European Journal of Operational Research 72 (1994) 262-276
North-Holland

Tools for reformulating logical forms into zero–one mixed integer programs

G. Mitra, C. Lucas and S. Moody
Department of Mathematics and Statistics, Brunel University, Uxbridge, Middlesex, UB8 3PH, UK

E. Hadjiconstantinou
The Management School, Imperial College, London, UK
Received May 1992; revised November 1992

Abstract: A systematic procedure for transforming a set of logical statements or logical conditions imposed on a model into an Integer Linear Programming (ILP) formulation or a Mixed Integer Programming (MIP) formulation is presented. A reformulation procedure which uses the extended reverse Polish notation to represent a compound logical form is then described. The syntax of an LP modelling language is extended to incorporate statements in propositional logic forms with linear algebraic forms whereby 0–1 MIP models can be automatically formulated. A prototype user interface by which logical forms can be reformulated and the corresponding MIP constructed and analysed within an existing Mathematical Programming modelling system is illustrated. Finally, the steps to formulate a discrete optimisation model in this way are illustrated by means of an example.

Keywords: Mathematical Programming; Linear Programming modelling languages; Propositional calculus; First order logic; Logical Programming

1. Introduction

Computer based languages for constructing and analysing Mathematical Programming models have been investigated over the last two decades. There are many experimental and commercial systems currently available which provide modelling support. For an uptodate review of such systems the reader is referred to Steiger and Sharda (1991) and Greenberg (1991). Most modern LP modelling systems enable the modeller to specify models in a declarative algebraic language. A set of algebraic statements in a modelling language both specifies and documents a model, whereas the generation of a machine readable constraints matrix takes place in the background.

Although some modelling systems have been extended to incorporate non-linearities (Bisschop and Meerus, 1982) and to help with a greater variety of discrete optimisation problems (Bisschop and Fourer, 1992), very little attention has been given to the modelling of discrete programming extensions of LP problems. Many Mathematical Programming problems involve logical restrictions which may be expressed relatively easily using propositional calculus, but the reformulation of such statements into Mixed Integer Program (MIPs) is conceptually difficult. This reformulation may be carried out systemati-

Correspondence to: Prof. Dr. G. Mitra, Department of Mathematics and Statistics, Brunel University, Uxbridge, Middlesex UB8 3PH, UK.

SSDI 0377-2217(93)E0114-D

cally (Williams, 1987; Williams and McKinnon, 1989) but as yet there is no computer support for this task within a Mathematical Programming modelling system.

In order to put our work in context we briefly consider its relationship to other wider modelling or knowledge representation paradigms. In the field of management science, within the structured modelling system SML (Geoffrion, 1990), it is possible to represent problems of first order logic, namely propositional and predicate calculus. 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 (1989) and Brown et al. (1989). When the constraints (usually linear) involve expressions in real numbers, the simplex algorithm is applied to achieve constraint satisfaction; CLP(R), due to Lassez (1987) and PROLOG III due to Colmerauer (1987) are two such CPSs. For constraints stated in discrete integers (natural numbers) the tree search method or interval arithmetic is applied to achieve constraint satisfaction. Hentenryck (1989) CHIP and Brown and Chinneck (1989) BNR-PROLOG (1985) report two systems of this type. 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 (Buchanan and Shortliffe, 1984), constraint satisfaction and planning (Allen, 1983), mathematical puzzles and Combinatorial Programming (Lauriere, 1978) are typical examples which are well suited for solution through the application of computational logic (Hooker, 1988).

In this paper we present a reformulation procedure for transforming statements in propositional logic into integer or mixed integer programs; this procedure makes novel use of the Reverse Polish notation and the resulting expression tree. We define a new syntax involving logical propositions and operators whereby the structures of an LP modelling language is extended. This method is particularly suitable as a modelling technique which allows one to automate the reformulation process to construct equivalent IP or MIP models. The final goal is to integrate this modelling function into an 'intelligent' mathematical programming modelling modelling support system. The rest of the paper is organized in the following way. Section 2 contains a summary description of the important results in propositional logic and the corresponding 0–1 discrete programming equivalent forms. In this section the systematic reformulation procedure is stated as an algorithm. In Section 3, a prototype system which supports this modelling function is described and an example is set out to illustrate both the reformulation procedure and its realization within the modelling system.

2. Reformulation of propositional logic statements into 0-1 discrete programming problems

2.1. Logic forms represented by 0-1 variables

The main task of reformulation is 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 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 of inequalities. In Table 1 and Table 2 we summarize the propositional connectives and a set of well known transformations of logical statements into equivalent forms.

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. The former is considered in this sub-section and the latter is explained in Section 2.2.

Let P_j denote the j-th logical variable which takes values TRUE (T) or FALSE (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

Table 1
Propositional connectives

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 disjuntion)	$(P \dot{\vee} Q)$ $P \not\equiv Q$	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 an only if Q/P is a 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 nand Q	not both P, Q

only the values 1 and 0 (binary). The connection of these variables to the propositions are defined by the following relations:

$$\delta_j = \begin{cases} 1 & \text{iff proposition } P_j \text{ is true,} \\ 0 & \text{iff proposition } P_j \text{ is FALSE.} \end{cases}$$

Imposition 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 propositional connectives given in Table 1, and the equivalent statements, given in Table 2, a list of standard form variable transformations T1.1... T1.23 are defined (see Table 3). These transformations are applied to compound propositions involving one or more atomic propositions P_j, whereby the compound propositions are restated in linear algebraic forms involving decision variables.

