TR/03/92 April 1992

TOOLS FOR REFORMULATING LOGICAL FORMS INTO ZERO-ONE MIXED INTEGER PROGRAMS (MIPS)

by

C. Lucas, G. Mitra and S. Moody

CONTENTS

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 representation of a compound logical form is then described. 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 demonstrated by means of an example.

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 [STESHAR91], [GREENB91]. Most modern 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 constraint matrix takes place in the background.

Although some modelling systems have been extended to incorporate non-linearities [BISMEE82] and to help with a greater variety of discrete optimisation problems [BISFOR92], 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 propositions calculus, but the reformulation of such statements into Mixed Integer Programs (MIPs) is conceptually difficult. This reformulation may be carried out systematically [WILLMS87], [WILLMS89] 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 [GEOFFR90], 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 [HENTRK89] and Chinneck et al [BCHNKM89]. When the constraints (usually linear) involve expressions in real numbers, the simplex algorithm is applied to achieve constraint satisfaction; Lassez CLP(R) [LASSZC87] 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. 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 [BSHORT84], constraint satisfaction and planning [ALLENJ83], mathematical puzzles and Combinatorial Programming [LARIER78] are typical examples which are well suited for solution through the application of computational logic [HOOKER88].

The focus of this paper is to develop a systematic approach for transforming statements in propositional logic into integer or mixed integer programs. 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 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 Background

A simple (or atomic) proposition is a statement which can take only one of the truth values, true or false. Propositional calculus enables compound propositions to be formed by modifying a simple proposition with the word "not" or by connecting propositions with the words "and", "or", "if ... then" (or implies) and "if and only if". These five words are called propositional or logical **connectives** and they 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.

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 disjunction)	$(P \vee Q)$ $P \not\equiv Q$	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 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

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

For example, $P \rightarrow Q \equiv \sim P \vee Q)$ and $\sim\sim P \equiv P$, are equivalent expressions. The following laws of propositional logic are know as **De Morgan's Laws** and **Distributive Laws:**

De Morgan's Laws

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

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

Distributive Laws

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

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

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

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

Similarly De Morgan's laws followed by the second distributive law are applied to transform R to an equivalent proposition of the form $S_1 \vee S_2 \vee \ldots S_m$ in which each S_j j=l,....m is a conjunction of atomic propositions or their negation. In this case $S_1 \vee S_2 \vee \ldots S_m$ is called a **Disjunctive Normal Form (DNF).**

In both normal forms, negation is only applied to atomic propositions. All conjunctions may be removed leaving an expression entirely in "$\sim$" 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 2. 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.

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	$\sim(P \wedge Q)$	$\sim P \vee \sim P$	
5	$P \rightarrow Q$	$\sim P \vee Q$	Implication
6	$P \leftrightarrow Q$ or $(P \equiv Q)$	$(P \rightarrow Q) \wedge (Q \rightarrow P)$ $(\sim PVQ) \wedge (\sim QVP)$	.
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	$P \vee \rightarrow R$	$(P \rightarrow R) \wedge (Q \rightarrow R)$	.
11	$P \wedge (Q \vee R)$	$(P \wedge Q) \vee (P \wedge R)$	Distributive Laws
12	$P \vee (Q \wedge R)$	$(P \vee Q) \wedge (P \vee R)$	

2.2 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 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. The former is considered in this sub-section and the latter is explained in subsection 2.3.

Let P_j denote the jth logical variable which takes values T or F and represents an

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

$$\delta_j = 1 \text{ iff proposition } P_j \text{ is TRUE}$$
$$\delta_j = 0 \text{ iff proposition } P_j \text{ is FALSE}$$

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" Tl.l ... T1.23 are defined. 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. The two expressions are logically equivalent.

