Comparison of Symbolic Quantum Operators Versus Dirac-Notation Quantum Operators

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

Introduction

This is a tutorial on the use of Quantum *Mathematica* add-on to define the action of new quantum operators on kets. It shows the similarities and differences between a symbolic operator (defined with DefineOperatorOnKets) and a Dirac-Notation operator (created with kets and bras).

Mathematica's Pattern Matching and Replacement Rules

Mathematica can recognize patterns in expressions and manipulate them. First a very simple example: *Mathematica* will find in the list any expression of the form c[y] and replace it with the number 3:

```
ReplaceAll[{c[x], c[y], b[y], b[x]}, Rule[c[y], 3]]
```

```
{c[x], 3, b[y], b[x]}
```

The previous calculation can be written in a more readable notation by using the infix version of the ReplaceAll command, a slash and a dot "/."
On the other hand, the infix version of Rule, which looks like an arrow, can be written by pressing [ESC]->[ESC] (the keys "escape","minus","greater","escape") or by selecting it from *Mathematica*'s palettes:

```
{c[x], c[y], b[y], b[x]} /. c[y] → 3
```

```
{c[x], 3, b[y], b[x]}
```

The following calculation shows that complex replacement rules can be entered by using "underscores" to indicate "Patterns". RuleDelayed must be used instead of Rule with "Patterns".
The infix version of RuleDelayed, which looks like two dots and an arrow, can be written by pressing the keys [ESC]:>[ESC] (the keys "escape","colon","greater","escape") or by selecting it from *Mathematica*'s palettes.
Notice that p has an underscore in the left side of the rule (arrow), but it does Not have it in the right hand side:

```
{c[x], c[y], b[y], b[x]} /. c[p_] ⧴ newc[p + 1]
```

```
{newc[1 + x], newc[1 + y], b[y], b[x]}
```

Several replacement rules can be applied at the same time by enclosing them with curly brackets:

```
{c[x], c[y], b[y], b[x]} /. {c[p_] :→ newc[p + 1], b[y] → foo[y]}
```

```
{newc[1 + x], newc[1 + y], foo[y], b[x]}
```

It is very important to understand the difference between the previous calculation (where b[x] did not change) and the next one (where both b[x] and b[y] are changed)

```
{c[x], c[y], b[y], b[x]} /. {c[p_] :→ newc[p + 1], b[y_] :→ foo[y]}
```

```
{newc[1 + x], newc[1 + y], foo[y], foo[x]}
```

Next the sintaxis of replacement rules will be used to create new Quantum operators to be used in *Mathematica*

Load the Package

First load the Quantum`Notation` package. Write:
Needs["Quantum`Notation`"]
then press at the same time the keys SHIFT-ENTER to evaluate. *Mathematica* will load the package:

```
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:
SetQuantumAliases[ ];
then press at the same time the keys SHIFT-ENTER to evaluate. The semicolon prevents *Mathematica* from printing the help message. Remember that SetQuantumAliases[ ] must be evaluated again in each new notebook:

```
SetQuantumAliases[];
```

The Command DefineOperatorOnKets[]

We will use the command DefineOperatorOnKets.
A basic help can be obtained by writting
? DefineOperatorOnKets
then press at the same time the keys SHIFT-ENTER to evaluate:

```
? DefineOperatorOnKets
```

DefineOperatorOnKets[op,{rules}] defines a new quantum operator. Rules must have the same sintax as the rules used in the Mathematica command ReplaceAll[expr,{rules}] and in the Mathematica sintax expr/.{rules}

Here is the first operator we will define. Its name is ope1, and its action is to transform eigenstates of base operator $\hat{q1}$

Write:

DefineOperatorOnKets[ope1, { [ESC]eket[ESC] [ESC]->[ESC] [ESC]eket[ESC] , [ESC]eket[ESC] [ESC]->[ESC] [ESC]eket[ESC] }]

then press the [TAB] several times in order to select the first place-holder (empty square) and write:

1 [TAB] q1 [TAB] 2 [TAB] q1 [TAB] 2 [TAB] q1 [TAB] 3 [TAB] q1

then press at the same time the keys SHIFT-ENTER to evaluate:

$$\text{DefineOperatorOnKets}\left[\text{ope1},\ \left\{\left|1_{\hat{q1}}\right\rangle \to \left|2_{\hat{q1}}\right\rangle,\ \left|2_{\hat{q1}}\right\rangle \to \left|3_{\hat{q1}}\right\rangle\right\}\right]$$

$$\text{ope1}\cdot\left|1_{\hat{q1}}\right\rangle \to \left|2_{\hat{q1}}\right\rangle$$
$$\text{ope1}\cdot\left|2_{\hat{q1}}\right\rangle \to \left|3_{\hat{q1}}\right\rangle$$

Now we can use our new operator.

Write:

ope1 [ESC]on[ESC] (b [ESC]eket[ESC] + c [ESC]eket[ESC])

then press the [TAB] several times in order to select the first place-holder (empty square) and write:

1 [TAB] q1 [TAB] 2 [TAB] q1

then press at the same time the keys SHIFT-ENTER to evaluate:

$$\text{ope1}\cdot\left(b\left|1_{\hat{q1}}\right\rangle + c\left|2_{\hat{q1}}\right\rangle\right)$$

$$b\left|2_{\hat{q1}}\right\rangle + c\left|3_{\hat{q1}}\right\rangle$$

In this simple case, we can use Dirac notation to write an operator that does something similar:

$$\text{dirac1} = \left|2_{\hat{q1}}\right\rangle\cdot\left\langle 1_{\hat{q1}}\right| + \left|3_{\hat{q1}}\right\rangle\cdot\left\langle 2_{\hat{q1}}\right|$$

$$\left|2_{\hat{q1}}\right\rangle\cdot\left\langle 1_{\hat{q1}}\right| + \left|3_{\hat{q1}}\right\rangle\cdot\left\langle 2_{\hat{q1}}\right|$$

Now in this simple calculation the operators ope1, which was defined with the command DefineOperatorOnKets, and dirac1, which was created directly from the Dirac notation, give the same result:

$$\text{dirac1}\cdot\left(b\left|1_{\hat{q1}}\right\rangle + c\left|2_{\hat{q1}}\right\rangle\right)$$

$$b\left|2_{\hat{q1}}\right\rangle + c\left|3_{\hat{q1}}\right\rangle$$

The *Mathematica* command Column gives a simple way to show both results at the same time:

```
Column[{
  ope1 · (b |1_q̂1⟩ + c |2_q̂1⟩),
  dirac1 · (b |1_q̂1⟩ + c |2_q̂1⟩)
 }]
```

```
b |2_q̂1⟩ + c |3_q̂1⟩
b |2_q̂1⟩ + c |3_q̂1⟩
```

The hermitian conjugate (entered as [ESC]her[ESC]) of both operators applied to a bra gives the expected answer (Notice that it is necessary to "Expand" for the hermitian of dirac1 on a Bra):

```
Column[{
  ⟨2_q̂1| · (ope1)†,
  ⟨2_q̂1| · (dirac1)†,
  Expand[⟨2_q̂1| · (dirac1)†]
 }]
```

```
⟨3_q̂1|
⟨2_q̂1| · ( |1_q̂1⟩ · ⟨2_q̂1| + |2_q̂1⟩ · ⟨3_q̂1| )
⟨3_q̂1|
```

On the other hand, there are situations where these two operators, created in different ways, are Not equivalent.
As you can see in the following example, when ope1 acts on a ket where its action was not defined, it remains unevaluated. On the other, dirac1 vanishes that ket:

```
Column[{
  ope1 · (b |1_q̂1⟩ + c |2_q̂1⟩ + d |3_q̂1⟩),
  dirac1 · (b |1_q̂1⟩ + c |2_q̂1⟩ + d |3_q̂1⟩)
 }]
```

```
d ope1 · |3_q̂1⟩ + b |2_q̂1⟩ + c |3_q̂1⟩
b |2_q̂1⟩ + c |3_q̂1⟩
```

Now the important advantage of the command DefineOperatorOnKets over the direct Dirac definition of an operator is the use of the powerful "Pattern" matching capabilities of *Mathematica*. Here we use the "Pattern" (the n with underscore, n_) and the "RuleDelayed" [ESC]:>[ESC] (the keys "escape","colon","greater","escape") in order to create a powerful "rising" operator, called ope2.
Notice that n has an underscore in the left side of the rule (arrow), but it does Not have it in the right hand side. Notice also the use of parenthesis in (n+1):

```
DefineOperatorOnKets[ope2, { | n_q̂1 〉 :→ | (n + 1) q̂1 〉}]
```

```
| n_q̂1 〉 :→ | (n + 1) q̂1 〉
```

Now we can compare the three rising operators:

```
Column[{
  ope1 · (b | 1 q̂1 〉 + c | 2 q̂1 〉 + d | 3 q̂1 〉),
  dirac1 · (b | 1 q̂1 〉 + c | 2 q̂1 〉 + d | 3 q̂1 〉),
  ope2 · (b | 1 q̂1 〉 + c | 2 q̂1 〉 + d | 3 q̂1 〉)
 }]
```

```
d ope1 · | 3 q̂1 〉 + b | 2 q̂1 〉 + c | 3 q̂1 〉
b | 2 q̂1 〉 + c | 3 q̂1 〉
b | 2 q̂1 〉 + c | 3 q̂1 〉 + d | 4 q̂1 〉
```

It can be created an operator using Dirac notation in *Mathematica* that works the same as ope2:

$$\text{dirac2} = \sum_{j=-\infty}^{\infty} \left| (j+1)_{\hat{q}1} \right\rangle \cdot \left\langle j_{\hat{q}1} \right|$$

$$\sum_{j=-\infty}^{\infty} \left| (j+1)_{\hat{q}1} \right\rangle \cdot \left\langle j_{\hat{q}1} \right|$$

Here the two operators do the same

```
Column[{
  ope2 · (b | 100 q̂1 〉 + c | 200 q̂1 〉 + d | 1020 q̂1 〉),
  dirac2 · (b | 100 q̂1 〉 + c | 200 q̂1 〉 + d | 1020 q̂1 〉)
 }, Dividers → All]
```

b | 101 q̂1 〉 + c | 201 q̂1 〉 + d | 1021 q̂1 〉
b | 101 q̂1 〉 + c | 201 q̂1 〉 + d | 1021 q̂1 〉

Powers of operators can easily be calculated

```
Column[{
  ope2⁴ · (b |100_q̂1⟩ + c |200_q̂1⟩ + d |1020_q̂1⟩),
  dirac2⁴ · (b |100_q̂1⟩ + c |200_q̂1⟩ + d |1020_q̂1⟩)
 }, Dividers → All]
```

b $\vert 104_{\hat{q1}}\rangle$ + c $\vert 204_{\hat{q1}}\rangle$ + d $\vert 1024_{\hat{q1}}\rangle$
b $\vert 104_{\hat{q1}}\rangle$ + c $\vert 204_{\hat{q1}}\rangle$ + d $\vert 1024_{\hat{q1}}\rangle$

Differences between an operator created with DefineOperatorOnKets[] and an operator created with kets and bras

These are the definitions of the two operators we are working with. The operator dirac2 is created from kets and bras, while the symbolic operator ope2 is created with the Quantum *Mathematica* command DefineOperatorOnKets[]:

```
Needs["Quantum`Notation`"];
dirac2 = Σ_{j=-∞}^{∞} |(j + 1)_q̂1⟩ · ⟨j_q̂1| ;
DefineOperatorOnKets[ope2, { |n__q̂1⟩ ⧴ |(n + 1)_q̂1⟩}];
```

Both have the same effect on a superposition of kets:

```
Column[{
  ope2 · (b |100_q̂1⟩ + c |200_q̂1⟩ + d |1020_q̂1⟩),
  dirac2 · (b |100_q̂1⟩ + c |200_q̂1⟩ + d |1020_q̂1⟩)
 }, Dividers → All]
```

b $\vert 101_{\hat{q1}}\rangle$ + c $\vert 201_{\hat{q1}}\rangle$ + d $\vert 1021_{\hat{q1}}\rangle$
b $\vert 101_{\hat{q1}}\rangle$ + c $\vert 201_{\hat{q1}}\rangle$ + d $\vert 1021_{\hat{q1}}\rangle$

However the **hermitian** conjugate of ope2 has no action **on kets**, while the hermitian conjugate of dirac2 has a very specific effect:

```
Column[{
  (ope2)† · (b |100_q̂1⟩ + c |200_q̂1⟩ + d |1020_q̂1⟩),
  (dirac2)† · (b |100_q̂1⟩ + c |200_q̂1⟩ + d |1020_q̂1⟩)
 }, Dividers → All]
```

ope2† · (b $\vert 100_{\hat{q1}}\rangle$ + c $\vert 200_{\hat{q1}}\rangle$ + d $\vert 1020_{\hat{q1}}\rangle$)
b $\vert 99_{\hat{q1}}\rangle$ + c $\vert 199_{\hat{q1}}\rangle$ + d $\vert 1019_{\hat{q1}}\rangle$

We can force the hermitian conjugate of ope2 to have the same effect as the hermitian conjugate of dirac2. Notice the use of the parenthesis in (n-1)

