DiracQ Tutorial

The purpose of this notebook is to familiarize the reader with the operation of the DiracQ package. This tutorial covers both the basic and more advanced uses of the package. The use of every function in the package is not covered in this notebook. See the "Glossary" notebook for an explanation of every function. For more in-depth examples demonstrating the types of manipulations DiracQ can perform, see the "Examplebook" notebook.

I: Loading the Package

There are a number of methods to load this or any other Mathematica package. Instructions can be obtained through the Wolfram Documentation Center. The command below will load the package so long as the package .m file is saved in the same directory as is saved the current notebook.

In[1]:=
```
SetDirectory[NotebookDirectory[]];
Get["DiracQ_V0.m"];
```

II: List of Functions and Symbols Provided by the Package

The Functions in the package are in the context "DiracQ". The command below calls a list of all functions provided to the user by the package and the symbols used by the package for both input and output. The notebook "Glossary_VX.nb" and its pdf version "Glossary_VX.pdf" contain the definitions, usage and simple examples of the functions and symbols listed in the next output. Brief usage paragraphs can be called for any of the functions below by clicking the function, assuming that you have already invoked DiracQ as shown above.

```
?DiracQ`*
```

▼ DiracQ`

AddOperator	FullOrganize	q
AllSymbols	function	QCoefficient
anticommutator	f†	SecondaryOperators
AntiCommutator	Humanize	SimplifyQ
AntiCommutatorDefinition	Identical	StandardOrderQ
ApplyDefinition	J	StandardReordering
b	n	TakeCPart
b†	NCcross	TakeQPart
commutator	OperatorProduct	TakeSummand
Commutator	Operators	Vacuum
CommutatorDefinition	Organize	x
CommutatorRule	OrganizedExpression	X
CommuteParts	OrganizedProduct	y
Decomposition	OrganizeQ	z
DeleteOperator	p	δ
DiracQPalette	PositionQ	ϵ
DropQ	ProductQ	σ
Evaluation	PushOperatorLeft	†
f	PushOperatorRight	$\hbar$

III: Basic Use and Notation

To use DiracQ at the most basic level is quite simple. When the package is loaded, the DiracQ palette should load automatically. The palette utilizes dynamic features. If the user receives a warning regarding dynamic features they should choose to activate dynamic content. The palette contains a list of the operators that are included with the package. In the section titled “Active Operators” there are two windows and a list of operators with check boxes next to them. Check the box of any operator(s) you want to use. Now that operator is active, meaning that the package will view this symbol as having the properties of the quantum mechanical operator to which it corresponds. The symbols for innactive operators will not be viewed as having any special properties.
Lets try a simple example: calculating the commutator or two spin operators.

- **Activate the Pauli matrix symbol "σ" by clicking the box to the right of the σ symbol within the Operator Controls section**

```
Commutator[σ[i, x], σ[j, y]]
```

```
2 i δ[i, j] ** σ[i, z]
```

Here we reproduce a familiar result, though at first glance it may appear somewhat cryptic. What we have shown is more familiarly written

$$\left[\sigma_{i,x},\ \sigma_{j,y}\right] = 2\, i\, \delta_{i,j}\, \sigma_{i,z}$$

where i and j represent site index. However, we cannot write it exactly like this and have Mathematica understand. Therefore the notation used throughout the package is as follows :

- **Indices of operators are written as arguments** (i.e. in brackets following the operators symbol)

$\sigma_{i,x} \implies \sigma$[i,x] $\qquad\qquad$ $\delta_{i,j} \implies \delta$[i,j]

• **Every operator needs at least one argument to specify site index**. If specifying a site is not relevant for the problem you are doing then simply specify all operators with the same arbitrary site index. Some operators, such as Pauli Spin Matrices, need additional arguments, in this case to specify direction. See the more detailed descriptions in this tutorial for more information on specific operators.

$\sigma_x + \sigma_y \Longrightarrow \sigma$[i,x]+$\sigma$[i,y]

• **NonCommutativeMultiply** (written as **) is used in place of Times (*) between objects to preserve the correct ordering. NonCommutativeMultiply must be used by the user when input involves the product of commutators (q numbers).

$\sigma_x \sigma_y \Longrightarrow \sigma$[i,x]**$\sigma$[i,y]

Now we see that our result above is actually familiar. In this example we used the Commutator function. This is one of the many function of the DiracQ package. Another commonly useful function is ProductQ, which allows the user to calculate the product of two exressions. This function, as do all other DiracQ functions, utilizes the properties of quantum mechanical operators. Here, we reproduce the familiar product of two Pauli spin matrices.

```
ProductQ[σ[i, x], σ[i, x]]
```

```
1
```

The above example can also be written using the “circle times” symbol to represent the function ProductQ. The input below is identical to that above.

```
σ[i, x] ⊗ σ[i, x]
```

```
1
```

The example below demonstrates more complex input being understood and evaluated by the Commutator function. All of the functions in the package can understand input of this complexity.

• **Activate the Pauli matrix symbol "σ" and the Fermi operator symbols f and f†**

```
A = ∑_i t[i] σ[i, y] ** σ[i, x] ** f[i];
B = σ[k, y] ** f†[k];
Commutator[A, B]
```

$$-2 \sum_i (t[i]\, \sigma[i, x]\, \sigma[i, y]\, \sigma[k, y]) ** f\dagger[k] ** f[i] + t[k]\, \sigma[k, x]\, \sigma[k, y]^2$$

IV: The DiracQ Palette

Essential to the operation of the package is the DiracQPalette. This palette allows users to control the options of package functions, designate symbols as operators, and input some of the special symbols and functions used frequently when operatoring the package. To open the palette, input the function DiracQPalette. The palette should also open automatically when the package is first loaded. The palette utilizes dynamic features. If the user receives a warning regarding dynamic features they should choose to activate dynamic content.

The DiracQ package contains a collection of commonly used quantum mechanical operators for which algebraic relations such as commutators, anticommutators, and products are known. The functions of the package recognize certain symbols as operators and use these algebraic relations to manipulate and evaluate user input. To avoid confusion the package requires explicit instructions from the user to view any certain symbol as an operator. By default no symbols are viewed as operators. This prevents users from making assumptions about the status of a symbol.

