TR/11/90 November 1990
Revised April 1991

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

E. Hadjiconstantinou and G. Mitra

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

E Hadjiconstantnou
Imperial College, London

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)

w9198854

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 Mixed Integer Programming (MIP) formulation 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 prepositional 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 and the methods of formulating these are discussed. A reformulation procedure which uses the extended reverse polish representation of a compound logical form is then described. These reformulation procedures are illustrated by two examples. A scheme of implementation.ithin an LP modelling system is outlined.

1. INTRODUCTION

In recent times first order logic in the form of prepositional 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 chaining 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 prepositional 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 successive generations of powerful computers logicians have been able to handle problems of limited size. Consequently, an alternative quantitative approach has been under invest-igation in recent years. The current indications are that these quantitative schemes to represent and solve problems of prepositional or predicate logic lead to computationally superior inference procedures. The upsurge of interest in applying mathematical programming to problems in prepositional 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 prepositional logic. This enables the development of modelling procedures by which statements in prepositional logic can be represented as discrete optimization problems involving 0-1 integer, and continuous variables; that is, integer programmes (IP) and mixed integer rogrammes (MIP).

(ii) Close parallels exist between some important concepts of prepositional 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 (prepositional 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 prepositional 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 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 prepositional logic and the corresponding 0-1 discrete programming equivalent forms. In section 4 these reformulation techniques are used in a progressivece 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 prepositional 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 [HOOKERS 8b]. 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 [HOOKER88a] surveys the application of quantitative methods, and integer programming methods in particular as applied to inference problems in prepositional logic. Williams [WILLMS77, WTLLMS85], has shown how such problems could be modeled 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 prepositional 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 [WILLMS74].

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 prepositional 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 relaxations are created). Taking into consideration the geometry of the convex hull Jeroslow [JERSLW85] 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 [BLARJL88] were apparently the first to solve non-trivial inference problems with mathematical programming methods. They examined the connections and parallels between prepositional 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 the branch-and-bound tree search. 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 prepositional 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 [HOOKER88b].

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 prepositional 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 [STEEL87] 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 [HENTRK89] and Chinneck et al [CHINBK89]. naturally the simplex algorithm is applied to achieve constraint satisfaction; Lassez CLP(R) [LASSEC87] and Colmerauer [COLMRA87] PROLOG III are two such CPSs. For constraints stated in discrete integers (natural numbers) tree search method or interval arithmetic is applied to achieve constraint satisfaction. Hentenryck [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 as 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 prepositional or logical **connectives** and they are 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.

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 disjuction	$P \vee Q$	P or Q	Either P or Q/at least one of P or Q
4	non-equivalence (exclusive disjunction)	$P \neq Q$ $(P \vee Q)$	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 iff Q	P if and only if Q/P is a necess- ary 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 nand Q	Not both P, Q

TABLE 1: Propositional Connectives

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: "$3x + 4y \leq z$"
(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:

~P	:	"It is **not** raining today"
Q v P	:	"Today is clear **or** today is raining"
P ↔ R	:	"If and only if, yesterday was cloudy today it is raining"
Q v (R → P)	:	"**Either** today is clear **or if** yesterday was cloudy then **it** is raining today"
~R ∧ 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 "~", conjunction "^", disjunctions "∨", ⊻, implication "→" and equivalence "↔". For example, **R ^ S→ P** means **(R ∧ S) → P** and **~R ∧ Q** means **(~R) ∧ 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 prepositional connectives described earlier, are shown in Table 2.

			7		7		
P	Q	~P	P^Q	PvQ	P∨&Q	P→Q	P≡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 prepositional connectives in terms of a subset of them. For example, they can all be defined in terms of the set (∧,∨, ,~) 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↔Q (or P≡Q),** that is **P** is equivalent to **Q.**

For example, **P→Q≡~PvQ,** and **~~P≡P,** are equivalent expressions. The following laws of prepositional logic are known as **De Morgan's Laws** and **Distributive Laws:**

De Morgan's Laws

~(P∨Q) ≡ ~P∧~Q
~(P∧Q) ≡ ~P∨~Q

Distributive Laws

P∨ (Q∧R) ≡ (P∨Q) ∧(P∨R)
P∧ (Q∨R) ≡ (P∧Q) ∨ (P∧R)

In the first law, "∨" distributes across "∧", while in the second law "∧" distributes across "∨".

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

Similarly De Morgan's laws followed by the second distributive law are applied to transform R to an equivalent proposition of the form $S_1 \vee S_2 \vee ... S_m$ in which each S_i,j=l, ..., m is a conjunction of atomic propositions or their negation. In this case $S_1 \vee S_2 \vee ... S_m$ 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 "$\sim$" and "$\vee$". Similarly, all disjunctions may be removed leaving an expression entirely in "$\sim$" and "$\wedge$". Clearly, $(\sim,\vee)$ or $(\sim,\wedge)$ define complete sets of connectives. This implies that any expression can be converted to a 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, whereby alternative (discrete) mathematical programming formulations can be constructed for a given logic form.

