Index Notation in Mathematics and Modelling Language LPL: Theory and Exercises

Tony Hürlimann

June 8, 2019

Abstract

This paper explains indexing notation in mathematics and its implementation in the modeling language LPL. Indexing is one of the most fundamental concept in mathematical notation. It is extremely powerful and allows the modeler to concisely formulate a complex mathematical model. A certain number of exercises are given and implemented in LPL to test the concepts.

Contents

1 Introduction

To work with a mass of data, a concise formalism and notation is needed. In mathematics, the so-called *indexed notation*. This notation is an integral and fundamental part of every mathematical modeling activity – and they are also used in mathematical modeling languages as in LPL [3] – to group various objects and entities. It is surprising how little has been published in the community of modeling language design on this concept. Even in mathematical textbooks, the indexed notation used in formulae is often taken for granted and not much thought is given to it[1]. This fact contrasts with my experience that students often have difficulties with the indexed notation in mathematics.

To represent single data, called *scalars*, names and expressions are used, such as:

$$x, \quad y \quad \text{or} \quad x - y$$

To specify further what we mean by a symbol, we write $x, y \in \mathbb{R}$, for example, saying that x and y represent any real numbers. We can attach a new symbol (z) to an expression as in:

$$z = x - y$$

meaning that z is a new number which is defined as the difference of x and y. We say "z substitutes the expression $x - y$". In this way, we can build complex expressions. We learnt this algebraic notation already in school. These symbols have no specific "meaning" besides the fact that they represent numbers. To use them in a modeling context, we then attach a meaning. For example, in an economical context we might interpret "x" as "total revenue" and "y" as "total costs". Then "z" might be interpreted as "total profit". There is nothing magic about these symbols, instead of writing $z = x - y$ we can also write:

$$\text{total profit} = \text{total revenue} - \text{total costs}$$

However, to economize our writings we prefer short "names". But there is nothing wrong with longer names to make expressions more readable in a specific context.

It seems to be a little bit less familiar that in the same way as using symbols for *scalars*, symbols can represent mass of data. Suppose that we want to express the profit of *several* profit centers in a company. Then we could write:

$$\begin{aligned}
\text{profit at center 1} &= \text{revenue at center 1} - \text{costs at center 1} \\
\text{profit at center 2} &= \text{revenue at center 2} - \text{costs at center 2} \\
\text{profit at center 3} &= \text{revenue at center 3} - \text{costs at center 3} \\
&\ldots \quad \textit{etc.} \quad \ldots
\end{aligned}$$

or

$$\begin{aligned}
z1 &= x1 - y1 \\
z2 &= x2 - y2
\end{aligned}$$

[1] A notable exception is [1], which devotes the whole Chapter 2 to indexed expressions and their use in mathematics.

$$z3 = x3 - y3$$
$$\ldots \quad etc. \quad \ldots$$

The "etc." means that we have to continue writing as many lines as we have profit centers – a rather boring task!

There is, however, a much more economical way in mathematics to represent *all* these expressions in a single statement. It is the *indexed notation*. To formulate them, we first introduce a set: the set I of all profit centers:

$$I = \{\text{center 1 }, \quad \text{center 2} \quad , \quad \ldots \quad \}$$

here again the "..." means that we continue writing all centers in a list. Often a short name form is used as follows:

$$I = \{1 \ldots n\} \qquad \text{where} \quad n \in \mathbb{N}^+$$

In the previous set definition we just mean that there are n centers, n being a positive number. We are not concerned right now of how many centers there are, we just say "n" – it can be 5 or 1000. Of course, in a concrete context we must specify the number n, but it is part of the data of that specific context.

In a second step, we introduce symbols for all profits, revenues and costs. Now, instead of using each time a new symbol, we just attach an subscript to the symbols: z_i, x_i, y_i to express the fact that they "mean" the profit, the revenue and the cost of the i-th center, together with the notation $i \in I$, which means that i just designates an arbitrary (i-th) element in I. Hence, All data can be written in a concise way as follows:

$$x_i, \quad y_i, \quad z_i \qquad \text{where} \quad i \in I$$

Mathematically speaking,three *vectors* to represent all data are declared. The list of expression then can be written in a single statement as follows:

$$z_i = x_i - y_i \qquad \text{with} \quad i \in I$$

This notation is in fact nothing else than an economical way of the following n expressions:

$$z_1 = x_1 - y_1$$
$$z_2 = x_2 - y_2$$
$$\ldots$$
$$z_n = x_n - y_n$$

where n is some positive integer.

In a similar way, we can concisely build an expression that sums all profits, for instance. For this purpose we use the mathematical operator $\sum$. The total profit p of all profit centers can be formulated as follows:

$$p = \sum_{i \in I} z_i$$

After this introductory example, this paper presents now a more precise way, on how the indexed notation is defined and how it can be used.

2 Definitions and Notations

Index sets are essential for mastering the complexity of large models. All elements in a model, such as variables, parameters, and constraints – as will be shown later on – can appear in groups, in the same way they are indexed in mathematical notation.

The convention in algebraic notation to denote a set of similar expressions is to use *indexes* and *index sets*. For example, a summation of n (numerical) terms

$$a_1 + a_2 + \cdots + a_{n-1} + a_n \tag{1}$$

is commonly abbreviated using the notation

$$\sum_{i=1}^{n} a_i \tag{2}$$

The expression (1) is called *three-dots notation* and expression (2) is called *sigma-notation*. Both expression are equivalent, but (2) is much shorter and more general. The later was introduced by Joseph Fourier in 1820, according to [1, p. 22]. There exist different variants of expression (2) :

$$\sum_{1\leq i\leq n} a_i \quad , \quad \sum_{i=1}^{n} a_i \quad , \quad \sum_{i\in\{1\ldots n\}} a_i \quad , \quad \sum_{i\in I} a_i \quad \text{with} \quad I = \{1\ldots n\} \tag{3}$$

All four expressions in (3) are equivalent and they have all their advantages and disadvantages. The last notation in (3) is more general, because the index set I can be an arbitrary set defined outside the summation expression.

2.1 Exercises in the Sigma Notation

Before we generalize the indexed notation, let us give some examples with the sigma form.

A notation $\sum_{i=1}^{5} x_i$ means that i is replaced by whole numbers starting with 1 until the number 5 is reached. Thus

$$\sum_{i=2}^{4} x_i = x_2 + x_3 + x_4$$

and

$$\sum_{i=2}^{5} x_i = x_2 + x_3 + x_4 + x_5$$

Hence, the notation $\sum_{i=1}^{n}$ tells us:

1. to add the numbers x_i,
2. to start with $i = 1$, that is, with x_1,
3. to stop with $i = n$, that is, with x_n (where n is some positive integer).

As an example, let us assign the following values: $x_1 = 10$, $x_2 = 8$, $x_3 = 2$, $x_4 = 15$, and $x_5 = 22$. Then we have:

$$\sum_{i=1}^{5} x_i = x_1 + x_2 + x_3 + x_4 + x_5 = 10 + 8 + 2 + 15 + 22 = 57$$

The name i is a dummy variable, any other name could be used. We could have used j, the expression would have be exactly the same, hence:

$$\sum_{i=1}^{5} x_i = \sum_{j=1}^{5} x_j$$

Now let us find $\sum_{i=1}^{4} 3x_i$ based on the previous values. Again we start with $i = 1$ and we replace $3x_i$ with its value:

$$\sum_{i=1}^{4} 3x_i = 3x_1 + 3x_2 + 3x_3 + 3x_4 = 3 \cdot 10 + 3 \cdot 8 + 3 \cdot 2 + 3 \cdot 15 = 105$$

Similarly, let use find $\sum_{i=2}^{5}(x_i - 8)$: This is:

$$\begin{aligned}\sum_{i=2}^{5}(x_i - 8) &= (x_2 - 8) + (x_3 - 8) + (x_4 - 8) + (x_5 - 8) \\ &= (8 - 8) + (2 - 8) + (15 - 8) + (22 - 8) = 17\end{aligned}$$

One should be careful with the parentheses. The expression $\sum_{i=2}^{5}(x_i - 8)$ is not the same as $\sum_{i=2}^{5} x_i - 8$. The later evaluates to (the 8 is not included in the sum):

$$\sum_{i=2}^{5} x_i - 8 = x_2 + x_3 + x_4 + x_5 - 8 = 29$$

We also use sigma notation in the following way:

$$\sum_{j=1}^{4} j^2 = 1^2 + 2^2 + 3^2 + 4^2 = 30$$

The same principle applies here. j is replaced in the expression j^2 by numbers starting with 1 and ending with 4, and then adding up all four terms.

