Independence of Negative Constraints

J.L. Lassez
IBM T.J. Watson Research Center

K. McAloon*
Brooklyn College and CUNY Graduate Center

Abstract

The independence of negative constraints is a recurring phenomenon in logic programming. This property has in fact a natural interpretation in the context of linear programming that we exploit here to address problems of canonical representations of positive and negative linear arithmetic constraints. Independence allows us to design polynomial time algorithms to decide feasibility and to generate a canonical form, thus avoiding the combinatorial explosion that the presence of negative constraints would otherwise induce. This canonical form allows us to decide by means of a simple syntactic check the equivalence of two sets of constraints and provides the starting point for a symbolic computation system. It has, moreover, other applications and we show in particular that it yields a completeness theorem for constraint propagation and is an appropriate tool to be used in connection with constraint based programming languages.

*Research partially supported by NSF Grant CCR-8703086

1 Introduction

Programming with constraints is an increasingly important element in research on programming languages and implementations. In languages such as CONSTRAINTS [Steele,Sussman] and THINGLAB [Borning] constraint solving has been used as a tool for declarative programming and for knowledge representation. Constraint solving is also used to drive rule-based systems such as ISIS [Fox] and MOLGEN [Stefik].

The constraint paradigm has also emerged in a significant way in logic programming. Thus in [Jaffar, Lassez] the Constraint Logic Programming scheme of languages is introduced which provides a formal framework for constraint programming. Moreover, the constraint point of view represents a significant extension of the logic programming paradigm beyond the original resolution based framework and is an important extension of the range of declarative programming [Clark]. Several such languages have been successfully implemented. The language CLP($\mathbf{R}$) for constraint programming over the real numbers has been implemented by [Jaffar, Michaylov] and employed in applications to decision support systems [Huynh,C. Lassez]. Colmerauer and his group have developed Prolog III which supports boolean and linear arithmetic constraints [Colmerauer 2]. The constraint logic programming system CHIP [Dincbas et al] has had very successful industrial applications.

At the same time, there have been dramatic mathematical developments in constraint solving ranging from the work of Karmakar [Karmakar] and others on interior point methods in linear programming to new work on the existential theory of $\mathbf{R}$ [Canny],[Renegar],[Grigor'ev,Vorobjov]. New avenues of research have also been opened by the use of 'analog' techniques such as simulated annealing and neural nets, *e.g.* [McClelland,Rumelhart]. Recent papers such as [Davis], [Pearl], [Dechter,Pearl] contain new developments in AI motivated constraint based techniques and on connections between AI and OR work.

Clearly the design and implementation of languages of the CLP scheme will make heavy use of work on constraint solving from various areas. However, the requirements on an arithmetic constraint solver in the OR or AI context are quite different from those that are imposed on a constraint solver which is embedded in a general purpose programming language. Designing languages to solve constraint satisfaction problems or optimization problems is a fundamentally different task from using the constraint paradigm to design programming languages. Thus problems of considerable importance for a CLP system such as the equivalence of sets of constraints do not appear to have attracted much attention in Operations Research or Artificial Intelligence, the work of Bradley [Bradley] and of Adler [Adler] being notable exceptions. In this paper we build on experience from the fundamental concepts and algorithms of logic programming itself and show how the 'independence of negative constraints' can be used in the context of an extended class of linear arithmetic constraints to develop a polynomial time canonical form algorithm for use in CLP languages. Moreover, this analysis leads to a completeness theorem for constraint propagation that should prove applicable in other contexts as well.

2 Independence and Logic Programming

The *independence of negative constraints* is a recurrent phenomenon in logic programming which we can describe schematically as follows: we are given 'positive' constraints $P_1, \ldots, P_n$ and other 'positive' constraints $Q_1, \ldots, Q_m$ which have 'negative' counterparts $\overline{Q_1}, \ldots, \overline{Q_m}$ and are asked to determine the feasibility of the combined system of positive and negative constraints $P_1, \ldots, P_n, \overline{Q_1}, \ldots, \overline{Q_m}$. This combined system is not feasible iff any solution that simultaneously satisfies the $P_1, ..., P_n$ must also satisfy at least one of the Q_j; in logical notation, this system is not feasible iff $P_1, \ldots, P_n \models Q_1 \vee \ldots \vee Q_m$. We say that negative constraints are independent if whenever $P_1, \ldots, P_n \models Q_1 \vee ... \vee Q_m$ then for some j_0 we have $P_1, \ldots, P_n \models Q_{j_0}$. When negative constraints are independent in this sense, it follows that verifying the consistency or feasibility of $P_1, \ldots, P_n, \overline{Q_1}, \ldots, \overline{Q_m}$ reduces to the simultaneous verification of the feasibility of the constraints $P_1, ..., P_n, \overline{Q_j}$.

