International Journal of Theoretical Physics, Vol. 40, No. 1, 2001

Symbolic Package for Quantum Groups, Noncommutative Algebras, and Logics[1]

Hanna Makaruk[2]

Received October 2, 1999; revised April 21, 2000

This paper describes packages for symbolic calculations in quantum groups, noncommutative differential geometry, and multivalued logic. The package for quantum groups and the program for logic are written in *Mathematica* 3.0 and/or 4.0. As an example, some results in the logic obtained using these packages are presented.

1. INTRODUCTION

Programs like *Mathematica* and *Maple* help with symbolic calculations. Commercial programs cover the standard mathematics widely used in science, engineering, medicine, or administration, but not new and developing branches of mathematics like quantum groups (Woronowicz, 1989), or braided logic (Chávez Rodríguez *et al.*, 1999, 2000).

It is difficult to overestimate the utility of symbolic programs for doing extensive calculations in quantum groups, noncommutative differential geometry, and braided logic. Although there are some programs for symbolic calculations in these areas, they are oriented toward particular calculations, and are not universal tools. The present package is designed to answer these needs.

2. PACKAGE FOR QUANTUM GROUPS AND NONCOMMUTATIVE ALGEBRAS

Micho Đurđevich is a coauthor of the computer package presented below. It is assumed the package will be shared with other users, which puts certain demands on its design:

[1] This work was supported by Universidad Nacional Autónoma de México, DGAPA, Programa de Apoyo a Proyectos de Investigación e Innovación Tecnológica, Proyecto IN-109599 (2000-2002), and by Programa de Apoyo a Proyectos de Desarrollo e Investigación en Informática 2000.

[2] Los Alamos National Laboratory, E-ET, Mail Stop E-517, Los Alamos, New Mexico 87545; e-mail: hanna_m@lanl.gov. On leave from Polska Akademia Nauk, Instytut Podstawowych Problemów Techniki, PL-00-049 Warsaw, Poland.

0020-7748/01/0100-0105$19.50/0 © 2001 Plenum Publishing Corporation

• There should be compatibility with the current version of *Mathematica* to allow a user to enjoy all the support provided by *Mathematica* and by other packages written for it. We avoided repetition of structures provided by other complementary packages in *Mathematica*, like Gröbner bases, differential one-forms in commutative differential geometry of one-sided modules, etc.

• Newly introduced objects should follow the convention of definitions in *Mathematica*. They also have the same kind of help as other objects in *Mathematica*, called by "?".

• Design and programming of the package should be easy to follow for a mathematician, not only for a programmer. The program is built from sets of definitions and sets of local rules. Loops, subprograms, and local variables are completely avoided in the main body of the program. They support only the graphical interface of the program (colorful menu, indices in Sweedler notation). This interface is optional; some users would probably welcome these additions, others would rather skip them and use only the core of the program, which is transparent to follow, and computes faster.

• The program follows the notation used in the literature (Woronowicz, 1987, 1989; Đurđevich, 1995, 1996, 1997, 1999, 2000a, b). Choices were unavoidable when two or more conventions coexist in the literature. Following the notation from the literature resulted in overloading some symbols like δ, Π. They have different meanings in different contexts. We deliberately made this choice, simultaneously trying to avoid any ambiguity in the program. Such cases are documented in the Help.

• The program runs on any computer with *Mathematica* 3.0 or higher. Efforts were made to keep the program fast and low in memory demand in order to be useful for small personal computers.

• The program is written on two levels: (a) to be used for a wide range of predefined noncommutative calculations, and also to use special libraries like for $S_qU(2)$, quantum torus, quantum Euclidean space, etc.; and (b) to allow the user to define and check his or her own noncommutative objects, like new algebras, define the relations inside it and check the consistency, or to expand the program for whatever other reason.

Keeping the structure based on definitions and sets of rules, we had in mind mainly the second type of user who can follow this structure in writing his or her sets of definitions and relations.

