2

AD-A233 698

TRANSFORMATION OF PROPOSITIONAL CALCULUS STATEMENTS INTO INTEGER AND MIXED INTEGER PROGRAMS: AN APPROACH TOWARDS AUTOMATIC REFORMULATION

E Hadjiconstantinou
Imperial College, London

and

G Mitra
Brunel University, Uxbridge, Middlesex

The research work reported in this document was supported by the U.S. Army's European Research Office, London

Contract No DAJA4S-87-C-003/USARDSG(UK)

CONTENTS

0. ABSTRACT

A systematic procedure for transforming a set of logical statements or logical conditions imposed on a model into an Integer Linear Progamming (ILP) formulation or a Mixed Integer Programming (MIP) respectively is presented. An ILP stated as a system of linear constraints involving integer variables and an objective function, provides a powerful representation of decision problems through a tightly interrelated closed system of choices. It supports direct representation of logical (Boolean or propositional calculus) expressions. Binary variables (hereafter called logical variables) are first introduced and methods of logically connecting these to other variables are then presented. Simple constraints can be combined to construct logical relationships; the methods of formulating these are also discussed. These reformulation procedures are illustrated by two examples. A scheme of implementation within an LP modelling system is outlined.

1. INTRODUCTION

In recent times first order logic in the form of propositional or predicate calculus has taken a central position in the formulation and solution of problems taken from diverse domains such as management science models, artificial (AI) intelligence, database applications, and programming languages. In AI for instance, not only automatic theorem proving via instantiation and resolution is a leading topic of interest; the simplest form of knowledge representation via production rules and diagnostic expert systems, which provide explanation through chairing procedures, also have many applications. All of these depend heavily on the underlying logic representation and the related computational issue of making deductive inference.

Thus central to these applications is the problem of logical inference which is the problem of determining if a particular conclusion in propositional logic follows from certain premises. The generally accepted type of inference procedure, symbolic (as opposed to numeric) calculation has failed to solve large inference problems. Even with the help of faster and larger computers emerging, logicians can only work with problems of limited size. Consequently, over recent years, an alternative emphasis has been under investigation for new and faster techniques in order to deal with problems that are formulated in propositional or predicate logic. The current trend for applying quantitative methods generally, and the methods of mathematical programming in particular, to problems in propositional logic can be explained by highlighting the three underlying reasons set out below:

(i) An "intelligent" mathematical programming system is highly structured; such a system can be used to exploit the high degree of mathematical structure inherent in propositional logic. This enables the development of modelling procedures where by statements in propositional logic can be represented as discrete optimization problems involving 0-1 integer, and 0-1 integer as well as continuous variables; that is, integer programmes (IP) and mixed integer programmes (MIP).

(ii) Close parallels exist between some important concepts of propositional logic and mathematical programming which can lead to better methods, both quantitative and symbolic, for solving logical problems more efficiently. Furthermore, the results of this research can also be applied to solve various types of optimization problems in the area of mathematical programming.

(iii) It is well known that inference is a very hard combinatorial problem. If a knowledge base is encoded in the simplest sort of logical language (Propositional Calculus), then the inference problem cannot be solved in better than "exponential" time. The situation is even worse when the data are expressed in first-order predicate logic. However, it has been proved that inference involving Horn clauses can be accomplished in linear time (Dowling and Gallier [DOWGLR84]), using a class of resolution techniques. The special structure of mathematical programming methods, which can be potentially very fast, can then be exploited to provide a more robust approach towards representing and solving problems considered by AI and expert systems (it is usually possible to solve large IP models in a reasonable period of time if they have a special structure).

The focus of this paper is to develop a systematic approach for transforming statements in propositional logic into integer or mixed integer programmes. This method is particularly suitable as a modelling technique which then allows one to automate the conversion to an IP or MIP model. The final goal is to integrate this modelling function into an "intelligent" mathematical programming modelling support system. The rest of the paper is organized in the following way. In section 2 the background and motivations of earlier work in this field are set out. Section 3 contains a summary description of the important results in propositional logic and the corresponding 0-1 discrete programming equivalent forms. In section 4 these reformulation techniques are used in a progressive sequence and a systematic reformulation procedure is enunciated. Illustrative examples and implementation issues within an LP modelling system are considered in sections 5 and 6 respectively.

2. PREVIOUS WORK

2.1 First-Order Logic, Symbolic and Quantitative Methods

Symbolic, as opposed to numeric, calculation is the mathematical manipulation of symbols. In the domain of logic it was adapted by Boole who devised a real workable system (he used 0 and 1 for truth values and arithmetic symbols for logical operations) - which is now well known as the Boolean Algebra. Today, problems in AI rely heavily on symbolic manipulation. The popular resolution method for inference [ROBINS65] is designed for first-order predicate logic. Resolution applied to propositional logic is called ground resolution and is part of Quine's algorithm [QUINEW55]. The difficulty of the resolution algorithm is that it has recently been shown to have exponential complexity and to become rapidly impractical as the problem increases in size [HOOKER90]. Due to the inability of traditional inference methods to deal with large knowledge bases, most of the recent work in this area has been directed toward automated theorem proving, which involves relatively small knowledge bases. Hooker [HOOKER88] surveys the application of quantitative methods, and integer programming methods in particular as applied to inference problems in propositional logic. Williams [WILLMS77,WILLMS85], has shown how such problems could be modelled as equations or inequalities involving 0-1 integer variables. That verification or refutation of an argument could be modelled as a maximization or minimization of an objective function in these variables leading to an Integer Programme (IP) is also shown in this paper.

2.2 A Quantitative Approach: Efficient Formulation and Solution Procedures

Statements in propositional calculus can be modelled as integer programmes in different ways: thus a given compound proposition may have more than one representation. It is, however, well known that from a computational point of view one of these representations is superior to the others [WILLMS87].

One obvious method of reformulation is to express a compound proposition into a Conjunctive Normal Form (CNF) and then convert it into integer programming constraints [BLARJL88] [WILSON90]. This is a cumbersome approach as it requires more than one constraint to represent a compound proposition. In general, a number of CNFs are possible and there is no guarantee that a unique representation is obtained.

Williams argues that when using IP algorithms based on LP relaxations for solving problems in propositional calculus, it is desirable to "disaggregate" the constraints so that the LP relaxation is as close to the convex hull of feasible integer solutions as possible (that is, tight LP relaxation are created). Taking into consideration the geometry of the convex hull Jeroslow [JEROSL85] deduced that it is generally better to express a model in the Disjunctive Normal Form (DNF) before converting it to a representation in linear inequalities in terms of 0-1 variables. Blair, Jeroslow and Lowe [BLARJL85] were apparently the first to solve non-trivial inference problems with mathematical programming methods. They examined the connections and parallels between propositional logic and integer programming and how these can be combined to create new inference methods.

It is well known that most successful IP algorithms are based on "Branch and Bound" or "Cutting-Plane" techniques or some combination of the two. Blair, Jeroslow and Lowe showed that a branch-and-bound approach not only solves satisfiability problems quickly, but it is closely related to a variant of the well known Davis-Putnam procedure in logic. Later, Jeroslow and Wang [JERWAN87] replaced the LP in the branch-and-bound method by a variable fixing heuristic and obtained a symbolic method even faster than branch-and-bound. Beaumont [BEAUMN87] approaches the computational issue from the other direction. He first converts a MIP model into a DNF and then solves the resulting model by an algorithm based on a branch-and-bound procedure.

A well known class of inference problems, those involving Horn clauses, define IPs whose duals have integral polytopes and that exhibit a dynamic programming structure (Jeroslow and Wang [JERWAN89]). Roehrig [ROEHRG88] considered problems in propositional logic and suggested the use of a variant of an effective IP heuristic to achieve fast inference. His technique proved to be computationally more efficient compared to the traditional symbolic methods. The resolution method for solving inference is related to a cutting plane method for solving IPs and resolution can be dramatically accelerated by treating resolvents as cutting planes.

2.3 Logic Programming, Artificial Intelligence: The Common Problems

The growth in AI based wider modelling techniques can be traced back to development of inference procedures and computational logic: thus developments in natural language understanding, theorem proving and rule based expert systems utilize the computational underpinning of first order logic.

