*i*Journals: International Journal of Software & Hardware Research in Engineering (IJSHRE)
ISSN-2347-4890
Volume 10 Issue 8 August 2022

Complex Number, the Success for the Uniqueness of Natural Language Programming

Author : Weihan Huang

Master of Computer Science Department, State University of New York, at Buffalo, U.S.A.

Master of Physics Department, National Hsing Hua University, Taiwan

Email : weihanh@yahoo.com.tw

DOI: 10.26821/IJSHRE.10.8.2022.100801

ABSTRACT

Firstly I give introductions to complex number and Natural Language Programming syntax, and I show that the expressive power of Natural Language Programming includes the complex number syntax inside. And next I show how enNLP(Natural Language Programming in English) which combines natural language programming and Object Oriented Programming implements complex number as a class in Object Oriented Programming. Then I give the code and the tests of the code. Lastly I compare enNLP with other programming languages I know so far and show that Natural Language Programming Language is the unique programming language that can let users implement complex number both semantically and syntactically.

***Keywords**: complex number, natural language programming, programming language*

1. INTRODUCTION TO COMPLEX NUMBER

Every complex number has its normal form (a+bi) or (a-bi). Where i=SQRT(-1) and i*i=-1, a and b are real numbers. For example, 3+4i and 5-7i are complex numbers.

A useful term is the Conjugate Complex Number. The conjugate complex number of complex number (a+bi) is (a-bi).

A property of conjugate complex number is that the product of the complex number and its conjugate number is a real number.

(a+bi)*(a-bi)=a^2+b^2.

This will be useful for us to compute the division of complex numbers. (c+di)/(e+fi). Multiply both the upper and lower part by the conjugate number of the divider, we get the new divider which is a real number e^2+f^2.

2. INTRODUCTION TO NATURAL LANGAUGE PROGRAMMING SYNTAX[1]

2.1 Natural Language Programming (NLP) Syntax

In Java[2], C[3], or C++[4], Pascal[5], lisp[6], and prolog[7], they adopt the mathematical syntax of functions, i.e. f(arg1, arg2). For example, bool father(x,y).

I think we can have a more flexible syntax such that the expression will be closer to the natural language syntax.

e.g. x is father of y

The flexibility comes from the allowance for the arguments to appear in any place of the whole function expression. And this syntax is called the Natural Language Programming(NLP) syntax.

Other examples are

1. add $object$ to $list$
2. stop for $number$ seconds
3. search $object$ in $array$

2.2 Formal Context Free Grammar of NLP syntax

We abbreviate the math function syntax as

f(x,y).

And we abbreviate the NLP syntax as

A x B y C

The formal context free grammar is in the following :

```
sentence :=term;
sentence := term sentence;
term := word;
term:= variable;
term:=basicObject;;
term :=functionCall;
basicObject := number;
basicObject:=string;
basicObject:=character;
functionCall:=(sentence)
```

2.3 NLP Syntax Includes Complex Number Syntax

To make the NLP syntax A x B y C be the same as the complex number syntax a+bi, we can make A=empty, x=a, B=+, y=b, and C=i. Therefore NLP syntax includes complex number syntax.

It is also true that NLP syntax A x B y C includes math function syntax f(x,y). Let A be f, x be x., B=empty, y=y, C=empty.

And it is obvious to see that the math function syntax f(x,y) and the complex number syntax a+bi are mutually exclusive.

All these mean that NLP can implement complex number syntactically while the math function programming languages cannot implement complex number syntactically.

Examples :

Examples of math function syntax : write “abc”;

Examples of complex number syntax : 2+3i

Examples of NLP syntax : write “ABC, 2+3i, stops for 5 second

3. enNLP CODE TO IMPLEMENT COMPLEX NUMBER

Here I implement complex number as an object oriented programming class in enNLP.

```
[class]
complex number;

[module]
enNLP : public codebooks : math : basic;

[data]
real part: number : 0;
imaginary part : number : 0;

 [constructor]
 new complex number $complex number${}

 new complex number $complex number$
                        real part$number1$
                        imaginary part$number2$ {
       let(real part of $complex number$)
                                            be$number1$;
    let(imaginary part of $complex number$)
                                          be$number2$;
}

[function]
string of $complex number$
              : return $string$ { …. }

i : return $complex number$ {
      return(new complex number
                    real part 0 imaginary part 1);
}

$number$ i : return $complex number$ {
      return(new complex number
        real part0 imaginary part $number$);
}

- $complex number 1$
                    : return $complex number 2$ {
        return(new complex number
            real part (- (real part of
                            $complex number 1$))
            imaginary part (-(imaginary part of
                            $complex number 1$));
}

$complex number1$+$complex number2$
    : return $complex number 3$ { …….}

$number1$+$complex number2$
```

```
50        : return $complex number 3$ { …….}
51
52  $complex number1$+$number2$
53      : return $complex number 3$ { …….}
54
55  $complex number1$-$complex number2$
56      : return $complex number 3$ { …….}
57
58  $number1$-$complex number2$
59    : return $complex number 3$ { …….}
60
61  $complex number1$-$number2$
62      : return $complex number 3$ { …….}
63
64  $complex number1$*$complex number2$
65    : return $complex number 3$ { …….}
66
67  $number1$*$complex number2$
68      : return $complex number 3$ { …….}
69
70  $complex number1$*$number2$
71      : return $complex number 3$ { …….}
72
73  $complex number1$/$complex number2$
74      : return $complex number 3$ { …….}
75
76  $number1$/$complex number2$
77      : return $complex number 3$ { …….}
78
79  $complex number1$/$number2$
80      : return $complex number 3$ { …….}
81
82  conjugate of $complex number1$ {
83      return (new complex number
84        real part (real part of
85                $complex number1$)
86        imaginary part(-(imaginary part of
87                  $complex number1$)))
88  }
```

4. EXPLANATION OF THE CODE

Line1-Line2 means this is an enNLP object oriented class whose name is “complex number”.