• For users who do not attempt to extend the program, but simply use the existing part, we recommend a menu which makes it possible to use the program without going through its structure.

• The program is *not* designed (1) to run under *Maple*, (2) to use a lower version of *Mathematica* than 3.0, (3) to stop calculations when a user

wants to apply an operation which is mathematically not allowed; a random answer may appear in such a case; (4) to perform nonalgorithmizable operations; it is a classical computer program; and (5) to simplify some expressions automatically; it needs one or two steps done by hand before further executing automatic simplifications. Usually this means removing superficial sets of parentheses. Implementing special rules for any combination of superficial parentheses which may potentially occur would extend the program tremendously, compromising both clarity and performance.

2.1. The Content of the Program

All symbols not declared as noncommutative are treated as commutative. Any symbol can be declared to be noncommutative either by putting it in parentheses or by making it equal to a symbol in parentheses. Three kinds of noncommutative multiplications with appropriate power operations are defined, including rules for interaction with other mathematical operations (distributivity of multiplications with respect to summation; putting a commutative coefficient in front of an expression; associativity of multiplications). Each multiplication basically had its own power operator, but tests showed that it was more convenient to overload one power operator; there is usually no ambiguity in which multiplication was shortened to the power, and which set of rules to call in the case of expanding the power. The version with different power operators for different multiplications is included in the package.

The multiplications are as follows:

· Multiplication of noncommutative elements

** Redefined to be a multiplication of matrices built of noncommutative elements

$\otimes$ Tensor operation for tensors built of noncommutative elements

Differentiation fulfilling Leibniz' rule and working with these noncommutative multiplications is defined. For commutative parts of expressions, it behaves as usual differentiation. Appropriate sets of transformation rules for expanding and for factorization of expressions are provided. Typical quantum group objects are provided: comultiplication, counit, identity operation, antipode, multiplication in an algebra, star operation in an algebra, etc. Libraries of rules for the most typical mathematical noncommutative structures, $S_qU(2)$, Euclidean plane, noncommutative torus, are included.

Matrices and multi-index objects built of noncommutative elements can be used; appropriate definitions are provided. Noncommutative differential one-forms (from bimodule) and appropriate sets of relations are defined. The projection operation is predefined.

The package is designed for interactive calculations; a user has to decide which part of the sequence of application of sets of rules is to be applied

and which form of presentation of the answer is desired. Our goal was not to produce a program which calculates things by itself—this would narrow the possible applications—but rather to expand the possibilities of symbolic programming in *Mathematica*, to include noncommutative objects and treat specific algebraic operations like a star in an algebra, comultiplication, etc., on an equal basis with standard, commutative ones.

2.2. Sweedler Notation

Implementation of Sweedler notation in *Mathematica* is quite challenging because changes in one place of the expression cause changes in numeration in other parts of expression which are not involved in the particular operation. We had to compromise, i.e., there is a program for calculation in Sweedler notation or, to be precise, two twin programs: one for users who prefer upper Sweedler notation, and the other, which is its exact counterpart, invented for those who prefer original lower Sweedler notation. The program for notation has to have some elements insuring proper graphical representation of the expression, a tool to check if a number in any part of the expression which was not involved in particular operation was appropriately changed, and so on. We ended up with a program which works, but we had to give up the demand of building everything from definitions and local rules in an easy-to-follow way. This is the only part in the whole package whose manner of working we do not explain—it is not beautiful, it is just robust enough to work. This part of the program code is also available for the user, but is not as easy to modify or expand as the main program.

There is rising interest in exploring all (braided) bigebra structures for a fixed given well-known algebra, like Clifford algebra (Đurđevich and Oziewicz, 1996; Oziewicz, 1997; Đurđevich, 2000). Another related important problem is to determine all coproducts which allow an antipode to exist for the given product with unit. These problems lead us directly into the braided categories with all problems related to explicit solutions of the Yang–Baxter equation. The main obstacle in this area of the finite-dimensional linear algebra is the fact that all logical calculations widely used in braided geometry are enormously time-consuming. On the other hand, they are fully algorithmizable. This is also an ideal area for symbolic computations on a computer.

