Logical and Integer Modeling

Tony Hürlimann
info@matmod.ch

November 30, 2022

Abstract

Logical and Boolean conditions are often used in real-live problem solving. However, its importance in practical modeling contrasts sharply with its treatment in modeling books and courses. Many modeling techniques and applications to formulate logical conditions are exposed in this paper.

It presents various methods of logical modeling, that is, mathematical modeling containing logical propositions and Boolean mixed with mathematical expressions within the constraints. Several modeling techniques are formulated in mathematical notation and in the modeling system LPL (see [18]). Concrete and executable model application examples are given for the different techniques.

A word of caution: It is not trivial in a practical context to use Boolean operators. One needs to analyze carefully the context to use the right operators. "and", "or", and other words in a a spoken text cannot be translated one-to-one to Boolean "and", "or", for instance. Check multiple times before proceeding!

A second word of caution: The modeling formulation may be correct, the implementation of the translation to a mathematical linear model might not always correspond to this logical modeling. This is due to the fact that in most cases, the automatically introduction of a new binary variable is realized in LPL as an implication instead of an equivalence. In most cases, this is justifies but not always. Hence, the resulting linear model in LPL should be checked anyway.

A ZIP file of all models can be found HERE

Contents

1 Introduction

Mathematical modeling using integer variables is a surprisingly powerful way to formulate and handle real problems. This is not obvious at a first glance. Clearly, if there are objects that cannot be divided into parts, then integer quantities are needed. The number of aeroplanes, for example, must be integer, 1.5 aeroplanes does not make sense in most context. On the other hand, if there are 3'345'678 or 3'345'678.5 hens in a country, the difference may not be important probably, one just can round the number up or down. More important from a modeling point of view, however, are yes/no decisions where integer programming is used, which is the main focus of this paper.

Of course, we know that models containing integer variables are much harder to solve. However, this paper focuses on the various operations which allow the modeler to express the model in a straightforward way and not on how to solve it basically. The focus is on modeling and formulation and less on how to solve these models.

Integer programming can be applied in various contexts:

1. Mathematical problem with logical conditions. This happens often in a mathematical model that must also satisfy some logical conditions. Examples are:
   (a) "If a depot is located at A, then the customer X must be delivered from it."
   (b) "If the journal is too expensive then we must make sure that at least two other journals in the same categories are ordered at a lower price."
   (c) "If product A is manufactured then at least one of product B and C must also be manufactured."
   (d) "If the factory A is closed, then at least one other factory in the region must be kept open."
   (e) "The number of items in a portfolio must not be larger than 5."
   (f) "Two operations A and B cannot be executed at the same time on the machine, either operation A is executed before operation B, or vice versa."
   (g) "Exactly 5 ingredients are allowed in a blend of this type of product."
2. A large category are combinatorial problems. These problems arise when allocating items or persons to different tasks, or carrying out operations in particular order. Examples are:
   (a) Sequencing problems: The sequence of operations must fulfill certain conditions. A particular problem is the *job-shop scheduling* where operations on a machine have to be executed in a defined order. Another particular problem is the *traveling salesman problem*, where the locations must be visited in a sequence that fulfills certain criteria.
   (b) Allocation problems: These problems arise when units or persons have to be allocated to certain tasks, divisions, services or locations, such as the *the depot location problems.*
3. Integer Programming can also be used for modeling non-linear problems. Typical examples are *fixed charge problems*, this occurs when some activity involves a threshold set-up cost, for example. Sometimes it makes sense to remodel a non-linear model as an linear model using additional integer variables. Examples are *piece-wise linear functions* that approximate an otherwise non-linear function.

4. A large category are network problems or problems that arise from graph theory. Some problems are easy problems, such as the *the transportation problem*, *the assignment problem*, *critical path problems*, and other *flow problems*. They all contains a special structure and can be solved without explicitly fix the variables to be integer. Other problems (most of them) are difficult problems, such as *graph vertex coloring*, *quadratic assignment problem*, finding a *maximal clique* in a graph and others.

5. Finally, we have problems were one must impose integer values, as already mentioned, such as number of aeroplanes, etc. This is the most evident kind of problems where integer quantities are used, they are also the least important problems from a practical point of view.

Using mathematical notation only, one can formulate all these variants, although one also can use Boolean propositions and predicate logic to express some of these problems. Whether the problem is formulated in a purely mathematical notational framework or partially in logic is basically the same, because one can translate logical expressions into mathematical once, and vice versa. We will see how this can be done in one direction: translating logical expressions into mathematical linear constraints. Advantages of translating logical into mathematical constraints are as follows:

1. Powerful IP- and MIP-solvers can be applied to solve complex problems or at least to find good lower and upper bound for a solution.

2. In the case where the Boolean constraints can be expressed as Horn clauses (Prolog clauses are all Horn-clauses), the model can be solved using LP without branching. Instead of using inference and resolution techniques to solve such problems, one may translate the problem into a LP problem and solve the transformed problem efficiently.

3. Logical and mathematical constraints can be mixed in a single defined modeling language to augment the readability of a model and to produce more compact models.

4. As mentioned, MIP-modeling is extremely rich, but also often tricky and not at all trivial. Often a more natural formulation and representation for many problems is (a subset of) predicate logic.

This paper contains two parts. The first part lists and explains all mathematical and logical operators that can be used in constraints. Model examples are given to understand the semantic of the different operations. The second part presents a translation procedure that translates all logical operators into mathematical linear constraints. This is also basically the procedure that is implemented in the LPL interpreter software. Model examples in LPL are also given.

2 The General Model and its Operators

A optimization problem can be formulated as follows:

$$\begin{array}{ll} \min & f(\mathbf{x}) \\ \text{subject to} & g_i(\mathbf{x}) \leq 0 \quad \text{forall } i \in \{1,\ldots,m\} \\ & \mathbf{x} \in \mathbf{X} \end{array}$$

where f, g_i are functions defined in $\mathbb{R}^n$ (or $\mathbb{N}^n$), $\mathbf{X}$ is a subset of $\mathbb{R}^n$ (or $\mathbb{N}^n$), and $\mathbf{x}$ is a vector of n components $x_1,\ldots,x_n$ (see [19]). The notation formalism is given in [17].

Commonly, the functions f, g_i are mathematical functions that consist of the usual operands and operators: *numbers*, *variables*, *parameters* and the operators for *addition*, *subtraction*, etc., and also function like *sin*, *log* etc. $f(\mathbf{x})$ is a function that results in a numerical value. The constraints, however, are Boolean expressions: $g_i(\mathbf{x}) \leq 0$ that are true or false. Of course, $g_i(\mathbf{x})$ also results in a numerical value which is smaller or equal zero (which means that the constraint holds (is true), otherwise it does not hold (it is false).

Let us now extend the syntax of the general optimization model: besides the usual operands (numbers, parameters, variables) and numerical operators in a constraint, *Boolean propositions*, and *predicates* as well as Boolean operators can be used. This allows one to mix logical and mathematical notation in one single expressions. In order to make sense for all such expressions, we introduce the convention that the two basic value in logic (*false* and *true*) are mapped to the two numerical values: *zero* and *not-zero* (respectively *one*)[1], this is also common practice in some programming languages. Hence, we adopt the convention for a Boolean proposition x that "$x = 0$" means "x is false" and in all other cases "x is true" (in particular when "$x = 1$"). The generalization of a mathematical-logical model is then as follows:

$$\begin{array}{ll} \min & F(\mathbf{x}) \\ \text{subject to} & G_i(\mathbf{x}) \quad \text{for } i \in \{1,\ldots,m\} \\ & \mathbf{x} \in \mathbb{N}^n (\text{or } \mathbb{R}^n) \end{array}$$

where $F(\mathbf{x})$ and $G_i(\mathbf{x})$ are expressions containing mathematical *and* logical operators. This is a powerful way to compactly specify a large variations of constraints in a uniformed syntax. Before using them in concrete model, these operators and their precedence within an expression is now defined. All the operators and operands that are allowed in a constraint are listed in Table 1, x and y are arbitrary (sub)-expressions. The precedence is in decreasing order within the table. Indexed operators are listed in Table 2. The indexed operators all have the same precedence.

Three examples that corresponds to a syntactical correct expression are given:

1. "If x and y are true or z is true then and only then w is false". This is a purely Boolean expression formulated as:

$$x \wedge y \vee z \dot{\vee} w$$

2. "If a is smaller or equal than b or if $3b$ is the same as $4c$ then x or y or both must be true". This is a mixed expression containing mathematical and logical operators formulated as:

$$(a \leq b \ \vee \ 3b = 4c) \rightarrow (x \vee y)$$

[1] When using integer numbers there is no ambiguity in the concept of "non-zero", however if floating point numbers are involved, there must carefully be defined what is meant by "non-zero": how far away from zero must a number be to be defined as non-zero? A (small) interval must be defined, which results from the context.

Operation	LPL syntax	meaning
$+(-)x$	`+x`	unary plus (minus)
$\neg x$ $(\overline{x})$	`~x`	Boolean negation
(x)	`(x)`	nesting, changing precedence
x^y	`x^y`	x power y ($x \geq 0$)
$x \mod y$	`x%y`	x modulo y
$\min(x, y)$	`Min(x,y)`	the smaller of x and y
$\max(x, y)$	`Max(x,y)`	the larger of x and y
x/y	`x/y`	division
xy	`x*y`	multiplication
$x - y$	`x-y`	substraction
$x + y$	`x+y`	addition
$f(x)$	`f(x)`	function like `sin(x)` etc.
$x \geq y$	`x>=y`	greater or equal
$x \leq y$	`x<=y`	less or equal
$x > y$	`x>y`	greater
$x < y$	`x<y`	less
$x = y$	`x=y`	equal
$x \neq y$	`x<>y`	not equal
$x \wedge y$	`x and y`	Boolean AND
$x \vee y$	`x or y`	Boolean OR
$x \dot{\vee} y$	`x xor y`	Boolean exclusive-OR
$x \leftrightarrow y$	`x <-> y`	Boolean equivalence
$x \rightarrow y$	`x -> y`	Boolean implication
$x \leftarrow y$	`x <- y`	Boolean reverse implication
$\overline{x \wedge y}$	`x nand y`	Boolean NAND
$\overline{x \vee y}$	`x nor y`	Boolean NOR
$x := y$	`x := y`	assignment (returns x)
x, y	`x , y`	list operator

Table 1: Operators and operands in an expression

symbol	LPL syntax	meaning
$\sum_i x_i$	`sum{i} x`	summation
$\prod_i x_i$	`prod{i} x`	product
$\min_i x_i$	`min{i} x`	smallest value
$\max_i x_i$	`max{i} x`	largest value
$\text{argmin}_i\, x_i$	`argmin{i} x`	index of the smallest value
$\text{argmax}_i\, x_i$	`argmax{i} x`	index of the largest value
$\text{if}(x, y, z)$	`if(x,y,z)`	returns y if x is true else z
$\forall_i x_i$	`forall{i} x`	indexed Boolean AND
$\bigwedge_i x_i$	`and{i} x`	indexed Boolean AND
$\exists_i x_i$	`exist{i} x`	indexed Boolean OR
$\bigvee_i x_i$	`or{i} x`	indexed Boolean OR
$\dot{\bigvee}_i x_i$	`xor{i} x`	indexed Boolean XOR
$\neg \bigwedge_i x_i$	`nand{i} x`	indexed Boolean NAND
$\neg \bigvee_i x_i$	`nor{i} x`	indexed Boolean NOR
$\text{atleast}(k)_i\ x_i$	`atleast(k){i} x`	at least k must be true
$\text{atmost}(k)_i\ x_i$	`atmost(k){i} x`	at most k must be true
$\text{exactly}(k)_i\ x_i$	`exactly{i} x`	exactly k must be true

Table 2: Indexed Operators in an expression

3. Let $i \in I$ be a set, P_i be an array of Boolean propositions, and N als be a Boolean proposition that can be true or false. We have the statement: "At most 3 out of all P_i are true or N and exactly one P_i is true, with $i \in I$". This is an expression with indexed operators formulated as:

$$\text{atmost}(3)_i\ P_i \vee (N \wedge \dot{\bigvee}_i P_i)$$

The third expression above may occur in a production planning context. Let P_i be the proposition that "product i is manufactured" and let N be the proposition that "most machines (say at least 80%) are maintained". Then the constraint may be interpreted as: "At most 3 different products can be manufactured or exactly one product must be manufactured when most machines are maintained (N)".

The definition of the various Boolean operations is given in Table 3.

x	y	$x \wedge y$	$x \vee y$	$x \dot{\vee} y$	$x \rightarrow y$	$x \leftrightarrow y$	$x \leftarrow y$	x nand y	x nor y
1	1	1	1	0	1	1	1	0	0
1	0	0	1	1	0	0	1	1	0
0	1	0	1	1	1	0	0	1	0
0	0	0	0	0	1	1	1	1	1

Table 3: The Boolean Operators

3 Definitions and Logical Identities

The over-bar notation $\overline{x}$ is used equivalently with $\neg x$ for the Boolean negation operator. First some definitions are introduced. A *Boolean proposition* is a statement that can be true or false (1 or 0). For example "It is raining" or "the street is wet" are two statements that can be true or false. Let x and y be two Boolean propositions. A negated or un-negated proposition is called *literal* (for example: x or $\overline{y}$). An expression formed of literals and an *and* operator is called *conjunction*. An expression formed of literals and an *or* operator is called *disjunction*. A *clause* is a – possibly empty – disjunction of literals. The following two expressions are examples of clauses :

$$\overline{x} \vee \overline{y} \vee z \vee \overline{w} \vee \overline{v}$$

$$x \vee \overline{y} \vee z \vee w \vee \overline{v}$$

A *Horn-clause* is a clause with at most one un-negated literal. Above, the first clause is a Horn-clause while the second is not[2]. A *conjunctive normal form (CNF)* is a formula which is a conjunction of a set of clauses. A *disjunctive normal form (DNF)* is just a CNF where the two operators $\wedge$ (*and*) and $\vee$ (*or*) are exchanged. In the following formula the first is a CNF, while the second is a DNF :

$$(x \vee \overline{y}) \ \wedge \ (y \vee z \vee \overline{w}) \ \wedge \ v$$

$$(y \wedge \overline{x}) \ \vee \ z \ \vee \ (w \wedge \overline{v})$$

Various Boolean operators can be transformed into each other (where the symbol $\Longleftrightarrow$ is used in the sense "is equivalent to"). 9 identities are listed below.

(1) It is possible to eliminate one of the three operators (*negation*, *and*, *or*) using the following identities (Morgan's law):

$$\begin{array}{lll} x \wedge y \iff \overline{\overline{x} \vee \overline{y}} & , & \overline{x \wedge y} \iff \overline{x} \vee \overline{y} \\ x \vee y \iff \overline{\overline{x} \wedge \overline{y}} & , & \overline{x \vee y} \iff \overline{x} \wedge \overline{y} \end{array}$$

(2) The implication and the reverse implication can be replaced using:

$$\begin{array}{ll} x \rightarrow y & \Longleftrightarrow \ \overline{x} \vee y \\ x \leftarrow y & \Longleftrightarrow \ x \vee \overline{y} \end{array}$$

(3) Boolean equivalence can be substituted using:

$$\begin{array}{lll} x \leftrightarrow y & \Longleftrightarrow & (x \rightarrow y) \wedge (x \leftarrow y) \iff \\ (x \vee \overline{y}) \wedge (\overline{x} \vee y) & \Longleftrightarrow & (x \wedge y) \vee (\overline{x} \wedge \overline{y}) \end{array}$$

(4) The exclusive *or* is also defined as follows:

$$\begin{array}{lll} x \dot{\vee} y & \Longleftrightarrow & \neg(x \leftrightarrow y) \iff \\ (x \vee y) \wedge (\overline{x} \vee \overline{y}) & \Longleftrightarrow & (x \wedge \overline{y}) \vee (\overline{x} \wedge y) \end{array}$$

[2]Horn clauses play a basic role in logic programming, such as in the Prolog language.

(5) The operators *nand* and *nor* are given by a negation in Table 1:

$$\begin{array}{lll} x \text{ nand } y & \iff \overline{x \wedge y} & \iff \overline{x} \vee \overline{y} \\ x \text{ nor } y & \iff \overline{x \vee y} & \iff \overline{x} \wedge \overline{y} \end{array}$$

(6) The indexed operators have the following meanings (where all x_i are Boolean propositions and $i \in I = \{1, \ldots, n\}, n > 0$):

$$\begin{array}{lll} \bigwedge_{i \in I} x_i & \iff (x_1 \wedge x_2 \wedge \ldots \wedge x_n) & \\ \bigvee_{i \in I} x_i & \iff (x_1 \vee x_2 \vee \ldots \vee x_n) & \\ \dot{\bigvee}_{i \in I} x_i & \iff \text{exactly}(1)_{i \in I}\ x_i & \iff \sum_{i \in I} x_i = 1 \\ \text{atleast}(k)_{i \in I}\ x_i & \iff \sum_{i \in I} x_i \geq k & \\ \text{atmost}(k)_{i \in I}\ x_i & \iff \sum_{i \in I} x_i \leq k & \\ \text{exactly}(k)_{i \in I}\ x_i & \iff \sum_{i \in I} x_i = k & \end{array}$$

(7) The *at-least* operator ($\text{atleast}(k)_{i \in I}$) can also be interpreted as a CNF:

$$\text{atleast}(k)_{i \in I}\ x_i \iff \bigwedge_{S \subset I \,||S| = n-k+1} \left(\bigvee_{i \in S} x_i \right)$$

The expression on the right hand side is a conjunction of $\binom{n}{n-k+1}$ clauses consisting of exactly k positive propositions each. The *at-least* operator can be interpreted as a compact form of a possibly very large (containing many clauses) CNF.

(8) The four indexed operators *and*, *or*, *nand*, and *nor* can be expressed using the *at-least* operator using :

$$\begin{array}{llllll} \bigwedge_{i \in I} x_i & & & \iff \text{atleast}(n)_{i \in I}\ x_i & & ((1)) \\ \bigvee_{i \in I} x_i & & & \iff \text{atleast}(1)_{i \in I}\ x_i & & ((2)) \\ \mathbf{nand}_{i \in I}\, x_i & \iff \neg \bigwedge_{i \in I} x_i & \iff \bigvee_{i \in I} (\neg x_i) & \iff \text{atleast}(1)_{i \in I}\ (\neg x_i) & & ((3)) \\ \mathbf{nor}_{i \in I}\, x_i & \iff \neg \bigvee_{i \in I} x_i & \iff \bigwedge_{i \in I} (\neg x_i) & \iff \text{atleast}(n)_{i \in I}\ (\neg x_i) & & ((4)) \end{array}$$

Interpretation: (1) If and only if all expressions are true then at least all are true; (2) if and only if one expression is true then at least one is true; (3) if not all are true then al least one is false; (4) if none is true then at least all are false.