. **Rule based expert systems**

The simplest yet the most successful examples of expert systems use production rule based knowledge representation. These are usually set out in the well known propositional logic forms (see section 3) and are called rules. These rules and especially their statements are often exploited through the explanations procedure. The end user of the ES application is given a meaningful reasoning as to how a deduction was made [BSHORT84]. The proposal that set covering IP models and their variation are used to provide explanation in diagnostic expert systems has been put forward by Yager [YAGERR85] and Reggia et al [REGNWN83].

. **Constraint satisfaction and planning**

AI planning and reasoning with time is a specialist area of study where the applications of logic is extended to the time domain. AI planning is concerned with the selection and sequencing of actions which achieve a set of desirable goals: the main domains of its application are job shop scheduling, production planning, maintenance scheduling, Steel [STEELS87] as well as Allen [ALLENJ83] discuss the application of logic in these deductive systems.

. **Games, Puzzles and Combinatorial Programming**

Mathematical puzzles and games provide a rich source of application of AI and logic. Crossword compilation Berghel [BERGHL87], cryptarithm and "Smith-Jones-Robinson" problem of recreational logic [GARDNR61] are typical examples which are well suited for solution through logic. A wide range of combinatorial problems can also be cast in this paradigm and Laurier's research focus [LARIER78] was indeed to unify the description and solution of these problems.

. **Constraint Logic Programming**

Constraint logic programming, also known as constraint programming systems (CPS), are in essence programming paradigms which seek to satisfy arithmetic constraints within an otherwise logic programming framework The motivations, methodologies and their scope of application are well discussed by Hentenryck [HENTRK96] and Chinneck et al [BCHNKM89]. When the constraints (usually linear) involve expressions in real numbers naturally the simplex algorithm is applied to achieve constraint satisfaction; Lassez CLP(R) [LASSEC87] and Colmerauer [COLMRA87] PROLOG III are two such CPS. For constraints stated in discrete integers (natural numbers) tree search method or interval arithmetic is applied to achieve constraint satisfaction. Hertenryck [HENTRK89] CHIP and Brown and Chinneck [BNPRLG88] BNR-PROLOG report two systems of this type.

3. REPRESENTATION IN PROPOSITIONAL LOGIC AND 0-1 DISCRETE PROGRAMMING

3.1 Basic concepts and notations in propositional logic

A "statement" defines a declarative sentence. For example, "Athens is the capital of Greece" and "Five is an even number" are statements. This type of statement, about which it is possible to say that it is either true or false, but not both, is called a **simple** or **individual** or **atomic proposition**, (propositions and statements are synonymous words). A proposition can take one of the **truth values** true or false, that is the truth value of a true proposition is **TRUE** (abbreviate to T) and the truth value of a false proposition is **FALSE** (abbreviate to F). As no other value is permitted, the calculus of propositions is referred to a two-valued logic.

Propositional calculus enables compound propositions to be formed by modifying a simple proposition with the word "not" or by connecting propositions with the words "and", "or", "if... then" (or implies) and "if and only if". These five words are called propositional or logical **connectives** and they ae known as the **negation, conjunction, disjunction, implication** and **equivalence,** respectively. By repeatedly applying the connectives, the compound propositions can be used in turn to create further compound propositions. The symbolic representation of these connectives and their interpretation are shown in Table 1.

There are two meanings of the disjunction connective: the **inclusive or** meaning that at least one disjunct is true (allowing for the possibility that both disjuncts hold) and the **exclusive or** which is true if exactly one disjunct is true but not both. The latter operation is also known as "non-equivalence". Using the **implication** connective, a compound proposition has the form "if ... then ...", the proposition following "if" is the **antecedent** and the proposition following "then" is the **consequent**. Thus, the antecedent "implies" the consequent.

It is convenient to represent arithmetic variables by small letters x, y, z, etc., and propositions by capital letters from the middle part of the alphabet, P, Q, etc. Thus, P, Q, ... are used to represent

(i) actions, options or yes/no decisions (that is, atomic propositions). For example, P: "product is manufactured".

(ii) linear restrictions, that is, (in)equalities involving LP (or IP) variables. For example, Q: "$3x + 4y \leq z$"

No	Name of connective	Symbol	Meaning of connective	Other common words
1	negation	$\sim P$	not P	
2	conjunction	$P \wedge Q$	P and Q	Both P and Q
3	inclusive disjunction	$P \vee Q$	P or Q	Either P or Q/at least one of P or Q
4	non-equivalence (exclusive disjunction)	$P \not\equiv Q$ $(P \dot\vee Q)$	P xor Q	Exactly one of P or Q is true
5	implication	$P \rightarrow Q$	if P then Q	P implies Q... P is a sufficient condition for Q
6	equivalence	$P \leftrightarrow Q$ $(P \equiv Q)$	P if and only if Q	P if Q...P necessary and sufficient condition for Q
7	joint denial	$\sim(P \vee Q)$	P nor Q	Neither P nor Q/None of P or Q is true
8	non-conjunction	$\sim(P \wedge Q)$	P and Q	?

TABLE 1: Propositional Connectives

(iii) compound propositions.

Let P, Q, R and S represent the atomic propositions

P: "It is raining today"

Q: "Today is clear"

R: "Yesterday was cloudy"

The following compound propositions can then be constructed:

$\sim P$	:	"It is **not** raining today"
$Q \vee P$	:	"Today is clear **or** today is raining"
$P \leftrightarrow R$	:	"If and only if, yesterday was cloudy today it is raining"
$Q \vee (R \rightarrow P)$	:	"**Either** today is clear **or if** yesterday was cloudy **then** it is raining today"
$\sim R \wedge Q$	:	"Yesterday was **not** cloudy **and** today is clear"

To avoid an excess of parentheses in writing compound propositions in symbolic form, the above connectives are considered to be binding in the conventional order of precedence: negation "$\sim$", conjunction "$\wedge$", disjunctions "$\vee$", $\dot\vee$, implication "$\rightarrow$" and equivalence "$\leftrightarrow$". For example, $R \wedge S \rightarrow P$ means $(R \wedge S) \rightarrow P$ and $\sim R \wedge Q$ means $(\sim R) \wedge Q$.

For any assignment of truth values T or F to atomic propositions, depending upon the connectives used, the truth value of a compound proposition can be computed in a mechanical way by means of **truth tables**.

The truth values of six compound propositions, defined in terms of the truth values of propositions P and Q, for the main propositional connectives described earlier, are shown below in Table 2.

P	Q	$\sim P$	$P \wedge Q$	$P \vee Q$	$P \vee Q$	$P \rightarrow Q$	$P \equiv Q$
1	1	0	1	1	0	1	1
1	0	0	0	1	1	0	0
0	1	1	0	1	1	1	0
0	0	1	0	0	0	1	1

TABLE 2: Definition of Connectives

3.2 Reductions to Normal Forms

It is possible to define all propositional connectives in terms of a subset of them. For example, they can all be defined in terms of the set $(\wedge, \vee, \sim)$ so that a given expression can be converted into a "**normal form**". Such a subset is known as a **complete set of connectives**. This is accomplished by replacing a certain expression by another "equivalent" expression involving other connectives. Two expressions are said to be "**equivalent**" if. and only if, their truth values are the same, and this is expressed as $P \leftrightarrow Q$, that is P is equivalent to Q.

For example, $P \rightarrow Q \equiv \sim P \vee Q$, $\sim\sim P \equiv P$, $\sim(P \vee Q) \equiv \sim P \wedge \sim Q$, $\sim(P \wedge Q) \equiv \sim P \vee \sim Q$ are laws of propositional logic (the latter two are called the **De Morgan Laws**). The following two are called **Distributive Laws**:

$$P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$$
$$P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$$

In the first law, "$\vee$" distributes across "$\wedge$", while in the second law "$\wedge$" distributes across "$\vee$".

By De Morgan's laws, conjunction can always be expressed in terms of negation and disjunction. First use De Morgan laws to get negations against atomic propositions, and then recursively distribute "$\vee$" over "$\wedge$" where it applies. This transforms a general compound proposition R to an equivalent proposition of the form $R_1 \wedge R_2 \wedge \ldots R_n$ in which every R_i, $i=1, \ldots, n$ is a disjunction of atomic propositions. The logical form $R_1 \wedge R_2 \wedge \ldots R_n$ is called a **Conjunctive Normal Form (CNF)** for R and the R_i are **clauses** of the CNF. For example, $\sim P \wedge (Q \vee R) \rightarrow (S \vee T)$ becomes $(P \vee \sim Q \vee S \vee T) \wedge (P \vee \sim R \vee S \vee T)$.