No	Statement	Equivalent Forms	
1	$\sim\sim P$	P	
2	P∨&Q	$(\sim P\wedge Q)\vee(P\wedge\sim Q)$	Exclusion
3	$\sim(P\vee Q)$	$\sim P\wedge\sim Q$	De Morgan's Laws
4	$\sim(P\wedge Q)$	$\sim P\vee\sim Q$	
5	$P\rightarrow Q$	$\sim P\vee Q$	Implication
6	$P\leftrightarrow Q$ or	$(P\rightarrow Q)\wedge(Q\rightarrow P)$	.
	$(P\equiv Q)$	$(\sim P\vee Q)\wedge(\sim Q\vee P)$	.
7	$P\rightarrow Q\wedge R$	$(P\rightarrow Q)\wedge(P\rightarrow R)$	.
8	$P\rightarrow Q\vee R$	$(P\rightarrow Q)\vee(P\rightarrow R)$	.
9	$P\wedge Q\rightarrow R$	$(P\rightarrow R)\wedge(Q\rightarrow R)$	.
10	$P\vee Q\rightarrow R$	$(P\rightarrow R)\wedge(Q\rightarrow R)$	.
11	$P\wedge(QVR)$	$(P\hat{}Q)\vee(P\hat{}R)$	Distributive Laws
12	$P\vee(Q\wedge R)$	$(P\vee Q)\wedge(PVR)$	

TABLE 3: Transformation of Logical Statements into Equivalent Forms

3.3 Polish Notation and Expression Trees

Using the normal precedence operators and the conventional evaluation of expressions the following logical form

$P \vee Q \vee \sim R \wedge S$

would be written as

$((P \vee Q) \vee ((\sim R) \wedge S))$.

Not using brackets as above but simply placing the operator symbols at the nodes, one can build up a tree representation which was discovered by LuKasiewiz [LUKSWZ63] and is well known as the Polish notation. Choice of the directions in which the variables and symbols are scanned leads to two well known variations, namely, forward (right to left scan) or reverse (left to right scan) Polish notation. The Polish notation for an expression is not unique and within forward Polish, for instance, early-operator form or late-operator form lead to two different notations and corresponds to inserting Church's brackets [CHURCH44] from the left or from the right respectively. The given expression can be written as

$((P \vee Q) \vee (\sim R \wedge S))$

or

$(P \vee (Q \vee (\sim R \wedge S)))$.

The tree representation for the first of these expression is shown in Diagram 3.1.

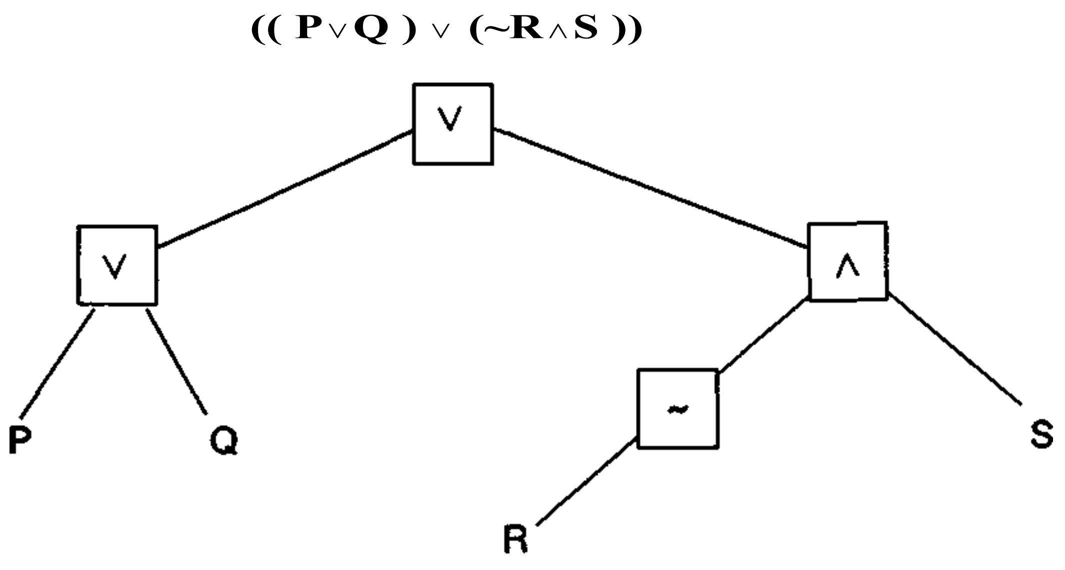


Diagram 3.1

For our purposes we call this an "expression tree". We note the limitation of Prepositional calculus with only unary and binary logical operators. The extended logical operators which involve n-tuples and n-place predicates (see next section where we introduce connectives such as ″exactly k out of n″, ″at most k out of n″) can be used to construct ″extended expression trees″. We give illustrations of extended expression trees in the examples discussed in section 5.