For the sigma notation we have three important transformation rules:

Rule 1: if c is a constant, then:

$$\sum_{i=1}^{n} cx_i = c\sum_{i=1}^{n} x_i$$

Rule 2: if c is a constant, then:

$$\sum_{i=1}^{n} c = nc$$

Rule 3: Adding the term can be distributed:

$$< qquad \sum_{i=1}^{n}(x_i + y_i) = \sum_{i=1}^{n} x_i + \sum_{i=1}^{n} y_i$$

Proof:

$$\sum_{i=1}^{n} cx_i = cx_1 + cx_2 + \ldots + cx_{n-1} + cx_n$$

$$= c \cdot (x_1 + x_2 + \ldots + x_{n-1} + x_n) = c \cdot \sum_{i=1}^{n} x_i$$

$$\sum_{i=1}^{n} c = \underbrace{c + c + \ldots + c}_{n-times} = n \times c = nc$$

$$\sum_{i=1}^{n}(x_i + y_i) = (x_1 + y_1) + \ldots + (x_n + y_n)$$

$$= (x_1 + \ldots + x_n) + (y_1 + \ldots + y_n) = \sum_{i=1}^{n} x_i + \sum_{i=1}^{n} y_i$$

End of proof

2.2 Formal Definition

More formally, let us call an *indexed notation* written as

$$\bigodot_{i \in I} E_i \tag{4}$$

a formal expression containing the following elements:

1. An *index operator*, here written as $\bigodot$. This operator can be any associative and commutative operator, such as $\sum$ (summation), for example.

2. An *index set*, represented by I, being any countable (finite or infinite) collection of *elements*. The elements can be any indivisible item. Often they are integers.

3. An *active index* i (in $i \in I$), attached to the index operator.

4. An *indexed expression* E_i, that normally contains the same index name again. We call this index *passive index*. The reason for that will be clear later on.

(1) The index operator: It is clear why the *index operator* must be commutative and associative: A mathematical set is unordered and its elements can be "traversed" in any order. As an example, the $\sum$ does not specify in which order the indexed expressions E_i are added up. Various operators fulfill these requirements. Besides the addition, represented by the operator $\sum$, the following index operators are commonly used (examples will be given later on):

- The product (multiplication) $\prod$: The indexed expressions are multiplied.
- The Max (Min) operator, written as $\max_{i \in I}$ ($\min_{i \in I}$): They return the largest (smallest) indexed expression in the list.
- In Boolean logical expressions we use the AND- and OR-operators ($\bigwedge$, $\bigvee$): They check whether all indexed expressions are true or at least one is true.
- An particular index operator is the *list operator* $\dot{\bigwedge}$. It is used to list indexed expressions.

(2) The index set: In the simplest case, the elements of the *index set* are integers,a s in

$$I = \{1, \ldots, n\}, \qquad \text{where } n > 0$$

However, they can also be identifiers (symbolic names), strings or any other items and even sets representing elements of a set. In the following index set I with:

$$I = \{ \text{ spring , summer , autumn , winter } \}$$

the four elements are identifiers. Sets are unordered as mentioned above. In modeling environments, however, we often must impose a specific order, for example, if a set consists of a number of time periods or time points. Then the natural order is the sequence of these periods. For example, the weekdays:

$$I = \{ \text{ Mon, Tue, Wen , Thu, Fri, Sat, Son } \}$$

The order is important in such a context, because we often need to refer to the "previous", the "next", the "last" or the "first" period in an indexed expression. In spacial arrangements the order might also be important. Consider a chessboard with $I = \{1 \ldots 8\}$ rows and columns, then it makes sense to refer to the "neighbor" columns at the left or right and the "neighbor" rows above or below a given cell. In another context, the ordering might have a semantical meaning. For example, if we want – for a specific purpose – impose an order on a list of products. We can order the product by "importance" or whatever criterion we need. Without ordering the statement to generate a "list with the

first 10 products" would have no meaning. Without any externally imposed criterion the ordering of the products would have no meaning, as in:

$$I = \{ \text{ bred, cheese, meat, eggs } \}$$

We also may have set of tuples, which is extremely common in modeling. An example is the list of all positive integer grid points in a Euclidean space:

$$I = \{(x, y) \mid x, y \in \mathrm{IN}^+\}$$

$$= \{(0,0), (1,0), (1,1), (0,1), (0,2), (1,2), (2,2), (2,1), \ldots\}$$

Another important case in modeling is "set of sets". As a concrete example, certain clients are served from certain warehouses. Let $I = \{1, \ldots m\}$ be a set of warehouses and let $J = \{1, \ldots n\}$ be a set of clients, then the set S:

$$S = \{Q_i \mid i \in I \ , \ Q_i \subseteq J\}$$

tells us which client j is served from which warehouse. Example:

$$S = \{\{1,2,3\}, \{2,3\}, \{\}, \{2\}\} \quad \text{, hence} \quad \begin{array}{l} Q_1 = \{1,2,3\} \\ Q_2 = \{2,3\} \\ Q_3 = \{\} \\ Q_4 = \{2\} \end{array}$$

says, that warehouse 1 serves client $1, 2, 3$, warehouse 2 serves clients $2, 3$, warehouse 3 do not serve any client, and warehouse 4 serves client 2, that is, Q_j is the set of clients served by warehouse $j \in J$. Note that often this kind of data can also be formulated as a tuple list:

$$S_{i,j} = \{(1,1), (1,2), (1,3), (2,2), (2,3), (4,2)\} \quad \forall\, i \in I, j \in J$$

(3) The active index: The *active index* is a (dummy) name (or an identifier) representing an arbitrary element in the index set. The *passive index* is also an identifier used in the indexed expression which must have the same spelling as the active index. The term $i \in I$, consisting of the active index and the index set, is called an *indexing term*. The active index and the passive index are used as place-holders to define a bijective mapping between the index set and a *set of indexed expressions*. Each element i in the index set maps to an indexed expression by replacing the passive index with the corresponding element i in the index set. This mapping is called the *index mechanism*. In fact, it defines the (bijective) function $f : i \xrightarrow{f} E_i$ with $i \in I$. This mapping supposes that the corresponding indexed expressions E_i exist. The *domain* of this function f contains the elements of the index set $\{1, 2, \ldots, n\}$, and the *codomain* (or the *range*) is the set of indexed expressions $\{E_1, E_2, \ldots, E_n\}$.

(4) The index expression: The *indexed expression* is an arbitrary expression. It is not necessary that it contains a passive index. For example, in the expression:

$$\sum_{i \in I} 3 \quad \text{with} \quad I = \{1 \ldots n\}$$

the indexed expression is just 3. The whole expression then reduces simply to:

$$\underbrace{3+3+\ldots+3}_{n-times} = 3 \cdot n$$

The indexed expression can alas be reduced to a passive index as in:

$$\sum_{i \in I} i \quad \text{with} \quad I = \{1 \ldots n\}$$

in which case the indexed expression is just i. The whole expression reduces to:

$$1 + 2 + \ldots + n$$

The indexed expression can itself contain index operators. For example:

$$\sum_{i \in I} \left(\sum_{j \in J} F_{i,j} \right) \quad \text{with} \quad I = \{1 \ldots n\}, \quad J = \{1 \ldots m\}$$

In this case, the expression first expands to:

$$\sum_{i \in I} (F_{i,1} + F_{i,2} + \ldots + F_{i,m}) \quad \text{with} \quad I = \{1 \ldots n\}$$

Finally, the outer sum is applied to get the expression

$$F_{1,1} + F_{1,2} + \ldots + F_{1,m} + F_{2,1} + F_{2,2} + \ldots\ldots + F_{n,m}$$

Definition: An *indexed notation* is a formula expressing the fact that the *index operator* is applied to the set of *indexed expressions* constructed by the *index mechanism.*, that is, a bijective mapping between the *elements of the index set* and the *indexed expressions.*

2.3 Extensions

The general indexed notation is given by the following notation as seen before:

$$\bigodot_{i \in I} E_i$$

This is really the most general notation, no further concepts are needed to write more complex index notations. This will be shown now by studying several "extensions", which all can be reduced to this simple form.

2.3.1 A list of (indexed) expressions

A list of expressions is normally written as follows:

$$z_i = x_i - y_i \qquad \text{with} \quad i \in I$$

This notation is very common. It says that the equations $z_i = x_i - y_i$ has to be repeated as many times as I has elements. However, this notation could be transformed into the general indexed notation by using the *list operator* as follows:

$$\dot{\bigwedge}_{i \in I} (z_i = x_i - y_i)$$

The index operator is the $\dot{\bigwedge}$-operator. The meaning of this operator is to list all indexed expressions. In this case, the indexed expression is normally an equation or an assignment, hence the "side effect" of this operator is to assign a list of values. Of course, in a mathematical text we would prefer the first notation, because it is so common. However, this example shows that it can really be seen as an indexed notation. This fact will be important in the context of modeling languages later on, because we can use the same syntax to specify a list of expressions. To express it, we only need to use the list operator instead of any other index operator, as for instance the sigma operator.

