Journal of Mathematical Chemistry 18 (1995) 37–72

Definition and quantum chemical applications of nested summation symbols and logical functions: Pedagogical artificial intelligence devices for formulae writing, sequential programming and automatic parallel implementation

Ramon Carbó and Emili Besalú

Institute of Computational Chemistry, University of Girona, Albereda 3-5, 17071 Girona, Spain

Received 27 February 1995

1. Introduction

In this paper two new kinds of mathematical symbols with application to several aspects of computational chemistry, related to programming techniques and artificial intelligence, are described. The first one constitutes a family of symbols, related to logical expressions. They are referred to here as *logical functions* (LFs). The other symbol described in this paper is called a *nested summation symbol* (NSS). We insist on the quantum chemical usefulness of these symbols, so, application examples of the NSS graphism over some computational quantum chemistry topics are given. The results show how some standard formulae become shorter and easier to be written, generalized, programmed and evaluated. In some NSS expressions and other mathematical applications, the use of such symbols makes the whole problem formulation compact and elegant. Moreover, both symbols constitute a powerful link between mathematical formalism and programming in high level languages: Essentially, NSS are connected with *general nested do loops* (GNDL) structures, a general programming concept developed in our Laboratory, and LFs are related to *general logical if sentences*. The pedagogical potential of both mathematical devices is also revealed.

Through our research path on some quantum chemical topics, we have been facing the need to obtain transparent useful mathematical expressions, based on elementary mathematical concepts, see for example ref. [1]. We asked ourselves many times about the ability of such a reformulation to exhibit, if possible, the attractive of being simple, computationally useful, elegant, general, pedagogical

and susceptible of immediate translation to any high level programming language.

Even if only some parts of such a conceivable mathematical ideal scheme could be obtained, quantum mechanical, computational and pedagogical theoretical structures may obtain an important profit. Also, because mathematical symbols will be ineluctably involved in the new ambitious framework, other scientific areas as computational chemistry and physics or applied mathematics in general can be benefitted too.

In our intention was, with high priority, implicit the condition that the final scheme, if feasible, can serve as a tool to build up a convenient bridge between mathematical general formulae writing and computationally valid general program structures. Thus, programming techniques can also be assisted by means of this process, also artificial intelligence algorithms may use the results of our ideal outline in order to increase the performances of formulae generation and translation programs.

With all this conditioning principles in mind, the present paper tries to describe in the first place the definition and properties of two fundamental symbols: *nested summation symbols* and *logical functions*.

The usefulness of both symbolic concepts is shown nest. Use of the NSS in the area of quantum chemistry is illustrated by means of several assorted problems, revisited by means of the formalism generated by the symbols we promote. Various mathematical examples and other related topics are given in the appendices in order to illustrate the unlimited power of the symbolic concepts presented here.

Both symbols are related to the primitive intentions we have stated initially. The authors hope these symbolic forms turn out to be as useful to the scientific community as they had been in the development of their quest of a valid computational scheme based on PC machinery, whose main features had been already explained by one of the authors, see for example ref. [2].

2. Framework of nested summation symbols

2.1. LOGICAL VECTOR DEFINITION

Let us define an n-dimensional vector whose elements are logical expressions $\boldsymbol{L} = \{L_i\}_{i=1,n}$. In this paper a vector of this kind will be called a *logical vector* (LV).

2.2. LOGICAL FUNCTION DEFINITION

Consider a Boolean function $\mathcal{L}(\boldsymbol{L})$ which has as argument a LV. A function of this kind will be called a *logical function* (LF). A LF returns a value of 1 (true) or 0 (false).

2.3. LOGICAL KRONECKER DELTA SYMBOL

In many computational algorithms or mathematical applications, in order to obtain compact and general formulae, it is needed a symbol such as the well known Kronecker delta, δ_{ij}, but involving other logical relationships than the equality between the indices i and j. One can easily define symbols which are far more general than the Kronecker delta. Here it is described as the logical Kronecker delta (LKD) symbol.

A LKD is a LF which can be simply defined as

$$\delta(L) = \begin{cases} 1 & \text{if } L \text{ is true}, \\ 0 & \text{if } L \text{ is false}, \end{cases} \tag{1}$$

L being any logical expression, that is, a monodimensional LV. The origin of this definition can be found in a primitive description of Carbó and Hernández [3], related to some research made on a MC SCF procedure.

In fact, the classical Kronecker delta symbol is nothing but a particular case of the LKD, that is: $\delta_{ij} = \delta([i = j])$, where the logical expression is obviously $L \equiv [i = j]$.

This mathematical symbol has an immediate translation in any high level language: It may be related to any kind of universal *logical if* statement.

2.4. GENERALIZED LOGICAL KRONECKER DELTA SYMBOL

A generalized logical Kronecker delta (GLKD) symbol is a LF which is defined as

$$\Delta(\boldsymbol{L}) = \begin{cases} 1 & \text{if } L_i \text{ is true } \forall i = 1, n, \\ 0 & \text{if } \exists i | L_i \text{ is false}, \end{cases} \tag{2}$$

where $\boldsymbol{L}$ is a LV.

It must be noted that a GLKD symbol can be substituted by a unique LKD symbol, using a logical *and* ($\bigwedge$) operator n times over the elements of the LV $\boldsymbol{L}$:

$$\Delta(\boldsymbol{L}) = \prod_{i=1}^{n} \delta(L_i) = \delta\left(\bigwedge_{i=1}^{n} L_i\right). \tag{3}$$

The GLKD symbol returns the unit value iff all the elements of the LV are true. A zero value is generated in any other circumstance.

Once the LKD and GLKD functions are described, other logical functions can be defined as more or less complex successive applications or manipulations of simple LKDs. They are discussed in appendix A.

2.5. NESTED SUMMATION SYMBOL [4]

Let us define an operator like

$$\sum_n(\boldsymbol{j}=\boldsymbol{i},\boldsymbol{f},\boldsymbol{s},\mathcal{L}(\boldsymbol{L}))\,, \tag{4}$$

and let us call it a *nested summation symbol*. The arguments in eq. (4) are vectors, except the last one, which is a LF with a LV argument. Index n is an integer called the *dimension of the NSS* and it signals the dimension of the involved index vectors $\boldsymbol{j}$, $\boldsymbol{i}$, $\boldsymbol{f}$ and $\boldsymbol{s}$. All these vectors together with the LF and the integer n, can be called the *parameters of the NSS*. The elements of the LV, $\boldsymbol{L}$, are supposed to be depending on the parameters of the NSS.

The meaning of such a convention about NSS corresponds to accomplish all the sums involved in the generation of all the possible forms of *index vector* $\boldsymbol{j}$. The allowed forms of this vector arise from the fulfillment of the two following rules:

1. The elements of the vector $\boldsymbol{j}$ are defined by the limits

$$\{i_k \leqslant j_k \leqslant f_k, \text{if } s_k \geqslant 0\} \quad \text{or} \quad \{i_k \geqslant j_k \geqslant f_k, \text{if } s_k \leqslant 0\}; \quad \forall k = 1,n\,, \tag{5}$$

   where the j_k indices can be incremented or decremented respectively in steps of length s_k.

2. All the possible forms of vector $\boldsymbol{j}$ are attached to the LF $\mathcal{L}$, in such a way that, when developing the NSS, only those vectors $\boldsymbol{j}$ that associated to a true value of $\mathcal{L}$ are taken into account.

So, the NSS stands for a set of summation symbols numbered from 1 to n and where the kth summation symbol is attached to the corresponding kth index of vectors $\boldsymbol{j}$, $\boldsymbol{i}$, $\boldsymbol{f}$ and $\boldsymbol{s}$.

In order to be more explicit, one can define a NSS as the following operator:

$$\sum_n(\boldsymbol{j}=\boldsymbol{i},\boldsymbol{f},\boldsymbol{s},\mathcal{L}(\boldsymbol{L}))\boldsymbol{A}(\boldsymbol{j}) \equiv \sum_{j_1=i_1}^{f_1(s_1)} \sum_{j_2=i_2}^{f_2(s_2)} \cdots \sum_{j_n=i_n}^{f_n(s_n)} \mathcal{L}(\boldsymbol{L})\boldsymbol{A}(\boldsymbol{j})\,, \tag{6}$$

where $\sum_{j_k=i_k}^{f_k(s_k)}$ means that $j_k = i_k,\ i_k + s_k,\ i_k + 2s_k,\ \ldots,\ f_k$. Thus, the NSS generates all vectors $\boldsymbol{j}$ whose elements fulfill the condition

$$\forall k = 1,n, \quad \{j_k = i_k + qs_k, q = 0,1,\ldots,Q_k\}\,, \tag{7}$$

where $Q_k = int((f_k - i_k)/s_k)$ and is defined only for non-negative arguments for the *integer* function.

For example, if the LF returns always a value of true (1), $n = 2$, $\boldsymbol{i} = (1, 2)$, $\boldsymbol{f} = (4, 0)$ and $\boldsymbol{s} = (3, -2)$, then

$$\sum_n(\boldsymbol{j}=\boldsymbol{i},\boldsymbol{f},\boldsymbol{s},1)\boldsymbol{A}(\boldsymbol{j}) = \boldsymbol{A}(1,2) + \boldsymbol{A}(1,0) + \boldsymbol{A}(4,2) + \boldsymbol{A}(4,0)\,. \tag{8}$$

In general, the number of terms in a nested sum is equal to

$$N = \prod_{k=1}^{n}(Q_k + 1)\,. \tag{9}$$

The mathematical usefulness, constituting the most conspicuous application potential of a NSS, can be easily recognized when the particular characteristic of these symbolic units is analyzed: The involved vector parameters, belonging to the nested sum, could be chosen with *arbitrary and variable dimensions*. In the following sections it will be shown that there are many scientific and mathematical formulae which will benefit from this property, when written in a paper or computationally implemented.

Other properties and the simplified notation of NSSs are described below. From now on, some of the simplifications discussed there will profusely appear in the text. The aspects related to the computational implementation of NSSs are described below and in appendix B.

Even more general features can be envisaged, supposing the NSS parameter vectors are real, for example, but they will not be discussed here. More information related to NSSs can be found in ref. [5].

2.6. PROPERTIES OF THE NSS

Some particularly interesting properties concerning NSS will be discussed next.

(a) NSSs can be recognized as *linear operators* with respect to any general expression placed inside the nested sum and depending on the index vector attached to the symbols:

$$\sum_n(\boldsymbol{j}=\boldsymbol{i},\boldsymbol{f},\boldsymbol{s},\mathcal{L}(\boldsymbol{L}))(\alpha F(\boldsymbol{j})+\beta G(\boldsymbol{j}))$$

$$=\alpha\sum_n(\boldsymbol{j}=\boldsymbol{i},\boldsymbol{f},\boldsymbol{s},\mathcal{L}(\boldsymbol{L}))F(\boldsymbol{j})+\beta\sum_n(\boldsymbol{j}=\boldsymbol{i},\boldsymbol{f},\boldsymbol{s},\mathcal{L}(\boldsymbol{L}))F(\boldsymbol{j})\,, \tag{10}$$

where $F(\boldsymbol{j})$ and $G(\boldsymbol{j})$ are arbitrary functions of the index values of the vector $\boldsymbol{j}$ and α and β scalars.

(b) It has to be taken into account that a product of two NSSs of dimension n and m is another NSS of dimension $n+m$, or:

$$\sum_n(\boldsymbol{j}=\boldsymbol{i},\boldsymbol{f},\boldsymbol{s},\mathcal{L}(\boldsymbol{L}))\sum_m(\boldsymbol{j}'=\boldsymbol{i}',\boldsymbol{f}',\boldsymbol{s}',\mathcal{L}(\boldsymbol{L}'))$$

$$=\sum_{n+m}(\boldsymbol{j}\oplus\boldsymbol{j}'=\boldsymbol{i}\oplus\boldsymbol{i}',\boldsymbol{f}\oplus\boldsymbol{f}',\boldsymbol{s}\oplus\boldsymbol{s}',\mathcal{L}'(\boldsymbol{L}\oplus\boldsymbol{L}'))\,, \tag{11}$$

where the new index vectors are constructed by the direct sum of the original vectors appearing in the product (11). Note that, in this general case, the LF should be redefined.

(c) The NSS $\sum_n(\boldsymbol{j}=\boldsymbol{i},\boldsymbol{f},\boldsymbol{s},\mathcal{L}(\boldsymbol{L}))$ is equivalent to the product $\sum_n(\boldsymbol{j}=\boldsymbol{i},\boldsymbol{f},\boldsymbol{s})\mathcal{L}(\boldsymbol{L})$ taking into account the comment number 2 of section 2.5.

(d) The symbol $\sum_0(\boldsymbol{j}=\boldsymbol{i},\boldsymbol{f},\boldsymbol{s},\mathcal{L}(\boldsymbol{L}))$ can be made equivalent, by convention, to the unit operator.