2.2. *Bound analysis / logically relating linear form constraints*

In order to formulate 'logical constraints in the general form', it is well known that finite upper or lower bounds on the linear form must be used (Simonnard, 1966; Brearley, Mitra and Williams, 1975; Williams and McKinnon, 1989).

Table 2
Transformation of logical statements into equivalent forms

No.	Statement	Equivalent forms	
1	$\sim \sim P$	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 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 $(P \equiv Q)$	$(P \rightarrow Q) \wedge (Q \rightarrow P)$ $(\sim P \vee Q) \wedge (\sim Q \vee P)$	·
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	$P \wedge Q \rightarrow R$	$(P \rightarrow R) \vee (Q \rightarrow R)$	·
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 laws
12	$P \vee (Q \wedge R)$	$(P \vee Q) \wedge (P \vee R)$	

Table 3
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
$P_1 \dot{\vee} P_2$	$\delta_1 + \delta_2 = 1$	T1.3
$P_1 \wedge P_2$	$\delta_1 = 1, \delta_2 = 1$	T1.4
$\sim(P_1 \wedge P_2)$	$\delta_1 = 0, \delta_2 = 0$	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
$P_1 \vee (P_2 \wedge P_3)$	$\delta + \delta_2 \geq 1, \delta_1 + \delta_3 \geq 1$	T1.15
Some general forms of transformations are stated below:		
$P_1 \vee P_2 \vee \cdots P_n$	$\delta_1 + \delta_2 \cdots + \delta_n \geq 1$	T1.16
$P_1 \dot{\vee} P_2 \dot{\vee} \cdots P_n$	$\delta_1 + \delta_2 \cdots + \delta_n = 1$	T1.17
$P_1 \wedge \cdots P_k \rightarrow P_{k+1} \vee \cdots P_n$	$(1-\delta_1) \cdots + \delta_{k+1} + \cdots + \delta_n \geq 1$	T1.18
'at least k out of n are TRUE'	$\delta_1 + \delta_2 \cdots + \delta n \geq k$	T1.19
'exactly k out of n are TRUE'	$\delta_1 + \delta_2 \cdots + \delta_n = k$	T1.20
'at most k out of n are TRUE'	$\delta_1 + \delta_2 \cdots \delta n \leq k$	T1.21
$P_n \equiv P_1 \vee \cdots P_k$	$\delta_1 + \delta_2 \cdots + \delta_k \geq \delta_n$, $(-\delta_j + \delta_n \geq 0, j = 1, \ldots, k)$	T1.22
$P_n \equiv P_1 \wedge P_2 \wedge \cdots P_k$	$-\delta_1 - \delta_2 \cdots \delta_k + \delta_n \geq 1 - k$, $(\delta_j - \delta_n \geq 0, j = 1, \ldots, k)$	T1.23

Consider the linear form restriction

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

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

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

Finite bounds L_k and U_k are used in the reformulation procedure. These bounds may be given or, alternatively, can be computed for finite ranges of x_j (Brearley, Mitra and Williams, 1975). For example if $\ell_j \leq x_j \leq u_j$ $(j = 1, \ldots, n)$ then

$$L_k = \sum_{j \in P_k} a_{kj} \ell_j + \sum_{j \in N_k} a_{kj} u_j - b_k \text{ and } U_k = \sum_{j \in P_k} a_{kj} u_j + \sum_{j \in N_k} a_{kj} \ell_j - b_k$$

where $P_k = \{j : a_{kj} > 0\}$ and $N_k = \{j : a_{kj} < 0\}$.

A 'Logical Constraint in the Implication Form' (LCIF) is a logical combination of simple constraint and is defined as

if *antecedent* then *consequent*,

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

Table 4
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$(\delta'_k = 1 \rightarrow \sum_j a_{kj} x_j \geq b_k)$	

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 variables in linked to the antecedent. Whether the linear form constraint LF_k applies or otherwise is indicated by a 0–1 variable δ'_k:

$$\delta'_k = \begin{cases} 1 & \text{iff the } k\text{-th linear restriction applies} \\ 0 & \text{iff the } k\text{-th linear restriction does not apply} \end{cases}$$

A set of constraint transformations T2.1.... T2.5 are defined and set out in Table 4. For each of these transformations the binary variable, namely the indicator variable of the antecedent, using the bound value relates to the linear form restriction, that is the consequent.

2.3. 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 Lukasiewicz (1963) 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 later-operator form lead to two different notations and corresponds to inserting Church's brackets (1944) from the left or from the right respectively. The given expression can be written as

$$((P \vee Q) \vee (\sim R \wedge S)) \quad \text{or} \quad (P \vee (Q \vee (\sim R \wedge S))).$$

The tree representation for the first of these expressions is shown in Figure 1.

For the purposes of this paper, this is referred to as an 'expression tree'. Here the propositional calculus is limited to only unary and binary logical operators. The extended logical operators which involve n-tuples and n-place predicates (see next section where connectives such as 'exactly k out of n', 'at most k out of n' are introduced) can be used to construct 'extended expression trees'. Illustrations of extended expression trees are provided in the example discussed in Section 3.

