PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1992,
edited by ICOT.

Reasoning With Constraints

Catherine Lassez

IBM T.J. Watson Research Center
P.O.Box 704, Yorktown Heights, NY 10598, USA
lassez@watson.ibm.com

Constraints are key elements in areas such as Operations Research, Constructive Solid Geometry, Robotics, CAD/CAM, Spreadsheets, Model-based Reasoning and AI. Languages have been designed specifically to solve constraints problems. More recently, the reverse problem of designing languages that use constraints as primitive elements has been addressed. Constraints handling techniques have been incorporated in programming languages and systems like CLP($\Re$), CHIP, CAL, CIL, Prolog III, 2LP, BNR-Prolog, Mathematica and Trilogy.

In the rule-based context of Logic Programming, the CLP scheme [5] provides a formal framework to reason with and about constraints. The key idea is that the important semantic properties of Horn clauses do not depend on the Herbrand Universe or Unification. These semantic properties and their associated programming methodology hold for arithmetic constraints and solvability (and in many other domains including strings, graphs, booleans,...). The CLP scheme is a main example of the use of constraints as the primitive building blocks of a class of programming languages, since logic formulae can be themselves considered as constraints.

In the same spirit constraints have been introduced in committed choice languages in Maher [14], and in the work of Saraswat [15], and in Database querying languages by Kanellakis, Kuper and Revesz [6]. The link between classical AI work on constraints, and Logic Programming has been described by van Hentenryck [17].

Not surprisingly there are many different paradigms reflecting the integration of constraints and languages. The main differences come from the aims of the language: general purpose programming language, database or knowledge based query language, or a tool for problem solving. In mathematical programming the focus is on optimization, in artificial intelligence the focus is on constraint satisfaction and constraint propagation, in program verification the focus is on solvability. This should be reflected in the design of appropriate languages, but constraint programming should also have its own focus and theory.

We have developped a general framework for a systematic treatment of specific domains of constraints. We recall that a logic formula is viewed as an implicit and concise representation of its set of logical consequences and that the answer to a query Q is a set of substitutions which establish a relationship between the variables of Q, satisfied if and only if Q is a logical consequence of the formula. The key point is that a single algorithm, *Resolution*, is sufficient to answer all queries. These properties of logic formulae have counterparts in other domains. In particular, Tarski's theorem for quantifier elimination in closed fields[16] establishes that an arithmetic formula can be viewed as representing the set of all its logical consequences, that is the set of all arithmetic formulae it entails. Furthermore, a single algorithm, *Quantifier Elimination*, is required in analogy with logic formulae and resolution.

At the design and implementation level, however, the problems are far more difficult than for logic formulae. To try and circumvent these problems one must make heavy use of results and algorithms from symbolic computation, operations research, computational geometry etc... Also, as in the case of logic formulae, we have to sacrifice generality to achieve acceptable efficiency by carefully selecting sets of constraints for which suitable algorithms can be found.

Parametric queries Applying the paradigmatic aspects of reasoning with logic formulae to linear arithmetic, we have that:

- a set of constraints is viewed as an implicit representation of the set of all constraints it entails
- there is a query system such that an answer to a query Q is a relationship that is satisfied if and only if the query is entailed by the system.
- there exists a *single* algorithm to answer all queries.

Given a set S of arithmetic constraints as a conjunction of linear equalities, inequalities and negative constraints (disjunctions of inequations), we define a *parametric query* [7] as:

$$\exists \alpha_1, \alpha_2, \ldots, \beta \; \forall x_1, x_2, \ldots : S \Rightarrow \alpha_1 x_1 + \alpha_2 x_2 + \ldots \leq \beta \\ \wedge R(\alpha_1, \alpha_2, \ldots, \beta)?$$

where S is the set of constraints in store and R is a set of linear relations on the parameters $\alpha_1, \alpha_2, \ldots, \beta$.