(e) The classical summation symbol $\sum_{j=i}^{f}$ is a particular case of the nested summation one. It can be rewritten by means of the symbol $\sum_1(j = i, f, 1, 1)$, where the LF returns a constant value of true (1).

(f) The so-called Einstein's convention, by which a set of nested sums are overridden from an expression, corresponds to omit a NSS like: $\sum_n(\boldsymbol{j} = \mathbf{1}, m\mathbf{1})$.

2.7. SIMPLIFIED NSS NOTATION

Despite the general form adopted here to write a NSS, sometimes it is superfluous to explicit all the involved parameters. When this circumstance does occur, some parts of the general form can be dropped in an arbitrary manner. The most important cases are:

(a) When the LF is omitted, it will mean that all the possible forms of the vector $\boldsymbol{j}$ have to be generated without restriction, except by the limits and rule imposed by vectors $\boldsymbol{i}, \boldsymbol{f}$ and $\boldsymbol{s}$ by means of eq. (5).

(b) When the step vector increment is irrelevant, obviously only the initial and final vector index parameters need to be explicitly written. In this case, the $\sum_n(\boldsymbol{j} = \boldsymbol{i}, \boldsymbol{f}, \mathcal{L}(\boldsymbol{L}))$ notation can be employed. Frequently, the step vector $\boldsymbol{s}$ is an n-dimensional vector such as $\boldsymbol{s} = \mathbf{1} = (1, 1, \ldots, 1)$.

(c) The same can be said of the final parameter vector $\boldsymbol{f}$, which may be a product by a scalar of vector $\mathbf{1}$. As an example, the notation $\sum_n(\boldsymbol{j} = \boldsymbol{1}, m\mathbf{1}, \mathbf{1}, \mathcal{L}(\boldsymbol{L}))$ displays the symbol which is constructed by n nested sums, whose indices have the same values within each sum in the interval $\{1, m\}$, each sum increment being unit everywhere. When writing NSS made of infinite power series the final parameter $\boldsymbol{f}$ may have all the elements infinite, then this situation can be written by means of the convention $\boldsymbol{f} \equiv \infty\mathbf{1}$.

(d) When initial, final, increment values and LF are implicit in the nested sum, a simplified symbol such as $\sum_n(\boldsymbol{j})$ may be also used.

(e) When the vector dimension n is obvious, then the subscript n can be omitted from the sum, as in $\sum(\boldsymbol{j} = \boldsymbol{i}, \boldsymbol{f}, \boldsymbol{s}, \mathcal{L}(\boldsymbol{L}))$, for instance.

(f) Combinations of the previous simplifications can also be used.

Some of this simplification conventions will be used throughout all the remaining text.

2.8. TWO USEFUL DEFINITIONS: A PARTICULAR KIND OF LV AND THE SUM OF THE ELEMENTS OF A VECTOR

Although the elements of the LV $\boldsymbol{L}$, appearing in NSSs in section 2.5 can be arbi-

trarily constructed, in this paper the following specific definition for the LV will be used several times. It corresponds to defining the LV elements:

$$\boldsymbol{L} = \boldsymbol{L}(\boldsymbol{j}, [\lambda]) = \{L_{pq} = [j_p \ .\lambda.\ j_q];\ p<q; p, q = 1, n\}\,, \tag{12}$$

where $.\lambda.$ stands for an arbitrary operator as, for example, *ne*($\neq$), *and* ($\wedge$), ... In this manner the expression $j_p \ .\lambda.\ j_q$ stands for a numerical relationship or a logical expression.

Also, another practical definition will be used throughout the text. It corresponds to the sum of the elements of an n-dimensional vector $\boldsymbol{v} = (v_1, v_2, \ldots, v_n)$, which is symbolized by $\langle \boldsymbol{v} \rangle$:

$$\langle \boldsymbol{v} \rangle = \sum_{i=1}^{n} v_i\,. \tag{13}$$

2.9. COMPUTATIONAL ALGORITHM. PARALLELISM.

Nested summation symbolism constitutes a perfect link between mathematical formalism and program implementation techniques, because successive generation of j index vector elements can be programmed in a general but simple way under any high level language. This can be achieved using a **unique** *do* or *for* loop statement construct, which is *general and independent of the dimension of the involved nested sums*. This kind of programming structure, when applied in any arbitrary computational environment, can be called a *general nested do loop* (GNDL).

```
      Parameter (n=?)    !   Dimension of the NSS
      Integer j(n),i(n),f(n),s(n)

* < Initial parameter values >

      do k=1,n
         i(k)=?
         f(k)=?
         s(k)=?
         j(k)=i(k)
      end do

* < GNDL procedure >

      k=n
      do while (k.gt.0)
         if ((j(k)-f(k))*s(k).gt.0) then
            j(k)=i(k)
            k=k-1
         else
            if (LFUN(n,j,i,f,s)) call Application(n,j)
            k=n
         end if
         if (k.gt.0) j(k)=j(k)+s(k)   ! Step
      end do
      END
```

Program 1. Fortran codification of the GNDL implementing a NSS like: $\sum_n(\boldsymbol{j} = \boldsymbol{i}, \boldsymbol{f}, \boldsymbol{s}, \mathcal{L}(\boldsymbol{L}))$.

GNDL are well suited devices for programming in sequential or parallel hardware and software as it will be shown now.

In order to prove in a practical manner the previous claims, program 1 represents a simple Fortran source code, generating all the j index vectors appearing in a general NSS. This algorithm corresponds to the implementation of a GNDL structure.

Program 1 is the codification of the practical implementation of a GNDL which represents a NSS of the type $\sum_n(\boldsymbol{j} = \boldsymbol{i}, \boldsymbol{f}, \boldsymbol{s}, \mathcal{L}(\boldsymbol{L}))$. The step vector $\boldsymbol{s}$ can have positive or negative indices. The values of the parameters depend on the particular application one wants when running the algorithm. So, they are not explicitly specified and a question mark (?) stands for its values. The code takes into account the use of a LV: it is assumed there exists a LF called *LFUN* which has several arguments (the parameters of the NSS or others) and returns a true value if all the logical expressions constituting the vector $\boldsymbol{L}$ (also depending on the specific application of the algorithm) are fulfilled. A call to the *Application* routine is only performed when a true result is returned by the *LFUN* function.

In Program 1, *Application* is a called procedure where the n nested loops converge and where their leading indices can be arbitrarily used in the corresponding desired internal structure. In fact, this is a general algorithm, which enables to perform a parallel *Application* implementation if the nature of the problem asks for such a process and the available hardware allows to run it in this manner. This feature of the nested sum translation can be founded on a very important property, such that the GNDL internal statements are always allowed to *run independently* into separate CPUs. That is, despite the forms of the vector j are generated in a sequential manner, the evaluation of the code using the vector form can be done independently of the remaining vector forms. This allows the parallel implementation of the algorithms which can be formulated in terms of NSS. In fact, it can be said that *any formula or algorithm which can be written in terms of NSS can be parallelized.* The practical way to do such a parallelization is described now:

(a) Generate all the forms of the vector j in a master CPU.

(b) Send these forms to a set of slave CPUs.

(c) Compute independently into each slave CPU the contribution to the NSS expression attached to the received form of vector j.

(d) Once each contribution is computed, each slave CPU sends the result to the master CPU.

(e) The master CPU collects, and adds up if necessary, all the partial contributions.

Also, another GNDL structure can be put aside *Application*, constituting the first step towards a *generalized nest of generalized nested do loops* (GNGNDL) program structure. We have developed some programming experience on this subject.

See appendix B for details about the implementation of a GNDL or a GNGNDL in C++ high level language. A previous discussion on generalized nested to loops, in a purely sequential programming framework, was initially made by Carbó and Bunge [6]. Other programs, including the implementation of some of the examples of the present paper, can be obtained from the authors upon request.

Some Fortran compilers, by the way, have no capacity of processing more than a limited number of classical do loops in a nest [7]. Thus, a simple scheme like the one shown in Program 1 is a good candidate to circumvent this limitation in any compiler, if it is present. The characteristics of the NSS together with those of the GNDL programming structure can be used as a starting point towards the codification of a *parallel do loop* high level language code.

3. Mathematical application examples

As an illustration of the possible use of the described symbols, we will first present some purely mathematical application examples of NSSs needed as a background to deal with quantum chemical problems in section 4 below. Other examples are given in appendix C as a complementary information source. They are related to mathematics in a broad sense and can be applied to other quantum chemical topics.

3.1. GENERATION OF VARIATIONS AND COMBINATIONS

A NSS can be used to generate variations and combinations of m elements belonging to an arbitrary set of mathematical objects. It is only necessary to number in a canonical order, from 1 to m, all the elements in the set. This will produce a completely formal development, which can be occasionally used for immediate implementation on any high level language. Although this direct translation will obviously lack programming refinement in the first bulk program scheme, it may be considered a not too bad starting point in order to obtain a given optimized code.

Then, one can easily construct the following objects.

3.1.1. Variations with repetition

Implementation of the $\sum_n(j = 1, m1)$ symbol generates inside the nested sum all the m^n variations with repetition, which can be formed making groups of n elements out of the m element set, with the condition $m \geqslant n$.

3.1.2. Variation without repetition

The implementation of the nested sum structure

$$\sum_n(j = \mathbf{1}, m\mathbf{1}, \mathbf{1}, \Delta(L)) \tag{14}$$

corresponds to the generation of the $m!/(m-n)!$ variations without repetition, which can be formed making groups of n objects taken from the m element set inside the nested sum. Here, Δ is a GLKD, see section 2.4, and $\boldsymbol{L}$ is a $\binom{n}{2}$-dimensional LV defined as $\boldsymbol{L} = \boldsymbol{L}(\boldsymbol{j}, [\neq])$ where the definition (12) of section 2.8 has been used.

3.1.3. Combinations

It is also possible to generate the $\binom{m}{n}$ combinations related to a set of m elements, when they are taken in groups of n out of the m element set. Such a generation can be performed using a similar structure as the one appearing in eq. (14), but redefining the LV by means of the following expression: $\boldsymbol{L} = \boldsymbol{L}(\boldsymbol{j}, [<])$, where the definition (12) has been used again.

3.2. EXPLICIT EXPRESSION OF THE DETERMINANT OF AN ARBITRARY SQUARE MATRIX

Using the NSS, one can rewrite the expression of an arbitrary $(n \times n)$ square matrix $\boldsymbol{A}$ determinant, Det $|\boldsymbol{A}|$, [8]. A compact formula of the determinant can be written in this way as

$$\mathrm{Det}|\boldsymbol{A}| = \sum_n(\boldsymbol{j} = \boldsymbol{1}, n\boldsymbol{1}, \Delta(\boldsymbol{L}))S(\boldsymbol{j}) \prod(\boldsymbol{j}, \boldsymbol{A}) , \tag{15}$$

where $\boldsymbol{L}$ is a $\binom{n}{2}$-dimensional LV defined as $\boldsymbol{L} = \boldsymbol{L}(\boldsymbol{j}, [\neq])$ where the definition (12) has been used.

The $S(\boldsymbol{j})$ factor is a sign, which can be expressed by

$$S(\boldsymbol{j}) = (-1)^{P(\boldsymbol{j})} , \tag{16}$$

where $\boldsymbol{P}(\boldsymbol{j})$ is the parity associated to the ordinal structure of the vector $\boldsymbol{j}$ index elements. It can be computed using the following LKD expression:

$$\boldsymbol{P}(\boldsymbol{j}) = \delta\left(\dot{2} \neq \sum_{p=1}^{n-1} \sum_{q=p+1}^{n} \delta(j_p > j_q)\right) . \tag{17}$$

Finally, the $\prod(\boldsymbol{j}, \boldsymbol{A})$ terms are defined by means of the product:

$$\prod(\boldsymbol{j}, \boldsymbol{A}) = \prod_{i=1}^{n} a_{i,j_i} . \tag{18}$$

Although the final determinant structure (15) can easily lead to an immediate construction of sequential or parallel routines, there cannot be a claim such that this procedure will be better, from a computational point of view, then well-stablished numerical ones, based on other grounds such as Cholesky decomposition, see refs. [8b,9] for more details.

The previous determinant development form in eq. (15) can be used as a very general interpretative formula, which can compete pedagogically and practically with other widespread alternatives, for example those usually employed in quan-

tum chemistry, see ref. [10] as a guide. In section 4 this determinant form is used, for example, to deal with Slater determinants.

One can easily see that, despite all criticisms which can arise from the programming technical side, the nested sum formalism permits to solve in a very elegant manner the following problem: *Program in a chosen high level language a function procedure which can be used to compute the determinant of a general square matrix using the Laplace expansion.*

4. Quantum chemical application examples

Several quantum chemical application examples of NSSs and LFs follow after the previous mathematically illustrative simple examples. Some of them had been chosen because they are related to the actual research in this field in our laboratory.

We do not pretend to give here an exhaustive account of all the possible applications of NSSs in quantum chemistry. Some areas, which for sure can be studied from the nested summation point of view, like the coupled cluster theory [11], are not included here.