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

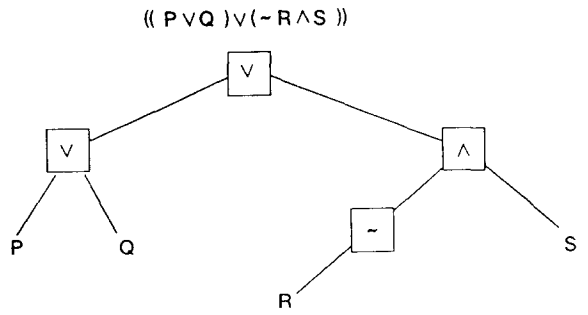

Figure 1

2.4. The algorithm

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 into the corresponding clauses. Each clause is then transformed into a linear constraint (applying transformation T1.16) so that the resulting conjunctive normal form can be represented by a system of constraints 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 2.3) 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 2) 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_j 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_j. 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_j or their negations as terminal nodes. Any subtree represents a compound proposition. Define 0–1 decision variables δ_j to represent the truth or falsity of each one of the simple propositions P_j, that is

$$\delta_j = 1/0 \quad \text{iff } P_j \text{ is true/false.}$$

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 δ'_k. Introduce δ'_k into the VSTACK. Convert the first-order compound propositions at the lowest

levels of the tree into associated linear restrictions using Table 3 of variable transformations. Introduce these into CSTACK. Apply the constraint transformations of Table 4 to convert the resulting LCIF $\delta'_k = 1 \rightarrow Q_k$ into an integer linear restrictions for this node. Pop-up the most recently placed constraint in CSTACK and then insert this new restriction into CSTACK. All terminal nodes are processed 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 variables δ'_k for any node k at an intermediate level in the tree, and update VSTACK. Produce an LCIF for this node involving δ'_k and the compound propositions $Q_{k_1}, \ldots, Q_{k_n}$ or their associated indicator variables $\delta'_{k_1}, \ldots, \delta'_{k_n}$, corresponding to the n branches of node k. Apply the variable and constraint transformations of Table 3 and 4, 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 3 is identified and all associated nodes are processed, do not introduce a new indicator variable for this node, but simply add the corresponding integer constraint, as obtained from Table 3, directly to CSTACK. The node is then considered processed.

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 VSTACK contains the decision and indicator variables used by these constraints.

3. Outline of a prototype system

3.1. An illustrative example

Consider the following problem:

In order to satisfy a country's energy demands, it is possible to import coal, gas and nuclear fuel from three neighbouring countries. There are three grades of coal and gas (low, medium and high) and one grade of nuclear fuel which may be imported.

The import costs for each fuel (in £s per gigajoule of energy obtained) are provided together with upper and lower limits on the fuel supplied by each country. The problem is to decide what quantities of each fuel should be imported from each country to that the total import cost is minimized and the country's energy requirements are met.

In addition, there are the following logical conditions which must also be satisfied:

(i) Each country can supply *either* up to three non-nuclear, low or medium grade fuels *or* nuclear fuel and one high grade fuel.

(ii) Environmental regulations require that nuclear fuel can be used *only if* medium and low grades of gas and coal are excluded.

(iii) If gas is imported then *either* the amount of gas energy imported must lie between 40–50% and the amount of coal energy must be between 20–30% of the total energy imported *or* the quantity of gas energy must lie between 50–60% and coal is not imported.

This problem, without the logical restrictions, may be expressed as follows:

Sets, indices

I = {low, med, high}: fuel grades.
J = {coal, gas}: (non-nuclear) fuel type.
K = {1, 2, 3}: countries.

Variables

$x_{ijk} \geq 0$ $(i \in I, j \in J, k \in K)$: Quantity of grade i, (non-nuclear) fuel energy j imported from country k (in gigajoules).

$y_k \geq 0$ $(k \in K)$: Quantity of nuclear energy imported from country k (in gigajoules).

Coefficients

c_{ijk} Unit cost of importing grade i, (non-nuclear) energy j (in gigajoules), from country k (in £s).
c_k^n Unit cost of importing nuclear energy (in gigajoules) from country k (in £s).
e Energy requirement of the importing country.
ℓ_{ijk} Lower limit on quantity of grade i, (non-nuclear) energy j, supplied by country k.
u_{ijk} Upper limit on quantity of grade i, (non-nuclear) energy j, supplied by country k.
ℓ_k^n Lower limit on nuclear energy supplied by country k.
u_k^n Upper limit on nuclear energy supplied by country k.