```
DefineOperatorOnKets[(ope2)†, {|n_q̂1⟩ :> |(n - 1)q̂1⟩}];
Column[{
   (ope2)† · (b |100q̂1⟩ + c |200q̂1⟩ + d |1020q̂1⟩),
   (dirac2)† · (b |100q̂1⟩ + c |200q̂1⟩ + d |1020q̂1⟩)
 }, Dividers → All]
```

$b\ \lvert 99_{\hat{q}1}\rangle + c\ \lvert 199_{\hat{q}1}\rangle + d\ \lvert 1019_{\hat{q}1}\rangle$
$b\ \lvert 99_{\hat{q}1}\rangle + c\ \lvert 199_{\hat{q}1}\rangle + d\ \lvert 1019_{\hat{q}1}\rangle$

Perhaps the main **difference** between ope2 (created with DefineOperatorOnKets[]) and dirac2 (created with kets and bras) is what happen when these operators are **not** acting on kets or bras:

```
Column[{
   ope2,
   dirac2
 }, Dividers → All]
```

ope2
$\sum_{j=-\infty}^{\infty} \lvert (j+1)_{\hat{q}1}\rangle \cdot \langle j_{\hat{q}1} \rvert$

Powers of the two operators look different when they are not acting on a ket

```
Column[{
   ope2^4,
   dirac2^4
 }, Dividers → All]
```

ope2^4
$\left(\sum_{j=-\infty}^{\infty} \lvert (j+1)_{\hat{q}1}\rangle \cdot \langle j_{\hat{q}1} \rvert\right)^4$

Results of algebraic expansions look different (although equivalent, they will have the same effect on a ket)

```
Column[{
   Expand[(ope2 + 1)^2],
   Expand[(dirac2 + 1)^2]
 }, Dividers → All]
```

$1 + 2\ \text{ope2} + \text{ope2}^2$
$1 + 2 \sum_{j=-\infty}^{\infty} \lvert (1+j)_{\hat{q}1}\rangle \cdot \langle j_{\hat{q}1} \rvert + \sum_{j1=-\infty}^{\infty} \lvert (2+j1)_{\hat{q}1}\rangle \cdot \langle j1_{\hat{q}1} \rvert$

Results of expansions using commutators look different

```
SetQuantumObject[qo];
Column[{
  CommutatorExpand[(ope2 + qo)^2],
  CommutatorExpand[(dirac2 + qo)^2]
 }, Dividers → All]
```

$\text{ope2}^2 + \text{qo}^2 - [\![\text{ope2}, \text{qo}]\!]_- + 2\,\text{ope2} \cdot \text{qo}$
$\text{qo}^2 + \sum_{j3=-\infty}^{\infty} \left\vert (2+j3)_{\hat{q1}} \right\rangle \cdot \left\langle j3_{\hat{q1}} \right\vert + \sum_{j=-\infty}^{\infty} \left\vert (1+j)_{\hat{q1}} \right\rangle \cdot \left\langle j_{\hat{q1}} \right\vert \cdot \text{qo} + \sum_{j=-\infty}^{\infty} \text{qo} \cdot \left\vert (1+j)_{\hat{q1}} \right\rangle \cdot \left\langle j_{\hat{q1}} \right\vert$

The two expressions above are equivalent, as can be seen when they are applied to the same ket:

```
SetQuantumObject[qo];
Column[{
  CommutatorExpand[(ope2 + qo)^2] · |100_q̂1〉,
  CommutatorExpand[(dirac2 + qo)^2] · |100_q̂1〉
 }, Dividers → All]
```

$\text{qo}^2 \cdot \left\vert 100_{\hat{q1}} \right\rangle + \text{qo} \cdot \left\vert 101_{\hat{q1}} \right\rangle + \text{ope2} \cdot \text{qo} \cdot \left\vert 100_{\hat{q1}} \right\rangle + \left\vert 102_{\hat{q1}} \right\rangle$
$\sum_{j=-\infty}^{\infty} \left\vert (1+j)_{\hat{q1}} \right\rangle \cdot \left\langle j_{\hat{q1}} \right\vert \cdot \text{qo} \cdot \left\vert 100_{\hat{q1}} \right\rangle + \text{qo}^2 \cdot \left\vert 100_{\hat{q1}} \right\rangle + \text{qo} \cdot \left\vert 101_{\hat{q1}} \right\rangle + \left\vert 102_{\hat{q1}} \right\rangle$

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