(9) One may also replace negation of the *at-least*, *at-most*, and *exactly* operators by each other:

$$
\begin{array}{lcl}
\text{atleast}(k)_{i\in I}\ x_i & \Longleftrightarrow & \text{atmost}(n-k)_{i\in I}\ (\neg x_i) \\
\text{atmost}(k)_{i\in I}\ x_i & \Longleftrightarrow & \text{atleast}(n-k)_{i\in I}\ (\neg x_i) \\
\text{exactly}(k)_{i\in I}\ x_i & \Longleftrightarrow & \text{atleast}(k)_{i\in I}\ x_i \wedge \text{atmost}(k)_{i\in I}\ x_i \\
\neg\text{atleast}(k)_{i\in I}\ x_i & \Longleftrightarrow & \text{atmost}(k-1)_{i\in I}\ x_i \\
\neg\text{atmost}(k)_{i\in I}\ x_i & \Longleftrightarrow & \text{atleast}(k+1)_{i\in I}\ x_i
\end{array}
$$

Note also that $\text{atleast}(k)_{\ldots}$ with $k > n$ and $\text{atmost}(k)_{\ldots}$ with $k < 0$ is defined to be false. Likewise, $\text{atmost}(k)_{\ldots}$ with $k \geq n$ and $\text{atleast}(k)_{\ldots}$ with $k \leq 0$ is defined to be true[3].

Using the substitutions (1) to (9) given above one can easy see that all logical operators can be reduced to a few once, for example, negation ($\neg$), *and* ($\wedge$), *or* ($\vee$), and the *at-least* (atleast()) operator. The next section shows how these identities can be used to translate logical expressions into pure mathematical formulas.

4 Translating Boolean Expressions

First of all, It is easy to verify that all Boolean operators in Table 3 can also be formulated as mathematical linear expressions if x and y are binary variables with $x, y \in \{0, 1\}$ as shown in Table 4. The true (1) and false (0) values of a proposition x in Table 3 is interpreted as numerical 1 and 0 values in Table 4. The negation $\neg x$ (or $\overline{x}$) can be substituted by $1 - x$.

x	y	$x \cdot y = 0$	$x + y \leq 1$	$x = 1 - y$	$x \geq y$	$x = y$	$x \leq y$	$x \cdot y = 0$	$x + y = 0$
1	1	1	1	0	1	1	1	0	0
1	0	0	1	1	0	0	1	1	0
0	1	0	1	1	1	0	0	1	0
0	0	0	0	0	1	1	1	1	1

Table 4: The Boolean Operators as Mathematical inequalities

Table 5 displays a collection of further equivalences and it is easy to check their identity by applying the true/false (0,1) values to each proposition (binary) and compare the result.

However, translating logical constraints into mathematical linear inequalities in a systematical way requires a precise procedure. The general idea is to transform the Boolean expression into a conjunctive normal form eventually by introducing additional (binary) variables, then it is easy to get the linear inequalities from the resulting clauses using the given identities. Before presenting a systematical procedure, let us try to translate a concrete Boolean expression into purely mathematical linear inequalities (the example is implemented in model logic1[4]) :

$$((x \wedge y) \vee z) \ \dot{\vee}\ w$$

[3] In the LPL modeling language syntax, we use the convention that $-n \leq k \leq 0$ means $0 \leq n - k \leq n$.
[4] https://lpl.matmod.ch/lpl/Solver.jsp?name=/logic1

Propositions: $P, X, Y, \ldots$	0-1 binary variables $p, x, y, \ldots$
X	$x \geq 1$
$\neg X$ (or $\overline{x}$)	$x \leq 0$
$X \vee Y$	$x + y \geq 1$
$X \vee Y \vee \ldots \vee Z$	$x + y + \ldots + z \geq 1$
$X \wedge Y$	$x \geq 1,\ y \geq 1$ (or $x + y \geq 2$
$X \wedge Y \wedge \ldots \wedge Z$	$x \geq 1,\ y \geq 1,\ \ldots,\ z \geq 1$ (or: $x + y + \ldots + z \geq n$)
$X \rightarrow Y$	$x \leq y$
$X \dot{\vee} Y$	$x + y = 1$
$X \dot{\vee} Y \dot{\vee} \ldots \dot{\vee} Z$	$x + y + \ldots + z = 1$
$X \leftrightarrow Y$	$x = y$
$X \leftrightarrow Y \leftrightarrow \ldots \leftrightarrow Z$	$x = y = \ldots = z$
X nor Y (or: $\overline{X \vee Y}$	$x \leq 0,\ y \leq 0$
$\overline{X \vee Y \vee \ldots \vee Z}$	$x + y + \ldots + z \leq 0$
X nand Y (or: $\overline{X \wedge Y}$	$x + y \leq 1$
$\overline{X \wedge Y \wedge \ldots \wedge Z}$	$x + y + \ldots + z \leq n - 1$
$X \leftrightarrow \neg Y$	$x + y \leq 1$
$P \rightarrow X \wedge Y \wedge \ldots \wedge Z$	$x \geq p,\ y \geq p,\ \ldots,\ z \geq p$ (or: $x + y + \ldots + z \geq np$)
$P \rightarrow X \vee Y \vee \ldots \vee Z$	$x + y + \ldots + z \geq p$
$X \wedge Y \wedge \ldots \wedge Z \rightarrow P$	$x + y + \ldots + z - p \leq n - 1$
$X \vee Y \vee \ldots \vee Z \rightarrow P$	$x \leq p,\ y \leq p,\ \ldots,\ z \leq p$ (or: $x + y + \ldots + z \leq np$)
$X \wedge (Y \vee Z)$	$x = 1,\ y + z \geq 1$
$X \vee (Y \wedge Z)$	$x + y \geq 1,\ x + z \geq 1$ (or: $2x + y + z \geq 2$)
$X_1 \wedge \ldots \wedge X_n \rightarrow Y_1 \vee \ldots \vee Y_n$	$x_1 + x_2 + \ldots + x_n -$ $(y_1 + y_2 + \ldots + y_n) \leq n - 1$
$P \leftrightarrow X \wedge Y \wedge \ldots \wedge Z$	$x + y + \ldots + z - np \leq n - 1,$ $x \geq p,\ y \geq p,\ \ldots,\ z \geq p$
$P \leftrightarrow X \vee Y \vee \ldots \vee Z$	$x + y + \ldots + z \geq p,$ $x \leq p,\ y \leq p,\ \ldots,\ z \leq p$

Table 5: Logic-mathematical Equivalences

Step 1: replace $a \dot{\vee} b$ with $(a \vee b) \wedge (\overline{a} \vee \overline{b})$. This gives:

$$((x \wedge y) \vee z \vee w) \ \wedge \ (\overline{(x \wedge y) \vee z} \vee \overline{w})$$

Step 2: Move the negation inwards (using the Morgan's law). This gives:

$$((x \wedge y) \vee z \vee w) \ \wedge \ (((\overline{x} \vee \overline{y}) \wedge \overline{z}) \vee \overline{w})$$

Step 3: Move the *or*-operators inwards over the *and*-operators. This gives the following conjunctive normal form (CNF) :

$$(x \vee z \vee w) \ \wedge \ (y \vee z \vee w) \ \wedge \ (\overline{x} \vee \overline{y} \vee \overline{w}) \ \wedge \ (\overline{z} \vee \overline{w})$$

Step 4: Translate the 4 clauses into 4 linear inequalities as follows:

$$\begin{array}{ll} x + z + w & \geq 1 \\ y + z + w & \geq 1 \\ 1 - x + 1 - y + 1 - w & \geq 1 \\ 1 - z + 1 - w & \geq 1 \end{array}$$

5 Translating Mixed Expressions

From a practical point of view, mixing logical (Boolean) and mathematical operations in a single formula is much more interesting and its translation into mathematics is more demanding than translating just Boolean formula into mathematics. One way to link Boolean propositions and mathematical (linear) constraints is the big-M method that is presented here[5].

A disadvantage of the big-M method is that it requires lower and upper bounds on the linear constraints as well as a small number ϵ. In the following, $U(y)$ is used as "upper bound value for y" and $L(y)$ as "lower bound value for y". In the formulas the following shortcuts are used[6]:

$$M = U\left(\sum_i a_i x_i - b\right) \quad \text{and} \quad m = L\left(\sum_i a_i x_i - b\right)$$

[5]Another method to represent a disjunction of linear constraints:

$$\mathbf{A}_1\mathbf{x} \geq \mathbf{b}_1 \ \vee \ \mathbf{A}_2\mathbf{x} \geq \mathbf{b}_2 \ \vee \ \ldots \ \vee \ \mathbf{A}_n\mathbf{x} \geq \mathbf{b}_n$$

Split the vector $\mathbf{x}$ into n components $(\mathbf{x}^{(i)}, i \in \{1, \ldots, n\})$. Then the disjunction can be formulated as:

$$\begin{array}{l} \mathbf{A}_i\mathbf{x}^{(i)} \geq \mathbf{b}_i\delta_i \\ \mathbf{x} = \sum_i \mathbf{x}^{(i)} \\ \sum_i \delta_i = 1 \\ \delta_i \in \{0,1\} \ , \ \mathbf{x}^{(i)} \geq 0 \ , \ i \in \{1, \ldots, n\} \end{array}$$

This formulation has been developed in [16]. The test is also a foundation is the concept of "MIP representability". An introduction can also be found in [7] and [6].

[6]LPL automatically calculates and uses these bounds provided that the involved variables are bounded by lower and upper bounds. An error message is generated if this is not the case

Case 1: Indicator variable implies an $\leq$ inequality

Rule	Constraint	Translation
50	$\left.\begin{array}{l}\delta \rightarrow \sum_i a_i x_i \leq b \\ \sum_i a_i x_i > b \rightarrow \overline{\delta}\end{array}\right\}$	$\sum_i a_i x_i - b \leq M \cdot (1-\delta)$
50'	$\left.\begin{array}{l}\delta \rightarrow x \leq 0 \\ x > 0 \rightarrow \overline{\delta}\end{array}\right\}$	$x \leq U(x) \cdot (1-\delta)$
50a	$\left.\begin{array}{l}\overline{\delta} \rightarrow \sum_i a_i x_i \leq b \\ \sum_i a_i x_i > b \rightarrow \delta\end{array}\right\}$	$\sum_i a_i x_i - b \leq M \cdot \delta$
50a'	$\left.\begin{array}{l}\overline{\delta} \rightarrow x \leq 0 \\ x > 0 \rightarrow \delta\end{array}\right\}$	$x \leq U(x) \cdot \delta$

Rule 50a is the same as Rule 50, except that δ has been replaced by $\overline{\delta}$. Rule 50' and 50a' are simplified versions of Rule 50 and 50a, the linear inequality has been replaced by just a single variable inequality.

Case 2: Indicator variable is implied by an $\leq$ inequality

Rule	Constraint	Translation
51	$\left.\begin{array}{l}\sum_i a_i x_i \leq b \rightarrow \delta \\ \overline{\delta} \rightarrow \sum_i a_i x_i > b \\ \overline{\delta} \rightarrow \sum_i a_i x_i \geq b + \epsilon\end{array}\right\}$	$\sum_i a_i x_i - b - \epsilon \geq (m-\epsilon) \cdot \delta$
51'	$\left.\begin{array}{l}x \leq 1 \rightarrow \delta \\ \overline{\delta} \rightarrow x > 1 \\ \overline{\delta} \rightarrow x \geq 1 + \epsilon\end{array}\right\}$	$x - 1 - \epsilon \geq (L(x-1)-\epsilon) \cdot \delta$
51a	$\left.\begin{array}{l}\sum_i a_i x_i - b \leq 0 \rightarrow \overline{\delta} \\ \delta \rightarrow \sum_i a_i x_i > b \\ \delta \rightarrow \sum_i a_i x_i \geq b + \epsilon\end{array}\right\}$	$\sum_i a_i x_i - b - \epsilon \geq (m-\epsilon) \cdot (1-\delta)$
51a'	$\left.\begin{array}{l}x \leq 1 \rightarrow \overline{\delta} \\ \delta \rightarrow x > 1 \\ \delta \rightarrow x \geq 1 + \epsilon\end{array}\right\}$	$x - 1 - \epsilon \geq (L(x-1)-\epsilon) \cdot (1-\delta)$

Case 2 cannot be properly modeled. An small number $\epsilon > 0$ must be introduced to deal with $\sum_i a_i x_i - b > 0$. One may verify the transformation first with the simpler case 51':

Suppose the lower bound of x is 0 and $\epsilon = 0.1$, then $(L(x-1)$ is -1. Suppose that $x \leq 1$, then the linear inequality reduces to $-1.1 + x \geq -1.1\delta$, and we must derive that $\delta = 1$, suppose now $x \geq 1 + \epsilon$ then the linear inequality reduces again to $s \geq -1.1\delta$ with $s \geq 0$, and δ maybe 0 or 1, verifying the implication.

Case 3: Indicator variable implies an $\geq$ inequality

Rule	Constraint	Translation
52	$\left.\begin{array}{l} \delta \rightarrow \sum_i a_i x_i \geq b \\ \sum_i a_i x_i < b \rightarrow \overline{\delta} \end{array}\right\}$	$\sum_i a_i x_i - b \geq m \cdot (1 - \delta)$
52'	$\left.\begin{array}{l} \delta \rightarrow x \geq 0 \\ x > 0 \rightarrow \overline{\delta} \end{array}\right\}$	$x \geq L(x) \cdot (1 - \delta)$
52a	$\left.\begin{array}{l} \overline{\delta} \rightarrow \sum_i a_i x_i \geq b \\ \sum_i a_i x_i < b \rightarrow \delta \end{array}\right\}$	$\sum_i a_i x_i - b \geq m \cdot \delta$
52a'	$\left.\begin{array}{l} \overline{\delta} \rightarrow x \geq 0 \\ x < 0 \rightarrow \delta \end{array}\right\}$	$x \geq L(x) \cdot \delta$

Rule 52a is the same as Rule 52, except that δ has been replaced by $\overline{\delta}$. Rule 52' and 52a' are simplified versions of Rule 52 and 52a, the linear inequality has been replaced by just a single variable inequality.

Case 4: Indicator variable is implied by an $\geq$ inequality

Rule	Constraint	Translation
53	$\left.\begin{array}{r} \sum_i a_i x_i \geq b \rightarrow \delta \\ \overline{\delta} \rightarrow \sum_i a_i x_i < b \\ \overline{\delta} \rightarrow \sum_i a_i x_i \leq b - \epsilon \end{array}\right\}$	$\sum_i a_i x_i - b + \epsilon \leq (M + \epsilon) \cdot \delta$
53'	$\left.\begin{array}{r} x \geq 1 \rightarrow \delta \\ \overline{\delta} \rightarrow x < 1 \\ \overline{\delta} \rightarrow x \leq 1 - \epsilon \end{array}\right\}$	$x - 1 + \epsilon \leq (U(x-1) + \epsilon) \cdot \delta$
53a	$\left.\begin{array}{r} \sum_i a_i x_i \geq b \rightarrow \overline{\delta} \\ \delta \rightarrow \sum_i a_i x_i < b \\ \delta \rightarrow \sum_i a_i x_i \leq b - \epsilon \end{array}\right\}$	$\sum_i a_i x_i - b + \epsilon \leq (M + \epsilon) \cdot (1 - \delta)$
53a'	$\left.\begin{array}{r} x \geq 1 \rightarrow \overline{\delta} \\ \delta \rightarrow x < 1 \\ \delta \rightarrow x \leq 1 - \epsilon \end{array}\right\}$	$x - 1 + \epsilon \leq (U(x-1) + \epsilon) \cdot (1 - \delta)$

Basically, the four cases (Case 1 to Case 4) can be reduced to two cases in fact. Case 4 is the same as case 1 if we replace $A < B$ by $A \leq B - \epsilon$, or in other words, in the linear inequalities of Case 4 the right hand side b is replaced by $b - \epsilon$ (hence M also changes). Equally, Case 2 can be reduced to Case 3 by replacing $A > B$ by $A \geq B + \epsilon$, or in other words, in the linear inequalities of Case 2 the right hand side b is replaced by $b + \epsilon$ (hence M also changes).

Note also that a case with equalities $\delta \rightarrow \sum_i a_i x_i = b$ can be modeled by applying two rules separately since:

$$\sum_i a_i x_i - b = 0 \iff \left(\sum_i a_i x_i - b \leq 0\right) \wedge \left(\sum_i a_i x_i - b \geq 0\right)$$

5.1 Some Examples

The five following examples are modeled in logic0[7] in the modeling language LPL.

Example 1: We would like to decide whether a particular depot should be built or not. The decision to build or not to build can be modeled by a binary variable δ: $\delta = 1$ mean "to build an depot" and $\delta = 0$ mean's "not to build the depot". Now this depends on whether there is a quantity $z > 0$ delivered from this depot. This means: "if there is a certain quantity delivered from a depot then the depot should be built, that is, supposing the upper bound of z is $M > 0$ (see rule 50a):

$$z > 0 \rightarrow \delta = 1 \quad \text{is translated to} \quad z \leq M\delta$$

[7] https://lpl.matmod.ch/lpl/Solver.jsp?name=/logic0

If equivalence is required, then

$$z > 0 \leftrightarrow \delta = 1 \quad \text{is translated to} \quad \begin{cases} z \leq M\delta \\ z \geq \epsilon\delta; \end{cases}$$

Example 2: Model the following statement: "If a product is manufactured at all then there is an additional setup (or fixed) cost." Suppose that $z \geq 0$ represents the quantity of a product manufactured at a marginal cost per unit of c_1. If the product is manufactured at all there is a additional unique setup cost of c_2. Hence, two cases need to be distinguished:

1. Either nothing is manufactured, then $z = 0$ and total cost $= 0$.
2. Or something is manufactured, then $z > 0$ and total cost $= c_1 z + c_2$

To model this situation, a Boolean (indicator) variable δ is introduced which is true if "something is manufactured" ($z > 0$), if nothing is manufactured the δ is false. Now the indicator variable can be linked with the quantity z by the implication (meaning "if the quantity z is strictly larger than 0 then something is manufactured"):

$$z > 0 \rightarrow \delta \qquad \text{or} \qquad \overline{\delta} \rightarrow z \leq 0$$

Applying rule 50a gives the linear constraint: $z \leq U(z) \cdot \delta$
The cost function is formulated as: cost $= c_1 z + c_2 \delta$
To verify the resulting model, the values of δ are checked:
(1) if $\delta = 1$ (true) then one has: $z \leq U(z)$, cost $= c_1 z + c_2$
(2) If $\delta = 0$ (false) then one has: $z \leq 0$, cost $= 0$
In this case in LPL it is even not needed to introduce a binary variable. The constraint can be formulated as:

$$c_1 z + c_2(z > 0)$$

The $z > 0$ is automatically replaced by an additional binary variable.