Parametric queries provide a general formalism to extract information from sets of constraints and to express standard operations. For instance:

1. is S solvable? If not, what are the causes of unsolvability?
2. does S contains redundancies or implicit equalities?
3. is S equivalent to S'?
4. is it true that $x = 2$ is implied by S?
5. does there exist α such that $x = \alpha$ is implied by S?
6. does there exist a linear relation $\alpha x + \beta y + \ldots = \gamma$ implied by S?
7. does there exist $\alpha_1, \alpha_2, \ldots \beta$ such that $S \Rightarrow \alpha_1 x + \alpha_2 y \ldots \leq \beta$ and $\alpha_1 = 2\alpha_2 - 1$?

The solvability query is typical of linear programming and corresponds to the first phase of the Simplex method. Finding the causes of unsolvability is a typical problem of constraints manipulation system where the constraints in store can be modified to restore solvability using feedback information provided by the solver. Queries 2 and 3 both address the problem of constraint representation. Redundancy is a major factor of complexity in constraints processing and the removal of redundancies and the detection of implicit equalities are key steps in building a suitable canonical representation for the constraints [10] [12]. Queries 4 and 5 are classic Constraint Satisfaction Problems (CSP) and queries 6 and 7 are generalization of CSP to linear relations: variables are bound to satisfy given linear relations instead of simply values.

A priori, there does not seem to be any real connections between these various queries. However, they can all be expressed as parametric queries which ask under what conditions on the parameters $\alpha_1, \alpha_2, \ldots, \beta$, the constraint in the query is implied by the constraints in store. By varying the parameters, specific queries can be formulated. For instance,

- *is x bound to a specific value a?*
  $\exists \alpha_1, \alpha_2, \ldots, \beta$, s.t. $S \Rightarrow \alpha_1 x_1 + \alpha_2 x_2 + \ldots = \beta$ with $\alpha_1 = 1, \alpha_2 = 0, \ldots, \beta = a$.
- *is x ground?*
  same as above but with β unconstrained.
- *does S implies $2x_1 + 3x_2 \leq 0$?*
  as above with $\alpha_1 = 2, \alpha_2 = 3, \ldots, \beta = 0$.
- *what are the constraints implied by the projection of S the $\{x_1, y_2\}$-plane?*
  All parameters except $\alpha_1, \alpha_2, \beta$ set to 0

The test for solvability and the classic optimization problem can also be expressed in this way:

- *is S solvable?*
  as above with all parameters $\alpha_1, \alpha_2, \ldots$ set to 0 except $\beta \geq 0$.
  (by Fourier's theorem, which states that a set of constraints is solvable if and only if the elimination of all the variables results in a tautology)
- *what are the upper and lower bounds of $f = x_1 + x_2 + x_3$?*
  as above with $\alpha_1 = 1, \alpha_2 = 1, \alpha_3 = 1$, all other parameters are set to 0 except $\beta \geq 0$. The answer gives the upper and lower bounds for β that correspond to the minimum and maximun of f.

Parametric queries generalize logic programming queries which ask if there exists an assignment of values to the variables in the query so that the query becomes a logical consequence of the program clauses. They also generalize CSP's queries which are restricted to constraints of the type $x = a$.

We now must address the problem of finding a finite representation for the answers to the queries. Parametric queries are more complex than simple conjunctions of constraints as they involve universal quantifiers, non linearity and implication. However by using a result linked to duality in linear programming [8], we can reduce the problem to a case of conjunction of linear constraints. The Subsumption Theorem states that *a constraint is implied by a set of constraints S iff it is a quasi-linear combination of constraints in S.* A quasi-linear combination of constraints is a positive linear combination with the addition of a positive constant on the righ-hand side. For instance, let S be the set

$$\{2x + 3y - z \leq 1,\ x - y + 2z \leq 2,\ x - y + z \leq 0\}$$

and Q be the query

$$\exists \alpha, \beta, \forall x, y, S \Rightarrow \alpha x + \beta y \leq 1?$$

The following relations express that the constraint in Q is a quasi-linear combination of the constraints in S.

$$\begin{aligned} &2\lambda_1 + \lambda_2 + \lambda_3 = \alpha \\ &3\lambda_1 - \lambda_2 - \lambda_3 = \beta \\ &-\lambda_1 + 2\lambda_2 + \lambda_3 = 0 \\ &\lambda_1 + 2\lambda_2 + q = 1 \\ &\lambda_1 \geq 0, \lambda_2 \geq 0, \lambda_3 \geq 0, q \geq 0 \end{aligned}$$

where the λ_i's are the multipliers of the constraints in S. It is from this simpler formulation that variables are eliminated.