2.3.2 Index Sets as Expressions

In many situations, the index set is not simply an explicit list of elements, as in $I = \{1 \dots n\}$. Of course, any set expression can be used to construct the index set I. Hence, I could be the union or the intersection of other sets as in the following expression:

$$I = J \cup K \qquad \text{or} \qquad I = J \cap K$$

This kind of modification does not change anything for the indexed notation. It is immaterial, how the index set is constructed. Sometimes, however, we construct I as a subset of another set, say J, such that $I \subseteq J$, and we use the notation:

$$I = \{j \in J \mid P(j)\}$$

where $P(j)$ is a property about $j \in J$ that is either true or false. The notation means: "for all $j \in J$ such that $P(j)$ is true". Clearly, if $P(j)$ is true for all elements in J then the two sets are equal ($I = J$). However, normally $P(j)$ expresses a property that is true only for a proper subset of J. Using I in an indexed notation, we could write:

$$\sum_{i \in I} E_i \qquad \text{with} \qquad I = \{j \in J \mid P(j)\}$$

However, it is more common and much shorter to use the notation which is as follows:

$$\sum_{j \in J | P(j)} E_j$$

Hence, we attach the property $P(j)$ directly to the *indexing term* starting it with the symbol $|$. This is extremely useful and very common in modeling. Suppose we have a list of products. Then it is very common to sum over different subsets of products, for instance "all products for which the demand is larger than x" or "all products that are green" or whatever. It would be exaggerated to generate an explicit named index set each time.

We give several examples: Suppose we have $x_1 = 10$, $x_2 = 8$, $x_3 = 2$, $x_4 = 15$, and $x_5 = 22$. Furthermore, $P(i)$ is defined to be "$x_i \leq 9$", and $Q(i)$ is defined as "$i \geq 4$" with $i \in I = \{1 \dots 5\}$. Then we evaluate the following expressions:

$$\sum_{i \in I | P(i)} x_i = x_2 + x_3 = 8 + 2 = 10$$

$$\sum_{i \in I | Q(i)} x_i = x_4 + x_5 = 15 + 22 = 37$$

In the first expression $P(i)$ is true only for x_2 and x_3, hence the sum take place only over these two terms. In the second expression, $Q(i)$ is true only for $i = 4$ and $i = 5$.

There is no need to explicitly name the properties; their corresponding expressions could be directly used in the indexing terms. Hence the two previous expressions could also be written as:

$$\sum_{i \in I | x_i \leq 9} x_i = x_2 + x_3 = 8 + 2 = 10$$

$$\sum_{i \in I | i \geq 4} x_1 = x_4 + x_5 = 15 + 22 = 37$$

We can write any Boolean expression after $| \ldots$ in the indexing term. This expression is called *indexing condition*.

Other examples with $k \in K = \{1 \ldots 100\}$ are as following:

$$\sum_{k \in K | k^2 \leq 100} k = \sum_{i \in \{1 \ldots 10\}} i = 1 + 2 + \ldots + 10$$

$$\sum_{k \in K | 3 \leq k \leq 5} k^2 = 3^2 + 4^2 + 5^2 = 50$$

$$\sum_{k \in K | k \text{ is prime}} k = 2 + 3 + 5 + 7 + 11 + 17 + ... + 97$$

In the first expression $k^2 \leq 100$ is only true for $1 \leq k \leq 10$. In the second expression, only $3, 4, 5$ are allow for k. In the third, we say to take every k up to 100, such that k is prime.
We saw, to extend the indexing term from "$i \in I$" to "$i \in I \mid P(i)$" does not change the meaning of the indexed notation. The expression $i \in I \mid P(i)$ is still a set albeit a subset of I that has no explicit name. However, no further concept has been added to the indexed notation.

2.3.3 Compound Index Notation

An indexed expression can itself contain an indexed notation, which generates nested indexed notations, as in the following expression:

$$\bigodot_{i \in I} \left(\bigotimes_{j \in J} F_{i,j} \right) \tag{5}$$

where $\bigodot$ and $\bigotimes$ are two arbitrary index operators. If the two operators are different then we need to evaluate the inner expression first and the outer afterwards. A notable example is a system of m linear equations with n variables that is commonly noted as follows:

$$\sum_{j \in \{1 \ldots n\}} a_{i,j} x_j = b_i \quad \text{with} \quad i \in \{1 \ldots m\}$$

Using the *list operator* $\dot{\bigwedge}$, these formula can be noted as a nested indexed notation, that is, a notation where the index expression contains itself an indexed notation, with the outer index operator $\dot{\bigwedge}$ and the inner index operator as $\sum$. The nested notation then is noted as follows:

$$\dot{\bigwedge}_{i\in\{1\ldots m\}} \left(\sum_{j\in\{1\ldots n\}} a_{i,j}x_j = b_i \right)$$

This notation is a very short writing of the following system of equations:

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + \ldots + a_{1,n-1}x_{n-1} + a_{1,n}x_n &= b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \ldots + a_{2,n-1}x_{n-1} + a_{2,n}x_n &= b_2 \\ \ldots &= \ldots \\ a_{m,1}x_1 + a_{m,2}x_2 + \ldots + a_{m,n-1}x_{n-1} + a_{m,n}x_n &= b_m \end{aligned}$$

On the other hand, if both index operators in (5) are identical, then the two index operators can be merged in the following way:

$$\bigodot_{i\in I} \left(\bigodot_{j\in J} F_{i,j} \right) = \bigodot_{i\in I,\ j\in J} F_{i,j}$$

It means that we merge the operator and unify also the indexing terms to be written as "$i \in I,\ j \in J$". It signifies that we build *every tuple* of the Cartesian Product of the two sets: $(i, j) \in I \times J$. As an example, suppose $I = \{1, 2\}$ and $J = \{a, b, c\}$ then the index set is $I \times J = \{(1, a), (1, b), (1, c), ((2, a), (2, b), (2, c)\}$ – it consists of 6 tuples. There are two active indexes i and j, or a tuple (i, j) of active indexes. Hence we also may write this in the following way:

$$\bigodot_{(i,j)\in I\times J} F_{i,j}$$

where the index set is $I \times J$ and the active index is replaced by the notation (i, j) which is called an *active index tuple*. In this case, the names within the active index tuples are called *active indices*. The active indices are now place-holders for the single *components* of the tuple.

The index set $I \times J$ is also a set, the set of all tuples in the Cartesian Product. Hence if we do not need to access the single components within the product, we can replace (i, j) by a single active index, say k, and $I \times J$ can be replaced by a set name, say K. Now we see that the previous formula really reduces to our general indexed notation, which is as follows:

$$\bigodot_{(i,j)\in I\times J} F_{i,j} = \bigodot_{k\in K} E_k$$

Since indexed notation with multiple indexing terms really is reducible to indexed notation with one indexing term, we only need to extend the definition of index sets:

Definition: A *compound index set* is any countable (finite or infinite) collection of elements, where all elements are either indivisible items called *atoms* or *tuples* containing all the same number of components.

It is interesting to note that the previous three indexed notations are essentially the same. In addition they allows one to specify a proper *subset* of the full Cartesian Product.

Sometimes access to the tuple itself *and* to its components is needed. In such cases, one could use a combined syntax as in:

$$\bigodot_{k=(i,j)\in I\times J} (E_k, F_{i,j}) \tag{6}$$

where $(E_k, F_{i,j})$ is the indexed expression.

The previous considerations can be generalized to tuples of more than two components: If $I_1, \ldots, I_p$ are p non-empty, not necessary different index sets containing only atoms, then any subset of the Cartesian product $I_1 \times \cdots \times I_p$ is also an index set, called *compound index set*. Its elements are ordered tuples denoted by $(i_1, \ldots, i_p)$ with $i_1 \in I_1, \ldots, i_p \in I_p$. Each item i_k with $k \in \{1 \ldots p\}$ in a tuple $(i_1, \ldots, i_p)$ is called a *component* within the tuple. The integer p determines the *arity (dimension) of the tuple*. All elements in a compound index set have the same arity, and the order of the components is significant. If the underlying index sets $I_1, \ldots, I_p$ are all ordered, then the order of the corresponding compound index set is determined by the lexicographic ordering: the right most component is varied first.

The *active index tuple* $(i_1, \ldots, i_p)$ contains the active indices of each component for the underlying index sets. If an active index is needed for the tuple itself, it can be preceded by a new unique name as in $k = (i_1, \ldots, i_p)$, called *named active index tuple*. k is now an active index for the tuple itself. This notation can be extended to any subset of components of the active index tuple. For example, if an active index is used for a sub-tuple consisting of the second, fourth and p-th component besides an index for the tuple itself, one could write this as: "$k_1 = (i_2, i_4, i_p), k = (i_1, \ldots, i_p)$".