Linear constraints

$$\text{minimize cost} = \sum_{i\in I}\sum_{j\in J}\sum_{k\in K} c_{ijk}x_{ijk} + \sum_{k\in K} c_k^n y_k$$

$$\text{subject to} \sum_{i\in I}\sum_{j\in J}\sum_{k\in K} x_{ijk} + \sum_{k\in K} y_k = e. \qquad \text{(energy import requirements)}$$

The supply limits for each country may be modelled by introducing two binary variables δ_{ijk} and δ_k^n such that

$$\delta_{ijk} = \begin{cases} 1 & \text{iff } x_{ijk} > 0, \\ 0 & \text{iff } x_{ijk} = 0, \end{cases} \qquad \delta_k^n = \begin{cases} 1 & \text{iff } y_k > 0, \\ 0 & \text{iff } y_k = 0. \end{cases}$$

The supply limits may be thus stated as

$$\text{(non-nuclear)} \qquad \ell_{ijk}\delta_{ijk} \le x_{ijk} \le u_{ijk}\delta_{ijk} \quad \forall i \in I,\ j \in J,\ k \in K, \tag{3.1.0a}$$

$$\text{(nuclear)} \qquad \ell_k^n\delta_k^n \le y_k \le u_k^n\delta_k^n \quad \forall k \in K. \tag{3.1.0b}$$

As these are logical restrictions, it is also possible to deduce these conditions using the reformulation procedure.

The modelling of the first logical condition is now used to illustrate the reformulation procedure. For a reformulation of all three conditions see (Lucas, Mitra and Moody, 1992).

Each country can supply either up to three (non-nuclear) low or medium grade fuels or nuclear fuel and one high grade fuel.

Step 1. The above statement does not completely represent the modeller's requirement as some of the implied logical conditions are not explicitly set out. Therefore it is necessary to rewrite the statement explicitly to obtain a logically correct and complete representation of the modeller's requirement which is restated below. Let S_1 label this statement:

"Each country can supply *either* at most three of {low coal, medium coal, low gas, medium gas} and no nuclear or high grade fuel *or* nuclear fuel and exactly one of {high coal, high gas} and no medium or low grade fuel".

Step 2. Define the simple (atomic propositions

P_{ijk} ='grade i, (non-nuclear) fuel j is imported from country k'.
N_k ='nuclear fuel is imported from country k'.
where, $i \in$ {low, medium, high}; $j \in$ {coal, gas}; $k \in \{1, 2, 3\}$.

S_1 may be written as

"Either at most 3 of P_{ijk} are TRUE (where $i \in I$, $i \neq$ high; $j \in J$) and not $P_{\text{high } jk}$ $(\forall j \in J)$ and not N_k or N_k and exactly one of P_{ij} is TRUE (where $i \in I$, $i =$ high; $j \in J$) and not P_{ijk} (where $i \neq$ high; $j \in J$) for all $k \in K$".

Step 3. The tree representation of S_1 is shown in Figure 2. Define decision variables δ_{ijk} $(i \in I,\ j \in J,\ k \in K)$ and δ_k $(k \in K)$ such that $\delta_{ijk} = 1/0$ iff P_{ijk} is true/false and $\delta_k = 1/0$ iff N_k is true/false for given $k \in K$.

Step 4. Consider the terminal nodes 3, 4, 7 and 8. Let Q_3, Q_4, Q_7 and Q_8 label the following compound propositions (subtrees):

Q_3: at most 3 of $\{P_{\text{lowgask}}, P_{\text{medgask}}, P_{\text{lowcoalk}}, P_{\text{medcoalk}}\}$ for a given $k \in K$,
Q_4: nor $\{P_{\text{highcoalk}}, P_{\text{highgask}}\}$ for a given $k \in K$,
Q_7: exactly one of $\{P_{\text{highcoalk}}, P_{\text{highgask}}\}$ for a given $k \in K$,
Q_8: nor $\{P_{\text{lowcoalk}}, P_{\text{lowgask}}, P_{\text{medcoalk}}, P_{\text{medgask}}\}$ for a given $k \in K$.

Assign a binary indicator variable for each node; i.e.

$\delta'_2 = 1/0$ iff Q_2 is true/false,
$\delta'_4 = 1/0$ iff Q_4 is true/false,
$\delta'_7 = 1/0$ iff Q_7 is true/false,
$\delta'_8 = 1/0$ iff Q_8 is true/false.

Apply variable transformations (Table 3) to Q_3, Q_4, Q_7 and Q_8 to convert them into linear constraints:

$$Q_3: \quad \sum_{\substack{i \in I \\ i \neq \text{high}}} \sum_{j \in J} \delta_{ijk} \leq 3 \quad \text{using(T1.21)} \quad \text{for a given } k \in K,$$

$$Q_4: \quad \delta_{\text{high}jk} = 0 (\forall j \in J) \quad \text{using(T1.5)} \quad \text{for a given } k \in K,$$

$$Q_7: \quad \sum_{j \in J} \delta_{ijk} = 1 (i = \text{high}) \quad \text{using(T1.17)} \quad \text{for a given } k \in K,$$

$$Q_8: \quad \delta_{ijk} = 0 (\forall i \neq \text{high}, i \in I, j \in J) \quad \text{using(T1.5)} \quad \text{for a given } k \in K.$$

The terminal nodes of the tree can be resolved by imposing the logical constraints (LCIF's):

$$\text{Node 3:} \quad \delta'_3 = 1 \rightarrow \sum_{\substack{i \in I \\ i \neq \text{high}}} \sum_{j \in J} \delta_{ijk} \leq 3 \quad \text{for a given } k \in K,$$

$$\text{Node 4:} \quad \delta'_4 = 1 \rightarrow \left.\begin{aligned} &\delta_{\text{highcoalk}} = 0, \\ &\delta_{\text{highgask}} = 0, \\ &\delta_k = 0, \end{aligned}\right\} \quad \text{for a given } k \in K,$$

$$\text{Node 7:} \quad \delta'_7 = 1 \rightarrow \sum_{j \in J} \delta_{\text{high}jk} = 1 \quad \text{for a given k} \in \text{K},$$

$$\text{Node 8:} \quad \delta'_8 = 1 \rightarrow \delta_{ijk} = 0 \quad (\forall i \neq \text{high}, i \in I, j \in J) \quad \text{for a given } k \in K.$$