The phenomenon of independence of negative constraints is central to logic programming. In fact, if $P_1, ..., P_n$ are definite Horn clauses and if $Q_1, ..., Q_m$ are elements of the Herbrand base, then indeed $P_1, ..., P_n \models Q_1 \vee ... \vee Q_m$ if and only if for some j, we have $P_1, ..., P_n \models Q_j$. This follows since Horn formulas are closed under products of models; Historically, it was this very property of Horn formulas - preservation under products of models - that motivated the research of Horn, Keisler and others [Chang and Keisler].

Another example of this phenomenon was discovered by Colmerauer in his work on Prolog II. Colmerauer considered equations and inequations of the form $s_1 = t_1, ..., s_n = t_n, u_1 \neq v_1, ..., u_m \neq v_m$ where the t_i, s_i, u_j, v_j are terms. In [Colmerauer], the independence of inequations in this context is established and used to develop an algorithm for deciding feasibility of equations and inequations for Prolog II. In [Lassez et al.] the concept of *dimension* was introduced in the term algebra context which brought together the results of Colmerauer on the independence of negative constraints, the work of Robinson on idempotent mgu's, and algebraic equation solving. It is the notion of dimension that also applies in the context of linear arithmetic constraints and which gives us the requisite independence property.

3 Independence and Linear Arithmetic Constraints

The language of *generalized linear constraints* is comprised first of *positive* constraints which are equations $ax = \beta$ and weak inequalities $ax \leq \beta$. Here a denotes an n-dimensional vector of real numbers, x denotes an n-dimensional vector of variables, β denotes a real number and juxtaposition denotes inner product. A basic *negative* constraint is a disjunction of inequations $a_i x \neq \beta_i, i = 1, \ldots, n$. Using DeMorgan's Law and matrix notation, a negative constraint can be written $\overline{\{Ax = b\}}$ which denotes the set of points x which lie in the complement of the affine space defined by the equations $Ax = b$. Conjunctions of negative constraints can be written $\overline{\{A_j x = b_j\}}, j = 1, \ldots, n$; conjunctions of equality constraints will be written $Ax = b$ and similarly conjunctions of weak inequality constraints will be written $Ax \leq b$.

We also admit strict inequality constraints $ax < b$. In matrix form conjunctions of strict inequality constraints are written $Ax < b$. This is a hybrid form of constraint in that it can be reduced to the combined positive and negative constraints $Ax \leq b$, $\overline{\{a_i x = \beta_i\}}$, $i = 1, \ldots, n$.

Thus a set of generalized linear constraints consists of positive constraints $Ex = f$ and $Ax \leq b$, strict inequality constraints $Gx < h$ and negative constraints $\overline{\{C_j x = d_j\}}, j = 1 \ldots n$.

By way of example, the constraints $z \geq 0, x - y + z < 1, x \geq 0, y \leq 0, \overline{\{y = 0, z = 0\}}$ define a wedge shaped polytope with a facet and an edge removed.

We want to develop efficient algorithms for testing the feasibility of a set of generalized linear constraints and to generate a canonical representation of such constraint sets. The presence of negative constraints introduces new problems. The point sets defined by the constraints are no longer convex sets and so the methods of convex analysis and of linear programming do not apply directly. Negative constraints themselves are *disjunctions* of inequations which enhances the expressive power of the constraint sets but which complicates the combinatorics of the situation. The key to dealing with these problems is the independence of negative constraints.

As mathematical preliminaries, we require basic results on polyhedral sets [Schrijver]. We establish the independence of negative constraints as well as a theorem which states that the solution set defined by a system of generalized linear constraints has a unique factorization into positive and negative components. For both results, dimensionality arguments play an essential role.

