Querying Constraints

Jean-Louis Lassez
IBM T.J. Watson Research Center
P.O.Box 704
Yorktown Heights, NY 10598
jll@ibm.com

Abstract

The design of languages to tackle constraint satisfaction problems has a long history. Only more recently the reverse problem of introducing constraints as primitive constructs in programming languages has been addressed. A main task that the designers and implementers of such languages face is to use and adapt the concepts and algorithms from the extensive studies on constraints done in areas such as Mathematical Programming, Symbolic Computation, Artificial Intelligence, Program Verification and Computational Geometry. In this paper, we illustrate this task in a simple and yet important domain: linear arithmetic constraints. We show how one can design a querying system for sets of linear constraints by using basic concepts from logic programming and symbolic computation, as well as algorithms from linear programming and computational geometry. We conclude by reporting briefly on how notions of negation and canonical representation used in linear constraints can be generalized to account for cases in term algebras, symbolic computation, affine geometry, and elsewhere.

1 Introduction

Various forms of declarative arithmetic are built in languages of the CLP class [JL] such as CLP($\Re$), CHIP, CAL, Prolog III, BNR-Prolog. Declarative arithmetic has also been introduced in languages not related to Logic Programming such as Mathematica and Trilogy, and in principle at least could be introduced in the paradigms of Functional and Object Oriented Programming. One could hope, and a strong case is made in the paper by Kanellakis, Kuper and Revesz (in this volume), that constraints will play a major role in query languages. It is fair to say that the major obstacle to the use of constraints as primitive elements in any programming system is the lack of efficiency (computation time as well as size of output and size of intermediate computation, as we know from Symbolic Computation). In that respect the linear case is interesting. It has a wide range of applications and since it has been thoroughly studied in various fields, there is a wealth of techniques we can draw upon.

In the main sections below, we address the problem of querying systems of linear constraints. We illustrate how we can make use, in our context, of concepts and techniques from Logic Programming, Symbolic Computation, Linear Programming and Computational Geometry.

We conclude with a few remarks on a the-

ory emanating from the embedding of constraints in programming systems, thus going further than a direct application of "foreign" techniques. We know that a number of elementary properties of linear constraints dealing with canonical representations and negation [LMc88,LMc89] also hold for other types of constraints such as term equations and inequations [LMM]. Such properties are important as they allow efficient handling of constraint propagation and tell us which constraints are efficiently "negatable". A complete treatment is to be found in [LMc] in which we abstract these properties and build an axiomatization to account for a variety of domains.

2 Querying systems of linear constraints

We consider here, as in linear programming, a system of constraints as a conjunction of equality and inequality constraints. We will draw an analogy with the situation in Logic Programming: the set of clauses is the program, the answer to a query $Q(x,y)$? is a set of substitutions. The substitutions establish a relationship between variables which is satisfied iff $Q(x,y)$ is a logical consequence of the program. A single algorithm, resolution, is used to compute the answer, regardless of the query. The program can be viewed as an implicit representation of a set of its logical consequences: the least model.

Here the program will be the set of constraints. Queries will be parameterized, the parameters playing the role of logical variables. The answer to a query will be a set of linear constraints on the parameters that is satisfied iff the query is implied by the program. The algorithm that is analoguous to resolution is *variable elimination*, this single algorithm will provide the required answer for any query. Finally the *subsumption cone*, which characterizes the set of all constraints implied by the program, provides the analogy with the notion of least model. We will motivate the introduction of such objects by a few examples.

Let S be the set of constraints in store. What type of information do we want to extract from S, during execution or at output time? First examples are, does $S \Rightarrow Q$? where Q is of the types: $x = 3$, $x + y - z = 1$, $x - y + 3z \leq 4$. For such queries a simple yes/no answer is required and can be obtained by showing that the constraint $S \wedge \neg Q$ is not solvable. This is the standard technique for subsumption in Theorem Proving.