Apply constraint transformations (Table 4) to convert the resulting LCIF's into integer linear constraints.

$$\text{Node 3:} \quad \sum_{\substack{i \in I \\ i \neq \text{high}}} \sum_{j \in J} \delta_{ijk} + \delta'_3 \leq 4 \quad \text{using(T2.3)} \quad \text{for a given } k \in K, \tag{3.1.1}$$

$$\text{Node 4:} \quad \left.\begin{aligned} &\delta_{\text{high}jk} + \delta'_4 \leq 1 \quad \forall j \in J \\ &\delta_k + \delta'_4 \leq 1 \end{aligned}\right\} \quad \text{using(T2.5)} \quad \text{for a given } k \in K, \tag{3.1.2}$$

$$\text{Node 7:} \quad \left.\begin{aligned} &\sum_{j \in J} \delta_{\text{high}jk} + \delta'_7 \leq 2 \\ &\sum_{j \in J} \delta_{\text{high}jk} \geq \delta'_7 \end{aligned}\right\} \quad \text{using(T2.5)} \quad \text{for a given } k \in K, \tag{3.1.3}$$

$$\text{Node 8:} \quad \delta_{ijk} + \delta'_8 \leq 1 \quad (\forall i \neq \text{high}, i \in I, j \in J) \quad \text{using(T2.5)} \quad \text{for a given } k \in K. \tag{3.1.4}$$

All terminal nodes are now resolved.

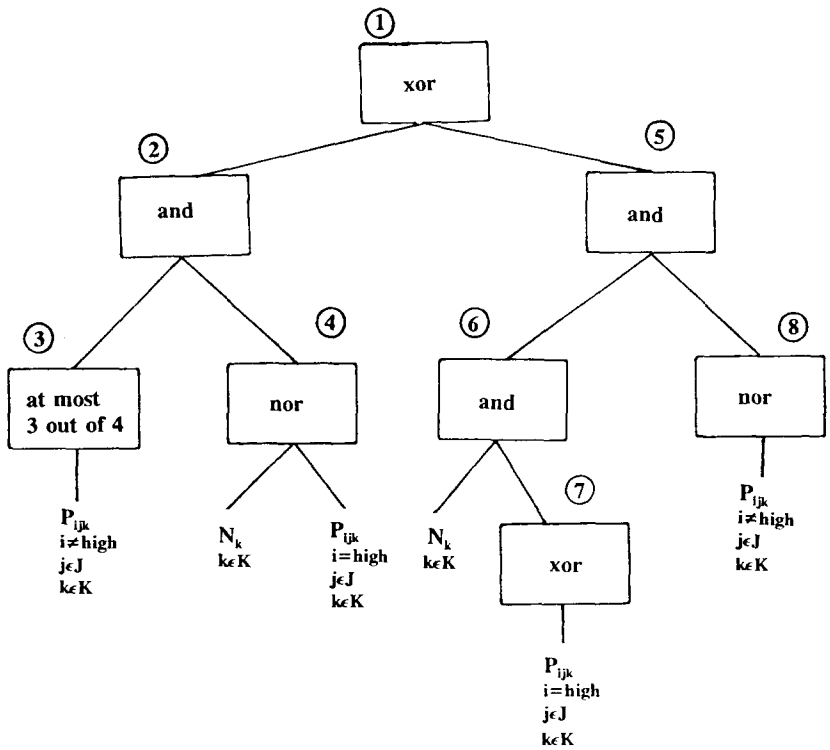

Figure 2

Step 5. Consider the intermediate nodes 2 and 6. Associate the indicator variables δ_2' and δ_6' and produce the following LCIFs:

$$\delta_2' = 1 \rightarrow \delta_3' = 1, \quad \delta_4' = 1 \quad \text{using(T1.4)} \quad \text{for a given } k \in K,$$

$$\delta_6' = 1 \rightarrow \delta_k = 1, \quad \delta_7' = 1 \quad \text{using(T1.4)} \quad \text{for a given } k \in K.$$

Applying constraint transformation (T2.5) gives:

$$\delta_3' \geq \delta_2', \tag{3.1.5}$$

$$\delta_4' \geq \delta_2', \tag{3.1.6}$$

$$\delta_k \geq \delta_6', \tag{3.1.7}$$

$$\delta_7' \geq \delta_6'. \tag{3.1.8}$$

Consider the intermediate node 5. Associate the indicator variable δ_5' and produce the following LCIF:

$$\delta_5' = 1 \rightarrow \delta_6' = 1, \quad \delta_8' = 1 \quad \text{using(T1.4)} \quad \text{for a given } k \in K.$$

Applying constraint transformation (T2.5) gives

$$\delta_6' \geq \delta_5', \tag{3.1.9}$$

$$\delta_8' \geq \delta_5'. \tag{3.1.10}$$

Consider node 1. The tree representation corresponding to variable transformation (T1.3) can be identified. Since nodes 2 and 5 are resolved, the root node 1 is resolved by simply adding the following integer linear constraint:

$$\delta_2' + \delta_5' = 1. \tag{3.1.11}$$

Step 6. All nodes are resolved. The complete IP representation is given by constraints (3.1.1)–(3.1.11) and $\delta_{ijk}, \delta_k, \delta'_2, \delta'_3, \delta'_4, \delta'_5, \delta'_6, \delta'_7, \delta'_8 \in \{0, 1\}$ $(i \in I, j \in J, k \in K)$.