Variable elimination is the key operation to obtain answers to queries. It plays the role of resolution in Logic Programming. With inequalities, the complexity problems are far more severe than in Logic Programming, even in the restricted domain of conjunctions of positive linear constraints.

Fourier's method The basic algorithm is Fourier's[2]. The severity of the problem is illustrated by the table below:

Number of variables eliminated	Number of constraints generated	Actual number of constraints needed
0	32	18
1	226	40
2	12,744	50
3	39,730,028	19
4	390,417,582,083,242	2

The middle column gives the size of the output of Fourier's method to eliminate between 1 to 4 variables from an initial set of 32 constraints. The right most column gives the minimum size of equivalent outputs. Fourier's elimination is in fact doubly exponential as it generates an enormous amount of redundant information. Even if we remove redundancy on the fly, we are still left with exponential size for intermediate computation and potential exponential size for output. To solve this problem, one must look for output bound algorithms (an important area of study in computational geometry), that will guarantee an output when its size is small, bypassing the problem of intermediate swell. Also in the case where the size of the output is unmanageable, there is no point in computing it. However, we may sacrifice completeness and search for an approximation of reasonable size. That brings us back to avoiding intermediate swell.

The extreme points method This method, derived from the formalism of parametric queries, is interesting as it shows that variable elimination can be viewed as a straightforward generalization of a linear program in its specification and as a generalization of the simplex in its execution. Let $S = Ax \leq b$ and let V be the set of variables to be eliminated, the associated generalised linear program GLP is defined as:

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

$$\Phi = \begin{cases} \sum \lambda_i a_{i_1} = \alpha_1 \\ \quad \vdots \\ \sum \lambda_i a_{i_k} = \alpha_k \\ \sum \lambda_i b_i = \beta \end{cases}$$

$$\Delta = \begin{cases} \sum \lambda_i a_{i_{k+1}} = 0 \\ \quad \vdots \\ \sum \lambda_i a_{i_m} = 0 \\ \sum \lambda_i = 1 \\ \lambda_i \geq 0 \end{cases}$$

where $extr$ denotes the set of extreme points. Δ represents the conditions to be satisfied by a combination of constraints of S that eliminates the required variables. The normalization of the λ's ensures that Δ is a polytope. $extr(\Phi(\Delta))$, solutions of GLP, determine a finite set of constraints which defines the projection of S. The coordinates of the extreme points of $\Phi(\Delta)$ are the coefficients of a set of constraints that define the projection. The objective function in the usual linear program can be viewed as a mapping from R^n to R, the image of the polyhedron defined by the constraints being an interval in R. The optimization consists in finding a maximum or a minimum, that is one of the extreme points of the interval. In a GLP, the objective function represents a mapping from R^n to R^m and instead of looking for one extreme point, we look for the set of all extreme points. At the operational level, we can execute this GLP by generalizing the simplex method. The extreme points of $\Phi(\Delta)$ are images of extreme points of Δ. So we compute the set of extreme points of Δ, map them by Φ and eliminate the images which are not extreme points. It is important to note that although the extreme points method is better that Fourier in general because it eliminates the costly intermediate steps, there are still two main problems: the computation of the extreme points of Δ can be extremely costly even when the size of the projection is small and also the method produces a highly redundant output [1].

