Use ACL2 to Verify the Correctness of Lowering Operator when $j = l + s$

Xiaohui Chen

April 26, 2015

Outline

Clebsch-Gordan Coefficient

Ket notation

$|j, m_j\rangle$, $|l, m_l\rangle$,$|s, m_s\rangle$, etc

$j, m_j, l, m_l, ...$ are quantum numbers

The Ket notation represents the state when the particle as the corresponding quantum numbers

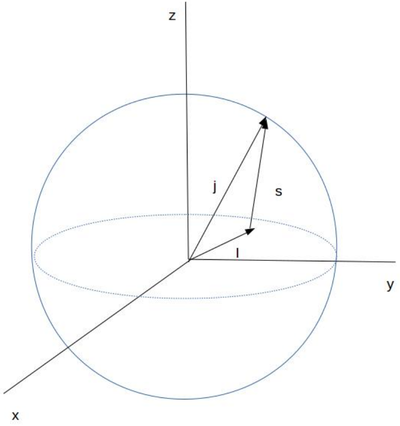
The coefficient

$$|j, m_j\rangle = \sum_{m_j = m_l + m_s} c_n |l, m_l\rangle |s, m_s\rangle$$

Here c_ns are called the Clebsch-Gordan Coefficients

$|c_n|^2 \equiv$ Given the combined state $|j, ml\rangle$ of a particle, we can know that the particle is in $|l, m_l\rangle|s, m_s\rangle$ with probability $|c_n|^2$

Therefore $\sum_{m_j = m_l + m_s} |c_n|^2 = 1$


z
j
s
l
y
x

Lowering-Operators

$$L_-|l\ m_l\rangle = \hbar\sqrt{l(l+1) - m_l(m_l \pm 1)}|l\ (m_l - 1)\rangle \tag{1}$$

$$S_-|s\ m_s\rangle = \hbar\sqrt{s(s+1) - m_s(m_s - 1)}|s\ (m_s - 1)\rangle \tag{2}$$

Since $\vec{j} = \vec{l} + \vec{s}$, J_- obeys the following properties:

$$J_- = L_- + S_- \tag{3}$$

$$J_-|j\ m_j\rangle = \hbar\sqrt{j(j+1) - m_j(m_j - 1)}|s\ (m_j - 1)\rangle \tag{4}$$

Example

We have an electron with angular momentum $l = 1$ and spin $s = \frac{1}{2}$

Therefore $m_l = -1, 0, -1$ and $m_s = \pm\frac{1}{2}$

This means $j = \frac{3}{2}$ and $m_j = -\frac{3}{2}, -\frac{1}{2}, \frac{1}{2}, \frac{3}{2}$

We know that the only possibility of forming $|j\ m_j\rangle = |\frac{3}{2}\ \frac{3}{2}\rangle$ is $|\frac{3}{2}\ \frac{3}{2}\rangle = |1\ 1\rangle|\frac{1}{2}\ \frac{1}{2}\rangle$.

Applying the lowering operator on the above equation

$$J_-|\tfrac{3}{2}\ \tfrac{3}{2}\rangle = \hbar\sqrt{\tfrac{3}{2}\left(\tfrac{3}{2}+1\right) - \tfrac{3}{2}\left(\tfrac{3}{2}-1\right)}|\tfrac{3}{2}\ \tfrac{1}{2}\rangle = \sqrt{3}\hbar|\tfrac{3}{2}\ \tfrac{1}{2}\rangle$$