TABLE 3: VARIABLE TRANSFORMATIONS

Statement	Constraint	Transformation
$\sim P_1$	$\delta_1 = 0$	T 1 .l
$P_1 \vee P_2$	$\delta_1 + \delta_2 \geq 1$	T 1.2
$P_1 \dot{\vee} P_2$	$\delta_1 + \delta_2 = 1$	T 1.3
$P_1 \wedge P_2$	$\delta_1 = 1, \delta_2 = 1$	T 1.4
$\sim(P_1 \vee P_2)$	$\delta_1 = 0, \delta_2 = 0$	T 1.5
$\sim(P_1 \vee P_2)$	$\delta_1 + \delta_2 \leq 1$	T 1.6
$P_1 \rightarrow \sim P_2$	$\delta_1 + \delta_2 \leq 1$	T 1.7
$P_1 \rightarrow P_2$	$\delta_1 - \delta_2 \leq 0$	T 1.8
$P_1 \leftrightarrow P_2$	$\delta_1 - \delta_2 = 0$	T 1.9
$P_1 \rightarrow P_2 \wedge P_3$	$\delta_1 \leq \delta_2, \delta_1 \leq \delta_3$	T 1.10
$P_1 \rightarrow P_2 \vee P_3$	$\delta_1 \leq \delta_2 + \delta_3$	T l.11
$P_1 \wedge P_2 \rightarrow P_3$	$\delta_1 + \delta_2 - \delta_3 \leq 1$	T 1.12
$P_1 \vee P_2 \rightarrow P_3$	$\delta_1 \leq \delta_3, \delta_2 \leq \delta_3$	T 1.13
$P_1 \wedge (P_2 \vee P_3)$	$\delta_1 = 1\ \delta_2 + \delta_3 \geq 1$	T 1.14
$P_1 \vee (P_2 \wedge P_3)$	$\delta_1 + \delta_2 \geq 1, \delta_1 + \delta_3 \geq 1$	T 1.15

Some general forms of transformations are stated below:

$P_1 \vee P_2 \vee ... P_n$	$\delta_1 + \delta_2 \ldots + \delta_n \geq 1$	T 1.16
$P_1 \dot{\vee} P_2 \dot{\vee} \ldots P_1$	$\delta_1 + \delta_2 \ldots + \delta_n = 1$	T 1.17

$P_1 \wedge \ldots P_k \rightarrow P_{k+1} \vee \ldots P_n$	$(1-\delta_1) \ldots + \delta_{k+1} + \ldots + \delta_n \geq 1$	T 1.18
"at least k out of n are TRUE"	$\delta_1 + \delta_2 \ldots + \delta_n \geq k$	T 1.19
"exactly k out of n are TRUE"	$\delta_1 + \delta_2 \ldots + \delta_n = k$	T 1.20
"at most k out of n are TRUE"	$\delta_1 + \delta_2 \ldots + \delta_n \leq k$	T 1.21
$P_n \equiv p_1 \vee p_2 \vee \ldots p_k$	$\delta_1 + \delta_2 \ldots + \delta_k \geq \delta_n$, $(-\delta_j + \delta_n \geq 0,\ j = 1, \ldots, k)$	T 1.22
$P_n \equiv p_1 \wedge p_2 \wedge \ldots p_k$	$-\delta_1 - \delta_2 \ldots - \delta_k + \delta_n, \geq 1-k,$ $(\delta_j - \delta_n \geq 0,\ j = 1, \ldots, k)$	T 1.23

2.3 Bound Analysis/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, [SIMNRD66], [BRMTWL75], [WILLMS89].

Consider the linear form restriction

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

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 [BRMTWL75]. For example, if $\ell_j < x_j < u_j (j = 1,...,n)$ then

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

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

2.4 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 [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 1.

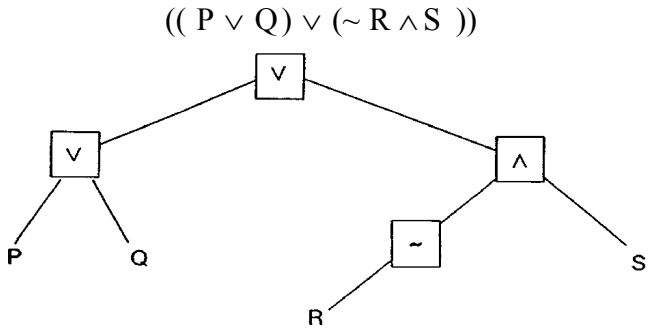


Diagram 1

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 and 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".

2.5 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, 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

formulate 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 2.4) 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 d: to represent the truth or falsity of each one of the simple propositions P_j, that is

$\delta_j = 1/0$ iff P_j 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 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 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 variable δ_k for any node k at intermediate to top levels in the tree, and update VSTACK.

Produce an LCIF for this node involving s: 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 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 the 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 so 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, medium, 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 import requirement.

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 supply by country k.

Linear Constraints

$$\text{Minimize cost} = \sum_{i \in I} \sum_{i \in J} \sum_{i \in K} c_{ijk} x_{ijk} + \sum_{k \in K} C_k^n Y_k$$

subject to:

$$\text{(energy import requirement)} \sum_{i \in I} \sum_{i \in J} \sum_{i \in K} x_{ijk} + \sum_{k \in K} Y_k = e \ .$$

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

$$\delta_{ijk} = \begin{cases} 1 \text{ iff } x_{ijk} > 0 \\ 0 \text{ iff } x_{ijk} = 0 \end{cases}$$

$$\delta_k^n = \begin{cases} 1 \text{ iff } Y_k > 0 \\ 0 \text{ iff } Y_k = 0 \end{cases}$$

The supply limits may be thus stated as:

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

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

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

The reformulation procedure is now used to illustrate the modelling of the three logical conditions set out in the example.

