June, 1998

LIDS- P 2421

Research Supported By:

Submitted to Journal of Automated Reasoning
US Army grant DAAL03-92-G-0115, CICS
Grant from Siemens AG

Mathematical Programming Embeddings of Logic

Borkar, V.S.
Chandru, V.
Micciancio, D.
Mitter, S.K.

Mathematical Programming Embeddings of Logic

Vivek S. Borkar
Department of Computer Science and Automation
Indian Institute of Science
Bangalore 560 012 INDIA

Vijay Chandru
Department of Computer Science and Automation
Indian Institute of Science
Bangalore INDIA 560012

D. Micciancio
Electrical Eng & Computer Science
School Of Engineering
Massachusetts Institute of Technology
Cambridge
Massachusetts 02139

Sanjoy K. Mitter
Laboratory for Information and Decision Systems
Massachusetts Institute of Technology
Cambridge
Massachusetts 02139

February 20th, 1998

Research supported by US Army grant DAAL03-92-G-0115, Center for Intelligent Control Systems

Mathematical Programming Embeddings of Logic

V S Borkar V Chandru D Micciancio S K Mitter

September, 1996

1 Introduction

Serious studies on spatial embeddings of logic were initiated by Robert Jeroslow (cf. [16]) a little over a decade ago. He essentially showed that, by posing inference in logic as mathematical programming problems, we open up the possibility of transfer of methodology from the geometric techniques of mathematical programming to the symbolic world of computational logic. Further, he demonstrated that these two perspectives can generate a symbiosis that adds both in structural insights and effective algorithm design for inference.

These embeddings have yielded beautiful structural results relating forward and backward chaining of logic with linear programming relaxations [4], resolvents with cutting planes[10,2] and explanation of inference with mathematical programming duality [16,27]. It has led us to exciting discoveries of new special structures in propositions such as Extended Horn [6] and Balanced Propositions [7]. Embeddings of logics of uncertainty (propositional logic plus various models of probabilistic and evidential reasoning) have also been effectively formulated as large scale linear programs [1,18, 21].

The embedding of first order (predicate) logical inference as mathematical programs was also initiated by Jeroslow [15]. His approach, and that of subsequent work on this topic [8,11,17], was to partially ground the predicate formula (as a partial Herbrand extension) to a propositional CNF formula and hence as a mathematical program and to use dynamic activation to resolve the ensuing unification conflicts. The emphasis has been on using the embedding for theorem proving and not for structural insights.

In this paper, we introduce new embeddings of inference in modal/temporal, predicate and partially interpreted logics, that provide for structural analyses. The framework is that of finite and infinite mathematical programs. We also describe applications of these embeddings. Using standard techniques of topology we show that Herbrand's Theorem is a simple consequence of compactness in certain infinite integer programs. Since Herbrand's Theorem is the cornerstone of first order theorem proving, we believe that our framework therefore provides a handle on the structural aspects of theorem proving. We are also able to prove the unique "minimal" model property of first order Horn logic via infinite dimensional linear programming, and thereby provide a new foundation for analyzing model theory of logic programming.

While the use of the embedding to analyze the Herbrand Extension of a first order formula is a nice application, the real challenge would be to "liberate" theorem proving from the clutches of the restrictive (and sometimes unnatural) Herbrand universe and yet maintain the semi-decidable complexity of theorem proving. The framework we provide offers a glimmer of hope for accomplishing this objective as our compactness theorems apply to infinite mixed integer programs whose constraints and variables need not be denumerable. We use this capability to show that inference in the constraint logic programming language CLP($\Re$) embeds as an infinite dimensional linear program.

An important application of logic is that it formalizes languages to describe and reason about computer programs. From this point of view, it would be interesting to see to what extent the spatial embeddings studied for propositional logic can be extended to other logic languages, such as dynamic logic [23] and process logic [22]. Finding embeddings of dynamic logic in the style of [5] is presumably a hard problem because of some non-compactness results that affect that logic. We have started exploring the feasibility of such embeddings by choosing a fairly simple subset of dynamic logic, namely modal logic.

In the next section we present the spatial embeddings. Section 3 contains the main compactness theorems. We then proceed, in section 4, to address the mathematical programming of logic programs (Horn formulas) in the Herbrand setting and embeddings of partially interpreted logics such as CLP($\Re$). We conclude with some remarks on the constructiveness of these frameworks and their possible application in hybrid systems modeling.

2 Embeddings

A fundamental problem in logic is determining whether a formula is satisfiable, i.e. there exists a valuation for the variables occurring in the formula that makes the whole formula true. Logical deduction can be easily reduced to satisfiability: formula ϕ is a logical consequence of a set of formulas A if and only if the set of formulas $A \cup \{\neg\phi\}$ is unsatisfiable. Therefore algorithms to decide the satisfiability of formulas can immediately be turned into procedures for logical deduction and automated reasoning.

2.1 Propositional

Satisfiability, the basic inference problem of propositional logic uses symbolic valuations of atomic propositions as either *True* or *False*. Mathematical programming, however, works with numerical valuations. Therefore, in order to usefully apply the methodology of mathematical programming to these inference problems, we need to embed them in familiar forms.

In $0-1$ linear programming, i.e. the solution of linear inequalities on $0-1$ variables, all the inequality constraints have to be satisfied simultaneously (in conjunction) by any feasible solution. It is natural therefore to formulate satisfiability of CNF propositions as $0-1$ linear programming models with clauses represented by constraints and atomic propositions represented by $0-1$ variables.

Consider, for example, the single clause

$$(x_2 \vee \neg x_3 \vee x_4)$$

The satisfiability of this clause is easily embedded as solubility of an inequality over (0,1) variables as follows.

$$x_2 + (1 - x_3) + x_4 \geq 1$$

It is conventional in mathematical programming to clear all the constants to the right hand side of a constraint. Thus a clause C_i is represented by $a_i x \geq b_i$ where for each j, a_{ij} is +1 if x_j is a positive literal in C_i, is -1 if $\neg x_j$ is a negative literal in C_i and is 0 otherwise. Also, b_i equals $(1 - n(C_i))$ where $n(C_i)$ is the number of negative literals in C_i. We shall refer to such inequalities as *clausal* . In general, satisfiability in propositional logic is equivalent to solubility of

$$Ax \geq b,\ x \in \{0,1\}^n \tag{1}$$

where the inequalities of $Ax \geq b$ are clausal. Notice that A is a matrix of 0's and $\pm 1's$ and each b_i equals 1 minus the number of -1's in row i of the matrix A. We are therefore looking for an extreme point of the unit hypercube in $\Re^n$ which is contained in all the half-spaces defined by the clausal inequalities. This is a *spatial* or geometric embedding of inference in propositional logic.

2.2 Embedding Modal Logic

Modal logic extends classical logic introducing new quantifiers over formulas, called modalities. The set of modalities may be different from one logic to the other. For example, dynamic logic can be viewed as a modal logic where the modalities are programs. Here we consider a special case of dynamic logic where programs are single atomic actions. More precisely the set of modalities is $\{\Box_a\}_{a\in\Sigma}$ (and their duals $\{\Diamond_a\}_{a\in\Sigma}$) where Σ is a set of symbols.

A model $M = (W, T, \sigma)$ for a modal formula ϕ is given by a set of *worlds* W, a family of transition functions $T = \{t_a : W \to 2^W\}_{a\in\Sigma}$ labelled by the symbols in Σ, and a valuation for the variables $\sigma : V \to 2^W$ that associate to each propositional variable the set of worlds in which the variable is true. Given a model $M = (W, T, \sigma)$ and a world s in W the truth value of a modal formula is defined by induction on the structure of the formula as follows:

- $M, s \models x$ iff $s \in \sigma(x)$,
- $M, s \models \phi \wedge \psi$ iff both $M, s \models \phi$ and $M, s \models \psi$,
- $M, s \models \phi \vee \psi$ iff either $M, s \models \phi$ or $M, s \models \psi$,
- $M, s \models \neg\phi$ iff not $M, s \models \phi$,
- $M, s \models \Box_a\phi$ iff $M, t \models \phi$ for all $t \in t_a(s)$,
- $M, s \models \Diamond_a\phi$ iff $M, t \models \phi$ for some $t \in t_a(s)$.