Example 3: Two quantities must be link logically. For example: "If product A is included in the blend then also at least a quantity b of product B must be included". That is: $z_A > 0 \rightarrow z_B \geq b$

To model this situation, an indicator variable δ is introduced with the meaning "A is included in the blend" that is: $z_A > 0 \rightarrow \delta$. And the requirement "if product A is included in blend then also at least b of product B must be included" can be modeled as: $\delta \rightarrow z_B \geq b$

The two constraints can be translated to mathematical inequalities by applying the two Rules 51a and 53a, giving:

$$\begin{aligned} z_A &\leq U(z_A) \cdot \delta \\ z_B &\geq L(z_B + b) \cdot \delta \end{aligned}$$

To verify the resulting model, check for the values of δ:
(1) If $\delta = 1$ (true) then $z_A \leq U(z_A)$, $z_B \geq b$ (supposing $L(z_B) = 0$)
(2) If $\delta = 0$ (false) then $Z_A \leq 0$, $z_B \geq 0$

Example 4: A constraint A must hold if and only if the indicator variable is true. For example: $2v + 3w \leq 1 \leftrightarrow \delta$. Hence they are:

$$2v + 3w \leq 1 \rightarrow \delta \quad \text{and} \quad \delta \rightarrow 2v + 3w \leq 1$$

Applying two rules, one gets (supposing $L(v) = L(w) = 0$, $U(v) = U(w) = 1$, $\epsilon = 0.001$):

$$2v + 3w + 4\delta \leq 5 \quad \text{and} \quad 2v + 3w + 1.001\delta \geq 1.001$$

Example 5: Model the following constraint, supposing $L(x) = L(y) = 0$, $U(x) = U(y) = 10$, and $\epsilon = 0.0001$:

$$3x + 4y \leq 5 \ \vee \ 2x + 3y \geq 4 \ \leftrightarrow \ x + 2y \geq 3 \ \vee \ 2x + 4y \leq 5$$

The details are given in the text logic0. Note that these examples are from [8] and [7].

6 Translation Procedure

The automated translation of a logical constraint to a linear mathematical constraint in LPL is done in several steps applying various rules. Each rule is applied recursively on an (constraint) expression tree. In the following transformation procedure, the index-set $i \in \{1, \ldots, n\}$ is used for indexed expressions, the names x and y or there indexed form x_i are used to denote arbitrary expressions or just variables. The letter x or v_i denotes a single or indexed binary variable.

To understand the transformation procedure, the concept of a "syntax tree" and the concept of "cut-off" are introduced first. A syntax tree is a particular representation of an arbitrary (constraint) expression. Figure 1 shows the syntax trees of the following three expressions:

$$3 + 4 \cdot 5 \quad , \quad (3 + 4) \cdot 5 \quad , \quad \neg x \wedge (y \vee z)$$

For binary operators like + (addition) or $\vee$ (or operator), the node (represented as a ellipse) has a left and a right child connected by an arc, for unary operators like $\neg$ there is only a left child (the right is a null pointer, see the node *not*.

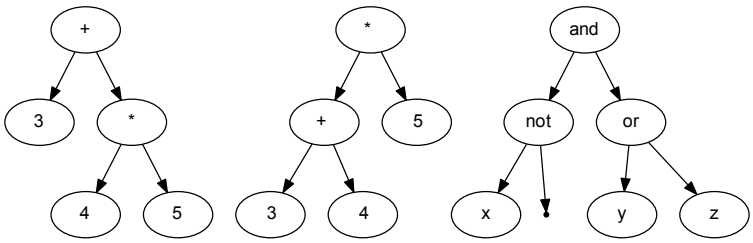

Figure 1: Syntax Trees I

Indexed operators as well as function calls are considered as unary operators: See Figure 2 which represents the three syntax trees of the expressions:

$$\sum_i x_i y \quad , \quad \bigvee_i (x_i \wedge y) \quad , \quad \text{if}(a < b, x, y)$$

The concept of "cut-off" is the operation of cutting a subtree off the main syntax tree. In mathematics, this operation is called "substitution": a (sub-)expression is replaced

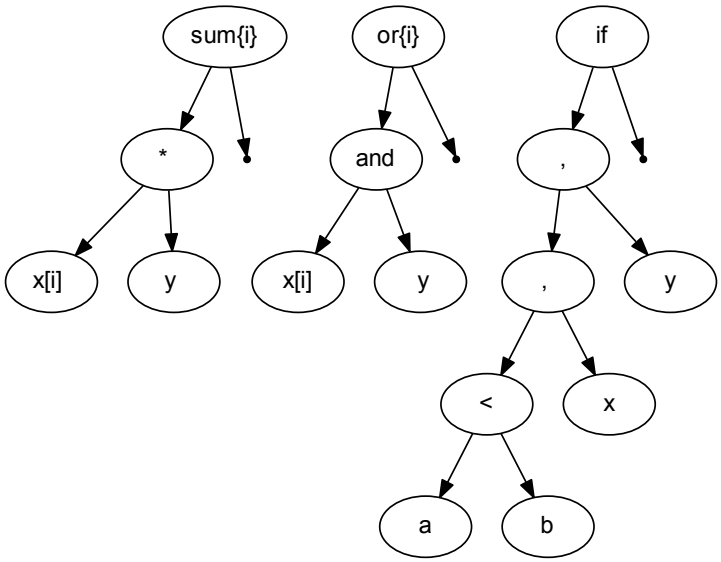

Figure 2: Syntax Trees II

by an additional variable as in the following expression where yz is substituted by an additional variable w :

$$x + yz = 17 \quad \text{becomes} \quad \begin{array}{ll} x + w & = 17 \\ w & = yz \end{array}$$

Represented as a syntax tree, the original tree is decomposed into two (smaller) trees as shown in Figure 3, the left trees is decomposed into the two expressions on the right.

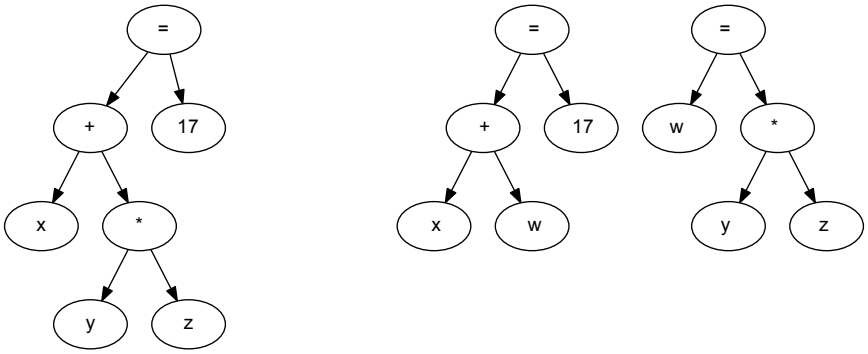

Figure 3: Cut-off a Subtree

Boolean expressions can also be cut-off. In this case, the equality sign ($=$) must be replaced by equivalence ($\leftrightarrow$). In most cases, however an implication ($\rightarrow$) is sufficient. An example is given as follows with x and y being Boolean propositions and p and q are numerical (real) variables (the trees are shown in Figure 4) :

$$(p + q \geq 10) \vee x \wedge y \quad \text{becomes} \quad \begin{array}{l} \delta \vee x \wedge y \\ \delta \rightarrow (p + q \geq 10) \end{array}$$

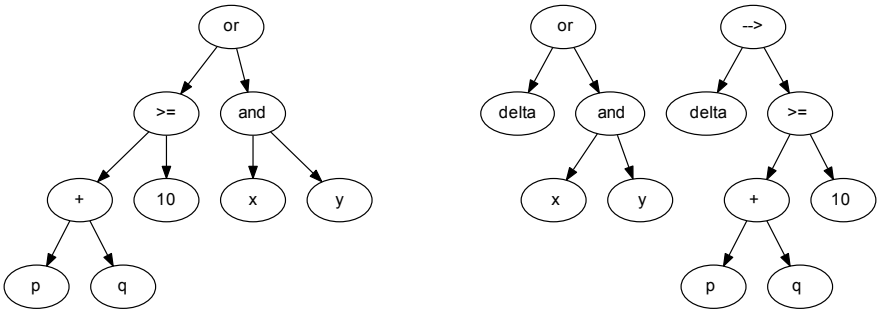

Figure 4: Cut-off a Boolean Subtree

In the procedure below, a subtree containing real variables, numerical or relational operators and where the root node is *not* a logical operator, is called *R-subtree*. For example, in the left part of Figure 4, the subtree with the root of $\geq$ is a R-tree. The sequence in which the following steps are applied has a large influence on on how efficient the resulting expressions are. The sequence can somewhat change depending on the expressions.

Step 0: Cut-off Subtrees

In this step a sub-tree of the syntax-tree is "cut off". An R-subtree A is cut from a tree T if and only if the path from the root of T to the root of A contains the Boolean operators of equivalence ($\leftrightarrow$) or the exclusive-or ($\dot{\vee}$).

Step 1: T0-Rules

Several operators are substituted by other operators defined by the previous definitions. Table 6 give the list of the operators. Note that all rules are recursively applied to all constraints. (These operators can be eliminated at parse time.)

Rule	Operator	replaced by
01	x nor y	$\overline{x \vee y}$
02	x nand y	$\overline{x \wedge y}$
03	$\bigwedge_i x_i$ (indexed-and)	$\forall_i x_i$ (forall-operator)
04	$\bigvee_i x_i$ (indexed-or)	$\exists_i x_i$ (exist-operator)
05	$\dot{\bigvee}_i x_i$ (indexed xor)	exactly$(1)_i$ x_i (exactly-one)
06	**nor**$_i$ x_i (indexed-nor)	atmost$(0)_i$ x_i
07	**nand**$_i$ x_i (indexed-nand)	atmost$(n-1)_i$ x_i
08	x op1 y op2 z ($\text{op1,op2} \in \{=, \neq, <, \leq, >, \geq\}$)	x op1 y $\wedge$ y op2 z

Table 6: T0-Rules

Step 2: T1-Rules

Several operators are substituted by other operators defined by the previous definitions. Table 7 give the list of the operators. Step 1 and Step 2 could be done in a single step.

Rule	Operator	replaced by
10	$x \rightarrow y$	$\overline{x} \vee y$
11	$x \leftarrow y$	$x \vee \overline{y}$
12	$\forall_i x_i$	$\text{atleast}(n)_i\ x_i$ (forall-operator)
13	$\exists_i x_i$	$\text{atleast}(1)_i\ x_i$ (exists-operator)
14	$\text{atleast}(n-k)_i\ x_i$	$\text{atmost}(k)_i\ \overline{x_i}$
15	$\text{atmost}(n-k)_i\ x_i$	$\text{atleast}(k)_i\ \overline{x_i}$
16	$\text{exactly}(0)_i\ x_i$	$\text{atmost}(0)_i\ x_i$

Table 7: T1-Rules

Step 3: T2-Rules

The NOT-Operator is pushed down the syntax tree and some other reductions are processed. All rules are listed in Table 8.

Rule	Operator	replaced by
20	$\neg(\neg x)$	x
21	$\neg(x \wedge y)$	$\neg x \vee \neg y$
22	$\neg(x \vee y)$	$\neg x \wedge \neg y$
23	$\neg(x \leftrightarrow y)$	$x \dot{\vee} y$
24	$\neg(x \dot{\vee} y)$	$x \leftrightarrow y$
25	$\neg(x\ [\leq, \geq, <, >, =, \neq]\ y)$	$x\ [>, <, \geq, \leq, \neq, =]\ y$
26	$\neg\text{atleast}(k)_i\ x_i\ ,\ (k \geq 1)$	$\text{atmost}(k-1)_i\ \neg x_i$
26a	$\neg\text{atleast}(0)_i\ x_i$	$\text{atleast}(1)_i\ \neg x_i$
27	$\neg\text{atmost}(k)_i\ x_i$	$\text{atleast}(k+1)_i\ \neg x_i$
28	$\neg\text{exactly}(k)_i\ x_i\ ,\ (k \geq 1)$	$\text{atmost}(k-1)_i\ x_i\ \vee\ \text{atleast}(k+2)_i\ x_i$
29	$x = 1\ (1 = x, x \neq 0, 0 \neq x)$	x
29a	$x = 0\ (0 = x, x \neq 1, 1 \neq x)$	$\overline{x}$
30	$x = y$	$x \leftrightarrow y$
31	$x \leftrightarrow y$	$(x \vee \overline{y})\ \wedge\ (\overline{x} \vee y)$
32	$x \neq y$	$x < y \vee x > y$
33	$x \dot{\vee} y$	$(x \vee y)\ \wedge\ \overline{(x \vee y)})$
34	$\text{exactly}(k)_i\ x_i\ ,\ (k \geq 1)$	$\sum_i x_i = k$
35	$\text{atmost}(k)_i\ x_i\ ,\ (k \geq 1)$	$\sum_i x_i \leq k$
35a	$\text{atmost}(0)_i\ x_i$	$\text{atleast}(0)_i\ \neg x_i$
36	$\text{atleast}(k)_i\ x_i\ ,\ (k \geq 2)$	$\sum_i x_i \geq k$

Table 8: T2-Rules

Step 4: Disconnect nested *and*-s

In this step a sub-tree of the syntax-tree is "cut off". Cut-off means to replace a subtree by an additional binary variable and to add a constraint corresponding to the subtree. Suppose we have the following expression, where E_1 and E_2 are mathematical expressions containing other than binary variables and operators (addition, multiplication, or relational ops) and x and y are two binary variables:

$$x \ \vee \ y \ \vee \ (E_1 \wedge E_2) \quad [\text{for example: } x \ \vee \ y \ \vee (a \leq 3 \ \wedge \ b \geq 2)]$$

In this case, the constraint is replace by the following two constraints (with the additional binary δ) :

$$x \vee y \vee \delta \qquad , \qquad \delta \rightarrow (E_1 \wedge E_2)$$

It is important to note that this cut-off is only applied on a **and** tree node – the and may be indexed – and only if the subtrees contains real variables. A binary **and** node is cut-off only if on the path from the node to the root at least 1 **or** node appears, and a indexed **and** node is only cut-off, if at least 2 **or** nodes appear.

Step 5: T3-Rules

In this step remaining **and**-s are pushed upwards in the syntax tree over the **or**-s as much as possible. The rules are listed in Table 9. Note that n is the cardinality of the set I and $\text{atleast}(n)_i$ is the same as the indexed **and**. This step can be applied more than once to force a CNF before subtrees are cut.

Rule	Operator	replaced by
40	$x \ \vee \ (y \wedge z)$	$(x \vee y) \wedge (x \vee z)$
40a	$(y \wedge z) \ \vee \ x$	$(y \vee x) \wedge (z \vee x)$
41	$x \ \vee \ \text{atleast}(n)_i \ y_i$	$\text{atleast}(n)_i \ (x \vee y_i)$
41a	$\text{atleast}(n)_i \ y_i \ \vee \ x$	$\text{atleast}(n)_i \ (y_i \vee x)$

Table 9: T3-Rules

Step 6: Disconnect Sub-trees

This step is similar to step 4: Sub-trees are "cut off" again. The cut-off is a little bit more involving. The cut-off takes place in several situations

1. Like in step 4, remaining *and*-s nested in *or*-s are cut-off in all cases (even if the subtree only contains binary variables) in the same way as in step 4. For example:

   $$x \vee (y \wedge z) \quad \text{will become} \quad x \vee \delta \ , \ \delta \rightarrow (y \wedge z)$$

2. Let E_1 and E_2 again be any mathematical expression. Then any expression of the form

   $$E_1 \wedge E_2 \quad \text{will become} \quad E_1 \ , \ E_2$$

   That is, the and-operator just decomposes the two operands into two separate constraints.

3. To normalize remaining logical expressions, an expression of the form

$$E_1 \vee x \quad \text{will become} \quad x \vee E_1$$

4. If a parent node in the syntax tree is not a logical operator and the child node is a logical operator or a relational operator then the subtree is disconnected. Example:

$$a \cdot (b > 0) \quad \text{will become} \quad a \cdot \delta \ , \ \delta \leftarrow b > 0$$

   We suppose that a is a parameter and b is a variable. (a may also be a variable in this case the model is non-linear.)

5. An expression of the form

$$E_1 \vee E_2 \quad \text{will become} \quad E_1 \rightarrow \delta \ , \delta \rightarrow E_2$$

6. Finally, expression with more than one *or*

$$E_1 \vee E_2 \vee x \vee E_3 \quad \text{will become} \quad \delta_1 + \delta_2 + x + \delta_3 \geq 1 \ , \ \delta_i \rightarrow E_i \ (i \in \{1, \ldots, 3\})$$

After this step, the only form with logical operators are $x \vee E_1$ or $\overline{x} \vee E_1$.

Step 7: T4-Rules

In this step several modifications are done and the final *or* operator is eliminated.

1. An expression x is transformed to $x \geq 1$, and $\overline{x}$ is transformed into $x \leq 0$, if x is a binary.

2. if $\overline{x}$ occurs in a nested expression then it is transformed to $1 - x$.

3. The two operators $<$ and $>$ are eliminated. If the expressions E_1 and E_2 are integer expressions then $\epsilon = 1$ else $\epsilon = 0.00001$.

$$E_1 < E_2 \quad \text{will become} \quad E_1 \leq E_2 - \epsilon$$

$$E_1 > E_2 \quad \text{will become} \quad E_1 \geq E_2 + \epsilon$$

4. Basically, there is only a single logical expression left to be translated: $x \vee E_1$ (or: $\overline{x} \rightarrow E_1$)., where E_1 is an equation or inequality.

7 Conclusion

A procedure was presented to translate mixed constraints, which contain mathematical and logical operators, into mixed integer constraints – some details have been left out. Such a procedure can be very useful, especially for large mathematical models which are enriched and extended by a few logical constraints. It is also helpful for symbolic manipulation of such constraints. However, there is certainly no claim here that such a translation procedure is practical in all circumstances.