(i) 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

Let S_1 be the following statement:

"Each country can supply either at most three of {low coal, medium coal, low gas, medium gas} or (nuclear fuel and exactly one of {high coal, high gas})"

STEP 2

Define the simple (atomic) propositions

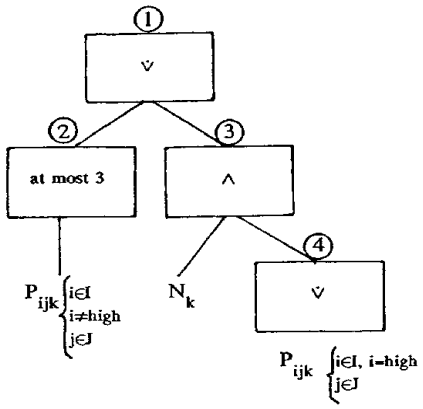
P_{ijk} = "grade i, fuel j is imported from country k"
N_k = "nuclear fuel is imported from country k"

S_1 may be rewritten ass

"Either at most 3 of P_{ijk} are TRUE (where $i \in I$, $i \neq$ high; $j \in J$) or N_k and exactly one of P_{ijk} is TRUE (where $i \in I$, $i =$ high; $j \in J$)"
for all $k \in K$.

STEP 3

The tree representation of S_1 is shown below. 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 2 and 4. Let Q_2 and Q_4 label the following compound propositions (subtrees):

Q_2 : "at most 3 of P_{ijk} $i \neq$ high, $i \in I$, $j \in J$ are TRUE" for a given $k \in K$
Q_4 : "exactly one of P_{ijk}. $i =$ high, $j \in J$ is TRUE" 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
and $\delta'_4 = 1/0$ iff Q_4 is true/false.

Apply variable transformations (Table 3) to Q_2 and Q_4 to convert them into linear constraints

Q_2 $\sum_{\substack{i\in I \\ i \neq \text{high}}} \sum_{j\in J} \delta_{ijk} \leq 3$ using (Tl .21) for a given $k \in K$

Q_4 $\sum_{j\in J} \delta_{ijk} = 1$ $(i = \text{high})$ using (T1.17) for a given $k \in K$

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

Node 2 : $\delta'_1 = 1 \rightarrow \sum_{\substack{i\in I \\ i \neq \text{high}}} \sum_{j\in J} \delta_{ijk} \leq 3$ for a given $k \in K$

Node 4 : $\delta'_4 = 1 \rightarrow \sum_{j\in J} \delta_{\text{high}jk} = 1$ for a given $k \in K$

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

$$\text{Node 2}: \quad \sum_{\substack{i\in I \\ i\neq high}} \sum_{j\in J} \delta'_2 + \delta'_2 \leq 4 \qquad \text{using (T2.3)} \qquad \text{for a given } k \in K \quad (3.1.1)$$

$$\left.\begin{array}{l} \text{Node 4}: \quad \displaystyle\sum_{j\in J} \delta_{highjk} + \delta'_4 \leq 2 \\ \\ \displaystyle\sum_{j\in J} \delta_{highjk} \geq \delta'_4 \end{array}\right\} \text{using (T2.5)} \qquad \text{for a given } k \in K \quad (3.1\ 2)$$

All terminal nodes are now resolved.

STEP 5

Consider the intermediate node 3. Associate the indicator variable δ'_3 and produce the following LCIF.

$$\delta'_3 = 1 \rightarrow N_k \wedge Q_4 \qquad \text{for a given } k \in K$$
$$\delta'_3 = 1 \rightarrow \delta_k = 1,\ \delta'_4 = 1 \qquad \text{using (T1.4)} \qquad \text{for a given } k \in K .$$

Applying constraint transformation (T2.5) gives:

$$\delta_k \leq 1 \qquad (3.1.3)$$

$$\delta_k \geq \delta'_3 \qquad (3.1.4)$$

$$\delta'_4 \leq 1 \qquad (3.1.5)$$

$$\delta_4 \leq \delta'_3 \qquad (3.1.6)$$

Consider node 1. The tree representation corresponding to variable transformation (T1.3) can be identified. Since nodes 2, 3 and 4 are resolved, the root node 1 is resolved by simply adding the following integer linear constraint

$$\delta'_2 + \delta'_3 = 1 . \qquad (3.1.7)$$

STEP 6

All nodes are resolved. The complete IP representation is given by constraints (3.1.1) - (3.1.7) and $\delta_{ijk}, \delta_k, \delta'_2, \delta'_3, \delta'_4 \in \{0,1\}$ ($i \in I,\ j \in J,\ k \in K$).