A formula ϕ is true in a model M, written $M \models \phi$, if ϕ is true in all worlds of M. A formula ϕ is satisfiable iff it is true in some model.

So far, we have defined a logic language that extends propositional logic and we have defined a notion of satisfiability for formulas in that languages. We want to embed the satisfiability of modal formulas into linear problems, as it has been done for propositional logic.

We would like to find a direct embedding of modal logic, that preferably preserves the finiteness property of propositional logic. The intuition behind the embedding that We am going to define is to use "timed" linear systems of the form

$$A_0x(t) + A_1x(t+1) \geq b$$

where the "time" t is used to express the "dynamic" associated to the modal operators. The above system is of a kind usually encountered in the study of dynamical systems and can be rewritten in a more compact way using a shift operator * as follows:

$$A_0x + A_1x^* \geq b.$$

Here variable x is a function of time t and the action of the shift operator on x is given by $x^*(t) = x(t+1)$.

In order to simplify the presentation in the rest of this section We will take $\Sigma = \{a\}$ so that we have only two modal operators $\Box$ and $\Diamond$ (the subsctipt a is omitted for brevity). However, everything We will say can be extended to the general case, with $|\Sigma|$ possibly greater than one, with obvious modifications.

Since the transition function t may associate to each world more than one successor (or even none), the dynamic expressed by the modal operators $\Box$ and $\Diamond$ has a branching structure. Therefore *time* is not the right concept to express the modalities. We will consider a notion of *generalized time* T. Variable x is still a function of T, but there are two shift operators $^\Box$ and $^\Diamond$ that can act over x. Putting it together, we want to embed the satisfiability problem for modal logic into a system of the kind

$$A_0x + A_1x^\Box + A_2x^\Diamond \geq b.$$

where the vector x is a function of T and the action associated to the shift operators $^\Box$ and $^\Diamond$ is the following. We have said that T can be thought as a "time" in a broad sense (carrying on this analogy, We call *istants* the elements of T). Each istant in T may have more than one immediate successor in T. Let τ be a function that associates to each istant the set of its immediate successors in T. The result $x^\Box$ of applying the $^\Box$ shift to x is the set of all possible values that vector x can take after one unit of "time". Analogously the result $x^\Diamond$ of applying the $^\Diamond$ shift to x is some of the possible values that vector x can take after one unit of "time".

Now we look at how modal formulas can be represented in this framework. Clearly any propositional formula that doesn't make use of the modalities can be embedded into a system with A_1 and A_2 both equal to the null matrix. Also, formulas that make very simple use of the modalities can be directly embedded. For example the formula

$$(x \vee \neg(\Diamond y \wedge z)) \wedge (\Box z \rightarrow z)$$

can be rewritten as

$$(x \vee \neg z \vee \Box\neg y) \wedge (\Diamond\neg z \vee z)$$

and then represented by the system

$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 1 \end{bmatrix} x + \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix} x^{\Box} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} x^{\Diamond} \geq \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

Things get harder if the formula makes a more complex use of modal operators. For example there is no direct way to express the formula $\Diamond\Box x$ directly into our system. It seems that the flat structure of the linear system does not allow us to represent nested modal operators. A more subtle problem arises when translating the formula $\Box x \vee \Box y$. One could be tempted to embed this formula into the system

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}^{\Box} \geq 1.$$

At first sight this seems correct but a more carefull exam shows that the meaning of the above system is the formula $\Box(x \vee y)$ which is not equivalent to $\Box x \vee \Box y$. In fact, the shift operator $^{\Box}$ acts on the vector $\begin{bmatrix} x \\ y \end{bmatrix}$ as a whole and therefore we cannot choose $x^{\Box}$ and $y^{\Box}$ independently of each other.

We will now illustrate an embedding technique that solves the above problems and allows the encoding of arbitrarily complex formulas into linear systems. The resulting system is finite, and its size is not significantly greater of the starting modal formula.

The method is based on the introduction of new variables associated to subexpressions of the logic formula and is defined as a recursive procedure **Embed**(ϕ). On input a formula ϕ of propositional modal logic, **Embed**(ϕ) returns a system of linear equations over the variables of ϕ, plus some fresh variables introduced during the execution of the procedure, whose solubility over $0-1$ variables is equivalent to the satisfiability of the origunal formula ϕ. First We define a procedure to embed formulas of the form $x \leftrightarrow \phi$:

Embed($x \leftrightarrow \phi$)

- if $\phi = x$, then return $\{x \geq 1\}$,
- if $\phi = \neg\psi$, then introduce a fresh variable z and return

$$\{-x - z \geq -1, x + z \geq 1\} \cup \mathbf{Embed}(z \leftrightarrow \psi)$$

- if $\phi = \psi_1 \wedge \psi_2$, then introduce two fresh variables z_1 and z_2 and return

$$\{-x + z_1 \geq 0, -x + z_2 \geq 0, x - z_1 - z_2 \geq -1\}$$

$$\cup\, \mathbf{Embed}(z_1 \leftrightarrow \psi_1) \cup \mathbf{Embed}(z_2 \leftrightarrow \psi_2)$$

- if $\phi = \psi_1 \vee \psi_2$, then introduce two fresh variables z_1 and z_2 and return

$$\{x - z_1 \geq 0, x - z_2 \geq 0, -x + z_1 + z_2 \geq 0\}$$

$$\cup\, \mathbf{Embed}(z_1 \leftrightarrow \psi_1) \cup \mathbf{Embed}(z_2 \leftrightarrow \psi_2)$$

- if $\phi = \Box\psi$, then introduce a fresh variable z and return

$$\{-x + z^{\Box} \geq 0, x - z^{\Box} \geq 0\} \cup \textbf{Embed}(z \leftrightarrow \psi)$$

- if $\phi = \Diamond\psi$, then introduce a fresh variable z and return

$$\{-x + z^{\Diamond} \geq 0, x - z^{\Box} \geq 0\} \cup \textbf{Embed}(z \leftrightarrow \psi)$$

The general case easily follows. Any formula ϕ can be embedded into the linear system $\{z \geq 1\} \cup \textbf{Embed}(z \leftrightarrow \phi)$ where z is a variable not occurring in ϕ.

Applying the function **Embed**$(\cdot)$ to the formula $\Diamond\Box x$ we get the system

$$\begin{aligned} z_1 &\geq 1 \\ -z_1 + z_2^{\Diamond} &\geq 0 \\ z_1 - z_2^{\Diamond} &\geq 0 \\ -z_2 + z_3^{\Box} &\geq 0 \\ z_2 - z_3^{\Box} &\geq 0 \\ -z_3 + x &\geq 0 \\ z_3 - x &\geq 0 \end{aligned}$$

Obviously we could have embedded the same formula in the smaller system

$$\begin{aligned} z^{\Diamond} &\geq 1 \\ -z + x^{\Box} &\geq 0 \\ z - x^{\Diamond} &\geq 0. \end{aligned}$$

However, even if the system obtained by applying **Embed**() is not the smallest possible, it can be formally proved the the result of the given procedure is never much bigger than necessary. Namely the system **Embed**(ϕ) has at most $3n+1$ rows where n is the size of the formula ϕ.

The last system can be written in matrix notation as

$$\begin{bmatrix} 0 & 0 \\ 0 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix}^{\Box} + \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix}^{\Diamond} \geq \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Here we see how the introduction of a new variable z allows us to represent a formula with nested modal operators.