In Logic Programming one goes a level higher: we do not merely ask for a yes/no answer, unless the query is ground (that is has no variables). For a general query, we obtain as a side effect of the solvability test, a set of substitutions which form a finite representation of the set of answers. So the examples of queries we just gave correspond to ground queries in a Logic Program, despite the presence of variables. A trivial case of a type of query that would correspond to the more powerful queries in Logic Programs is: do the constraints in store imply that the variable x has a fixed value? More formally: $\exists\alpha\forall x : S \Rightarrow x = \alpha$? Clearly here we not only want to know whether x has a fixed value, but if so we want to know its value.

More generally, linear relationships between the program variables may be implied by the constraints in store. This information is essential for representing constraints in a canonical form, for output standardization, constraint propagation, the elimination of redundancy in parallel (see [LHM], [LMc88], [LMc89] for details). In CLP($\Re$), a non-linear expression such as $z = log(x - y)$ is delayed. The knowledge that $x - y$ has a fixed value α for some α will resolve the delay.

Let us consider now inequality queries. In CLP($\Re$) the output should represent the relationships between the input variables only. However the constraints in store contain aux-

iliary variables introduced during the execution of the rules in the program. For example, let x, y be the input variables and u, v be the auxiliary variables in $S = \{x + 2y + u \leq 1, -y - u + v \leq 2, x + u - v \leq 0\}$. After eliminating u and v we obtain $\{x - y \leq 2\}$ as the output. A related situation occurs in order to resolve guards in committed choice languages [M], [S]. The guards contain existential queries, and their execution leads to a similar problem of variable (or quantifier) elimination.

These problems can be formalized as answering *parameterized queries* of the form: $\exists\alpha, \beta, \gamma, ...\forall x, y, ... : S \Rightarrow \alpha x + \beta y + ... \leq$ (or $=$) γ and $R(\alpha, \beta, ..., \gamma)$? where $R(\alpha, \beta, ..., \gamma)$ is a set of linear relations on the parameters such as $\alpha = 0$, $\beta \leq 2\gamma$. What we request is a finite representation of the set of answers. In the case where the query is simply $0 \leq \alpha$? the answer will tell us if the system is satisfiable or not. A query such as $\alpha x \leq \beta$? is asking for the range of the variable x, that is a classical linear programming problem, since the range is given by the max and the min of the objective function x. The fact that the answer to a query is a set of constraints on the parameters will be made clear in the following section.

3 Executable Specifications

Here we will see how we can specify the conditions under which a query is implied by a set of constraints, and provide a simple variable elimination procedure, due to Fourier, that provides the answer. Let us first say a few words about Fourier's elimination procedure and solvability algorithm [F]. Despite its simplicity, its historical and theoretical importance, it is not well known and for sake of being self contained it is described here informally:

Let S be a set of inequality constraints $ax + by + ... \leq c$. We first select a variable, say x, and consider all possible pairs of constraints from S where x appears with coefficients of opposite signs. If this set is empty, that is if x appears with coefficients of the same sign in all constraints, we delete all constraints containing x. If the set of pairs is not empty, from each pair we generate a new constraint which does not contain x, by computing an appropriate linear combination of the two constraints. These new constraints replace in S the constraints that contained x, giving a set $S1$. Now S is solvable if and only if $S1$ is solvable. In geometric terms $S1$ represents the projection wrt to the x-axis of the polyhedral set associated to S. So variable elimination is a projection operation. The process is repeated until all variables have been eliminated.

Fourier's theorem tells us that S is not solvable if and only if a contradiction $0 \leq c$ (where c is a negative number) has been generated in the process. It has often been remarked that Fourier's procedure bears a strong analogy with resolution (selecting two literals of opposite signs, unifying, building the resolvent clause, etc).

As was mentioned in [DE], Fourier's result can be used to establish in a straightforward manner fundamental theorems in Linear Programming. We also know from Symbolic Computation that variable (or quantifier) elimination is a powerful tool. Let us see a few examples. As a first illustration consider the problem of computing the convex hull of a set of points directly from its specification: A point P is in the convex hull of points $P_1, ..., P_n$ iff $\exists\lambda_1, ..., \lambda_n$, $\lambda_i \geq 0$ $\forall i$ and $\sum \lambda_i = 1$ such that $P = \sum \lambda_i P_i$.