The DiracQ palette contains a section titled "Operator Controls". The purpose of this section is to control

how the functions of the package view a symbol. This section contains a list of the operators that are included for use in the package and the symbols used to represent them. For a symbol on this list to be viewed as an operator it must first be activated by the user. This is done by locating the desired symbol(s) in the subsection titled "Active Operators" and checking the box to the right of the symbol(s). The symbol should now populate the list of primary operators. Any additional secondary operators which are composites of the primary operators will be found in the list "Secondary Operators". Once a symbol is activated that symbol will be viewed as an operator, or q number, by all of the DiracQ functions. The symbol will also be given the special algebraic properties of the quantum mechanical operator to which it corresponds. The same symbol when unactivated will be viewed as a c number with no special algebraic properties. The example below demonstrates this point. This example utilizes the DiracQ function Commutator, which will evaluate the commutator of two arguments, as explained by the usage paragraph for commutator.

```
? Commutator
```

Commutator is used to calculate the commutators of expressions
involving operators with known commutation relations. Commutator[A,B] is defined
as AB−BA. Output of these functions is ordered according to standard order.

- **Activate the Pauli matrix symbol "σ" by clicking the box to the right of the σ symbol within the Operator Controls section**

```
Commutator[σ[i, x], σ[j, y]]
```

```
2 i δ[i, j] ** σ[i, z]
```

- **Unactivate the Pauli matrix symbol "σ" by either unchecking the corresponding box of clicking the "Return Default Palette Settings" button**

```
Commutator[σ[i, x], σ[j, y]]
```

```
0
```

It will be assumed from this point forward in this tutorial that the pallet is set to default settings at the beginning of an example. Therefore, no instructions will be given to return the palette to default settings at the end of an example.
To return default settings click the “Return Default Palette Settings Button” near the bottom of the palette.

The above example shows that when σ is active it is imbued with the properties of the Pauli matrices. When σ is inactive it behaves like a c number. All symbols that are used to represent operators have been exported by the package. Information about the symbol can be obtained by calling the usage as shown below.

```
? σ
```

σ is the Pauli spin matrix. This operator requires two arguments. The first is site
index and the second is coordinate direction. An optional third argument is used to denote
different spin species. Also included are the Pauli raising and lowering operators, denoted by
σ^{Plus} and σ^{Minus} respectively. The raising and lowering operators require only one argument
corresponding to site. A second argument (optional) will be taken to represent spin species.

Notice that all symbols used to represent operators have been exported by the DiracQ package and therefore appear in black (the color used by Mathematica to represent a symbol which has a definition). This does not mean that these symbols cannot be given definition and used as would any other symbol. The example below shows that we can even give σ a numerical value provided the symbol is not activated on the palette.

```
σ = 2
```

```
2
```

```
σ[i, x]
```

```
2[i, x]
```

However if σ is activated in the palette, uncontrollable errors will arise by trying to give it an additional numerical definition. Therefore, ensure that a symbol has not been defined before it is used to represent an operator. A convenient way to clean the slate is to use the "Clear" function.

```
Clear[σ]
```

- **Activate the Pauli matrix symbol "σ" and its standard properties are restored.**

```
Commutator[σ[i, x], σ[j, y]]
```

```
2 i δ[i, j] ** σ[i, z]
```

By activating the All Symbols button every non-numerical symbol will be viewed as an operator. Symbols other than those used to represent operators will not have algebraic relationships such as commutator and anticommutator defined. Therefore the definitions of the commutator and the anticommutator are used.

- **Activate "All Symbols"**

```
Clear[A, B];
Commutator[A, B]
```

```
A ** B - B ** A
```

The section of the palette titled "Function Options" (near the top) allows the user to set options that apply to several functions. Apply Definition is an option which allows users to specify whether or not predefined algebraic relationships between operators should be used while manipulating input. For example, the canonical commutator of position and momentum operators is known to be equal to $i\hbar$. By default Apply Definition is True, so if this commutator is encountered anywhere within an expression it will be replaced by $i\hbar$. If however, Appy Definition was False, the commutator would remain unevaluated. This is demostrated below.

- **Activate the canonical position and momentum operators matrix symbol "q and p"**

```
Commutator[q[i], p[i]]
```

```
i ℏ
```

- **Set Apply Definition to False**

```
Commutator[q[i], p[i]]
```

```
commutator[q[i], p[i]]
```

Decomposition is an option which determines how commutators of more than two operators will be decomposed. Commutators of more than two operators can be decomposed into an expression involving commutators of only two operators or anticommutators of only two operators, as shown below.

[A,BC]=[A,B] C-B[C,A]
[A,BC]={A,B} C-B{C,A}

By setting Decomposition to Commutator the former definition will be used to decompose commutators of more than two terms. By setting decomposition to AntiCommutator the latter definition will be used. It is worth noting that a similar rule does not also hold for anticommutators. Anticommutators of more than two operators are decomposed into an expression involving both commutators and anticommutators. Therefore this setting will not effect the operators of the anticommutator function. Also, as a result of the decomposition of anticommutators it is not always possibly to decompose large expressions entirely into anticommutators. The ability to do so is dependent on the number of operators contained in each expression. The example below demonstrates the use of Decomposition.

- **Activate "All Symbols"**

- **Set Apply Definition to False**
- **Set Decomposition to Commutator**

```
Commutator[A, B ** C]
```

```
-B ** commutator[C, A] + commutator[A, B] ** C
```

- **Set Decomposition to AntiCommutator**

```
Commutator[A, B ** C]
```

```
-B ** anticommutator[C, A] + anticommutator[A, B] ** C
```

The remaining section of the palette contains typesetting icons similar to the typesetting sections of other Mathematica palettes. This section is included only for convenience and is not necessary to the operation of the DiracQ package.

There are a few technical points worth mentioning about using the palette. Once the palette is opened a second palette can be called. The options settings are mirrored between palettes but the operator settings are not. The settings used will be those that have most recently been updated. To avoid confusion keep only one copy of the palette open at a time.