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

STEP 1

Let S_2 be the following statement

"If nuclear fuel is imported then none of the medium and low grades of gas and coal are imported"

STEP 2

Define the simple (atomic) propositions

N_k = "nuclear fuel is imported from country k"

P_{ijk} = "grade i, (non-nuclear) fuel j is imported from country k"

S_2 may be rewritten as.

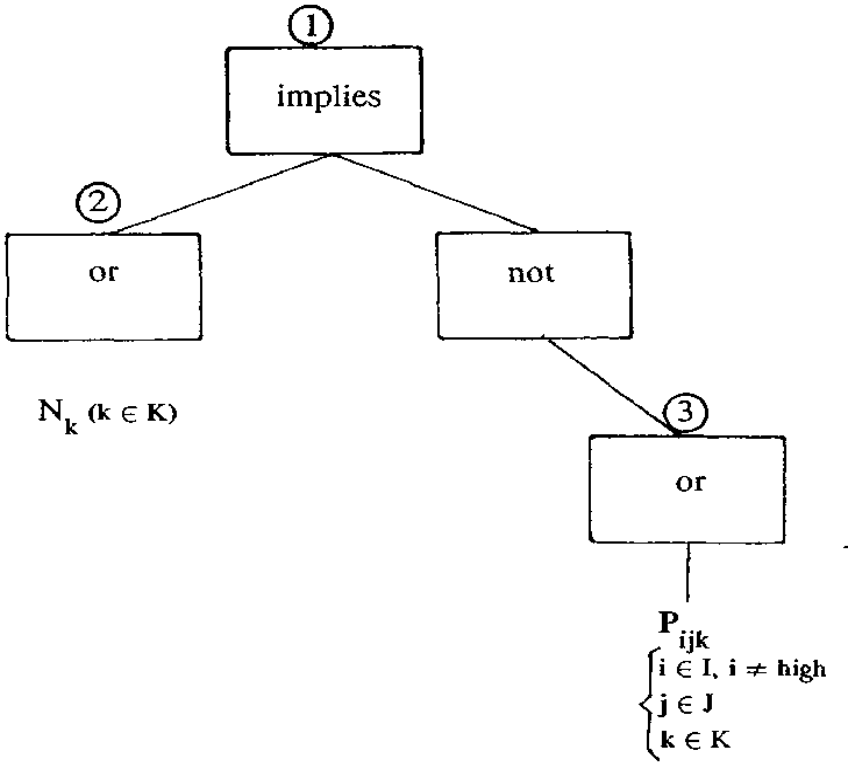
"If at least one of N_k ($k \in K$) is TRUE then none of P_{ijk} ($i \neq$ high, $i \in I$, $j \in J$, $k \in K$) is TRUE.

STEP 3

The tree representation of S_4 is shown below. Define 0-1 decision variables

(i) δ_k ($k \in K$) such that $\delta_k = 1/0$ iff N_k is true/false and

(ii) δ_{ijk} ($i \in I$, $j \in J$, $k \in K$) such that $\delta_{ijk} = 1/0$ iff P_{ijk} is true/false.



(3.1.8) - (3.1.10) and $\delta_{ijk}, \delta_k, \delta'_2, \delta'_3 \in \{0,1\}$ $(i \in I;\ j \in J,\ k \in K)$. $\{0,1\}$ $(i \in I; j \in J, k \in K)$.

(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.

STEP 1

Let S_3 be the following statement:

"If gas is imported then either the total imported energy consists of between 40 – 50% gas and between 20 - 30% coal or it consists of 50 - 60% gas and coal is not imported".