The second distributive law can be used to transform R to an equivalent proposition of the form $S_1 \vee S_2 \vee \ldots S_m$ in which each $S_j, j=1, \ldots, m$ is a conjunction of atomic propositions or their negation. In this case $S_1 \vee S_2 \vee \ldots S_m$ is called a **Disjunctive Normal Form (DNF).** In both normal forms, negation is only applied to atomic propositions. All conjunctions may be removed leaving an expression entirely in "~" and "$\vee$". Similarly, all disjunctions may be removed leaving an expression entirely in "~" and "$\wedge$". Clearly, ($\sim,\vee$) or ($\sim,\wedge$) define complete sets of connectives. This implies that any expression can be converted to conjunction or disjunction of clauses by using the equivalent statements given in Table 3. It should be pointed out, however, that in general, a number of conjunctive or disjunctive normal forms are possible, leading to more than one representation for a particular compound proposition. Using the method described above, the most computationally efficient representation of a logical form is not necessarily achieved. The authors therefore aim to provide a systematic reformulation procedure with computer support that offers increased scope and applicability of mathematical programming.

No	Statement	Equivalent Forms	
1	$\sim P$	P	
2	$P \dot{\vee} Q$	$(\sim P \wedge Q) \vee (P \wedge \sim Q)$	Exclusion
3	$\sim(P \vee Q)$	$\sim P \wedge \sim Q$	De Morgan's Law
4	$\sim(P \wedge Q)$	$(P \rightarrow Q)) \wedge (Q \rightarrow P)$	
5	$P \rightarrow Q$	$\sim P \vee Q$	Implication
6	$P \leftrightarrow Q$ $P \equiv Q$	$(P \leftrightarrow Q) \wedge (Q \rightarrow P)$ $\sim P \wedge \sim Q \vee P \wedge Q$	
7	$P \rightarrow Q \wedge R$	$(P \rightarrow Q) \wedge (P \rightarrow R)$	
8	$P \rightarrow Q \vee R$	$(P \rightarrow R) \vee (P \rightarrow R)$	
9	$P \wedge Q \rightarrow R$	$(P \rightarrow R) \vee (Q \rightarrow R)$	
10	$P \vee Q \rightarrow R$	$(P \rightarrow R) \wedge (Q \rightarrow R)$	
11	$P \wedge (Q \vee R)$	$(P \wedge Q) \vee (P \wedge R)$	Distributive Law
12	$P \vee (Q \wedge R)$	$(P \vee Q) \wedge (P \vee R)$	
13	$P \wedge P \vee Q$	P	

TABLE 3: Transformation of Logical Statements into Equivalent Forms

3.3 Logic Forms represented by 0-1 variables and Linear (In) equalities

We wish to transform a compound proposition into a system of linear constraints so that the logical equivalence of the transformed expressions is maintained. The resulting system of constraints clearly must have the same truth table as the original statement, that is, the truth or falsity of the statement is represented by the satisfaction or otherwise of the corresponding set of linear equations and inequalities.

In order to explain the transformation process and the underlying principles more clearly, two cases are distinguished, namely, connecting logical variables and logically relating linear form constraints.

(i) Connecting logical variables

Let P_j denote the jth logical variable which takes values T or F and represents an atomic proposition describing an action, option or decision. Associate an integer variable with each type of action (or option). This variable, known as the binary **decision variable**, is denoted by "δ_j" and can take only the values 0 and 1 (binary). The connection of these variables to the propositions are defined by the following relations:

$$\delta_j = 1 \text{ iff proposition } P_j \text{ is TRUE}$$

$$\delta_j = 0 \text{ iff proposition } P_j \text{ is FALSE}$$

Imposing various logical conditions linking these different actions in a model, can be achieved by expressing these conditions in the form of linear constraints connecting the associated decision variables.

Using the propositional connectives given in Table 1, and the equivalent statements, given in Table 3, a list of standard form simple transformations T1.1 ... T1.23 are defined. These transformations are applied to the atomic proposition P_j and the associated 0-1 variable δ_j and using these transformations compound propositions are restated in linear algebraic forms, involving decision variables, so that the two expression are logically equivalent.

VARIABLE TRANSFORMATIONS

Statement	**Constrain**	**Transformation**
$\sim P_1$	$\delta_1 = 0$	T1.1
$P_1 \vee P_2$	$\delta_1 + \delta_2 \geq 1$	T1.2
$P_1 \dot{\vee} P_2$	$\delta_1 + \delta_2 = 1$	T1.3
$P_1 \wedge P_2$	$\delta_1 = 1, \delta_2 = 1$	T1.4
$\sim(P_1 \vee P_2)$	$\delta_1 = 0, \delta_2 = 0$	T1.5
$\sim(P_1 \wedge P_2)$	$\delta_1 + \delta_2 \leq 1$	T1.6
$P_1 \rightarrow \sim P_2$	$1 - \delta_2 \geq \delta_1$	T1.7
$P_1 \rightarrow P_2$	$\delta_1 - \delta_2 \leq 0$	T1.8
$P_1 \leftrightarrow P_2$	$\delta_1 - \delta_2 = 0$	T1.9
$P_1 \rightarrow P_2 \wedge P_3$	$\delta_1 \leq \delta_2, \delta_1$	T1.10

$P_1 \rightarrow P_2 \vee P_3$	$\delta_1 \leq \delta_2 + \delta_3$	T1.11
$P_1 \wedge P_2 \rightarrow P_3$	$\delta_1 + \delta_2 - \delta_3 \leq 1$	T1.12
$P_1 \vee P_2 \rightarrow P_3$	$\delta_1 \leq \delta_3, \delta_2 \leq \delta_3$	T1.13
$P_1 \wedge (P_2 \vee P_3)$	$\delta_1 = 1, \delta_2 + \delta_3 \geq 1$	T1.14
$\cdot_1 \vee (P_2 \wedge P_3)$	$\delta_1 + \delta_2 \geq 1, \delta_1 + \delta_3 \geq 1$	T1.15

The general forms of transformations T1.2, T1.3, T1.11, T1.12, may be stated as:

$P_1 \vee P_2 \vee ... P_n$	$\delta_1 + \delta_2 ... + \delta_n \geq 1$	T1.16
$P_1 \dot{\vee} P_2 \dot{\vee} ... P_n$	$\delta_1 + \delta_2 ... + \delta_n = 1$	T1.17
$P_1 \wedge ... P_k \rightarrow P_{k+1} \vee ... P_n$	$(1 - \delta_1) +...+ \delta_{k+1} +...+ \delta_n \geq 1$	T1.18
"at least K out of n are TRUE"	$\delta_1 + \delta_2 ... + \delta_n \geq k$	T1.19
"exactly k out of n are TRUE"	$\delta_1 + \delta_2 ... + \delta_n = k$	T1.20
"at most k out of n are TRUE"	$\delta_1 + \delta_2 ... \delta_n \leq k$	T1.21
$P_n \equiv P_1 \vee P_2 \vee ... P_k$	$\delta_1 + \delta_2 ... + \delta_k \geq \delta_n$, $(- \delta_j + \delta_n \geq 0, j = 1, ...l)$	T1.22
$P_n \equiv P_1 \wedge P_2 \wedge ... P_k$	$- \delta_1 - \delta_2 ... - \delta_k + \delta_n \geq 1 - k$, $(\delta_j - \delta_n \geq 0, j = 1, ..., k)$	T1.23

(ii) Logically relating linear form constraints

In order to reformulate "logical constraints in the general form", it is well known that finite upper or lower bounds on the linear form must be used Simonnard [SIMNRD66], Brearly, Mitra et al [BRMTWL75], Williams [WILLMS89].