Reverse Polish notation found natural application in algebraic expression evaluation in compilers for automatic computers. It is no coincidence therefore that we find that our reformulation (translation) procedure is based upon this fundamental representation that is the ″extended expression tree″.

3.4 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_i denote the $_i$th 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:

$\delta_j = 1$ iff proposition P_j is **TRUE**

$\delta_j = 0$ iff proposition P_j is **FALSE**

Impositioin of logical conditions linking the different actions in a model is achieved by expressing these conditions in the form of linear constraints connecting the associated decision variables.

Using the prepositional connectives given in Table 1, and the equivalent statements, given in Table 3, a list of standard form ″variable transformations″ Tl.1 ... T1.23 are defined. These transformations are applied to compound propositions in volving one or more atomic propositions P_i, whereby the compound propositions are restated in linear algebraic forms involving decision variables. The two expressions are logically equivalent.

TABLE 4: VARIABLE TRANSFORMATIONS

Statement	Constraint	Transformation
$\sim P_1$	$\delta_1 = 0$	T1.1
$P_1 \vee P_2$	$\delta_1 + \delta_2 \geq 1$	T1.2
$P_1 \vee\!\!\& P_2$	$\delta_1 + \delta_2 = 1$	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
$P_1 \rightarrow \sim P_2$	$\delta_1 + \delta_2 \leq 1$	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
$P_1 \rightarrow P_2 \wedge P_3$	$\delta_1 \leq \delta_2, \delta_1 \leq \delta_3$	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
$P_1 \vee (P_2 \wedge P_3)$	$\delta_1 + \delta_2 \geq 1,\ \delta_1 + \delta_3 \geq 1$	T1.15

Some general forms of transformations are stated below:

$P_1 \vee P_2 \vee \ldots P_n$	$\delta_1 + \delta_2 \ldots + \delta_n > 1$	T1.16
$P_1 \vee\!\!\& P_2 \vee\!\!\& \ldots P_n$	$\delta_1 + \delta_2 \ldots + \delta_n = 1$	T1.17
$P_1 \wedge \ldots P_k \rightarrow P_{k+1} \vee \ldots P_n$	$(1 - \delta_1) \ldots + \delta_{k+1} + \ldots + \delta_n \geq 1$	T1.18
"at least K out of n are TRUE"	$\delta_1 + \delta_2 \ldots + \delta_n \geq k$	T1.19
"exactly k out of n are TRUE"	$\delta_1 + \delta_2 \ldots + \delta_n = k$	T1.20
"at most k out of n are TRUE"	$\delta_1 + \delta_2 \ldots + \delta_n \leq k$	T1.21

$P_n \equiv P_1 \vee P_2 \vee \ldots P_k$ $\delta_1 + \delta_2 \ldots + \delta_k \geq \delta_n,$ T1.22
$(-\delta_j + \delta_n \geq 0, j = 1, \ldots, k)$

$P_n \equiv P_1 \wedge P_2 \wedge \ldots P_k$ $-\delta_1 - \delta_2 \ldots - \delta_k + \delta_n \geq 1-k,$ T1.23
$(\delta_j - \delta_n \geq 0, j = 1, \ldots, k)$

(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 form restriction

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

where ρ defines the type of mathematical relation, $\rho \in \{\leq, \geq, =\}$. Let L_k, U_k, 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_i [BRMTWL75].

A "Logical Constraint in the Implication Form" (LCIF) 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 LCIF using standard transformations. To model the LCIF, a 0-1 indicator variable is linked to the antecedent. Whether the linear form constraint $\mathbf{LF_k}$ applies or otherwise is indicated by a 0-1 variable δ'_k ,

$\delta'_k = 1$ iff the kth linear restriction applies

$= 0$ iff the kth linear restriction does not apply

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

TABLE 5: CONSTRAINT TRANSFORMATIONS

Statement	**Constraint**	**Transform**
$\delta'_k = 1 \rightarrow x_k \geq L_k$	$x_k \geq L_k \, \delta'_k$	T2.1
$\delta'_k = 0 \rightarrow x_k \leq 0$	$x_k \leq U_k \, \delta'\, k$	T2.2
$\delta'_k = 1 \rightarrow \sum_j a_{kj} x_j \leq b_k$	$\sum_j a_{kj} x_j \; -b_k \leq U_k(1- \delta'_k)$	T2.3
$\delta'_k = 1 \rightarrow \sum_j a_{kj} x_j \geq b_k$	$\sum_j a_{kj} x_j \; -b_k \geq L_k \, (1- \delta'_k)$	T2.4
$\delta'_k = 1 \rightarrow \sum_j a_{kj} x_j = b_k$	T2.3 ($\delta'_k = 1 \rightarrow \sum_j a_{kj} x_j \leq b_k$)	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.