Theorem 1 (Independence of Negative Constraints) *A system $Ex = f, Ax \leq b$, $\overline{C_j x = d_j}, j = 1 \ldots n$ of constraints is feasible if and only if for each j_0, the subsystem $Ex = f$, $Ax \leq b$, $\overline{\{C_{j_0} x = d_{j_0}\}}$ is feasible.*

The next result requires two definitions. Suppose that $Ex = f, Ax \leq b, \overline{\{C_j x = d_j\}}, j = 1 \ldots n$ is a feasible set of generalized linear constraints and that the positive constraints define the polyhedral set P; then a negative constraint $\overline{\{C_i x = d_i\}}$ is said to be *relevant* if $\{x : C_i x = d_i\} \cap P$ is not empty. Suppose that P is a polyhedral set defined by a system of positive constraints; a negative constraint $\overline{\{Cx = d\}}$ is said to be *P-precise* if $\overline{\{Cx = d\}}$ is relevant and $\{x : Cx = d\} = Aff(P \cap \{x : Cx = d\})$, where Aff denotes affine closure.

Theorem 2 (Unique Factorization) *Suppose that two systems of generalized linear constraints define the same non-empty solution set. Then the positive constraints in the two systems define the same polyhedral set P. Moreover if the negative constraints in both systems are P-precise, then the complements of the negative constraints in the two systems define the same union of affine sets.*

4 Canonical Representation

Next we define the canonical representation of a set of generalized linear constraints and describe the algorithms to compute it, including an algorithm to decide feasibility.

A set of linear equations

$$y_1 = a_{1,1}x_1 + \ldots + a_{n,1}x_n + c_1$$
$$\vdots$$
$$y_m = a_{m,1}x_1 + \ldots + a_{m,n}x_n + c_m$$

is said to be in *solved form* if the variables $y_1, \ldots, y_m$ and $x_1, \ldots, x_n$ are all distinct. The variables $y_1, \ldots, y_m$ are called the *eliminable variables* and the variables $x_1, \ldots, x_n$ are called the *parameters* of the solved form. Further, we will say that a negative constraint $\overline{\{Cx = d\}}$ is in solved form if the complementary equality constraint $Cx = d$ is given in solved form.

A set of generalized linear constraints is in *canonical* form if it is non-redundant and consists of (1) a set of equations in solved form with parameters x which define the affine hull of the solution set (2) a set of inequality constraints $Ax \leq b$ which define a full dimensional polyhedral set P in the parameter space and (3) a set of P-precise negative constraints in solved form.

A system of generalized linear constraints in canonical form is thus partitioned into three modules (E,I,N) where E is a set of equality constraints, I is a set of weak inequality constraints and N is a set of negative constraints. What we develop is an algorithm CanForm that maps generalized linear constraints to triples of this form in such a way that if two constraint sets define the same solution set they are mapped to the same triple (E,I,N).

By way of example, consider the constraints in four variables $x_1 + x_3 \leq x_4$, $x_1 + x_3 \leq 10$, $x_4 \leq x_3, x_2 \leq x_3 + x_4, x_3 \leq x_1 + x_4, 0 \leq x_2, \overline{\{x_4 = 0\}}$. The CanForm procedure will return

$$E = \{x_1 = 0, x_3 = x_4\}$$
$$I = \{x_4 \leq 10, x_2 - 2x_4 \leq 0, -x_2 \leq 0\}$$
$$N = \{\overline{\{x_2 = 0, x_4 = 0\}}\}$$

The constraints thus define a two dimensional point set, a triangle with a vertex removed.

In the definition of canonical form no mention is made of strict inequality constraints. As noted above, each strict inequality constraint $e_k x < \phi_k$ can be replaced by the pair $e_k x \leq \phi_k, \overline{\{e_k x = \phi_k\}}$. From the algorithmic point of view, we can suppose that this transformation has been made throughout; we return to this point later and show how to restore the strict inequality constraints at the end of the simplification process.

The independence property allows us to avoid the combinatorial explosion that the presence of negative constraints would normally introduce and also allows for a large degree of parallelism in the treatment of the negative constraints. Moreover, we show that the positive and negative constraints can be separated out in the treatment of redundancy. This will serve to reduce eliminating redundancy among negative constraints to a parallel sieving process. An important step in the algorithm is the computation

of the affine hull of the solution set defined by the constraints. This allows for the replacement of Linear Programming routines by Gaussian Elimination for part of the feasibility check in the algorithm. Also, going to the affine hull brings us to a full dimensional situation. In order to overcome the sequentiality that is typically found when eliminating redundancy [Karwan et al], we introduce a classification of redundant inequality constraints, which combined with affine hull arguments leads to a procedure with highly decomposable parallelism for eliminating redundancy. For further details see [Lassez, McAloon].

In the theorem that follows, the uniqueness of the equality constraints and negative constraints depends on a fixed ordering of the variables in the system.