Consider the linear restriction

$$LF_k : \sum_{j=1}^{n} a_{kj} x_j \left\{ \rho \right\} b_k$$

where ρ defines the type of mathematical relation, $\rho \epsilon \{ \leq, \geq, = \}$. Let L_k, U_i, denote the lower and upper bounds respectively, on the corresponding linear form, that is

$$L_k \leq \sum_{j=1}^{n} a_{kj}x_j - b_k \leq U_k$$

In our reformulation procedure, we use the finite bounds L_k and U_k. These bounds may be given or, alternatively, can be computed for finite ranges of x_j [BRMTWL75]. A "Logical Constraint in the Implication Form" (LGIF) is a logical combination of simple consraints and is defined as

If **antecedent** then **consequent**

where the antecedent is a logical variable and the consequent is a linear form constraint.

A "logical constraint in the general form" can be always reduced to an LGIF using well known transformations set out in Table 3. To model the LGIF, a 0-1 indicator variable is linked to the antecedent. Whether the constraint $\mathbf{LF_k}$ applies or otherwise is indicated by a 0-1 variable δ'_k,

$\delta'_k = 1$	iff the kth linear constraint applies
$= 0$	iff the kth linear constraint does not apply

A set of transformations T2 are defined below which illustrate this binary variable, namely the indicator variable of the antecedent using the bound value relates to the linear form restriction, that is the consequent.

CONSTRAINT TRANSFORMATIONS

Statement	**Constraint**	**Transform**
$\delta'_k = 1 \rightarrow x_k \geq L_k$	$x_k \geq L_k \delta'_k$	T2.1
$\delta'_k = 0 \rightarrow x_i \leq 0$	$x_k \leq U_k \delta'_k$	T2.2
$\delta'_k = 1 \rightarrow \sum_j a_{kj} x_j \leq b_i$	$\sum_j a_{kj} x_k - b_k \leq U_k(1-\delta'_k)$	T2.3

$\delta'_k = 1 \rightarrow \sum_j a_{kj} x_j \geq b_i$	$\sum_j a_{kj} x_k - b_k \geq L_i (1 - \delta'_k)$	T2.4
$\delta'_k = 1 \leftrightarrow \sum_j a_{kj} x_j = b_i$	T2.3 $(\delta'_k = 1 \rightarrow \sum_j a_{kj} x_k \geq b_k)$	T2.5
	T2.4 $(\delta'_k = 1 \rightarrow \sum_j a_{kj} x_j \geq b_k)$	

4. A SYSTEMATIC PROCEDURE FOR REFORMULATION

Having represented in the previous sections, compound propositions as (in)equalities, the next step is to model more complicated logical statements by further inequalities. As a result of the many, but equivalent, forms any logical statement can take, there are often different ways of generating the same or equivalent mathematical reformulations.

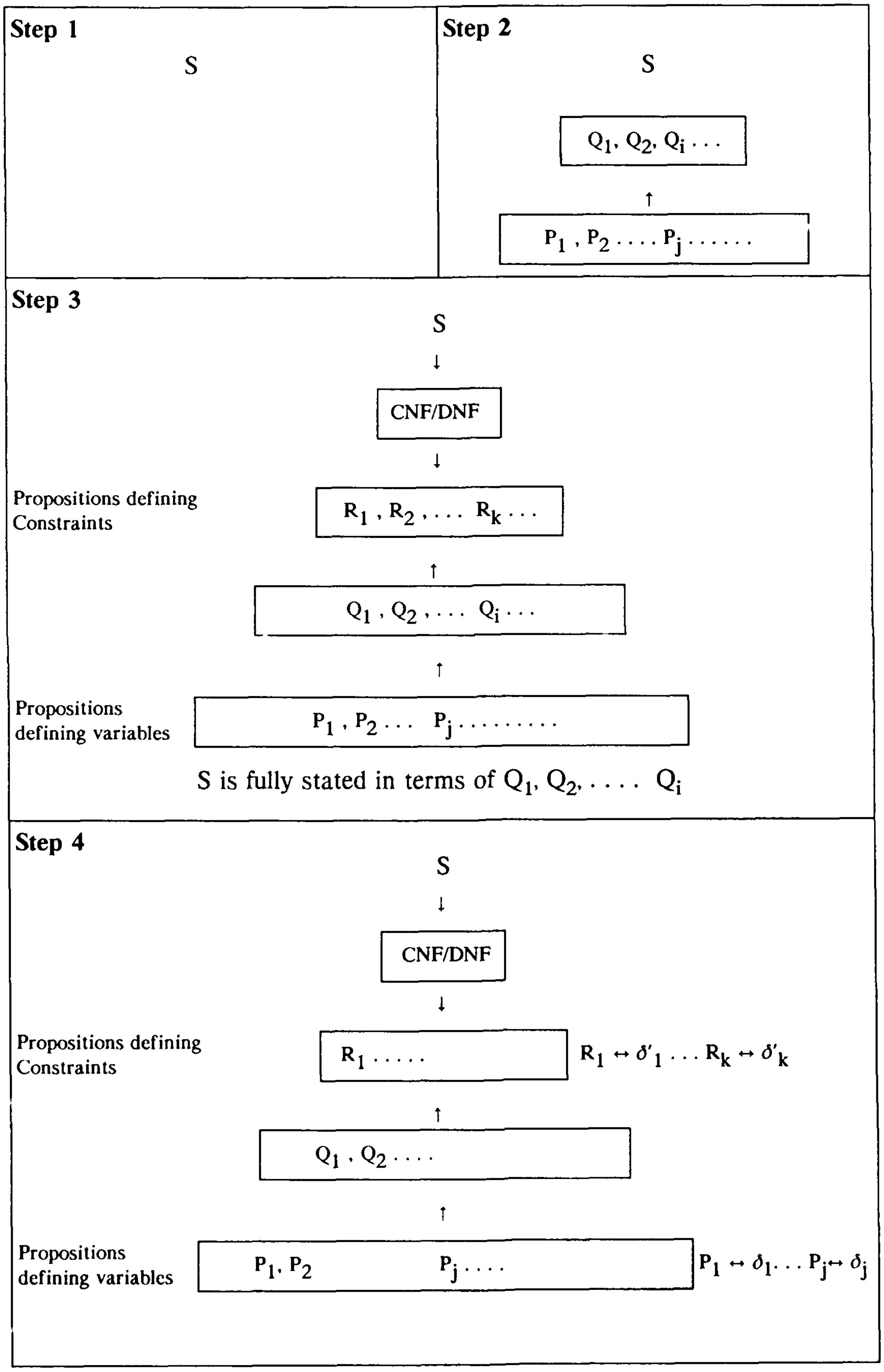
One possible way would be to convert the desired expression into a normal form such as the conjunction of disjunctive terms, the clauses. Each clause is then transformed into a linear constraint (applying transformation T1.16) so that the resulting CNF can be represented by a system of constraints, derived in this manner, which have to be satisfied invoking the logical "and" operation (an illustration of this method is presented in Example 1).