When the palette is closed a window will open that enables the user to save the DiracQ palette notebook. This is not necessary and should not be done. Doing so will not save palette setttings for the next use but will rather save the palette as a notebook in a location to be decided by the user.

V: Operators Included in the Package

There are some general properties shared by all operators. Most importantly, an operator must be called as a function with an argument. If only the head of the operator is used, it will not be recognized. The first argument is always taken to be the site at which the operator acts. Only one index or list of index may be used to specify site index.

Below are explanation of the usage and properties for each individual type of operator.

Bosonic Operators

```
? b
```

b is the bosonic annihilation operator. This operator requires one index denoting site, and a second optional index can be used to denote spin. Also included is the bosonic number operator, represented by n_b. The argument scheme for the number operator is identical to that of the annihilation operator.

```
? b†
```

b† is the bosonic creation operator. This operator requires one index denoting site, and a second optional index can be used to denote spin. Also included is the bosonic number operator, represented by n_b. The argument scheme for the number operator is identical to that of the creation operator.

Fermionic Operators

```
? f
```

f is the fermionic annihilation operator. This operator requires one index denoting site, and a second optional index can be used to denote spin. Also included is the fermionic number operator, represented by n_f. The argument scheme for the number operator is identical to that of the annihilation operator.

```
? f†
```

f† is the fermionic creation operator. This operator requires one index denoting site, and a second optional index can be used to denote spin. Also included is the fermionic number operator, represented by n_f. The argument scheme for the number operator is identical to that of the creation operator.

■ Number Operators

```
? n
```

n is the number operator. To specify the number operator for bosons use n_b and for fermions use n_f.

An example demonstrating the use of the number operator is given below.

```
nb[i]
```

```
b†[i] ** b[i]
```

```
b†[i] ** b[i]
```

```
b†[i] ** b[i]
```

```
nf[i, up]
```

```
f†[i, up] ** f[i, up]
```

The basic (anti)commutators are known to DiracQ, e.g.

• Activate Fermionic Creation and Annihilation Operators "f and f†"

```
Commutator[f[i, up], nf[i, up]]
```

```
f[i, up]
```

Here the symbol "up" could be replaced by any other convenient one, say "1".

■ Canonical Angular Momentum Operators

```
? J
```

J is the canonical angular momentum operator. This operator requires two arguments. The first is site index and the second is coordinate direction. An optional third argument is used to denote different species. Also included are the angular momentum raising and lowering operators, denoted by J^{Plus} and J^{Minus} respectively. The raising and lowering operators accept only one argument corresponding to site. A second (optional) argument will be taken to represent species.

Use of symbols other than x, y or z to represent coordinate direction will result in incomplete or incorrect output.

■ Pauli Matrices

```
? σ
```

σ is the Pauli spin matrix. This operator requires two arguments. The first is site index and the second is coordinate direction. An optional third argument is used to denote different spin species. Also included are the Pauli raising and lowering operators, denoted by σ^{Plus} and σ^{Minus} respectively. The raising and lowering operators require only one argument corresponding to site. A second argument (optional) will be taken to represent spin species.

Use of symbols other than x, y or z to represent coordinate direction will result in incomplete or incorrect output.

Hubbard "X" operators

```
? X
```

X is the Hubbard Operator. Three arguments are required. The first argument represents site. The second argument is taken to be the direction of the 'Ket' spin and the third argument is taken to be the direction of the 'Bra spin

The Hubbard X operator is defined below

$$\mathbf{X}_{\mathbf{i}}^{\sigma_1 \sigma_2} = |\sigma_1\rangle \langle \sigma_2|$$

An example of the use of the Hubbard "X" operators is provided below.

Activate Hubbard "X" Operators

```
Commutator[X[i, 1, 2], X[j, 2, 3]]
```

```
2 X[i, 1, 2] ** X[j, 2, 3] + δ[i, j] ** X[i, 1, 3] - 2 δ[i, j] ** X[i, 1, 2] ** X[j, 2, 3]
```

Bra and Ket Vectors

```
? Bra
```

Bra[x] represents a bra vector x using Dirac notation

```
? Ket
```

Ket[x] represents a ket vector x using Dirac notation

The Bra and Ket vectors accept one argument : the name of the vector. Using "Vacuum" as the argument leads to the vector having the special properties of the vacuum state.

```
? Vacuum
```

Vacuum is the symbol used to represent the vacuum state. In general different operators are taken to act on different basis and therefore Vacuum represents the direct product of the vacuum state of several different basis

The example below demonstrates the use of the Bra and Ket vectors and the Vacuum state.

Activate Fermionic Creation and Annihilation Operators "f and f†" and Bra and Ket Vectors

```
f†[i] ⊗ Ket[Vacuum]
```

```
f†[i] ** |Vacuum⟩
```

```
f[i] ⊗ Ket[Vacuum]
```

```
0
```

Canonical Position and Momentum Operators

```
? p
```

p is the canonical momentum operator. This operator can be called with one argument, taken to be site index, or two arguments. The second argument will be taken to be coordinate direction. Also included is the 3 dimensional canonical momentum vector, represented by OverVector[p], or $\vec{p}$.

```
? q
```

q is the canonical position operator. This operator can be called with one argument, taken to be site index, or two arguments. The second argument will be taken to be coordinate direction. Also included is the 3 dimensional canonical position vector, represented by OverVector[q], or $\vec{q}$.

VI: Simplifyfing and Manipulating Expressions

Many of the functions of the DiracQ package function assist users in simplifying or manipulating expressions. These functions are outlined below. These functions allow the user to perform many of the manipulations normally performed by hand.

- **SimplifyQ**

The most basic function for simplifying expressions is the SimplifyQ function. This function' s use is similar to the Mathematica function Simplify, and should be used in place of Simplify when simplifying expressions while using DiracQ. Results of DiracQ functions are most often already simplified to the greatest degree possible.