3. LOGIC

There is interest in exploring braided logic and related problems (Chávez Rodríguez *et al.*, 1999, 2000). The problem could have applications in quantum computers which allow for multivalued logic. The main obstacle in the area of braided logic is the fact that all logical calculations widely used in

logic are enormously time-consuming. On the other hand, they are fully algorithmizable. This is an ideal area for symbolic computations on a computer.

There is an encouraging fact that the majority of typical grafted operations in universal algebra and logic (as in the example in the next subsection) can be performed in *Mathematica* simply as a nested operation of calling a list element. From the programming point of view, this is one of the most efficient techniques in symbolic programming. This program does not deal with the Artin relation since a program for computing this relation already exists (Chávez Rodríguez and López Gómez, 2000). Grafted operations are important in logic. The program is rather a system of rules and operations, which are computer counterparts of typical nodes in logical diagrams.

The program is written in a form that makes it possible to do calculations in multivalued logic. To be consistent in notation, for multivalued logic, the program denotes values by natural numbers for any logic: 2-valued logic: 1 and 2; n-valued logic: $1, \ldots, n$.

Every binary operation is given in terms of a matrix. The way to generate a table built of matrices of all binary operations defined in the n-valued logic in *Mathematica* is

$$A_n = \mathit{Table}[\{\{i_1, \ldots, i_n\}, \{i_{n+1}, \ldots, i_{2n}\}, \ldots, \{i_{n^n-n+1}, \ldots, i_{n^n}\}\}, \{i_1, n\}, \ldots, \{i_n, n\}] \quad (1)$$

where dots means that some elements of the code were skipped. In *Mathematica*, they have to be written in the code explicitly. Each binary operation of the n-valued logic is described by an $n \times n$ matrix. Each matrix defines the outcome of one logical binary operation. Here is how we read the matrix: the columns are numbered by the logical value of the first element, the rows are numbered by the logical value of the second element, and at the column−row intersection, the program writes the logical output of particular binary operation. The n-valued logic has n^{nn} binary operations, i.e., $2^4 = 16$ for 2-valued logic, $3^9 = 19{,}683$ for 3-valued logic, $4^{16} = 4{,}294{,}967{,}296$ for 4-valued logic.

The matrices of the 16 logical binary operations for 2-valued logic are numbered in a (1)-way as follows:

$$1)\begin{Bmatrix} & t & f \\ t & t & t \\ f & t & t \end{Bmatrix} \quad 2)\begin{Bmatrix} & t & f \\ t & t & t \\ f & t & f \end{Bmatrix} \quad 3)\begin{Bmatrix} & t & f \\ t & t & t \\ f & f & t \end{Bmatrix} \quad 4)\begin{Bmatrix} & t & f \\ t & t & t \\ f & f & f \end{Bmatrix}$$

$$5)\begin{Bmatrix} & t & f \\ t & t & f \\ f & t & t \end{Bmatrix} \quad 6)\begin{Bmatrix} & t & f \\ t & t & f \\ f & t & f \end{Bmatrix} \quad 7)\begin{Bmatrix} & t & f \\ t & t & f \\ f & f & t \end{Bmatrix} \quad 8)\begin{Bmatrix} & t & f \\ t & t & f \\ f & f & f \end{Bmatrix}$$

9) $\begin{Bmatrix} & t & f \\ t & f & t \\ f & t & t \end{Bmatrix}$ 10) $\begin{Bmatrix} & t & f \\ t & f & t \\ f & t & f \end{Bmatrix}$ 11) $\begin{Bmatrix} & t & f \\ t & f & t \\ f & f & t \end{Bmatrix}$ 12) $\begin{Bmatrix} & t & f \\ t & f & t \\ f & f & f \end{Bmatrix}$