Now consider the formula $\Box x \vee \Box y$, We have already remarked that this formula cannot be straightforwardly translated into the system

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}^{\Box} \geq 1$$

which in fact represent a different formula, namely $\Box(x \vee y)$. Let's see how expressions with multiple modal operators in the same clause are handled. The result of applying the embedding function to formula $\Box x \vee \Box y$ is the system

$$\begin{aligned} z_1 &\geq 1 \\ -z_1 + z_2 + z_3 &\geq 0 \\ z_1 - z_2 &\geq 0 \\ z_1 - z_3 &\geq 0 \\ -z_2 + z_4^\Box &\geq 0 \\ z_2 - z_4^\Box &\geq 0 \\ -z_3 + z_5^\Box &\geq 0 \\ z_3 - z_5^\Box &\geq 0 \\ -z_4 + x &\geq 0 \\ z_4 - x &\geq 0 \\ -z_5 + y &\geq 0 \\ z_5 - y &\geq 0 \end{aligned}$$

or with a few simplifications

$$\begin{aligned} z + y^\Box &\geq 1 \\ -z + x^\Box &\geq 0 \\ z - x^\Box &\geq 0. \end{aligned}$$

This last example shows how introducing a new variable z we can split a clause with multiple occurrences of the same modal operator into the conjunction of several clauses each of which contains at most one modal operator.

We have showed how any formula of modal logic can be translated into a "small" linear system of the form

$$A_0 x + A_1 x^\Box + A_2 x^\Diamond \geq b.$$

The equivalence of the system with the modal formula can be easily proved by induction on the size of the formula. The linear system has the same "clausal form" property shown in [5] for the propositional logic embedding. An other property enjoyed by this linear system is that each row of the matrices A_1 and A_2 has at most one non-null entry. It is because of this last property that the shift operators $^\Box$ and $^\Diamond$ can be applied to the unknown vector x as a whole, as opposed to being applied componentwise.

2.3 Modal Logic and Bisimulation

The relevance of modal logic in the context of modeling distributed computing is exemplified by its relationship with bisimulation, a widely accepted equivalence relation between labeled transition systems.

A labeled transition system is a graph whose edges are labeled with symbols from some alphabet Σ. Formally a labeled transition system is a tuple (N, E, L) where N is a set of nodes, E is a binary relation on N and L is a function from E to Σ. The nodes N represent the possible internal states of a process or set of processes, the labels Σ are actions the system may perform, and the edges of the graph E express how the internal state of the system changes following the execution of an action. Usually some node $s \in N$ is designated as the starting node, the initial state of the process represented by the transition system.

Two labelled transition systems (N_1, E_1, L_1, s_1) and (N_2, E_2, L_2, s_2) are bisimilar if there exists a binary relation $R \subseteq N_1 \times N_2$ such that

- $(s_1, s_2) \in R$
- for all $(t_1, t_2) \in R$:
  - if $(t_1, t_1') \in E_1$, then there exists some t_2' such that $(t_2, t_2') \in E_2$, $L_2(t_2, t_2') = L_1(t_1, t_1')$ and $(t_1', t_2') \in R$.
  - if $(t_2, t_2') \in E_2$, then there exists some t_1' such that $(t_1, t_1') \in E_1$, $L_2(t_1, t_1') = L_1(t_2, t_2')$ and $(t_1', t_2') \in R$.

It is natural to view labelled transition systems as models for modal logic. The nodes in the graph are the worlds of the model and the transition relation t_a maps node s to the set $\{t : (s, t) \in E\}$. We can ask when two labelled transition systems can be distinguished by modal formulas. In other words, given two labelled transition systems we look for some formula that is true in one system but false in the other. Two labelled transition systems are considered equivalent if no such formula exists.

It turns out that this notion of equivalence is exactly bisimilarity. Two labelled transition systems are bisimilar if and only if they satisfy the same set of modal formulas. For a formal proof of this statement together with a more accurate description of the relationship between modal logic and bisimulation the reader is referred to [3].

Here We will only illustrate the mentioned result on a simple scheduler example taken from [19]. The scheduler described in [19] communicates with a set $\{P_i\}_i$ of n processes through the actions a_i and b_i ($i = 1, \dots, n$). These actions have the following meaning:

- action a_i signals P_i has started executing,
- action b_i signals P_i has finished its performance.

Each process wishes to perform its task repeatedly. The scheduler is required to satisfy the following specification:

- actions $a_1, \dots, a_n$ are performed cyclically starting with a_1,
- action a_i and b_i are preformed alternatively.

$$\begin{tikzcd}
& C \arrow[ddr, "a_1"] & \\
& D \arrow[dl, "a_2"'] & \\
A \arrow[uur, "b_2"] \arrow[ddr, "a_1"'] & & B \arrow[ul, "b_1"'] \arrow[dl, "a_2"] \\
& E \arrow[ul, "b_1"] & \\
& F \arrow[uur, "b_2"'] &
\end{tikzcd}$$

Figure 1: Simple Scheduler Implementation

Informally processes start their task in cyclic order starting with P_1 and each process finish one performance before it begins another.

Then a modular implementation of the scheduler is suggested. The implementation is based on a set of n components $C_1, \ldots, C_n$ connected in cycle that pass a token each other in cyclic order. There is exactly one token, initially owned by C_1, going around. Furthermore, each component C_i performs action a_i after receiving the token and before passing it to $a_{(i \bmod n)+1}$. Then after the token has been passed to $a_{(i \bmod n)+1}$, C_i performs b_i before receiving the token again. For a more accurate description of this example the reader is referred to the original text [19, pages 113–123] where both the specification and the implementation of the scheduler are formally given using the *CCS* language.

If the number n of processes being scheduled equals two, the specification is given by the labelled transition system shown in figure 2, while the implementation gives the system described by the labeled transition system in figure 1.

If the system $[C_1|\ldots|C_n]$ were a correct implementation of the specification, the two systems in figure 1 and 2 would not be distinguishable by any modal formula. However this is not the case since formula $s \to \Box_{a_1}\Box_{a_2}\Diamond_{b_2}t^1$ is true in the system depicted in figure 2 but not in the one shown in figure 1. The formula $s \to \Box_{a_1}\Box_{a_2}\Diamond_{b_2}t$ can be translated into the linear system

$$\begin{aligned}
-s + x^{\Box_{a_1}} &\geq 0 \\
-x + y^{\Box_{a_2}} &\geq 0 \\
x - y^{\Diamond_{a_2}} &\geq 0 \\
-y + t^{\Diamond_{b_2}} &\geq 0 \\
y - t^{\Box_{b_2}} &\geq 0
\end{aligned}$$

[1]Here s is a predicate true only in the starting state and t is a predicate always true

$$\begin{tikzcd}
 & Spec(1,\{1\}) \arrow[dl, "b_1"'] & \\
Spec(1,\phi) \arrow[dr, "a_2"] & Spec(1,\{2\}) \arrow[l, "b_2"'] \arrow[dr, "a_2"] & Spec(1,\{1,2\}) \arrow[ul, "b_2"'] \arrow[l, "b_1"'] \\
Spec(2,\phi) \arrow[ur, "a_1"] & Spec(2,\{1\}) \arrow[l, "b_1"'] \arrow[ur, "a_1"] & Spec(2,\{1,2\}) \arrow[l, "b_2"'] \arrow[dl, "b_1"] \\
 & Spec(2,\{2\}) \arrow[ul, "b_2"] &
\end{tikzcd}$$

Figure 2: Simple Scheduler Specification

$$t \geq 0$$

which has solution

$$\begin{aligned}
s &= \{Spec(1,\emptyset)\}\\
x &= \{Spec(2,\{1\})\}\\
y &= \{Spec(1,\{1,2\})\}\\
t &= \{Spec(i,S) : i \in \{1,2\}, S \subseteq \{1,2\}\}
\end{aligned}$$

in the model associated to the system in figure 2 but has no solution in the model associated to the implementation. In conclusion the linear system shows that the proposed implementation of the scheduler does not satisfy the given specification.

