Iterated Quantum Products, Sums, and Infinite Sums

by José Luis Gómez-Muñoz
http://homepage.cem.itesm.mx/lgomez/quantum/
jose.luis.gomez@itesm.mx
With contributions by Jean-Daniel Bancal

Introduction

This is a tutorial on the use of Quantum Mathematica add-on to work with iterated quantum products $\otimes$ and sums $\sum$ of expressions in Dirac notation.

Load the Package

First load the Quantum'Notation' package. Write:

\[ \text{Needs["Quantum'Notation"];} \]

then press at the same time the keys $\text{Shift}+\text{Enter}$ to evaluate. Mathematica will load the package:

\[
\text{Needs["Quantum'Notation"]}
\]

Quantum'Notation' Version 2.2.0, (July 2010)
A Mathematica package for Quantum calculations in Dirac bra-ket notation
by José Luis Gómez-Muñoz

Execute SetQuantumAliases[] in order to use
the keyboard to enter quantum objects in Dirac's notation
SetQuantumAliases[] must be executed again in each new
notebook that is created, only one time per notebook.

In order to use the keyboard to enter quantum objects write:

\[ \text{SetQuantumAliases[];} \]

then press at the same time the keys $\text{Shift}+\text{Enter}$ to evaluate. The semicolon prevents Mathematica from printing the help message. Remember that SetQuantumAliases[ ] must be evaluated again in each new notebook:

\[ \text{SetQuantumAliases[;]} \]

Iterated Quantum Products $\otimes_{j=1}^{n} a_j$

This is a large quantum product entered in a naive way. Press $\text{ESC}eket\text{ESC}$ for each eigen-ket template, and press $\text{ESC}tp\text{ESC}$ for each infix tensor product symbol:
There is a more convenient notation for large quantum products. Press the keys:
\[\text{[ESC]}\text{qp[ESC]}\]
then press the key [TAB] several times in order to select the lowest "place-holder" (square). Then press the keys:
\[j=4[\text{[TAB]}8[\text{[TAB]}\text{[ESC]}\text{ket[ESC]}[\text{[TAB]}\text{[ESC]}\text{po[ESC]}\]
then press the key [TAB] several times to select the remaining leftmost "place-holder" (square). Then press the keys:
\[j[\text{[TAB]}2[\text{[TAB]}]\]
Finally press at the same time the keys \[\text{[SHIFT]}-\text{[ENTER]}\] to evaluate:

\[
\begin{array}{c|c}
8 & (j)_{3}^{2} \\
3=4 & \\
\hline
| 16_{6}, 25_{5}, 36_{6}, 49_{5}, 64_{8} \rangle
\end{array}
\]

The "InputForm" version is the command QuantumProduct, which has the same syntax as the standard Mathematica commands Product, Sum and Table:

\[
\text{QuantumProduct}[ | (j)_{3}^{2}], (j, 4, 8) ]
\]

\[
| 16_{6}, 25_{5}, 36_{6}, 49_{5}, 64_{8} \rangle
\]

An increment different from one can be specified using the "InputForm" version, QuantumProduct. It has the same syntax as standard Mathematica commands like Product, Sum and Table:

\[
\text{QuantumProduct}[ | (j)_{3}^{2}], (j, 4, 8, 2) ]
\]

\[
| 16_{6}, 36_{6}, 64_{8} \rangle
\]

If the initial value is omitted, then it is assumed to be one:

\[
\begin{array}{c|c}
8 & (j)_{3}^{2} \\
3=4 & \\
\hline
| 1_{4}, 4_{2}, 9_{3}, 16_{4}, 25_{5}, 36_{6}, 49_{5}, 64_{8} \rangle
\end{array}
\]

InputForm version giving the same result as above:

\[
\text{QuantumProduct}[ | (j)_{3}^{2}], (j, 8) ]
\]

\[
| 1_{4}, 4_{2}, 9_{3}, 16_{4}, 25_{5}, 36_{6}, 49_{5}, 64_{8} \rangle
\]

A specific list of index values can be given:
This is the quantum product of a quantum function (operator) evaluated at different values of its argument:

```
SetQuantumObject[q];
\[\otimes\] q[2 k]
\[\otimes\] q[2 k]
k=5
```

```
q[10] \cdot q[12] \cdot q[14] \cdot q[16] \cdot q[18] \cdot q[20] \cdot q[22]
```

InputForm version giving the same result as above:

```
SetQuantumObject[q];
QuantumProduct[q[2 k], {k, 5, 11}]
```

```
q[10] \cdot q[12] \cdot q[14] \cdot q[16] \cdot q[18] \cdot q[20] \cdot q[22]
```

Iterated quantum products can be calculated, notice that the upper limit of the inner index depends on the outer index:

```
SetQuantumObject[b];
\[\otimes\] b[k, j]
\[\otimes\] b[k, j]
k=3 \ j=1
```

```
b[3, 1] \cdot b[4, 1] \cdot b[4, 2] \cdot b[5, 1] \cdot b[5, 2] \cdot b[5, 3]
```