Theorem 3 (Canonical Form Theorem) *If two sets of constraints define the same solution set the canonical form procedure returns the same equations to define the affine hull, the same inequality constraints (up to multiplication by positive scalars) and the same set of negative constraints.*

At this point, if strict inequality constraints are to be returned in the canonical form, let us note that a strict inequality corresponds to a pair $(ax \leq \beta, \overline{\{cx = \delta\}})$ where $ax \leq \beta$ is a positive weak inequality constraint in the canonical form and $\overline{\{cx = \delta\}}$ is a negative constraint such that the vector c, δ is a scalar multiple of the vector a, β. This pair can then be replaced by the strict inequality constraint $ax < \beta$. As in the use of linear programming and Gaussian elimination in the decision procedures of the CanForm algorithm, here too sufficient precision arithmetic is required. This variant on the canonical form algorithm is a natural one in the context of symbolic processing of generalized linear constraints and in the context of output constraints where strict inequality information can be significant.

If, in the example above, the constraint $0 \leq x_2$ is sharpened to $0 < x_2$, then after transforming this constraint into the pair $0 \leq x_2, \overline{\{x_2 = 0\}}$, the canonical form procedure will return

$$E = \{x_1 = 0, x_3 = x_4\}$$
$$I = \{x_4 \leq 10, x_2 - 2x_4 \leq 0, -x_2 \leq 0\}$$
$$N = \{\overline{\{x_2 = 0\}}\}$$

The negative constraint $\overline{\{x_2 = 0, x_4 = 0\}}$ has been eliminated because it is now redundant. Since the vector $(1, 0, 0)$ is a scalar multiple of $(-1, 0, 0)$ the constraints $-x_2 \leq 0$ and $\overline{\{x_2 = 0\}}$ can be replaced by the strict inequality constraint $-x_2 < 0$.

5 Computational Complexity

In this section we address considerations of computational complexity for the canonical form algorithm. We have

Theorem 4 *The canonical form procedure CanForm is a polynomial time algorithm.*

Moreover, in terms of the PRAM model of parallel computation, the processor complexity of the algorithm is bounded by the number of constraints and its time complexity by the sequential complexity of linear programming. Further, the number of arithmetic operations in the algorithm is determined by Gaussian elimination and linear programming; thus if a strongly polynomial algorithm were to be found for linear programming, then the canonical form problem would also enjoy a strongly polynomial solution. On the other hand, it is not to be expected that an NC algorithm can be found: the canonical form problem subsumes the Phase I or Feasibility Problem of linear programming and this is known to be P-Complete.

The algorithm can be applied to give polynomial time procedures for computing two normal forms for systems consisting of equality and weak inequality constraints. In [Schrijver] a normal form is defined for a set of positive linear constraints which involves computing the affine hull of the polyhedral set defined by the constraints and representing this set in terms of its facets; in the non-full-dimensional case, the facets are defined by means of supporting hyperplanes in the full space chosen orthogonal to the affine hull. We can also show that the normal form of [Adler] for linear programs with objective function is polynomial time computable using our algorithm. Adler's normal form, the *core* of a linear program involves both the program and its dual and both the primal and dual constraint sets are analyzed in the process.

6 Constraint Propagation

Finally, we consider the canonical form in the context of constraint propagation and constraint based programming.

Linear arithmetic constraints arise naturally in constraint programming and in AI situations. When determining the solvability and/or the solutions to a set of constraints, a standard strategy is to work forward in an incremental way by starting with the 'most constraining' conditions and propagating these constraints throughout the rest of the computation. Intuitively a 'more constraining' condition corresponds to a set of smaller dimension in the solution space, information which is in general not explicit in a given system S of constraints. The canonical form procedure returns in *explicit* form a representation of equality, inequality and negative information in packets of smallest possible dimension. In fact, the canonical form leads to a soundness and completeness theorem for constraint propagation.

Theorem 5 (Constraint Propagation Completeness Theorem) *Let S be a consistent system of generalized linear constraints with canonical form (E, I, N). Let y be the eliminable variables and x the parameters of (E, I, N). Then we have*

(1) $S \models ay = bx$ *iff* $E \models ay = bx$

(2) $S \models \overline{ax \leq \beta}$ *iff* $I \models \overline{ax \leq \beta}$

(3) $S \models \overline{\{Cx = d\}}$ *iff* $N \models \overline{\{Cx = d\}}$

where $\overline{\{Cx = d\}}$ *is a precise negative constraint.*

This theorem also has application to constraint logic programming languages in the CLP class [Jaffar and Lassez]. In particular, for CLP(**R**) [Jaffar and Michaylov] and Prolog III, it shows how the canonical form is the analog in the constraint context of the most general unifier.