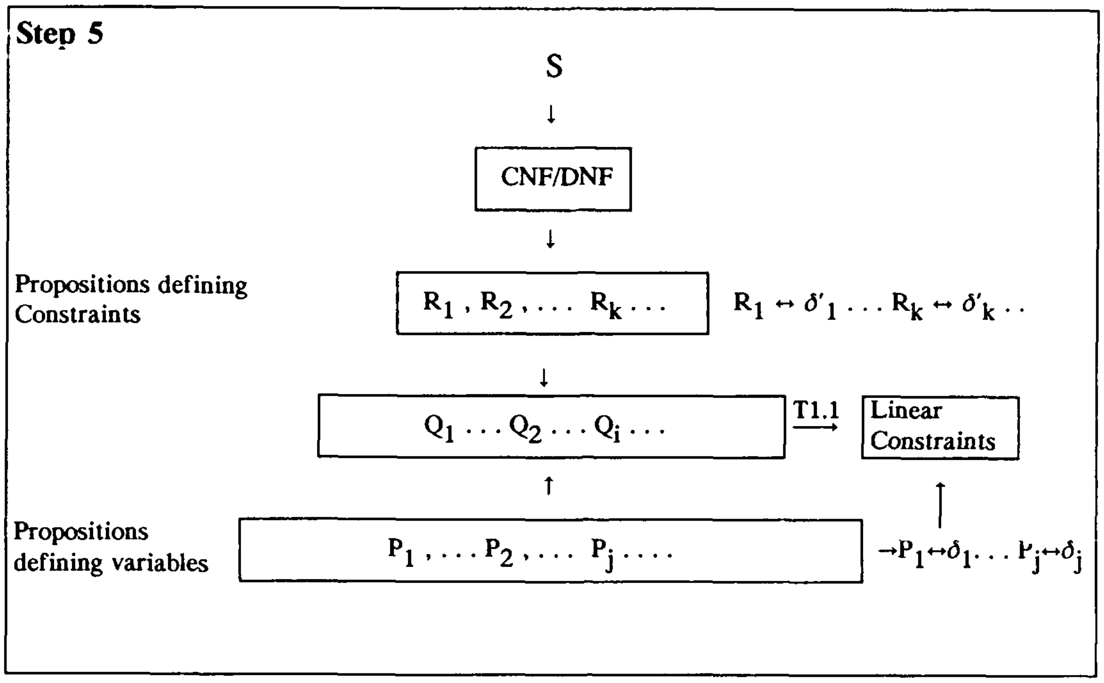
In the absence of a systematic approach, the above process appears to be unduly complicated. This has motivated the authors to propose a systematic procedure to reformulate a logical condition imposed on a model into a set of integer linear constraints. Such a procedure can be encapsulated by a sequence of progressive steps whereby the simple propositions are used to build compound propositions which are then combined further into more complicated statements. The way of proceeding is to reduce such statements into standard forms by using equivalent expressions and progressively transform them into mixed integer linear forms. The procedure involves the use of 0-1 decision variables and indicator variables as defined in the previous sections. Variable and constraint transformations are applied to the decision or indicator variables or both as the logical specification of the model requires. The conceptual build up of the model is illustrated by diagram 1 and diagram 2.

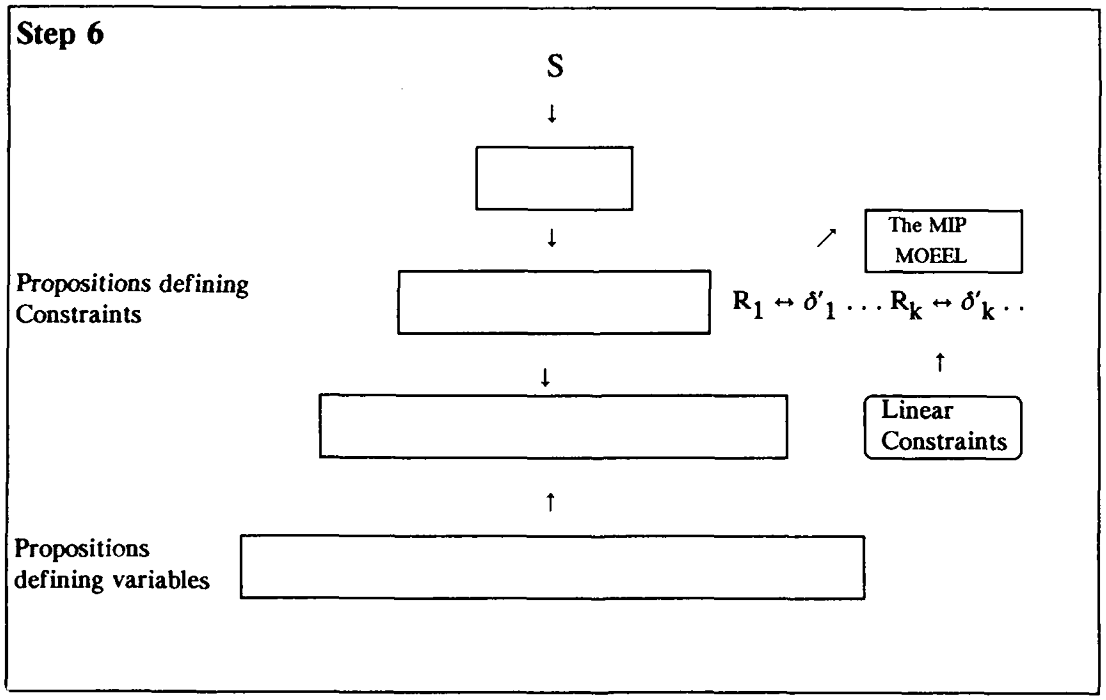
Step 1 Write explicitly the required condition, in words, in the form of a logical compound statement using known logical operators. Let S be the statement of the required condition.

Step 2 Decompose the statement S into simple (atomic) propositions: define these as atomic propositions P_j. Construct intermediate compound propositions Q_i using the atomic propositions P_j.

Step 3 Rewrite S in a symbolic form using the compound propositions Q_i and the connectives defined in Table 1. Simplify S into a normal form CNF


Step 1
S
Step 2
S
$Q_1, Q_2, Q_i \ldots$
$P_1, P_2 \ldots P_j \ldots$
Step 3
S
CNF/DNF
Propositions defining Constraints
$R_1, R_2, \ldots R_k \ldots$
$Q_1, Q_2, \ldots Q_i \ldots$
Propositions defining variables
$P_1, P_2 \ldots P_j \ldots$
S is fully stated in terms of $Q_1, Q_2, \ldots Q_i$
Step 4
S
CNF/DNF
Propositions defining Constraints
$R_1 \ldots$
$R_1 \leftrightarrow \delta'_1 \ldots R_k \leftrightarrow \delta'_k$
$Q_1, Q_2 \ldots$
Propositions defining variables
$P_1, P_2 \quad P_j \ldots$
$P_1 \leftrightarrow \delta_1 \ldots P_j \leftrightarrow \delta_j$


Step 5
S
CNF/DNF
Propositions defining Constraints
$R_1, R_2, \ldots R_k \ldots$
$R_1 \leftrightarrow \delta'_1 \ldots R_k \leftrightarrow \delta'_k \ldots$
$Q_1 \ldots Q_2 \ldots Q_i \ldots$
T1.1
Linear Constraints
Propositions defining variables
$P_1, \ldots P_2, \ldots P_j \ldots$
$\rightarrow P_1 \leftrightarrow \delta_1 \ldots P_j \leftrightarrow \delta_j$


Step 6
S
The MIP MOEEL
Propositions defining Constraints
$R_1 \leftrightarrow \delta'_1 \ldots R_k \leftrightarrow \delta'_k \ldots$
Linear Constraints
Propositions defining variables

or DNF using equivalent logical forms set out in Table 3. Identify the compound propositions R_k stated in terms of Q_i where R_k relate to the constraints of the model.

Step 4 Define 0-1 decision variables δ_j to represent the truth or falsity of each atomic proposition P_j. Also define 0-1 indicator variables δ'_k for each R_k which relate logically the different constraints of the problem.

Step 5 Transform each compound proposition Q_i into a linear form mathematical constraint using variable transformations (T1.1, T1.2 ...).

Step 6 Express the restrictions of the problem as LGIF's by linking the indicator variables to the mathematical constraints. Depending on the context, the decision variables can take the role of indicator variables. Transform the logical constraints into mathematical (in)equalities repeated applications of variable or constraint transformations. Compute bounds on linear forms as required. When a straightforward application of either transformation is not possible, it may be necessary to introduce more indicator variables and use them to replace a complex constraint by simpler ones.

5. ILLUSTRATIVE EXAMPLES

The following two examples are set out to illustrate the modelling of a logical condition using the reformulation procedure.

5.1 Example 1 ([WILLMS87])

"If 3 or more of products {1 to 5} are made, or less than 4 products {3 to 6, 8, 9} are made then at least 2 of products {7 to 9} must be made unless none of products {5 to 7} are made".

Step 1

Let S denote the above statement of the problem.

Step 2

Define the atomic propositions P_j : "Product j is made" ($j \epsilon J, J = \{1,2, ..., 9\}$).

Also define compound propositions

Q_1 : "at least 3 of P_1, P_2, P_3, P_4, P_5 are TRUE"

Q_2 : "at most 3 of $P_3, P_4, P_5, P_6, P_8, P_9$ are TRUE"

Q_3 : "at least 2 of P_7, P_8, P_9 are TRUE"

Q_4 : "none of P_5, P_6, or P_7 is TRUE" i.e. "$\sim(P_5 \vee P \vee P_7)$"

Step 3

Write S in symbolic form: $(Q_1 \vee Q_2) \wedge {\sim}Q_4 \rightarrow Q_3$.