Example :

The following shows a use of SimplifyQ to simplify an expression.

```
a[i_] := Sqrt[m ω / (2 ħ)] (q[i] + i p[i] / (m ω));
```

```
a†[i_] := Sqrt[m ω / (2 ħ)] (q[i] - i p[i] / (m ω));
```

```
n[i_] := a†[i] ** a[i]
```

```
SimplifyQ[n[i]]
```

$$-\frac{1}{2}\,i\,\frac{1}{\hbar}\;**\;p[i]\;**\;q[i]+\frac{1}{2}\,i\,\frac{1}{\hbar}\;**\;q[i]\;**\;p[i]+\frac{1}{2}\,\frac{1}{m\,\omega\,\hbar}\;**\;p[i]\;**\;p[i]+\frac{1}{2}\,\frac{m\,\omega}{\hbar}\;**\;q[i]\;**\;q[i]$$

- **StandardOrderQ**

Another function which can be utilized for simplification is StandardOrderQ. StandardOrderQ is similar to SimplifyQ but will reorder operators according to a standard order. The operators will be sorted according to operator type as well as arguments of the operators such as site index. When operators are reordered the commutation relations of the operators are accounted for. This is sometimes helpful, but can also make an expression more complicated, as reordering operators can create new terms that arise out of commutators.

Example :

Using the same example as used for SimplifyQ, we can see the benefit of reordering expressions so that the operators are in a standard order.

```
StandardOrderQ[n[i]]
```

$$-\frac{1}{2}+\frac{1}{2}\,\frac{1}{m\,\omega\,\hbar}\;**\;p[i]\;**\;p[i]+\frac{1}{2}\,\frac{m\,\omega}{\hbar}\;**\;q[i]\;**\;q[i]$$

- **CommuteParts**

CommuteParts is used to "manually" reorganize the order if operators in an expression.

```
? CommuteParts
```

CommuteParts[A,B,C] will reverse the order of the noncommuting objects specified by the lists B and C. B and C are lists of consecutive noncommutative objects found in the expression A specified by numerical ordering of the noncommutative objects found in A. Therefore, to permute the second and third NCOs found in A with the fourth NCO found in A, B would be {2,3}, and C would be {4}.

- **Examples :**

```
A = σ[1, x] ** σ[2, x] ** σ[1, z] ** σ[2, z];
```

In this example we will move $\sigma[1,z]$ to the left of the first two terms:

```
CommuteParts[A, {1, 2}, {3}]
```

```
-2 i σ[1, y] ** σ[2, x] ** σ[2, z] + σ[1, z] ** σ[1, x] ** σ[2, x] ** σ[2, z]
```

Here we move $\sigma[2,z]$ to the first slot, so we need to slide it past the first three elements:

```
CommuteParts[A, {1, 3}, {4}]
```

```
-2 i σ[1, x] ** σ[1, z] ** σ[2, y] + σ[2, z] ** σ[1, x] ** σ[2, x] ** σ[1, z]
```

If the commutator of the two terms that the user requests to permute is zero, the output will simply be the input with the two requested terms permuted. However, the result is automatically resorted, so no change is observed.

```
CommuteParts[A, {3}, {4}]
```

```
σ[1, x] ** σ[2, x] ** σ[2, z] ** σ[1, z]
```

■ PushOperatorLeft and PushOperatorRight

Sometimes operators need to be reordered "manually", but specifying individual operators to move is time consuming or difficult for large expressions.
PushOperatorLeft and PushOperatorRight allow the user to move one operator all the way to the right or left in the ordering

Example:

- **In the DiracQ Palette activate Canonical Position and Momentum Operators (p and q).**

In the example below, we want to move the operator p[i, z] all the way to the left in every term.

```
PushOperatorLeft[
 -i (e^2 Z ħ) ** q[i, y] ** q[i, y] ** p[i, z] - i (e^2 Z ħ) ** q[i, z] ** q[i, z] ** p[i, z], p[i, z]]
```

```
2 (e^2 Z ħ^2) ** q[i, z] - i (e^2 Z ħ) ** p[i, z] ** q[i, y] ** q[i, y] -
 i (e^2 Z ħ) ** p[i, z] ** q[i, z] ** q[i, z]
```

The operator p[i,z] is now pushed all the way to the left. The operation of PushOperatorRight is identical.

VII: Sums

■ Basic Use

The following is an explanation of how the Sum function is used in *Mathematica*. If you are familiar with using this function, feel free to skip to the section titled “Use of Sums in the Package”.

Sums can be written in a few ways. One is to explicitly write the Sum function, the other to use sigma sum notation. The explanation of the Sum function given here is by no means complete; for a thorough explanation of this function see the Mathematica Documentation Center. The first argument of Sum is the expression to be summed over. Each successive argument given is taken to be an index of summation. The examples below show the correct way to use Sum and some common errors.

- **Ex. (1): Basic Input**

The indefinite sum over some expression can be written as follows.

```
Sum[a[i] + b[i], i]
```

$$\sum_{i} (a[i] + b[i])$$

Note that if the expression given does not depend on the index of summation Mathematica will automatically evaluate the sum.

```
Sum[a + b, i]
```

$$(a + b)\, i$$

For definite sums, the minimum and maximum value of the sum are specified with the summation index in a list.

```
Sum[a[i] + b[i], {i, 1, ∞}]
```

$$\sum_{i=1}^{\infty} (a[i] + b[i])$$

Several indices of summation can be specified in the function

```
Clear[a, b, i, j];
Sum[a[i, j] + b[i, j], {i, 1, ∞}, {j, 1, ∞}]
```

$$\sum_{i=1}^{\infty} \sum_{j=1}^{\infty} (a[i, j] + b[i, j])$$

- **Ex. (2) : Sigma Notation**

To use sigma notation first call a sigma sum function by typing the following : ESC sumt ESC.
The following should appear

$$\sum_{\square=\square}^{\square} \square$$

The summation sigma can also be called from the DiracQ palette typesetting section or any of the other *Mathematica* palettes. Now the blank spaces can be filled as desired. If a space is not to be used simply highlight it and delete it. Below is the equivalent of the input from Ex. (1) written in sigma notation.

$$\sum_{i} (a[i] + b[i])$$

$$\sum_{i} (a[i] + b[i])$$

We can see the equivalence between the two forms as follows.

```
FullForm[∑_i (a[i] + b[i])]
```

```
Sum[Plus[a[i], b[i]], i]
```

It is not possible to sum over several indices using one sigma function, as is conventionally done by hand. A separate sigma must be called for each index of summation.