Here we obtain the same result as above using two nested QuantumProduct:

```
SetQuantumObject[b];
QuantumProduct[QuantumProduct[b[k, j], {j, 1, k - 2}], {k, 3, 5}]
```

```
b[3, 1] \cdot b[4, 1] \cdot b[4, 2] \cdot b[5, 1] \cdot b[5, 2] \cdot b[5, 3]
```

Here we obtain the same result as above using one QuantumProduct with two iterators. Notice the order of the iterators, which follows the conventions of the iterators used in standard Mathematica commands like Product, Sum and Table:
Complex products can be generated:

\[
\text{SetQuantumObject}[b];
\]
\[
\text{QuantumProduct}[b[k, j], \{k, 3, 5\}, \{j, 1, k - 2\}]
\]
\[
\text{b}[3, 1] \cdot \text{b}[4, 1] \cdot \text{b}[4, 2] \cdot \text{b}[5, 1] \cdot \text{b}[5, 2] \cdot \text{b}[5, 3]
\]

We clear the definitions used in the previous examples:

\[
\text{SetQuantumScalar}[b, q];
\]
\[
\text{Clear}[b, q]
\]

**Finite Sums**

You can calculate sums that include Quantum expressions. Press [ESC]si[ESC] for the sigma template. The keys [ESC]qs[ESC] (as in quantum sum) give the same template:

\[
\sum_{j=4}^{8} j_q\}
\]
\[
|4_q\rangle + |5_q\rangle + |6_q\rangle + |7_q\rangle + |8_q\rangle
\]

The "InputForm" version must be typed using the command "QuantumSum" instead of "Sum":

\[
\text{QuantumSum}[j_q, \{j, 4, 8\}]
\]
\[
|4_q\rangle + |5_q\rangle + |6_q\rangle + |7_q\rangle + |8_q\rangle
\]

An increment different from one can be specified using the "InputForm" version, QuantumSum. It has the same syntax as standard Mathematica commands like Product, Sum and Table:

\[
\text{QuantumSum}[j_q, \{j, 4, 8, 2\}]
\]
\[
|4_q\rangle + |6_q\rangle + |8_q\rangle
\]

Iterated sums:
\[
\sum_{k=1}^{2} \sum_{j=1}^{3} \sum_{i=1}^{4} | i_2 \rangle \cdot (| j + k \rangle_2) \\
| 1_2 \rangle \cdot (| 2_2 \rangle + | 3_2 \rangle) \\
2 | 3_2 \rangle \cdot (| 5_2 \rangle + | 4_2 \rangle) \\
2 | 4_2 \rangle \cdot (| 4_2 \rangle + | 5_2 \rangle) \\
\]

And the TraditionalForm of the expression can be easier to visualize:

\[
\text{TraditionalForm} \left[ \sum_{k=1}^{2} \sum_{j=1}^{3} \sum_{i=1}^{4} | i_2 \rangle \cdot (| j + k \rangle_2) \right] \\
| 1_2 \rangle (| 2_2 \rangle + | 3_2 \rangle) + | 2_2 \rangle (| 2_2 \rangle + | 3_2 \rangle + | 4_2 \rangle + | 5_2 \rangle + | 4_2 \rangle + | 5_2 \rangle) + | 3_2 \rangle (| 4_2 \rangle + | 5_2 \rangle) + | 4_2 \rangle (| 4_2 \rangle + | 5_2 \rangle) + | 5_2 \rangle (| 4_2 \rangle + | 5_2 \rangle) \\
\]

The "InputForm" version must be typed using the command "QuantumSum" instead of "Sum":

\[
\text{QuantumSum} \left[ | i_2 \rangle \cdot (| j + k \rangle_2) \right] , \{ k, 1, 2 \} , \{ j, 1, 3 \} , \{ i, 1, 4 \} \\
| 1_2 \rangle (| 2_2 \rangle + | 3_2 \rangle) + | 2_2 \rangle (| 2_2 \rangle + | 3_2 \rangle + | 4_2 \rangle + | 5_2 \rangle + | 4_2 \rangle + | 5_2 \rangle) + | 3_2 \rangle (| 4_2 \rangle + | 5_2 \rangle) + | 4_2 \rangle (| 4_2 \rangle + | 5_2 \rangle) + | 5_2 \rangle (| 4_2 \rangle + | 5_2 \rangle) \\
\]

Here is an example where the expressions depends on only one summation index and the lower and upper limits are symbolic. It is slow, it takes several seconds to evaluate:

\[
\sum_{k=1}^{p} \sum_{j=1}^{a} \sum_{i=1}^{m} | j_2 \rangle \cdot (| j + 1 \rangle_2) \\
(1 - c + m) (1 - a + p) \sum_{j=1}^{a} | j_2 \rangle \cdot (| 1 + j \rangle_2) \\
\]