In the absence of a systematic approach, the above process appears to be unduly complicated. This has motivated us to propose a systematic procedure to reformulate a logical condition imposed on a model into a set of integer linear constraints. Our approach, in essence, involves identifying a precise compound statement of the problem and then processing this statement. This compound statement (S) is represented as an extended expression tree by the Polish notation (see section 3.3) and two working stack mechanisms, namely VSTACK for variables and CSTACK for constraints are created. The expression tree is traversed, that is, the expression is analysed and constraints are created (using variable and constraint transformations of section 3) in CSTACK using variables which are introduced in VSTACK. The steps of the procedure which fully processes and resolves the tree are set out below.

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

STEP 2 Identify simple (atomic) propositions P_i which can be used to state S. Express S in terms of the (extended) set of logical connectives - ″not″, ″and″, ″or″, ″implies″ (see Table 1), ″at least k out of n″, ″exactly k out of n″, ″at most k out of n″ (see Table 4 for these extensions), and the atomic propositions P_i. Use Church's brackets to indicate precedence of sub-expressions. If necessary, apply transformations from Table 3 to obtain an equivalent statement of S.

STEP 3 Construct the expression tree for S based on the forward/backward Polish notation whereby each logical connective in S is used as a predicate, that is, connective - name (list of arguments).

Construct this tree using the (extended) set of connectives as intermediate nodes and the simple propositions P_i or their negations as terminal nodes. Any subtree represents a compound proposition.

Define 0-1 decision variables δ_i to represent the truth or falsity of each one of the simple propositions P_J, that is

$$\delta_j = 1/0 \qquad \text{iff } P_i \text{ is true/false}$$

Introduce the variables δ_j into VSTACK.

STEP 4 Traverse the tree from the bottom, that is, use the terminal node index k to identify the corresponding compound proposition Q_k (subtree) and the related 0-1 indicator variable $\delta_k^{'}$. Introduce $\delta_k^{'}$ again into the VSTACK.

Convert the first-order compound propositions at the lowest levels of the tree into associated linear restrictions using Table 4 of variable transformations. Introduce these into CSTACK.

Apply the constraint transformations of Table 5 to convert the resulting LCIF $\delta_k^{'} = 1 \rightarrow Q_k$, into an integer linear restriction for this node. Pop-up the most recently placed constraint in CSTACK and then insert this new restriction into CSTACK. All terminal nodes are resolved in this way and the resulting integer linear constraints are inserted in the 'constraint' stack.

STEP 5 Continue traversal of the tree upwards by processing all nodes of the tree in the following way.

Introduce an indicator variable $\delta_k^{'}$ for any node k at intermediate to top levels in the tree, and update VSTACK.

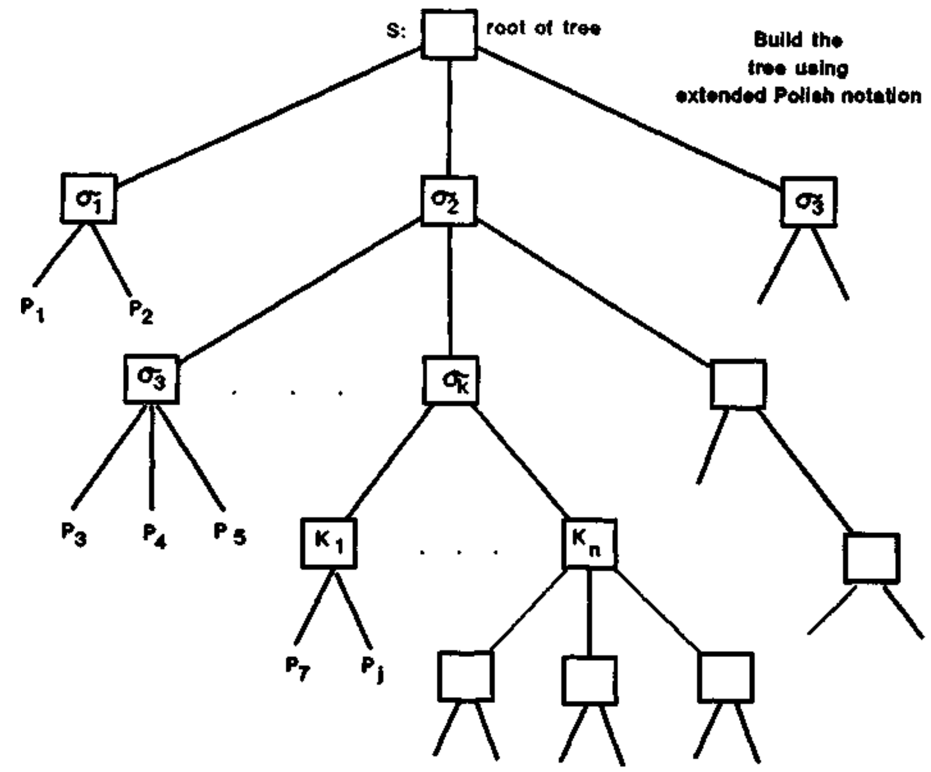
Produce an LCIF for this node involving $\delta_k^{'}$ and the compound propositions $Q_{k1,...,}Q_{kn}$ or their associated indicator variables $\delta_{k1}^{'}$,..., $\delta_{kn}^{'}$ corresponding to the n branches of node k.