Let $(x,y,..)$ be the coordinates of P. If we eliminate the λ's from this specification we obtain a relationship between solely the coordinates $(x,y,...)$ which is solvable if and only if the initial relationship is. It therefore represents the desired convex hull.

Example: let P_1=(1,0,0), P_2=(0,1,0), P_3=(0,0,1) and P=(x,y,z). We have $x = \lambda_1, y = \lambda_2, z = \lambda_3, \lambda_1 + \lambda_2 + \lambda_3 =$

$1, \lambda_1 \geq 0, \lambda_2 \geq 0, \lambda_3 \geq 0$. We trivially obtain a representation of the convex hull by eliminating the λ's: $x + y + z = 1, x \geq 0, y \geq 0, z \geq 0$. So variable elimination provides us with a systematic way of characterizing interesting sets of constraints directly from an existential specification.

We address now the problem of obtaining existential specifications for sets of constraints implied by a set S of inequality constraints. A constraint C is a *quasi-linear combination* of constraints of $S = \{C_1, ..., C_n\}$ iff C is obtained by adding a positive number to the right hand side of a non-negative linear combination of constraints of S. The following theorem is a direct corollary to Fourier's theorem. It is very rarely mentioned, does not seem to have been used in that form at least, but is in fact equivalent to the fundamental Duality Theorem in Linear Programming [A]. It provides a simple characterization of the set of constraints implied by S, and forms the basis for our approach. (For sake of simplicity we will use the word combination for non-negative combination unless specifically stated).

Theorem 1 (Subsumption Theorem [A]) *A constraint C is implied by a set of constraints S iff C is a quasi-linear combination of constraints of S.*

Now we can specify that a constraint $C = \alpha x + \beta y + ... \leq \gamma$ is implied by S: Let the constraints in S be $\{a_1 x + b_1 y + ... \leq c_1, a_2 x + b_2 y + ... \leq c_2, ...\}$. Then C is implied by S iff $\exists \lambda_1 \geq 0, \lambda_2 \geq 0...$, and $q \geq 0$ such that

$$\sum \lambda_i a_i = \alpha$$

$$\sum \lambda_i b_i = \beta$$

$$\vdots$$

$$\sum \lambda_i c_i + q = \gamma$$

Define the *subsumption cone* of S, denoted $SC(S)$ as the polyhedral set obtained by elimination of the λ's and q from the above specification. By Fourier's elimination we have

Proposition 1 *Let S be a set of linear inequalities, a constraint $\alpha x + \beta y + ... \leq \gamma$ is implied by S iff the point $(\alpha, \beta, ..., \gamma)$ belongs to the subsumption cone of S.*

Because of this proposition we are justified in claiming that there is an analogy between the subsumption cone and the least model in Logic Programming. To show the interest of this notion, we will now present a few properties of the subsumption cone.

Example: $S = \{-x \leq 0, -y \leq 0, x + y \leq 1\}$. We have $\alpha = -\lambda_1 + \lambda_3, \beta = -\lambda_2 + \lambda_3, \gamma = \lambda_3 + q, \lambda_1 \geq 0, \lambda_2 \geq 0, \lambda_3 \geq 0, q \geq 0$. From which we derive $SC(S) = \{-\alpha + \gamma \geq 0, -\beta + \gamma \geq 0, \gamma \geq 0\}$. We use the word cone in the definition because as no constant appears in the specification, the resulting set of inequalities is always an homogeneous system defining a cone.

This notion of subsumption cone is useful when we have to test repeatedly for implication. Using S requires running a linear program, using $SC(S)$ requires a simple evaluation of the constraints. $SC(S)$ also gives us information about cutting and supporting hyperplanes. The points in $SC(S)$ characterize the implied constraints. Dually the constraints which are incompatible with S are characterized by the open cone symmetric to $SC(S)$ with respect to the origin. Consequently the constraints which are not implied and are not incompatible correspond to points in the complement of the two cones, which is straightforward to check. The faces of these constraints are the hyperplanes which cut S. The supporting hyperplanes of S correspond to points on the facets of $SC(S)$ as they are both implied by S and at the limit of cutting.