Incorrect:

$$\sum_{i,j} a[i, j]$$

Correct:

$$\sum_{i}\sum_{j} a[i, j]$$

$$\sum_{i}\sum_{j} a[i, j]$$

■ Use of Sums in the Package: Definite and Indefinite Sums

Both definite and indefinite sums are recognized by the package. When indefinite sums are used every relevant value is taken to lie within the bounds of the sum. This means that if an index with an undefined value is encountered along with a summed index in a delta function, the delta function will always evaluate to one as it is assumed that the undefined index, whatever it' s value, will lie within the bounds of the sum. This is demonstrated below. (Remember to select the fermi operators as active on the palette).

$$\text{AntiCommutator}\left[\sum_{i} f[i], f\dagger[j]\right]$$

1

Although the value of j and the bounds of the sum are both not specified, it is assumed the value of j lies somewhere within the bounds of the sum and the answer evaluates to unity in the above example. Definite sums come in two types : with bounds that are defined and with undefined bounds. For example, the bounds of the sum below are specified and have definite values.

$$\sum_{i=1}^{12} f[i]$$

$$f[1] + f[2] + f[3] + f[4] + f[5] + f[6] + f[7] + f[8] + f[9] + f[10] + f[11] + f[12]$$

The sum below, however, is definite with undefined bounds.

$$\sum_{i=a}^{b} f[i]$$

$$\sum_{i=a}^{b} f[i]$$

These two types of definite sums are treated differently by the package and by Mathematica as a whole. Sums with defined bounds are evaluated immediately. Because the sum is evaluated immediately there is no difficulty evaluating expression with defined sums, as shown below.

$$\text{AntiCommutator}\left[\sum_{i=1}^{12} f[i], f\dagger[5]\right]$$

1

$$\text{AntiCommutator}\left[\sum_{i=1}^{12} f[i], f\dagger[13]\right]$$

0

Sums with undefined bounds remain unevaluated. When the package recognizes that a sum has been included with bounds that are unspecified, it does not evaluate δ's. This is because it is unknown in general whether a value lies between the bounds of the sum. The example below demonstrates this.

$$\text{AntiCommutator}\left[\sum_{i=1}^{T} f[i], f\dagger[j]\right]$$

$$\sum_{i=1}^{T} \delta[i, j]$$

It is unknown whether 1 < j < T, so the sum remains unevaluated. The example below demonstrates an unfortunate consequence of this feature inherited from the rules of Mathematica.

```
AntiCommutator[Sum[f[i], {i, 1, T}], f†[T - 1]]
```

$$\sum_{i=1}^{T} \delta[i, -1 + T]$$

It is obvious that this expression should evaluate to one. However, because Mathematica does not recognize this, the input remains unevaluated.

```
(T - 1) < T
```

```
-1 + T < T
```

However, if T is given a value this expression will evaluate to True.

```
T = 7;
(T - 1) < T
```

```
True
```

At this point then it is necessary to leave δ' s unevaluated or cancel them by hand when using undefined bounds in a definite sum.

VIII: Vector or Tensor Input

The functions of the DiracQ package are capable of evaluating input in the form of tensors of any order. This ability is useful for performing calculations involving vector operators, such as the position and momentum vector operators $\vec{p}$ and $\vec{q}$. These operators are included in the package. An example of their use is shown below.

Calling the position and momentum vectors as input will produce their definitions.

```
q⃗[i]
```

```
{q[i, x], q[i, y], q[i, z]}
```

```
p⃗[i]
```

```
{p[i, x], p[i, y], p[i, z]}
```

The functions of the package are written to be able to accept these operators as input.

- **Activate "q and p"**

```
Commutator[q⃗[i], p⃗[i]]
```

```
Requesting the commutator of an array A, with another array B gives an array C
  consisting of the Mathematica output C = Outer[Commutator, A, B]. For example if A
  is an array of length 'a' and B is a scalar the result is an array of length 'a'.
```

```
{{iℏ, 0, 0}, {0, iℏ, 0}, {0, 0, iℏ}}
```

The printed note informs us how the result is obtained. The package is able to perform such computations with tensors of any order. Users are free to create any tensor using any combination of operators and are not resricted to using a predefined vector such as $\vec{p}$. An example using 3 rd rank tensors of Pauli Matrices is given below.

- **Activate Pauli Spin Matrices "σ"**

First I define two 3 rd rank tensors using Pauli Matrices. These tensors are arbitrary.

```
Tensor1 = {{{σ[i, x]}, {σ[j, y]}}, {{σ[j, y]}, {σ[i, z]}}};
Tensor2 = {{{σ[i, z]}, {σ[j, x]}}, {{σ[j, x]}, {σ[i, y]}}};
```

Using the Mathematica functions TensorRank and MatrixForm provide additional information about our tensors.

```
TensorRank[Tensor1]
```

```
3
```

```
MatrixForm[Tensor1]
MatrixForm[Tensor2]
```

$$\begin{pmatrix} (\sigma[i, x]) & (\sigma[j, y]) \\ (\sigma[j, y]) & (\sigma[i, z]) \end{pmatrix}$$

$$\begin{pmatrix} (\sigma[i, z]) & (\sigma[j, x]) \\ (\sigma[j, x]) & (\sigma[i, y]) \end{pmatrix}$$

The DiracQ functions Commutator and ProductQ are capable of performing operations involving the noncommutative objects within these tensors.

```
tensorcommutator = Commutator[Tensor1, Tensor2]
```

```
Requesting the commutator of an array A, with another array B gives an array C
  consisting of the Mathematica output C = Outer[Commutator, A, B]. For example if A
  is an array of length 'a' and B is a scalar the result is an array of length 'a'.
```

```
{{{{{{-2 i σ[i, y]}, {0}}, {{0}, {2 i σ[i, z]}}}}},
  {{{{2 i δ[i, j] ** σ[j, x]}, {-2 i σ[j, z]}}, {{-2 i σ[j, z]}, {0}}}}},
 {{{{2 i δ[i, j] ** σ[j, x]}, {-2 i σ[j, z]}}, {{-2 i σ[j, z]}, {0}}}},
  {{{{0}, {2 i δ[i, j] ** σ[i, y]}}, {{2 i δ[i, j] ** σ[i, y]}, {-2 i σ[i, x]}}}}}}
```