Line4-Line5 is the module(package) name of this class. A module can contain several classes.

Line7-Line9 is the class data(properties). For complex number, there are 2 data : real part and imaginary part. They have default values 0.

Line11 is the constructor section, where constructors get defined here. A constructor can generate an object of its class.

Line 12 rewrites the default constructor. The data of the generated object has default values 0.

Line 14-21 is a constructor that programmers can assign real part and imaginary part of the new generated object.

Line 23 means this is the start of defining functions.

There are totally 17 functions defined, counted in the following.

For each binary operator %, there are 3 functions defined

complex number % complex number

number % complex number

complex number % number

There are totally 4 binary operators + - * /. So there are 4*3=12 functions defined.

Line 46-Lin53 define + functions.
Line 55-Lin61 define - functions.
Line 64-Lin71 define * functions.
Line 73-Lin80 define / functions.

There is a unary operator -. And also two functions (i) and ($number$ i) need to be defined. So there are totally 12+3=15functions.

There is an auxiliary function "conjugate of $complex number$" defined in Line82-Line88.

There is an auxiliary function "string of $complex number$" defined.in Line 24-Line25.

So there are totally 15+2=17 functions defined
.

5. TESTS OF CODE

Test complex number.enProgram --------------------

```
new complex number $x$ real part 20 imaginary part
                                           50;
write a row $x$;    // 20+50i computed by hands

write a row (-(2i+3)); // -3-2i
write a row (-2i+3);    // 3-2i
write a row (-i+3);    // 3-i
write a row (2i+3);    // 3+2i

write a row (2i+3+4+5i+6+7i+8i+9);    // 22+22i
write a row (2i-2i+3i-4+5-6i-i);      // 1-4i
write a row (2i-2i*3i-9i+1+2);         // 9-7i
write a row ((2i+3)+(4+5i)+(6+7i)+(8i+9));
                                        // 22+22i
write a row ((5i+5)/(4i+3)); //    1.4-0.2i
write a row ((5i+5)*(4i+3)); //    -5+35i

new complex number $c$ is (4i);
write a row ((i+$c$)*(2i+2)); // -10+10i

let $c$ be (1+i);
write a row ((i+2)*($c$+1)*(1+i));    // -1+7i

let $c$ be (3+4i);
write a row ($c$*3/(4-3i));    // 3i

[use module]
enNLP : public codebooks : math : basic;
```

Test outputs ----------------------------------

```
20+50i
-3-2i
3-2i
3-1i
3+2i
22+22i
1-4i
9-7i
22+22i
1.4-0.2i
-5+35i
-10+10i
-1+7i
3i
```

So the test results match the hand computation results. The test is successful!

6. COMPARISONS WITH OTHER PROGRAMMING LANGUAGES

6.1 Compare with C[3], Pascal[5], C++[4], and Java[2]

The main reason the above 4 programming languages cannot implement the complex number syntactically is due to that "i" is a variable in these programming languages. So if we write (3+2i) in the languages, the compilers will give an error message "unknown identifier i" or "undeclared variable i".

6.2 Compare with Lisp[6]

The problem in Lisp is that it is an operator-preorder programming. So (3+2i)is written as (+ 3 '(2 i)). Hence this is syntactically different from our desirable form of complex number 3+2i.

6.3 Compare with Prolog[7]

Prolog provides two ways to implement Complex Number a+bi

1. List [H|T] : [a|b]
2. First order term complexNumber(a,b)

Both the list form and first order term differ syntactically from complex number a+bi.

6.4 Compare with Fortran[8], Python[9]

In Fortran and Python Complex Number is built-in. This would be like to implement the complex number by another new compiler in lower level language to make complex number built in.

7. CONCLUSION

In this paper, by the comparisons with other programming languages I know so far, I try to show that Natural Language Programming is the unique programming language that can let users implement the Complex Number not only semantically but also syntactically, while the other programming languages can implement the Complex Number only semantically, but not syntactically. The success of the uniqueness comes from the expressive power of Natural Programming Language syntax. It is also precious to show that enNLP(Natural Language Programming in English) which combines both Object Oriented Programming and Natural Language Programming, can define the Complex Number as a class in Object Oriented Programming.

8. REFERENCES

[1] Weihan Huang, (2021), "Natural Language Programming", International Journal of Advances in Electronics and Computer Science-IJAECS, Volume-8,Issue-9 (Sep,2021) pp15
Link to the paper :
http://ijaecs.iraj.in//paper_detail.php?paper_id=18189&nameNatural_Language_Programming

[2] Java
https://en.wikipedia.org/wiki/Java_(programming_language)

[3] C
https://en.wikipedia.org/wiki/C_(programming_language)

[4]C++
https://en.wikipedia.org/wiki/C%2B%2B

[5] Pascal
https://en.wikipedia.org/wiki/Pascal_(programming_language)

[6] Lisp
https://en.wikipedia.org/wiki/Lisp_(programming_language)

[7] Prolog
https://en.wikipedia.org/wiki/Prolog

[8] Fortran
https://en.wikipedia.org/wiki/Fortran

[9] Python
https://en.wikipedia.org/wiki/Python_(programming_language)