3.2. An extension of the modelling language syntax

The syntax of algebraic modelling languages for formulating LPs is well established and understood (Ellison and Mitra, 1982; Fourer, Gay and Kernighan, 1987; Greenberg, 1990; Hürlimann, 1990; Maximal Software, 1991). In order to reformulate logical statements into 0–1 mixed integer programs, the syntax has to be extended such that propositional calculus statements can be analysed and corresponding restrictions generated. Then using the following metalinguistic notations:

(i) :: = is the meta language connective meaning 'is syntactically defined as'.
(ii) Optional components of syntax are placed in square brackets [...] while braces {...} indicate obligatory components of the syntax.
(iii) Alternatives may appear within braces or within square brackets and are written above each other.

A proposition is defined as

$$\langle\text{proposition}\rangle :: = \left[\begin{array}{l} \langle\text{simple proposition}\rangle \\ \langle\text{unary operator}\rangle\langle\text{proposition}\rangle \\ \langle\text{proposition}\rangle\langle\text{binary opeator}\rangle\langle\text{proposition}\rangle \\ \langle\text{binary prefix operator}\rangle\langle\text{proposition}\rangle\langle\text{proposition}\rangle \\ \langle\text{multiway operator}\rangle\langle\text{proposition}\rangle \end{array}\right].$$

The operators available set out in Table 5

The following example is used to illustrate the syntax. Consider the following propositions:

$$P_i : x_i \geq 0, \quad i = 1, \ldots, m,$$

$$Q_j : y_j = 1, \quad j = 1, \ldots, n,$$

$$R_j = Q_j \vee (P_1 \vee P_2 \vee \cdots \vee P_{m-1}), \quad j = 1, \ldots, n.$$

Having defined the simple propositions P_i and Q_j, the propositions R_j are defined as follows:

$$R(j) : Q(j)\text{OR}[\text{OR OVER } i : i \leq (m-1)P(i)].$$

Table 5

Operator		Type	Description
NOT		unary	Negation
IMPLIES		binary	Implication
IFF		binary	Equivalence
OR		binary	Inclusive disjunction
XOR		binary	Exclusive disjunction
AND		binary	Conjunction
NONE		binary prefix	Joint denial
NAND		binary prefix	Non-conjunction
ATMOST k		multiway	At most k out of n are TRUE
ATLEAST k		multiway	At least k out of n are TRUE
EXACTLY k		multiway	Exactly k out of n are TRUE
OR	OVER k	multiway	Inclusive disjunction over k propositions
XOR	OVER k	multiway	Exclusive disjunction over k propositions
AND	OVER k	multiway	Conjunction over k propositions
NONE	OVER k	multiway	Denial over k propositions
NAND	OVER k	multiway	Non-conjunction over k propositions

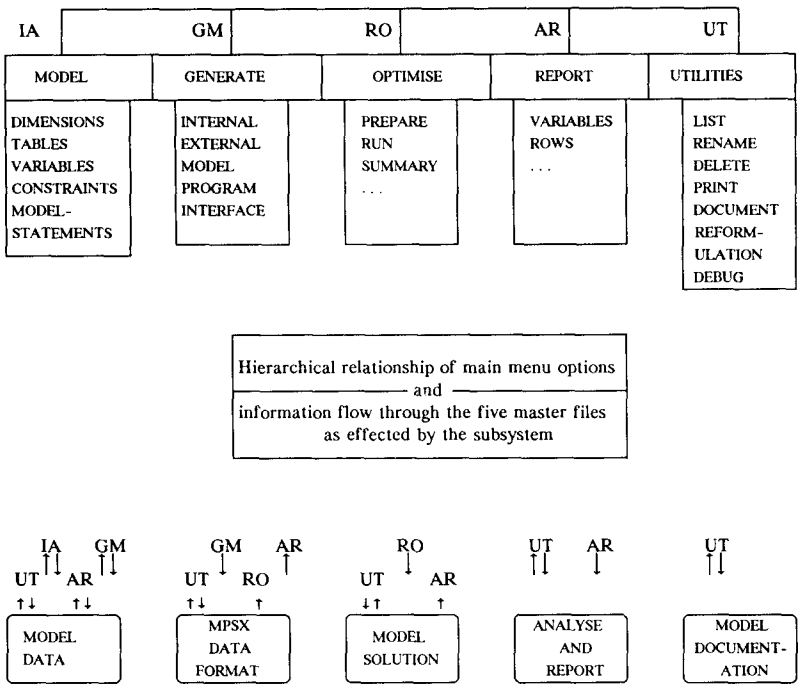

Figure 3

3.3. Realization within a modelling system

The following shows how reformulation support for logical statements is to be incorporated into the modelling system, CAMPS (Lucas and Mitra, 1988). The pull down menus, and the information flow diagram of CAMPS are set out in Figure 2. The primary revisions to CAMPS concerns the model specification, whereby a model is declared as the minimisation or maximisation of a function subject to certain propositions. A further utility is planned whereby the generated reformulations can be browsed. The DOCUMENT option of the UTILITIES menu enables the user to obtain an annotated statement of their model.