Appendix: Several Model Examples

8 Satisfibility I (sat1)

Logical inference means to deduce logical propositions from a set of premises. The conventional way to solve logical problems is by truth tables, Boolean algebra (transformation laws), and by a well known tree searching procedure, called resolution. Another symbolic method to this inference problem is natural deduction. Still another way is to use numeric (or quantitative) methods to solve an inference problem; that is to convert the Boolean expressions into linear (or non-linear) constraints in order to prove the logical argument using mathematical programming. This has two advantages: (a) in some cases this leads to faster algorithms, (b) the analysis of the quantitative models leads to the unveiling of hidden mathematical structure. Two further advantages are: (c) symbolic (logical) and numeric (mathematical) knowledge can be mixed and represented in the same framework, (d) default or other non-monotonic knowledge can also be modeled using the same framework.

The *satisfiability problem (SAT)* is to decide whether a Boolean statement F follows from another statement E; or, in other words, whether $E \rightarrow F$ is valid. This problem was the first problem proven to be NP-complete. The model here is a small instance of the general satisbiability problem.

One way to solve this problem using mathematical programming is to translate E and F into two CNF. From these two CNFs it is easy to build the two linear system $\mathbf{Ax} \geq \mathbf{a}$ and $\mathbf{Bx} \geq \mathbf{b}$. Then we solve the problem:

$$\begin{array}{ll} \min & x_0 \\ \text{subject to} & x_0 e + \mathbf{Bx} \geq \mathbf{b} \\ & \mathbf{Ax} \geq \mathbf{a} \\ & \mathbf{x} \in \{0,1\} \end{array}$$

(where e is a column unit vector, x_0 is an additional binary variable). If the minimum value x_0 is 1, then $E \rightarrow F$ is valid, that is, E implies F.

Another method, that does not use an additional binary x_0, is to pick an arbitrary inequations (clause) within $Bx \geq b$, say $B^{(k)}x \geq b_k$, and to solve the linear problem

$$\begin{array}{lll} \min & B^{(k)}x & \\ \text{subject to} & B^{(i)}x \geq b_i & \forall\, i : i \neq k \\ & \mathbf{Ax} \geq \mathbf{a} & \\ & \mathbf{x} \in \{0,1\} & \end{array}$$

If $\lceil B^{(k)}x^* \rceil \geq b_k$ (with the optimal solution x^*), then E implies F. (LPL adopt the second method).[8] Here is a small problem instance.

Problem: Is the following argument correct? "If fallout shelters are built, other countries will feel endangered and our people will get a false sense of security. If other countries will feel endangered they may start a preventive war. If our people will get a false sense of security, they will put less effort into preserving peace. If fallout shelters are not built, we run the risk of tremendous losses in the event of war. Hence, either other countries

[8] Interestingly, the expression $B^{(k)}x^*$ needs not to be integer. Hooker [9] asserts that 88% of all random generated SAT problems can be solved this way without branching.

may start a preventive war and our people will put less effort into preserving peace, or we run the risk of tremendous losses in the event of war." (see [5]).

Modeling Steps

1. The following 6 Boolean propositions (binary variables) are introduced:

```
p "fallout shelters are built"
q "other countries will feel endangered"
r "other people will get a false sense of security"
s "other countries may start a preventive war"
t "our people will put less effort into preserving pease"
u "we run the risk of tremendous losses in the event of war"
```

2. Then the first statement can be formulated as: $p \to q \wedge r$.
3. The second is to be formulated as: $q \to s$.
4. The third is: $r \to t$.
5. The fourth is: $\neg p \to u$.
6. And the final last statement is: $s \wedge t \vee u$.

We therefore have to prove $E \to F$, with :

$$E \iff (p \to q \wedge r) \wedge (q \to s) \wedge (r \to t) \wedge (\neg p \to u) \quad \text{and} \quad F \iff (s \wedge t \vee u)$$

Transforming the expressions into CNFs gives:

$$E \iff (\overline{p} \vee q) \wedge (\overline{p} \vee r) \wedge (\overline{q} \vee s) \wedge (\overline{r} \vee t) \wedge (p \vee u)$$

$$F \iff (u \vee s) \wedge (u \vee t)$$

We choose the first clause in F :

$$\min\ B^{(k)}x \quad \iff \quad \min\ u + s$$

The other clause in F is $B^{(i)}x \geq b_i \iff u + t \geq 1$

The 5 clauses in E corresponds to the five inequalities as follows:

$$-p + q \geq 0, \quad -p + r \geq 0, \quad -q + s \geq 0, \quad -r + t \geq 0, \quad p + u \geq 1$$

The complete optimization problem is:

$$\begin{array}{ll} \min & u + s \\ \text{subject to} & u + t \geq 1 \\ & -p + q \geq 0 \\ & -p + r \geq 0 \\ & -q + s \geq 0 \\ & -r + t \geq 0 \\ & p + u \geq 1 \\ & p, q, r, s, t, u \in \{0, 1\} \end{array}$$

Listing 1: The Complete Model implemented in LPL [18]

```
model SAT1 "Satisfibility I";
  binary variable
    p "fallout shelters are built";
    q "other countries will feel endangered";
    r "other people will get a false sense of security";
    s "other countries may start a preventive war";
    t "our people will put less effort into preserving pease";
    u "we run the risk of tremendous losses in the event of war";
  constraint
    s1: p -> q and r;
    s2: q -> s;
    s3: r -> t;
    s4: ~p -> u;
  minimize
    s5: s and t or u;
  Write('%s', if (s5,'The argumentation is correct','The argumentation
     is not correct'));
  --Write('EQU');
end
```

The IP solver finds that $u + s \geq 1$ – that is the objective function is larger or equal than 1. This means that the argument is correct.

Further Comments: The common technique to solve this problem is by *resolution and unification*, a technique well known in logic and in logic programming languages, such as Prolog. The first step is to negate the consequence and to transform the whole statement into a conjunctive (or disjunctive) normal form, then to prove a contradiction. The CNF of $E \rightarrow \neg F$ of the problem is as follows:

$$\begin{array}{ll} \overline{p} \vee q & (1a) \\ \overline{p} \vee r & (1b) \\ \overline{q} \vee s & (2) \\ \overline{r} \vee t & (3) \\ p \vee u & (4) \\ \overline{s} \vee \overline{u} & (5a) \\ \overline{t} \vee \overline{u} & (5b) \end{array}$$

A possible resolution sequence is:

- (4) together with (5a) generates the resolvent: $p \vee \overline{s}$ (6),
- (1a) together with (2) generates the resolvent: $\overline{p} \vee s$ (7),
- (6) together with (7) generates the resolvent: $\overline{p} \vee p$.

which produces an empty clause, proving the contradiction. Hence the argument is valid.

Question (Answer see 8)

1. Verify how LPL translates these formulae into 0-1-constraints by generating the EQU-file. (Add the Instruction `Write('EQU');`).
2. Form the LP relaxation of the linear problem generated by LPL and solve it. What do you observe?

Answer (Question see 8)

1. The result of LPL is the same as derived above :

```
min s5:  + u + s
s1:  + q - p >= 0
s2:  + s - q >= 0
s3:  + t - r >= 0
s4:  + u + p >= 1
X37:  + r - p >= 0
X38:  + u + t >= 1
```

2. Solve the following LP relaxation

```
model x;
  variable p;q;r;s;t;u;
  minimize obj:  + u + s;
  constraint
    s1:  + q - p >= 0;
    s2:  + s - q >= 0;
    s3:  + t - r >= 0;
    s4:  + u + p >= 1;
    X58X:  + r - p >= 0;
    X59X:  + u + t >= 1;
end
```

The optimum is 1. Since we have $\lceil B^{(k)}x^* \rceil \geq 1$ and $b_k = 1$, we also have $\lceil B^{(k)}x^* \rceil \geq b_k$, and the argument has been proven using the LP relaxation only. By the way, all clauses are *Horn clauses*, therefore, the problem is solvable by the LP-relaxation.

9 Satisfibility II (sat2)

Problem: Is the following set of statements logically consistent? "If the bond market goes up or if interest rates decrease, either the stock market goes down or taxes are not raised. The stock market goes down when and only when the bond market goes up and taxes are raised. If interest rates decrease, then the stock market does not go down and the bond market does not go up. Either taxes are raised or the stock market goes down and interest rates decrease." (see [5]).

Modeling Steps

1. We introduce four Boolean propositions:

```
p  "the bond market goes up"
q  "interest rates decrease"
r  "the stock market goes down"
s  "taxes are not raised"
```

2. Then the first statement can be formulated as: $(p \vee q) \rightarrow (r \vee s)$.

3. The second then is: $r \leftrightarrow (p \wedge \overline{s})$.

4. The third is: $q \rightarrow (\overline{r} \wedge \overline{p})$.

5. The final statement (the consequence) is: $\overline{s} \vee (r \wedge q)$

Hence we need to prove the following statement:

$$[((p \vee q) \rightarrow (r \vee s)) \wedge (r \leftrightarrow (p \wedge \overline{s})) \wedge (q \rightarrow (\overline{r} \wedge \overline{p}))] \rightarrow [\overline{s} \vee (r \wedge q)]$$

We minimize the last statement $\overline{s} \vee (r \wedge q)$. If the minimal value of this expression is 1, then we know, that the argument holds. Why? Because this expression must be true under all interpretations. Therefore it follows from the first three statements.

Listing 2: The Complete Model implemented in LPL [18]

```
model SAT2 "Satisfibility II";
  binary variable
    p "the bond market goes up";
    q "interest rates decrease";
    r "the stock market goes down";
    s "taxes are not raised";
  constraint
    s1: (p or q) -> (r or s);
    s2: r <-> (p and ~s);
    s3: q -> (~r and ~p);
  minimize s4:  ~s or (r and q);
  Write('%s', if (s4,'The argument is consistent','The argument is not
     consistent'));
end
```

LPL translates this statement into the following linear model:

```
min:  + r - s;
s1:  + s + r - p >= 0;
s2:  + s - p + r >= 0;
s3:  - r - q >= -1;
X55X:  + s + r - q >= 0;
X56X:  + p - r >= 0;
X57X:  - p - q >= -1;
X58X:  + q - s >= 0;
X59X:  - s - r >= -1;
```

Solution: The argument is *not* correct.

Further Comments: Note: Williams [5] gives the following formula (22): $p+q+r <= 1$ for $s3$ which is not correct (a correct formula is produced by LPL) also formula (20) in Williams is not correct.

Question (Answer see 9)

1. Verify that LPL's translation is correct, by generating the CNF using the method given in model sat1[9]
2. Verify that the argument *cannot* be proven with the LP relaxation.

Answer (Question see 9)

1. Go to model sat1[10]
2. Build the LP relaxation of the problem and solve it. The optimum is -1. This is smaller than the value of $B^(k)$ which is 0. Therefore we have no proof.

[9] https://lpl.matmod.ch/lpl/Solver.jsp?name=/sat1
[10] https://lpl.matmod.ch/lpl/Solver.jsp?name=/sat1

10 Satisfibility III (sat3)

Problem: Is the following set of statements correct? "If Superman were able and willing to prevent evil, he would do so. If Superman were unable to prevent evil, he would be impotent and if he were unwilling to prevent evil, he would be malevolent. Superman does not prevent evil. If Superman exists, he is neither impotent nor malevolent. Therefore, Superman does not exist." (see [4]).

Modeling Steps

1. We introduce 6 Boolean propositions as follows:

```
a   "Superman is able to prevent evil"
w   "Superman is willing to prevent evil"
i   "Superman is impotent"
m   "Superman is malevolent"
p   "Superman prevents evil"
e   "Superman exists"
```

2. Then the first statement can be formulated as: $a \wedge w \rightarrow p$.
3. The second is: $(\overline{a} \rightarrow i) \wedge (\overline{w} \rightarrow m)$.
4. The third is: $\overline{p}$.
5. The fourth is: $e \rightarrow \overline{i \vee m}$.
6. The last statement (the consequence) is: $\overline{e}$.

Hence, we need to show the following Boolean expression:

$$[(a \wedge w \rightarrow p) \wedge (\overline{a} \rightarrow i) \wedge (\overline{w} \rightarrow m) \wedge \overline{p} \wedge (e \rightarrow \overline{i \vee m})] \rightarrow (\overline{e})$$

Listing 3: The Complete Model implemented in LPL [18]

```
model SAT3 "Satisfibility III";
  binary variable
    a "Superman is able to prevent evil";
    w "Superman is willing to prevent evil";
    i "Superman is impotent";
    m "Superman is malevolent";
    p "Superman prevents evil";
    e "Superman exists";
  constraint
    s1:  a and w -> p;
    s2:  (~a -> i) and (~w -> m);
    s3:  ~p;
    s4:  e -> i nor m;
  minimize ne: ~e "Superman does not exist";
  Write('%s', if (e,'Superman exists.','Superman does not exist'));
end
```

Solution: The Boolean statement is correct, that is, the argumentation is correct and unfortunately, Superman does not exist.

Question (Answer see 10)

1. Verify that LPL's translation is correct, by generating the CNF using the method given in model sat1[11]

2. Verify that the argument *cannot* be proven with the LP relaxation.

Answer (Question see 10)

1. The LPL translation is as follows

```
min:   -e;
s1:     p - w - a >= -1;
s2:     i + a >= 1;
s4:    -i - e >= -1;
X56X:   m + w >= 1;
X57X:  -m - e >= -1;
```

2. Build the LP relaxation of the problem and solve it. The optimum is -1. This is smaller than the value of $B^(k)$ which is 0. Therefore we have no proof.

[11] https://lpl.matmod.ch/lpl/Solver.jsp?name=/sat1

11 Satisfibility IV (sat4)

Problem: Check whether x_2 follows from the following conjunctive normal form (CNF):

$$(x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_1} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3} \vee x_4) \wedge (x_1 \vee \overline{x_3})$$

Modeling Steps

1. Each of the five clauses can be formulated as a constraint, since a list of constraints is the same as a conjunction of constraints.

2. Minimize the expression x_2 subject to the five clauses is all we need to do. Why? Suppose minimizing x_2 result in $x_2 = 1$. This means that it is impossible to set $x_2 = 0$. We conclude that if the five clauses are true then x_2 must also be true. Suppose conversely, that the minimizing gives $x_2 = 0$. This means that x_2 does not follow from the five clauses, since the five constraints are true and x_2 is false. Hence x_2 does not follow from the constraints.

3. We conclude that the objective function expression follows from the constraints if and only if its value is greater or equal one (1).

Listing 4: The Complete Model implemented in LPL [18]

```
model SAT4 "Satisfibility IV";
  binary variable x1; x2; x3; x4;
  constraint
    R1: x1 or x2 or x3;
    R2: ~x1 or x2 or ~x4;
    R3: ~x1 or x3;
    R4: ~x1 or ~x3  or x4;
    R5: x1 or ~x3;
  minimize obj: x2;
  Writep(x1,x2,x3,x4);
end
```

Solution: Since the optimal value for this model is $x_2 = 1$, we conclude that x_2 follows from the five clauses.

Further Comments: LPL translates this Boolean model into a pure 0-1 linear program as follows:

```
model SAT4;
  binary variable x1; x2; x3; x4;
  constraint
    R1: x1+x2+x3   >= 1;
    R2: -x1+x2-x4  >= 1-2;
    R3: -x1+x3     >= 1-1;
    R4: -x1-x3+x4  >= 1-2;
    R5: x1-x3      >= 1-1;
  minimize obj: x2;
end
```

12 Satisfibility V (Horn Clauses) (sat5)

Problem: Check whether x is implied by the three following (Horn) clauses:

$$x \wedge y \rightarrow z, \quad w \wedge z \wedge y \rightarrow x, \quad x \rightarrow y$$

Modeling Steps

It is a CNF (Conjunctive Normal Form) where the clauses are Horn clauses (clauses that have at most one positive literal, see sat1[12].

Listing 5: The Complete Model implemented in LPL [18]

```
model SAT5 "Satisfibility V (Horn Clauses)";
  binary variable x; y; z; w;
  constraint
    R1: x and y -> z;
    R2: w and z and y -> x;
    R3: x -> y;
  minimize obj: x;
  Writep(x,y,z,w,obj);
end
```

If a model consists of Horn clauses then one can solve the LP relaxation of the corresponding 0-1-ILP to solve the problem, although the solution may not be necessarily have an integer solution. We can nevertheless deduce the result from the optimal value. Hence, no branching is needed.

Solution: In this model, the LP relaxation give an optimal value of zero. Hence, x does not follow from the premises. As one can easily verify, the feasible points are:

```
(x,y,z,w) =
(1,1,1,1)
(1,1,1,0)
(0,1,1,0)
(0,1,0,1)
(0,1,0,0)
(0,0,1,1)
(0,0,1,0)
(0,0,0,1)
(0,0,0,0)
```

This list shows that x is not true under all interpretations.

Further Comments: This model corresponds to the following Prolog program (since all constraints are Horn clauses):

```
z :- x, y          // three rules
x :- w, z, y
y :- x
?- x               // the goal to prove
```

Question (Answer see 12)

[12] https://lpl.matmod.ch/lpl/Solver.jsp?name=/sat1

1. Translate the following Prolog program into LPL and solve it. What is the optimum of the 0-1-ILP, what is the optimum of the LP relaxation?

```
x, y, z
x :- y, z
?- y, z
```

Answer (Question see 12)

1. The model in LPL is as follows and gives the optimal value solution of -1

```
model Horn1;
  binary variable x; y; z;
  constraint
    c1: ~x or ~y or ~z;
    c2: x or ~y or ~z;
  minimize obj: ~y or ~z;
end
```

The LP relaxation is as follows and give an optimum of -1.5:

```
model Horn1;
  variable x[0..1];y[0..1];z[0..1];
  minimize obj:  -z-y;
  constraint c1: -z - y - x >= -2;
             c2: -z - y + x >= -1;
end
```

(The expression `~y or ~z` is a separating cut.)

13 Chemical Synthesis (sat6)

Problem: The following chemical reactions are given:

```
MgO + H2 --> Mg + H2O      (H2O p.e. means water)
C + O2 --> CO2
CO2 + H2O --> H2CO3
```

Show that from a blend of `MgO, H2, O2`, and `C`, the molecule `H2CO3` will be produced.