In fact our interest in the present formulation, the use of NSSs and LFs, was aroused when studying the integrals over Cartesian exponential type orbitals [4]. The use of both symbols in this case has been extensively studied in the above reference, so we will not repeat here the already published arguments.

4.1. SLATER DETERMINANTS AND MATRIX ELEMENTS

4.1.1. General definition

As it is shown in section 3.2, using nested sum terminology, the general expression for any determinant can be obtained. In this manner, this formulation can be transferred into the study of Slater determinants [10], constructed by n spin-orbitals associated to n electrons. We adopt the following structure and notation for normalized Slater determinants:

$$|D(\boldsymbol{n})\rangle = (n!)^{-1/2}\mathrm{Det}|\psi_1\psi_2\ldots\psi_n| = (n!)^{-1/2}\sum_n(\boldsymbol{j}=\boldsymbol{1},\boldsymbol{n1},\Delta(\boldsymbol{L}))S(\boldsymbol{j})|\Psi(\boldsymbol{j})\rangle\,, \tag{19}$$

where the $|\Psi(\boldsymbol{j})\rangle$ terms are built up with the spin orbital products:

$$|\Psi(\boldsymbol{j})\rangle = \prod_{k=1}^{n}\psi_{j_k}(k)\,. \tag{20}$$

In this formulation, the electron coordinates are kept canonically ordered, and the spin-orbital functions are combined and reordered by means of the NSS index flow, as it can be seen in eq. (20). Another approach is also valid and may be as use-

ful as the present one: it fixes the vector index of the spin-orbitals in canonical order and reorganizes the ordering of the electron coordinate indices, that is,

$$|\Psi(\boldsymbol{j})\rangle = \prod_{k=1}^{n} \psi_k(j_k)\,, \tag{21}$$

but the first choice is more interesting for developing other quantum chemical terms involving Slater determinants, as we shall discuss below.

4.1.2. Matrix elements of arbitrary n-electron operators

An operator, depending on an arbitrary number of electron coordinates, has an easily expressible set of matrix elements, using two Slater determinants $D(\boldsymbol{j})$ and $D(\boldsymbol{k})$.

The term $D(\boldsymbol{j})$ is a Slater determinant, formed by n functions chosen from a set of m available spin-orbitals, and ordered following the actual internal values of the $\boldsymbol{j}$ index vectors, That is,

$$D(\boldsymbol{j}) = (n!)^{-1/2}\mathrm{Det}|\psi_{j_1}\psi_{j_2}\ldots\psi_{j_n}|\,, \tag{22}$$

where the usual abbreviated form for a Slater determinant has been used as in equation (19).

Both determinants $D(\boldsymbol{j})$ and $D(\boldsymbol{k})$ can be considered to be built up as defined in eq. (22). The number of different spin-orbitals appearing in both determinants can produce a zero result for the matrix element, as it is well known for one and two electron operators, see ref. [10]. Generalization to integrals over any number of electrons can be performed as follows: Suppose an r-electron operator to be written as $\Omega(\boldsymbol{r})$, with the r-dimensional vector $\boldsymbol{r}$ representing the coordinates of the canonically ordered $r(\leqslant n)$ electron set: $(\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_r)$. The matrix element between two Slater determinants can be written as

$$\begin{aligned}\langle D(\boldsymbol{j})|\Omega(\boldsymbol{r})|D(\boldsymbol{k})\rangle = (n!)^{-1}\textstyle\sum_n(\boldsymbol{p} = \mathbf{1}, n\mathbf{1}, \Delta(\boldsymbol{L}(\boldsymbol{p})))\\ \cdot \textstyle\sum_n(\boldsymbol{q} = \mathbf{1}, n\mathbf{1}, \Delta(\boldsymbol{L}(\boldsymbol{q})))\\ \cdot S(\boldsymbol{p})S(\boldsymbol{q})\langle\Psi(\boldsymbol{j}[\boldsymbol{p}])|\Omega(\boldsymbol{r})|\Psi(\boldsymbol{k}[\boldsymbol{q}])\rangle\,,\end{aligned} \tag{23}$$

where the symbol $\boldsymbol{j}[\boldsymbol{p}]$ means that a permutation $\boldsymbol{p}$ is performed over the parameter vector $\boldsymbol{j}$ subindices. The integral over the spin-orbital products, appearing as the right-most term of eq. (23), can now be simplified. Because the canonical ordering of the electrons is preserved by convention in the spin-orbital products implicit in eq. (19), as discussed before, on can write the integral appearing in eq. (23) using only the first r spin-orbitals of the successive products, which in turn will be the ones connected with $\Omega(\boldsymbol{r})$, the r-electron operator:

$$\langle \Psi(\boldsymbol{j}[\boldsymbol{p}])|\Omega(\boldsymbol{r})|\Psi(\boldsymbol{k}[\boldsymbol{q}])\rangle = \left\langle \prod_{i=1}^{r} \psi_{j[p_i]}(i) \middle| \Omega(\boldsymbol{r}) \middle| \prod_{i=1}^{r} \Psi_{k[q_i]}(i) \right\rangle \cdot \delta(j[p_j] = k[q_j];\ \forall j = r+1, n)\,. \tag{24}$$

The LKD appearing at the end of eq. (24) when integration is performed over the coordinates of the remaining n-r electrons can be easily substituted by the equivalent logical expression

$$\delta(j[p_j] = k[q_j];\ \forall j = r+1, n) = \delta(\|\boldsymbol{j}[\boldsymbol{p}] - \boldsymbol{k}[\boldsymbol{q}]\|_1 = \sum_{i=1}^{r} |j[p_i] - k[q_i]|)\,, \tag{25}$$

where the Minkowski norm of the difference between the permuted vectors $\boldsymbol{j}[\boldsymbol{p}]$ and $\boldsymbol{j}[\boldsymbol{q}]$ must be equal to the sum of the absolute values of the differences between the first rth components of both vectors.

The right-hand part of the last equality (25) may be substituted in eq. (24) and the resulting formula substituted into the expression (23), giving

$$\langle D(\boldsymbol{j})|\Omega(\boldsymbol{r})|D(\boldsymbol{k})\rangle = (n!)^{-1} \sum_n (\boldsymbol{p} = \mathbf{1}, n\mathbf{1}, \Delta(\boldsymbol{L}(\boldsymbol{p}))) \sum_n (\boldsymbol{q} = \mathbf{1}, n\mathbf{1}, \Delta(\boldsymbol{L}(\boldsymbol{q}))) \cdot S(\boldsymbol{p}) S(\boldsymbol{q}) \left\langle \prod_{i=1}^{r} \psi_{j[p_i]}(i) \middle| \Omega(\boldsymbol{r}) \middle| \prod_{i=1}^{r} \psi_{k[q_i]}(i) \right\rangle \cdot \delta\left(\|\boldsymbol{j}[\boldsymbol{p}] - \boldsymbol{k}[\boldsymbol{q}]\|_1 = \sum_{i=1}^{r} |j[p_i] - k[q_i]| \right). \tag{26}$$

This final result indicates fairly well one that can have *at least r differences* between the spin-orbitals involved in constructing both determinants in order that the integral (23) keeps being not automatically zero. This result encompass the well-described zero-, one- and two-electron operator cases [10,12], generalizing in this way the rules governing the calculation of operator matrix elements between two Slater determinants. One can, thus, say that the general rule in order to prevent automatic integral nullity is: *r-electron operators allow a maximal amount of r spin-orbital differences.*

The same expression can be used with the appropriate restrictions to obtain matrix elements over Slater determinants made from non-orthogonal one-electron functions. The LKD expression, appearing in eq. (24), as defined in eq. (25) must be substituted by a product of overlap integrals, S_{ij}, between the involved spin orbitals, that is

$$\langle \Psi(\boldsymbol{j}[\boldsymbol{p}])|\Omega(\boldsymbol{r})|\Psi(\boldsymbol{k}[\boldsymbol{q}])\rangle = \left\langle \prod_{i=1}^{r} \psi_{j[p_i]}(i) \middle| \Omega(\boldsymbol{r}) \middle| \prod_{i=1}^{r} \psi_{k[q_i]}(i) \right\rangle \cdot \prod_{j=r+1}^{n} S_{j[p_j],k[q_j]}\,. \tag{27}$$

4.2. CI WAVEFUNCTIONS

Using the approach already described for combination generation, as outlined in section 3.1.3, one can formulate in a short but completely general way the CI wavefunction [12,13] structure.

This kind of wavefunctions, in the complete CI framework, as Knowles and Handy [13e] have proved feasible, for a system of m spin-orbitals and n ($\leqslant m$) electrons can be written using a NSS formalism as

$$\Phi = \sum_n (j = \mathbf{1}, m\mathbf{1}, \Delta(\boldsymbol{L}))C(j)D(j)\,, \tag{28}$$

where the same LV as defined in section 3.1.3 has been used here. The terms $D(j)$ are Slater determinants constructed as the one defined in eq. (22). The $C(j)$ factors are variational coefficients attached to each Slater determinant.

Also, an alternative formulation of eq. (28) can be conceived if one wants to distinguish between the ground state, monoexcitations, biexcitations and so on. Such a possibility is symbolized in the following CI wavefunction expression for n electrons, constructed so as to include Slater determinants up to the pth ($p \leqslant n$) excited order. One can initially start from n occupied spin-orbitals $\{\psi_j\}_{j=1,n}$ and $m(\geqslant p)$ unoccupied ones $\{\varphi_k\}_{k=1,m}$. Then, the CI wavefunction is written in this case as the linear combination

$$\Phi = \sum_{e=0}^{p} \sum_{n-e}(j = \mathbf{1}, n\mathbf{1}, \Delta(\boldsymbol{L}(j)))\sum_e(k = \mathbf{1}, m\mathbf{1}, \Delta(\boldsymbol{L}(k))) \cdot C(j \oplus k)D(j \oplus k)\,, \tag{29}$$

where the index e, appearing in the first classical sum, signals the excitation order. That is, for $e = 0$ one has the ground state, for $e = 1$ the monoexcitations are obtained, etc.

In eq. (29) the $D(j \oplus k)$ terms are n-electron Slater determinants formed by the spin-orbitals numbered by means of the direct sum: $j \oplus k$ of the vector index parameters attached to the involved nested sums and to the occupied–unoccupied orbitals respectively. That is,

$$D(j \oplus k) = (n!)^{-1/2}\,\mathrm{Det}|\psi_{j_1}\psi_{j_2}\ldots\psi_{j_{n-e}}\phi_{k_1}\phi_{k_2}\ldots\phi_{k_e}|\,. \tag{30}$$

These two general CI function expressions, along with the results obtained in section 4.1 above, permit to compute the expected value form of any quantum mechanical operator in a most complete general way.

4.3. DENSITY FUNCTIONS

4.3.1. General theory: Löwdin–McWeeny scheme

Density functions of any order can be constructed by means of Löwdin or McWeeny descriptions [14]. Using a normalized n-electron wavefunction $|\Phi(j)\rangle$,

usually taken as a Slater determinant or a linear combination of them, the nth order density matrix element $\rho^{(n)}(\boldsymbol{j}, \boldsymbol{j}')$ can be written as

$$\rho^{(n)}(\boldsymbol{j}, \boldsymbol{j}') = |\Phi(\boldsymbol{j})\rangle\langle\Phi(\boldsymbol{j})|\,, \tag{31}$$

where $\boldsymbol{j}$ and $\boldsymbol{j}'$ are n-dimensional vectors collecting the coordinates of the n particles.

A recurrent procedure can be defined to obtain the remaining lesser order density matrix elements. The $(n-1)$th order density matrix element is obtained from the nth order one, integrating over the coordinates of one electron. The result is

$$\rho^{(n-1)}(\boldsymbol{j}, \boldsymbol{j}') = n \int\int \delta(j'_n - j_n)\Phi(\boldsymbol{j}\oplus j_n)\Phi^*(\boldsymbol{j}'\oplus j'_n)\, dj'_n\, dj_n\,, \tag{32}$$

where δ is the Dirac delta function. The n-dimensional vector argument of the system wavefunction has been expressed in terms of the direct sum of two subvectors, this requires a n-dimensional vector partition in a $(n-1)$-dimensional vector plus a monodimensional one. The last one has been taken arbitrarily as the last index of the original n-dimensional vector.

Using this recurrence formula, it is possible to generate all the possible density matrix elements of order n-m up to the first. The explicit expression for the $(n$-$m)$th order matrix element is

$$\rho^{(n-m)}(\boldsymbol{j},\boldsymbol{j}') = \binom{m}{n} \int\int \left(\prod_{k=1}^{m} \delta(j'_{n-k+1} - j_{n-k+1})\right) \Phi(\boldsymbol{j}\oplus\boldsymbol{i})\Phi^*(\boldsymbol{j}'\oplus\boldsymbol{i}')\, d\boldsymbol{i}'\, d\boldsymbol{i}\,, \tag{33}$$

where the vector $\boldsymbol{i}$ is constructed as $\boldsymbol{i} = (j_{n-m+1}, j_{n-m+2}, \ldots, j_n)$ and the differential $d\boldsymbol{i}$ is defined as $d\boldsymbol{i} = dj_{n-m+1} dj_{n-m+2} \ldots dj_n$. Similar expressions hold for the primed terms.