2.4 Infinite Dimensional Embeddings of Predicate Logics

We will assume that the reader has some familiarity with the basic concepts of predicate logic. An excellent modern treatment of the constructs of logic that are useful for computer scientists is given in [26]. Predicate logic is an extension of propositional logic with the additional concepts of quantified variables, constants, functions and predicates.

Example 2.1 *As an illustration of the expressive power of predicate logic we show how modal logic is formulated as a special case of predicate logic by using quantification over variables to express modalities. More precisely, the following translation function can be used to associate to a modal formula ϕ a first order formula $\mathbf{f}(\phi)$.*

$$\begin{aligned}
\mathbf{f}(x) &= X(w)\\
\mathbf{f}(\neg\phi) &= \neg\mathbf{f}(\phi)\\
\mathbf{f}(\phi\wedge\psi) &= \mathbf{f}(\phi)\wedge\mathbf{f}(\psi)\\
\mathbf{f}(\phi\vee\psi) &= \mathbf{f}(\phi)\vee\mathbf{f}(\psi)\\
\mathbf{f}(\Box_a\phi) &= \forall v.(R_a(w,v)\Rightarrow\mathbf{f}(\phi)[v/w])\\
\mathbf{f}(\Diamond_a\phi) &= \exists v.(R_a(w,v)\wedge\mathbf{f}(\phi)[v/w])
\end{aligned}$$

Here, variables w, v range over worlds, a unary predicate X is introduced for each propositional variable x and a family of binary predicate $\{R_a\}_{a\in\Sigma}$ is used to represent the transition functions. Clearly the first order formula $\forall x.\mathbf{f}(\phi)$ is satisfiable iff ϕ has a model. However, this reduction is not very useful since we have reduced a finite problem (inference in modal logic) to an infinite one (inference in predicate logic).

Given a well-formed formula $\mathcal{W}$ in predicate logic it is known that by renaming variables and introducing function symbols if necessary, $\mathcal{W}$ can be converted to $\mathcal{F}$ in (Skolem) Normal Form (SNF) so that $\mathcal{F}$ is satisfiable if and only if $\mathcal{W}$ is. An SNF formula has the form

$$\mathcal{F} = \forall y_1\forall y_2\cdots\forall y_k\ F^*$$

The features are that the quantifiers are all universal, all variables are bound by a quantifier and the matrix F^* is in conjunctive normal form (the predicates playing the role of atoms).

The variables $\{y_i\}$ have to be interpreted to construct a satisfying truth assignment (model) of the SNF formula. The following is an example formula which is satisfiable but only by an infinite interpretation.

Schönfinkel-Bernays Formula

$$\begin{aligned}
\forall x &: [P(x,f(x))]\\
\wedge\,\forall(u,v,w) &: [(P(u,v)\wedge P(v,w)\rightarrow P(u,w)]\\
\wedge\,\forall y &: [\neg P(y,y)]
\end{aligned}$$

Hence any complete embedding of predicate logic must contend with infinite structures. Consider a mathematical program of the form

$$\mathcal{D} = \{x\in\{0,1\}^{\omega} : \mathcal{A}x\geq\beta\} \tag{2}$$

where ω denotes (uncountable) infinity. Each row of the matrix $\mathcal{A}$ has entries that are $0,\pm1$, and each entry of the (uncountably) infinite column β is 1$-$ the number of -1's in the corresponding row of $\mathcal{A}$. So this is just an infinite version of (1). The finite support of the rows of $\mathcal{A}$ is the important structural property that permits the compactness theorems based on product topologies to go through in the ensuing development. It is a natural restriction in the context of first order logic as it corresponds to the finite "matrix" property of first order formulae. Note that compactness theorems can be pushed through for more general infinite mathematical programs using the so called "weak * topologies" but this shall not concern us.

In discussing Horn logic, we will encounter the continuous (linear programming) relaxation of our infinite mathematical program (2).

$$\bar{\mathcal{D}} = \{ x \in [0,1]^{\omega} : \mathcal{A}x \geq \beta \} \tag{3}$$

Let $\{\mathcal{A}_{\alpha}x \geq \beta_{\alpha}\}_{\alpha \in \mathcal{I}}$ denote a suitable indexing of all finite subfamilies of $\{\mathcal{A}x \geq \beta\}$. And for each α in the uncountable set $\mathcal{I}$ let

$$\mathcal{D}_{\alpha} = \{ x \in \{0,1\}^{\omega} : \mathcal{A}_{\alpha}x \geq \beta_{\alpha} \}$$

$$\bar{\mathcal{D}}_{\alpha} = \{ x \in [0,1]^{\omega} : \mathcal{A}_{\alpha}x \geq \beta_{\alpha} \}$$

Thus,

$$\mathcal{D} = \bigcap_{\alpha \in \mathcal{I}} \mathcal{D}_{\alpha}$$

$$\bar{\mathcal{D}} = \bigcap_{\alpha \in \mathcal{I}} \bar{\mathcal{D}}_{\alpha}$$

The analysis of finite dimensional mathematical programs such as (1) is based on elementary techniques from combinatorics and polyhedral theory. The situation in the infinite dimensional case gets more complicated. Constraint qualification is a sticky issue even for semi-infinite mathematical programs. The standard approach in infinite dimensional mathematical programming is to impose an appropriate (weak) topological framework on the feasible region and then use the power of functional analysis to develop the structural theory.

3 Compactness Theorems

A classical result in finite dimensional linear programming states that if a finite system of linear inequalities in $\Re^d$ is infeasible, there is a "small" $(d+1)$ subsystem that is also infeasible. This compactness theorem is a special case of the ubiquitous Helly's Theorem. Analogous theorems are also known for linear constraints on integer valued variables (cf. [25]). In the infinite dimensional case, we could hope for the "small" witness of infeasibility to simply be a *finite* witness. This is exactly what we prove for infinite 0-1 programs, linear programs and mixed integer programs with structure relating to embeddings of various fragments of predicate logic.

3.1 Infinite 0-1 Integer Linear Programs

Let $\mathcal{S}_{\gamma}$, $\gamma \in \mathcal{G}$, be copies of a Hausdorff space $\mathcal{S}$. Let $\mathcal{S}^{\mathcal{G}} = \prod_{\gamma \in \mathcal{G}} \mathcal{S}_{\gamma}$. The *product topology* on $\mathcal{S}^{\mathcal{G}}$ is the topology defined by a basis $\prod_{\gamma} O_{\gamma}$ where the O_{γ} are open in $\mathcal{S}_{\gamma}$ and $O_{\gamma} = \mathcal{S}_{\gamma}$ for all but at most finitely many $\gamma \in \mathcal{G}$. A classical theorem on compact sets with product topology is that of Tychonoff(cf. [20], page 232) which states that

Theorem 3.1 *Arbitrary (uncountable) products of compact sets with product topology are compact.*

Taking $\{0,1\}$ ($[0,1]$) as a compact set of a Hausdorff space $\{0,1\}$ ($[0,1]$) and applying Tychonoff's theorem we get

Corollary 3.2 *$\{x \in \{0,1\}^\omega\}$ ($\{x \in [0,1]^\omega\}$) (with product topology) is compact.*

Next we show that $\mathcal{D}_\alpha$ and $\bar{\mathcal{D}}_\alpha$, with product topologies, are also compact for any α in $\mathcal{I}$. This follows from the corollary and the lemma below.

Lemma 3.3 *The set $\{x : \mathcal{A}_\alpha x \geq \beta_\alpha\}$ ($\alpha \in \mathcal{I}$) is closed and hence $\bar{\mathcal{D}}_\alpha$ is compact for all ($\alpha \in \mathcal{I}$).*

Proof: Let y be a point in the complement of $\{x : \mathcal{A}_\alpha \geq \beta_\alpha\}$. So, there must be at least one violated constraint in the system $\mathcal{A}_\alpha x \geq \beta_\alpha$ of the form

$$\sum_{j \in J_i} \mathcal{A}_{ij} y_j < \beta_i$$

Noting that $|J_i|$ is finite, we can assert that

$$B_\epsilon = \{z : |z_j - y_j| < \epsilon \forall j \in J_i\}$$

is an open set. And for sufficiently small ϵ we have $B_\epsilon \subset \{x : \mathcal{A}_\alpha x \geq \beta_\alpha\}^C$. Hence, $\{x : \mathcal{A}_\alpha x \geq \beta_\alpha\}^C$ is open and $\{x : \mathcal{A}_\alpha x \geq \beta_\alpha\}$ is closed. □

Now we are ready for the main compactness theorem for 0-1 programs and their linear programming relaxations.