$$J_-|1\ 1\rangle|\tfrac{1}{2}\ \tfrac{1}{2}\rangle = (L_- + S_-)|1\ 1\rangle|\tfrac{1}{2}\ \tfrac{1}{2}\rangle = L_-|1\ 1\rangle|\tfrac{1}{2}\ \tfrac{1}{2}\rangle + |1\ 1\rangle S_-|\tfrac{1}{2}\ \tfrac{1}{2}\rangle = \hbar\sqrt{1(1+1) - 1(1-1)}|1\ 0\rangle|\tfrac{1}{2}\ \tfrac{1}{2}\rangle + \hbar\sqrt{\tfrac{1}{2}\left(\tfrac{1}{2}+1\right) - \tfrac{1}{2}\left(\tfrac{1}{2}-1\right)}|1\ 1\rangle|\tfrac{1}{2}\ -\tfrac{1}{2}\rangle = \hbar\sqrt{2}|1\ 0\rangle|\tfrac{1}{2}\ \tfrac{1}{2}\rangle + \hbar|1\ 1\rangle|\tfrac{1}{2}\ -\tfrac{1}{2}\rangle$$

$$\sqrt{3}\hbar|\tfrac{3}{2}\ \tfrac{1}{2}\rangle = \hbar\sqrt{2}|1\ 0\rangle|\tfrac{1}{2}\ \tfrac{1}{2}\rangle + \hbar|1\ 1\rangle|\tfrac{1}{2}\ \ -\tfrac{1}{2}\rangle$$

$$\therefore |\tfrac{3}{2}\ \tfrac{1}{2}\rangle = \sqrt{\tfrac{2}{3}}|1\ 0\rangle|\tfrac{1}{2}\ \tfrac{1}{2}\rangle + \tfrac{1}{\sqrt{3}}|1\ 1\rangle|\tfrac{1}{2}\ \ -\tfrac{1}{2}\rangle$$

My Model

```
((A . (j . m_j)) . ;j-state
(B . ((l . m_l) . (s . m_s))) ;coupled-state
(C . ((l . m_l) . (s . m_s)))
...)
```

or

```
(0 . 0) ;This happens when the lowering operators are
    operated on the lowest states
```

Properties of Quantum States

1. $A = B + C + \dots$
2. $j = |l-s|, |l-s|+1, ..., l+s$. But here we only consider the case where $j = l+s$
3. $m_j = m_l + m_s$
4. $m_l = -l, -l+1, ..., l$, $m_s = -s, -s+1, ..., s$
5. From the previous condition, we can know that $m_j = -j, -j+1, ..., j$
6. l, s can be integers or half-integers (with denominator 2)
7. m_l, m_s can be integers or haldf-integers

8. In my model, j-state and coupled-states are either both 0 or pairs and lists as shown in previous slides

My Implementation

j-state

(A . (j . m_j))

Properties

```lisp
(defun j-coefficient (x)
       (car x))

(defun quantum-j (x)
       (car (cdr x)))

(defun quantum-mj (x)
       (cdr (cdr x)))

(defun lower-or-lowest-jstate (x)
        (or (>= (- (quantum-mj x)) (quantum-j x))
            (equal (j-coefficient x) 0)))
```

```
(defun half-or-full-integer (x)
       (xor (equal (denominator x) 1)
            (equal (denominator x) 2)))

(defun half-full-match (x y)
       (and (iff (equal (denominator x) 1)
                 (equal (denominator y) 1))
            (iff (equal (denominator x) 2)
                 (equal (denominator y) 2))))

(defun rational-pair (x)
       (if (atom x)
           nil
           (and (rationalp (car x))
```

```
                    (< 0 (car x))
                    (rationalp (cdr x))
                    (natp (- (car x) (abs (cdr x))))
                    (half-or-full-integer (car x))
                    (half-or-full-integer (cdr x))
                    (half-full-match (car x) (cdr x))
                    (<= (abs (cdr x)) (car x)))))

(defun true-jstate (a)
        (if (atom a)
            (equal a 0)
            (and (rationalp (car a))
                 (<= 0 (car a))
                 (rational-pair (cdr a)))))
```

J Lowering Operator

```
(defun j-lowering-operator (x)
        (if (atom x)
            0
            (if (lower-or-lowest-jstate x)
                0
                (cons (* (j-coefficient x)
                         (+ (expt (quantum-j x) 2)
                            (quantum-j x)
                            (- (expt (quantum-mj x) 2))
                            (quantum-mj x)))
                      (cons (quantum-j x)
                            (- (quantum-mj x) 1))))))
```

j-state Theorem

```
(defthm j-lowering-valid
        (implies (true-jstate x)
                 (true-jstate (j-lowering-operator x)))
:hints (("Goal" :in-theory
                (disable same-denominator-add
                         remove-strict-inequalities
                         remove-weak-inequalities))))
```

