

[BRZH02] R. Bagnara, E. Ricci, E. Zaffanella, and P. M. Hill. Possibly not closed convex polyhedra and the parma polyhedra library. In M. V. Hermenegildo and G. Puebla, editors, *9th International Symposium on Static Analysis, SAS'02*, Madrid, Spain, September 2002. LNCS 2477.

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *5th ACM Symposium on Principles of Programming Languages, POPL'78*, Tucson (Arizona), January 1978.

[Che68] N. V. Chernikova. Algorithm for discovering the set of all solutions of a linear programming problem. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 8(6):282–293, 1968.

[CL98] Ph. Clauss and V. Loechner. Parametric analysis of polyhedral iteration spaces. *Journal of VLSI Signal Processing*, 19(2), July 1998.

[DRS01] N. Dor, M. Rodeh, and M. Sagiv. Cleanness checking of string manipulations in C programs via integer analysis. In P. Cousot, editor, *SAS'01*, Paris, July 2001. LNCS 2126.

[Fea96] P. Feautrier. Automatic parallelization in the polytope model. In *The Data Parallel Programming Model: Foundations, HPF Realization, and Scientific Applications*, pages 79–103. LNCS 1132, Springer Verlag, 1996.

[GDD+04] D. Gopan, F. DiMaio, N. Dor, T. Reps, and M. Sagiv. Numeric domains with summarized dimensions. In *TACAS'04*, pages 512–529, Barcelona, 2004.

[HHWT97] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:110–122, 1997.

[HMPV03] N. Halbwachs, D. Merchat, and C. Parent-Vigouroux. Cartesian factoring of polyhedra in linear relation analysis. In *Static Analysis Symposium, SAS'03*, San Diego, June 2003.

[HPR97] N. Halbwachs, Y.E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, August 1997.

[IJT91] F. Irigoin, P. Jouvelot, and R. Triolet. Semantical interprocedural parallelization: An overview of the PIPS project. In *ACM Int. Conf. on Supercomputing, ICS'91, Köln*, 1991.

[IN04] F. Irigoin and D. Nguyen. Private communication, 2004.

[Kar76] M. Karr. Affine relationships among variables of a program. *Acta Informatica*, 6:133–151, 1976.

[LeV92] H. LeVerge. A note on Chernikova's algorithm. Research Report 635, IRISA, February 1992.

[LM95] V. Loechner and C. Mongenet. A toolbox for affine recurrence equations parallelization. In *International Conference and Exhibition on High-Performance Computing and Networking*, pages 263–268, May 1995.

[LMQ91] H. LeVerge, Ch. Mauras, and P. Quinton. The alpha language and its use for the design of systolic arrays. *Journal of VLSI Signal Processing Systems*, 3(3):173–182, September 1991.

[MRTT53] T. S. Motzkin, H. Raiffa, G. L. Thompson, and R. M. Thrall. The double description method. In H. W. Kuhn and A. W. Tucker, editors, *Contribution to the Theory of Games – Volume II*. Annals of Mathematic Studies, nr 28, Princeton University Press, 1953.

[Muc97] S. S. Muchnick. *Advanced Compiler Design Implementation*. Morgan Kaufmann Pub., 1997.

[QR00] F. Quilleré and S. Rajopadhye. Optimizing memory usage in the polyhedral model. *ACM TOPLAS*, 22(5), September 2000.

[Tip95] F. Tip. A survey of program slicing techniques. *Journal of Programming Languages*, 3(3):121–189, September 1995.

[Wil93] D. K. Wilde. A library for doing polyhedral operations. Research Report 785, IRISA, December 1993.