Figure 3 contains the documentation listing for the example of Section 3.1. Associated with each variable, bound and constraint is a unique proposition. These propositions are logically related in the MODEL STATEMENT section to specify the energy problem (see Figure 4).

Specification of the example energy model

Sets

i = {low, med, high} /*different quality grades*/
j = {gas, coal} /*non nuclear energy sources*/
k = {gb, fr, ic} /*countries exporting energy*/

Tables

ENERGYCST(i, j, k) /*nonnuclear energy cost, c_{ijk}*/
NUCLRCST(k) /*nuclear energy cost, c_k^n*/
MINNNTAK(i, j, k) /*minimum export amount of nonnuclear, ℓ_{ijk}*/
MAXNNTAK(i, j, k) /*maximum export amount of nonnuclear, u_{ijk}*/
MINNUTAK(k) /*minimum export amount nuclear, ℓ_k^n*/
MAXNUTAK(k) /*maximum export amount nuclear, u_k^n*/

MINREQE(j) /*minimum energy take if coal and gas*/
MAXREQE(j) /*maximum energy take if coal and gas*/
MINALT /*minimum gas take if gas energy only, scalars*/
MAXALT /*maximum gas take if gas energy only, scalars*/
ENREQ /*energy requirement of the importing country, e*/

Variables
/*Proposition name Variable name*/
IFNONNUC(j, j, k): AMNTENGY(i, j, k) /*P_{ijk} and x^*_{ijk}/
IFNUCLAR(k): AMNTNUCLR(k) /*N_k and y_k*/

Bounds
/*Proposition name Bound definition*/
LWNONNUC(i, j, k): AMNTENGY(i, j, k) >= MINNNTAK(i, j, k) /*Q_{ijk}*/
UPNONNUC(i, j, k): AMNTENGY(i, j, k) <= MAXNNTAK(i, j, k) /*R_{ijk}*/
LWNUCLAR(k): AMNTNUCLR(k) >= MINNUTAK(k)
UPNUCLAR(k): AMNTNUCLR(k) <= MAXNUTAK(k).

Rows
COST: SUM OVER i, j, k ENERGYCST(i, j, k)*AMNTENGY(i, j, k) +
SUM OVER k NUCLRCST(k)*AMNTNUCL(k)
/*definition of the objective function*/
ENGYBAL: SUM OVER i, j, k AMNTENG(i, j, k) +
SUM OVER k AMNTNUCL(k) = ENREQ
/*Energy balance row to meet country's energy requirement*/
MIXLOW1(j): SUM OVER i, k AMNTENGY(i, j, k) >= MINREQE(j)*ENREQ
/*lower bound requirement if both gas and coal imported*/
MIXHIGH1(j): SUM OVER i, k AMNTENGY(i, j, k) <= MAXREQE(j)*ENREQ
/*upper bound requirements if both gas and coal imported*/
MIXLOW2: SUM OVER i, k AMNTENGY(i, gas, k) >= MINALT*ENREQ
/*lower bound energy mix requirement if gas and not coal imported*/
MIXHIGH2: SUM OVER i, k AMNTENGY(i, gas, k) <= MAXALT*ENREQ
/*upper bound energy mix requirement if gas and not coal imported*/

MODEL STATEMENT

NAME	ENERGY
MINIMISE	COST
SUBJECT TO	
ENGYBAL	
SUPLCOND(k):	[[ATMOST 3 OVER i: $i \neq$ high, j IFNONNUC(i, j, k)]
	AND [[NONE OVER j, IFNONNUC(high, i, k)]
	AND NOT IFNUCLAR(k)]]
	XOR
	[[IFNUCLAR(k) AND [XOR OVER j IFNONNUC(high, j, k)]
	AND [NONE OVER i: $i \neq$ high, j, IFNONNUC(i, j, k)]]
ENVIRON:	[ATLEAST 1 OVER k IFNUCLAR(k)] IMPLIES
	[NONE OVER i: $i \neq$ high, j, k IFNONNUC(i, j, k)]
ALTMIX:	[ATLEAST 1 OVER i, k IFNONNNUC(, gas, k)]IMPLIES
	[[MIXLOW1(j) AND MIXHIGH1(j)]XOR
	[MIXLOW2 AND MIXHIGH2]]
BNDCOND1(i, j, k):	IFNONNUC(i, j, k) IMPLIES [LWNONNUC(i, j, k) AND UPNONNUC(i, j, k)]
BNDCOND2(k):	IFNUCLAR(k) IMPLIES [LWNUCLAR(k) AND UPNUCLAR(k)]

Figure 4

4. Concluding remarks

The zero–one mixed integer programming formulation of logical conditions presented as propositional calculus statements is an important research topic in discrete modelling. In this paper the established syntax of representing LP models in an algebraic form is extended to incorporate logical restrictions set out as propositional calculus statements. The methods described in this paper do not necessarily achieve the most computationally efficient model after reformulation. Indeed one referee pointed out that tighter reformulations can be found. This topic is discussed in Williams (1974). In this work our main aim has been to reduce the chore for an experienced analyst, and also to provide support for a problem owner who is capable of describing his problem but may not be experienced in reformulation techniques. A system constructed in this way not only provides discrete modelling support but can also be used as a teaching aid to new modellers in MIP reformulation techniques. The work reported in this paper also relates to the broader issues of knowledge representation within the logic programming and AI modelling paradigm.