Clearly two sets of constraints that have the same implications, that is the same sub-

sumption cone, are equivalent: they define the same polyhedral set. So a subsumption cone uniquely determines a polyhedral set. Now a subsumption cone may be pointed or not. When pointed, it is the convex closure of its extreme rays. The other case is a little bit too involved to be addressed here. Let us just mention that these two cases correspond to the fact that S may or may not be full dimensional. The complete treatment requires the use of the canonical form [LMc88] and results about the structure of polyhedral cones [GT].

Proposition 2 *Let S be a set of linear inequalities. The set of constraints derived from the set of extreme rays of the subsumption cone of S is a set of constraints equivalent to S (under assumption of full dimensionality).*

Consider the previous example. Each of the constraints $-\alpha + \gamma \geq 0, -\beta + \gamma \geq 0, \gamma \geq 0$ defines a facet of the subsumption cone. To obtain the extreme rays of the cone we intersect the constraints in all possible pairs of adjacent hyperplanes supporting the facets. That is we solve the systems: $\{-\alpha + \gamma = 0, -\beta + \gamma = 0, \gamma \geq 0\}$, $\{-\alpha + \gamma = 0, \gamma = 0, -\beta + \gamma \geq 0\}$, $\{-\beta + \gamma = 0, \gamma = 0, -\alpha + \gamma \geq 0\}$ Simplifying we obtain : $\alpha = \beta = \gamma, \gamma \geq 0$ which gives the constraint $\gamma x + \gamma y \leq \gamma$ equivalent to $x + y \leq 1$, while $\alpha = \gamma = 0, \beta \leq 0$ is equivalent to $-y \leq 0$, and $\beta = \gamma = 0$, $\alpha \leq 0$ is equivalent to $-x \leq 0$. Using this technique we can find an executable specification for the convex hull $CH(P_i, 1 \leq i \leq n)$ of a collection of polyhedral sets P_i.

Proposition 3 $CH(P_i, 1 \leq i \leq n)$ = *extreme rays* $\{\bigcap_i^n SC(P_i)\}$ *(under assumption of full dimensionality).*

Indeed a constraint is implied by the convex hull iff it is implied by all P_i's. Consequently the subsumption cone of the convex hull is equal to the intersection of the subsumption cones of the P_i's. The extreme ray extraction gives us a set of constraints defining the affine hull. Algorithmically we can compute in parallel the $SC(P_i)$ by variable elimination, the intersection is trivial as we just collect all the sets of constraints together, finally computing extreme rays is a classical problem. The general case will be discussed in the full paper.

We will later mention a more efficient method than Fourier's to compute the subsumption cone. However, one should also consider adapting the powerful convex hull algorithms from Computational Geometry.

So to answer a parametric query, one can first compute the subsumption cone and add to it the relations that the parameters must satisfy in the query. Else we can express directly using theorem 1 that the constraint in the query is implied by the system, and eliminate the λ's and q.

The subsumption cone is therefore a new tool to reason about sets of constraints. Other sets can be defined similarly, for instance we can express that a parametric constraint is implied by a system $S1$, while its opposite is implied by a system $S2$. Eliminating the appropriate variables will give us a relation which characterizes the set of hyperplanes separating $S1$ and $S2$. A simple variant would be to characterize pairs of parallel hyperplanes at a fixed distance d from each other and which separate $S1$ and $S2$. Another interesting application of variable elimination is the computation of the image of the polyhedral set by a linear application. One needs only eliminate the source variables in the specification. The image is given by the resulting constraints. We leave as a simple exercise the following construction: given a set of inequality constraints whose constants on the right hand side are parameters $r_1, r_2, ..., r_n$, find the relation on the r_i's which is satisfied iff the system is solvable.