In order to simplify S introduce compound propositions R_1 and R_2,

$R_1 : Q_1 \vee Q_2$

$R_2 : {\sim}Q_4$

Step 4

Define decision variables δ_j for each proposition P_j such that

$\delta_j = 1$ iff product j is made (P_j is TRUE), $j \epsilon J$

$\delta_j = 0$ otherwise (P_j is FALSE)

Introduce indicator variables δ'_1 and δ'_2 such that

$\delta'_1 = 1$ iff proposition R_1 (: $Q_1 \vee Q_2$) is FALSE

$= 0$ iff proposition R_1 is TRUE

$\delta'_2 = 1$ iff proposition R_2 (: ${\sim}Q_4$) is FALSE

$= 0$ iff proposition R_2 is TRUE

The logic of the definition of the indicator variables in this example has been reversed in order to derive a formulation which is comparable to that given by Williams.

Step 5

Apply variable transformations to propositions Q_i ($i = 1, \ldots, 4$) in order to convert them into linear constraints.

Using T1.19, proposition Q_1 can be represented by

$$\delta_1 + \delta_2 + \delta_3 + \delta_4 + \delta_5 \geq 3$$

Using T1.21, proposition Q_2 can be represented by

$$\delta_3 + \delta_4 + \delta_5 + \delta_6 + \delta_8 + \delta_9 \leq 3$$

Using T1.5, proposition Q_4 can be represented by

$$\delta_5 = 0,\ \delta_6 = 0,\ \delta_7 = 0$$

Using T1.19, proposition Q_3 can be represented by

$$\delta_7 + \delta_8 + \delta_9 \geq 2$$

Step 6

Relate logically the various constraints of the problem firstly by imposing the following conditions (LGIF's) :

LGIF - 1 : $\delta'_1 = 1 \rightarrow \sim (Q_1 \vee Q_2)$

LGIF - 2 : $\delta'_2 = 1 \rightarrow \sim (\sim Q_4)$

LGIF - 3 : $(\delta'_1 = 1 \wedge \delta'_2 = 1) \rightarrow Q_3$

Convert the above conditions in equivalent forms.

LGIF - 1: $\delta'_1 = 1 \rightarrow \sim Q_1 \wedge \sim Q_2$ – De Morgan's Law

$\delta'_1 = 1 \rightarrow \sim Q_1$

$\delta'_1 = 1 \rightarrow \sim Q_2$

$\delta'_1 = 1 \rightarrow \delta_1 + \delta_2 + \delta_3 + \delta_4 + \delta_5 \leq 2$ (1a)

$\delta'_1 = 1 \rightarrow \delta_3 + \delta_4 + \delta_5 + \delta_6 + \delta_8 + \delta_9 \geq 2$ (1b)

LGIF - 2: $\delta'_2 = 1 \rightarrow Q_4$

$\delta'_2 = 1 \rightarrow \delta_5 = 0 \wedge \delta_7 = 0$ (2)

LGIF - 3: $\delta'_1 = 1 \wedge \delta'_2 = 1) \rightarrow \delta_7 + \delta_8 + \delta_9 \geq 2$ (3)

Convert into linear constraints using constraint or variable transformations. Using T2.3, U may be taken as 3 (= 1+1+1+1+1- 2). This gives the following constraint representation of LGIF - 1_a.

$$\delta_1 + \delta_2 + \delta_3 + \delta_4 + \delta_5 + 3\,\delta'_1 \leq 5 \qquad (5.1.1)$$

Using T2.4, L may be taken as -4 (= 0+0+0+0+0+0-4). This gives the following constraint representation of LGIF - 1b :

$$\delta_3 + \delta_4 + \delta_5 + \delta_6 + \delta_8 + \delta_9 \geq 4\,\delta'_1 \qquad (5.1.2)$$

Using T1.10, we impose condition LGIF - 2 by the constraints

$$1 - \delta_5 \geq \delta'_2,\ 1 - \delta_6 \geq \delta'_2,\ 1 - \delta_7 \geq \delta'_2 \qquad (5.1.3)$$

To apply any constraint transformation on condition LGIF - 3, we need to simplify further by introducing another indicator variable δ'_3 such that

$\delta'_3 = 1$ if $(\delta'_1 = 0 \wedge \delta'_2 = 0)$, that is, both propositions R_1 and R_2 are TRUE.

Condition LGIF - 3 is then replaced by the following two conditions

$$(\delta'_1 = 0 \wedge \delta'_2 = 0) \rightarrow \delta'_3 = 1 \qquad (3a)$$

$$\delta'_3 = 1 \rightarrow \delta_7 + \delta_8 + \delta_9 \geq 2 \qquad (3b)$$

Using T1.12, condition LGIF - 3a is represented by

$$\delta'_1 + \delta'_2 + \delta'_3 \geq 1 \qquad (5.1.4)$$

and using condition T2.4 condition LGIF - 3b is represented by

$$\delta'_7 + \delta'_8 + \delta'_9 \geq 3\ \delta'_3 \qquad (5.1.5)$$

The complete IP representation of the condition S is given by the set of constraints (5.1.1) - (5.1.5):

$$\delta_1 + \delta_2 + \delta_3 + \delta_4 + \delta_5 + 3\ \delta'_1 \geq 5$$

$$\delta_3 + \delta_4 + \delta_5 + \delta_6 + \delta_8 + \delta_9 \geq 4\ \delta'_1$$

$$1 - \delta_5 \geq \delta'_2$$

$$1 - \delta_6 \geq \delta'_2$$

$$1 - \delta_7 \geq \delta'_2$$

$$\delta'_1 + \delta'_2 + \delta'_3 \geq 1$$

$$\delta_7 + \delta_8 + \delta_9 \geq 2\ \delta'_3$$

$$\delta'_1 , \delta'_2 , \delta'_3 \in \{ 0,1 \}$$

5.2 Example 2 : Crossword Compilation ([WILSON89])

This problem has a logical structure; it can be formulated in terms of Boolean Algebra and then converted into an integer programming system of constraints. The objective is to fill in an nxn full puzzle with complete interlocking using words from a given lexicon.

Define the following sets

I	the set of rows ($i \in I$)
M	the set of columns ($m \in M$)

J the set of letters of the alphabet ($j \in J$)

and let $n = |M| = |I|$. Given also is a lexicon of n-letter words.

To formulate the problem, the following set of logical conditions have to be modelled:

C1. Each cell (i,m) of the matrix must be occupied by **exactly one** letter of the alphabet.

C2. **If** cell (i,m) is occupied by letter j **then at least (n-1)** cells (i,m'), m'm must be occupied by letters $j' \in J_1(j)$ **and at least** (n-1) cells (i',m), $i' \neq i$ must be occupied by letters $j'' \in J_2(j)$ where

$j_1(j)$: set of letters which by virtue of the lexicon could appear in cells (i, m'), $i' \neq m$ given that letter j appears in cell (i,m).

$J_2(j)$: set of letters which by virtue of the lexicon could appear in cells (i', m), $i' \neq i$ given that letter j is in cell (i, m).

Conditions C1 and C2 are true for all cells (i, m), i=1, ..., n , m=1, ..., n and $j \in J$.

To model the above logical conditions, we apply the procedure of section 4.