We may call this a *list of named active index tuples*. It is important to note that every name for a sub-tuple (the names before the equal signs) as well as all active indices in the *last* (complete) active index tuple must be different from each other; however, all names in the *other* sub-tuples must occur in the last tuple, because they only define a selection from the last tuple. (Of course, the same selection may turn up more than once, defining several active indices for the same sub-tuple.)

The previous considerations can now be generalized to a list of p indexing terms as follows (where $K \subseteq I_1 \times \cdots \times I_p$):

$$\bigodot_{i_1\in I_1,\ldots,i_p\in I_p} E_{i_1,\ldots,i_p}\ , \qquad \bigodot_{(i_1,\ldots,i_p)\in I_1\times\cdots\times I_p} E_{i_1,\ldots,i_p}\ , \qquad \bigodot_{k\in I_1\times\cdots\times I_p} E_k\ , \qquad \bigodot_{k\in K} E_k$$

All tuples in the indexing terms have arity p. All four indexing expressions represent a p-dimensional table of expressions. *Simple index sets* (which contain only atoms) can be interpreted as compound index sets with arity 1. In this case, the parentheses around the "tuple" are not needed.

The use of indexed notation together with *compound index sets*, that is, sets which are subsets of some Cartesian Product, might be considered as a purely theoretical exercise.

However, this is not the case. This notation is extremely useful in practice, and it is worth while to understand the mechanism and the notation very well. Therefore, let us give an example.

Exercise 1:

Suppose we have a set of 3 products $P = \{1, 2, 3\}$ and a set of 4 factory locations $F = \{1, 2, 3, 4\}$, where these products are manufactured. The products are transported from one factory to another at unit cost $c_{i,j}$ with $i, j \in F$. Furthermore, each product has a value v_p with $p \in P$. Suppose also, that we transport $x_{p,i,j}$ unities of product p from factory i to factory j with $i, j \in F$ and $p \in P$. The data are as follows:

$$v = \begin{pmatrix} 6 & 2 & 3 \end{pmatrix} \qquad c = \begin{pmatrix} - & 1 & 5 & 2 \\ 2 & - & 4 & 2 \\ 1 & 2 & - & 3 \\ 1 & 3 & 1 & - \end{pmatrix} \qquad x_{p=1} = \begin{pmatrix} - & 48 & 37 & 45 \\ 36 & - & 43 & 46 \\ 44 & 36 & - & 33 \\ 36 & 39 & 34 & - \end{pmatrix}$$

$$x_{p=2} = \begin{pmatrix} - & 46 & 35 & 39 \\ 32 & - & 47 & 35 \\ 45 & 49 & - & 39 \\ 47 & 46 & 30 & - \end{pmatrix} \qquad x_{p=3} = \begin{pmatrix} - & 32 & 32 & 40 \\ 30 & - & 41 & 30 \\ 45 & 43 & - & 45 \\ 44 & 41 & 34 & - \end{pmatrix}$$

Using these data, we now can calculate various data using indexed notation:
(1) The total transportation costs T is calculated as follows:

$$T = \sum_{p \in P,\ i,j \in F} c_{i,j} \cdot x_{p,i,j} = 3180$$

(2) The total value transported is as follows:

$$V = \sum_{p \in P,\ i,j \in F} v_p \cdot x_{p,i,j} = 5213$$

(3) The total transportation costs C_p for each product $p \in P$ is:

$$\dot{\bigwedge}_{p \in P} \left(C_p = \sum_{i,j \in F} c_{i,j} \cdot x_{p,i,j} \right) = \begin{pmatrix} 1061 \\ 1096 \\ 1023 \end{pmatrix}$$

(4) The value $W_{i,j}$ transported between all factories $(i, j) \in F \times F$ is:

$$\dot{\bigwedge}_{i,j \in F} \left(W_{i,j} = \sum_{p \in P} v_p \cdot x_{p,i,j} \right) = \begin{pmatrix} - & 476 & 388 & 468 \\ 370 & - & 475 & 436 \\ 489 & 443 & - & 411 \\ 442 & 449 & 366 & - \end{pmatrix}$$

(5) The maximum quantity M_i transported from a factory $i \in F$ is:

$$\dot{\bigwedge}_{i \in F} \left(M_i = \max_{p \in P,\ j \in F} x_{p,i,j} \right) = \begin{pmatrix} 48 & 47 & 49 & 47 \end{pmatrix}$$

(6) For each product $p \in P$ the sum of the maximum quantity S_i transported from a factory $i \in F$ is:

$$\dot{\bigwedge}_{i \in F} \left(S_i = \max_{p \in P} \left(\sum_{j \in F} x_{p,i,j} \right) \right) = \begin{pmatrix} 130 \\ 125 \\ 133 \\ 123 \end{pmatrix}$$

2.3.4 Indexed index sets

Index sets occurring in indexing terms can be themselves indexed. Let us define an index set I and based on it an *indexed index set* $J = \{J_i\}$ where J_i is an index set and $i \in I$. The elements of J are index sets themselves. In this case the index mechanism is not applied to a singleton but to a set. As an example, suppose $I = \{a, b\}$, $J_a = \{1, 2\}$, and $J_b = \{2, 3\}$ then $J = \{\{1, 2\}, \{2, 3\}\}$. Based on these two sets I and J, one can form the following indexed notation:

$$\bigodot_{i \in I} \left(\bigodot_{j \in J_i} E_{i,j} \right) \quad \text{or} \quad \bigodot_{i \in I,\ j \in J_i} E_{i,j} \tag{7}$$

It is important to note, however, that the index i in $i \in I$ is an active index, whereas the i in J_i is a passive index. Now (7) can be interpreted as follows:

$$\bigodot_{(i,j) \in K} E_{i,j} \quad \text{with} \quad K \subseteq I \times (J_1 \cup \cdots \cup J_{|I|}) \tag{8}$$

This means that the concept of indexed index set can be reduced to compound index set and does not add any new features. Nevertheless, a notation like (7) may sometimes be more convenient, because the tuples are presented as a hierarchical structure instead of a flat list.

2.3.5 Ordering

We saw that in various context an index set has a natural order. If the set is a sequence of time periods or time points, for instance, then we order the elements within this set according to its natural sequence and we expect that the index mechanism is applied in this order. We take an example: Suppose P is a set of time periods or time points. Furthermore, x_p is the production level in period $p \in P$, d_p is the demand in period p, and s_p is the stock level at the end of period p. Then we can build a balance equation at each period:

$$\begin{gathered} \text{production level at p} - \text{demand at p} \\ = \text{stock at the end of p} - \text{stock at the beginning of p} \end{gathered}$$

Formally this can be written as follows:

$$x_p - d_p = s_p - s_{p-1} \quad \text{with} \quad p \in P$$

In this example we make a reference to s_{p-1}, that is, to the time period before p. This would have no meaning, if the set P were not ordered. However, if we make reference to a different element then we must be careful to check of whether the element exists. In the expression above there *is* an undefined reference: if $p = 1$ then the expression $s_{p-1} = s_0$ is not defined. Hence strictly speaking, the expression above is only valid for $p \in P - \{1\}$, we must exclude the first period and the balance equation of the first period is treated apart, or we must define s_0 separately.

3 Index Notation in LPL

LPL is a computer language that implements basically the index mechanism described. In this section, this implementation is presented. For a general reference of the LPL language see the Reference Manual (see [2]). LPL implements the indexed notation as close as possible to the mathematical notation. But there are some differences. By mean of several examples, we explain the relationship.

In an LPL code, one needs to specify the index sets first. They are available throughout the whole model code. This is done by a declaration that begins with a keyword `set`. It is followed by the name of the set and an assignment of the elements as follows:

```
set I := 1..10;
set J := 1..100;
set K := 1..n;
parameter n;
```

This declaration defines the three sets $I = \{1 \dots 10\}$, $J = \{1 \dots 100\}$ and a set $K = \{1 \dots n\}$. Note that the identifiers can be declared in any order in LPL. n can or cannot have a value.

This set declarations can be used in expressions. The following expressions:

$$A = \sum_{i \in I} 1 \quad , \quad B = \sum_{j \in J} j \quad , \quad C = \sum_{x \in J | 10 \leq x \leq 20} x^2$$

can be implemented directly as follows:

```
parameter A := sum{i in I} 1;
parameter B := sum{j in J} j;
parameter C := sum{x in J|x>=10 and x<=20} x^2;
```

The index operator $\sum$ is the keyword `sum`. The indexing term $i \in I$ is appended to the index operator and bracketed with `{` and `}`. We also need to declare new parameters `A`, `B`, and `C` that hold the calculated value. The indexing condition is a proper Boolean expression in the language. Hence $10 \leq x \leq 20$ must be formulated as `x>=10 and x<=20`. Note that we use `:=` as an assignment operator.