Theorem 3.4 *$\mathcal{D}$ ($\bar{\mathcal{D}}$) is empty if and only if $\mathcal{D}_\alpha$ ($\bar{\mathcal{D}}_\alpha$) is empty for some $\alpha \in \mathcal{I}$.*

Proof: Suppose $\mathcal{D}$ is empty and $\mathcal{D}_\alpha$ is nonempty for all $\alpha \in \mathcal{I}$. Then, for every $\mathcal{K} \subset \mathcal{I}$ with $|\mathcal{K}| < \infty$ we know that

$$\bigcap_{\alpha \in \mathcal{K}} \mathcal{D}_\alpha \neq \emptyset$$

So by the finite intersection property (cf. [20] page 171) we know that $\mathcal{D}$ is nonempty - a contradiction. The proof for $\bar{\mathcal{D}}$ is identical. □

Remark: An interesting question is whether there is an upper bound on the size of the finite witness of unsolvability of these infinite $0-1$ integer programs. It is not difficult to construct a quadratic first-order formula (quadratic because each clause is allowed at most two predicates) such that the size of the finite witness grows arbitrarily large.

3.2 Infinite Linear Programs

The compactness theorem (Theorem 3.4) applies to infinite linear programs that arise as the relaxation of 0-1 programs. In such programs, all the variables are bound by the interval $[0,1]$. If we permit variables to take arbitrary values in $\Re$, compactness can be obtained only under certain assumptions on the recession cones of the underlying convex sets.

Let I, L denote possibly uncountable index sets. Let $\Re_i$ = a replica of $\Re$ for $i \in L$ and $\Re^I = \pi_{i \in L} \Re_i$ with product topology. Assume I to be well ordered and write $X \in \Re^I$ as $x = [x_\alpha, \alpha \in I]$.

For finite $J \subset I$, denote by $x_J = \Re^{|J|}$ the appropriately ordered $|J|$-tuple $[x_\alpha, \alpha \in I]$. Also, define $||x||_\infty = \sup_\alpha |x_\alpha|$ (possibly $+\infty$).

For each $i \in L$, we have a constraint C_i of the type

$$A_i x_{j(i)} \geq b_i$$

where $J(i) \subset I$ is finite, $A_i \in \Re^{J(i)| \times |J(i)|}, b_i \in \Re^{|J(i)|}$.

Without loss of generality, let $\cup_{i \in L} J(i) = I$.

Let θ be the zero vector in $\Re^I$ and θ_n the zero vector in $\Re^n$.

Assumption:

$$\bigcap_{i \in L} \{x \mid A_i x_{J(i)} \geq \theta_{|J(i)|}\} = \{\theta\} \tag{4}$$

(i.e., the convex sets defined by $C_i, i \in L$, have no common direction of recession).

Theorem 3.5 *$\exists$ finite $M \subset L$ such that for $J \triangleq \cup_{i \in M} J(i)$,*

$$\bigcap_{i \in M} \left\{x_J \in \Re^{|J|} \mid A_i x_{J(i)} \geq \theta_{|J(i)|}\right\} = \{\theta_J\} \tag{2}$$

Proof : From (1), we have

$$\bigcap_{i \in L} \left\{x \mid A_i x_{J(i)} \geq \theta_{|J(i)|}, ||x||_\infty = 1\right\} = \phi. \tag{3}$$

Each set above is compact (being a closed subset of $\{x \mid ||x||_\infty = 1\}$ which is compact by Tychonoff's theorem). Thus by the finite intersection property of families of compact sets, $\exists$ a finite $M \subset L$ such that

$$\bigcap_{i \in M} \left\{x \mid A_i x_{J(i)} \geq \theta_{|J(i)|}, ||x||_\infty = 1\right\} = \phi.$$

Hence,

$$\bigcap_{i \in M} \left\{x_J \in \Re^{|J|} \mid A_i x_{J(i)} \geq \theta_{|J(i)|}, \max_{\alpha \in J} |x|_\alpha = 1\right\} = \phi.$$

Suppose $\exists \bar{x} \in \Re^{|J|}$ such that

$$\begin{aligned} A_i \bar{x}_{J(i)} &\geq \theta_{|J(i)|}, \quad i \in M, \\ \bar{x}_{J(i)} &\neq \theta_{|J(i)|}, \text{ for at least one } i. \end{aligned}$$

Then $a \triangleq \max_{\alpha \in J} |x_\alpha| > 0$. Define $\tilde{x} \in \Re^{|J|}$ by:

$$\tilde{x}_\alpha = x_\alpha / a, \quad \alpha \in J, \quad \tilde{x}_\alpha = 0 \text{ for } \alpha \notin J,$$

Then $||\tilde{x}||_\infty = 1, \quad A_i \tilde{x}_{J(i)} \geq \theta_{|J(i)|} \; \forall \; i \in M$, i.e., $\tilde{x}$ is in the l.h.s. of (3), a contradiction. Therefore no such $\bar{x}$ can exist. In other words, (2) holds. □

By abuse of notation, let C_i denote the closed convex subset of $\Re^I$ for which $A_i x_{J(i)} \geq b_i$.

Theorem 3.6 *Under the above assumption, if $\cap_{i \in L} C_i = \phi$, then there exists a finite $K \subset L$ such that $\cap_{i \in K} C_i = \phi$.*

Proof: Let $M, J(i), i \in M, J$ be defined as in the preceding theorem. Let

$$C_i' = \{x^J \in \Re^{|J|} \mid x \in C_i\}$$

denote the projection of C_i to $\Re^{|J|}$ under the map $x \to x_J$. It suffices to show that $\exists$ a finite $K \subset L$ for which $\cap_{i \in K} C_i' = \phi$. Define $\overline{C}_i = C_i' \cap (\cap_{J \in M} C_J'), i \in L$. By the preceding theorem, $C_j', \phi j \in M$, do not have a common direction of recession and therefore $\cap_{j \in M} C_j'$ is bounded (cf. Rockafellar [24], pp. 60-61). It is also closed and therefore compact. Thus $\overline{C_i}, i \in L$, are compact and $\cap_{i \in L} \overline{C}_i = \phi$. By the finite intersection property of families of compact sets, it follows that there exists a finite $T \subset L$ such that $\cap_{i \in T} \overline{C}_i = \phi$. Let $K = T \cup M$. Then $\cap_{i \in K} C_i' = \phi$. □

The following examples show that the assumptions cannot be relaxed.

Example 3.1 $I = \{1\}$, $C_i = \{x \mid X \geq i\}, i \geq 1$.
Then $\cap_i C_i = \phi$, but no finite intersection is empty.

Example 3.2 *This example shows that the assumption is needed even when $\{A_i\}, \{b_i\}$ remain bounded.*

$$\begin{aligned} I &= \{1,2\}, C_i, i = 0,1,2,\ldots \quad \textit{defined by}: \\ C_0 &= \{[x,y] \mid x \leq 0\}, C_n = \{[x,y] \mid x + \frac{1}{n}y \geq 1\} n \geq 1. \end{aligned}$$

In fact, the 'assumption' is both necessary and sufficient. (For sufficiency, simply note that if there is a common recession direction for $C_i, i \in I$, any finite intersection of the C_i's will have a ray along that direction and is therefore nonempty).