The result is made more clear using MatrixForm.

```
MatrixForm[tensorcommutator]
```

$$\begin{pmatrix} \begin{pmatrix} \begin{pmatrix} -2\,i\,\sigma[i, y] \\ 0 \end{pmatrix} & \begin{pmatrix} 0 \\ 2\,i\,\sigma[i, z] \end{pmatrix} \end{pmatrix} & \begin{pmatrix} \begin{pmatrix} 2\,i\,\delta[i, j] ** \sigma[j, x] \\ -2\,i\,\sigma[j, z] \end{pmatrix} & \begin{pmatrix} -2\,i\,\sigma[j, z] \\ 0 \end{pmatrix} \end{pmatrix} \\ \begin{pmatrix} \begin{pmatrix} 2\,i\,\delta[i, j] ** \sigma[j, x] \\ -2\,i\,\sigma[j, z] \end{pmatrix} & \begin{pmatrix} -2\,i\,\sigma[j, z] \\ 0 \end{pmatrix} \end{pmatrix} & \begin{pmatrix} \begin{pmatrix} 0 \\ 2\,i\,\delta[i, j] ** \sigma[i, y] \end{pmatrix} & \begin{pmatrix} 2\,i\,\delta[i, j] ** \sigma[i, y] \\ -2\,i\,\sigma[i, x] \end{pmatrix} \end{pmatrix} \end{pmatrix}$$

Using TensorRank we see we now have a tensor of rank 6

```
TensorRank[tensorcommutator]
```

```
6
```

The product of the tensors can be computed as well.

```
MatrixForm[Tensor1 ⊗ Tensor2]
```

```
Requesting the product of an array A, with another array B gives an array C consisting
  of the Mathematica output C = Outer[ProductQ, A, B]. For example if A is an
  array of length 'a' and B is a scalar the result is an array of length 'a'.
```

$$\begin{pmatrix} \begin{pmatrix} \begin{pmatrix} -i\,\sigma[i, y] \\ \sigma[i, x] ** \sigma[j, x] \end{pmatrix} & \begin{pmatrix} \sigma[i, x] ** \sigma[j, x] \\ i\,\sigma[i, z] \end{pmatrix} \end{pmatrix} & \begin{pmatrix} \begin{pmatrix} \sigma[j, y] ** \sigma[i, z] \\ -i\,\sigma[j, z] \end{pmatrix} & \begin{pmatrix} -i\,\sigma[j, z] \\ \sigma[j, y] ** \sigma[i, y] \end{pmatrix} \end{pmatrix} \\ \begin{pmatrix} \begin{pmatrix} \sigma[j, y] ** \sigma[i, z] \\ -i\,\sigma[j, z] \end{pmatrix} & \begin{pmatrix} -i\,\sigma[j, z] \\ \sigma[j, y] ** \sigma[i, y] \end{pmatrix} \end{pmatrix} & \begin{pmatrix} \begin{pmatrix} 1 \\ \sigma[i, z] ** \sigma[j, x] \end{pmatrix} & \begin{pmatrix} \sigma[i, z] ** \sigma[j, x] \\ -i\,\sigma[i, x] \end{pmatrix} \end{pmatrix} \end{pmatrix}$$

- **NCCross**

The function NCCross is capable of performing cross products.

Example:

As an example consider the orbital angular momentum operator $\vec{L}$ and a further cross product with the position vector $\vec{q}$. Note that $\vec{q}$ and $\vec{p}$ need to be specified with an argument, arbitrarily chosen here to be i.

- **Activate "q and p"**

```
L = NCcross[q[i], p[i]]
```

```
{q[i, y] ** p[i, z] - q[i, z] ** p[i, y],
 -q[i, x] ** p[i, z] + q[i, z] ** p[i, x], q[i, x] ** p[i, y] - q[i, y] ** p[i, x]}
```

```
v1 = NCcross[q[i], L]
```

```
{q[i, y] ** (-q[i, y] ** p[i, x]) - q[i, z] ** (-q[i, x] ** p[i, z]) +
  q[i, y] ** q[i, x] ** p[i, y] - q[i, z] ** q[i, z] ** p[i, x],
 -q[i, x] ** (-q[i, y] ** p[i, x]) + q[i, z] ** (-q[i, z] ** p[i, y]) -
  q[i, x] ** q[i, x] ** p[i, y] + q[i, z] ** q[i, y] ** p[i, z],
 q[i, x] ** (-q[i, x] ** p[i, z]) - q[i, y] ** (-q[i, z] ** p[i, y]) +
  q[i, x] ** q[i, z] ** p[i, x] - q[i, y] ** q[i, y] ** p[i, z]}
```

Here we see that the non commutative nature is redundant in defining $\vec{L}$ as such, since the canonical pairs $q[x]$ and $p[x]$ are kept away from each other already. However, in the next cross product, it is relevant since the canonical pairs are no longer separated. This function is used in the Runge Lenz vector definition as an example.

Further note: if one wants a neater answer with negative signs pulled out, we can map the DiracQ function SimplifyQ to each component of the output.

```
SimplifyQ /@ v1
```

```
{q[i, x] ** q[i, y] ** p[i, y] + q[i, x] ** q[i, z] ** p[i, z] -
  q[i, y] ** q[i, y] ** p[i, x] - q[i, z] ** q[i, z] ** p[i, x],
 -q[i, x] ** q[i, x] ** p[i, y] + q[i, x] ** q[i, y] ** p[i, x] + q[i, y] ** q[i, z] ** p[i, z] -
  q[i, z] ** q[i, z] ** p[i, y], -q[i, x] ** q[i, x] ** p[i, z] +
  q[i, x] ** q[i, z] ** p[i, x] - q[i, y] ** q[i, y] ** p[i, z] + q[i, y] ** q[i, z] ** p[i, y]}
```

IX: Adding/Deleting Operators and Defining New Commutation Relations

The DiracQ Package allows the user to augment the package by defining new operators that will be recognized by the DiracQ functions. The process for doing so is simple and is explained here using the the Virasoro Lie algebra of 2 dimensional conformal theory with a central extension.