STEP 2

Define the simple (atomic) propositions

$P_{ijk.}$ = "grade i, fuel j is imported from country k" $(i \in I, j \in J, k \in K)$

$$Q_{lowj} = "e\ell_j \leq \sum_{i \in I} \sum_{k \in K} x_{ijk} \quad \text{where } \ell_{gas} = 0.4,\ \ell_{coal} = 0.2"$$

$$Q_{highj} = "\sum_{i \in I} \sum_{k \in K} x_{ijk} \leq eu_j \text{ where } u_{gas} = 0.5,\ u_{coal} = 0.3"$$

$$R_{lowgas} = "0.5e \leq \sum_{i \in I} \sum_{k \in K} x_{igask}"$$

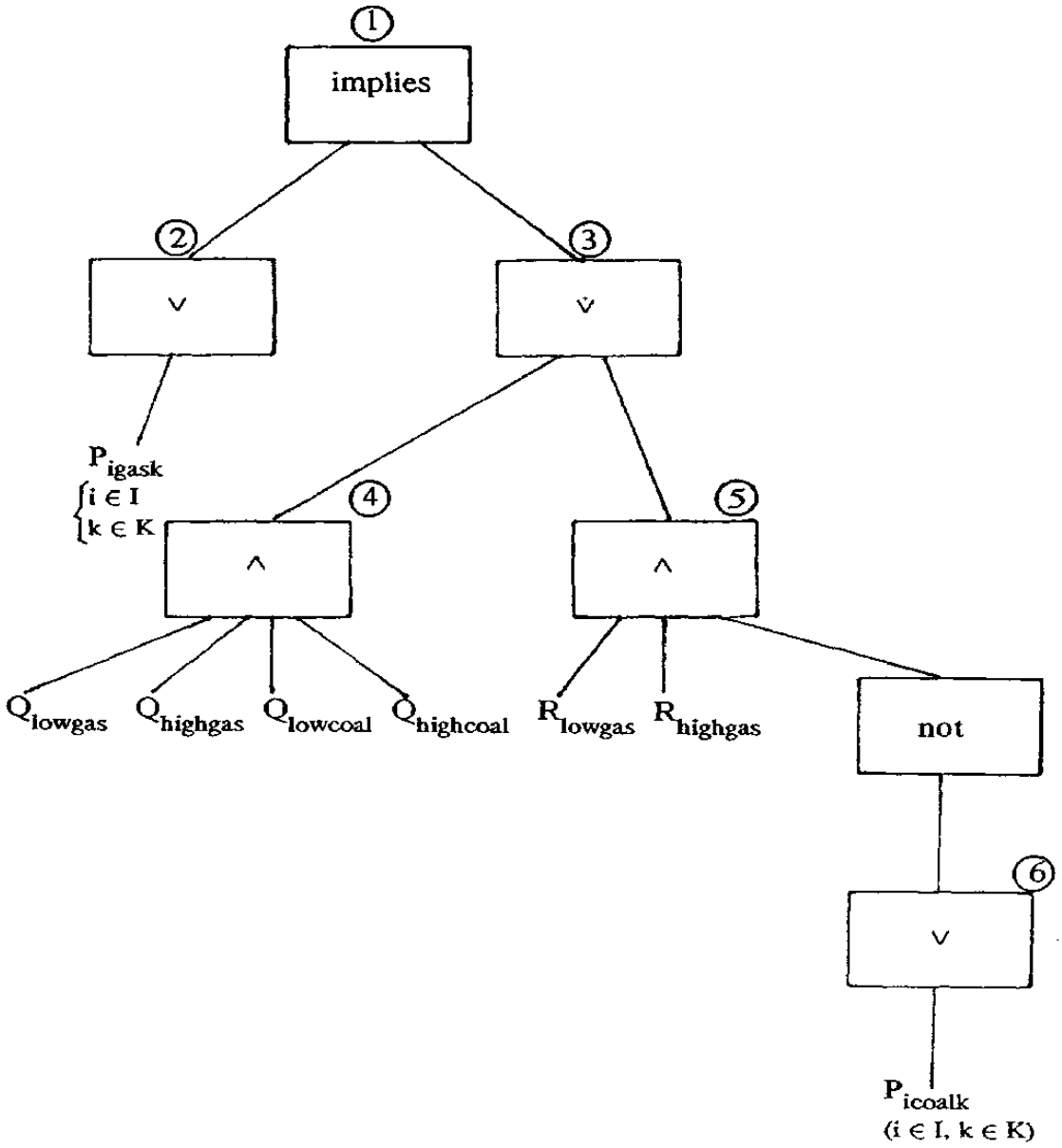
$$R_{highgas} = \sum_{i \in I} \sum_{k \in K} x_{igask} \leq 0.6e"$$

S_3 may be rewritten as

"If any of P_{igask} $(i \in I, k \in K)$ then either (Q_{lowj} and Q_{highj}) $(j \in J)$ or (R_{lowgas} and $R_{highgas}$ and p_{icoalk} $(I \in I, k \in K)$"

STEP 3

Tree representation of S_3 :