3.3 Infinite 0-1 Mixed Integer Programs

In the context of partially interpreted logics, we will need compactness results for infinite linear programs which have a subset of variables bound to $\{0,1\}$ along with real-valued variables with no explicit bounds on them. The compactness theorem (Theorem 3.6) that we just saw, can be extended to this case as well. The addition of $0-1$ variables causes no difficulty to compactness since they are bounded. The assumption that the constraint regions have no common recession direction is modified to assuming that the projection of the constraint regions onto the space of real-valued variables have no common recession direction.

Let I, K, L denote possibly uncountable index sets. Let $\Re_i$ be a replica of $\Re$ for $i \in L$ and $\Re^I = \pi_{i \in L} \Re_i$ with product topology. Assume I to be well ordered and write $x \in \Re^I$ as $x = [x_\alpha, \alpha \in I]$. Similarly let $\{0,1\}_i$ denote a replica of $\{0,1\}$ for $i \in L$ and $\{0,1\}^K = \pi_{i \in L}\{0,1\}_i$ with product topology. Assume K to be well ordered and write $u \in \{0,1\}^K$ as $u = [u_\beta, \beta \in K]$.

For finite $J \subset I$, denote by $(x_J) \in \Re^{|J|}$ the appropriately ordered $|J|$-tuple $[(x_\alpha), \alpha \in I]$. Similarly, for finite $T \subset K$, denote by $(u_T) \in \{0,1\}^{|T|}$ the appropriately ordered $|K|$-tuple $[(u_\beta), \beta \in K]$.

For each $i \in L$, we have a constraint C_i of the type

$$A_i x_{J(i)} + B_i u_{T(i)} \geq h_i$$

where $J(i) \subset I$ is finite, $T(i) \subset K$ is finite, $A_i \in \Re^{M(i)|} \times \Re^{|J(i)|}, B_i \in \Re^{M(i)|} \times \Re^{|T(i)|}$, $h_i \in \Re^{|M(i)|}$, and $M(i)$ is finite.

Without loss of generality, let $\cup_{i \in L} J(i) = I$ and $\cup_{i \in L} K(i) = K$.

Let $\theta =$ the zero vector in $\Re^I$ and θ_n the zero vector in $\Re^n$.

Assumption:

$$\bigcap_{i \in L} \mathcal{P}_x \{x \mid A_i x_{J(i)} + B_i u_{K(i)} \geq \theta_{|M(i)|}\} = \{\theta\} \tag{5}$$

(Here $\mathcal{P}_x$ denotes the projection operator which projects a given set in x, u-space onto x-space. Note that each $C_i, i \in L$ represents a union of convex sets. The assumption is that the x-projection of these sets have no common direction of recession).

The compactness results are now derived exactly as they were for the case of infinite linear programs. Note that convexity of the constraint regions was never used in the compactness proofs in that case. The result for the case of infinite $0-1$ mixed integer programs is summarized by the theorem below.

Theorem 3.7 *Under the above assumption, if $\cap_{i \in L} C_i = \phi$, then there exists a finite $K \subset L$ such that $\cap_{i \in K} C_i = \phi$.*

Since the case of infinite linear programs is a special case of the infinite $0 - 1$ mixed integer programs (where all the u variables are bound to 0 or 1), it follows that the "assumption" is both necessary and sufficient.

4 The Mathematical Programming of Herbrand's Theorem

Starting with an SNF formula $\mathcal{F}$, we define the Herbrand universe $U^{\mathcal{H}} = \mathcal{D}(\mathcal{F})$ in the usual way. If the matrix F^* contains some constant symbols we use them and if not we introduce a Skolem constant a and define $\mathcal{D}(\mathcal{F})$ by instantiating all variables in all terms on these constant symbols. The Herbrand expansion of $\mathcal{F}$ is then given by

$$E(\mathcal{F}) = \bigcap \{F^*[y_1/t_1][y_2/t_2] \cdots [y_k/t_k] \mid t_1, t_2, \cdots, t_k \in \mathcal{D}(\mathcal{F})\}$$

Notice that $E(\mathcal{F})$ really an infinite propositional CNF formula since all the variables have been substituted to fully ground terms. A classical result in theorem proving (attributed independently to Gödel, Skolem and Herbrand in the literature) is that the SNF formula $\mathcal{F}$ is satisfiable (in a predicate logic sense) if and only if the CNF formula $E(\mathcal{F})$ is (in a propositional sense). We know how to embed satisfiability of propositional formulae as $0-1$ linear programs. The fact that the number of propositions is infinite (countable) as are the number of clauses means that the embedding will be a special case of (2). And then applying Theorem 3.4, we obtain Herbrand's theorem.

Theorem 4.1 *A Skolem Normal Form formula $\mathcal{F}$ is unsatisfiable if and only if there is a finite subformula of the Herbrand Expansion $E(\mathcal{F})$ which is unsatisfiable.*

This theorem may be viewed as the cornerstone of theorem proving in predicate logic since it implies that proving a formula unsatisfiable (if we already know that it is so) is decidable (simply develop the Herbrand Expansion - one instantiation at a time - and check the resulting finite CNF formula for propositional satisfiability). Of course there have been many sophistications to this scheme since Herbrand but the basic construct remains the same.

4.1 The Least Herbrand Model of Definite Programs

We believe that the infinite $0-1$ embedding that was just used to prove Herbrand's theorem can also be specialized and honed to shed light on these more modern aspects of theorem proving. As an illustration we consider the case of Horn formulae (each clause of $\mathcal{F}$ contains at most one positive atom) and show that in this case we can restrict our attention to the linear programming relaxation embedding (3) and still obtain the well known result on unique minimal models for definite programs.

Assuming now that H is a Horn formula as defined above, we formulate the following infinite dimensional optimization problem.

$$\inf\left\{\sum x_j \mid \mathcal{A}x \geq \beta, x \in [0,1]^{\omega}\right\} \tag{6}$$

where the linear inequalities $\mathcal{A}x \geq \beta$ are simply the clausal inequalities corresponding to the ground clauses of H. The syntactic restriction on Horn clauses translates to the restriction that each row of $\mathcal{A}$ has at most one $+1$ entry (all other entries are either 0 or -1's - only finitely many of the latter though). We shall prove now that if the infinite linear program (6) has a feasible solution then it has an integer optimal $(0-1)$ solution. Moreover, this solution will be a least element of the feasible space i.e., it will simultaneously minimize all components over all feasible solutions.

Lemma 4.2 *If the linear program (6) is feasible then it has a minimum solution.*

Proof: Let $\psi_n = \sum_{j=1}^{n} x_j$ and $\Psi = \sup_n \psi_n$. As the supremum of continuous functions, we know that Ψ is lower semi-continuous (*lsc*). The the optimization problem (6) seeks to find the infimum of an *lsc* function over a compact set. Therefore, the minimum is attained. □

Lemma 4.3 *If x^1 and x^2 are both feasible solutions for (6) then so is $\{\bar{x}_j = \min x_j^1, x_j^2\}$.*

Proof: Let x^i be partitioned into (y^i, z^i) $(i = 1, 2)$ such that the components of y^1 are no larger than the components of y^2 and the components of z^2 are no larger than the components of z^1. Now if an inequality in the constraints of (6) has a $+1$ coefficient on a y variable (or if the inequality has no $+1$ coefficient at all) we note that (y^1, z^1) satisfies the inequality and therefore so does (y^1, z^2) since the z-coefficients are all nonpositive. Similarly, if an inequality in the constraints of (6) has a $+1$ coefficient on a z variable we note that (y^2, z^2) satisfies the inequality and therefore so does (y^1, z^2) since the y-coefficients are all nonpositive. Therefore, in all cases, (y^1, z^2) is feasible. □

Theorem 4.4 *If the linear program (6) is feasible, then it has a unique* $0-1$ *optimal solution which is the least element of the feasible set.*

Proof: If the feasible region of (6) is nonempty, we know that an optimal solution exists. Let x^* be such an optimal solution. If x^* has all $0-1$ components there is nothing to prove. Else let $\tilde{\mathcal{A}}\tilde{x} \geq \tilde{\beta}$ be obtained from $\mathcal{A}x \geq \beta$ by fixing all components $x_j = x_j^*$ for all $0-1$ valued x_j and clearing the constants to the right-hand-side of the inequalities to obtain $\tilde{\beta}$. Note that $\tilde{\beta}$ is integer valued. Now, every $\tilde{\beta}$ coefficient must be nonpositive. Since otherwise, we would have at least one inequality with a right-hand-side of $+1$ or larger and a left-hand-side of fractional coefficients no more than one of which is positive and such an inequality is impossible to satisfy with $\tilde{x}_j$ in $[0,1]$. Hence we can set the $\tilde{x}$ to 0 and maintain feasibility. This contradicts the optimality of x^* in (6). $\square$

The interpretation of this theorem in the logic setting is that if a Horn formula H has a model then it has a least model (a unique minimal model). This is an important result in model theory (semantics) of so-called definite logic programs.