Coupled State

Properties

```
(defun first-coupled-state (x)
       (car x))

(defun first-coupled-state-pair (x)
       (cdr (car x)))

(defun first-coupled-coefficient (x)
       (car (first-coupled-state x)))

(defun first-coupled-l-state (x)
       (car (cdr (first-coupled-state x))))
```

```
(defun first-coupled-s-state (x)
        (cdr (cdr (first-coupled-state x))))

(defun first-coupled-l (x)
        (car (first-coupled-l-state x)))

(defun first-coupled-ml (x)
        (cdr (first-coupled-l-state x)))

(defun first-coupled-s (x)
        (car (first-coupled-s-state x)))

(defun first-coupled-ms (x)
        (cdr (first-coupled-s-state x)))
```

```
(defun true-coupled-list (x)
        (if (atom x)
            t
            (and (rationalp (first-coupled-coefficient
                    x))
                 (<= 0 (first-coupled-coefficient x))
                 (rational-pair (first-coupled-l-state
                      x))
                 (rational-pair (first-coupled-s-state
                      x))
                 (true-coupled-list (cdr x)))))

(defun true-coupled-state (x)
        (if (atom x)
```

```
(equal x 0)
(true-coupled-list x)))
```

Clean up Zeros in the List

```
(defun clean-up-zero-list (x)
        (if (atom x)
            x
            (if (equal (car x) 0)
                (clean-up-zero-list (cdr x))
                (cons (car x)
                      (clean-up-zero-list (cdr x))))))
(defun clean-up-zero (x)
        (if (atom x)
            0
            (if (all-zeros x)
                0
                (clean-up-zero-list x))))
```

clean-up-zero Theorem

```
(defthm clean-up-zero-valid
        (implies (true-coupled-state x)
                 (true-coupled-state (clean-up-zero x))
                    )
:hints (("Goal" :in-theory
                (disable same-denominator-add
                         remove-strict-inequalities
                         remove-weak-inequalities))))
```

l-lowering-operator

```
(defun l-lowering-operator-helper (x)
        (if (atom x)
            nil
            (cons (l-lowering-to-state x)
                  (l-lowering-operator-helper (cdr x)))
                     ))

(defun l-lowering-operator (x)
        (if (atom x)
            0
            (clean-up-zero (l-lowering-operator-helper
               x))))
```

l-lowering Theorem

```
(defthm l-lowering-valid
        (implies (true-coupled-state x)
                 (true-coupled-state (
                    l-lowering-operator x)))
:hints (("Goal" :in-theory
                (disable same-denominator-add
                         remove-strict-inequalities
                         remove-weak-inequalities))))
```

s-lowering-operator

```
(defun s-lowering-operator-helper (x)
      (if (atom x)
          nil
          (cons (s-lowering-to-state x)
                (s-lowering-operator-helper (cdr x)))
                  ))

(defun s-lowering-operator (x)
      (if (atom x)
          0
          (clean-up-zero (s-lowering-operator-helper
            x)))))
```

s-lowering Theorem

```
(defthm s-lowering-valid
        (implies (true-coupled-state x)
                 (true-coupled-state (
                    s-lowering-operator x)))
:hints (("Goal" :in-theory
                (disable same-denominator-add
                         remove-strict-inequalities
                         remove-weak-inequalities))))
```

Merge and Append

After l-lowering:

((A . ((l . m_l) . (s . m_s)))
(B . ((l' . m_l') . (s' . m_s')))))

After s-lowering:

((C . ((l' . m_l') . (s' . m_s')))
(D . ((l'' . m_l'') . (s'' . m_s'')))))