If we use tables such as v_i, $a_{i,j}$, and $b_{i,j,k}$ with $i \in I,\ j \in J,\ k \in K$, we must declare them first in the code as follows:

```
parameter v{i in I};
parameter a{i in I, j in J};
parameter b{i in I, j in J, k in K};
```

Using these declaration we now can build other expressions like:

$$D = \sum_{i \in I} v_i \quad , \quad E = \max_{i \in I} \left(\sum_{j \in J} a_{i,j} \right) \quad , \quad F_{k \in K} = \sum_{i \in I,\ j \in J} b_{i,j,k}$$

using the following syntax:

```
parameter D := sum{i in I} v[i];
parameter E := max{i in I} (sum{j in J} a[i,j]);
parameter F{k in K} := sum{i in I, j in J} b[i,j,k];
```

The passive indexes in the indexed expression are enclosed in the brackets `[` and `]`. So there is a clear syntactical distinction between the passive and the active indexes in the language.

In LPL there exist a slightly shorter syntax that could also be used for an indexing term. The examples above could be coded as follows:

```
set i := 1..10;
set j := 1..100;
parameter n;
set k := 1..n;

parameter A := sum{i} 1;
parameter B := sum{j} j;
parameter C := sum{j|j>=10 and j<=20} j^2;

parameter v{i};
parameter a{i,j};
parameter b{i,j,k};

parameter D := sum{i} v[i];
parameter E := max{i} (sum{j} a[i,j]);
parameter F{k} := sum{i,j} b[i,j,k];
```

In this code, the active index has *the same name* as the corresponding index set. Hence, `{i in I}`, an indexing term, can just be written as `{i}`. This makes the expressions more readable and – in most context – much shorter. However, this can lead to difficulties as soon as the same index set is used more than once in an indexed expression as in the following simple expression:

$$H = \sum_{i,j \in I} c_{i,j}$$

In such a case, LPL allows one to define one or two `alias` names for the same index set. We then could code this as following:

```
set i, j := 1..10;
parameter c{i,j};
parameter H := sum{i,j} c[i,j];
```

Of course, one always can use the more "traditional" notation as follows:

```
set I := 1..10;
parameter c{i in I, j in J}
parameter H := sum{i in I,j in I} c[i,j];
```

The alias names – separated by commas – are just other names for the index set and can be used as unique active indexes name in indexed notations. LPL allows at most two aliases for each index sets. If we use more than two, then we always need to use the longer syntax as in the following example:

$$L = \sum_{i,j,k,p \in I} d_{i,j,k,p}$$

in which case we need to code this as:

```
set i, j,k := 1..10;
parameter d{i,j,k,p in i};
parameter L := sum{i,j,k,p in i} d[i,j,k,p];
```

The two versions of the LPL code can be downloaded or executed directly at: exercise0a[2] and exercise0b[3]

[2] http://lpl.unifr.ch/lpl/Solver.jsp?name=/exercise0a
[3] http://lpl.unifr.ch/lpl/Solver.jsp?name=/exercise0b

Let us now code the **Exercise 1** given in section 2.3.3 as an LPL model. First we implement the declarations: The set of products and factories which are $P = \{1, 2, 3\}$ and $F = \{1, 2, 3, 4\}$ the cost table $c_{i,j}$, the value table v_p and the quantity table $x_{p,i,j}$ with $i, j \in F$ and $p \in P$. In LPL these data tables can be implemented as follows:

```
set p    := 1..3;
set i, j := 1..4;
parameter c{i,j|i<>j} := [. 1 5 2 , 2 . 4 2 , 1 2 . 3 , 1 3 1 .];
parameter v{p}        := [6 2 3];
parameter x{p,i,j|i<>j} := [
    . 48 37 45 36 . 43 46 44 36 . 33 36 39 34 . ,
    . 46 35 39 32 . 47 35 45 49 . 39 47 46 30 . ,
    . 32 32 40 30 . 41 30 45 43 . 45 44 41 34 .  ];
```

In table `c` and `x` we use the indexing condition `i<>j` because there is no cost or quantity defined form a factory to itself. The data itself are listed in a lexico-graphical order. The expressions given in section 2.3.3 are then formulated as follows in LPL:

```
parameter T     := sum{p,i,j} c[i,j]*x[p,i,j];
parameter V     := sum{p,i,j} v[p]*x[p,i,j];
parameter C{p}  := sum{i,j} c[i,j]*x[p,i,j];
parameter W{i,j}:= sum{p} v[p]*x[p,i,j];
parameter M{i}  := max{p,j} x[p,i,j];
parameter S{i}  := max{p} (sum{j} x[p,i,j]);
```

The complete model can be executed at: exercise1a[4].

Exercise 1 (continued):

Let us now add a requirement that the transportation can only take place between a small subset of all (i, j) links between the factories $i, j \in F$. Hence, we need to introduce a property $K(i, j)$ which is true if the transportation can take place between factory i and factory j. We define it as:

$$K_{i,j \in F} = \{(1,2) \quad (1,4) \quad (2,3) \quad (3,2) \quad (4,3)\}$$

The tuple list means that the transportation can only occur from factory 1 to 2, from 1 to 4, from 2 to 3, from 3 to 2 and from 4 to 3. Only 5 transportation links are allowed. The corresponding expressions are then as follows:

(1) The total transportation costs T is calculated as follows:

$$T = \sum_{p \in P,\ i,j \in F | K(i,j)} c_{i,j} \cdot x_{p,i,j} = 1524$$

(2) The total value transported is as follows:

$$V = \sum_{p \in P,\ i,j \in F | K(i,j)} v_p \cdot x_{p,i,j} = 2148$$

(3) The total transportation costs C_p for each product $p \in P$ is:

$$C_{p \in P} = \sum_{i,j \in F | K(i,j)} c_{i,j} \cdot x_{p,i,j} = \begin{pmatrix} 511 \\ 537 \\ 476 \end{pmatrix}$$

[4] http://lpl.unifr.ch/lpl/Solver.jsp?name=/exercise1a

(4) The value $W_{i,j}$ transported between all factories $(i,j) \in F \times F$ is:

$$W_{i,j\in F|K(i,j)} = \sum_{p\in P} v_p \cdot x_{p,i,j} = \begin{pmatrix} - & 476 & 388 & - \\ - & - & 475 & - \\ - & 443 & - & - \\ - & - & 366 & - \end{pmatrix}$$

(5) The maximum quantity M_i transported from a factory $i \in F$ is:

$$M_{i\in F} = \max_{p\in P,\ j\in F|K(i,j)} x_{p,i,j} = \begin{pmatrix} 48 & 47 & 49 & 34 \end{pmatrix}$$

(6) For each product $p \in P$ the sum of the maximum quantity S_i transported from a factory $i \in F$ is:

$$S_{i\in F} = \max_{p\in P} \left(\sum_{j\in F|K(i,j)} x_{p,i,j} \right) = \begin{pmatrix} 85 \\ 47 \\ 49 \\ 34 \end{pmatrix}$$

It is straightforward to code this in LPL. First we declare the table of property $K_{i,j}$ as following:

```
set k{i,j} := [ 1 2 , 1 4 , 2 3 , 3 2 , 4 3 ];
```

The property $K(i,j)$ is simply declared as a set. In fact, in LPL `k{i,j}` defines a tuple list which is a subset of the Cartesian Product of $(i,j) \in I \times I$. In LPL, we can code the expressions as follows:

```
parameter T    := sum{p,i,j|k[i,j]} c[i,j]*x[p,i,j];
parameter V    := sum{p,i,j|k[i,j]} v[p]*x[p,i,j];
parameter C{p} := sum{i,j|k[i,j]} c[i,j]*x[p,i,j];
parameter W{i,j|k[i,j]} := sum{p} v[p]*x[p,i,j];
parameter M{i} := max{p,j|k[i,j]} x[p,i,j];
parameter S{i} := max{p} (sum{j|k[i,j]} x[p,i,j]);
```

Since in LPL `k{i,j}` is itself a set, it can be used as an index set. Hence, the previous code could be written also as:

```
parameter T    := sum{p,k[i,j]} c[i,j]*x[p,i,j];
parameter V    := sum{p,k[i,j]} v[p]*x[p,i,j];
parameter C{p} := sum{k[i,j]} c[i,j]*x[p,i,j];
parameter W{k[i,j]} := sum{p} v[p]*x[p,i,j];
parameter M{i} := max{p,j|k[i,j]} x[p,i,j];
parameter S{i} := max{p} (sum{j|k[i,j]} x[p,i,j]);
```

The difference is in the first four expressions. We wrote `{p,k[i,j]}` instead of `{p,i,j|k[i,j]}`. These two indexing terms define the same set. Computationally, however, there can be a huge difference. While the first indexing term `{p,i,j|k[i,j]}` has cardinality of $|\ P \times I \times I\ |$ the second indexing term `{p,k[i,j]}` only has cardinality of $|\ P \times K(i,j)\ |$. The two version of LPL code can be downloaded or executed directly at: exercise1b[5] and exercise1c[6].