Diagonal terms of the density matrix are obtained when the primed arguments become equal to the unprimed ones. The nth order diagonal element of the density matrix is then expressed as

$$\rho^{(n)}(\boldsymbol{j},\boldsymbol{j}) = \rho^{(n)}(\boldsymbol{j}) = |\Phi(\boldsymbol{j})\rangle\langle\Phi(\boldsymbol{j})|\,, \tag{34}$$

and can be named the nth order density function. Lesser order density functions can be obtained in a similar way as described in eq. (33):

$$\rho^{(n-m)}(\boldsymbol{j}) = \binom{m}{n} \int\int \Phi(\boldsymbol{j}\oplus\boldsymbol{i})\Phi^*(\boldsymbol{j}\oplus\boldsymbol{i})\, d\boldsymbol{i}\,. \tag{35}$$

The integration of the first order density function is equal to n, the number of involved particles. The pth order density function has the following meaning: the term $\rho^{(p)}(\boldsymbol{j})d\boldsymbol{j}$ gives, multiplied by n, the probability to find, within the volume element $d\boldsymbol{j}$, p particles having the space-spin coordinates $\{j_1, j_2, \ldots, j_p\}$, respectively, averaged over all the space-spin positions of the remaining n-p particles.

Here we will give alternative explicit expressions for non–diagonal terms of the density matrix, which will be general in the sense that diagonal elements appear as particular cases.

4.3.2. Density matrix elements and n-particle operators

Density matrices are well suited in order to express expectation values of operators as it is shown in ref. [14a]. Here, a reformulation in terms of NSS will be given.

Let an n-particle operator be expressible in terms of other zero-, one-, ..., many-particle operators. This general n-particle operator can be written in terms of the NSS formalism as

$$\Omega(1,2,\ldots,n) = \Omega(\boldsymbol{n}) = \sum_{i=0}^{m} \sum\nolimits_i(\boldsymbol{j}) \Omega^{(i)}(\boldsymbol{j}) \,. \tag{36}$$

If the operators are symmetrical with respect to the particle indices, then, an equivalent expression for the operator (36) can be written as

$$\Omega(\boldsymbol{n}) = \sum_{i=0}^{m} \sum\nolimits_i(\boldsymbol{j}, \Delta(\boldsymbol{L})) \Omega^{(i)}(\boldsymbol{j}) \,, \tag{37}$$

where the generalized logical Kronecker delta, $\Delta(\boldsymbol{L})$, included in the NSSs is used to avoid redundant terms in eq. (37). This requires the logical vector, $\boldsymbol{L}$, to be defined as $\boldsymbol{L} = \boldsymbol{L}(\boldsymbol{j}, [<])$, following the notation of eq. (12).

In the general case, shown in eq. (36), the expectation value of the operator can be obtained by means of the following integral:

$$\begin{aligned} \langle \Omega(\boldsymbol{n}) \rangle &= \langle \Phi(\boldsymbol{j}) | \sum_{i=0}^{m} \sum\nolimits_i(\boldsymbol{j}) \Omega^{(i)}(\boldsymbol{j}) | \Phi(\boldsymbol{j}) \rangle \\ &= \sum_{i=0}^{m} \sum\nolimits_i(\boldsymbol{j}) \int \Phi^*(\boldsymbol{j}) \Omega^{(i)}(\boldsymbol{j}) \Phi(\boldsymbol{j}) \, d\boldsymbol{j} \\ &= \sum_{i=0}^{m} \sum\nolimits_i(\boldsymbol{j}) \int \delta(\boldsymbol{j}' - \boldsymbol{j}) \Omega^{(i)}(\boldsymbol{j}) \rho^{(i)}(\boldsymbol{j}', \boldsymbol{j}) d\boldsymbol{j}' d\boldsymbol{j} \,, \end{aligned} \tag{38}$$

where $|\Phi(\boldsymbol{j})\rangle$ is the system wavefunction and it is assumed that the primed variables are just a copy of the unprimed ones, but the many-particle operator only acts over the later ones.

With respect to the above comments, it can be said that density matrix elements can be considered as n-particle operators. This feature is related to the comments outlined in section 4.1.2. and it will be used when developing the *quantum similarity* theoretical framework in section 4 below.

4.3.3. NSS explicit expression for density matrix elements

Using a normalized n-electron Slater determinant, $|D(\boldsymbol{j})\rangle$, as a system wavefunc-

tion, constructed as discussed in section 4.1, one can write the nth order density matrix element $\rho^{(n)}(j, j')$ as

$$\rho^{(n)}(j,j') = |D(j)\rangle\langle D(j')|$$
$$= (n!)^{-1}\sum_n(\boldsymbol{p} = \mathbf{1}, n\mathbf{1}, \Delta(\boldsymbol{L}(\boldsymbol{p})))\sum_n(\boldsymbol{q} = \mathbf{1}, n\mathbf{1}, \Delta(\boldsymbol{L}(\boldsymbol{q})))$$
$$\cdot S(\boldsymbol{p})S(\boldsymbol{q})|\Psi(j[\boldsymbol{p}])\rangle\langle\Psi(j'[\boldsymbol{q}])|\,, \tag{39}$$

where, the $|\Psi(j[\boldsymbol{p}])\rangle$ terms are spin-orbital products as they were defined in eq. (20); the symbol $j[\boldsymbol{p}]$ means a permutation $\boldsymbol{p}$ is performed over the parameter vector j subindices.

Generalization of this one term function result is straightforward. If the system wavefunction is a linear combination of Slater determinants, $\{|D_p(j)\rangle\}$, as in the expression

$$|\Phi(j)\rangle = \sum_{p=1}^{m} C_p|D_p(j)\rangle\,, \tag{40}$$

then, the nth order density matrix element takes the following form:

$$\rho^{(n)}(j,j') = |\Phi(j)\rangle\langle\Phi(j')| = \sum_{p=1}^{m}\sum_{q=1}^{m} C_pC_q^*|D_p(j)\rangle\langle D_q(j')|\,. \tag{41}$$

The determined products appearing in the right-most side of eq. (41) bear the same structure as those appearing in expression (39); they differ in that the two involved determinants are constructed with two non-equivalent sets of spin-orbitals.

Here we will only deal with monodeterminantal functions to describe a simplified theoretical development. The interesting practical result concerning nth order density matrix elements, constructed using Slater determinants as basis sets, appears once spin-orbitals are described by means of the LCAO approach. This possibility will be described now.

4.3.4. LCAO form of density matrix elements

Let us write the LCAO approach for a given spin-orbital set $\{\psi_k\}$ as

$$\psi_k = \sum_{a=1}^{M_k} c_{ak}\chi_a\,, \tag{42}$$

where each spin-orbital has been expressed as a linear combination of atomic spin-orbitals using a M_k-dimensional basis set $X = (\chi_1, \chi_2, \ldots \chi_{M_k})$. Then, within the NSS formalism, a product of spin-orbitals like (20) can be structured by means of the linear combination (42) as

$$|\Psi(\boldsymbol{j}[\boldsymbol{p}])\rangle = \prod_{i=1}^{n}\left(\sum_{a_i=1}^{M_{j[p_i]}} c_{a_i j[p_i]}\chi_{a_i}(i)\right)$$

$$= \sum_n(\boldsymbol{a}=\boldsymbol{1},\boldsymbol{M})\boldsymbol{C}(\boldsymbol{a},\boldsymbol{j}[\boldsymbol{p}])\boldsymbol{X}(\boldsymbol{a})\,, \tag{43}$$

where a simplified NSS notation has been used, $\boldsymbol{M}$ is the vector containing the elements $\{M_{j[p_i]}\}$, The symbol

$$\boldsymbol{C}(\boldsymbol{a},\boldsymbol{j}[\boldsymbol{p}]) = \prod_{i=1}^{n} c_{a_i j[p_i]} \tag{44}$$

is a product of LCAO coefficients, and

$$\boldsymbol{X}(\boldsymbol{a}) = \prod_{i=1}^{n}\chi_{a_i}(i) \tag{45}$$

is a basis function product, respectively. When using a NSS formalism, it can be seen how the product of LCAO spin-orbital expressions can be written in a similar form to the one sum function of eq. (42). Using eq. (43) in the spin-orbital product appearing in the right-most side of eq. (39), one obtains

$$|\Psi(\boldsymbol{j}[\boldsymbol{p}])\rangle\langle\Psi(\boldsymbol{j}'[\boldsymbol{q}])| = \sum(\boldsymbol{a}\oplus\boldsymbol{b})\boldsymbol{C}(\boldsymbol{a},\boldsymbol{j}[\boldsymbol{p}])\boldsymbol{C}^*(\boldsymbol{b},\boldsymbol{j}[\boldsymbol{q}])\cdot\boldsymbol{X}(\boldsymbol{a})\boldsymbol{X}^*(\boldsymbol{b}')\,, \tag{46}$$

where a simplified NSS notation has been used. Primed indices note the spin-orbital dependence with respect to the bra coordinates.

Finally, the nth order density matrix elements can be expressed in terms of the atomic spin-orbital products as

$$\rho^{(n)}(\boldsymbol{j},\boldsymbol{j}') = \sum(\boldsymbol{a}\oplus\boldsymbol{b})\boldsymbol{Q}^{(n)}(\boldsymbol{a},\boldsymbol{b})\boldsymbol{X}(\boldsymbol{a})\boldsymbol{X}^*(\boldsymbol{b}')\,, \tag{47}$$

being the nth order charge and bond order hypermatrices, $\boldsymbol{Q}^{(n)}(\boldsymbol{a},\boldsymbol{b})$, defined in turn as

$$\boldsymbol{Q}^{(n)}(\boldsymbol{a},\boldsymbol{b}) = (n!)^{-1}\sum_n(\boldsymbol{p}=\boldsymbol{1},n\boldsymbol{1},\Delta(\boldsymbol{L}(\boldsymbol{p})))\sum_n(\boldsymbol{q}=\boldsymbol{1},n\boldsymbol{1},\Delta(\boldsymbol{L}(\boldsymbol{q})))$$

$$\cdot\,\omega(\boldsymbol{p},\boldsymbol{q})\boldsymbol{C}(\boldsymbol{a},\boldsymbol{j}[\boldsymbol{p}])\boldsymbol{C}^*(\boldsymbol{b},\boldsymbol{j}[\boldsymbol{q}])\,, \tag{48}$$

using the *occupation* hypermatrix elements

$$\omega(\boldsymbol{p},\boldsymbol{q}) = \boldsymbol{S}(\boldsymbol{p})\boldsymbol{S}(\boldsymbol{q})\,. \tag{49}$$

Equation (47) has the same structure as the well-known LCAO form of the first order density function [10]. This is not surprising, because a characteristic of the NSS formalism consists in reproducing the structure of monodimensional formula expressions within a complete general n-dimensional framework.

Into this formulation, it is easy to find and expression for the density function

in terms of spatial orbitals only. It can be done considering the spin-orbital product (45) split into two parts: the spatial orbitals and the spin functions product:

$$\boldsymbol{X}(\boldsymbol{a}) = \varphi(\boldsymbol{a})\sum(\boldsymbol{a})\,, \tag{50}$$

where

$$\varphi(\boldsymbol{a}) = \prod_{i=1}^{n} \varphi_{a_i}(i) \tag{51}$$

is a product of spatial orbitals and

$$\Sigma(\boldsymbol{a}) = \prod_{i=1}^{n} \sigma_{a_i}(s_i) \tag{52}$$

is a product of spin functions. The function $\sigma_{a_i}(s_i)$ stands for the spin function associated to electron i with spin variable s_i.

Once these product functions are substituted in the expression for the density function derived from eq. (47) and the integration over the spin variables is performed, the LCAO expression for the spatial n-particle density function arises:

$$\rho_e^{(n)}(\boldsymbol{j}) = \sum(\boldsymbol{a}\oplus\boldsymbol{b}, \Delta(\boldsymbol{L}))\boldsymbol{Q}^{(n)}(\boldsymbol{a},\boldsymbol{b})\varphi(\boldsymbol{a})\varphi^*(\boldsymbol{b})\,, \tag{53}$$

where the GLKD function LV argument is defined as

$$\boldsymbol{L} = \boldsymbol{L}(\boldsymbol{a},\boldsymbol{b}) = \{\sigma_{a_i} = \sigma_{b_i}; \forall i = 1,n\}\,. \tag{54}$$

4.4. PERTURBATION THEORY

Here we briefly discuss the quantum chemical problem consisting in finding the energy and wavefunction correction expressions of a system within the perturbation theory (PT) framework. However, this problem transcends in various aspects the quantum chemical scene and also be considered as belonging to a broader quantum mechanical circle.