Step 2

Define the individual propositions

p_{imj} : "Cell (i, m) is occupied by letter j" for all $i \in I$, $m \in M$ and $j \in j$

Define compound propositions

Q^1_{imj} : "at least (n-1) of $P_{im'j'}$ are TRUE for given $m' \neq m$, $j' \in J_1(j)$"

Q^2_{imj} : "at least (n-1) of $P_{i'mj''}$ are TRUE for given $i' \neq i$, $j'' \in J_2(j)$"

Step 3

Express conditions C1 and C2 in symbolic form:

C1 : $P_{im''A''} \dot{\vee} P_{im''B''} \dot{\vee} \ldots \dot{\vee} P_{im''Z''}$ for all $i \in I$, $m \in M$ and $j \in J$

C2 : $P_{imj} \rightarrow Q^1_{imj} \wedge Q^2_{imj}$ for all $i \in I$, $m \in M$ and $j \in J$

Using 3.7 of Table 3, C2 is equivalent to the statement :

$$(P_{imj} \rightarrow Q^1_{imj}) \wedge (P_{imj} \rightarrow Q^2_{imj}).$$

Step 4

Define decision variables δ_{imj} for each proposition P_{imj} such that

$$\delta_{imj} = 1 \text{ if letter } j \text{ is placed in cell } (i, m)$$
$$= 0 \text{ otherwise}$$

Step 5

Apply variable transformations to propositions $Q^1{}_{imj}$ and $Q^2{}_{imj}$ to obtain the following linear constraints

Using T1.19, $Q^1{}_{imj}$ can be represented by

$$\sum_{m' \neq m, j' \epsilon J_1(j)} \delta_{im'j'} \geq n-1 \qquad \text{for given } i, j, m$$

and $Q^2{}_{imj}$ can be represented by

$$\sum_{i' \neq i, j'' \epsilon J_2(j)} \delta_{i'mj''} \geq n-1 \qquad \text{for given } i, j, m$$

Step 6

Using transformation T1.3, condition C1 can be represented by

$$\sum_{j e J} \delta_{imj} = 1 \qquad \text{for all } i=1, \ldots, n \text{ and } m=1, \ldots, n \qquad (5.2.1)$$

C2 can be replaced by the following logical constraints:

$$\delta_{imj}=1 \rightarrow \sum_{m' \neq m, j' \epsilon J_1(j)} \delta_{im'j'} \geq n-1 \qquad \text{for all } i,j,m \quad (5.2.2)$$

$$\delta_{imj}=1 \rightarrow \sum_{i' \neq i, j'' \epsilon J_2(J)} \delta_{i'mj''} \geq n-1 \qquad \text{for all } i,j,m \quad (5.2.3)$$

(5.2.2) and (5.2.3) can be converted into the following inequalities by applying transformation T2.4 where the lower bound is taken as -(n-1):

$$\sum_{m' \neq m, j' \epsilon J_1(j)} \delta_{im'j'} \geq (n-1)\delta_{imj} \qquad \text{for all } i,j,m \quad (5.2.4)$$

$$\sum_{i' \neq i, j'' \epsilon J_2(j)} \delta_{i'mj''} \geq (n-1)\, \delta_{imj} \qquad \text{for all } i,j,m \quad (5.2.5)$$

This leads to the complete formulation of the problem set out below:

$$\sum_{j e J} \delta_{imj} = 1 \qquad \text{for all } i=1, \ldots, n \text{ and } m=1, \ldots, n$$

$$\sum_{m' \neq m, j' \epsilon J_1(j)} \delta_{im'j'} \geq (n-1)\delta_{imj} \qquad \text{for all } i,j,m$$

$$\sum_{i' \neq i, j'' \epsilon J_2(j)} \delta_{i'mj''} \geq (n-1)\delta_{imj} \qquad \text{for all } i,j,m$$

$$\delta_{imj} \epsilon \{0,1\} \vee i,=1, \ldots, n,\ m=1, \ldots, n \text{ and } j \epsilon J$$

6. IMPLEMENTATION WITHIN AN LP MODELLING SYSTEM

Systems and languages for specifying LP and IP models are well established as practical tools for constructing optimisation applications. Fourer [FOURER83] provided an excellent summary of the state of the art at the beginning of the 80's and an update of this information can be found in the two special issues [MITRAG89] on this topic. We intend to incorporate the reformulation method described in this paper into CAMPS which is an interactive modelling system [LUCMIT88]. The design objectives of this modelling system are broad: the system is set out to help non-expert LP users to come to grips with the task of conceptualising and describing LP models, whereas the expert LP user is also supported in his requirements to construct large and complex LP models. We are not aware of any MP modelling system which provides reformulation support as described in this paper.

Consider the main menu, the modelling menu (MODEL), and the information flow diagram of CAMPS as set out in Diagrams 3, 4 and 5. The option REFORMULATION in Diagram 5 is introduced to encapsulate the automatic transformations and constraint generation described in this paper. The REFORMULATION menu in the prototype form is shown in Diagram 6.

1. MODEL
2. GENERATE
3. OPTIMISE
4. REPORT
5. UTILITIES
6. LOGOUT

DIAGRAM 3

1. NAMES
2. DIMENSIONS
3. TABLES
4. VARIABLES
5. CONSTRAINTS
6. REFORMULATION
7. RETURN

DIAGRAM 5

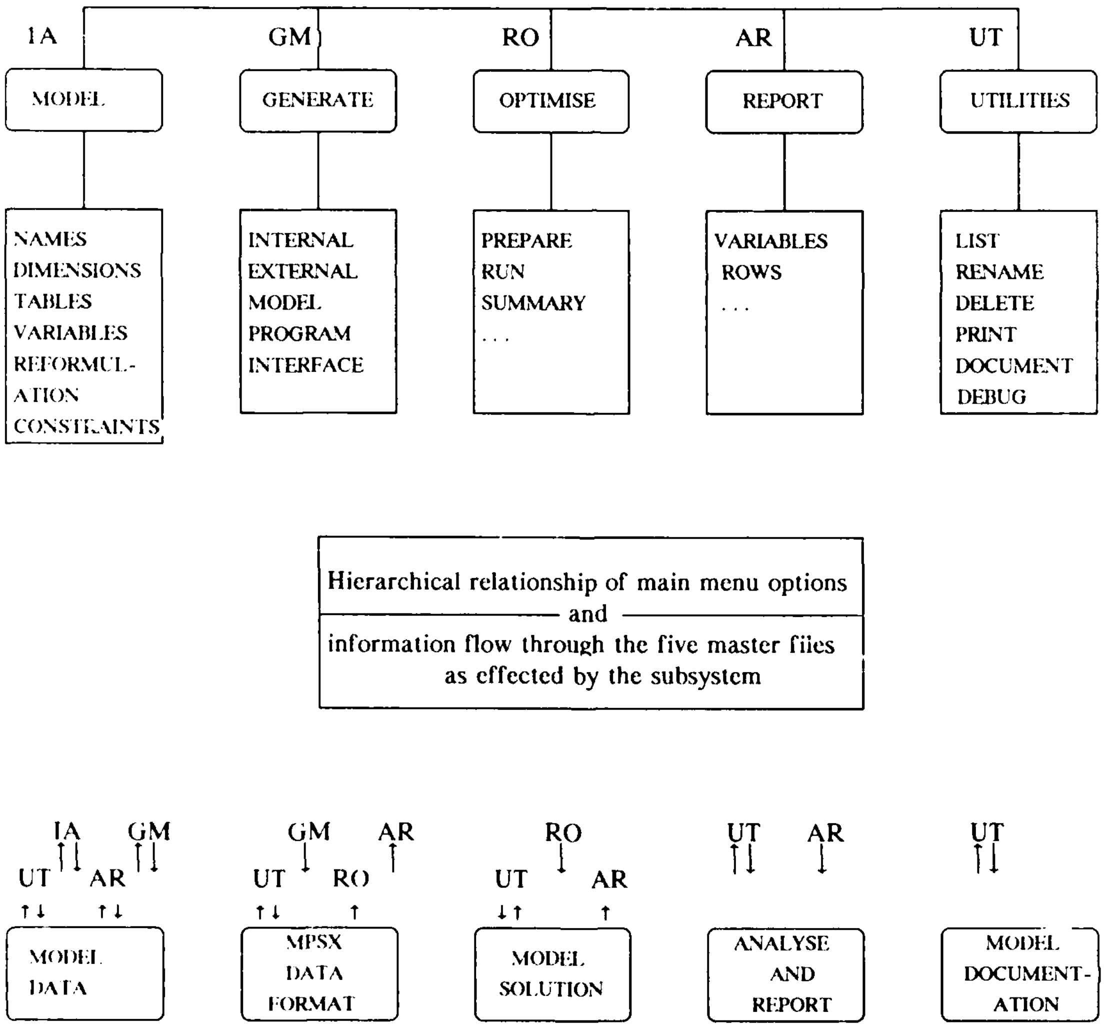

IA
GM
RO
AR
UT
MODEL
GENERATE
OPTIMISE
REPORT
UTILITIES
NAMES DIMENSIONS TABLES VARIABLES REFORMUL-ATION CONSTRAINTS
INTERNAL EXTERNAL MODEL PROGRAM INTERFACE
PREPARE RUN SUMMARY . . .
VARIABLES ROWS . . .
LIST RENAME DELETE PRINT DOCUMENT DEBUG
Hierarchical relationship of main menu options
and
information flow through the five master files as effected by the subsystem
IA GM UT AR
MODEL DATA
GM AR UT RO
MPSX DATA FORMAT
RO UT AR
MODEL SOLUTION
UT AR
ANALYSE AND REPORT
UT
MODEL DOCUMENT-ATION

DIAGRAM 4

REFORMULATION

1. STATEMENT
2. DIMENSIONS
3. PROPOSITIONS
4. IP - VARIABLES
5. EQUIVALENT FORMS
6. TRANSFORMATIONS
7. BOUNDS
8. RETURN

Diagram 6.