[5] http://lpl.unifr.ch/lpl/Solver.jsp?name=/exercise1b
[6] http://lpl.unifr.ch/lpl/Solver.jsp?name=/exercise1c

4 A Complete Problem: Exercise 2

In this second exercise we are going to explore the cost of a given production plan. Suppose we want to manufacture 7 different products. We have 4 identical factories where 5 different machines are installed. 20 different raw products are used to assemble a product. Hence, we have a set of products $P = \{A, B, C, D, E, F, G\}$, a set of factories $F = \{F1, F2, F3, F4\}$, a set of machines $M = \{M1, M2, M3, M4, M5\}$, and a set of raw materials $R = \{1 \dots 20\}$. Additionally, various tables are given with $p \in P,\ f \in F,\ m \in M,\ r \in R$: A table $q_{p,m}$ defines which product can be manufactured on which machine. The table $R_{r,p}$ says how many units of raw materials r are used to manufacture a unit of product p. The table $H_{p,m|q(p,m)}$ says how many hours of machine m it takes to manufacture one unit of product p. The table Rc_r gives the cost of a unit of raw materials, the table Mc_m gives the cost of machine m per hour, and the table S_p is the selling price of product p. Finally, the table $Q_{f,p,m|q(p,m)}$ is a given production plan. It says how many units products to manufacture on which machine and in which factory. (We do not discuss here whether this production plan is good or not.) We suppose that all products can be sold, and that a machine can only process one product at the same time. We do not consider any other costs. A concrete data set is given in the LPL code and at the end of this paper (see exercise2[7]).

We now wanted to calculate various amounts:

(1) The total selling value TS is:

$$TS = \sum_{f \in F,\ p \in P,\ m \in M | q(p,m)} S_p \cdot Q_{f,p,m} = 383646$$

(2) The total cost of raw material RC is:

$$RC = \sum_{f \in F,\ p \in P,\ m \in M,\ r \in R | q(p,m)} R_{r,p} \cdot Rc_r \cdot Q_{f,p,m} = 305882$$

(3) The total machine costs MC is:

$$MC = \sum_{f \in F,\ p \in P,\ m \in M | q_{p,m}} H_{p,m} \cdot Mc_m \cdot Q_{f,p,m} = 24882$$

(4) The total profit TS is:

$$TP = TS - RC - MC = 52882$$

(5) The total quantity TQ_p produced for each product p is:

$$TQ_p = \sum_{f \in F,\ m \in M | q_{p,m}} Q_{f,p,m} = \begin{pmatrix} 188 & 228 & 175 & 135 & 89 & 261 & 391 \end{pmatrix}$$

(6a) How long (in hours) does the production take on each machine $m \in M$ and in each factory $f \in F$?

[7] http://lpl.unifr.ch/lpl/Solver.jsp?name=/exercise2

$$Ti_{f\in F,m\in M} = \sum_{p\in P|q(p,m)} H_{p,m}\cdot Q_{f,p,m} = \begin{pmatrix} 515 & 312 & 542 & 290 & 330 \\ 548 & 224 & 554 & 324 & 273 \\ 416 & 392 & 309 & 154 & 246 \\ 472 & 216 & 695 & 154 & 452 \end{pmatrix}$$

(6b) How long (in hours) does the production take maximally until the last product leaves a factory $f \in F$?

$$TI_{f\in F} = \max_{m\in M}\left(\sum_{p\in P|q(p,m)} H_{p,m}\cdot Q_{f,p,m}\right) = \begin{pmatrix} 542 & 554 & 416 & 695 \end{pmatrix}$$

(7a) How big is the loss (gain) of each product ($Lo_{p\in P}$)?

$$Lo_{p\in P} = \sum_{f,m} S_p\cdot Q_{f,p,m} - \sum_{f,m}\left(\sum_r R_{r,p}\cdot Rc_r + H_{p,m}\cdot Mc_m\right)\cdot Q_{f,p,m}$$
$$= \begin{Bmatrix} 55792 & 33450 & -6320 & -19305 & 10769 & -3046 & -18458 \end{Bmatrix}$$

(7b) Which products generate a loss (set $LO_{p\in P}$)?

$$LO_{p\in P} = \sum_{f,m} S_p\cdot Q_{f,p,m} - \sum_{f,m}\left(\sum_r R_{r,p}\cdot Rc_r + H_{p,m}\cdot Mc_m\right)\cdot Q_{f,p,m} < 0$$
$$= \begin{Bmatrix} C & D & F & G \end{Bmatrix}$$

(8) Which machines in which factories work more than 500 hours (set $MO_{f\in F,\ m\in M}$)?

$$MO_{f,m} = \sum_p H_{p,m}\cdot Q_{f,p,m} > 500$$
$$= \begin{Bmatrix} (F1,M1) & (F1,M3) & (F2,M1) & (F2,M3) & (F4,M3) \end{Bmatrix}$$

(9) Which machine and in which factory works more than any other machine (set $MA_{f\in F,\ m\in M}$)? [8]

$$MA_{f,m} =$$
$$f = \operatorname*{argmax}_f\left(\sum_p H_{p,m}\cdot Q_{f,p,m}\right) \quad\wedge\quad m = \operatorname*{argmax}_m\left(\sum_p H_{p,m}\cdot Q_{f,p,m}\right)$$
$$= \begin{Bmatrix} (F4,M3) \end{Bmatrix}$$

(10) Which machine and in which factory has the least work to do (in hours) (set $MI_{f\in F,\ m\in M}$)?

[8] The operators argmax and argmin are two other index operators returning the element for which the indexed expression has the largest and the smallest value.

$$MI_{f,m} = \\ f = \underset{f}{\operatorname{argmin}} \left(\sum_p H_{p,m} \cdot Q_{f,p,m} \right) \quad \wedge \quad m = \underset{m}{\operatorname{argmin}} \left(\sum_p H_{p,m} \cdot Q_{f,p,m} \right) \\ = \{(F3, M4)\}$$

It is straightforward to implement this examples in the language LPL. First we declare the given index sets and the tables as follows (the concrete data for the parameter tables can be found in the LPL model):

```
set
  p := [A B C D E F G];      r := [1..20];
  m := [M1 M2 M3 M4 M5];     f := [F1 F2 F3 F4];
  q{p,m};
parameter
  Q{f,q};    S{p};    R{r,p};     H{q};     Rc{r};     Mc{m};
```

Based on these given data, we can then calculate the various expressions as follows[9]:

```
parameter
  TS := sum{f,q[p,m]} Q*S;
  RC := sum{f,p,m,r} R*Rc * Q;
  MC := sum{f,p,m} H*Mc * Q;
  TP := TS - RC - MC;
  Ti{f,m}:= sum{p} H*Q;
  TI{f}:= max{m} sum{p} Mc*Q;
  Lo{p} := sum{f,m} S*Q - sum{f,m}(sum{r} R*Rc + H*Mc)* Q;
set
  LO{p}  := sum{f,m} S*Q - sum{f,m}(sum{r} R*Rc + H*Mc)* Q < 0;
  MO{f,m}:= sum{p} H*Q > 300;
  MA{f,m}:= f=argmax{f}(sum{p} Mc*Q) and m=argmax{m}(sum{p} Mc*Q);
  MI{f,m}:= f=argmin{f}(sum{p} Mc*Q) and m=argmin{m}(sum{p} Mc*Q);
```

5 Data for Exercise 2

The data of Exercise 2 are as follows:

$$q^{10} = \begin{pmatrix} - & - & 1 & - & 1 \\ 1 & - & 1 & - & - \\ 1 & - & - & 1 & - \\ - & - & - & - & 1 \\ 1 & - & - & - & - \\ - & 1 & 1 & - & - \\ 1 & - & 1 & 1 & - \end{pmatrix} \qquad H = \begin{pmatrix} - & - & 8 & - & 8 \\ 4 & - & 7 & - & - \\ 7 & - & - & 2 & - \\ - & - & - & - & 3 \\ 5 & - & - & - & - \\ - & 8 & 2 & - & - \\ 3 & - & 2 & 2 & - \end{pmatrix} \qquad S = \begin{pmatrix} 491 \\ 391 \\ 156 \\ 120 \\ 221 \\ 257 \\ 184 \end{pmatrix}$$

[9] In LPL, it is quite common to leave out the passive indexes: this is very practical and shortens the expressions. However, this practice should only be used by advanced users.