Some years ago one of us was interested in studying general expressions of Rayleigh–Schrödinger (R–S) PT [15]. We feel that the use of NSS is an ideal framework to obtain a *really general PT* scheme.

In many text books [16] can be found the development of the PT and the associated methodology to obtain the corrections for the energy and the perturbed wavefunction. The resolvent technique [12,16a,c,17] allows one to express in a formal manner all the corrections of the wavefunction and also those of the energy values. In fact, the PT may appear cumbersome because as the order of the correction increases a large number of summation symbols in the explicit formulae for these corrections are needed [12,18]. NSSs override this problem as it will be shown below. More details can be obtained in ref. [19], where the Brillouin–Wigner

(B–W) PT is implemented. Also, in ref. [20], the reader can find the development of a generalized R–S PT in matrix form.

4.4.1. Brillouin–Wigner perturbation theory

In the B–W perturbation formalism, the nth order wavefunction correction [16c] can be easily written, using the NSS formalism, as [19]

$$|n;i\rangle = \sum_n(\boldsymbol{j}, \Delta(\boldsymbol{L}))R_i(\boldsymbol{j})|0;i\rangle; \quad n > 0\,, \tag{55}$$

and the corrections over the B–W energies can be expressed, using again the NSS operators, as

$$E_i^{(n)} = \sum_{n-1}(\boldsymbol{j}, \Delta(\boldsymbol{L}))\langle i;0|V|R_i(\boldsymbol{j})|0;i\rangle;\ n > 1\,, \tag{56}$$

where V is the perturbation operator.

In eq. (55) and (56) the LV is defined as $\boldsymbol{L}(\boldsymbol{j}) = \{L_k = [j_k \neq i]; k = 1,n\}$. The operator $R_i(\boldsymbol{j})$ is written as

$$R_i(\boldsymbol{j}) = \prod_{p=1}^{n} Z_{i,j_p}\,, \tag{57}$$

where $Z_{p,q}$ is a projector-like operator defined in turn as

$$Z_{p,q} = (E_p - E_q^{(0)})^{-1}|0;q\rangle\langle q;0|V\,. \tag{58}$$

Thus, one can see the NSS as a useful device which permits to write in a compact manner the B–W perturbation general equations (55) and (56) as it is show in ref. [19].

4.4.2. General Rayleigh–Schrödinger perturbation theory

The use of NSS as an ideal framework to construct a *really general R–S PT* scheme encompassing as a particular case the study of multiple perturbations as those discussed recently by Kutzelnigg [21].

Let us write a perturbed Hamiltonian in a general form by a set of k independent perturbation operators using the following expression involving a NSS:

$$H = \sum_k(\boldsymbol{p} = \boldsymbol{0}, \boldsymbol{K})\lambda(\boldsymbol{p})H(\boldsymbol{p})\,. \tag{59}$$

In eq. (59) the first parameter vector value gives the unperturbed Hamiltonian $H(\boldsymbol{0})$. The final parameter vector $\boldsymbol{K}$ contains the maximal order of the perturbation, which can be different for every operator. The symbol $\lambda(\boldsymbol{p})$ is any element of the scalar set of perturbation parameters. Both $H(\boldsymbol{p})$ and $\lambda(\boldsymbol{p})$ can be considered as products of pth order perturbation operators and pth powers of the attached parameters, respectively.

The perturbed energies and wavefunctions for the ith system state can be

expressed in a similar way as in scalar PT but substituting the scalar perturbation order by a *vector perturbation order* $\boldsymbol{n}$:

$$E_i = \sum_k (\boldsymbol{n} = \mathbf{0}, \infty\mathbf{1}) \lambda(\boldsymbol{n}) E_i(\boldsymbol{n}) \tag{60}$$

and

$$|i\rangle = \sum_k (\boldsymbol{n} = \mathbf{0}, \infty\mathbf{1}) \lambda(\boldsymbol{n}) |\boldsymbol{n}; i\rangle \,. \tag{61}$$

Substituting eqs. (59), (60) and (61) into the perturbed Schrödinger secular equation produces the $\boldsymbol{n}$th order equation. After an easy manipulation, the $\boldsymbol{n}$th order energy correction for the ith system's state can be written as

$$E_i(\boldsymbol{n}) = \sum_k (\boldsymbol{p} = \mathbf{0}, \boldsymbol{n}, \delta(\boldsymbol{p} \neq \mathbf{0})) \langle i; \mathbf{0} | H(\boldsymbol{p}) | \boldsymbol{n} - \boldsymbol{p}; i \rangle \tag{62}$$

provided that the orthogonality condition $\langle i; \mathbf{0} | \boldsymbol{p}; i \rangle = \delta(\boldsymbol{p} = \mathbf{0})$ holds.

Wavefunction corrections can be obtained similarly through a resolvent operator technique. The $\boldsymbol{n}$th wavefunction correction for the ith state of the perturbed system can be written: by means of a linear combination of the unperturbed state wavefunctions, excluding the ith unperturbed state:

$$|\boldsymbol{n}; i\rangle = \sum_j{}' a_{ji} |\mathbf{0}; j\rangle \,. \tag{63}$$

After some straightforward manipulation, one can obtain the $\boldsymbol{n}$th order wavefunction correction:

$$|\boldsymbol{n}; i\rangle = \sum_k (\boldsymbol{p}, \delta(\boldsymbol{p} \neq \mathbf{0})) R_i(\boldsymbol{p}) |\boldsymbol{n} - \boldsymbol{p}; i\rangle \,, \tag{64}$$

where a set of resolvent operators $\{R_i(\boldsymbol{p})|$ for the ith state is easily defined as follows:

$$R_i(\boldsymbol{p}) = \sum_j{}' (E_i(0 - E_j(\mathbf{0}^{-1} |\mathbf{0}; j\rangle \langle j; \mathbf{0}| (H(\boldsymbol{p}) - E_i(\boldsymbol{p})) \,. \tag{65}$$

Equations (62) and (64) can be considered as forming a completely general PT for *nondegenerate* systems.

4.5. GTO INTEGRAL FORMULATION

The vast use made of GTO functions when performing atomic and molecular computations [22] is well known. We are interested in problems related to *quantum similarity measures* (QSM) [23], which can be connected in turn with products of an arbitrary number of atomic orbital functions. For instance, the TRIDENT program [23h], which uses integral measures over triple products of first order density functions, requires the manipulation and integration of function products, involving up to six AO functions.

A description of a general Gaussian product theorem (GGPT) will allow us to

obtain a convenient algorithm, in order to contract the exponential part of each of the GTO functions involved in the product. Also, when primitive GTO functions are employed in practical calculations, the manipulation of Cartesian angular factor power products is common practice. Using nested summation symbolism the notation of integral formulae becomes more compact and directly programmable in both sequential and parallel environments.

4.5.1. General Gaussian product theorem

As a starting point, one can consider the general form which adopts a product of an arbitrary number of primite GTO. Let N such GTO functions be known with well-defined scaling factors, which can be collected in the vector $\boldsymbol{a} = (\alpha_1, \alpha_2, \ldots, \alpha_N)$, angular quantum numbers $\{\boldsymbol{n}_i\}_{i=1,N}$ and supposed centered at the points $\{\boldsymbol{A}_i\}_{i=1,N}$. The set of these primitive Gaussian functions will be represented by $\Gamma = \{\gamma(\boldsymbol{A}_i, \boldsymbol{n}_i, \alpha_i)\}_{i=1,N}$. The product of the whole Gaussian function set Γ can be expressed in terms of NSS notation as follows:

$$\prod_{i=1}^{N} \gamma(\boldsymbol{A}_i, \boldsymbol{n}_i, \alpha_i) = \kappa \sum\nolimits_3 \left(\boldsymbol{j} = \boldsymbol{0}, \sum_{k-1}^{N} \boldsymbol{n}_k \right) f(\boldsymbol{j}, \boldsymbol{n}, \boldsymbol{A}, \boldsymbol{P}) \gamma(\boldsymbol{P}, \boldsymbol{j}, \langle \boldsymbol{a} \rangle) , \tag{66}$$

where $\boldsymbol{A}$ and $\boldsymbol{n}$ are $(n \times 3)$ rectangular matrices, whose rows, $\{\boldsymbol{A}_i\}_{i=1,N}$ and $\{\boldsymbol{n}_i\}_{i=1,N}$, are the coordinates of the centers and the quantum numbers of each GTO, respectively.

A theorem involving the product of N GTO basis functions has been used, generalizing the well-known two Gaussian product theorem (TGPT) [24]. To prove this final result, a simple inductive proof can be used, which is given in appendix D. In eq. (66), the constant κ is defined as $\kappa = \exp(\langle \boldsymbol{K} \rangle)$ with

$$\langle \boldsymbol{K} \rangle = \langle \boldsymbol{a} \rangle^{-1} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \alpha_i \alpha_j (\boldsymbol{A}_i - \boldsymbol{A}_j)^2 , \tag{67}$$

where the row vectors $\boldsymbol{A}_i = \{A_{iq}\}_{q=1,3}$. The point $\boldsymbol{P}$, as a row vector, is defined through the weighted mean:

$$\boldsymbol{P} = \langle \boldsymbol{a} \rangle^{-1} \sum_{i=1}^{N} \alpha_i \boldsymbol{A}_i . \tag{68}$$

The f coefficients can be defined as

$$f(\boldsymbol{j}, \boldsymbol{n}, \boldsymbol{A}, \boldsymbol{P}) = \prod_{q=1}^{3} g(j_q, [\boldsymbol{n}]_q, [\boldsymbol{d}]_q) , \tag{69}$$

where $\boldsymbol{d}$ is a $(n \times 3)$ matrix build up by means of the difference

$$\boldsymbol{d} = \boldsymbol{A} - \boldsymbol{Q} , \tag{70}$$

with the $(N \times 3)$ matrix $\boldsymbol{Q}$ constructed with all the rows equal to vector $\boldsymbol{P}$. The symbols $[\boldsymbol{n}]_q$ and $[\boldsymbol{d}]_q$ describe the qth columns of the matrices $\boldsymbol{n}$ and $\boldsymbol{d}$.

The g coefficients can be computed in general using a NSS formalism as

$$g(j,\boldsymbol{u},\boldsymbol{v}) = \sum_N(\boldsymbol{j}=\boldsymbol{0},\boldsymbol{u},\delta(j=\langle\boldsymbol{j}\rangle))\prod_{i=1}^{N} v_i^{u_i-j_i}\binom{u_i}{j_i}, \tag{71}$$

where $\boldsymbol{u}$ and $\boldsymbol{v}$ are column vectors of dimension N.

4.5.2. Compact form of the product of an arbitrary number of GTO functions

In this manner, one can write in short the product of N Gaussian functions as

$$\prod_{i=1}^{N}\gamma_i = \kappa\sum(\boldsymbol{j})f(\boldsymbol{j})\gamma(\boldsymbol{j}), \tag{72}$$

keeping in mind the meaning of $f(\boldsymbol{j})$ as in eq. (69) and considering that $\gamma(\boldsymbol{j})$ is a primitive Gaussian function as those defined as in eq. (66).

4.5.3. Integrals over an arbitrary number of GTO functions

Starting from the previous results, it is easy to find a general formula to express the AO integrals bearing arbitrary products of primitive GTO functions and general operators:

$$\begin{aligned}\Big\langle \prod_i \gamma_i(r)\Big|\Omega(r,s)\Big|\prod_j \gamma'_j(s)\Big\rangle &= \kappa\kappa'\sum(\boldsymbol{j})f(\boldsymbol{j})\sum(\boldsymbol{k})f'(\boldsymbol{k})\cdot\langle\gamma(\boldsymbol{j})(r)|\Omega(r,s)|\gamma'(\boldsymbol{k})(s)\rangle \\ &= \kappa\kappa'\sum(\boldsymbol{j}\oplus\boldsymbol{k})f(\boldsymbol{j})f'(\boldsymbol{k})\cdot\langle\gamma(\boldsymbol{j})(r)|\Omega(r,s)|\gamma'(\boldsymbol{k})(s)\rangle,\end{aligned} \tag{73}$$

where $\Omega(r, s)$ is a mono ($r = s$) or bi ($r \neq s$) electronic operator. The $\langle\gamma(\boldsymbol{j})(r)|\Omega(r, s)|\gamma'(\boldsymbol{k})(s)\rangle$ integrals become mono- or bicentric depending on the nature of the operator $\Omega(r, s)$. Generalization to integrals involving arbitrary number of electrons is straightforward:

$$\begin{aligned}&\langle\textstyle\prod_p\prod_i\gamma_i(p)|\Omega(1,2,\ldots,n)|\prod_q\prod_j\gamma'_j(q)\rangle \\ &\quad= \sum(\boldsymbol{k}_1\oplus\boldsymbol{k}_2\oplus\ldots\oplus\boldsymbol{k}_n)\sum(\boldsymbol{l}_1\oplus\boldsymbol{l}_2\oplus\ldots\oplus\boldsymbol{l}_n)\{\textstyle\prod_r f(\boldsymbol{k}_r)\}\{\prod_s f'(\boldsymbol{l}_s)\} \\ &\qquad\times\langle\textstyle\prod_p\gamma_p(\boldsymbol{k}_p)(p)|\Omega(1,2,\ldots,n)|\prod_q\gamma'_q(\boldsymbol{l}_q)(q)\rangle.\end{aligned} \tag{74}$$

Here $\Omega(1, 2, \ldots, n)$ is an n-electron operator and the products over p and q run over all the electrons. Each electron is associated with a set of Gaussian function which are considered in the products over i and j. So, it has been shown which form takes the generalization to integrals involving an arbitrary number of electrons when dealing with GTO functions.