We will now propose an alternative to Fourier's method for variable elimination which leads to a more practical system.

4 Variable elimination via extreme points

If we eliminate all variables but one, say x, from a set S of constraints, then we compute the range of the variable x. This can be done by the classical methods of linear programming via the maximization and minimization of the objective function x. These methods are far more efficient than Fourier's elimination. If we eliminate all variables but n, we look for the range of a point in an n dimensional space. Clearly we need to generalise the notion of objective function to capture this range. This is what we do now.

Assume we want to eliminate given variables from a set of constraints S. Consider the parametric query Q where the existentially quantified variables are the variables from S that we want to keep. $\exists\alpha,\beta,\gamma,...\forall x,y,... : S \Rightarrow \alpha x + \beta y + ... \leq \gamma$? The set of answers to this query represents the set of constraints implied by the projection of S in the (x, y, ...) space. It is easy to see that we can in fact restrict ourselves to linear combinations, rather than the quasi-linear. So the projection we want to compute will be a minimal set of generators for the set of linear combinations which are answers to the query. We express now that the constraint in the query is a linear combination of the constraints in S. Let the constraints in S be $\{a_1x + b_1y + ... \leq c_1, a_2x + b_2y + ... \leq c_2, ...\}$ we then have the relations:

$$\sum \lambda_i a_i = \alpha$$

$$\sum \lambda_i b_i = \beta$$

$$\vdots$$

$$\sum \lambda_i c_i = \gamma$$

$$\sum \lambda_i d_i = 0$$

$$\vdots$$

$$\sum \lambda_i = 1$$

$$\lambda_i \geq 0$$

where the equalities whose right-hand-side is zero correspond to eliminated variables. We have normalized the coefficients of the linear combination so that their sum is equal to one, without loss of generality. This is in fact more than syntactic convenience. One sees easily that the set of solutions to the above system is closed for normalized linear combinations. In geometrical terms it means that the set of points whose coordinates are the coefficients of the linear combinations is a convex set. A classical theorem states that the set of points in a polytope is the convex closure of a finite set of extreme points. So if we can show that our convex set has a finite set of extreme points E we will have a characterization of the set of answers which is a specialization of the subsumption theorem: (where G is the set of constraints which correspond to E).

Theorem 2 *A constraint C is an answer to the query Q iff it is a quasi-linear combination of the finite set of constraints in G.*

In order to establish this result and provide a way of computing G we generalize the optimization function in Linear Programming in the following way.

In the above system, the set of constraints that are not parameterized represents, but for the lack of objective function, a linear programming problem in standard form. An objective function Φ is a mapping of R^n into R. Let Δ be the polyhedral set associated with the constraints. $\Phi(\Delta)$ is an interval in R, and the linear programming problem is to determine its maximum or minimum (when they exist). That is we have to compute one or the other of the *extreme points* of the image of a polyhedral set. What we also know is that the value of the maximum or minimum is obtained as an image of an extreme point of Δ. We generalize this

picture by taking as objective function a function Φ, from R^n to R^m this time, defined by the parametric constraints in the above system, and by considering the set of extreme points of $\Phi(\Delta)$ instead of a minimum or maximum. We have again the fact that the extreme points in $\Phi(\Delta)$ are images of the extreme points of Δ, and as Δ is a polytope, $\Phi(\Delta)$ has a finite number of extreme points. This provides an informal proof for the result we needed:

Theorem 3 *Let P be the generalized linear program:*

$$extr(\Phi(\Delta))$$

$$\Phi = \begin{cases} \sum \lambda_i a_i = \alpha \\ \sum \lambda_i b_i = \beta \\ \vdots \\ \sum \lambda_i c_i = \gamma \end{cases}$$

$$\Delta = \begin{cases} \sum \lambda_i d_i = 0 \\ \vdots \\ \sum \lambda_i = 1 \\ \lambda_i \geq 0 \end{cases}$$

The solutions to that program determine a finite set of constraints which defines the projection of S.