[10] "1" in the matrix means that the combination is possible. For example $q_{B,M1} = 1$ means that product B can be manufactured on machine $M1$.

$$R = \begin{pmatrix} - & - & - & - & - & - & 1 \\ - & - & - & - & - & - & - \\ - & 8 & 7 & 6 & - & - & - \\ - & 6 & - & - & 5 & - & - \\ 7 & 8 & - & - & - & - & 5 \\ - & 4 & - & 6 & - & - & - \\ 7 & - & - & 3 & 4 & - & 4 \\ 5 & - & - & 8 & - & 7 & - \\ - & - & - & - & - & 8 & 2 \\ - & - & 3 & - & - & - & - \\ - & - & - & - & - & - & 4 \\ 5 & - & 6 & 4 & - & 7 & 6 \\ - & 2 & 5 & - & - & - & 8 \\ - & - & 4 & - & - & - & 5 \\ - & - & 5 & 3 & - & - & - \\ - & 1 & - & - & - & - & - \\ - & 5 & - & 1 & - & 6 & 6 \\ - & - & 5 & 4 & 8 & 6 & - \\ - & 4 & - & - & 3 & - & - \\ 4 & - & - & 8 & - & 5 & 3 \end{pmatrix} \qquad Rc = \begin{pmatrix} 5 \\ 6 \\ 7 \\ 3 \\ 7 \\ 7 \\ 3 \\ 5 \\ 8 \\ 4 \\ 4 \\ 8 \\ 4 \\ 2 \\ 3 \\ 8 \\ 5 \\ 6 \\ 5 \\ 6 \end{pmatrix} \qquad Mc = \begin{pmatrix} 2 \\ 3 \\ 5 \\ 2 \\ 4 \end{pmatrix}$$

$$Q_{f=F1} = \begin{pmatrix} - & - & 15 & - & 27 \\ 21 & - & 40 & - & - \\ 19 & - & - & 16 & - \\ - & - & - & - & 38 \\ 38 & - & - & - & - \\ - & 39 & 40 & - & - \\ 36 & - & 31 & 43 & - \end{pmatrix} \qquad Q_{f=F2} = \begin{pmatrix} - & - & 15 & - & 18 \\ 11 & - & 48 & - & - \\ 41 & - & - & 18 & - \\ - & - & - & - & 43 \\ 17 & - & - & - & - \\ - & 28 & 32 & - & - \\ 44 & - & 17 & 48 & - \end{pmatrix}$$

$$Q_{f=F3} = \begin{pmatrix} - & - & 10 & - & 24 \\ 16 & - & 15 & - & - \\ 24 & - & - & 20 & - \\ - & - & - & - & 18 \\ 11 & - & - & - & - \\ - & 49 & 31 & - & - \\ 43 & - & 31 & 19 & - \end{pmatrix} \qquad Q_{f=F4} = \begin{pmatrix} - & - & 36 & - & 43 \\ 34 & - & 43 & - & - \\ 23 & - & - & 14 & - \\ - & - & - & - & 36 \\ 23 & - & - & - & - \\ - & 27 & 15 & - & - \\ 20 & - & 38 & 21 & - \end{pmatrix}$$

6 An Optimizing Problem Version

In the previous Exercise 2, several tables have been given to specify a concrete production plan. They are as follows:

1. Given 7 different products ($p \in P$)
2. Manufactured in 4 identical factories ($f \in F$)

3. By means of 5 identical machines in each factory ($m \in M$)
4. Using 20 different raw products ($r \in R$)
5. A table $q_{p,m}$ defining which product p can be manufactured on which machine m
6. A table $R_{r,p}$ saying how many units of raw materials r are used to manufacture a unit of product p
7. A table $H_{p,m|q(p,m)}$ defining the hours of machine m that is taken to manufacture one unit of product p
8. A table Rc_r giving the cost of a unit of raw materials r
9. A table Mc_m specifying the cost of machine m per hour of work
10. A table S_p giving the selling price of product p
11. Finally, a table $Q_{f,p,m|q_{p,m}}$ saying how many unit of products p are manufactured in factory f by means of machine m.

Especially, the last table Q is a given table that specifies the *production plan*. It is a given arbitrary plan without asking the question whether this plan is *optimal* in any sense. Let us now ask the question whether this plan could be changed without changing the manufactured quantity of each product, just by switching the production quantity from one machine to another.

6.1 Version 1: the easy problem

Let's take a concrete example: The present production plan say to manufacture 40 units of the second product (B) in factory $F1$ with machine $M3$, that is we have: $Q_{F1,B,M3} = 40$. These 40 units could also have been manufactured by machine $M1$, such that $Q_{F1,B,M3}$ becomes 0 and $Q_{F1,B,M1}$ becomes 61. Doing this, implies that the total machine costs MC would now be reduced to:

$$MC = \sum_{f \in F,\ p \in P,\ m \in M|q(p,m)} H_{p,m} \cdot Mc_r \cdot Q_{f,p,m} = 23802$$

instead of 24882, which is a cost reduction of 1080 units. We could use probably other reassignment of the production plan in order to reduce the cost even further. However, we would like to do this not on an experimental level, but on a systematic way. Hence, the question is: How should the production plan be, in order to minimize the total machine costs? (Note that the raw material costs do not change, supposing that the quantities of products manufactured do not change.) Expressed in another way: how should the production plan, that is the matrix $Q_{f,p,m}$, be assigned in order to make the machine costs as small as possible.

At this point, thing become interesting: From the mathematical point of view, we now have a model in which some data are unknown, that is the matrix $Q_{f,p,m}$ becomes an unknown, because we want to know how this matrix has to be filled to attain our goal (the minimizing of costs). Our model is as follows:

1. Let us introduce a new variable (that is the unknown matrix) to represent the new production plan as follows: $QQ_{f,p,m|q_{p,m}} \geq 0$, where $QQ_{f,p,m}$ is the (unknown) quantity of product p to be manufactured at factory f by the means of the machine m

2. We would like to minimize the total machine costs

3. We also would like to produce the same quantity of each product as in the previous plan

There three conditions could be translated into the following mathematical model. This model has to be solved to find the minimizing cost production plan. The model is a s follows:

$$\min \sum_{f \in F,\ p \in P,\ m \in M | q(p,m)} H_{p,m} \cdot Mc_r \cdot QQ_{f,p,m}$$

subject to

$$\sum_{f \in F,\ m \in M | q_{p,m}} QQ_{f,p,m} = TQ_p \quad , \quad \texttt{forall}\ p \in P$$

The formulation of this model in LPL is as following (see exercise2a[11] for a complete code):

```
....
integer variable QQ{f,q};
constraint CA{p}: sum{f,m} QQ = sum{f,m} Q;
minimize obj1: sum{f,p,m} H*Mc * QQ;
....
```

The new assigned (optimal, that is "cost minimal") production plan QQ is as follows:

$$Q_{f=F1} = \begin{pmatrix} - & - & - & - & 188 \\ 228 & - & - & - & - \\ - & - & - & 175 & - \\ - & - & - & - & 135 \\ 89 & - & - & - & - \\ - & - & 261 & - & - \\ 391 & - & - & - & - \end{pmatrix} \qquad Q_{f=F2} = 0, \quad Q_{f=F3} = 0, \quad Q_{f=F4} = 0$$

And the total machine costs are:

$$MC = \sum_{f \in F,\ p \in P,\ m \in M | q_{p,m}} H_{p,m} \cdot Mc_r \cdot QQ_{f,p,m} = 16006$$

This is a substantial machine cost reduction of more than 35% compared to the initial (somewhat arbitrary) production plan of Exercise 2 (from 24882 units to 16006 units).

However, there is no need to use the mathematical models approach to find this trivial result! We could have found it by a simple thought: "One should assign as much work as possible to the cheapest machine!" For example, machine $M4$ only costs 4 to manufacture one unit of product C, while machine $M1$ costs 14, the only alternative to produce C.

[11] http://lpl.unifr.ch/lpl/Solver.jsp?name=/exercise2a

Furthermore it is not important in which factory the product is manufactured, since they are all identical. Hence, the total quantity 175 of product C should be manufactured by the means of machine $M4$. The idea is similar for the other products.

To find the cheapest unit cost for each product and the machine that generates this, we can use two indexed expressions as follows:

(1) Calculate the cheapest amount of machine cost to manufacture a particular product p (Table Cw_p).

$$Cw_{p\in P} = \min_{m|q_{p,m}} H_{p,m} \cdot Mc_m$$

$$= \{32 \quad 8 \quad 4 \quad 12 \quad 10 \quad 10 \quad 6\}$$

(2) What is the machine m that manufactures the cheapest product p (Table $CW_{p,m}$) ?

$$CW_{p,m} = (m = argmin_{m1\in M|q_{p,m1}} H_{p,m1} \cdot Mc_{m1}$$

$$= \{(A,M5) \quad (B,M1) \quad (C,M4) \quad (D,M5) \quad (E,M1) \quad (F,M3) \quad (G,M1)\}$$

Nevertheless, there is a problem with this solution. It might be the cost minimizing production plan, however the occupation time of the 5 machine in factory $F1$ is now as follows:

$$Ti_{f\in F,m\in M} = \sum_{p\in P|q(p,m)} H_{p,m} \cdot QQ_{f,p,m} = \begin{pmatrix} 2530 & 0 & 522 & 350 & 1909 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Machine $M1$ in factory $F1$ is heavily occupied while all machines in the other factories are idle! (Well, we might evenly distribute the production between the four factories by dividing the quantities $QQ_{f,p,m}$ by four. If the division results not in integers, then we shall round up two and round down two.) Nevertheless, some machines are idle and others are heavily occupied.