Apply the variable and constraint transformations of Tables 4 and 5, respectively, to convert the resulting LCIF into an integer linear restriction and add it to CSTACK.

If at any node in the two highest levels of the tree, a standard tree representation from Table 4 is identified and all associated nodes are resolved, do not introduce a new indicator variable for this node, but simply add the corresponding integer constraint, as obtained from Table 4, directly to CSTACK. The node is then considered resolved.

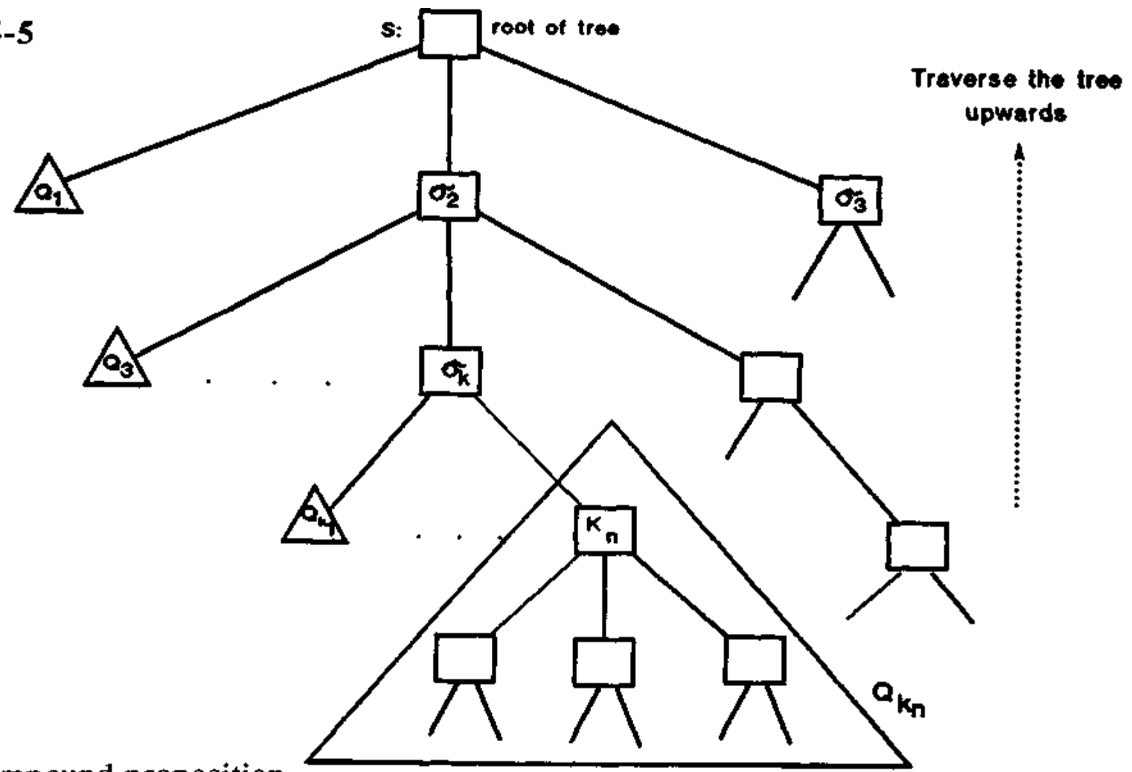
STEP 6 If all nodes of the tree are resolved then stop. At the end of the procedure, CSTACK contains all integer linear constraints and the VSTACK contains the decision and indicator variables used by these constraints.

STEPS 1-3



σ_k a symbol taken from the extended set of connectives
P_j atomic or simple proposition or its negation represented by terminal node.

STEPS 4-5



Q_k Compound proposition
Q_k replaces the subtree which has been resolved to derive this proposition at node k

Apply constraint transformations (Table 5) to convert the resulting LCIF's into integer linear restrictions:

Node 4 (using T2.4) : $\delta_1 + \delta_2 + \delta_3 + \delta_4 + \delta_5 - 3\ \delta'_4 \geq 0$ (5.1.1)

Node 5 (using T2.3) : $\delta_3 + \delta_4 + \delta_5 + \delta_6 + \delta_8 + \delta_9 + 3\,\delta'_4 \leq 6$ (5.1.2)

Node 6 (using T2.3) : $\delta'_6 \leq 1 - \delta_5$ (5.1.3)
(orT1.10)

$\delta'_6 \leq 1 - \delta_6$ (5.1.41)

$\delta'_6 \leq 1 - \delta_7$ (5.1.5)

Node 7 (using T2.4) : $\delta_7 + \delta_8 + \delta_9 - 2\ \delta'_7 \geq 0$ (5.1.6)

All terminal nodes are considered resolved.