The convex hull method Variable elimination has long been treated as algebraic manipulations based on the syntax of the constraints rather than their semantics. Fourier's Procedure and EPM are no exceptions. Consequently, the complexity of these methods is tied to the initial polyhedral set instead of to the projection itself. Quantifier elimination can also be viewed as an operation of projection. Exploiting this remark in a systematic way leads to more output bound algorithms which guarantee an output when its size is reasonable and an approximation otherwise [9]. In the bounded case, the idea is trivial: by running linear programs we compute constraints whose supporting hyperplanes bound both the polytope to be projected and its projection. The traces of these hyperplanes on the projection space provide an approximation containing the projection. At the same time the extreme points provided by the linear programs project on points of the projection. The convex hull of these points is a polytope that is included in the projection. Iterating this process leads to the projection. Whether we have an output bound algorithm or not will however depend on the choice of points. The difficulties that remain are that we do not want to make any assumption on the input polyhedral set which can be bounded or not, full dimensional or not, redundant or not, empty or not. Standard linear programming techniques can be used to determine solvability and to transform the input if required into a set of equations defining its affine hull and a set of inequalities defining a full-dimensional polyhedral set in a smaller space. A straightforward variable elimination in the set of equations gives the affine hull of the projection which will be part of the final output.

This simplification based on geometrical considerations allows us to eliminate as many variables as possible by using only linear programming and gaussian elimination before getting into the costly part of elimination.

In the bounded case, the algorithm works directly on the input constraints. The projection is computed by successive refinements of an initial approximation obtained by computing with linear programs enough extreme points of the projection so that their convex hull is full-dimensional. Successive refinements consist in adding new extreme points and updating the convex hull. The costly convex hull construction is done in the projection space thus the main complexity of the algorithm is linked to the size of the output. The process stops when either the projection has been found or the size of the approximation has reached a user-supplied bound.

In the unbounded case, the problem is reformulated using the generalised linear program representation which is bounded by definition. $\Phi(\Delta)$ is computed by projection. The output will consist of the convex hull of $\Phi(\Delta)$ but also the set of its extreme points, from which the constraints defining the projection are derived. The advantage over the extreme points method is that we compute directly the extreme points of the projection. We do not need to compute the extreme points of Δ, this computation being the source of enormous intermediate computation and high redundancy in the output.

Implicit equalities and causes of unsolvability Fourier's algorithm can be used to trace all subsets of constraints in S that cause unsolvability or that are implicit equalities [11].

By using the *quasi-dual* formulation, we can acheive the same effects by running linear programs. The quasi-dual formulation which corresponds to Fourier's algorithm is

$$\Phi: \ \beta = b^T\lambda$$
$$\Delta: \begin{cases} A^T\lambda = 0, \\ \Sigma\lambda_i = 1, \\ \lambda_i \geq 0 \ \forall i. \end{cases}$$

Here Φ maps $\Re^m$ to $\Re$, where m is the number of constraints in S. Since we want to compute the minimum of Φ subject to Δ we need to solve the following linear program D:

$$\begin{array}{ll} \text{minimize} & b^T\lambda \\ \text{subject to} & A^T\lambda = 0 \\ & \Sigma\lambda_i = 1 \\ & \lambda_i \geq 0 \ \forall i. \end{array}$$

It is obvious that, in general, solving S in this manner is far more efficient than using Fourier's algorithm. Since D is a variant of the dual simplex in Linear Programming, it inherits nice properties from the standard dual simplex such as good incremental behavior, no need to introduce slack variables and no restriction to positive variables. More importantly as a side effect of the solvability test we obtain information about the algebraic properties of the constraints and about the geometric structure of the associated polyhedron. The properties of D are summarized in the table.