If the initial value of an index is omitted, then it is assumed to be equal to one:

\[
\sum_{j=1}^{3} | j_2 \rangle \cdot (| j + 1 \rangle_2) \\
| 1_2 \rangle \cdot (| 2_2 \rangle + | 3_2 \rangle) + | 2_2 \rangle (| 3_2 \rangle + | 4_2 \rangle) \\
\]

A list of explicit values can be given for the index
It is possible to operate with sums. The piecewise answer to this calculation depends on the value of n:

\[
\sum_{j=1}^{(50,100,200)} |j_2| \cdot ((j+1)_2 | \n
\sum_{j=1}^{50} |5_2| \cdot (51_2 | + | 100_2| \cdot (101_2 | + | 200_2| \cdot (201_2 |

You can use the standard Mathematica notation for assumptions:

\[
\text{Simplify}\left\{(7_q | \cdot \sum_{i=1}^{n} f[i] | i_q) \right\}, n > 10
\]

\[
f[q]
\]

The Assuming command is another way to specify assumptions. Remember that Assuming only works on those Mathematica commands that have the option Assumptions (commands like Simplify, FullSimplify, Integrate, etc, but other commands just ignore the Assuming command):

\[
\text{Assuming}\left\{n > 10, \text{Simplify}\left\{(7_q | \cdot \sum_{i=1}^{n} f[i] | i_q) \right\}\right\]
\]

\[
f[q]
\]

**Known Issue: Operations with Several Nested Sums are Very Slow**

When complex symbolic calculations are involved (complex for a computer is not the same as complex for a human), the calculation can be very slow. Next calculation can take several minutes in your computer. Notice that the first and third sums are actually identity operators, therefore the answer is equivalent to the second sum:

\[
\left(\sum_{i=1}^{d} |i_2| \cdot (i_2 | \right) \cdot \left(\sum_{j=1}^{d} \sum_{k=1}^{d} \text{foo}[j, k] \cdot |j_2| \cdot (k_2 | \right) \cdot \left(\sum_{m=1}^{d} |m_2| \cdot (m_2 | \right)
\]

\[
\sum_{m=1}^{d} \sum_{j=1}^{d} \text{foo}[j, m] \cdot |j_2| \cdot (m_2 |
\]

**Infinite Sums**

Infinite sums can be used:
The dummy index \( m \) in the first sum and the dummy index \( m \) in the second sum are treated as different symbols. Notice the dot [ESC]on[ESC] between the sums, which means that the sums are not being multiplied, there is a "noncommutative quantum application" between them, each sum is consider as an operator:

\[
\sum_{m=-\infty}^{\infty} | (3 \, m) \rangle \cdot \langle k | \cdot \left( \sum_{m=-\infty}^{\infty} | (5 \, m) \rangle \cdot \langle m | \right) \cdot | 10 \rangle
\]

\[
| 150 \rangle
\]

The command DefineOperatorOnKets[ ] can be used instead of \( \sum \) for creating faster operators.

We can use a doubly-infinite sum to define a rising-index operator

\[
\text{diracl} = \sum_{k=-\infty}^{\infty} | (k + 1) \rangle \cdot \langle k | \cdot \left( \sum_{m=-\infty}^{\infty} | (5 \, m) \rangle \cdot \langle m | \right) \cdot | 10 \rangle
\]

Now diracl is an operator

\[
\text{diracl} \cdot | 7 \rangle
\]

\[
| 8 \rangle
\]

Powers of this operator give the expected answer

\[
\text{diracl}^3 \cdot | 7 \rangle
\]

\[
| 10 \rangle
\]

The hermitian conjugate of the operator also works.

\[
(\text{diracl})^\dagger \cdot | 7 \rangle
\]

\[
| 6 \rangle
\]
It is possible to define new operators this other way. Notice that we must define the action of both the operator and its hermitian conjugate:

\[
\text{DefineOperatorOnKets}\{\text{ope2}, \{ | x\_m\rangle \Rightarrow | (x + 1)\_m\rangle \}\}; \\
\text{DefineOperatorOnKets}\{(\text{ope2})\sp\dagger, \{ | x\_m\rangle \Rightarrow | (x - 1)\_m\rangle \}\}; \\
\]

Now ope2 works the same as dirac1

\[
\text{ope2} \cdot | 7\_z\rangle \\
| 8\_z\rangle \\
\]

Powers of this operator also give the expected answer

\[
\text{ope2}^2 \cdot | 7\_z\rangle \\
| 10\_z\rangle \\
\]

The hermitian conjugate of the operator also works. Notice that the action of both the operator and its hermitian conjugate was defined above with two separated DefineOperatorOnKets[]

\[
(\text{ope2})\sp\dagger \cdot | 7\_z\rangle \\
| 6\_z\rangle \\
\]

The operators defined with DefineOperatorOnKets are usually faster than operators defined with sums of Dirac expressions.

by José Luis Gómez-Muñoz
http://homepage.cem.itesm.mx/lgomez/quantum/
jose.luis.gomez@itesm.mx