After append-and-merge-states

```
((A . ((l . m_l) . (s . m_s)))
(E . ((l' . m_l') . (s' . m_s')))
(D . ((l'' . m_l'') . (s'' . m_s''))))
```

$$E = [(l+s)^2 - (m_l+m_s)^2 + l + s - m_l - m_s] * \frac{(2l)!(2s)!(l+s+m_l+m_s)!(l+s-m_l-m_s)!}{(2*(l+s))!(l+m_l)!(l-m_l)!(s+m_s)!(s-m_s)!}$$ **[?]**

Append and Merge Theorem

```
(defthm append-valid
        (implies (and (true-coupled-state x)
                      (true-coupled-state y))
              (true-coupled-state (
                 append-and-merge-states x y))))
```

Quantum State

```
(defun get-jstate (x)
        (car x))

(defun get-coupled-state (x)
        (cdr x))

(defun sum-of-coupled-coefficient (a)
        (if (atom a)
            0
            (+ (first-coupled-coefficient a)
               (sum-of-coupled-coefficient (cdr a)))))
```

```lisp
(defun equal-j (x y)
       (if (atom y)
           t
           (and (equal (quantum-j x)
                       (+ (first-coupled-l y)
                          (first-coupled-s y)))
                (equal-j x (cdr y)))))
(defun equal-mj (x y)
       (if (atom y)
           t
           (and (equal (quantum-mj x)
                       (+ (first-coupled-ml y)
                          (first-coupled-ms y)))
                (equal-mj x (cdr y)))))
```

What is a real Quantum state

```
(defun true-quantum-state (x)
        (xor (and (equal (get-jstate x) 0)
                  (equal (get-coupled-state x) 0))
             (and (true-jstate (get-jstate x))
                  (true-coupled-state (
                     get-coupled-state x))
                  (equal (sum-of-coupled-coefficient
                                   (get-coupled-state x))
                         (caar x))
                  (equal-j (get-jstate x)
                           (get-coupled-state x))
                  (equal-mj (get-jstate x)
                            (get-coupled-state x))))))
```

Normalize

```
(defun normalize-state (x)
        (if (and (equal (get-jstate x) 0)
                 (equal (get-coupled-state x) 0))
            x
            (if (< 0 (car (get-jstate x)))
                (cons (cons '1 (cdr (get-jstate x)))
                          (normalize-helper (car (
                              get-jstate x))
                                          (
                                             get-coupled-sta
                                              x)))
                (cons '0 '0))))
```

Normalize Theorem

```
(defthm normalize-valid
        (implies (true-quantum-state x)
                 (true-quantum-state (normalize-state x
                    ))))
```

Quantum Operator

```
(defun quantum-operator-helper (x)
       (cons (j-lowering-operator (get-jstate x))
             (append-and-merge-states
                (l-lowering-operator (get-coupled-state
                    x))
                (s-lowering-operator (get-coupled-state
                    x)))))
(defun quantum-operator (x)
        (if (and (equal (get-jstate x) 0)
                 (equal (get-coupled-state x) 0))
            x
        (quantum-operator-helper (normalize-state x))))
```

Initial State

1. $((1 \;.\; (j \;.\; m_j)) \;.\; ((1 \;.\; ((l \;.\; m_l) \;.\; (\mathsf{s} \;.\; m_s)))))$

2. $m_l = l$, $m_s = s$, $m_j = j$

For initial state, we can be sure that the coefficients on both sides are 1

Main Theorem 1

```
(DEFTHM INITIAL-STATE-LOWERING-VALID
        (IMPLIES (INITIAL-QUANTUM-STATE X)
                 (TRUE-QUANTUM-STATE (QUANTUM-OPERATOR
                    X))))
```

Main Theorem 2 (To be proved in the future)

```
(skip-proofs
(defun all-lowering-valid (x)
        (if (equal x (cons '0 '0))
            t
            (and (true-quantum-state
                        (quantum-operator x))
                 (all-lowering-valid
                        (quantum-operator x))))))

(defthm all-quantum-lowering-valid
        (implies (inital-quantum-state x)
                 (all-lowering-valid x)))
```

References

[1] M. Tukerman. The general problem.