4.6. GENERAL FORMULATION OF QSM

4.6.1. Beyond the expectation value concept

The role played by the expectation value concept is well known in quantum mechanics [25]. Here it is shown how this concept also applies to function structures providing the concept of *expectation functions*. This idea leads towards one of the innermost definitions of the QSM theory: general density transformations (DT), which we will discuss next.

General DT. Let us define the *transform* of a function $f(\boldsymbol{r}, \boldsymbol{u}, \boldsymbol{p})$ as the following integral [26]:

$$T(\boldsymbol{r}, \boldsymbol{s}, \boldsymbol{p}) = \int \Omega(\boldsymbol{r}, \boldsymbol{s}, \boldsymbol{u}, \boldsymbol{p}) f(\boldsymbol{r}, \boldsymbol{u}, \boldsymbol{p})\, d\boldsymbol{u}\,, \tag{75}$$

where the operator $\Omega(\boldsymbol{r}, \boldsymbol{s}, \boldsymbol{u}, \boldsymbol{p})$ is the *kernel* of the transform. The optional vector $\boldsymbol{p}$ provides the dependence of the function, the operator or both with respect of a parameter set.

Starting from eq. (75), a *n-th order density integral transform* can be defined as the transformation of a density matrix element of the same order:

$$P^{(n)}(\boldsymbol{r}, \boldsymbol{s}, \boldsymbol{p}) = \int \Omega(\boldsymbol{r}, \boldsymbol{s}, \boldsymbol{u}, \boldsymbol{p}) \rho^{(n)}(\boldsymbol{r}, \boldsymbol{u}, \boldsymbol{p})\, d\boldsymbol{u}\,, \tag{76}$$

where $\boldsymbol{p}$ is the optional parameter vector.

This last integral in eq. (76) will be widely used in the following general quantum similarity theory development.

Expectation functions and expectation values. It can be said that the DT in eq. (76) constructs *n-th order expectation functions* of the operator Ω. From an expectation function of order n it can be obtained some *expectation value* by means of the evaluation of the expectation function at a fixed point $(\boldsymbol{r}_0, \boldsymbol{s}_0, \boldsymbol{p}_0)$, using the following integral definition:

$$\langle P^{(n)}(\boldsymbol{r}_0, \boldsymbol{s}_0, \boldsymbol{p}_0) \rangle = \int\int\int \delta(\boldsymbol{r} - \boldsymbol{r}_0)\delta(\boldsymbol{s} - \boldsymbol{s}_0)\delta(\boldsymbol{p} - \boldsymbol{p}_0) \cdot P^{(n)}(\boldsymbol{r}, \boldsymbol{s}, \boldsymbol{p})\, d\boldsymbol{r}\, d\boldsymbol{s}\, d\boldsymbol{p}\,, \tag{77}$$

where integration over all the coordinates is obviously not compulsive, allowing in this manner to obtain expectation values which in fact are functions of some of the parameters $\boldsymbol{r}$, $\boldsymbol{s}$ or $\boldsymbol{p}$.

Some cases worth of mentioning arise from the analysis of eq. (76) when the kernel takes precise particular forms. As an example we can consider expectation functions:

(a) *The density matrix elements.* When the operator Ω appearing in eq. (76) is defined as $\Omega(\boldsymbol{r}, \boldsymbol{s}, \boldsymbol{u}, \boldsymbol{p}) = \Omega(\boldsymbol{s}, \boldsymbol{u}) = \delta(\boldsymbol{u} - \boldsymbol{s})$, then the DT becomes $P^{(n)}(\boldsymbol{r}, \boldsymbol{s}, \boldsymbol{p}) = \rho^{(n)}(\boldsymbol{r}, \boldsymbol{s}, \boldsymbol{p})$ and the transformation leaves invariant the density matrix

element. Moreover, if Ω is defined as $\Omega(\boldsymbol{r}, \boldsymbol{s}, \boldsymbol{u}, \boldsymbol{p}) = \delta(\boldsymbol{u} - \boldsymbol{r})$, then $P^{(n)}(\boldsymbol{r}, \boldsymbol{s}, \boldsymbol{p}) = P^{(n)}(\boldsymbol{r}, \boldsymbol{p}) = \rho^{(n)}(\boldsymbol{r}, \boldsymbol{p})$ and the transformation returns a diagonal element of the density matrix: the nth order density function.

(b) *The electrostatic potentials.* When within the DT definition the operator form $\Omega(\boldsymbol{r}, \boldsymbol{s}, \boldsymbol{u}, \boldsymbol{p}) = |\boldsymbol{u} - \boldsymbol{s}|^{-1}$s used, then $P^{(n)}(\boldsymbol{r}, \boldsymbol{s}, \boldsymbol{p}) = V^{(n)}(\boldsymbol{r}, \boldsymbol{s}, \boldsymbol{p})$. That is, a generalized form of the electrostatic potential is obtained. One can call this general formulation an *n-th order electrostatic potential.* A particular case appears when using the density function $\rho^{(1)}(\boldsymbol{s}, \boldsymbol{p})$ in eq. (76); then the obtained transform coincides with the usual electrostatic potential function.

4.6.2. General structure of QSM

In this section, the general QSM integral will be defined and it practical computer implementation will be given within the LCAO MO framework.

The *n-th order quantum similarity measure* between a set of m systems collected in the vector $\boldsymbol{A} = (A_1, A_2, \ldots, A_m)$, with respect to an operator $\boldsymbol{W}$, can be defined as an integral of the following kind [23j]:

$$Z_A^{(n)}(\boldsymbol{W}, \boldsymbol{R}^{(m+1,q)}, \boldsymbol{S}^{(m+1,q)}, \boldsymbol{p}) = \int\int \boldsymbol{W}(\boldsymbol{R}^{(1,q)}, \boldsymbol{S}^{(1,q)}, \boldsymbol{p}) \left(\prod_{i=1}^{m} P_{A_i}^{(n_i)}(\boldsymbol{r}_i, \boldsymbol{s}_i, \boldsymbol{p}) \right) d\boldsymbol{R}^{(1,m)}\, d\boldsymbol{S}^{(1,m)} . \tag{78}$$

Considering the previous equation (78), some particular aspects of the theory and comments on the nature of the QSM elements involved have to be taken into account:

(a) The vector $\boldsymbol{n} = (n_1, n_2, \ldots, n_m)$ collects the order of the DT of the set of systems which are active in the vector $\boldsymbol{A}$.

(b) Vector $\boldsymbol{R}$ is defined as containing a set of vector coordinates: $\boldsymbol{R}^{(i,j)} = (\boldsymbol{r}_i, \boldsymbol{r}_{i+1}, \ldots, \boldsymbol{r}_j)$. The corresponding differential vector has a related structure: $d\boldsymbol{R}^{(i,j)} = d\boldsymbol{r}_i d\boldsymbol{r}_{i+1} \ldots d\boldsymbol{r}_j$. The same definitions can be considered for the vector $\boldsymbol{S}$.

(c) If $m \geqslant q$ the measure in eq. (78) does not depend on $\boldsymbol{r}$ or $\boldsymbol{s}$ coordinates.

(d) It is possible to define the operator $\boldsymbol{W}$ in eq. (78) with an embedded Dirac delta function, appearing as a multiplicative factor. The interest in this construction appears when using the Dirac functional properties: upon integration with respect to one of the variables appearing in the delta function, two coordinate vectors of the whole integrand become identical. However, this situation restricts the order of the involved DT to be the same.

(e) A word of caution must be said concerning the operator or DT definition. The introduction of pure atomic density terms as Dirac delta functions of nuclear coordinates can produce divergent integral elements when using eq. (78) and, thus, it will be the same for any previous definition. This drawback can be

avoided in some cases by introducing within the operator $\boldsymbol{W}$ some kind of weight function [27]. This warning must be taken carefully into account when using electrostatic molecular potentials as DT appearing in the similarity measure.

(f) The operator $\boldsymbol{W}$ is usually taken as a positive definite operator, but in a recent work [23g] the possibility of using any kind of operator is slightly explored.

4.6.3. LCAO expressions of QSM

Usually, the DTs resulting form eq. (76) are the density functions themselves. In this case one can write, for example, the LCAO form of the general integral (78) using the NSS formalism relative to the density functions, outlined in section 4.3 by means of eq. (47):

$$Z_A^{(n)}(\boldsymbol{W}, \boldsymbol{R}^{m+1,q}, \boldsymbol{S}^{m+1,q}) = \left\{ \prod_{i=1}^{m} \sum_{n_i} (\boldsymbol{a}_i \oplus \boldsymbol{b}_i) Q_{A_i}^{(n_i)}(\boldsymbol{a}_i, \boldsymbol{b}_i) \right\} \cdot \int \int \boldsymbol{W}(\boldsymbol{R}^{1,q}, \boldsymbol{S}^{1,q}) \boldsymbol{X}(\boldsymbol{a}_i) \boldsymbol{X}^*(\boldsymbol{b}_i)\, d\boldsymbol{R}^{1,m}\, d\boldsymbol{S}^{1,m} \quad (79)$$

Here, the parameter dependence in $\boldsymbol{p}$ has been omitted for simplicity's sake.

More details about MQS and its practical computer application can be found in ref. [23j]. There, several particular and more practical cases derived from eq. (78) are treated. here it has only been shown that the NSS formalism and notation can provide very compact but general formulae.

5. Conclusions

Two useful symbolic conventions: *Nested summation symbols* and *logical functions* have been defined, discussed and some examples given.

Apart from being able to simplify typographical structures, they constitute the cornerstone of a completely general framework, allowing to write mathematical formulae, in such a manner that *immediate translation* to any high level programming language is feasible, producing a *complete general* code, which in turn can be kept sequential or simply parallelized.

Pedagogical and in many cases mnemotechnical formula structures appear to be also deduced at a very generic level as a consequence of the use of this kind of device.

The obtained mathematical patterns seem to be also fairly well adapted to artificial intelligence formula writing programming philosophy.

An assorted set of quantum chemical and purely mathematical application examples prove the generalization power and flexibility of this presently described symbolic framework.

When NSSs and LFs are adopted as working tools, both structures appear to

trigger some sort of thinking machine, in such a way that once a given problem is solved, new study areas immediately appear to be a promising future application field in the focus of the imagination eye.

One can conclude that a *robust and powerful theoretical machinery* has been described, possessing general, far reaching imaginative possibilities.

Translation of LFs already forms part of the usual scientific programming languages, being connected to *logical if* sentences. Sadly enough, NSS, as far as our knowledge goes, *do not have a direct* translation to the usual high level languages. Present-day compilers or standard language rules ignore such an interesting feature, see for example ref. [28], where the practical final form of the standard Fortran 90 language is described.

It looks simple to introduce GNDL in the family of repetitive sentences of high level programming languages, as appendix B proves. So we feel that a claim in this direction to language and compiler builders can be sincerely made here. Some immediate computational benefits may be obtained.

Perhaps there are hidden in the symbolic limbo other possible similar tools, even better than those described here. We are confident that this paper will stimulate the research interest in this direction.

Acknowledgements

This work has been financed by the "Comisión Interministerial de Ciencia y Tecnologia" of the "Generalitat de Catalunya" through grant #QFN91-4206. One of us (E.B.) benefits from a grant from the "Department d'Ensenyament de la Generalitat de Catalunya". The authors want to explicitly thank Professor J. Karwowski and Professor C. Bunge for their interest in NSS theory and applications. Their ideas and suggestions developed throughout many discussions have been of great help.

Appendix A. Other logical functions

Here are defined other symbols which belong to the family of LFs.

A.1. RECIPROCAL LOGICAL KRONECKER DELTA (RLKD)

A RLKD is another LF defined as

$$\bar{\delta}(L) = \begin{cases} 0 & \text{if } L \text{ is true}, \\ 1 & \text{if } L \text{ is false}, \end{cases} \tag{80}$$

L being any logical expression. Also, a RLKD can be defined using a LKD formalism:

$$\bar{\delta}(L) = \delta(\neg L) = 1 - \delta(L)\,, \tag{81}$$

where a logical *negation* sign ($\neg$) symbol has been used.