However, the finite set of constraints should be minimized by redundancy elimination to obtain a better representation of the projection of S on the $(x, y, \ldots)$ space. As there are algorithms to compute sets of extreme points [MR] we can effectively obtain G. Implementation issues are not trivial and will be treated in [HL]. Let us just mention here that it is possible to have an output of exponential size. Consequently it would be unrealistic to assume that one can arbitrarily query anything more than very special sets of constraints. Of course very small sets or cases where the number of variables is small, or cases where the number of variables is close to the number of constraints (even if these numbers are large), or sparse systems will be suitable. The range of applications of such a querying system needs to be established. Also it would be interesting to look at approximate symbolic answers when the actual answer is known to be of unmanageable size, or eventually compute alternate (dual for instance) representations.

5 Existence of implicit equalities and causes of unsolvability

In the previous sections we were concerned with generating answers to queries. Here we address a complementary problem: what are the causes, that is, what are the subsets of constraints in a system, that imply a given answer? This approach will allow us to provide a simple solution to the problem of implicit equalities which is shown to be very similar to the problem of detecting causes of unsatisfiability.

Much is known about the handling of equality queries. In a set of inequality constraints, those which can be replaced by an equality constraint by simply replacing $\leq$ by $=$ without changing the semantics are called *implicit equalities*. This set of constraints plays a role analogous to the set G in the previous section. Implicit equalities in fact define the affine hull of the polyhedral set associated to a set S of inequality constraints. There are a number of methods to compute them, a standard one being to run a linear program for each constraint in the set. This is not very efficient, particularly in the case where there are no implicit equalities present, which occurs frequently for a large class of problems. In the case of CLP($\Re$) where backtracking may occur, all this work is wasted. There are far more sophisticated ways of computing implicit equalities by using a single linear program [FRT]. However in these methods the size of the prob-

lem is substantially increased as well as the number of variables. In logic based programming languages, we would rather want to reduce the number of variables than increase it. So these algorithms also lead to a substantial overhead in our setting.

What we need here is an efficient algorithm that answers the existential query first, so that we pay the price of generating the answers only when we have a guarantee that they exist. Recently, it was found that Fourier's algorithm for solvability of inequality constraints has this property [LM]. It was established in [LM] that implicit equalities exist if and only if Fourier's algorithm generates a tautology $0 \leq 0$. So we have "for free" the information we request as a side effect of solvability.

We will use the results of the previous section to provide an efficient method of determining the existence of implicit equalities as a side effect of a simple and efficient solvability test, and to separately generate them when required. The results from [LM] are also used to establish correctness. Essentially we formulate the solvability problem in a variant of duality in linear programming. It is a variant in that first we view the dual space as a space of linear combinations and not according to its usual economic or geometrical interpretation. So we are justified in adding a normalization constraint which would not be meaningful otherwise. (We could do without, in principle, but as it forces the polyhedral set to be finite it considerably simplifies the algorithms). And also, of course, it is extended so as to take care of implicit equalities. Let S be the set of constraints, its quasi-dual formulation D expresses that we have normalized linear combinations of constraints in S that eliminate all variables. It is therefore an application of Theorem 1 in a particular case. The objective function is obtained as in the case of linear programming but its use will be adapted to our purpose. We have now the theorem:

Theorem 4

1. *If the quasi-dual linear program D is not solvable then S is solvable and contains no implicit equalities.*

2. *If the quasi-dual linear program D is solvable then:*

   (a) *If the objective function has a strictly positive minimum then S is solvable and does not contain any implicit equality.*

   (b) *If the objective function has zero as a minimum then S is solvable and contains implicit equalities.*

   (c) *If in the process of minimizing the objective function a negative value is obtained, then S is not solvable.*

3. *When implicit equalities exist they can be obtained by generating the set of extreme points of D with the objective constraint set to zero.*

The proof of this theorem is obtained as a consequence of [LM] and the arguments in the previous section. It is important to note that we have not given an algorithm here strictly speaking, but rather a different formulation of the problem. Any solving algorithm can be used with this formulation for parts 1 and 2 in the above theorem. As for part 3 any algorithm which generates all extreme points can be applied. (Or more efficient methods if need be). This technique to find the causes of tautologies $0 \leq 0$, can be easily extended to find the causes of inconsistencies: one has to record for each successive negative value taken by the objective function, the combination of constraints that are responsible.