Define 0-1 decision variables $\delta^{p}_{ijk}, \delta^{Qlow}_{j}, \delta^{Qhigh}_{j}, \delta^{R}_{lowgas}, \delta^{R}_{highgas} \quad \forall \; i \in I, j \in J, k \in K$ such that

$\delta^{p}_{ijk} = 1/0$ iff P_{ijk} is true/false

$\delta^{Qlow}_{j} = 1/0$ iff Q_{lowj} is true/false

$\delta^{Qhigh}_{j} = 1/0$ iff Q_{highj} is true/false

$\delta^{R}_{lowgas} = 1/0$ iff R_{lowgas} istrue/false

$\delta^{R}_{highgas} = 1/0$ iff $R_{highgas}$ true/false

STEP 4

Consider terminal nodes 2, 4 and 6. Let Q_2 Q_4 and Q_6 be the first order compound propositions (subtrees):

Q_2 = "at least one of P_{igask} ($i \in$ l, $k \in K$)"

Q_4 = "all of Q_{lowj} and Q_{highj} ($j \in J$)"

Q_6 = "none of P_{icoalk} ($i, \in I, k \in K$)"

Assign corresponding 0-1 indicator variables

i.e. $\delta'_2 = 1/0$ iff Q_2 is true/false

$\delta'_4 = 1/0$ iff Q_4 is true/false

$\delta'_6 = 1/0$ iff Q_6 is true/false

Apply variable transformations (Table 3) to obtain the following LCIFs:

Node 2 : $\delta'_2 = 1 \; \square^{\cdot\cdot} \ddagger^{\prime\prime}_{i \square I} \; \ddagger^{\prime\prime}_{k \square K} \; \delta^{p}_{igask}$ ¡Ýl using (T1.16)

Node 4 : $\delta'_4 \; 1 \; \square^{\cdot\cdot} \delta^{Qlow}_{j} = 1, \delta^{Qhigh}_{j} = 1 \; \square$Íj $\square$Ţ using (T1.4)

Node 6 : $\delta'_6 = \square^{\cdot\cdot} \delta^{p}_{icoalk} = 0 \quad \square$Íi $\square$Ţ, k $\square$Ţ K using (T1.5)

Apply constraint transformations (Table 4) to convert these LCIFs into integer linear restrictions.

Node 2 :

$$\sum_{i \in I} \sum_{k \in K} \delta^{p}_{igask} \geq \delta_2 \qquad \text{using (T2.4)} \qquad (3.1.11)$$

Node 4 :

$$\delta^{Qlow}_{j} \geq 0 \qquad j \in J$$

$$\delta^{Qhigh}_{j} \geq 0 \qquad j \in J$$

$$\delta^{Qlow}_{j} \geq \delta_4 \qquad j \in J \qquad \text{using (T2.5)} \qquad (3.1.12)$$

$$\delta^{Qhigh}_{j} \geq \delta \qquad j \in J \qquad (3.1.13)$$

Node 6 :

$$\delta^{p}_{icoalk} \geq 0 - \delta_6 \qquad i \in I, k \in K \qquad \text{using (T2.5)} \qquad (3.1.14)$$

$$\delta^{p}_{icoalk} \geq 0 \qquad i \in I, k \in K$$

STEP 5

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

$$\delta_5 = 1 \text{ if } \delta^{R}_{lowgas} = 1 \text{ , } \delta^{R}_{highgas} = 1 \text{ , } \delta_6 = 1 \qquad \text{using (T1.4)}$$

Apply constraint transformations (T2.5) to obtain:

.

$$\delta^{R}_{lowgas} \geq \delta_5 \qquad (3.1.15)$$

$$\delta^{R}_{highgas} \geq \delta_5 \qquad (3.1.16)$$

$$\delta_6 \geq \delta_5 \qquad (3.1.17)$$

Consider intermediate node 3. Associate the indicator variable δ_3 and produce the following LCIF:

$$\delta_3 = 1 \text{ if } \delta_4 = 1 \text{ or } \delta_5 = 1 \qquad \text{using (T1.3).}$$

Using constraint transformation T2.5 to obtain the integer linear restrictions:

$$\delta_3 \leq 2 - \delta_4 - \delta_5 \qquad (3.1.18)$$

$$\delta_3 \geq \delta_4 \qquad \delta_3 \geq \delta_5 \qquad (3.1.19)$$

Consider node 1, the tree representation of T1.8 is identified, hence the integer linear constraint

$$\delta'_2 \sim \delta'_3 \; ¡Ü 0 \qquad (3.1.20)$$

STEP 6

All nodes are resolved.