A.2. RECIPROCAL GENERALIZED LOGICAL KRONECKER DELTA (RGLKD)

A RGLKD is a LF which is defined as

$$\bar{\Delta}(L) = \begin{cases} 1 & \text{if } L_i \text{ is false } \forall i = 1, n\,, \\ 0 & \text{if } \exists i | L_i \text{ is true}\,. \end{cases} \tag{82}$$

Contrary to the GLKD, this symbol now gives a value one iff all the LV elements are false, otherwise it produces a zero results. An alternative definition can be easily constructed by means of the LKD or the RLKD symbols as

$$\bar{\Delta}(L) = \Delta(\{\neg L_i;\ \forall i = 1, n\}) = \prod_{i=1}^{n} \delta(\neg L_i) = \prod_{i=1}^{n} \bar{\delta}(L_i) = \delta\left(\bigwedge_{i=1}^{n} \neg L_i\right), \tag{83}$$

where the vector L is an arbitrary LV.

Appendix B. C++ Computational implementation of NSS

A practical implementation of a NSS in terms of the C++ language [29] is presented next. There it is shown how NSS translation into a GNDL can be easily used as a pseudoinstruction.

Here, it is shown a particular codification of a NSS of the kind $\sum_n(\boldsymbol{j} = \boldsymbol{i}, \boldsymbol{f}, \boldsymbol{s})$ in terms of the C++ language by means of an object-oriented programming strategy. It will be seen how once the header of the program is defined, the use of the NSS appears to be an *ad hoc* language capability. The sequential version of the program is presented here.

Header 1 shows the header file needed to implement the NSS object and related methods. This implementation covers the ability to define arbitrary dimensional NSSs. Error tests have been omitted in order to simplify the program listing and make it more transparent.

As an example, Program 2 shows a main program using Header 1: It defines a NSS and calls the *Application* procedure. Here *Application* is a routine showing the current values of the NSS index vector $\boldsymbol{j}$. In Program 2 has been defined the 3-dimensional NSS $\sum_3(\boldsymbol{j} = (0, 0, 0,), (1, 1, 1))$ generating all the 3 digit binary numbers in canonical order.

Header 1 in this way being constructed, inside the *Application* routine can be defined another NSS object, allowing the creation of generalized nests of generalized nested do loops. Program 3, shows the basic scheme where a set of three GNDL are executed in a nested way.

```
//--------------------------------------------------------------------------------
class NSS     // Defines the NSS object
    {
         int dimension,n,k;  // n, k: Auxiliar variables
         int *j,*i,*f,*s;    // Pointers to the Parameters of the NSS
    public:
                    NSS(const int, ...);  // Creator
                    ~NSS(void);           // Destructor
         unsigned int RUNNING(void);       // Generates next leader vector
                void INITIALIZE(void);     // Initializes NSS
                 int DIMENSION(void);      // Returns NSS dimension
                 int J(int);               // Return NSS J index value
    };
//--------------------------------------------------------------------------------
unsigned int NSS::RUNNING(void) // Generates next leader vector form
 {
 if (k>=0) *(j+k)+=*(s+k);  // Step

 while (k>=0)
       if ( (*(j+k)-*(f+k))*(*(s+k))>0 ) // Limit exceeded
          {
          *(j+k)=*(i+k);
          k--;
          if (k>=0) *(j+k)+=*(s+k);  // Step
          }
       else
          {
          k=n;
          return 1; // Another vector J form is achieved
          }
  NSS::INITIALIZE();  // Initialize NSS for further use
  return 0;           // Ends NSS vector generation
 }
//--------------------------------------------------------------------------------
int NSS::DIMENSION(void) { return dimension; } // Returns the NSS dimension
//--------------------------------------------------------------------------------
// Returns the NSS vector index m value (indices numbered from 1 to dimension)
int NSS::J(int m) { return *(j+m-1); }
//--------------------------------------------------------------------------------
NSS::NSS(const int dim, ...) // Constructor
 {
 va_list index;
 va_start(index,dim);

   j=new int[dim]; // Allocate needed vectors
   i=new int[dim];
   f=new int[dim];
   s=new int[dim];
   dimension=dim;  // Store auxiliar variables
   n=dim-1;

 // Initial, Final and Step vectors
   for (int m=0;m<dim;m++) (*(i+m)=va_arg(index,int);)
   for (m=0;m<dim;m++) (*(f+m)=va_arg(index,int);)
   for (m=0;m<dim;m++) *(s+m)=va_arg(index,int);

 va_end(index);
 NSS::INITIALIZE(); // Initialize NSS
 }
//--------------------------------------------------------------------------------
void NSS::INITIALIZE(void) // Initializes leader vector J of the NSS
 {
 for (int m=0 ; m<n ; m++) *(j+m)=*(i+m);
 *(j+n)=*(i+n)-*(s+n); // Last index decremented
 k=n;
 }
//--------------------------------------------------------------------------------
NSS::~NSS(void) // Destructor
 {
 dimension=n=k=0;
 delete j,i,f,s; // Deallocates vectors
 }
//--------------------------------------------------------------------------------
```

Header 1. Codification of the NSS object definition and related methods.

```
#include <stdio.h>
#include <process.h>
#include <stdarg.h>
#include <iostream.h>
#include "Header.1"  // File including the NSS methods

// Application shows the J vector index values of the NSS
void Application(NSS &A)
{
   for (int i=1 ; i<=A.DIMENSION() ; i++) cout << A.J(i);
   puts("");
}

void main()
{
// Defines the 3-dimensional NSS called A
   NSS A(3, 0,0,0, 1,1,1, 1,1,1);

// Generates forms of vector J and calls Application
   while (A.RUNNING()) Application(A);
}
```

Program 2. C++ codification of the NSS use: $\sum_3(j = (0, 0, 0), (1, 1, 1), (1, 1, 1))$ by means of Header 1.

Appendix C. Other mathematical applications of NSSs and LFs

In this appendix other mathematical applications of the symbology which we promote are described. Some of the new formulations have immediate applications to quantum chemical problems.

C.1. TAYLOR SERIES EXPANSION OF AN n-VARIABLE FUNCTION

The complete formula for the Taylor series expansion attached to an n-variable function $f(x)$ in the neighbourhood of the point x_0 possesses the following peculiar simple structure when using NSS formalism:

$$f(x) = \sum_{m=0}^{\infty} (m!)^{-1} \sum_m (j = \mathbf{1}, n\mathbf{1}) \prod^{(m)} (j, x - x_0) \partial^{(m)}(j)[f(x_0)], \tag{84}$$

where the $\prod^{(m)}(j, z)$ terms are defined by means of the following product:

```
void main()
{
// Defines the 3 NSS's called A, B and C
   NSS A(1, 0, 1, 1);
   NSS B(2, 0,0, 1,1, 1,1);
   NSS C(3, 0,0,0, 1,1,1, 1,1,1);

// Runs over the NSS's and calls an Application that uses all
// of them.
   while (A.RUNNING())
         while (B.RUNNING())
               while (C.RUNNING())
                     Applic(A,B,C);
}
```

Program 3. Main routine showing the use of NSSs in a nested way.

$$\prod^{(m)}(\boldsymbol{j}, \boldsymbol{z}) = \prod_{i=1}^{m} z_{j_i};\ m \neq 0 \quad \wedge \quad \prod^{(0)}(j, z) = 1\,. \tag{85}$$

Also, the $\partial^{(m)}(\boldsymbol{j})[f(\boldsymbol{x}_0)]$ expression depends on the high order partial derivative operators, acting first over the function $f(\boldsymbol{x})$ and then evaluated at the point $\boldsymbol{x}_0$. The differential operators can be defined in the same manner as the terms appearing in eq. (85), but using as the second argument the nabla vector, that is,

$$\partial^{(m)}(\boldsymbol{j}) = \prod^{(m)}(\boldsymbol{j}, \nabla);\ m \neq 0 \quad \wedge \quad \partial^{(0)}(\boldsymbol{j}) = \boldsymbol{I}\,. \tag{86}$$

The expression (84) is very useful in the sense that one can control the series truncation. This is so because the parameter m gives the order of the derivatives appearing in the expansion.

Although there are some general textbook approaches to the expression (84) – see ref. [30] for example – we have not found an expression for the Taylor expansion in full as simple as it has been presented here. The pedagogical possibilities of NSSs appear again in this case. Moreover, many potential Taylor expansions are used in various physical and chemical applications; for instance in theoretical studies of molecular vibrational spectra [31], in the development of a generalized PT [20] and other quantum chemical topics, see for example ref. [32]. Then, the possibility to dispose of a compact and complete potential expression may be useful.

C.2. MULTIDIMENSIONAL NUMERICAL INTEGRATION

Here, as another example of the large number of possibilities of the NSS symbol, we will analyze how to generally write and program an arbitrary multidimensional numerical integration.

In general, a monodimensional integration or quadrature can be expressed by means of a summation symbol which has the following form [33]:

$$I_1 = \int_a^b f(x)\, dx = \sum_{j=0}^{m} C_j f(v_j)\,, \tag{87}$$

where the C_j coefficients depend on the integration method available and the v_j parameters usually take values in the interval $[a, b]$.

An extension of the previous equation, valid for the integration of an n-dimensional function can be written as

$$I_n = \int_{a_1}^{b_1} \int_{a_2}^{b_2} \cdots \int_{a_n}^{b_n} f(\boldsymbol{x})\, d\boldsymbol{x} = \sum_n (\boldsymbol{j} = \boldsymbol{0}, \boldsymbol{m}) \boldsymbol{C}(\boldsymbol{j}) f(\boldsymbol{v})\,, \tag{88}$$

where the NSS notation has been used. The integrand function depends on n variables which are collected in the vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$; also the vectors $\boldsymbol{m} = (m_1, m_2, \ldots, m_n)$ and $\boldsymbol{v} = (v_{j_1}, v_{j_2}, \ldots, v_{j_n})$ are needed. The symbol C($\boldsymbol{j}$) is the product

$$C(j) = \prod_{k=1}^{n} C_{jk} \,. \tag{89}$$

The advantage of using a NSS formalism lies in the dimension of the problem, n, which can be variable. That is, here a method is described to construct a general routine which numerically integrates functions within an arbitrary dimension space. Moreover, comparing eq. (87) with the structure of the multidimensional case shown in eq. (88), it can be seen how the use of NSS formalism leads again to an interesting mnemotechnical rule.

C.3. HOMOGENEOUS POLYNOMIALS OF m-TH ORDER

Nested summation symbology allows to express compactly a homogeneous polynomial expression of order m, $H_m(\boldsymbol{x})$ [34]. The involved homogeneous polynomial variables form the elements of the n-dimensional vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, and

$$H_m(\boldsymbol{x}) = \sum_n (\boldsymbol{j} = \boldsymbol{0}, m\boldsymbol{1}, \delta(\langle \boldsymbol{j} \rangle = m)) A(\boldsymbol{j}) \prod(\boldsymbol{j}) \,, \tag{90}$$

$A(\boldsymbol{j})$ are the polynomial coefficients and the $\prod(\boldsymbol{j})$ terms are defined as the product:

$$\prod(\boldsymbol{j}) = \prod_{i=1}^{n} x_i^{j_i} \,. \tag{91}$$

C.4. NSSs AND THE MULTINOMIAL FORMULA

Using the nested summation symbology, the multinomial formula, see for instance ref. [30], can be easily written in an explicit form:

$$\left(\sum_{i=1}^{n} x_i \right)^m = \sum_n (\boldsymbol{j} = \boldsymbol{0}, m\boldsymbol{1}, \delta(\langle \boldsymbol{j} \rangle = m)) \prod_{i=1}^{n} x_i^{j_i} \,, \tag{92}$$

where besides the nested sum, a LKD and the vector elements sum $\langle \boldsymbol{j} \rangle$ have been also used. This general form can be connected with eq. (90), a multinomial expression is, in this manner, some kind of homogeneous polynomial.

C.5. PRODUCT OF AN ARBITRARY NUMBER OF TERMS FORMED BY THE SUM OF A PRODUCT OF AN ARBITRARY NUMBER OF FACTORS

Suppose one is facing the construction of the product of an arbitrary number of terms $\{t_i\}_{i=1,n}$, each one constituted by a sum of an indeterminate number of p factors $\{f(i, k)\}_{k=1,p}$. That is:

$$t_i = \sum_{j_i=1}^{n_i} \left\{ \prod_{k=1}^{p} f(j_i, k) \right\}, \tag{93}$$

One can write the product as a nested sum:

$$T = \prod_{i=1}^{n} t_i = \sum_n (\boldsymbol{j} = \mathbf{1}, \boldsymbol{n}) \left\{ \prod_{k=1}^{p} F(\boldsymbol{j}, k) \right\}, \tag{94}$$

where the following convention has been used:

$$F(\boldsymbol{j}, k) = \prod_{i=1}^{n} f(j_i, k), \tag{95}$$

and the final index vector in the nested sum is defined by the final indices of the sum in each term t_i: $\boldsymbol{n} = (n_1, n_2, \ldots n_n)$.