6 Negatable constraints and canonical representations

One of the main theoretical and practical issues in Logic Programming is an efficient implementation of negation. One approach is to have a weaker form of negation, as in intuitonistic logic, another approach is to restrict the use of negation to simple subformulas. In [LMc], we introduce, in an abstract setting, the notion of *negatable constraints*. They represent the constraints which can be negated "for free" in a system that is without increasing the complexity of the associated solver. This corresponds to an intuitionistic behavior of disjunction (hence the dubious label of "crypto-intuitionism"). This property is closely linked to the existence of a canonical form which is particularly suitable for constraint propagation. The proposed axiomatization is sufficiently general to account for a variety of examples that come up in affine geometry, group theory, symbolic computation, term algebras and elsewhere. It is also interesting to note that some of the axioms appear in a characterization of matroids. We will use our previous setting of linear constraints to introduce these notions and suggest how they may be systematically abstracted.

Linear equality constraints define affine spaces. The well known geometric property "an affine space is contained in a union of affine spaces iff it is contained in one member of this union" is used in [KKR] to show the homomorphism theorem. Let us call this property the *strong compactness property*. It implies that equality constraints are negatable. Indeed consider a system of equality constraints and a conjunction of inequations (that is negated equalities). How do we test this system for solvability? A replacement of each inequation by a disjunction of strict inequalities would lead to a combinatorial explosion and transform the problem into a linear programming problem. However this is not needed since the strong compactness property implies that we can test the set of equality constraints independently with each inequation. The set is not solvable iff the hyperplane associated with one of the inequations contains the affine space defined by the equality constraints. The key factor behind the strong compactness property is the notion of dimension: one cannot cover an object of dimension d with a finite (even denumerable in that case) number of objects of dimension strictly smaller than d.

In [LMM] the problem of sets of equations and inequations in the Herbrand universe was addressed using the analogy with the situation of linear arithmetic equalities and inequations that we just described. All that was needed was to introduce the notion of dimension of the set of solutions to an equality in the Herbrand universe: the number of domain variables in an idempotent mgu (which is an invariant). To test for solvability does not lead to a combinatorial explosion, and the unification algorithm remains sufficient.

We can easily find similar examples of negatable constraints in domains that benefit from the notion of dimension. Let us give a simple example with a different flavor. Let the domain be a completely divisible group G. Let the constraints be with one argument: $H(x)$ is satisfied iff x belongs to subgroup H of G. All constraints are negatable. The strong compactness property is established by an application of the pigeon-hole principle.

Let us now consider the problem of canonical form, in the case of (positive) linear constraints. A first use is for standardisation of representation: two sets of constraints in canonical form should be equal iff they represent the same polyhedral set. This can be easily achieved if the polyhedral set is full dimensional: a triangle in a two dimensional space is uniquely described by a set of three constraints. If the set is not full dimensional there is an infinite number of non redundant equivalent sets of constraints defining the same poly-

hedral set. The solution adopted in [LMc88] is quite natural: break the syntactic representation in two parts. One is the uniquely representable set of equalities defining the affine hull of the polyhedral set, the other is a unique set of inequalities as we are back to the full dimensional case. What is the link with the equalities as negatable constraints?

Let us note first that the strong compactness property still holds: "A polyhedral set is contained in an union of hyperplanes iff it is contained in one member of this union". So the equality constraints are still negatable, we have no combinatorial explosion and solvability is still a linear programming problem despite the presence of inequations. But if the constraints are in canonical representation we are in a better situation as the solvability problem reduces to Gaussian elimination despite the presence of inequalities. What happens is that a polyhedral set behaves exactly in the same way as its affine hull in the presence of inequations: an hyperplane contains a polyhedral set iff it contains its affine hull.