The complete IP representation is given by constraints (3.1.11) - (3.1.20) and δ^{P}_{ijk}, δ^{Qlow}_{j}, δ^{Qhigh}_{j}, δ^{R}_{lowgas}, $\delta^{R}_{highgas}$, $\delta_2 \ldots \delta_6 \in \{0,1\}$ ($i \in I$, $j \in J$, $k \in K$).

3.2 An Extension of the Modelling Language Syntax

The syntax of algebraic modelling languages for formulating LPs is well established and understood [ELSMIT82], [FOGAKER87], [GREENB901], [HURLIM90], [MAXIMAL91]. 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

$$\text{< proposition > ::=} \left\{ \begin{array}{l} \text{< simple proposition >} \\ \text{< unary operator > < proposition >} \\ \text{< proposition > < binary operator > < proposition >} \\ \text{< binary prefix operator > < proposition > < proposition >} \\ \text{< multiway operator > < proposition >} \end{array} \right\}$$

The operators available are set out below

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

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

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

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

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

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

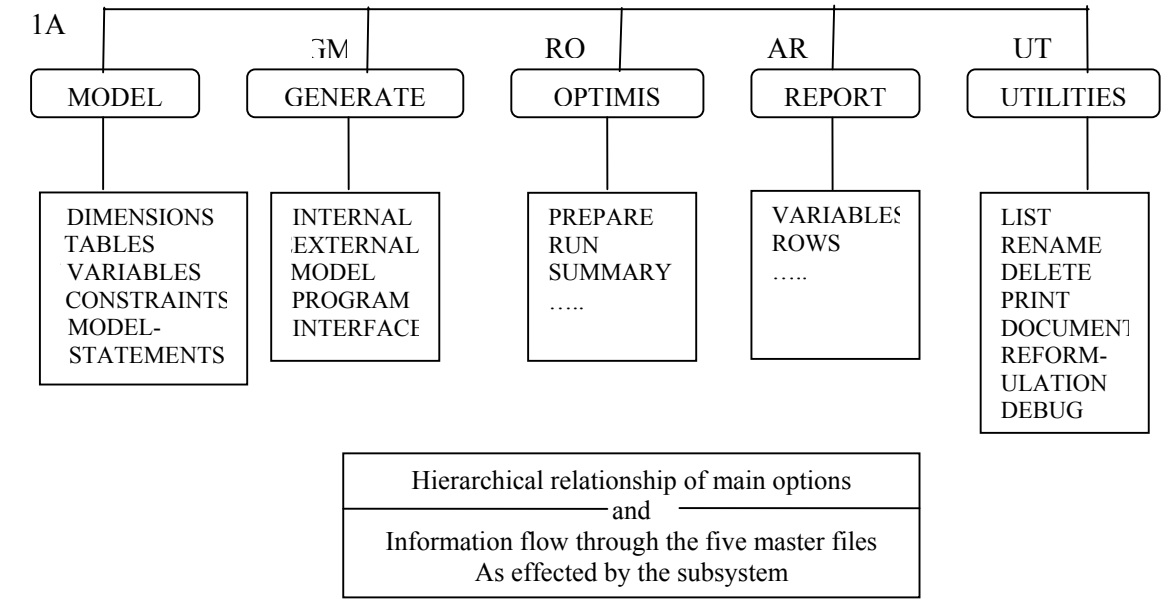
R(j) : Q(j) OR [OR OVER i : i < m P(i)] .

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, [LUCMIT88]. The modelling menu (MODEL), and the information flow diagram of CAMPS is set out in Diagrams 2 and 3. 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.

1. DIMENSIONS
2. TABLES
3. VARIABLES
4. CONSTRAINTS
5. MODEL STATEMENTS
6. RETURN

DIAGRAM 2



Hierarchical relationship of main options
and
Information flow through the five master files
As effected by the subsystem

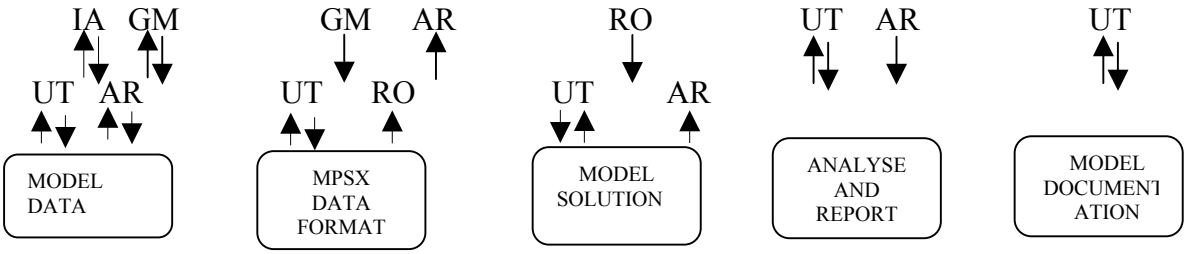


DIAGRAM 3

Diagram 4 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.