Newly, a very important universal result is obtained: *The general form of the product resembles the primitive structure of the building blocks* t_i. Thus powerful mnemotechnical rules can again be easily described, whenever formulae with the above conditional structure can be written. This new formulation can be immediately applied, for example, to the notation of product of polynomials, the product of binomial powers, products of linear combinations or the product of an arbitrary number of power series.

C.6. GENERAL PRODUCT OF AN ARBITRARY NUMBER OF MATRICES OF ARBITRARY DIMENSION

Suppose one has $N + 1$ matrices $\boldsymbol{A}^{(0)}, \boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}$ with dimensions $(n_0 \times n_1)$, $(n_1 \times n_2), \ldots, (n_N \times n_{N+1})$, respectively. Their product can be constructed from the expression of the resulting matrix elements:

$$\left\{ \prod_{i=0}^{N} \boldsymbol{A}^{(i)} \right\}_{pq} = \sum_N (\boldsymbol{j} = \mathbf{1}, \boldsymbol{n}) \alpha(p, \boldsymbol{j}, q), \quad \forall p = 1, n_0, \quad \forall_q = 1, n_{N+1}, \tag{96}$$

the α terms being written as

$$\alpha(p, \boldsymbol{j}, q) = a^{(0)}_{p, j_1} \left[\prod_{i=1}^{N-1} a^{(i)}_{j_i, j_{i+1}} \right] a^{(N)}_{j_n, q}, \tag{97}$$

and where the vector $\boldsymbol{n}$ has the following structure: $\boldsymbol{n} = (n_1, n_2, \ldots, n_N)$.

Appendix D. General Gaussian product theorem (GGPT) proof

Let a set of N s-Gaussian functions be placed at the centers $\{\boldsymbol{A}_i\}_{i=1,N}$ and having as scaling factors $\boldsymbol{a} = \{\alpha_i\}_{i=1,N}$: $\{\gamma(\boldsymbol{A}_i, 0, \alpha_i)\}_{i=1,N}$. The GGPT states that their product is another s-Gaussian function expressed as

$$\prod_{i=1}^{N} \gamma(\boldsymbol{A}_i, 0, \alpha_i) = \kappa\gamma(\boldsymbol{P}, 0, \langle a \rangle) , \tag{98}$$

where $\langle \boldsymbol{a} \rangle$ is the sum of the elements of vector $\boldsymbol{a}$, according to the convention used in this paper, and the point $\boldsymbol{P}$ is defined by the weighted mean:

$$\boldsymbol{P} = \langle \boldsymbol{a} \rangle^{-1} \sum_{i=1}^{N} \alpha_i \boldsymbol{A}_i , \tag{99}$$

where $\kappa = \exp(\langle \boldsymbol{K}_N \rangle)$, and the $\boldsymbol{K}_N$ vector elements are defined, in an equivalent from to as in eq. (67), as

$$K_{Nq} = \langle \boldsymbol{a} \rangle^{-1} \sum_{i=1}^{N} \left\{ A_{iq}^2 \alpha_i (\langle \boldsymbol{a} \rangle - \alpha_i) - 2 \sum_{j=1}^{N} \delta(j > i) A_{iq} A_{jq} \alpha_i \alpha_j \right\}, \quad \forall q = 1, 3 , \tag{100}$$

where the vectors $\boldsymbol{A}_i = \{A_{iq}\}_{q=1,3}$.

The GGPT can be proved working with the exponential terms which are implicit in eq. (98) and due to the separability of all these terms with respect to each coordinate, it is only necessary to work out only a generic coordinate q.

The proof can be done by means of the induction method:

1. The GGPT holds for $N = 1$ if it is supposed that $K_{1q} = 0$. Also, the GGPT formulation is true for $N + 2$, since it gives

$$P_q = (\alpha_1 + \alpha_2)^{-1} (\alpha_1 A_{1q} + \alpha_2 A_{2q}) , \tag{101}$$

and using eq. (100) one obtains

$$K_{2q} = (\alpha_1 + \alpha_2)^{-1} \alpha_1 \alpha_2 (A_{1q} - A_{2q})^2 , \tag{102}$$

arriving to the same result as the one given for the well-known two Gaussian product theorem (TGPT) [24].

2. One can assume that the theorem is valid for $N = k$, that is when a product of k s-Gaussian functions is performed, it holds that:

$$K_{kq} = \langle \boldsymbol{a} \rangle^{-1} \sum_{i=1}^{k} \sum_{j=1}^{k} \delta(j > i) \alpha_i \alpha_j (A_{iq} - A_{jq})^2 , \tag{103}$$

$$P_q = \langle \boldsymbol{a} \rangle^{-1} \sum_{i=1}^{k} \alpha_i A_{iq} . \tag{104}$$

3. Now it is necessary to prove that a similar relationship to the one expressed in the preceding equations holds for $N = k + 1$, that is, adding a new function to the product of k Gaussians. The product of these $k + 1$ functions gives

$$P_{k+1,q} = (\langle \boldsymbol{a} \rangle + \alpha_{k+1})^{-1} \langle \boldsymbol{a} \rangle P_{k,q} + \alpha_{k+1} A_{k+1,q} , \tag{105}$$

or

$$P_{k+1,q} = \langle \boldsymbol{a}_{k+1} \rangle^{-1} \sum_{i=1}^{k+1} \alpha_i A_{iq} , \tag{106}$$

where it has been defined that

$$\langle \boldsymbol{a}_{k+1} \rangle = \sum_{i=1}^{k+1} \alpha_i . \tag{107}$$

It only remains to find the expression for $K_{k+1,q}$.

The exponential term involved in the product of the $k+1$ Gaussians is

$$K_{Nq} - \langle \boldsymbol{a} \rangle (r_q - P_q)^2 - \alpha_{k+1} (r - A_{k+1,q})^2 . \tag{108}$$

Applying the TGPT to the two right-most terms, the $k+1$ s-Gaussian functions product is generated: it is a new s-Gaussian function centered at $P_{k+1,q}$ and the pre-multiplicative factor has as exponent:

$$M = \langle a_{k+1} \rangle^{-1} \langle \boldsymbol{a} \rangle \alpha_{k+1} (P_q - A_{k+1,q})^2 + \langle \boldsymbol{a} \rangle^{-1} \sum_{i=1}^{N} \sum_{j=1}^{N} \delta(j>i) \alpha_i \alpha_j (A_{iq} - A_{jq})^2 . \tag{109}$$

It is needed to prove that $M = K_{k+1,q}$. It can be shown by performing the sum appearing in (109) and using the multinomial formula:

$$\left(\sum_{i=1}^{n} x_i \right)^2 = \sum_{i=1}^{n} x_i^2 + 2 \sum_{i=1}^{N} \sum_{j=1}^{N} \delta(j>i) x_i x_j , \tag{110}$$

taken from eq. (92) with $m = 2$, and the relationship:

$$\sum_{i=1}^{N} \sum_{j=1}^{N} \delta(j>i) \alpha_i \alpha_j (A_{iq}^2 + A_{jq}^2) = \sum_{i=1}^{N} A_{iq}^2 \alpha_i (\langle \boldsymbol{a} \rangle - \alpha_i) . \tag{111}$$

One can now show that

$$M = \langle \boldsymbol{a}_{k+1} \rangle^{-1} \sum_{i=1}^{k+1} \sum_{j=1}^{k+1} \delta(j>i) \alpha_i \alpha_j (A_{iq} - A_{jq})^2 = K_{j+1,q} , \tag{112}$$

and the GGPT is proved.

References

[1] (a) R. Carbó and J.M. Riera, *A General SCF Theory* (Springer, Berlin, 1978). (b) R. Carbó and O. Gropen, Adv. Quant. Chem. 12 (1980) 159. (c) R. Carbó, Ll. Domingo and J.J. Peris, Adv.

Quant. Chem. 15 (1982) 215. (d) R. Carbó, J. Miró, J.J. Novoa and Ll. Domingo, Adv. Quant. Chem. 20 (1989) 375.

[2] R. Carbó and B. Calabuig, in: *Quantum Chemistry, Basic Aspects, Actual Trends. Studies in Physical and Theoretical Chemistry*, R. Carbó (ed.) Vol. 62 (Elsevier, Amsterdam, 1989) p. 73.

[3] R. Carbó and J.A. Hernández, Chem. Phys. Lett. 47 (1977) 85.

[4] (a) R. Carbó and E. Besalú, Adv. Quant. Chem. 24 (1992) 115. (b) R. Carbó and E. Besalú, Can. J. Chem. 70 (1992) 353.

[5] R. Carbó and E. Besalú, Comp. Chem., in press.

[6] R. Carbó and C. Bunge, PC Actual Magazine (September 1989) 124.

[7] (a) VAX Fortran Document AA-D034D-TE (Digital Equipment Corporation, Maynard, Mass, 1984) p. E28. (b) NDP Fortran 386 v3.0, *User's Manual* (Microway, Kingston, 1990). (c) NDP Fortran 486 v4.0.2, *User's Manual and Library* (Microway, Kingston, 1992). (d) Lahey Fortran V4.0, *Language Reference* (Lahey Computer Systems, Incline Village, 1990).

[8] (a) F. Ayers, *Theory and Problems of Matrices* (Schaum, New York, 1962). (b) R. Carbó and Ll. Domingo, *Algebra Matricial y Lineal* (McGraw-Hill, Madrid, 1987).

[9] J.H. Wilkinson and C. Reinsch, *Linear Algebra* (Springer, Berlin 1971).

[10] R.G. Parr, *The Quantum Theory of Molecular Electronic Structure* (Benjamin, New York, 1963).

[11] K. Jankowski, Electron correlation in atoms, in: *Methods in Computational Chemistry*, Vol. 1, *Electron Correlation in Atoms and Molecules*, ed. S. Wilson (Plenum Press, New York, 1987) p. 1.

[12] A. Szabo and N.S. Ostlund, *Modern Quantum Chemistry* (McGraw-Hill, New York, 1989).

[13] (a) B. Roos, The configuration interaction method, in: *Computational Techniques in Quantum Chemistry and Molecular Physics*, Vol. D, eds. G.H.F. Diercksen, B.T. Sutcliffe and A. Veillard (Reidel, Dordrecht, 1975) p. 251. (b) I. Shavitt, The method of configuration interaction, in: *Methods of Electronic Structure Theory*, Vol. 3, ed. H.F. Schaefer (Plenum Press, New York, 1977). (c) P.E.M. Siegbahn, The externally contracted CI method, in: *Current Aspects of Quantum Chemistry 1981*, ed. R. Carbó (Elsevier, Amsterdam, 1982) p. 65. (d) P.E.M. Siegbahn, The direct CI method, in: *Methods in Computational Molecular Physics*, eds. G.H.F. Diercksen and S. Wilson (Reidel, Dordrecht, 1983) p. 189. (e) P.J. Knowles and N.C. Handy, J. Chem. Phys. 91 (1989) 2396.

[14] (a) P.O. Löwdin, Phys. Rev. 97 (1955) 1474. (b) R. McWeeny, Proc. Roy. Soc. A232 (1955) 114. (c) R. McWeeny, Proc. Roy. Soc. A 235 (1956) 496.

[15] (a) R. Carbó, Theor. Chim. Acta 17 (1970) 74. (b) R. Carbó, Rev. Roum. Chim. 16 (1971) 1155. (c) R. Carbó and R. Gallifa, Nuovo Cimento 10 (1972) 576. (d) R. Carbó, Int. J. Quant. Chem. 6 (1972) 609. (e) R. Carbó and R. Gallifa, Anal. Física 68 (1972) 197. (f) R. Carbó and R. Gallifa, Nuovo Cimento 17 (1973) 46. (g) R. Carbó and R. Gallifa, Anal. Física 69 (1973) 331.

[16] (a) L. Pauling and E.B. Wilson, jr., *Introduction to Quantum Mechanics* (McGraw-Hill, New York, 1935). (b) H. Eyring, J. Walter and G.E. Kimball, *Quantum Chemistry* (Wiley, New York, 1948). (c) C.H. Wilcox (ed.), *Perturbation Theory and its Applications in Quantum mechanics* (Wiley, New York, 1966). (d) F.L. Pilar, *Elementary Quantum Chemistry* (McGraw-Hill, New York, 1968). (e) R. Carbó and J.A. Hernández, *Introducción a la Teoría de Matrices* (Editorial Alhambra, S.A. Madrid, 1983).

[17] J.O. Hirschfelder, Int. J. Quant. Chem. 3 (1969) 731.

[18] J.O. Hirschfelder, in ref. [16c] p. 3: Applicability of perturbation theory to molecular problems.

[19] R. Carbó and E. Besalú, J. Math. Chem. 13 (1993) 331.

[20] E. Besalú and R. Carbó, J. Math. Chem. 15 (1994) 397–406.

[21] W. Kutzelnigg, Theor. Chim. Acta 83 (1992) 263.