The same results would be achieved with convex sets instead of polyhedral sets, or even taking dense subsets of affine spaces. An axiomatization of this phenomenon leads to three axioms which are part of the characterization of matroids, the missing axiom being the exchange property. From these one can extend the notion of canonical representation to sytems of constraints containing negative constraints. Important results follow. Essentially they imply that we can separate, in the canonical form, the different types of constraints in a semantically meaningful way. In terms of implementation it means that we can use the solvers more efficiently. In terms of querying the system, it means that if the query is about say negatable constraints then we need consult only the negatable part of the canonical representation.

Acknowledgements

I thank T. Huynh, J. Jaffar, C. Lassez, M. Maher and K. McAloon for many fruitful discussions.

References

[A] S.Achmanov, *Programmation Linéaire*, Editions Mir, Moscou 1984.

[DE] G.B. Dantzig and B.C. Eaves, Fourier-Motzkin Elimination and Its Dual, *Journal of Combinatorial Theory Ser. A*, 14 (1973) 288-297.

[F] J.B.J. Fourier, reported in: Analyse des travaux de l'Académie Royale des Sciences, pendant l'année 1824, Partie mathématique, *Histoire de l'Académie Royale des Sciences de l'Institut de France* 7 (1827) xlvii-lv. (Partial English translation in: D.A. Kohler, Translation of a Report by Fourier on his work on Linear Inequalities, *Opsearch* 10(1973) 38-42.)

[FRT] R.M. Freund, R. Roundy and M.J. Todd, Identifying the Set of Always-Active Constraints in a System of Linear Inequalities by a Single Linear Program, Technical Report, Sloan School of Management, Massachusetts Institute of Technology, October 1985.

[GT] A.J. Goldman and A.W. Tucker, Polyhedral Convex Cones in Linear Inequalities and Related Systems. *Annals of Mathematical Studies 38.* Princeton University Press, 1956.

[HL] T. Huynh and J-L. Lassez, Design and Implementation of Algorithms for Variable Elimination in Linear Arithmetic Constraints, forthcoming.

[JL] J. Jaffar and J-L. Lassez, Constraint Logic Programming, *POPL 87*, 111-119.

[JMSY] J. Jaffar, S. Michaylov, P. Stuckey and R. Yap, The CLP($\Re$) Language and System, IBM Research Report, T.J. Watson Research Center, forthcoming.

[KKR] P. Kanellakis, G. Kuper and P. Revesz, Constraint Query Languages *PODS 90*, Nashville.

[LHM] J-L. Lassez, T. Huynh and K. McAloon, Simplification and Elimination of redundant arithmetic constraints, *Proceedings of NACLP 89*, MIT Press.

[LM] J-L. Lassez and M.J. Maher, On Fourier's Algorithm for Linear Arithmetic Constraints, IBM Research Report, T.J. Watson Research Center, 1988.

[LMM] J-L. Lassez, M.J. Maher and K. Marriott, Unification Revisited, *Foundations of Logic Programming and Deductive Databases*, J. Minker ed., Morgan-Kaufmann 1988.

[LMc] J-L. Lassez and K. McAloon, A Constraint Sequent Calculus, submitted 1989.

[LMc88] J-L. Lassez and K. McAloon, Applications of a Canonical Form for Generalized Linear Constraints, *Proceedings of the FGCS Conference*, Tokyo, December 1988, 703-710.

[LMc89] J-L. Lassez and K. McAloon, Independence of Negative Constraints, *TAPSOFT 89*, Advanced Seminar on Foundations of Innovative Software Development, LNCS 351 Springer Verlag 89.

[M] M. Maher, A Logic Semantics for a class of Committed Choice Languages, *Proceedings of ICLP4*, MIT Press 87.

[MR] T.H. Matheiss and D.S. Rubin, A Survey of Comparison of Methods for Finding All Vertices of Convex Polyhedral Sets, *Mathematics of Operations Research*, 5 (1980) 167-185.

[S] V. Saraswat, Concurrent Constraint Logic Programming, Ph.D. Dissertation, Carnegie Mellon University 1989.