6.2 Version 2: Limit Capacity

At this point the problem is getting interesting if we add capacity requirements. Suppose that all machines work 10 hour per days, 5 days a week and we would like to manufacture all products within 10 weeks. Is this possible? This problem is all but easy to solve. In fact, we might use the same model as before with the additional constraints that no machine should work more than 500 hours and still we want to minimize costs.

This constraint can be formulated as follows:

$$\sum_{p\in P} H_{p,q} \cdot QQ_{f,p,m} \leq 500 \quad , \quad \texttt{forall}\ m \in M, f \in F$$

The formulation of this model in LPL is as following (see exercise2b[12] for a complete code):

[12] http://lpl.unifr.ch/lpl/Solver.jsp?name=/exercise2b

```
....
integer variable QQ{f,q};
constraint CA{p}: sum{f,m} QQ = sum{f,m} Q;
           CB{m,f}: sum{p} H*QQ <= 500;
minimize obj1: sum{f,p,m} H*Mc * QQ;
....
```

The new assigned (optimal, that is "cost minimal") production plan QQ is now as follows:

$$Q_{f=F1} = \begin{pmatrix} - & - & - & - & 62 \\ 124 & - & - & - & - \\ - & - & - & - & - \\ - & - & - & - & - \\ - & - & - & - & - \\ - & - & - & - & - \\ 1 & - & - & - & - \end{pmatrix} \qquad Q_{f=F2} = \begin{pmatrix} - & - & - & - & 62 \\ 89 & - & - & - & - \\ - & - & - & 175 & - \\ - & - & - & - & 1 \\ - & - & - & - & - \\ - & - & 250 & - & - \\ 48 & - & - & - & - \end{pmatrix}$$

$$Q_{f=F3} = \begin{pmatrix} - & - & - & - & 12 \\ 1 & - & - & - & - \\ - & - & - & - & - \\ - & - & - & - & 134 \\ 1 & - & - & - & - \\ - & - & 11 & - & - \\ 163 & - & - & - & - \end{pmatrix} \qquad Q_{f=F4} = \begin{pmatrix} - & - & - & - & 52 \\ 14 & - & - & - & - \\ - & - & - & - & - \\ - & - & - & - & - \\ 88 & - & - & - & - \\ - & - & - & - & - \\ - & - & 179 & - & - \end{pmatrix}$$

The following table give the occupation of each machine with this production plan:

$$Ti_{f\in F,m\in M} = \sum_{p\in P|q(p,m)} H_{p,m} \cdot QQ_{f,p,m} = \begin{pmatrix} 499 & 0 & 0 & 0 & 496 \\ 500 & 0 & 500 & 350 & 499 \\ 498 & 0 & 22 & 0 & 498 \\ 496 & 0 & 358 & 0 & 416 \end{pmatrix}$$

We see that no machine exceeds the occupation time of 500 hours. All products will be manufactured within 10 weeks.

Still some machine are not at their capacity limit and some are still idle. Could we reduce the overall production time and to what extend? The most simple way to do this is to experiment with the maximal occupation time. Very quickly we find that we can reduce that time up to 330 hours, which means that the production can be terminated within less than 7 weeks. The corresponding production plan now is as follows:

$$Q_{f=F1} = \begin{pmatrix} - & - & 33 & - & 41 \\ 3 & - & - & - & - \\ - & - & - & - & - \\ - & - & - & - & - \\ 63 & - & - & - & - \\ - & 41 & - & - & - \\ 1 & - & 32 & 55 & - \end{pmatrix} \qquad Q_{f=F2} = \begin{pmatrix} - & - & 41 & - & - \\ 80 & - & - & - & - \\ - & - & - & 9 & - \\ - & - & - & - & 110 \\ 2 & - & - & - & - \\ - & 38 & - & - & - \\ - & - & 1 & 50 & - \end{pmatrix}$$

$$Q_{f=F3} = \begin{pmatrix} - & - & - & - & 38 \\ 80 & - & 10 & - & - \\ - & - & - & - & - \\ - & - & - & - & 6 \\ 2 & - & - & - & - \\ - & 41 & 99 & - & - \\ - & - & 30 & 55 & - \end{pmatrix} \qquad Q_{f=F4} = \begin{pmatrix} - & - & - & - & 34 \\ 54 & - & - & - & - \\ - & - & 165 & - & - \\ - & - & - & - & 18 \\ 22 & - & - & - & - \\ - & 41 & - & - & - \\ - & - & 165 & - & - \end{pmatrix}$$

The following table give the occupation of each machine with this production plan:

$$Ti_{f\in F,m\in M} = \sum_{p\in P|q(p,m)} H_{p,m} \cdot QQ_{f,p,m} = \begin{pmatrix} 330 & 328 & 330 & 330 & 328 \\ 330 & 304 & 330 & 326 & 330 \\ 330 & 328 & 330 & 330 & 330 \\ 330 & 328 & 330 & 330 & 329 \end{pmatrix}$$

As we can see, all machine work almost at 100% of 330 hours. However, the total machine costs are now 21004, 4998 units over the minimal costs of 16006, found in the previous model. This is the price we have to pay for a more balanced machine occupation.

6.3 Version 3: with Setup Costs

Of course, several important aspects of a real problem have not been considered. For example, we did not take into account setup costs, that is, the costs (or time) to switch the production from one to another product on a particular machine. Suppose that the setup costs to switch production from one product to another on any machine are constant for all machines and all products: 7 units. What is now the optimal production plan?

To model this question we need to "count" the numbers of setups. In the last production plan, for example, we would have to add 3 times a setup cost of 7 for the first machine $M1$ in first factory $F1$, etc. However, the number of setups is itself a variable and we cannot count it "afterwards". The best way is to introduce a "indicator" variable $y_{f,p,m|q_{p,m}} \in \{0,1\}$ (which is zero or one) for each $QQ_{f,p,m|q_{p,m}}$ quantity that can be produced. Whenever $QQ_{f,p,m|q_{p,m}}$ is strictly greater than zero then $y_{f,p,m|q_{p,m}}$ must be one. This can be formulated as follows:

$$QQ_{f,p,m} > 0 \quad \rightarrow \quad y_{f,p,m} = 1$$

$$\texttt{forall } f \in F, p \in P, m \in M \quad \texttt{with } (p,m) \in q_{p,m}$$

This is not a mathematical linear function and we need to reformulate it as a proper mathematical constraint in order to use standard general solvers. Fortunately, it is easy to translate this constraint into a linear now as follows (where U is a large number):

$$QQ_{f,p,m} \leq U \cdot y_{f,p,m}$$

$$\texttt{forall } f \in F, p \in P, m \in M \quad \texttt{with } (p,m) \in q_{p,m}$$

To see that these two constraints express the same condition, we only need to check that y is one if QQ is strictly positive. However, this is always the case, hence the two expressions express the same constraint.

We now have to add all setup costs to the objective function. This function now changed to:

$$\min \sum_{f \in F,\ p \in P,\ m \in M | q(p,m)} (H_{p,m} \cdot Mc_r \cdot QQ_{f,p,m} + Setup_p \cdot y_{f,p,m})$$

The formulation of this model in LPL is as following (The value of U has been fixed to 400 and the total machine occupation has slightly been increased to 337):

```
....
integer variable QQ{f,q};
binary  variable y{f,q};
constraint CA{p}: sum{f,m} QQ = sum{f,m} Q;
           CB{m,f}: sum{p} H*QQ <= 337;
           CC{f,q}: QQ <= 400*y;
minimize obj1: sum{f,p,m} H*Mc * QQ + Setup*y;
....
```

The solution can be look up by running the model exercise2c[13].

7 Conclusion

This paper gave a comprehensive overview of the indexed notation in mathematical modeling. The general notation is defined and several "extensions" have been explained. It is fundamental to understand, read and write mathematical models. We then gave some examples and how they are implemented in the modeling language LPL. Finally, a complete production model with several variants have been given in order to exercise the mathematical notation on a concrete example.

References

[1] R.L. Graham, D.E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison–Wesley Publ. Comp, second edition, 1994.

[2] T. Hürlimann. Reference Manual for the LPL Modelling Language, most recent version. www.virtual-optima.com.

[3] T. Hürlimann. *Mathematical Modelling and Optimization – An Essay for the Design of Computer-Based Modelling Tools*. Kluwer Academic Publ., Dordrecht, 1999.

[13] http://lpl.unifr.ch/lpl/Solver.jsp?name=/exercise2c