Acknowledgements

One of the authors, Ms. S. Moody, was supported by SERC and the Numerical Algorithms Group (NAG) for her Ph.D. research work. In addition this work was sponsored by the US Army European Research Office. We are grateful to the referees for their comments.

References

Allen, J.F. (1983), "Maintaining knowledge about temporal intervals", *Communications of the ACM* 26, 832–843.

BNR-PROLOG (1988), Bell Northern Research, Prolog language description, Version 1.0, Ottawa, Canada.

Brown, R.G., Chinneck, J.W., and Karam, G.M. (1989), "Optimization with constraint programming systems", in: R. Sharda et al. (eds.) *Impact of Recent Computers Advances Operations Research*, North-Holland, Amsterdam 463–473.

Bisschop, J., and Fourer, R. (1992), "New constructs for the description of combinatorial optimization problem in algebraic modelling languages", to appear in: I. Maros and G. Mitra (eds.), *Annals of OR, Applied Mathematical Programming and Modelling*, Baltzer Press, Basel.

Bisschop, J., and Meeraus, A. (1982), "On the development of a general algebraic modelling system in a strategic planning environment", *Mathematical Programming Study* 20.

Bell Northern Research (1988) "Prolog language description", Version 1.0, Ottawa, Canada.

Brearley, A.L., Mitra, G., and Williams, H.P. (1975), "Analysis of mathematical programming problems prior to applying the simplex algorithm", *Mathematical Programming* 8, 54–83.

Buchanan, B.G., and Shortliffe, E.H. (1984), *Rule Based Expert Systems: The MYCIN Experiments of the Heuristic Programming Project*, Addison-Wesley, Reading, MA.

Church, A. (1944), "Introduction to mathematical logic", *Annals of Mathematical Studies* 13 Part 1.

Colmerauer, A. (1987), "Opening the PROLOG III Universe", *Byte Magazine*, August 1987.

Ellison, E.F.D., and Mitra, G. (1982), "UIMP: User interface for mathematical programming", *ACM Transactions on Mathematical Software* 8/3, 229–255.

Fourer, R., Gay, D.M. and Kernighan, B.W. (1987), "AMPL: A mathematical programming language", Computer Science Technical Report No. 133, AT&T Bell Laboratories.

Geoffrion, A.M. (1990), "The SML language for structured modeling", Working Paper No. 378, Western Management Science Institute, University of California, Los Angeles, CA.

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

Greenberg, H.J. (1991), "A comparison of mathematical programming modelling systems", Mathematics Department, University of Colorado at Denver.

Hentenryck, P.V. (1989), *Constraint Satisfaction in Logic Programming*, MIT Press, Cambridge, MA.

Hooker, J.N. (1988), "A quantitative approach to logical inference", *Decision Support Systems* 4, 45–69.

Hürlimann, T. (1990), "Reference manual for the LPL modeling language (Version 3.5)", Working Paper No. 175, Institute for Automation and Operations Research, University of Fribourg, Switzerland.

Lassez, C. (1987), "Constraint logic programming" *Byte Magazine*, August 1987, 171–176.

Lauriere, J.L. (1978), "A language and a program for stating and solving combinatorial problems", *Artificial Intelligence* 10, 29–127.

Lucas, C., and Mitra, G. (1988), "Computer-assisted mathematical programming (modelling) system: CAMPS", *The Computer Journal* 31/4, 364–376.

Lucas, C., Mitra, G., and Moody, S. (1992), "Tools for reformulating logical forms into zero–one mixed integer programs (MIPS)", Department of Mathematics and Statistics, Brunel University TR/O3/92.

Lukasiewicz, J. (1963), *Elements of Mathematics Logic* (English translation from Polish), Pergamon Press, Oxford.

Maximal Software (1991), "MPL Modelling System Release 2 User Manual", Maximal Software.

Simonnard, M. (1966), *Linear Programming*, Prentice-Hall, Englewood Cliffs, NY.

Steiger, D., and Sharda, R. (1991), "LP modelling languages for personal computers: A comparison", Oklahoma State University: also to appear in: I. Maros and G. Mitra (eds.), *Annals of Operations Research, Applied Mathematical Programming and Modelling*, Baltzer Press, Basel.

Williams, H.P. (1974), "Experiments in the formulation of integer programming problems" *Mathematical Programming Study* 2.

Williams, H.P. (1987), "Linear and integer programming applied to the propositional calculus", *International Journal of Systems Research and Information Science* 2, 81–100.

Williams, H.P., and McKinnon, K.I.M. (1989), "Constructing integer programming model by the predicate calculus", *Annals of Operations Research* 21, 227–246.