Remark: In the context of propositional logic, Jeroslow and Wang (cf. [16]) showed that the optimal solution to the dual of the linear programming relaxation of an unsatisfiable Horn formula, is a signature of the number of times clauses are used in a resolution proof of unsatisfiability. The compactness theorem implies that a similar result must hold for the predicate case as well since compactness gives us a finite grounding of the Horn formula that is already unsatisfiable.

4.2 The Mathematical Programming of CLP($\Re$)

In most early implementations of logic programming (viz. ProLog), the language designers found it necessary to include partially interpreted formulas via so-called "built-in predicates". This was deemed to be a practical necessity, since in programming with pure logic, i.e. uninterpreted symbols, it would take too much effort to exploit the problem-solving capabilities developed in several numerical and algebraic domains. There are also other reasons for including built-in predicates emanating from programming ease. Of course, this meant that the theoretical framework , in particular the Herbrand interpretation, cannot be used to analyze the semantics of such programs. The constraint logic programming (CLP) scheme [12,13,14] was proposed in the mid-80's by Jaffar, Lassez and Maher to address this conflict between theory and practice of logic programming. CLP works with partially interpreted Horn formulas, where some of the predicates and variables have specific interpretations as constraints on domains which have useful expressive power and have efficient solution methods. In CLP, compactness properties of constraint domains are combined with compactness in the Herbrand universe (on the pure logic predicates) to generalize Herbrand's Theorem in a richer setting.

Thus Constraint Logic Programming began as a natural merger of two declarative paradigms: constraint solving and logic programming. This combination helps make CLP programs both expressive and flexible, and in some cases more efficient than other kinds of programs. We apply our embedding technique to a particular kind of CLP known as CLP($\Re$) to bring out its inherent mathematical programming nature. Constraints in CLP($\Re$) are linear inequalities on real-valued variables. Thus CLP($\Re$) brings together the techniques of linear programming and logic programming in a declarative programming language setting.

Constraint Logic Programming: Some Definitions[12,13,14]

If Σ is a signature, a Σ-structure $\mathcal{M}$ consists of a set D and an assignment of functions and relations on D to the symbols of Σ which respects the arities of the symbols. A Σ-theory $\mathcal{T}$ is a collection of closed Σ-formulas. A model of a Σ-theory $\mathcal{T}$ is a Σ-structure $\mathcal{M}$ such that all formulas of $\mathcal{T}$ evaluate to true under the interpretation provided by $\mathcal{M}$. A primitive constraint has the form $p(t_1, \ldots, t_n)$, where $t_1, \ldots, t_n$ are Σ-terms and $p \in \Sigma$. A constraint (first-order) formula is built from the primitive constraints in the usual way using logical connectives and quantifiers [12,14].

In constraint logic programming there is also a signature Π comprising of the uninterpreted predicates that are defined by a logic program. A CLP atom has the form $p(t_1, \ldots, t_n)$ where $t_1, \ldots, t_n$ are terms and $p \in \Pi$. A program P is of the form $p(\bar{x}) \leftarrow C, \bar{q}(\bar{y})$ where $p(\bar{x})$ is an atom, $\bar{q}(\bar{y})$ is a finite sequence of atoms in the body of the program and C is a conjunction of constraints. A goal G is a conjunction of constraints and atoms. A rule of the form $p(\bar{x}) \leftarrow C$ is called a *fact.*

We assume that programs and goals are in the following standard form.

- All arguments in atoms are variables and each variable occurs in at most one atom. This involves no loss of generality since a rule such as

$$p(\bar{t}) \leftarrow C, q(\bar{s})$$

can be replaced by the rule

$$p(\bar{x}) \leftarrow \bar{x} = \bar{t}, \bar{y} = \bar{s}, C, q(\bar{y})$$

- All rules defining the same predicate have the same head and no two rules have any other variables in common (this is simply a matter of renaming).

For any signature Σ. let $\mathcal{M}$ be a Σ structure (the domain of computation) and $\mathcal{L}$ be a class of Σ-formulas (the constraints). We call the pair $(\mathcal{M}, \mathcal{L})$ a constraint domain. We also make the following assumptions.

- The binary predicate symbol "=" is contained in Σ and interpretated as identity in $\mathcal{M}$.

- There are constraints `true` and `false` in $\mathcal{L}$ which are respectively true and false in $\mathcal{M}$ respectively.

- The class of constraints in $\mathcal{L}$ is closed under variable renaming, conjunction and existential quantification.

Semantics

A valuation σ is a mapping from variables to the domain D. A $\mathcal{M}$-interpretation of a formula is an interpretation of the formula with the same domain as that of $\mathcal{M}$ and the same interpretation for the symbols in Σ as $\mathcal{M}$. It can be represented as a subset of $\mathcal{B}_{\mathcal{M}}$ where $\mathcal{B}_{\mathcal{M}} = \{ \ p(\bar{d}) \mid p \in$

$\Pi, \bar{d} \in D_k$ for some k} . A $\mathcal{M}$-model of a closed formula is a $\mathcal{M}$-interpretation which is a model of the formula. The usual logical semantics are based on the $\mathcal{M}$-models of P .

CLP($\Re$) DEFINED [14]

Let Σ contain the constants 0 and 1, the binary function symbols + and $*$, and the binary predicate symbols $=, <$, and $\leq$. Let D be the set of real numbers and let $\mathcal{M}$ interpret the symbols of Σ as usual (i.e. + is interpretated as addition etc.). Let $\mathcal{L}$ be the constraints generated by the primitive constraints. The $\Re = (\mathcal{M}, \mathcal{L})$ is the constraint domain of arithmetic over the real numbers. For our purpose *we will consider only function symbol + and only predicate symbol* $\geq$ in Σ. A typical rule in CLP($\Re$) will look like $p(x) \leftarrow (2x + 3y \geq 2), (4y \geq 3), q(y)$, where $x, y \in \Re$. When we associate the variables in a rule in CLP($\Re$) with values over the reals $\Re$ we obtain a ground instance of that rule. It is easy to see that **the ground instances of a rule in CLP($\Re$) are uncountable.**

THE EMBEDDING AS AN INFINITE $0-1$ MIXED INTEGER PROGRAM

In order to illustrate the formulation, let us assume that a rule R_i in a given CLP($\Re$) is of the form

$$p(x) \leftarrow \tilde{c}_1, \tilde{c}_2, q_1(y_1), q_2(y_2)$$

where $\tilde{c}_1$ and $\tilde{c}_2$ are primitive constraints of the form $f(x,y) \geq 0$ and $g(x,y) \geq 0$ respectively, and $f(x,y), g(x,y)$ are linear functions of x, y. The q_i are atoms. We associate a linear (clausal) inequality lc(R_i) as follows

$$v_{p(x)} + \sum_{i=1}^{2}(1 - u_{\tilde{c}_i}) + \sum_{i=1}^{2}(1 - v_{q(y_i)}) \geq 1$$

This clausal inequality can be rewritten as

$$v_{p(x)} - \sum_{i=1}^{2} u_{\tilde{c}_i} - \sum_{i=1}^{2} v_{q(y_i)} \geq (1 - k)$$

, where k is the total number of primitive constraints and atoms in the body of the rule.