Modeling Steps

To be able to model this with Boolean logic, some suppositions and simplifications are needed:

1. A Boolean variable for each molecule is introduced with the meaning that the molecule "takes part or does not take part" of a chemical reaction.
2. The molecules which are the inputs in a reaction are linked by a $\wedge$-connector meaning that they must all be present in a chemical reaction at the same time.
3. Also the molecules which are output of a chemical reaction are connected by a $\wedge$ operator meaning that they are all produced at the same time.
4. Input and outputs of a reactions are linked by implication. This expresses the fact that if the inputs are present then the reaction takes place and the outputs must result.

With these assumptions, the "reaction" is modeled as a simple Boolean (satisfibility) problem. It must be shown that:

$$
\begin{array}{ll}
E \rightarrow F & \\
\text{with} \quad E = & (MgO \wedge H_2 \rightarrow Mg \wedge H_2O) \wedge (C \wedge O_2 \rightarrow CO_2) \wedge \\
& (CO_2 \wedge H_2O \rightarrow H_2CO_3) \wedge MgO \wedge H_2 \wedge O_2 \wedge C \\
F = & H_2CO_3
\end{array}
$$

Listing 6: The Complete Model implemented in LPL [18]

```
model Make_H2CO3 "Chemical Synthesis";
  binary variable MgO; H2; O2; C; H2CO3; Mg; CO2; H2O;
  constraint
    A1:  MgO and H2 -> Mg and H2O;
    A2:  C and O2 -> CO2;
    A3:  CO2 and H2O -> H2CO3;
    A4:  MgO and H2 and O2 and C;
  minimize obj: H2CO3;
  Write('Yes, H2CO3 is produced');
  Write('No, H2CO3 is not produced');
end
```

The constraints `A1` to `A3` form the reaction. `A4` means that these four molecules are present. In the implementation, E is the set of constraints (which must hold). The objective function represents F. It is minimized, if the objective function F is zero (false)

then F does not follow from E, that is $E \to F$ is false. On the other side, if F is one (true), then F is the consequence of E and $E \to F$ is true.

Solution: In this model, H_2CO_3 can be (logically) produced from the reaction.

14 Simple expressions (logic0)

Problem: Study how the five following modeling situation can be handled and translated into a mathematical 0-1-ILP formulation (these examples are all from [8] and [7]).

1. We would like to decide whether a particular depot should be built or not. The depot should be built, if a product is delivered to at least one costumer from that depot.
2. If a product is manufactured at all then there is an additional setup (or fixed) cost.
3. If product A is included in the blend then also at least a quantity 4 of product B must be included.
4. If and only if we open a mine then a certain linear condition must hold.
5. Formulate the condition matheamtically: if and only if $3x + 4y \leq 5 \ \vee \ 2x + 3y \geq 4$ holds then also $x + 2y \geq 3 \ \vee \ 2x + 4y \leq 5$ must hold.

Modeling Steps

[Run the model and generate the EQU-file to see how LPL translates the logical statements into Math.]

Listing 7: The Complete Model implemented in LPL [18]

```
model Logic0 "Simple expressions";
  parameter c:=1  "choose example";
  !! Example 1
  variable z [0..10]; binary d;
  constraint C1 if c=1:  z>0 -> d=1;
  constraint C1a if c=1: z>0 <-> d=1;
  !!Example 2
  variable cost;
  constraint C2 if c=2: cost = 5*z + 8*(z>0);
  !! Example 3
  variable zA [0..20]; zB [0..30];
  constraint C3 if c=3: zA>0 -> zB>=4;
  constraint C3a if c=3: zA=4 -> zB=5;
  !! Example 4
  variable v [0..1]; w [0..1];
  constraint C4 if c=4: 2*v+3*w <= 1 <-> d;
  set i:=1..5;
  variable aa{i};
  constraint C4a if c=4: d or and{i} (aa>5);
  !! Example 5
  variable x[0..4]; y [0..5];
  constraint C5 if c=5: 3*x+4*y<=5 or 2*x+3*y>=4 <-> x+2*y>=3 or 2*x+4*
      y<=5;
  !! Example 6
  binary variable x1; x2; x3;
  constraint D1 if c=6: x3 = (x1 or x2);
  constraint D2 if c=6: x3 = (x1 and x2);
  constraint D3 if c=6: x1 = ~x2;
  solve 'nosolve';
  Write('EQU');
end
```

Example 1: The decision to build or not can be modeled by a binary variable δ: $\delta = 1$ mean "to build an depot" and $\delta = 0$ mean's "not to build the depot". Now this depends on whether there is a positive quantity $z > 0$ delivered from this depot. Hence we may have: "if there is a certain (not-zero) quantity delivered from the depot then the depot should be built", that is (let M be an upper bound for z):

$$z > 0 \rightarrow \delta = 1$$

This can be modeled by the following linear inequality:

$$z \leq M\delta$$

Note that M is positive number, so $M\delta$ can only be strictly positive if $\delta = 1$, therefore, if $x > 0$ then δ must be 1). Inversely, if the depot is not built then nothing can be delivered from that depot ($\delta = 0 \rightarrow z \leq 0$). On the other side, if $x = 0$ then δ may be 1 or 0, that is, if nothing is delivered from the depot then the depot may be built or not.

If the condition must be imposed that the depot should *only* be built if some product are delivered from that depot, then equivalence is needed:

$$z > 0 \leftrightarrow \delta = 1$$

This can be modeled by the following two linear inequalities (supposing the lower bound for z is 0 and the minimal strictly positive quantity is ϵ):

$$z \leq M\delta \quad , \quad z \geq \epsilon\delta$$

LPL generates the linear constraint:

```
C1:  + z - 10 d <= 0;
```

For equivalence the LPL code is ($\epsilon = 0.0001$);

```
C1a:  + z - 10 d <= 0;
X14X: + z - 0.0001 d >= 0;
```

Example 2: Suppose that z represents the quantity of a product manufactured at a marginal cost per unit of 5. If the product is manufactured at all there is a additional unique setup cost of 8. Hence, we need to distinguish two cases:

(1) Either nothing is manufactured then $z = 0$, and cost $= 0$.

(2) Or a quantity z is manufactured then $z > 0$, and cost $= 5z + 8$

To model this situation, we introduce a Boolean (indicator) variable δ which is true if "something is manufactured" ($z > 0$), if nothing is manufactured the δ is false. Now we can link the indicator variable with the quantity z by the implication (meaning "if the quantity z is strictly larger than 0 then something is manufactured"):

$$z > 0 \rightarrow \delta \qquad \text{or} \qquad \overline{\delta} \rightarrow z \leq 0$$

This is modeled by the linear constraint (M being an upper bound for z): $z \leq M\delta$ and the cost function is formulated as: cost $= 5z + 8\delta$

To verify the resulting model, we check for the values of δ:

(1) if $\delta = 1$ (true) then we have: $z \leq U(z)$, cost $= 5z + 8$

(2) If $\delta = 0$ (false) then we have: $z \leq 0$, cost $= 0$

In LPL one may formulated the cost function directly – without needing to introduce explicitly a binary variable (although this is possible too) – as follows:

```
constraint C2: cost = 5*z + 8*(z>0);
```

It generates automatically the two linear constraints (where `X13X` is the additional binary variable):

```
C2:  - 8 X13X - 5 z + cost = 0;
X14X:  + z - 10 X13X <= 0;
```

Example 3: If z_A and z_B are the quantities in a blend of the two products A and B, then the logical formulation is: $z_A > 0 \rightarrow z_B \geq b$.

To model this situation, let's introduce an indicator variable δ with the meaning "A is included in the blend" that is: $z_A > 0 \rightarrow \delta$. And the requirement "if product A is included in the blend, then also at least b of product B must be included" can be modeled as: $\delta \rightarrow z_B \geq b$. The two constraints can be translated to mathematical inequalities as follows, where $U(z_A)$ is an upper bound for z_A and $L(z_B)$ is a lower bound for z_B:

$$\begin{array}{rl} z_A \leq & U(z_A) \cdot \delta \\ z_B \geq & L(z_B + b) \cdot \delta \end{array}$$

To verify the resulting model, we check for the values of δ:
(1) If $\delta = 1$ (true) then $z_A \leq U(z_A) \quad , \quad z_B \geq b$ (supposing $L(z_B) = 0$
(2) If $\delta = 0$ (false) then $z_A \leq 0 \quad , \quad z_B \geq 0$

Example 4: A constraint A must hold if and only if the indicator variable is true. For example: $2v + 3w \leq 1 \leftrightarrow \delta$. This is equivalent to:

$$2v + 3w \leq 1 \rightarrow \delta \quad \text{and} \quad \delta \rightarrow 2v + 3w \leq 1$$

Supposing $L(v) = L(w) = 0$, $U(v) = U(w) = 1$, $\epsilon = 0.001$ the resulting linear constraints are:

$$2v + 3w + 4\delta \leq 5 \quad \text{and} \quad 2v + 3w + 1.001\delta \geq 1.001$$

Example 5: A general version of this exercise is explained in [7] Chap 3.5. We first present the solution given by Williams in [7]. Let's start with the constraint and repeat it:

$$\sum_j a_{1,j}x_j \leq b_1 \ \vee \ \sum_j a_{2,j}x_j \geq b_2 \quad \leftrightarrow \quad \sum_j a_{3,j}x_j \geq b_3 \ \vee \ \sum_j a_{4,j}x_j \leq b_4$$

Let the four top inequalities be P_i with $i \in \{1, \ldots, 4\}$, as well as their lower and upper bound m_i, M_i with $i \in \{1, \ldots, 4\}$. Then we have:

$P_1 \ \vee \ P_2 \quad \leftrightarrow \quad P_3 \ \vee \ P_4$ which gives

$((P_1 \ \vee \ P_2) \ \vee \ (\overline{P_3 \ \vee \ P_4})) \quad \wedge \quad ((\overline{P_1 \ \vee \ P_2}) \ \vee \ (P_3 \ \vee \ P_4))$ which gives

$(P_1 \ \vee \ P_2 \ \vee \ (\overline{P_3} \ \wedge \ \overline{P_4})) \quad \wedge \quad ((\overline{P_1} \ \wedge \ \overline{P_2}) \ \vee \ P_3 \ \vee \ P_4)$

Let's the indicator variables be δ_i with $i \in \{1, \ldots, 4\}$, δ_5, and δ_6 as follows:

$$\begin{array}{rcll} \delta_i = 1 & \rightarrow & P_i & \text{for } i \in \{1, \ldots, 4\} \\ \delta_5 = 1 & \rightarrow & \overline{P_1} & \\ \delta_5 = 1 & \rightarrow & \overline{P_2} & \\ \delta_6 = 1 & \rightarrow & \overline{P_3} & \\ \delta_6 = 1 & \rightarrow & \overline{P_4} & \end{array}$$

Hence, the top constraint reduces to:

$$(\delta_1 \ \vee \ \delta_2 \ \vee \ \delta_6) \ \ \wedge \ \ (\delta_3 \ \vee \ \delta_4 \ \vee \ \delta_5)$$

The 8 indicator variable definitions together with the Boolean constraint generated the following 10 linear inequalities:

$$\begin{array}{lcl}
\sum_j a_{1,j}x_j + (M_1 - b_1)\delta_1 & \leq & M_1 \\
\sum_j a_{2,j}x_j + (m_2 - b_2)\delta_2 & \leq & m_2 \\
\sum_j a_{3,j}x_j + (m_3 - b_3)\delta_3 & \leq & m_3 \\
\sum_j a_{4,j}x_j + (M_4 - b_4)\delta_4 & \leq & M_4 \\
\sum_j a_{1,j}x_j - (m_1 - b_1 + \epsilon)(1 - \delta_5) & \geq & b_1 + \epsilon \\
\sum_j a_{2,j}x_j - (M_2 - b_2 + \epsilon)(1 - \delta_5) & \geq & b_2 - \epsilon \\
\sum_j a_{3,j}x_j - (M_3 - b_3 + \epsilon)(1 - \delta_6) & \geq & b_3 - \epsilon \\
\sum_j a_{4,j}x_j - (m_4 - b_4 + \epsilon)(1 - \delta_6) & \geq & b_4 + \epsilon \\
\delta_1 + \delta_2 + \delta_6 & \geq & 1 \\
\delta_3 + \delta_4 + \delta_5 & \geq & 1
\end{array}$$

This is the translation given in [7]. In LPL, a somewhat different translation approach is chosen: *before* distributing and eliminating the equivalence ($\leftrightarrow$), the inequalities are defined by indicator variables:

$$\delta_i = 1 \ \rightarrow \ P_i \quad , \text{ for all } i \in \{1, \ldots, 4\}$$

Then the top constraint becomes:

$$\begin{array}{ll}
\delta_1 \ \vee \ \delta_2 \ \leftrightarrow \ \delta_3 \ \vee \ \delta_4 & \text{which gives} \\
((\delta_1 \ \vee \ \delta_2) \ \vee \ (\overline{\delta_3 \ \vee \ \delta_4})) \ \ \wedge \ \ ((\overline{\delta_1 \ \vee \ \delta_2}) \ \vee \ (\delta_3 \ \vee \ \delta_4)) & \text{which gives} \\
(\delta_1 \ \vee \ \delta_2 \ \vee \ (\overline{\delta_3} \ \wedge \ \overline{\delta_4})) \ \ \wedge \ \ ((\overline{\delta_1} \ \wedge \ \overline{\delta_2}) \ \vee \ \delta_3 \ \vee \ \delta_4) & \text{which gives} \\
(\delta_1 \vee \delta_2 \vee \overline{\delta_3}) \ \wedge \ (\delta_1 \vee \delta_2 \vee \overline{\delta_4}) \ \wedge \ (\overline{\delta_1} \vee \delta_3 \vee \delta_4) \ \wedge \ (\overline{\delta_2} \vee \delta_3 \vee \delta_4) &
\end{array}$$

The last line is an expression of four clauses, for which one can directly derive the four linear inequalities. Finaly, this translation generates only 4 additional binaries and totally 8 linear constraints, in contrast with the translation given in [7] which generates 6 binaries and 10 linear constraints.

The code of LPL is as follows:

```
variable x[0..4]; y [0..5];
constraint C5: 3*x+4*y<=5 or 2*x+3*y>=4 <-> x+2*y>=3 or 2*x+4*y<=5;
```

LPL generates the following 8 inequalities (together with the 4 binary variables `X50X`, `X52X`, `X54X`, `X56X` :

```
C5:  + X56X + X54X - X50X >= 0;
X51X:  + 4 y + 3 x + 27 X50X <= 32;
X53X:  + 3 y + 2 x - 4 X52X >= 0;
X55X:  + 2 y + x - 3 X54X >= 0;
X57X:  + 4 y + 2 x + 23 X56X <= 28;
X58X:  + X52X + X50X - X54X >= 0;
X59X:  + X56X + X54X - X52X >= 0;
X60X:  + X52X + X50X - X56X >= 0;
```

Question (Answer see 14)

1. Check the output of LPL for example 5.

Answer (Question see 14)

1. Taken the first definition: $\delta_1 = 1 \rightarrow P_1$, this gives

$$3x + 4y + (M_1 - b_1)\delta_1 \leq M_1$$

Since $b_1 = 5$ and $M_1 = 32$, this gives :

$$3x + 4y + 27\delta_1 \leq 32$$

corresponding to constraint `X51X`.

15 Boolean Expressions (logic1)

Problem: Study how the four following Boolean expressions A, B, C, D are translated into a mathematical 0-1-ILP formulation:

$$\begin{array}{ll} A: & x \wedge y \vee z \ \dot{\vee} \ w \\ B: & (x \wedge y \,\dot{\vee}\, z) \vee w \\ C: & \bigvee_{i \in I} (q_i \wedge p_i) \\ D: & \bigwedge_{i \in I} (q_i \vee p_i) \\ \text{with} & I = \{1, \ldots, n\}\ , n > 0 \end{array}$$

Modeling Steps

Expression A: $((x \wedge y) \vee z) \,\dot{\vee}\, w$

1. The first step is to apply a definition of the logical operator $\dot{\vee}$:

   $$(x \,\dot{\vee}\, y) \equiv ((x \vee y) \wedge (\neg x \vee \neg y))$$

   this gives:

   $$((x \wedge y \vee z) \vee w) \quad \wedge \quad ((\neg(x \wedge y \vee z) \vee \neg w))$$

2. The first part gives $((x \vee z) \wedge (y \vee z) \vee w)$ which is:

   $$(x \vee z \vee w) \wedge (y \vee z \vee w)$$

3. The second part gives $((\neg(x \vee y) \wedge \neg z) \vee \neg w$ which is:

   $$(\neg x \vee \neg y \vee \neg w) \wedge (\neg z \vee \neg w)$$

4. Putting the two parts together gives four clauses for A:

   $$(x \vee z \vee w) \wedge (y \vee z \vee w) \wedge (\neg x \vee \neg y \vee \neg w) \wedge (\neg z \vee \neg w)$$

   This is the CNF (conjunctive normal form) of the constraint A. Each single clause now can be formulated as a linear constraints as follows:

   $$\begin{array}{ll} x + z + w & \geq 1 \\ y + z + w & \geq 1 \\ 1 - x + 1 - y + 1 - w & \geq 1 \\ 1 - z + 1 - w & \geq 1 \end{array}$$

Expression B: $((x \wedge y) \,\dot{\vee}\, z) \vee w$

1. Again, we apply the definition of $\dot{\vee}$. Hence the expression becomes:

   $$(((x \wedge y) \vee z) \wedge ((\neg(x \wedge y) \vee \neg z)) \vee w$$

2. Redistributing $\vee$ on the left and inverse moving $\neg$ gives:

$$((x \vee z) \wedge (y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)) \vee w$$

3. Redistribute $\vee$ an second time gives the following CNF:

$$(x \vee z \vee w) \wedge (y \vee z \vee w) \wedge (\neg x \vee \neg y \vee \neg z \vee w)$$

4. Again each single clause can be reformulated as a linear constraint:

$$\begin{array}{ll} x + z + w & \geq 1 \\ y + z + w & \geq 1 \\ 1 - x + 1 - y + 1 - z + w & \geq 1 \end{array}$$

Expression C: $\bigvee_{i \in I}(p_i \wedge q_i)$

The operator $\bigvee$ is the indexed operator for "or". The constraint is in fact (note that we have $I = \{1, \ldots, n\}$, $n > 0$):

$$\underbrace{(p_1 \wedge q_1) \vee (p_2 \wedge q_2) \vee \ldots \vee (p_n \wedge q_n)}_{\text{n times}}$$

1. Introduce a new binary variable r_i with $i \in I$ and add the constraints[13]

$$r_i \rightarrow (p_i \wedge q_i) \quad , \text{ forall } i \in I$$

2. Substitute $p_i \wedge q_i$ by r_i in constraint C. This gives the system of the two constraint types:

$$\begin{array}{l} \bigvee_{i \in I} r_i \\ r_i \rightarrow \quad (p_i \wedge q_i) \quad , \text{ forall } i \in I \end{array}$$

3. Replacing implication and distribute gives:

$$\begin{array}{l} \bigvee_{i \in I} r_i \\ (\neg r_i \wedge p_i) \vee (\neg r_i \wedge q_i) \quad , \text{ forall } i \in I \end{array}$$

4. This can be translated straight away into linear inequalities as:

$$\begin{array}{lll} \sum_{i \in I} r_i & \geq 1 & \\ 1 - r_i + p_i & \geq 1 & \text{forall } i \in I \\ 1 - r_i + q_i & \geq 1 & \text{forall } i \in I \end{array}$$

[13] Note that in many cases an implication is all what is needed, however there are situations where an equivalence is imperatively required. LPL used implication whenever possible. If equivalence is needed the modeler must manually impose it.