STEP 5

Consider the intermediate node 3.

Associate the indicator variable δ'_3 and produce the following LCIF:

$$\delta'_3 = 1 \rightarrow Q_4 \text{ V } Q_5$$

or (using T1 .2) $\delta'_3 = 1 \rightarrow + \delta'_4 + \delta'_5 \geq 1$

or (using T2.4) $\delta'_4 + \delta'_5 - \delta'_3 \geq 0$ (5.1.7)

Consider node 2.

The tree representation corresponding to variable transformation T1.12 can be identified. Since nodes 3, 6 and 7 are resolved, the root node 1 is resolved by simply inserting the following integer linear constraint in CSTACK.

$$\delta'_3 + (1 - \delta'_6) - \delta'_7 \leq 1$$

or $\delta'_3 - \delta'_6 - \delta'_7 \leq 0$ (5.1.8)

STEP 6

All nodes are resolved. The complete IP representation is given by constraints (5.1.1) - (5.1.8) and

$$\delta_1, \ldots.\ \delta_9, \quad \delta'_{3,\ldots}\delta'_7 \in \{0,1$$

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 n × n 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 alphasbet ($j \in J$)

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.

STEP 1

Cl : "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 $(i,m')_($ $m'\neq m$, 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).

$J_1(j)$: set of letters which by virtue of the lexicon could appear in cells (i, m'), $m' \neq m$ 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).

STEP 2

Define atomic propositions

P_{imj} : "Cell (i- ^{m}) i^{s} occupied by letter j" $\forall$ i $\in$ I, m $\in$ M, j $\in$ J.

Rewrite C1 : "Exactly one of propositions $P_{imA} \dot{\vee} P_{imB} \dot{\vee} P_{imC} \ldots \dot{\vee} P_{imZ}$"

for all i $\in$ I, m $\in$ M.

Rewrite C2 : "If {P_{imj}- is TRUE] then

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

and

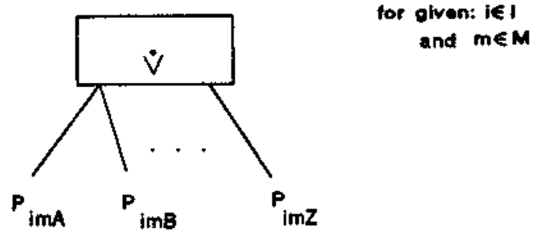
{at least (n-1) of $P_{i'mj''}$ are TRUE for i' $\neq$i, j" $\in J_2$ (j)})"

for all i $\in$ I, m $\in$ M, j $\in$ J.

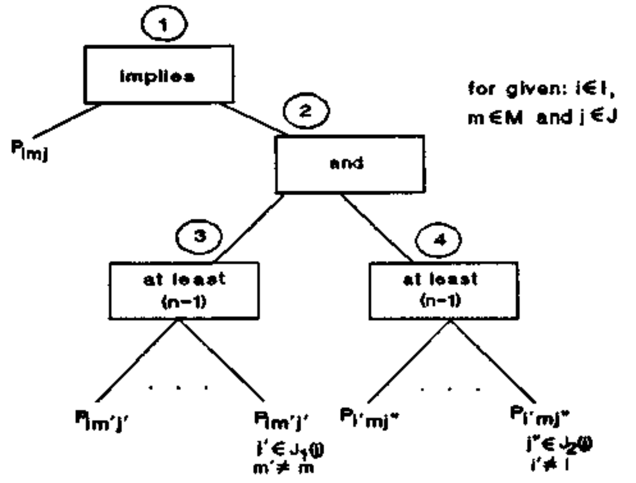
STEP 3

Tree representation of:

C1:



C2:



Define 0-1 decision variables $\delta_{i,m,j}$ $\forall$ i,m,j

$\delta_{i,m,j}$ = 1, letter j is placed in cell (i,m) that is $P_{i,m,j}$: is TRUE

= 0 otherwise.

STEP 4

C1 : Using T1.20, compound proposition Q_1 corresponding to node 1

(Q_1 : $P_{imA} \dot{\vee} P_{imB} \dot{\vee} \ldots \dot{\vee} P_{imZ}$) can be represented by

$$\sum_{j \in J} \delta_{imj} = 1 \quad \text{for given i,m} \qquad (5.2.1)$$

C2 : Assign 0-1 indicator variables to nodes 3 and 4.