We also associate the linear equalities

$$f(x,y) + (1 - u_{\tilde{c}_1})M \geq 0$$

$$g(x,y) + (1 - u_{\tilde{c}_2})M \geq 0$$

with the rule R_i , where M is an arbitrary large number. Note that a constraint must be solvable if the corresponding u variable is to take value 1. Also, if a particular value of u is feasible, then so are all smaller values of u (as far as these inequalities are concerned). We rewrite these inequalities as

$$f(x,y) - Mu_{\tilde{c}_1} \geq -M$$

$$g(x,y) - Mu_{\tilde{c}_2} \geq -M$$

respectively and denote them as le(R_i).

Given a CLP($\Re$) program $\mathcal{P}$ we construct a $0-1$ mixed integer program $\mathcal{F}_{\mathcal{P}}\{0,1\}$ as follows,

1. For each rule R in $\mathcal{P}$, the linear inequality $\mathrm{lc}(R)$ is in $\mathcal{F}_{\mathcal{P}}\{0,1\}$.
2. For each rule R in $\mathcal{P}$, the inequality $\mathrm{le}(R)$ corresponding to the constraints appearing in $\mathcal{P}$ is in $\mathcal{F}_{\mathcal{P}}\{0,1\}$.
3. For any atom $p(\bar{x})$ appearing in $F_{\mathcal{P}}\{0,1\}$ the constraint $v_{p(\bar{x})} \in \{0,1\}$ is in $\mathcal{F}_{\mathcal{P}}\{0,1\}$.
4. For every primitive constraint $\tilde{c}$ appearing in $\mathcal{P}$, $u_{\tilde{c}} \in \{0,1\}$ is in $\mathcal{F}_{\mathcal{P}}\{0,1\}$.
5. For every variable x appearing in $\mathcal{P}$, the constraint $x \in \Re$ is in $\mathcal{F}_{\mathcal{P}}\{0,1\}$.

When we replace the restriction (3) by $v_{p(\bar{x})} \in [0,1]$ and 4 by $u_{\tilde{c}} \in [0,1]$, we get $\mathcal{F}_{\mathcal{P}}[0,1]$.

If we ground the formulation $\mathcal{F}_{\mathcal{P}}\{0,1\}$, by grounding the logical variables on the Herbrand Universe and the interpreted variables x on the reals, we would obtain an infinite $0-1$ mixed integer program. Under suitable assumptions, we could obtain a compactness theorem akin to Theorem 3.7. In addition, we obtain a least model property for CLP($\Re$) by noting that the following linear program

$$\inf\left\{\sum v + \sum u \mid u, v, x \text{ satisfy ground } \mathcal{F}_{\mathcal{P}}[0,1]\right\} \tag{7}$$

has a minimum v, u solution that is guaranteed to be $0-1$ valued. A formal statement and proof of this result is completely analogous to Theorem 6.

5 Concluding Remarks

An important issue related to the uncountable nature of the embeddings, presented herein, is whether the proof of the compactness theorem can be made constructive. One idea is to be able to identify a countable subsystem to restrict the search to. In addition, a natural enumeration scheme is required for the countable subsystem to construct decision procedures. This is in effect what is done in classical first-order logic since the Herbrand Universe and the the Herbrand Extension provide just such a substructure.

We believe that the embedding results of this paper can be usefully applied to better our understanding of Hybrid Systems. In such systems, there is a mix of discrete structures (logic) with mathematical programming (control theory) structures. The embeddings presented in this paper offer unified frameworks for carrying out this integration.

References

[1] Andersen, K. A., and J. N. Hooker, A linear programming framework for logics of uncertainty, manuscript, Mathematical Institute, Århus University, 8000 Århus C, Denmark (1992).

[2] Araque G. J.R., and V. Chandru, Some facets of satisfiability, Technical Report CC-91-13, Institute for Interdisciplinary Engineering Studies, Purdue University, West Lafayette, IN 47907 USA (1991).

[3] J. van Benthem, J. van Eijck, V. Stebletsova, *Modal logic, transition systems and processes* Math Centrum, CS-R9321 (1993)

[4] Blair, C., R. G. Jeroslow, and J. K. Lowe, Some results and experiments in programming techniques for propositional logic, *Computers and Operations Research* **13** (1988) 633-645.

[5] V. S. Borkar, V. Chandru, S. K. Mitter, *A Linear Programming Model of First Order Logic* Indian Institute of Science, TR IISc-CSA-95-5 (1995)

[6] Chandru, V., and J.N. Hooker, Extended Horn Sets in Propositional Logic, *Journal of the ACM*, 38, 1, pp. 205-221.

[7] Conforti, M., and G. Cornuéjols, A class of logical inference problems soluble by linear programming, *FOCS* 1992.

[8] Gallo, G., and G. Rago, A hypergraph approach to logical inference for datalog formulae, working paper, Dip. di Informatica, University of Pisa, Italy (September 1990).

[9] Hooker, J. N., A mathematical programming model for probabilistic logic, working paper 05-88-89, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA 15213 (July 1988).

[10] Hooker, J. N., Input proofs and rank one cutting planes, *ORSA Journal on Computing* **1** (1989) 137-145.

[11] Hooker, J.N., New Methods for Computing Inferences in First Order Logic, *Annals of Operations Research*, (1993).

[12] Jaffar, J.; and Lassez, J.-L.; Constraint Logic Programming, Technical Report 86/73, Department of Computer Science, Monash University, 1986.

[13] Jaffar, J.; and Lassez, J.-L.; *Constraint Logic Programming* in Proc. 14th symposium on Principles of programming Languages, Munich; (Jan. 1987), pp. 111–119.

[14] Jaffar, J.; and Maher, M.J., *Constraint Logic Programming: A survey*, Jour. of Logic Programming 19/20 (1994) pp. 503–581.

[15] Jeroslow, R. G., Computation-oriented reductions of predicate to propositional logic, *Decision Support Systems* **4** (1988) 183-197.

[16] Jeroslow, R. G., *Logic-Based Decision Support: Mixed Integer Model Formulation, Annals of Discrete Mathematics* **40**. North-Holland (Amsterdam 1989).

[17] Kagan, V., A. Nerode and V.S.Subrahmanian, Computing Definite Logic Programs by Partial Instantiation and Linear Programming, *Technical Report 93-15*, Mathematical Sciences Institute, Cornell University, (1993).

[18] Kavvadias, D., and C. H. Papadimitriou, A linear programming approach to reasoning about probabilities, *Annals of Mathematics and Artificial Intelligence* **1** (1990) 189-206.

[19] R. Milner, *Communication and Concurrency.* Prentice Hall, London (1989)

[20] Munkres, J.R., *Topology: A First Course*, Prentice-Hall, (1975).

[21] Nilsson, N. J., Probabilistic logic, *Artificial Intelligence* **28** (1986) 71-87.

[22] V. R. Pratt, *Process Logic*, Proc. 6th Ann. ACM Symp. on Principle of Programming Languages (Jan. 1979)

[23] V. R. Pratt, *Dynamic Logic*, Proc. 6th International Congress for Logic, Philisophy, and Methodology of Science, (Hanover, Aug. 1979)

[24] Rockafeller, R.T., *Convex Analysis*, Princeton University Press, 1970.

[25] Schrijver, A., *Theory of Linear and Integer Programming*, Wiley (New York, 1986).

[26] Schöning, U., *Logic for Computer Scientists*, Birkhäuser, (1989).

[27] Wang, J-C., and J.VandeVate J., Question-asking strategies for Horn clause systems, *Annals of Mathematics and Artificial Intelligence*, Vol. 1, pp. (1990).