"Operators" is the list of operators that are currently active, as explained in the usage paragraph below.

```
? Operators
```

Operators is the list of symbols that are currently being recognized as operators. The population of the list should correspond with the operators selected using the DiracQ palette as well as any user defined operators.

Since we have no operators activated, the list should be empty, as shown below.

```
Operators
```

```
{}
```

The function AddOperators is used to add user defined operators. DeleteOperators will remove user defined operators.

```
? AddOperator
```

AddOperator allows users to expand the number of symbols that can be specified as noncommutative objects. The argument of AddOperator is the symbol that represents the new operator. Algebraic relations for new operators such as basic commutators, anticommutators, and products must be defined by the user. To define a basic commutator for two operators's α and β, input CommutatorDefinition[α,β]:=_ , where the blank is the definition. AntiCommutatorDefinition[α,β] is the equivelant function for anticommutators and OperatorProduct[α,β] is the equivelant function for definition of operator products. Any number of such definitions can be input. If a function calls a definition that has not been input by the user the output will read 'Null'.

```
? DeleteOperator
```

DeleteOperator will remove a user defined operator from the list of possible
operators. The argument of delete operators is the symbol by which the operator represented.

This is easy, except for defining the algebraic relations of a new operator. Let' s try an example.
We will define a new operator that is represented by the symbol "L". This operator is not to be confused with the angular momentum operator, but denotes a generator of the Virasoro algebra arising in confromal field theory as detailed below.

```
AddOperator[L]
```

```
Please enter all neccessary basic commutation
  and anticommutation relations. For help type ?AddOperator
```

L is now on our list of operators. We can check it explicitly using the command "Operators".

```
Operators
```

```
{L}
```

L will be treated as an operator by the package. The package does not know any of the properties of L, however. Below we evaluate a commutator of two L operators. The package uses the basic definition of a commutator.

```
Commutator[L[i], L[j]]
```

```
L[i] ** L[j] - L[j] ** L[i]
```

It is useful to define algebraic relations of our new operator, such as simplified evaluations of the commutator of two θ operators. We do so according to the method described in the AddOperators usage paragraph (?AddOperator). The commutator definition I use here is from the Virasoro algebra of conformal algebra.

```
CommutatorDefinition[L[i_], L[j_]] := (i - j) L[i + j] + c / 12 δ[i + j, 0] (i^3 - i)
```

Here the central charge "c" is a c-number, commuting with all operators. We now see that when we perform the commutator of two L operators the above definition is utilized correctly:

```
Commutator[L[2], L[-1]]
```

```
3 L[1]
```

```
Commutator[L[2], L[-2]]
```

$$\frac{c}{2} + 4\,L[0]$$

We can easily show that L[0], L[1] and L[-1] form a subgroup not involing the central charge "c".

```
Commutator[L[1], L[-1]]
```

```
2 L[0]
```

```
Commutator[L[1], L[0]]
```

```
L[1]
```

```
Commutator[L[-1], L[0]]
```

```
-L[-1]
```

The commutator definition can be utilized in more complicated operations as well.

```
StandardOrderQ[Commutator[ ∑_i L[i] ** L[-i], L[j]]]
```

$$\frac{1}{6}\,(c\,j)\,**\,L[j]-\frac{1}{6}\,\left(c\,j^3\right)\,**\,L[j]-\sum_i i\,**\,L[i]\,**\,L[-i+j]+$$
$$\sum_i i\,**\,L[i+j]\,**\,L[-i]-\sum_i j\,**\,L[i]\,**\,L[-i+j]-\sum_i j\,**\,L[i+j]\,**\,L[-i]$$

By default the commutator of L with an operator whose ``head'' is not L is taken to be zero.

```
Commutator[L[i], f†[i]]
```

```
0
```

If we want to change this we must enter in a different CommutatorDefinition. Similar rules apply to AntiCommutator and as well.

The processes of defining anticommutation relations and product relations are identical except that AntiCommutatorDefinition and OperatorProduct are used respectively.

If we no longer need L around as an operator, we can delete it through DeleteOperator

```
DeleteOperator[L]
```

```
{}
```

We now see the list of operators without L. The output below shows that L is no longer treated as a noncommutative operator.

```
Commutator[L[a], L[b]]
```

```
0
```

X: Special Symbols

Special symbols and the shorthand for writing them can be found in the DiracQ palette as well as any of the *Mathematica* palettes. In general special symbols are called by sandwiching some series of icons between two [ESC] keystrikes. A list of special symbols used in this paper and their shorthand is given below.

```
Text[Grid[{{Object, Symbol, Keyboard Entry}, {Sigma, σ, [ESC]s[ESC]}, {Hbar, ℏ, [ESC]hb[ESC]},
   {Dagger, †, [ESC]dg[ESC]}, {Delta, δ, [ESC]d[ESC]}, {CircleProduct, "⊗", "[ESC]c*[ESC]"},
   {Subscript, "\!\(\*SubscriptBox[\(□\), \(□\)]\)", "[CTRL]-"},
   {Superscipt, "\!\(\*SuperscriptBox[\(□\), \(□\)]\)", [CTRL]6},
   {"Vector Head", "\!\(\*OverscriptBox[\(□\), \(⇀\)]\)", [CTRL]7 [ESC]vec[ESC]}}, Frame -> All]]
```

Object	Symbol	Entry Keyboard
Sigma	σ	[ESC]s[ESC]
Hbar	$\hbar$	[ESC]hb[ESC]
Dagger	$\dagger$	[ESC]dg[ESC]
Delta	δ	[ESC]d[ESC]
CircleProduct	$\otimes$	[ESC]c*[ESC]
Subscript	$\square_{\square}$	[CTRL]−
Superscipt	$\square^{\square}$	[CTRL]6
Vector Head	$\vec{\square}$	[CTRL]7 [ESC]vec[ESC]

XI: The Kronecker δ and the Levi - Civita Symbol "ϵ"