13) $\begin{Bmatrix} & t & f \\ t & f & f \\ f & t & t \end{Bmatrix}$ 14) $\begin{Bmatrix} & t & f \\ t & f & f \\ f & t & f \end{Bmatrix}$ 15) $\begin{Bmatrix} & t & f \\ t & f & f \\ f & f & t \end{Bmatrix}$ 16) $\begin{Bmatrix} & t & f \\ t & f & f \\ f & f & f \end{Bmatrix}$

Compare with the tables in Chávez Rodríguez and López Gómez, (2000), where binary operations are classified with respect to essentiality and associativity.

Matrices are available from the author of all binary operations A_n for 3-valued logics. The diagrams for 4- and higher valued logics, generated by the same program, should be used as a step in computer calculations when needed. We would rather not report all of them in any explicit printed versions due to the size of the table A_n, but it is still relatively easy to obtain from the program for A_n what is the matrix form of a particular binary operation number k ($k \leq n$).

The table A_n defines uniquely the order of binary operations in an n-valued logic. We will assume this order further in the paper, and refer to a binary operation by its number in the appropriate A_n table. Unfortunately, this order does not take into account the essentiality of the binary operation, for example, the constant binary operations are not grouped together in one bloc.

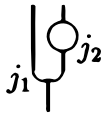
Higher type operations exist also, e.g., ternary operations and generally n-ary operations. Each n-ary operation in m-valued logic is represented by an n-index matrix $m \times m \times \ldots \times m$. The set of all such operations can be calculated and used analogously to the ones discussed for binary operations. All the operations can be done in exactly the same way as for binary operations, with one obvious change—there would be correspondingly more internal indices for summation. The design of the package is such that these operations can be easily and consistently updated. What stops us from implementing this part explicitly in this version of the package is the high demand for computer power. One can easily modify the program to improve performance many times. But even 100 times is still not a solution when dealing with contemporary computers.

3.1. Example: Binary Operations on Binary Operations

Let $j \equiv \{j_i | i = 1, \ldots, n^{nn}\}$ be a set of all binary operations on n-element carriers. In this example, we are interested in the binary operations on this set j, $\{B: j \times j \rightarrow j\}$. Each binary operation B on j can be given

by a grafted graph. As in Chávez Rodrígez and López Gómez (2000), graphs must be read from the top (inputs) to the bottom (output). All the combinations of logical binary operations from a set j such that we have two inputs (in our convention on the top) and one output (on the bottom) are equivalent to one of the elementary operations from a set j for which the matrix is an element of the table A_n, no matter how complicated the graph (diagram) is.

Let us consider, as an example, the following grafted binary operation $B(j_1, j_2) \in j$, where j_1 and j_2 as variables run through the whole set j of all binary operations:



The not labeled first node means the diagonal duplication $a \mapsto (a, a)$. The answer will be returned in the form of matrices B, where value of the element $B[j_1, j_2] \in j$ is the number of operations to which such a diagram with particular operations j_1 and j_2 is equal. In our example, the logical operations are numbered as in table A_n from (1). Alternative numbering, related to specific fibration of the set j, could be more appropriate; however, this is not considered here (Chávez Rodríguez and López Gómez, 2000).

The program for the above grafted graph B is

$$r = Table[rr, \{i, n\}, \{j, n\}] \tag{2}$$

n is the valency of the logic; (2) is the definition of an $n \times n$ matrix named $r \in j$; i, j, k, m are summation indices; A_n is a three-index table built of matrices of all the operations in the n-valued logic; and $b[., .]$ is an auxiliary function,

$$\begin{aligned} &Do[\{Do[r[[\, i, j\,]] = A[[l, A[[\, k, i, j\,]], j\,]], \{i, n\}, \{j, n\}] \\ &\quad If[r = A[[m]], b[k, l] = m]\}, \{k, n\}, \{l, n\}, \{m, n\}] \end{aligned} \tag{3}$$

This double loop calls appropriate elements of the operation matrices and, in the "if" operation, reads the operation to which the result is equivalent.