An alternative of constraint C is not to introduce a new variable r_i, but to generate directly a CNF. However, the CNF would be exponential large in the size of set I. Hence, from a practical point of view, it is necessary to introduce a new binary variable r_i.

Expression D: $\bigwedge_{i \in I}(p_i \vee q_i)$

The operator $\bigwedge$ is the indexed operator for "and". The constraint is in fact (note that we have $I = \{1, \ldots, n\}$, $n > 0$):

$$\underbrace{(p_1 \vee q_1) \wedge (p_2 \vee q_2) \wedge \ldots \wedge (p_n \vee q_n)}_{\text{n times}}$$

This constraint is already in the conjunctive normal form. Hence, the translation into linear inequalities is straightforward as:

$$p_i + q_i \geq 1 \quad , \text{ forall } i \in I$$

Question (Answer see 15)

1. Start LPL and look what it generates by looking at the EQU-file and the LPY-file.

Answer (Question see 15)

1. the first constraint A generates the following linear constraints:

```
+x +z +w >= 1;
+w +z +y >= 1;
-x -y -w >= -2;
-w -z >= -1;
```

The second constraint B produces the following linear constraints:

```
x >= 1-w -z
y >= 1-w -z
1-x + 1-y + 1-z >= 1-w
```

The third constraint C produces the following linear constraints:

```
C:  + X50[1] + X50[2] + X50[3] + X50[4] + X50[5]
    + X50[6] + X50[7] + X50[8] + X50[9] >= 1;
X51[1]:  + q[1] - X50[1] >= 0;
X51[2]:  + q[2] - X50[2] >= 0;
X51[3]:  + q[3] - X50[3] >= 0;
X51[4]:  + q[4] - X50[4] >= 0;
X51[5]:  + q[5] - X50[5] >= 0;
X51[6]:  + q[6] - X50[6] >= 0;
X51[7]:  + q[7] - X50[7] >= 0;
X51[8]:  + q[8] - X50[8] >= 0;
X51[9]:  + q[9] - X50[9] >= 0;
X52[1]:  + p[1] - X50[1] >= 0;
X52[2]:  + p[2] - X50[2] >= 0;
X52[3]:  + p[3] - X50[3] >= 0;
X52[4]:  + p[4] - X50[4] >= 0;
```

```
X52[5]:  + p[5] - X50[5] >= 0;
X52[6]:  + p[6] - X50[6] >= 0;
X52[7]:  + p[7] - X50[7] >= 0;
X52[8]:  + p[8] - X50[8] >= 0;
X52[9]:  + p[9] - X50[9] >= 0;
```

where `X50{i}` are the new binary variables introduced.

The fourth constraint D produces the following linear constraints:

```
D[1]:  + p[1] + q[1] >= 1;
D[2]:  + p[2] + q[2] >= 1;
D[3]:  + p[3] + q[3] >= 1;
D[4]:  + p[4] + q[4] >= 1;
D[5]:  + p[5] + q[5] >= 1;
D[6]:  + p[6] + q[6] >= 1;
D[7]:  + p[7] + q[7] >= 1;
D[8]:  + p[8] + q[8] >= 1;
D[9]:  + p[9] + q[9] >= 1;
```

16 Math-Logic Expression III (logic2)

Problem: Model the following six Boolean constraints as linear inequalities (where $i \in I$:

$$\begin{array}{lll} C1: & d \leftrightarrow & (x \wedge y) \\ C2: & d \leftrightarrow & \bigwedge_{i \in I} z_i \\ D1: & d \leftrightarrow & (x \vee y) \\ D2: & d \leftrightarrow & \bigvee_{i \in I} z_i \\ E1: & d \leftrightarrow & (x \veebar y) \\ F1: & d \leftrightarrow & (x \leftrightarrow y) \end{array}$$

Modeling Steps

Listing 8: The Complete Model implemented in LPL [18]

```
model Logic2 "Math-Logic Expression III";
  parameter c:=1 "choose constraint";
  set i:=[1..9];
  binary variable d; x; y; z{i};
  constraint
    C1 if c=1: d <-> x and y;
    C2 if c=2: d <-> and{i} z;
    D1 if c=3: d <-> x or y;
    D2 if c=4: d <-> or{i} z;
    E1 if c=5: d <-> (x xor y);
    F1 if c=6: d <-> (x <-> y);
  solve 'nosolve';
  Write('EQU');
end
```

Constraint C1: Graphically, the constraint can be represented by a unit cube, where each of the 8 corners represent a true/false combination of the three variables (x, y, d) (see Figure 5). The expression $C1$ is true for the four black points and false otherwise. The black points can be separated from the others by two parallel planes defined by the points $\{(1,1,1),(0.5,0,1),(0,0,0)\}$ and $\{(0,1,0),(0.5,1,1),(0,0,1)\}$ (see Figure 5, left side)). The two planes can be formulated mathematically as: $x+y-2d=0$ and $x+y-2d=1$. Hence, the expression $C1$ could be expressed mathematically by the intersection of the two half-spaces :

$$0 \leq x+y-2d \leq 1 \qquad \text{(C1a)}$$

For a different formulation, consider the three planes defined by three points each as follows: $\{(010),(111),(001)\}$, $\{(111),(101),(000)\}$, and $\{(110),(000),(001)\}$ (see Figure 5, right side). These three planes are formulated mathematically as: $x+y-d=1$, $x=d$, $y=d$ Hence, the constraint $C1$ could also be expressed mathematically by the intersection of the three half-spaces as follows :

$$\begin{array}{rl} x+y-d \leq 1 & \\ x \geq d & \text{(C1b)} \\ y \geq d & \end{array} \qquad (1)$$

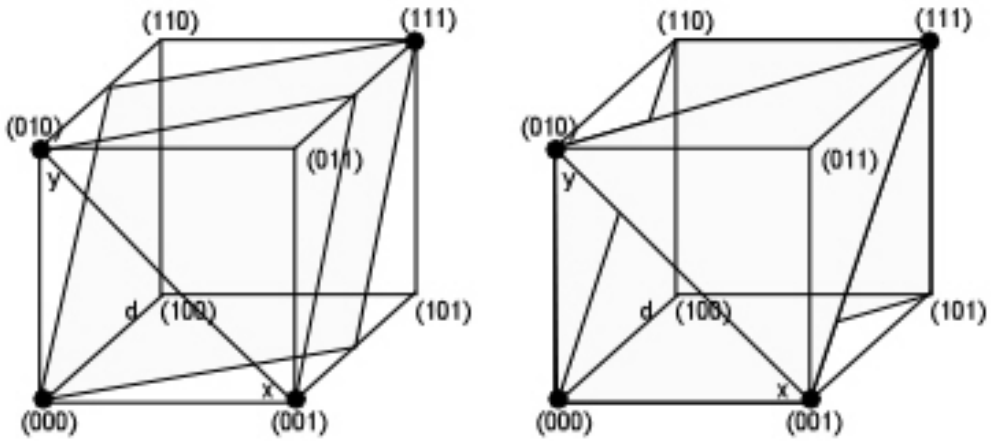

Figure 5: The Unit Cube with the 4 Feasible Points of $C1$

Both mathematical formulations are correct. However, the second one with three linear inequalities is "better" because its LP-relaxation is tighter.[14] LPL automatically generates the tighter formulation:

```
C1:    + x - d >= 0;
X43X:  - y - x + d >= -1;
X44X:  + y - d >= 0;
```

The transformation to a CNF (conjunctive normal form) is as follows:

$$d \leftrightarrow (x \wedge y) \quad \equiv \quad (d \vee (\overline{x \wedge y})) \ \wedge \ (\overline{d} \vee (x \wedge y)) \quad \equiv$$
$$(d \vee \overline{x} \vee \overline{y}) \ \wedge \ (\overline{d} \vee x) \ \wedge \ (\overline{d} \vee y)$$

Constraint C2: In the same way as $C1$, this constraint $C2$ can be formulated by two parallel planes as follows (n is the cardinality of the set i):

$$0 \leq \sum_{i=1}^{n} z_i - nd \leq n - 1$$

A tighter formulation is given by:

$$\begin{aligned} -\sum_{i=1}^{n} x_i + d &\geq 1 - n \\ x_i &\geq d \qquad \text{for all } i \in \{1, \ldots, n\} \end{aligned}$$

Constraint D1: Graphically, the constraint is true for the black points in the unit cube and false otherwise (see in Figure 6). These black points can be separated from the others by two parallel planes containing the points $\{(1,1,0),(0,0,0.5),(1,0,1)\}$ and $\{(1,1,1),(0.5,0,1),(0,0,0)\}$. The two planes are: $x + y - 2d = 0$ and $x + y - 2d = -1$. The intersection of the two half-spaces represents $D1$, it is:

$$-1 \leq x + y - 2d \leq 0$$

[14] The LP-relaxation is obtained by replacing $x, y, d \in \{0, 1\}$ with $0 \leq x, y, d \leq 1$. "tighter" means that $C1b \subset C1a$ and this is "better" because most solver start with a LP-relaxation as initial solution. For a more profound theory see [10] or [14].

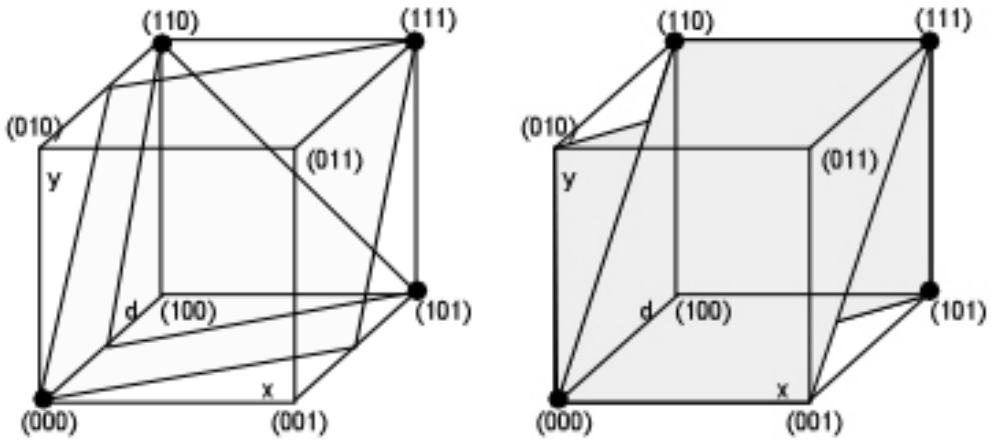

Figure 6: The Unit Cube with the 4 Feasible Points of $D1$

A tighter formulation is as follows:

$$\begin{array}{ll} x+y & \geq d \\ x & \leq d \\ y & \leq d \end{array}$$

The three corresponding planes are represented by the following coordinates in the (x,y,d) cube: $\{(110, 000, 101)\}$, $\{(111), (101), (000), (010)\}$, and $\{(110), (000), (001), (111)\}$ shown in Figure 6 (right side). LPL also generates the tighter form.

Constraint D2: Like $D1$ the constraint $D2$ corresponds to the two reformulations as follows:

$$-(n-1) \leq \sum_{i=1}^{n} z_i \leq nd$$

And the tighter form:

$$\begin{array}{ll} \sum_{i=1}^{n} z_i & \geq d \\ -z_i + d & \geq 0 \quad \text{for all } i \in \{1, \ldots, n\} \end{array}$$

Constraint E1: This constraint can be formulated in a tight form as:

$$\begin{array}{ll} x+y-d & \geq 0 \\ x+y+d & \leq 2 \\ x-y+d & \geq 0 \\ -x+y+d & \geq 0 \end{array}$$

Graphically the solution is given by the four corner points in Figure 7 (left side). Each half-space excludes a single corner of the unit cube.

The four corresponding planes are defined by three points each as: $\{(000), (011), (110)\}$, $\{(110), (011), (101)\}$, $\{(000), (011), (101)\}$, and $\{(000), (101), (110)\}$.

Constraint F1: This constraint can be formulated in the thight form as:

$$\begin{array}{ll} -x+y-d & \geq -1 \\ x-y-d & \leq -1 \\ x+y+d & \geq 1 \\ -x-y+d & \geq -1 \end{array}$$

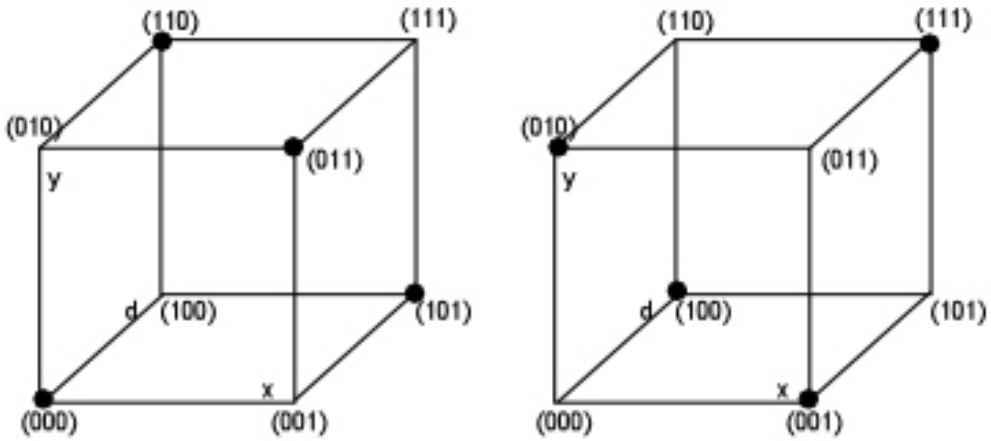

Figure 7: Four corner points

Graphically the solution is given by the four corner points in Figure 7 (right side). The four planes can be constructed in the same way as in the previous example.

17 Indexed Boolean Expression (logic3)

Problem: Given 9 products $I = \{1, \ldots, 9\}$, the following logical condition should be modeled: "If 3 or more out of the products $\{1, 2, 3, 4, 5\}$ are made, or less than 4 of products $\{3, 4, 5, 6, 8, 9\}$ are made then at least 2 of products $\{7, 8, 9\}$ must be made unless none of products $\{5, 6, 7\}$ are made." This example has been published in [12].

Modeling Steps

To formulate this constraint, a Boolean variable P_i with $i \in I$ is introduced with the meaning "product i is made if $P_i = 1$ (that is, if P_i is true) otherwise $P_i = 0$ (or P_i is false)". The logical statement can then be reformulated as follows:

"At least 3 of P_i with $i \in I_1 = \{1, 2, 3, 4, 5\}$ are true or at most 3 of P_i with $i \in I_2 = \{3, 4, 5, 6, 8, 9\}$ are true and not none of P_i with $i \in I_3 = \{5, 6, 7\}$ is true implies that at least 2 of P_i with $i \in I_4 = \{7, 8, 9\}$ are true". Or formally:

$$\big((\text{atleast}(3)_{i \in I_1}\ P_i \vee \text{atmost}(3)_{i \in I_2}\ P_i)\ \wedge\ \neg(\underset{i \in I_3}{\textbf{nor}}\ P_i)\big)\ \ \rightarrow\ \ \text{atleast}(2)_{i \in I_4}\ P_i$$

Reducing implication and pushing not, gives:

$$\neg(\text{atleast}(3)_{i \in I_1}\ P_i \wedge \neg\text{atmost}(3)_{i \in I_2}\ P_i)\ \vee\ \underset{i \in I_3}{\textbf{nor}}\ P_i\ \vee\ \ \text{atleast}(2)_{i \in I_4}\ P_i$$

Applying some other rules for the indexed operators gives:

$$(\text{atmost}(2)_{i \in I_1}\ P_i \wedge \text{atleast}(4)_{i \in I_2}\ P_i)\ \vee\ \text{atleast}(0)_{i \in I_3}\ \neg P_i\ \vee\ \ \text{atleast}(2)_{i \in I_4}\ P_i$$

or:

$$\big(\sum_{i \in I_1} P_i \leq 2\ \wedge\ \sum_{i \in I_2} P_i \geq 4\big)\ \vee\ \bigwedge_{i \in I_3} \neg P_i\ \vee\ \sum_{i \in I_4} P_i \geq 2$$

Let us introduce three binary variables:

$$\begin{aligned}
\delta_1 = 1\ &\rightarrow\ \sum_{i \in I_1} P_i \leq 2 \\
\delta_1 = 1\ &\rightarrow\ \sum_{i \in I_2} P_i \geq 4 \\
\delta_2 = 1\ &\rightarrow\ \bigwedge_{i \in I_3} \neg P_i \\
\delta_3 = 1\ &\rightarrow\ \sum_{i \in I_4} P_i \geq 2
\end{aligned}$$

Hence, the whole constraint can be reduced to 5 linear inequalities as follows:

$$
\begin{aligned}
&\sum_{i \in I_1} P_i + 3\delta_1 \leq 5 \\
&\sum_{i \in I_2} P_i - 4\delta_1 \geq 0 \\
&P_i + \delta_2 \leq 1 \qquad \text{for all } i \in I_3 \\
&\sum_{i \in I_4} P_i - 2\delta_3 \geq 0 \\
&\delta_1 + \delta_2 + \delta_3 \geq 1
\end{aligned}
$$