The Kronecker δ and the Levi - Civita Symbol ϵ are useful shorthand notations. The attributes of these notational shorthands are given in (33) and (34)

$$\begin{aligned} \delta_{i,j} &= 1 \quad \text{if } i = j \\ &= 0 \quad \text{otherwise} \end{aligned} \tag{1}$$

$$\begin{aligned} \epsilon_{i,j,k} &= 1 \quad \text{if } \{i, j, k\} = \{1, 2, 3\},\ \{2, 3, 1\}, \text{ or } \{3, 1, 2\} \\ &= -1 \text{ if } \{i, j, k\} = \{1, 3, 2\}, \{2, 1, 3\}, \{3, 2, 1\} \\ &= 0 \quad \text{if } i = j,\ j = k, \text{ or } k = i \end{aligned} \tag{2}$$

Mathematica includes the functions KroneckerDelta and LeviCivitaSymbol corresponding to these objects. Although these functions are useful, the package includes it's own version of these two functions. The Kronecker δ used in the package is simply represented by the symbol δ. This function is almost identical to that of the existing *Mathematica* function. Its use is demonstrated in the examples below.

```
δ[1, 1]
```

```
1
```

```
δ[1, 2]
```

```
0
```

```
δ[i, j]
```

```
δ[i, j]
```

Note that if two symbols are used that have not yet been assigned numerical values the δ function is not evaluated. The package does not "know" if i is equal to j, so it leaves the δ unevaluated. If it is required that each symbol is viewed as being not identical to another symbol the option Evaluation should be set to Identical, as shown below.

```
? Evaluation
```

```
Evaluation is an option for the Kronecker–δ function. If Evaluation
    is set to Identical the δ will evaluate to zero unless both arguments are indentical.
```

```
δ[i, j, Evaluation → Identical]
```

```
0
```

This option is useful when using the Kronecker delta for coordinate directions, for example, as we know that x is not equal to y.

The package includes the function ϵ for the Levi-Civita symbol. It is defined only for coordinate directions x, y, and z. For use with symbols other than these use LeviCivitaTensor. Examples of the use of the ϵ function are given below.

```
ϵ[x, y, z]
```

```
1
```

```
ϵ[z, y, x]
```

```
-1
```

```
ϵ[x, y, x]
```

```
0
```

XII: Use of Times, NonCommutativeMultiply, and ProductQ

During the use of this package it is necessary to use the *Mathematica* functions Times (input as "*" or a space between terms) and NonCommutativeMultiply ("**"). The operation of these functions is not affected by the DiracQ package. Furthermore, the DiracQ package contains the function ProductQ, used to multiply two terms. This means that there are three functions used for multiplication. Determining which of the available three multiplication functions is applicable in a certain situation is nontrivial. All of which are useful, and occasionally they can be interchanged but in general the correct function must be used. **Mathematica automatically sorts all input of the Times function**. For this reason we cannot use Times between operators, as the order of the operators may be changed before any other function can step in to preserve the order. **Therefore NonCommutativeMultiply (**) must always be used between operators.** ProductQ (⊗) is used when simplification and application of algebraic relations is desired. **NonCommutativeMultiply is a *Mathematica* function and will not apply proprerties known to the DiracQ package.**

The rules for using these functions can be summarized as follows :

- Use Times between two terms if the input is identical under permutation of said terms (i.e. if operator ordering is unimportant or if working with commutative objects).
- Use NonCommutativeMultiply between two terms if the input depends on the order of the terms and if the terms contain items that are connected by only Times or NonCommutativeMultiply.
- Use ProductQ (also invokable through ⊗) between two terms if the terms involve a variety of functions (Sum, Plus, etc.) or to sort and apply definitions to the result. This is more powerful than NonCommutativeMultiply, and makes use of the known properties of the various operators.

■ **Ex. (1) :** In this example, taken from the tJ model, I try to use Times between operators and then apply NonCommutativeMultiply. Notice that the order of the operators is reversed. Activate Hubbard (X) operators.

Incorrect Input:

```
Hamiltonian = ∑j ∑k ∑σj ∑σk t[j, k] X[k, σk, 0] X[j, 0, σj] /. Times → NonCommutativeMultiply
```

$$\sum_{j}\sum_{k}\sum_{\sigma j}\sum_{\sigma k} t[j, k] ** X[j, 0, \sigma j] ** X[k, \sigma k, 0]$$

Notice that the order of the X operators has been reversed in the output.

Correct Input:

```
Hamiltonian = ∑j ∑k ∑σj ∑σk t[j, k] X[k, σk, 0] ** X[j, 0, σj]
```

$$\sum_{j}\sum_{k}\sum_{\sigma j}\sum_{\sigma k} X[k, \sigma k, 0] ** X[j, 0, \sigma j]\, t[j, k]$$

Mathematica has sorted the resultant expression, and the operators have not been permuted.

■ **Ex. (2):** Here we see that NonCommutativeMultiply will not apply properties that are known to the DiracQ package. Activate Pauli Spin Matrices (σ).

Product using NonCommutativeMultiply:

```
σ[i, x] ** σ[i, x]
```

```
σ[i, x] ** σ[i, x]
```

Product using ProductQ (ProductQ can also be entered with the ⊗ symbol, both input styles are shown below).

```
ProductQ[σ[i, x], σ[i, x]]
```

```
1
```

```
σ[i, x] ⊗ σ[i, x]
```

```
1
```

Users should be warned that here the result of squaring the Pauli matrix is taken as unity, in the informal language of DiracQ. More rigorously the result is an identity matrix, but sticking to Dirac's informality of expression has many advantages, as most quantum physicists recognize.

- **Ex. 3:** Here we see that it is not necessary to use ⊗ between every term in a long expression. The entire expression will be simplified if ⊗ is used once. Multiple uses of ⊗ will significantly slow down longer computations.

Correct but redundant use of ⊗:

```
((σ[1, x] ⊗ σ[2, z]) ⊗ σ[1, x]) ⊗ σ[2, z]
```

```
1
```

More efficient Input :

```
σ[1, x] ** σ[2, z] ** σ[1, x] ⊗ σ[2, z]
```

```
1
```

Another Method using SimplifyQ :

```
SimplifyQ[σ[1, x] ** σ[2, z] ** σ[1, x] ** σ[2, z]]
```

```
1
```