STATEMENT simply records a compact textual natural language statement of the problem against the global model statement S.

Since REFORMULATION is a submenu of MODEL all the existing attributes of a given model such as DIMENSIONS, VARIABLES and CONSTRAINTS defined in the model are inherited. The DIMENSION option is therefore a continuation of DIMENSION in the parent menu. Under PROPOSITIONS, P_j, Q_i, and R_k (see last two sections) are defined in that order. As in main CAMPS approach where arithmetic operators are prompted here logical oprators are prompted and chosen to define the LOGICAL Forms. The full statements of the propositions are also entered here and used later for the purpose of documentation.

IP-VARIABLES option is used to define the δ_i, δ'_k, decision and variables. EQUIVALENT FORMS and TRANSFORMATIONS simply display the information in Table 3 and the list of transformations T1.1...T1.18, respectively. They also allow these to be chosen for the reformulation procedure. In relating constraints logically to each other it may be necessary to compute bounds on the linear forms. Such bounds can be derived by invoking the BOUNDS option. A full system specification for implementation of the reformulation support is given in [MT...90]. This report also contains examples of dialogue for the illustrative problems.

7. DISCUSSION AND CONCLUSIONS

In this paper we have first reviewed the relationship between logical forms, the methods of computing inferences either symbolically or quantitatively and the discrete programming methods. The important connectives with AI and logic programming have also been reviewed. A systematic procedure for reformulating logic forms to IP and MIP forms is described and illustrated by two representative examples. A blue print for integrating this automatic procedure within an iteractive modelling system is then put forward. Constraint logic programming uses simple unsophisticated algorithms for constraint satisfaction. In contrast computational mathematical programming is concerned with efficient algorithms exploiting problem structure and has many instances of success in large and complex applications. The ideas put forward here add to the conceptual foundations of intelligent modelling systems for Mathematical Programming. We also hope the research reported in this paper will provide motivation to bring the work of CLP and MP communities closer together.

8. REFERENCES

ALLENJ83] Allen, J. F., 1983, "Maintaining knowledge about temporal intervals", **Comm. ACM., 26.**

[BCHNKM89] Brown, R. G., Chinneck, J. W. and Karam, G. M., 1989, "Optimization with constraint programming systems", **Impact of Recent Advances in Computing Science on Operations Research,** R Sharda et al editors, North Holland, 463-473.

[BEAUMN87] Beaumont, N., 1987, "An algorithm for disjunctive orograms", Monash University, Victoria 3168, Australia.

[BERGHZ87] Berghel, H., 1987, "Crossword compilation with horn clauses", **The Computer Journal, 30,** 183-188.

[BNPRLG88] Bell Northern Research, 1988, Prolog Language Description, Version 1.0, **O Hawa**, Canada.

[BRMTWL75] Blair, C. E., Jeroslow, R. G., Lowe, J. K., 1988, "Some results and experiments in programming techniques for propositional logic", **Computers and Operations Research, 13,** (5), 633-645.

[BREAR75] Brearley, A. L., Mitra, G., Williams, H. P., 1975, "Analysis of mathematical programming problems prior to applying the simplex algorithm", **Mathematical Programming, 8,** 54-83.

[BSHORT84] Buchanan, B. G. and Shortliffe, E. H., 1984, "Rule based expert systems: the MYCIN experiments of the heuristic programming project, **Addison-Wesley,** Reading, Mass., USA.

[CHINBK89] Chinneck, J. W., Brown, R. C., Karam, C. M., 1989, "Optimisation with constraint programming systems", **Proceedings of "Impact of Recent Advances in Computer Science on Operations Research",** Williamsburg.

[COLMRA87] Colmerauer, A., 1987, "Opening the PROLOG III Universe", **Byte Magazine,** August 1987.

[DOWGLR84] Dowling, W. F. and Gallier, J. H., 1984, "Linear time algorithms for testing satisfiability and horn forumlae, **Journal of Logic Programming, 3,** 267-284.

[GARDNR61] Gardener, M., 1961, "More mathematical puzzles and diversions", **Penguin Books.**

[HNRYCK89] Hentenryck, P. V., 1989, "Constraint satisfaction in logic Programming", **MIT Press,** Massachusetts, USA.

[HOOKER88] Hooker, J. N., 1988, "A quantitative approach to logical inference", **Decision Support Systems, 4,** 45-69.

[HOOK ?] Hooker, J. N., ?, "Resolution versus cutting plane solution of inference problems: some computational experience", **Operations Research Letters ?**

[JERSLW85] Jeroslow, R., 1985, "An extension of mixed integer programming models and techniques to some database and artificial intelligence settings", **Working Paper**, Georgia Institute of Technology, Atlanta, California.

[JERWAN87] Jeroslow, R. G., Wang, J., 1987, "Solving propositional satisfiability problems", working paper, Georgia Institute of Technology, Atlanta, CA.

[JERWAN89] Jeroslow, R., Wang, J., 1989, "Programming integer polyhedra and horn clause knowledge bases", **ORSA Journal on Computing, 1,** (1), 7-19.

[LARIER78] Lauriere, J. L., 1978, "A language and a program for stating and solving combinatorial problems", **Artificial Intelligence, 10,** 29-127.

[LASSZC87] Lassez, C., 1987, "Constraint logic programming", **Byte Magazine,** August 1987, 171-176.

[LAUR76] Lauriere, 1976

[LUCMIT88] Lucas, C. and Mitra, G., 1988, "Computer-assisted mathematical programming (modelling0 system: CAMPS" , **The Computer Journal, 31,** 4, 1988.

[MITRA87] Mitra, G., 1989, ?? **IMA Journal of Mathematics in Management,** Oxford University Press.

[POST ?] Post, S., ? , "Reasoning with incomplete and uncertain knowledge as an integer linear program", Planning Research Corporation, Virginia 22102.

ehgmrefs

[QUINEW55] Quine, W. V., 1955, "Away to somplify truth functions", **American Mathematical Monthly, 62**, 627-631.

[REGNWN83] Reggia, J. A., Nau, D. S. and Wang, P. Y., 1983, "Diagnostic expert systems based on a set covering model", **Int. Jour. Man-Machine Studies, 19**, 437-460.

[ROBINS65] Robinson, J.A., 1965 "A mchine oriented logic basic on the resolution principle", **Journal of the ACM, 12**, 23-41.

[ROEHR88] Roehrig, S. F., 1988, "A pivoting approach to the solution of inference problems", US Coast Guard R&D Center, Croton CT, 06340.

[SIMNRD66] Simmonard, M., *1966, Linear Programming, Prentice Hall.*

[STEEL] Steel, S., 1987, "Tutorial notes", **AISB Tutorial.**

[WILLMS77] Williams, H. P., 1977, "Logical problems and integer programming", **Bulletin of the Institute of Mathematics and its Applications", 13**, 18-20.

[WILLMS85] Williams, H. P., 1985, "Model building in mathematical Programming", Wiley, New York.

[WILLMS87] Williams, H. P., 1987, "Linear and integer programming applied to the propositional calculus", **International Journal of Systems Resarch and Information Science, 2,** 81-100.

[WILLMS 89] Williams, H. P., 1989, "Constructing integer programming models fby the predicate calculus", **Annals of Operations Research,** ?

[WILSON89] Wilson, J. M., 1989, "Crossword compilation using integer programming", **The Computer Journal, 32,** (3), 273-275.

[YAGERR85] Yager, R. R., 1985, "Explanatory models in expert systems", **Int. Jour. Man-Machine Studies, 23**, 539-549.