Listing 9: The Complete Model implemented in LPL [18]

```
model Logic3 "Indexed Boolean Expression";
  set i   := [1..9]          "A set of products";
    i1{i} := [1 2 3 4 5]     "Sublist: products 1 to 5";
    i2{i} := [3 4 5 6 8 9];
    i3{i} := [5 6 7];
    i4{i} := [7 8 9];
  binary variable P{i} "Product i is produced?";
  constraint Cond:
     (atleast(3){i1} P[i1] or atmost(3){i2} P[i2])
     and ~(nor{i3} P[i3])  ->  atleast(2){i4} P[i4];
  maximize obj: sum{i} P;
  Writep(P);
end
```

Further Comments: LPL generates the following constraints, which corresponds exactly the solution above. It is also identical with the solution given in [12]:

```
Cond: X50X + X48X + X45X >= 1;
X46X: P[1] + P[2] + P[3] + P[4] + P[5] + 3 X45X <= 5;
X47X: P[3] + P[4] + P[5] + P[6] + P[8] + P[9] - 4 X45X >= 0;
X49X[5]: - X48X - P[5] >= -1;
X49X[6]: - X48X - P[6] >= -1;
X49X[7]: - X48X - P[7] >= -1;
X51X:  P[7] + P[8] + P[9] - 2 X50X >= 0;
```

Question (Answer see 17)

1. For some reasons, the company wants only produce two products. Which product is certainly not in that list.

Answer (Question see 17)

1. If only two product are produced then certainly product 5 and 6 cannot be part of them. Why? Because this makes the premise of the implication true and the consequence false, since then less than two of set I_4 are part and not none of set I_3 are in the part. One can check this by adding the two constraints (the model becomes infeasible – saying that there is no way to make this true):

```
constraint D: sum{i} P <= 2;
constraint E: P[5] or P[6];
```

18 Disjunctive Constraint (logic4)

Problem: Model the disjunctive grey space defined in the Figure 8

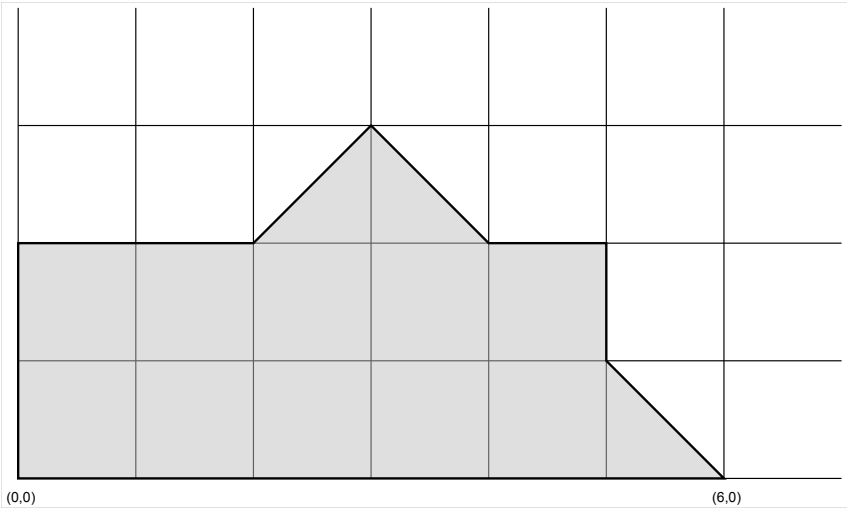

Figure 8: A Disjunctive Space

Modeling Steps

Informally, a set is concave, if some points on a straight segment with its endpoints in the set are outside the set. For a definition see Wikipedia. The grey set (space) in Figure 8 is concave (or disjunctive) and we know that this space cannot be modeled using a single LP – that is using linear inequalities, since the feasible space of an LP is always convex. A set of constraints can be interpreted as the *intersection* of all constraints in the set. In the Figure 8 the grey surface can be interpreted as the *union* of two convex figures: (1) a rectangle with the four corners $(0,0)$, $(0,2)$, $(5,2)$, and $(5,0)$. (2) a triangle with the corners $(0,0)$, $(3,3)$, and $(6,0)$.

1. We introduce two variables $x \geq 0$ (horizontal extension) and $y \geq 0$ (vertical extension).

2. The rectangle space R can be modeled as a convex space (intersection of two linear constraints):
$$R: \; x \leq 6 \; \wedge \; y \leq 2$$

3. The triangle space T can be modeled as a convex space:
$$T: \; y \leq x \; \wedge \; y \leq 6 - x$$

4. The union of R and T is therefore:
$$(x \leq 6 \; \wedge \; y \leq 2) \quad \vee \quad (y \leq x \; \wedge \; y \leq 6 - x)$$

Listing 10: The Complete Model implemented in LPL [18]

```
model Logic4 "Disjunctive Constraint";
  variable x[0..10]; y[0..20];
  constraint Grey: (x<=5 and y<=2) or (y<=x and y<=6-x);
  maximize obj: x+y;
  Write('Optimum is: (%d,%d)\n' ,x,y);
end
```

Solution: Depending on the objective function, various optimal points are found and they are easy to verify :

1. With the function max $x + y$, the optimum is $(5, 2)$.
2. With the function max y, the optimum is $(3, 3)$.
3. With the function max x, the optimum is $(6, 0)$.
4. With the function max $y - x$, the optimum is $(0, 2)$.

LPL transforms the disjunctive constraint into the following 5 linear constraints, adding two binary variables `X40` and `X43`:

```
max:  + y + x;
Grey:  + X43X + X40X >= 1;
X41X:  + x + 5 X40X <= 10;
X42X:  + y + 18 X40X <= 20;
X44X:  - x + y + 20 X43X <= 20;
X45X:  + x + y + 24 X43X <= 30;
```

19 Two Liquid Containers Problem (logic5)

Problem: A company produces two liquids A and B at unknown quantities x and y. The liquids are stored in two containers of capacity a and b. The two liquids can be stored in both containers, but cannot be mixed in the same container. Normally, liquid A is stored in the first and liquid B in the second container. The company could therefore produce the maximum quantity a of liquid A and a maximum quantity b of liquid B. But in some situations, it may be more advantageous to produce more liquid of the same type and none of the other. In this case, the company may produce one liquid at a maximum quantity of $a + b$ and to store it in both containers. Hence, either the company must produce both liquids at quantities less than a and b respectively, or it must produce only one liquid with quantity less than $a + b$ (see Figure 9. Formulate the capacity constraint (This problem is from [3]).

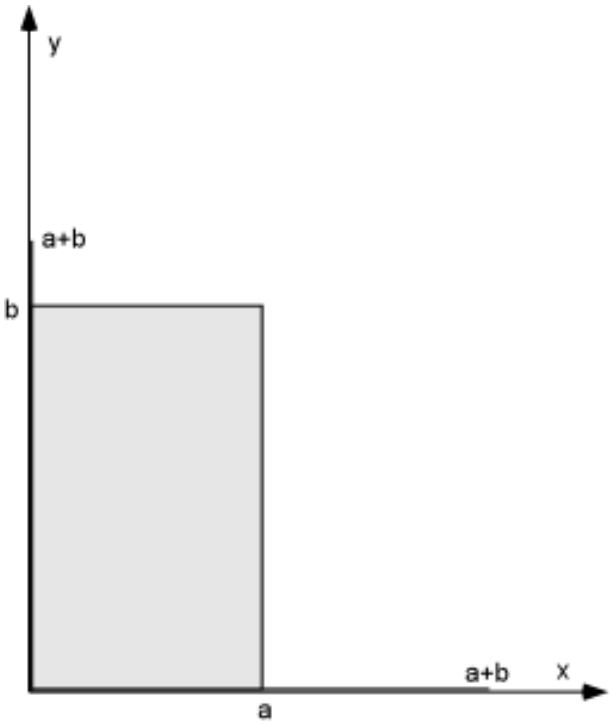

Figure 9: The Feasible Space of the Liquid Container Problem

Modeling Steps

This is a typical very simple problem of a *disjunctive set*.

1. We introduce two variables for the quantities of the the two liquids: x and y.
2. Three cases can be distinguished: Either we produce only the first liquid A and store it in both containers, or we produce only the second liquid B and store it also in both containers or we produce both then we have to store the first in one and the second in the second container separately. The feasible space is shown in Figure 9.

First case:	$y = 0$ and $x \leq a + b$	segment $(0, 0)$ to $(a + b, 0)$
Second case:	$x = 0$ and $y \leq a + b$	segment $(0, 0)$ to $(0, a + b)$
Third case:	$x \leq a$ and $y \leq b$	rectangle $(0, 0)$ to (a, b)

3. At least one of the three cases must be true, hence:

$$(y = 0 \wedge x \leq a + b) \ \vee \ (x = 0 \wedge y \leq a + b) \ \vee \ (x \leq a \wedge y \leq b)$$

4. It is easy to verify that this expression corresponds also to (we suppose that $x, y \geq 0$):

$$(x > a \rightarrow y \leq 0) \wedge (y > b \rightarrow x \leq 0)$$

Listing 11: The Complete Model implemented in LPL [18]

```
model Logic5 "Two Liquid Containers Problem";
  parameter a:=10; b:=12;
  variable x [0..a+b]; y [0..a+b];
  constraint capa: (x>a -> y<=0) and (y>b -> x<=0);
  --constraint capa: y=0 and x<=a+b or x=0 and y<=a+b or x<=a and y<=b;
  maximize obj:x+y;
  Writep(x,y);
end
```

Solution: The logical capacity constraint can be translated into the following linear constraints by adding two binary variables p and q:

$$\begin{array}{lcl} x - a & \leq & b \cdot p \\ y - b & \leq & a \cdot q \\ x & \leq & (a + b) \cdot (1 - q) \\ y & \leq & (a + b) \cdot (1 - p) \\ p, q \in \{0, 1\} & , & 0 \leq x, y \leq a + b \end{array}$$

Further Comments: To translate the logical constraint into the linear inequalities, do the following:

1. Start with the expression:

$$(x > a \rightarrow y \leq 0) \ \wedge \ (y > b \rightarrow x \leq 0)$$

2. Replacing the implication gives:

$$(x \leq a \vee y \leq 0) \ \wedge \ (y \leq b \vee x \leq 0)$$

3. Add four binary variables and substitute the four subexpressions[15]:

$$\begin{array}{lcl} p_1 = 1 & \rightarrow & x \leq a \\ p_2 = 1 & \rightarrow & y \leq 0 \\ q_1 = 1 & \rightarrow & y \leq b \\ q_2 = 1 & \rightarrow & x \leq 0 \\ p_1, p_2, q_1, q_2 \in \{0, 1\} & & \end{array}$$

4. The expression becomes :

$$(p_1 \vee p_2) \ \wedge \ (q_1 \vee q_2)$$

[15] In this case we substitute the expressions by an implication: we force the subexpression to be true if the binary variable is true, if the binary variable is false than nothing is forced. If we want also force the expression to be false if the binary is false then we must replace the implication by equivalence.

5. which is :

$$p_1 + p_2 \geq 1 \quad , \quad q_1 + q_2 \geq 1$$

6. Translating the four definitions into linear constraints gives (where $U(\alpha)$ is an upper bound for α):

$$\begin{array}{lcl} x - a & \leq & U(x-a) \cdot (1 - p_1) \\ y & \leq & U(y) \cdot (1 - p_2) \\ y - b & \leq & U(y-b) \cdot (1 - q_1) \\ x & \leq & U(x) \cdot (1 - q_2) \end{array}$$

7. Since $U(x) = U(y) = a + b$ we have the following model:

$$\begin{array}{lcl} p_1 + p_2 & \geq & 1 \\ q_1 + q_2 & \geq & 1 \\ x - a & \leq & b \cdot (1 - p_1) \\ y & \leq & (a+b) \cdot (1 - p_2) \\ y - b & \leq & a \cdot (1 - q_1) \\ x & \leq & (a+b) \cdot (1 - q_2) \end{array}$$

8. A further reduction can be obtained by observing that p_2 can be replaced by $1 - p_1$, and q_2 by $1 - q_1$ and the solution follows.

LPL translates the code

```
parameter a:=10; b:=12;
variable x [0..a+b]; y [0..a+b];
constraint capa: (x>a -> y<=0) and (y>b -> x<=0);
```

into the following linear constraints:

```
  capa:  + x - 12*X39X <= 10;
  X42X:  + y - 10*X41X <= 12;
  X43X:  + y + 22*X39X <= 22;
  X44X:  + x + 22*X41X <= 22;
  — bounds —
  x <= 22;
  y <= 22;
  X39X <= 1 , binary;
  X41X <= 1 , binary;
```

This is exactly what we have obtained above. Note, however, that the linear inequalities are logically not equivalent with the logical constraint, but they are *implied*.

Question (Answer see 19)

1. Verify that the two logical expressions above are equivalent.
2. The logical constraint is translated into a mixed integer formulation as given above. Show that this linear formulation is correct.
3. What happens if the first liquid A generates 10 times the profit of the second and we want to maximize profit?

Answer (Question see 19)

1. In the expression

$$(x > a \rightarrow y \leq 0) \ \wedge \ (y > b \rightarrow x \leq 0)$$

replacing $a \rightarrow b$ by $\bar{a} \vee b$ gives:

$$(x \leq a \vee y \leq 0) \ \wedge \ (y \leq b \vee x \leq 0)$$

Redistributing $\vee$ gives:

$$(x \leq a \wedge y \leq b) \ \vee \ (x \leq a \wedge y \leq 0) \ \vee \ (y \leq b \wedge x \leq 0) \ \vee \ (y \leq 0 \wedge x \leq 0)$$

Now the last term is contained in all the previous ones, so it can be removed and the formula follows.

2. Suppose $p = 0$ and $q = 0$, then $x \leq 10, y \leq 12$, that is, the two containers are used separatedly to store the two liquids.

   Suppose $p = 1$ and $q = 0$, then $x \leq 22, y \leq 0$, that is, both containers are used to store the first liquid and the second liquid is not produced.

   Suppose $p = 0$ and $q = 1$, then $x \leq 0, y \leq 22$, that is, both containers are used to store the second liquid and the first liquid is not produced.

   Suppose $p = 1$ and $q = 1$ then $x \leq 0, y \leq 0$. This case will not be realised since at least one (x or y) is maximized. Hence, the model correctly reflects the three cases discussed above.

3. We change the maximizing function to `maximize obj:10*x+y;` . Solving the model then generates the solution $x = 22$ and $y = 0$.

20 Logical Conditions in an LP (logic6)

Problem: In a food blending problem, where several ingredients are mixeed to produce a food, the modeler wants – in addition to the common mathematical balancing and capacity restrictions – to impose the following logical constraints:

1. The food may never be made up of more than three oils (ingredients).
2. If an oil is used at least 20 tons and at most 200 tons must be used.
3. If either `Veg1` or `Veg2` is used (two vegetal oils) then `Oil3` (a non-vegetal oil) must also be used in the blend.

How can these conditions be formulated mathematically? This model part belongs to a model presented in [8]

Modeling Steps

1. For each ingredient $i \in I$ there is a variable $x_i \geq 0$ which is the quantity of the ingredient.
2. For each ingredient i a binary variable d_i is introduced with the meaning "there is (or is not) a certain quantity of the ingredient i in a blend (that means, zero quantity (for $d_i = 0$) *or* at least 20 and at most 200) (for $d_i = 1$)".
3. The binary variable d_i is linked to x_i in the following way: $d_i \rightarrow (20 \leq x_i \leq 200)$ with $i \in I$, that is, if d_i is true then x_i must be between 20 and 200.

Listing 12: The Complete Model implemented in LPL [18]

```
model Logic6 "Logical Conditions in an LP";
  set i := /Veg1 Veg2 Oil1 Oil2 Oil3/  "Ingredients in a blend";
  parameter
    Lo:=20  "Lower bound in use";
    Up:=200 "Upper bound in use";
  variable x{i} [0..300] "Quantity of ingredient used in the blend";
  binary variable d{i}; //: Lo <= x <= Up;
  constraint
    Cond1: sum{i} d <= 3;
    Cond2{i}: d -> (Lo <= x <= Up);
    Cond3: d['Veg1'] or d['Veg2'] -> d['Oil3'];
  solve; //minimize obj: sum{i} x "Minimize the output";
  //Writep(obj, x, d);
end
```

Note that the lower and upper bounds of x_i is not the same as the minimal and maximal quantity *used in the blend*. Therefore, the second condition is translated into:

$$\begin{array}{ll} x_i \leq 20\delta_i & \text{for all } i \in I \\ x_i + 100\delta_i \leq 300 & \text{for all } i \in I \end{array}$$

Further Comments: In LPL the second condition can also be integrated in the decraation of the binary variable as follows;

```
binary variable d{i}: Lo <= x <= Up
```

In this case, the assignment is interpreted as an implication. In both cases, LPL generates the following linear model:

```
Cond1:  + d[Veg1] + d[Veg2] + d[Oil1] + d[Oil2] + d[Oil3] <= 3;
Cond2[Veg1]:  - x[Veg1] + 20 d[Veg1] <= 0;
Cond2[Veg2]:  - x[Veg2] + 20 d[Veg2] <= 0;
Cond2[Oil1]:  - x[Oil1] + 20 d[Oil1] <= 0;
Cond2[Oil2]:  - x[Oil2] + 20 d[Oil2] <= 0;
Cond2[Oil3]:  - x[Oil3] + 20 d[Oil3] <= 0;
Cond3:  - d[Veg1] + d[Oil3] >= 0;
X45X[Veg1]:  + x[Veg1] + 100 d[Veg1] <= 300;
X45X[Veg2]:  + x[Veg2] + 100 d[Veg2] <= 300;
X45X[Oil1]:  + x[Oil1] + 100 d[Oil1] <= 300;
X45X[Oil2]:  + x[Oil2] + 100 d[Oil2] <= 300;
X45X[Oil3]:  + x[Oil3] + 100 d[Oil3] <= 300;
X46X:  - d[Veg2] + d[Oil3] >= 0;
```

21 Math-Logic Expression II (logic7)

Problem: Formulate the following constraint: "If at most 2 kg of the ingredient **A** or at most 3 kg of the ingredient **B** is in a mixture then at most 8 kg of the ingredient **E** or at least 7 kg of the ingredient **F** is in the mixture and vice versa." This example is from [12].

/MOmodeling Suppose the quantity of kilograms of each ingredient is a, b, e, and f, then this statement can be formulated as follows:

$$a \leq 2 \vee b \leq 3 \quad \leftrightarrow \quad e \leq 8 \vee f \geq 7$$

One can apply exactly the same procedure as in model logic0[16], Example 5.