The next operation is for representation of the results in the form of the answer matrix B,

$$B = Table[b[\, i, j\,], \{i, n\}, \{j, n\}] \tag{4}$$

Table B (the answer matrix) of solutions for the above example of a grafted 3-node graph $B(j_1, j_2) \in j$ for the 2-valued logic in our convention (1) is as follows:

$$B = \begin{Bmatrix}
1 & 1 & 1 & 1 & 6 & 6 & 6 & 6 & 11 & 11 & 11 & 11 & 16 & 16 & 16 & 16 \\
1 & 2 & 1 & 2 & 5 & 6 & 5 & 6 & 11 & 12 & 11 & 12 & 15 & 16 & 15 & 16 \\
1 & 1 & 3 & 3 & 6 & 6 & 8 & 8 & 9 & 9 & 11 & 11 & 14 & 14 & 16 & 16 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\
1 & 5 & 1 & 5 & 2 & 6 & 2 & 6 & 11 & 15 & 11 & 15 & 12 & 16 & 12 & 16 \\
1 & 6 & 1 & 6 & 1 & 6 & 1 & 6 & 11 & 16 & 11 & 16 & 11 & 16 & 11 & 16 \\
1 & 5 & 3 & 7 & 2 & 6 & 4 & 8 & 9 & 13 & 11 & 15 & 10 & 14 & 12 & 16 \\
1 & 6 & 3 & 8 & 1 & 6 & 3 & 8 & 9 & 14 & 11 & 16 & 9 & 14 & 11 & 16 \\
1 & 1 & 9 & 9 & 6 & 6 & 14 & 14 & 3 & 3 & 11 & 11 & 8 & 8 & 16 & 16 \\
1 & 2 & 9 & 10 & 5 & 6 & 13 & 14 & 3 & 4 & 11 & 12 & 7 & 8 & 15 & 16 \\
1 & 1 & 11 & 11 & 6 & 6 & 16 & 16 & 1 & 1 & 11 & 11 & 6 & 6 & 16 & 16 \\
1 & 2 & 11 & 12 & 5 & 6 & 15 & 16 & 1 & 2 & 11 & 12 & 5 & 6 & 15 & 16 \\
1 & 5 & 9 & 13 & 2 & 6 & 10 & 14 & 3 & 7 & 11 & 15 & 4 & 8 & 12 & 16 \\
1 & 6 & 9 & 14 & 1 & 6 & 9 & 14 & 3 & 8 & 11 & 16 & 3 & 8 & 11 & 16 \\
1 & 5 & 11 & 15 & 2 & 6 & 12 & 16 & 1 & 5 & 11 & 15 & 2 & 6 & 12 & 16 \\
1 & 6 & 11 & 16 & 1 & 6 & 11 & 16 & 1 & 6 & 11 & 16 & 1 & 6 & 11 & 16
\end{Bmatrix}$$

An analogous answer matrix for the 3-valued logic of the size 19,683 $\times$ 19,683 is also computed.

Any other binary grafted graph $j \times j \to j$ is represented in the program in a very similar way. The core of the calculations is to represent a diagram as a nested calling of elements from the table A_n with appropriate summation indices. The matrix r plays an auxiliary role in finding to which operation the diagram is equivalent. The last line of the code (4) is simply for returning the results in the form of a matrix when it is convenient.

In the case of more complicated grafted graphs with 3 and more nodes from j, for example, in the case of the ternary grafted graph $j \times j \times j \to j$, to which any operation can be applied, the answer is presented not as a matrix, but as a list of lists: $\{j_1, j_2, \ldots j_k, answer\}$, where first k places refer to operations on k nodes of the diagrams and the last place returns the statement to which binary operation such a graph is equal.