SPECIFICATION OF THE EXAMPLE ENERGY MODEL

SETS

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

TABLES

ENRGYCST(i,j,k) /*non nuclear energy cost, c_{ijk}*/
NUCLRCST(k) /*nuclear energy cost, c_k^n */
MINNNTAK(i,J,k) /*minimum export amount of non nuclear, ℓ_{ijk} */
MAXNNTAK(i,j,k) /*maximum export amount of non nuclear, u_{ijk}*/
MINNUTAK(k) /*minimum export amount nuclear, ℓ_k^n */
MAXNUTAK(h) /*maximum export amount nuclear, u_k^n */
MINREQE /*minimum energy take if coal and gas, scalars*/
MAXREQE /*maximum energy take if coal and gas, scalars*/
MINALT /*minimurn gas take if gas energy only, scalars*/
MAXALT /*maximum gas take if gas energy only, scalars*/
ENREQ /*desired energy import amount, e*/

VARIABLES

/*Proposition name Variable name*/
IFNONNUC(i,j,k) : AMNTENGY(i,j,k)/*P_{i1k} 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 ENRGYCST(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 AMNTENGY(i,j,k) +
SUM OVER k AMNTNUCL(k) = ENREQ
/*Energy balance row to meet countries energy requirement*/

MIXLOWl(j) : SUM OVER i,j AMNTENGY(i,j.k) >= MINREQE*ENREQ
/*lower bound requirements if both gas and coal imported*/
MIXHIGHl(j) : SUM OVER i,k AMNTENGY(i,j,k) <= MAXREQE*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 ≠ high, j IFNONNUC(i,j,k)]
XOR
[IFNUCLAR(k) AND [XOR OVER j IFNONNUC(high,j,k)]
ENVIRON : [ATLEAST 1 OVER k IFNUCLAR(k)] IMPLIES
[NONE OVER i : i ≠ high, j, k IFNONNUC(i,j.k)]
ALTMIX : [ATLEAST 1 OVER i, k IFNONNUC(i,gas,k)] IMPLIES
[[MIXLOWIO) AND MIXHIGHl(j)] XOR
[MIXLOW2 AND MIXHIGH2]]
BNDCOND l (ij,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)]

DIAGRAM 4

4. CONCLUDING REMARKS

The quantitative formulation of logical conditions set out 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. 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.

5. 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.

[BISFOR92] Bisschop, J. and Fourer, R., 1992, "New Constructs for the Description of Combinatorial Optimization Problems in Algebraic Modelling Languages" to appear in Annals of OR Publication

[BISMEE82] Bisschop, J. and Meeraus, A., 1982, "On the Development of a General Algebraic Modelling System in a Strategic Planning Environment", Mathematical Programming Study 20, North-Holland, Amsterdam.

[BNPRLG88] Bell Northern Research, 1988, Prolog Language Description, Version 1.0, Ottawa, Canada.

[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.

[ELSMTT82] Ellison, E.F.D., Mitra, G., 1982, ″UIMP: User Interface for Mathematical Programming, ACM Transactions on Mathematical Software, Vol.8, No.3, p.229-255.

[FOGAKER87] Fourer, R., Gay, D.M., Kernighan, B.W., 1987, ″AMPL: A Mathematical Programming Language″, Computing Science Technical Report No. 133, AT&T Bell Laboratories.

[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.

[GREENB91] Greenberg, Harvey, J., 1991, ″A Comparison of Mathematical Programming Modelling Systems″, University of Colorado at Denvery.

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

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

[HURLIM90] Hürlimann, T., 1990, ″Reference Manual for the LPL Modeling Language″ (Version 3.5), Working Paper No. 175, Institute for Automation & Operations Research.

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

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

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

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

[MAXIMAL91] Maximal Software, 1991, MPL Modelling System Release 2 User Manual, Maximal Software.

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

[STESHAR91] Steiger, D. and Sharda, R., 1991, ″LP Modelling Languages for Personal Computers: A Comparison″, Oklahoma State University.

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

[WILLMS 89] Williams, H. P. and McKinnon, K.I.M., 1989, ″Constructing integer programming models by the predicate calculus″, Annals of Operations Research, Vol.21, p.227-246.