Q_3 : "at least (n-1) of $P_{im'j'}$ ($m' \neq m$, $j' \in J_1(j)$)"

Q_4 : "at least (n-1) of $P_{i'mj''}$ ($i' \neq i$, $j' \in J_2(j)$)"

Apply variable transformation T1.19 to convert Q_3 and Q_4 into integer linear constraints.

$$Q_3: \sum_{\substack{j' \in J_1(j) \\ m' \neq m}} \delta_{im'j'} \geq n-1 \qquad \text{for given i, m, j}$$

$$Q_4: \sum_{\substack{j'' \in J_2(j) \\ i' \neq i}} \delta_{i'mj''} \geq n-1 \qquad \text{for given i, m, j.}$$

$$\delta'_3 = 1 \rightarrow \sum_{\substack{j' \in J_1(j) \\ m' \neq m}} \delta_{im'j'} \geq n-1 \quad : \text{Node 3}$$

$$\delta'_4 = 1 \rightarrow \sum_{\substack{j'' \in J_2(j) \\ i' \neq i}} \delta_{i'mj''} \geq n-1 : \text{Node 4}$$

Apply constraint transformation (Table 5) T2.4 to convert the above LCIF's into integer linear constraints:

$$\text{Node 3} \quad : \sum_{\substack{j' \in j_1 \\ n' \neq m}} \delta_{im'j'} \geq (n-1)\delta_3 \qquad (5.2.2)$$

$$\text{Node 4} \quad : \sum_{\substack{j'' \in J_2(j) \\ i' \neq i}} \delta_{i'mj''} \geq (n-1)\delta'_4 \qquad (5.2.3)$$

Terminal nodes 3 and 4 are resolved.

STEP 5

Consider node 2. The tree representation corresponding to variable transformation T1.10 can be identified, that is,

$P_{imj} \rightarrow Q_3 \wedge Q_4$ and the following linear constraints can be inserted in CSTACK.

$\delta_{imj} \leq \delta'_3$ (5.2.4)

$\delta_{imj} \leq \delta'_4$ (5.2.5)

Node 2 is resolved.

STEP 6

The complete IP representation for logical conditions Cl and C2 is given by constraints (5.2.1) - (5.2.5) $\forall\ i \in I, m \in M, j \in J$ and the integrality conditions.

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 [MITRAG87] on this topic. The importance of including logic forms within quantitative modelling paradigm is now well accepted. In the MODLER system of Greenberg [GREENB90] it is possible to combine logical and linear restrictions, and the structural modelling language (SML) of Geoffrion [GEOFFR90] includes prepositional calculus and predicate calculus modelling in levels two and three respectively. 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.

REFORMULATION

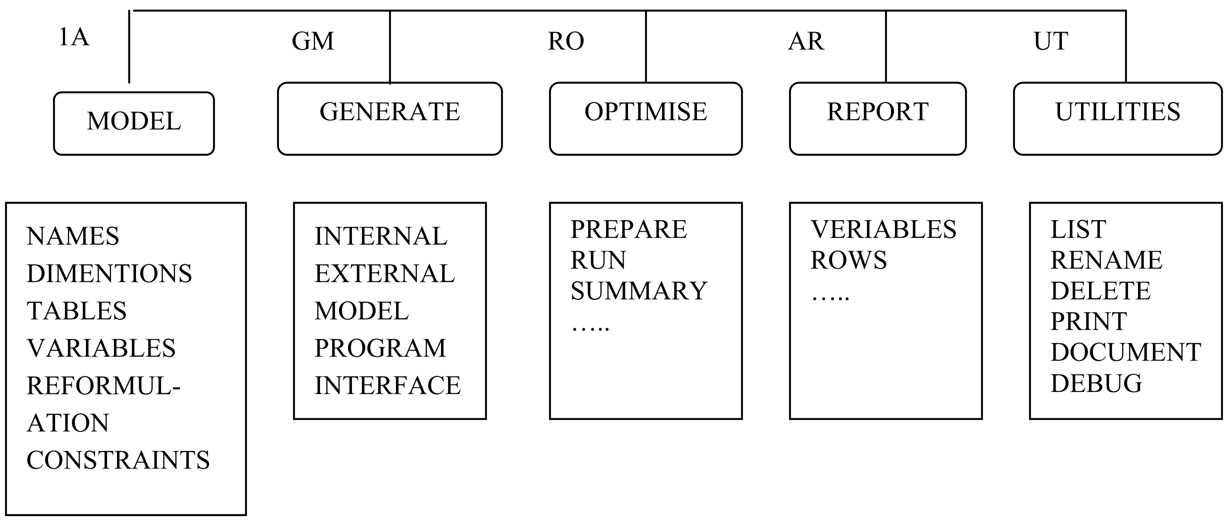
1. STATEMENT
2. DIMENSIONS
3. PROPOSITIONS
4. IP - VARIABLES
5. EQUIVALENT FORMS
6. TRANSFORMSIONS
7. BOUNDS
8. RETURN

Diagram 6.