A small change in the program allows for returning the answer not as a diagram random number, but as the logical values on the exits as the functions of the logical values on enters and of operations performed on every node. This kind of output is usually long and difficult to follow, but sometimes unavoidable, for example, in the case of complicated graphs with many entrances and more than one exit, like these entering into the Artin braid relation (Chávez Rodríguez and López Gómez, 2000).

The program is written in such a way that as many operations as possible in *Mathematica* are executed just by calling an appropriate element of a list,

which makes the program relatively fast and low in memory requirements. The version of the program for the 2-valued logic runs on any computer with *Mathematica* and computation is so fast that it usually can be performed in a dialogue mode. The version for the 3-valued logic has also been run by the author on a home PC, but the computation time takes weeks even for relatively simple diagrams.

ACKNOWLEDGMENTS

Dr. Micho Đurđevich is coauthor of the computer package presented here and I appreciate collaboration with him on the preparation of the package for calculations in quantum groups as well as for many fruitful discussions and much encouragement.

I would like to thank Prof. Stanisław Lech Woronowicz for continued encouragement to write the quantum group part of the package and to Prof. Zhigniew Oziewicz for stating the problem of braided logic computer calculations and helpful discussions. I also appreciate discussions with Dr. Robert Owczarek.

This work was undertaken at Los Alamos National Laboratory, operated by the University of California for the U.S. Department of Energy.

REFERENCES

Chávez, Rodríguez María Ernestina, Ewa Graczyńska, Angel López Gómez, and Zbigniew Oziewicz (1999). Braided logic and clone of plants, presented at the AMS-SMM Meeting, Denton, May 1999.

Chávez, Rodríguez María Ernestina, and Angel López Gómez (2001). Braided logic: The simplest models, *International Journal of Theoretical Physics*, **40**, 95–103.

Đurđevich, Micho (1995). Classical spinor structures on quantum spaces, in: Rafał Abłamowicz, Editor, *Clifford Algebras and Spinor Structures*, Kluwer, Dordrecht, pp. 365–377; http://www.matem.unam.mx/micho.

Đurđevich, Micho (1996). Geometry of quantum principal bundles I, *Communications in Mathematical Physics*, **175**, 457–520.

Đurđevich, Micho (1997). Geometry of quantum principal bundles II, *Reviews of Mathematical Physics*, **9**, 531–607.

Đurđevich, Micho (1999). General frame structures on quantum principal bundles, *Reports on Mathematical Physics*, **44**, 53–70.

Đurđevich, Micho (2001a). Quantum spinor structures for quantum spaces, *International Journal of Theoretical Physics*, **40**, 115–138.

Đurđevich, Micho (2001b). Braided Clifford algebras as quantum deformations, *International Journal of Theoretical Physics*, **40**, 15–24.

Đurđevich, Micho, and Zbigniew Oziewicz (1996). Clifford algebras and spinors for arbitrary braids, in: Julian Lawrynowicz, Editor, *Generalizations of Complex Analysis*, Banach Center Publications, Warsaw, pp. 315–325.

Graczyńska, Ewa, and Zbigniew Oziewicz (1999). Birkhoff's theorems via tree operad, *Bulletin of the Section of Logic of the Polish Academy of Science* (University Łódź), **28**, 159–170; http://www1.uni.lodz.pl/bulletin/v283.html.

Oziewicz, Zbigniew (1997). Clifford Hopf-gebra and bi-universal Hopf-gebra, *Czechoslovak Journal of Physics*, **47**, 1267–1274.

Woronowicz, Stanisław (1987). Compact matrix pseudogroups, *Communications in Mathematical Physics*, **111**, 613–665.

Woronowicz, Stanisław (1989). Differential calculus on compact matrix pseudogroups (quantum groups), *Communication in Mathematical Physics*, **122**, 125–170.