Listing 13: The Complete Model implemented in LPL [18]

```
model Logic7 "Math-Logic Expression II";
  variable a [1..20]; b [0..100]; e[2..20]; f[3..20];
  constraint S: a<=2 or b<=3 <-> e<=8 or f>=7;
  minimize obj: a+b+e+f;
  Writep(obj);
end
```

Further Comments: McKinnon ([12]) give a solution with 6 additional indicator variables and 10 linear constraints, while LPL generates a model with 4 additional binaries and 8 linear constraints as follows:

```
S:   + X48X + X46X - X42X >= 0;
X43X:  + a + 18 X42X <= 20;
X45X:  + b + 97 X44X <= 100;
X47X:  + e + 12 X46X <= 20;
X49X:  + f - 4 X48X >= 3;
X50X:  + X44X + X42X - X46X >= 0;
X51X:  + X48X + X46X - X44X >= 0;
X52X:  + X44X + X42X - X48X >= 0;
```

Question (Answer see 21)

1. Change e<=8 to e>=8 and compare the two objective values, What do you notice?

Answer (Question see 21)

1. The solution to the first model is 6 while to the second it is 10. Minimizing the sum of the four variables will set them to their lower bound, that is: $a = 1$, $b = 0$, $e = 2$, and $f = 3$, since the left side and the right side of the equivalence are both true, the constraint is fulfiled.

   In the second case, the right side is true only if $f \geq 7$, hence the solution is $a + b + c + d = 10$. If, on the other hand, the left would be false then $a \geq 3$ and $b \geq 4$ must hold and f and g would be at there lower bound, hence $a + b + c + d \geq 12$ which is larger than 10.

[16] https://lpl.matmod.ch/lpl/Solver.jsp?name=/logic0

22 Transformations of logical statements (logic8)

Problem: A list of rules

Listing 14: The Complete Model implemented in LPL [18]

```
model logic8 "Transformations of logical statements";
  set i := [1..5];
      j := [1..10];
  parameter a;
  binary variable
    x{i}; xx{i}; xxx{i,j}; xxxx{j}; v; w; y; z; zz;
  real variable
    p [0..10]; r[0..10]; rr [10..30]; --u; o; q{i}; t{i}; tt{j,i};
  alldiff variable
    d{i} [1..5];
  constraint
  -- T0 rules
  Rule01  : y nor z;
  Rule02  : y nand z;
  Rule03  : and{i} x;
  Rule04  : or{i} x;
  Rule05  : xor{i} x;
  Rule06  : nor{i} x;
  Rule07  : nand{i} x;
  Rule08  : y <= z <= p;
  Rule08a : y <= z <= p <= v;
  -- T1 rules
  Rule10  : y->z;
  Rule11  : z<-y;
  Rule12  : forall{i} x;
  Rule13  : exist{i} x;
  Ruel14  : atleast(-2){i} x;
  Ruel15  : atmost(-2){i} x;
  Rule16  : exactly(0){i} x;
  -- T2 rules:
  Rule20  : ~(~y);
  Rule21  : ~(y and z);
  Rule22  : ~(y or z);
  Rule23  : ~(y<->z);
  Rule24  : ~(y xor z);
  Rule25  : ~(y<z);
  Rule26  : ~(atleast(3){i} x);
  Rule26a : ~(atleast(0){i} x);  // 0 means #i
  Rule27  : ~(atmost(2){i} x);
  Rule28  : ~(exactly(3){i} x);
  Rule29  : 0 <> y;
  Rule29a : 0=y;
  Rule30  : y = z;
  Rule31  : y <-> z;
  Rule32  : y <> z;
  Rule33  : y xor z;
  Rule34  : exactly(2){i} x;
  Rule35  : atmost(2){i} x;
  Rule33a : atmost(0){i} x;
  Rule36  : atleast(2){i} x;
  -- T3 rules
  Rule40  : z or (v and y);
```

```
  Rule40a : (v and y) or z;
  Rule41  : y or atleast(0){i} x;
  Rule41a : atleast(0){i} x or y;
  solve;
end
```

23 Importing Energy (import)

Problem: Coal, gas and nuclear fuel can be imported in order to satisfy the energy demands of a country. Three grades of gas and coal (low, medium, high) and one grade of nuclear fuel may be imported. The costs are known. Furthermore, there are upper and lower limits in the imported quantities from a country. What quantities should be imported if the import cost have to be minimized? (The model is from [13]). In addition, three additional conditions must be fulfilled:

1. A supply condition: Each country can supply either up to three (non-nuclear) low or medium grade fuels or nuclear fuel and one high grade fuel.
2. Environmental restriction: Environmental regulations require that nuclear fuel can be used only if medium and low grades of gas and coal are excluded.
3. Energy mixing condition: If gas is imported then either the amount of gas energy imported must lie between 40-50 percent and the amount of coal energy must be between 20-30 percent of the total energy imported or the quantity of gas energy must lie between 50-60 percent and coal is not imported.

Modeling Steps

Three index-set define are defined: $i \in I = \{\text{low,medium,high}\}$ the grade fuel, $j \in J = \{\text{coal,gas}\}$ the non-nuclear energy sources, and $k \in K = \{\text{GR,FR,IC}\}$ the countries from where to import. Nuclear energy is treated apart because it come only in one grade.

1. The data are defined and explained in the model code (see parameter section).
2. We first introduce two continuous variables: the quantity of non-nuclear fuel ($x_{i,j,k}$) and the quantity of nuclear fuel (y_k).
3. The sum of all imports must be the desired energy amount e. Hence we have
$$\sum_{i,j,k} x_{i,j,k} + \sum_{k} y_k = e$$
4. The import costs are to be minimized. Hence:
$$\min \sum_{i,j,k} c_{i,j,k} x_{i,j,k} + \sum_{k} nc_k y_k)$$
5. The three logical requirements are modeled as follows. Boolean predicates (binaries) are introduced to link the mathematical with the logical part of the model. Let $P_{i,j,k}$ be a predicate with the definition: "If at least $l_{i,j,k}$ or at most $u_{i,j,k}$ of non-nuclear energy is imported then $P_{i,j,k}$". For the nuclear energy a similar definition is introduced: "If at least nl_k or at most nu_k of nuclear energy is imported then N_k". We have:
$$\begin{array}{lll} P_{i,j,k} & \leftarrow (l_{i,j,k} \leq x_{i,j,k} \leq u_{i,j,k}) & \text{for all } i,j,k \\ N_k & \leftarrow (nl_k \leq y_k \leq nu_k) & \text{for all } k \end{array}$$

6. In condition 3, two other predicates are needed: "If the total amount of non-nuclear imported fuel, lays between 40-50% for gas and 20-30% for coal of the total imported non-nuclear fuel then Q_j", and "if imported gas is between 50-60% of the total imported non-nuclear fuel then R". Hence:

$$\begin{array}{ll} Q_j & \leftarrow (eL_j \leq \sum_{i,k} x_{i,j,k} \leq eU_j) \quad \text{for all } j \\ R & \leftarrow (eA \leq \sum_{i,k} x_{i,'gas',k} \leq eB) \end{array}$$

7. Using these predicates, the three conditions are formulated as follows:

   (a) "For each country k, either at most 3 of $P_{i,j,k}$ (with $i \neq$ 'high') are true or N_k is true and exactly one $P_{\text{'high'},j,k}$ is true"

   (b) "At least one N_k is true, only if none of $P_{i,j,k}$ is true (with $i \neq$ 'high')" and

   (c) "If any of $P_{i,\text{'gas'},k}$ is true, then either Q_j is true or R is true and none of $P_{i,\text{'coal'},k}$ is true.

Some remarks on the formulations

1. The first condition has the form: "for each k, either A or B". Normally, "either ... or" is translated as exclusive or, but sometimes we just mean or (non-exclusive). It depends on the context. Here we want to say that A and B cannot both be true at the same time. Therefore we have: $A \dot{\vee} B$.

2. The second condition has the form: "A only if B" which is simply $A \rightarrow B$.

3. The third condition has the form: "If A then (either B or (C and D))", which is $A \rightarrow (B \dot{\vee} (C \wedge D))$.

Listing 15: The Complete Model implemented in LPL [18]

```
model ImportEnergy "Importing Energy";
  set
    i   := [low med high] "Quality grades of energy";
    j   := [gas, coal]    "Non-nuclear energy sources";
    k   := [GB FR IC]     "Countries exporting energy";
  parameter
    c{i,j,k} "Non-nuclear energy cost";
    l{i,j,k} "Lower bound on the quantity of non-nuclear energy";
    u{i,j,k} "Upper bound on the quantity of non-nuclear energy";
    nc{k}    "Nuclear energy cost";
    nl{k}    "Lower bound on the quantity of nuclear energy";
    nu{k}    "Upper bound on the quantity of nuclear energy";
    e        "Desired energy amount to import";
    L{j}     "Min percent of energy if coal and gas are imported";
    U{j}     "Max percent of energy if coal and gas are imported";
    A        "Min percent of gas if gas energy only is imported";
    B        "Max percent of gas if gas energy only is imported";
  variable
    x{i,j,k} [0..u] "Quantity of non-nucl energy imported";
    y{k} [0..nu]    "Quantity of nucl energy imported";
  binary variable
    P{i,j,k} <- l<=x;
    N{k}     <- nl<=y;
```

```
    Q{j}      <- e*L <= sum{i,k} x <= e*U;
    R         <- e*A <= sum{i,k} x[i,'gas',k] <= e*B;
  constraint
    ImportReq : sum{i,j,k} x + sum{k} y = e;
    C1{k}: atmost(3){i,j|i<>'high'} P xor (N and
           xor{j} P['high',j,k]);
    C2: or{k} N  -> nor{i,j,k|i<>'high'} P;
    C3: or{i,k}P[i,'gas',k] -> and{j}Q xor R and
        nor{i,k}P[i,'coal',k];
  minimize Cost: sum{i,j,k} c*x + sum{k} nc*y;
  Writep(Cost,x,y,P,N,Q,R);
  model data;
    e         := 12000;
    c{i,j,k}  := Rnd(1,4);      nc{k}     := Rnd(1,3);
    l{i,j,k}  := Rnd(10,40);    u{i,j,k}  := Rnd(1000,2000);
    nl{k}     := Rnd(20,30);    nu{k}     := Rnd(1000,5000);
    L{j}      := [0.20 0.40];   U{j}      := [0.30 0.50];
    A         := 0.50;          B         := 0.60;
  end
end
```

24 Assembling a Radio (radio)

Problem: To assemble a radio any of three types (T1,T2,T3) of tubes can be used. The box may be either of wood W or of plastic material P. When using P, then dimensionality requirements impose the choice of T2 and a special power supply F (since there is no space for a transformer S). T1 needs F. When T2 or T3 is chosen then we need S and not F. The price of each component is given. A radio made of wood or of plastic has different selling prices. Should we build a radio of wood or of plastic when we want to maximize profit? (The problem is from [15]).

Modeling Steps

This model shows how mathematical and logical constraints can be used side by side.

1. There are 7 possible components that are used to build a radio. For each component a binary variable is introduces, that says whether to use it or not to build our radio.

2. The constraints easy to derive. The objective function is the selling prices minus the costs of all components. Since the radio is either of wood or plastic only one of the variable `W` or `P` is differnt from zero.

Listing 16: The Complete Model implemented in LPL [18]

```
model Radio "Assembling a Radio";
  binary variable
    T1 "Tube of type I";
    T2 "Tube of type II";
    T3 "Tube of type III";
    W  "a wooden box";
    P  "a plastic box";
    F  "a transformator";
    S  "a special power supply";
  constraint
    R1: T1+T2+T3=1      "Use one tube";
    R2: W xor P         "Wood or plastic?";
    R3: P -> T2 and S   "If Plastic then use T2 and S";
    R4: T1 -> F         "T1 needs a transformator";
    R5: T2 or T3 -> S   "When choosing T2 or T3, we need S";
    R6: F xor S         "Either F or S must be used";
  expression cost:=55*T1+58*T2+56*T3+25*F+23*S+9*W+6*P+10;
  maximize Profit: 110*W + 105*P - cost;
  Write('The␣profit␣is:␣%d,␣the␣radio␣is␣made␣of␣\
␣␣␣␣␣␣␣␣␣%s.\n',Profit,if(W,'wood','plastic'));
end
```

Solution: The profit for one radio made of wood is 12. It uses tube III and a special power supply.

Question (Answer see 24)

1. What is the profit if the radio is made of plastic?

Answer (Question see 24)

1. The profit is 8. It uses tube II and also a special power supply. This can be found by adding the constraint:

```
constraint R7: ~W;    //do not use a wooden box !
```

25 Pigeonhole Problem (pihole)

Problem: The pigeon hole problem is to place $n+1$ pigeons in n holes so that no hole contains more than one pigeon. The problem is clearly infeasible. The problem is interesting for its concise formulation as a SAT (satisfiability) problem and because it is very difficult to solve using the method of resolution.

A SAT formulation is (with $x_{i,j} = \{\text{true,false}\}$):

$$\bigvee_{j}^{n} x_{i,j}, \quad \text{for all } i = 1,\ldots,n+1$$

$$\neg x_{i,j} \ \vee \ \neg x_{k,j} \ , \quad \text{for all } i,k = 1,\ldots,n+1 \ , \ i \neq k \ , \ j = 1,\ldots,n$$

Formulated as a IP linear problem, it can be *very easy* or *very difficult* to prove infeasibility, depending on how it is formulated and depending on the solvers methods.

The problem was described in (see [1], p.110, and [2], p.13).

Modeling Steps

1. We introduce a binary variable for each (i,j)-combination, that is, $x_{i,j} = 1$ if pigeon i is in hole j, otherwise $x_{i,j} = 0$.

2. The first constraint states that each pigeon is placed in exactly one hole.

$$\sum_{j=1}^{n} x_{i,j} = 1 \ , \quad \text{for all } i = 1,\ldots,n+1$$

3. The second constraint states that for each pair of pigeons $(i,k), i \neq k$, both do not occupy the same hole j (constraint `S0`):

$$x_{i,j} + x_{k,j} \leq 1 \ , \quad \text{for all } i,k = 1,\ldots,n+1 \ , \ i \neq k \ , \ j = 1,\ldots,n$$

4. There is another way to formulated this last constraint: For each hole j there can be at most one pigeon (constraint `S1`):

$$\sum_{i}^{n+1} x_{i,j} \leq 1 \ , \quad \text{for all } j = 1,\ldots,n$$

Listing 17: The Complete Model implemented in LPL [18]

```
model Pihole "Pigeonhole Problem";
  --SetSolver(glpkSol);
  parameter n:=10  "number of holes";
  set i,k:=1..n+1  "n+1 Pigeons";
  set j  :=1..n    "n Holes";
  binary variable x{i,j};
  constraint
    R{i}: sum{j} x = 1 "Each pigeon i must be in exactly one hole";
    --S0{i,k,j|i<>k}: x[i,j] + x[k,j] <= 1;
    S1{j}: sum{i} x <= 1;
  solve;
end
```

Note that the first formulation (with constraint `S0`) has $n(n+1)^2$ constraint, while the second formulation (with constraint `S1` instead) is much smaller and contains only $2n+1$ constraint. Furthermore, the LP relaxation of the first formulation is feasible with $x_{i,j} = \frac{1}{n}$, while the LP relaxation of the second formulation is infeasible. Thus, IP solvers – based on LP relaxations– will immediately discover the infeasibility of the pigeon hole problem in the second formulation, while in the first formulation the infeasibility could be very difficult to be proven. Therefore, the free solver *glpk* found the feasibility of the second formulation immediately, while it was not possible to prove infeasibility for the first formulation in reasonable time.

Further Comments: Alternatively, the constraints can also be formulated in a purly logical way in LPL as follows:

```
constraint
  R{i}: exactly(1){j} x  "Each pigeon i must be in exactly one hole";
  --S0{i,k,j|i<>k}: x[i,j] nand x[k,j];
  S1{j}: atmost(1){i} x;
```

Question (Answer see 25)

1. Try to solve both model versions – once with S0 and without S1 and once with S1 and without S0, with the two solvers glpk and lp_solve.
2. Try a problem with $n = 100$ and solve it with a commercial solver – like Gurobi. Try both formulations.

Answer (Question see 25)

1. Prove infeasibility with S1 is easy and the solver takes no time. With S1 the solvers take a long time to prove infeasibility.
2. Depending on commercial solvers, it is easy or difficult to solve with S0. For Gurobi it was easy to solve it.

References

[1] V. Chandru and J.N. Hooker. *Optimization Methods for Logical Inference.* J. Wiley Sons, Inc., 1999.

[2] Bertsimas D. and Weismantel R. *Optimization over Integers.* Dynamic Ideas, Belmont, 2005.

[3] Meier G. and Düsing R. Zur Modellierung logischer Aussagen ergänzend zu Linearen Programmen, Grundlagen und Entwurfsüberlegungen für einen Modellgenerator. *OR-Spektrum, Vol 14, p. 149–160*, 1992.

[4] D. Gries and F.B. Schneider. *A Logical Approach to Discrete Math.* Springer, p 37, 1994.

[5] Williams H.P. Logical problems and integer programming. *Bull. Inst. Math. Appl.*, 13:18–20, 1977.

[6] Williams H.P. An alternative explanation of disjunctive formulations. *European Journal of Operations Research*, 72:200–203, 1994.

[7] Williams H.P. *Logic and Integer Programming.* Springer, 2009.

[8] Williams H.P. *Model Building in Mathematical Programming.* John Wiley, Fifth Edition, 2013.

[9] Hooker J.N. A Quantitative Approach to Logical Inference. *Workshop "Mathematics and AI", Vol II, FAW Ulm, 19th-22nd December 1988*, pages 289–314, 1988.

[10] Wolsey L.A. *Integer Programming.* Wiley, 1998.

[11] MatMod. Homepage for Learning Mathematical Modeling : https://matmod.ch .

[12] McKinnon and Williams. 1989.

[13] G. Mitra, C. Lucas, and S. Moody. Tools for reformulation logical forms into zero-one mixed integer programs. *European Journal of Operational Research*, 72:2:262–276, 1994.

[14] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization.* Wiley, 1988.

[15] Barth P. *Logic-Based 0-1 Constraint Programming.* Kluwer Academic Publishers, p.28, 1996.

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

[17] Hürlimann T. Index Notation in Mathematics and Modeling Language LPL: Theory and Exercises. https://matmod.ch/lpl/doc/indexing.pdf.

[18] Hürlimann T. Reference Manual for the LPL Modeling Language, most recent version. https://matmod.ch/lpl/doc/manual.pdf.

[19] Hürlimann T. Various Model Types. https://matmod.ch/lpl/doc/variants.pdf.