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

1.	MODEL	1.	NAMES
2.	GENERATE	2.	DIMENSIONS
3.	OPTIMISE	3.	TABLES
4.	REPORT	4.	VARIABLES
5.	UTILITIES	5.	CONSTRAINTS
6.	LOGOUT	6.	REFORMULATION
		7.	RETURN

DIAGRAM 3 **DIAGRAM 5**



Hierarchical relationship of main menu options
and
information flow through the five aster files
as effected by the subsystem

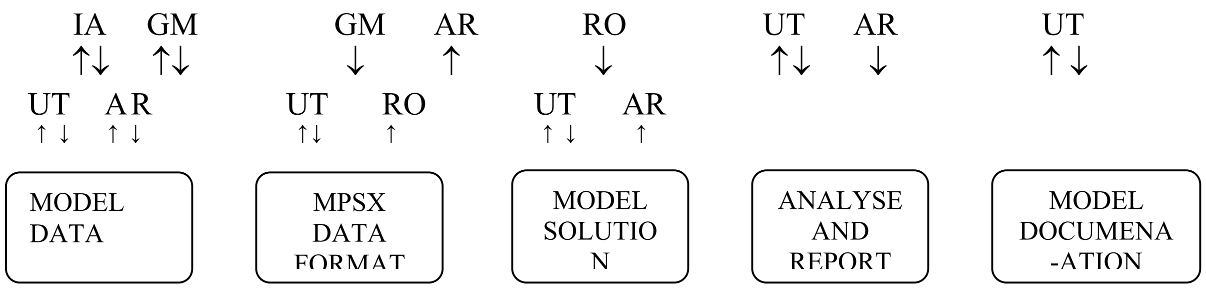


DIAGRAM 4

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, Pj, Q_j, 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, $\delta'_{k'}$ 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 programs", Monash University, Victoria 3168, Australia.

[BERGH87] 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.

[BLARJL88] Blair, C. E., Jeroslow, R. G., Lowe, J. K., 1988, "Some results and experiments in programming techniques for prepositional logic", **Computers and Operations Research, 13,** (5), 633-645.

[BRMTWL75] 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.**

[CHURCH44] Church, A., 1944, "Introduction to Mathematical Logic", **Annals of Mathematical Studies, 13,** Part 1.

[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.

[FOURER83] Fourer, R., 1983, "Modeling languages versus matrix generators for linear programming". **ACM Trans. Math. Soft ware 9,** 143-183.

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

[GEOFFR90] Geoffrion, A.M., 1990, "The SML language for strucutural modeling", Working Paper No.378, Western Management Science Institute, University of California, Los Angeles.

[GREENB90] Greenberg, H.J., 1990, "A primer for MODLER: Modeling by object-driven linear element relationships", Mathematics Department, University of Colorado at Denver.

[HENTRK89] Hentenryck, P. V., 1989, "Constraint satisfaction in logic Programming", MIT Press, Massachusetts, **USA.**

[HOOKER88a] Hooker, **J.** N., 1988, "A quantitative approach to logical inference", **Decision Support Systems, 4,** 45-69.

[HOOKER88b] Hooker, J. N., 1988,"Resolution versus cutting plane solution of inference problems: some computational experience", **Operations Research Letters 7, No.l,** 1-7.

[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 prepositional 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.

[LARJER78] 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.

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

[LUKSWZ63] Lukasiewicz, J., 1963, "Elements of Mathematics Logic" (English Translation from Polish), **Pergamon Press,** Oxford.

[MITRA87] Mitra, G., 1987, Guest editor, Mathematical Programming Modelling Systems, **IMA Journal of Mathematics in Management,** Oxford University Press, Special issues Vol.1, No. 3 and No.4.

[POST87] Post, S., 1987, "Reasoning with incomplete and uncertain knowledge as an integer linear program", Planning Research Corporation, Virginia 22102. Also in the Proceedings of **Expert Systems and Their Application,** Avignon, France.

[QUINEW55] Quine, W. V., 1955, "A way to simplify 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 machine oriented logic based 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, Groton CT, 06340.

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

[STEEL87] Steel,S.,1987,"Tutorialnotes",**AISB Tutorial Essex** University,UK.

[WBLLMS74] Williams, H.P., 1974, "Experiments in the formulation of integer programming problems", **Mathematical Programming Study,** Vol.2, pp. 180-197.

[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 prepositional calculus", **International Journal of Systems Resarch and Information Science,** 2, 81-100.

[WILLMS 89] Williams, H. P. and McKinnon, K.I.M., 1989, "Constructing integer programming models by the predicate calculus", **Annals of Operations Research,** Vol.2 1,p.227-246.

[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.