Bibliography

[Adler]
I. Adler, The Core of a Linear Program, Technical Report, Department of Operations Research, Berkeley
[Borning]
A. Borning, The Programming Language Aspects of THINGLAB, a Constraint Oriented Simulation Laboratory, *ACM Transactions on Programming Languages and Systems* 3 (1981) 252-387
[Bradley]
G. Bradley, Equivalent Integer Programs, *Proceedings of the Fifth International Conference on Operations Research*, ed. J. Lawrence, Venice 1969
[Canny]
J. Canny, Some Algebraic and Geometric Computations in PSPACE, *STOC*, 1988
[Chang and Keisler]
C.C. Chang and H.J. Keisler, *Model Theory*, North-Holland 1973
[Clark]
K. Clark, Logic Programming Schemes, *Proceedings of 1988 FGCS Conference*, Tokyo
[Colmerauer]
A. Colmerauer, Equations and Inequations on Finite and Infinite Trees, *Proceedings of 1984 FGCS Conference*, Tokyo
[Colmerauer 2]
A. Colmerauer, An Introduction to Prolog III, Technical Report, Groupe d'Intelligence Artificielle (1987)
[Davis]
E. Davis, Constraint Propagation, *Artificial Intelligence* 1988
[Dechter,Pearl]
R. Dechter and J. Pearl, Network-based Heuristics for Constraint Satisfaction Problems, *Artificial Intelligence*, 34 (1987) 1-38
[Dincbas et al]
M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf and F. Berthier, The Constraint Logic Programming Language CHIP, *Proceedings of the 1988 FGCS Conference*, Tokyo
[Fox]
M. Fox, *Constraint Directed Search: A Case Study of Job-Shop Scheduling*, Morgan Kaufmann, 1988
[Grigor'ev,Vorobjov]
D. Grigor'ev and N. Vorobjov, Solving Systems of Polynomial Inequalities in Subexpo-

nential Time, *Journal of Symbolic Computation* 5 (1988) 37-64
[Huynh,C. Lassez]
T. Huynh and C. Lassez, A CLP(**R**) Options Analysis System, *Proceedings of the 1988 Logic Programming Symposium*
[Jaffar, Lassez]
J. Jaffar and J.L. Lassez, Constraint Logic Programming, *Proceedings of POPL 1987*, Munich
[Jaffar, Michaylov]
J. Jaffar and S. Michaylov, Methodology and Implementation of a CLP System, *Proceedings of the 1987 Logic Programming Conference*, Melbourne, MIT Press
[Karwan et al.]
M.H. Karwan, V. Lofti, J. Telgen and S. Zionts, *Redundancy in Mathematical Programming*, Lecture Notes in Economics and Mathematical Systems 206, Springer-Verlag 1983
[Lassez et al.]
J.L. Lassez, M. Maher and K. Marriott, Unification revisited, *Foundations of Deductive Databases and Logic Programming*, J. Minker editor, Morgan Kaufmann 1988
[Lassez, McAloon]
J-L. Lassez and K. McAloon, Applications of a Canonical Form for Generalized Linear Constraints, *Proceeding of FGCS 1988*, Tokyo.
[McClelland,Rumelhart]
J. McClelland and D. Rumelhart, *Explorations in Parallel Distributed Processing*, MIT Press, 1988
[Pearl]
J. Pearl, Constraints and Heuristics, *Artificial Intelligence* 1988
[Renegar]
J. Renegar, A Faster PSPACE ALgorithm for Deciding the Existential Theory of the Reals, *FOCS 1988*, pp 291-285
[Schrijver]
A. Schrijver, *Theory of Linear and Integer Programming*, Wiley 1986
[Steele,Sussman]
G. Steele and G. Sussman, CONSTRAINTS - A Language for Expressing Almost Hierarchical Descriptions, *Artificial Intelligence* 1980
[Stefik]
M. Stefik, Planning with Constraints (MOLGEN: Part 1), *Artificial Intelligence* 16 (1984) 111-140

J-L. Lassez
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights NY 10598

K. McAloon
Logic Based Systems Lab
Brooklyn College CUNY
Brooklyn NY 11210