Quasi-Dual D	Properties of S
Unsolvable	• *Strongly* solvable • Full dimensional • No implicit equalities • Unbounded and • no projection has parallel facets • Inscribed spheres have unbounded radius
$min(b^T\lambda) > 0$	• Solvable • Full dimensional • No implicit equalities • Bounded or • exists projection with parallel facets • Inscribed spheres have bounded radius
$min(b^T\lambda) = 0$	• *Weakly* Solvable • Not full dimensional • Exists implicit equalities • An evident minimal subset of implicit equalities • Inscribed spheres have radius zero
$\exists\lambda : b^T\lambda < 0$	• Unsolvable • An evident minimal infeasible subset

Conclusion Much of the existing work on constraints has been done in diverse domains with their own distinctive requirements. Even in the restricted domain of linear arithmetic constraints, there is a wealth of knowledge and algorithms. To build systems to reason with constraints requires borrowing and synthesizing various notions and this led to the emerging concept of a unified framework of a single representation, the parametric query, and solution technique, variable elimination, for handling all the different operations on constraints. This approach shares key aspects with Logic Programming, with variable elimination playing the rule of resolution. The viability of this approach, both from a knowledge representation and knowledge processing aspects, is beeing tested with applications in the domain of spatial reasoning [3] and graphic user-interface [4]. Empirical results with an initial implementation have shown that a variety of small (about a hundred inequalities in two dimensions) and fairly large problems (up to about 2,000 inequalities over 70 variables) can be processed in times ranging from less than a second to a few minutes. Ongoing work includes the design and implementation of an integrated system based on the proposed framework and incorporating several solvers. The potential applicability of more recent interior points method is also investigated. Many properties of linear arithmetic constraints hold for constraints in other domains. These properties have been abstracted and generalized in [13].

References

[1] T. Huynh, C. Lassez and J-L. Lassez, Practical Issues on the Projection of Polyhedral Sets, *to appear Annals of Maths and AI.*

[2] T. Huynh, C. lassez and J-L Lassez, Fourier Algorithm Revisited, *2nd International Conference on Algebraic and Logic Programming* Springer-Verlag Lecture Notes in Computer Sciences, 1990.

[3] T. Huynh, L. Joskowicz, C. Lassez and J-L. Lassez, Reasoning About Linear Constraints Using Parametric Queries, *in Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Sciences*, Springer-Verlag vol. 472 December 1990.

[4] R. Helm, T. Huynh, C. Lassez and K. Marriott, A Linear Constraint Technology for User Interfaces, to appear *Proceedings of Graphics Interface'92*

[5] J. Jaffar and J.L. Lassez, Constraint Logic Programming, *Proceedings of POPL 1987*, Munich.

[6] P. Kanellakis, G. Kuper and P. Revesz, Constraint Query Languages, *Proceedings of the ACM Conference on Principles of Database Systems*, Nashville 90.

[7] J-L. Lassez, Querying Constraints, *Proceedings of the ACM conference on Principles of Database Systems*, Nashville 1990.

[8] J-L Lassez, Parametric Queries, Linear Constraints and Variable Elimination *Proceedings of DISCO 90*, Springer-Verlag Lecture Notes in Computer Sciences.

[9] C. Lassez and J-L. Lassez, Quantifier Elimination for Conjunctions of Linear Constraints via a Convex Hull Algorithm, IBM research Report, T.J. Watson Research Center, RC 16779 (1991), *to appear Academic Press.*

[10] J-L. Lassez, T. Huynh and K. McAloon, Simplification and Elimination of Redundant Arithmetic Constraints, *Proceedings of NACLP 89*, MIT Press.

[11] J-L. Lassez and M.J. Maher, On Fourier's Algorithm for Linear Arithmetic Constraints, IBM Research Report, T.J. Watson Research Center, RC 14114 (1988). To appear *Journal of Automated Reasoning.*

[12] J-L. Lassez and K. McAloon, A Canonical Form for Generalized Linear Constraints, IBM Research Report, T.J. Watson Research Center, RC 15004 (1989), to appear *Journal of Symbolic Computation.*

[13] J-L Lassez, K. McAloon, A Constraint Sequent Calculus *LICS 90.* Philadelphia.

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

[15] V. Saraswat, Concurrent Constraint Logic Programming, *to appear MIT Press.*

[16] L. Van Den Vries, Alfred's Tarski's Elimination Theory for Closed Fields, *The Journal of Symbolic Logic*, vol.53 n.1, March 1988.

[17] P. van Hentenryck, Constraint Satisfaction in Logic Programming, *The MIT Press*, 1989.