International Journal of Software & Hardware Research in Engineering (IJSHRE)
ISSN:2347-4890, ijournals.in/ijshre
Volume 14 Issue 8, August 2026

Attribute Grammar by 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

https://doi.org/10.26821/IJSHRE.14.08.2026.140801

ABSTRACT

This paper mainly proposes that the compositional semantics of a natural language sentence can be expressed as a natural language programming function. The grammar categories in natural language sentences can be expressed as natural language programming classes. And the grammar in natural language can be expressed in the function name of the function which expresses compositional semantics. Therefore, this paper can be treated as a special implementation that realizes the idea of attribute grammar. Nonetheless, for the other realizer of attribute grammar, the Definite Clause Grammar in Prolog, its grammar specification uses the non-Prolog language, while in my paper the grammar specification uses natural language programming functions. So, this is the major difference between Definite Clause Grammar and my work. In the following of my paper, I will construct the parsing from input sentence and eventually to the update of knowledge database step by step. These steps will show how the compositional semantics of a natural language sentence can be expressed as a natural language programming function.

Keywords—*attribute grammar, natural language programming*

I. FIRST STEP : INITIALIZATION

A. Tokenize the input string :

e.g. "Next, I will tell" =>{ "Next",",", "I", "will", "tell"}

B. Lower case the words in sentence :

e.g. "Next", => "next",

C. Special characters change to English words

e.g. "," => "comma", "." => "dot"

II. SECOND STEP: WRITE THE GRAMMAR CATEGORY CLASSES

[class] tense;

[module] NLU by NLP : grammar category;

[parent class] grammar category;

[use module] NLU by NLP : basic structure;

Please note that the class tense is a subclass of class "grammar category". So it inherits the property "string list" and function "string of". Here is the definition of "grammar category" below.

[class] grammar category;

[module] NLU by NLP : basic structure;

[data]

```
string list : string list :{};
meaning list : list : {}
[function]
string of $grammar category$ : return $string$ {
   let $string$ be "";
    loop $i$ from 1 up to (length of (string list of
      $grammar category$)), do {    if($i$!=1), then {
                  $string$ increases by " ";
               }
               $string$ increases by ((string list of
                  $grammar category$) $i$);
      }
   }
   return $string$;
}
```

Similarly it is the same for the grammar categories "noun", "verb" and "adjective".

```
[class] noun;
[module] NLU by NLP : grammar category;
[parent class] grammar category;
[use module] NLU by NLP : basic structure;

[class] verb;
[module] NLU by NLP : grammar category;
[parent class] grammar category;
[use module] NLU by NLP : basic structure;

[class] adjective;
[module] NLU by NLP : grammar category;
[parent class] grammar category;
[use module] NLU by NLP : basic structure;
```

But please note that "pronoun" is a sub-category of "noun phrase". So we write in Natural Language Programming[1] that "pronoun" is a subclass of "noun phrase".

```
[class] pronoun;
[module] NLU by NLP : grammar category;
[parent class] noun phrase;
[use module] NLU by NLP : basic structure;
```

Hence in this section, we know how to write a grammar category as a NLP class. Next, we will show how to write the lexicon functions.

III. THIRD STEP : WRITE THE LEXICON FUNCTIONS

Note that each word in the sentence is a function that returns its grammar category class. There are two advantages by making the words as functions. Firstly, it completes the Context Free Grammar of the sentence such as

$tense$ -> "next"
$tense$ -> "will"

Secondly the word can be run by "dynamically run"[2] in natural language programming(NLP). And this makes compositional semantics can be run by "dynamically run" the sentence string parsed by parser. These will be shown in the following sections.

```
[codebook] lexicons;
[module] NLU by NLP : lexicons;
[global variable] dictionary : dictionary : new
dictionary;
```

```
[function]
next : return $tense$ {
    new tense $t$ {
    string list : {"next"}
    meaning list : (get lexicon semantics of "next"
from $$$dictionary$$$) ;}
    return $t$;
}
comma : return $comma$ {
    new comma $c$ {
    string list : {"comma"};
    meaning list : (get lexicon semantics of "comma"
from $$$dictionary$$$) ;}
    return $c$;
}
i : return $pronoun$ {
    new pronoun $p$ {
        string list : {"i"};
        meaning list : (get lexicon semantics of "i"
        from $$$dictionary$$$) ;}
    return $p$;
}
will : return $tense$ {
    new tense $t$ {
        string list : {"will"};
        meaning list : (get lexicon semantics of "will"
        from $$$dictionary$$$) ;
    }
    return $t$;
}
tell : return $verb$ {
    new verb $v$ {
        string list : {"tell"};
        meaning list : (get lexicon semantics of "tell"
            from $$$dictionary$$$) ;
    }
    return $v$;
}
the : return $determiner$ {
    new determiner $d$ {
        string list : {"the"};
        meaning list : (get lexicon semantics of "the"
            from $$$dictionary$$$) ;
    }
    return $d$;
}
story : return $noun$ {
    new noun $n$ {
        string list : {"story"};
        meaning list : (get lexicon semantics of
            "story" from $$$dictionary$$$) ;
    }
    return $n$;
}
of : return $possession$ {
    new possession $p$ {
        string list : {"of"};
        meaning list : (get lexicon semantics of "of"
            from $$$dictionary$$$) ;
    }
    return $p$;
}
three : return $quantity$ {
    new quantity $q$ {
        string list : {"three"};
        meaning list : (get lexicon semantics of
```

```
            "three" from $$$dictionary$$$) ;
    }
    return $q$;
}
little : return $adjective$ {
    new adjective $a$ {
        string list : {"little"};
        meaning list : (get lexicon semantics of
            "little" from $$$dictionary$$$) ;
    }
    return $a$;
}
pigs : return $noun$ {
    new noun $n$ {
        string list : {"pigs"};
        meaning list : (get lexicon semantics of "pigs"
            from $$$dictionary$$$) ;
    }
    return $n$;
}
dot : return $dot$ {
    new dot $d$ {
        string list : {"dot"};
        meaning list : (get lexicon semantics of "dot"
            from $$$dictionary$$$) ;
    }
    return $d$;
}
[use module]
NLU by NLP : basic structure;
NLU by NLP : grammar category;
```

Please note that in the global variable section, there is a $$dictionary$$$. Firstly, the "global variable"[3] means that it can be accessed from any place of codes. Secondly the $$$dictionary$$$ provides lexicon semantics of each word, which is a list of meanings. And the "meaning" is as a NLP class defined in the following which contains sense and reference of the string.

```
[class] meaning;
[module] NLU by NLP : basic structure;
[data]
string : string ;
sense : string list;
reference : NLU object;
class name : string;
[constructor] new meaning $meaning$ {}
[use module] NLU by NLP : NLU classes;
```

IV. FOURTH STEP : CREATE GRAMMAR RULES AND COMPOSITION SEMANTICSS

```
[codebook] grammar rules;
[module] NLU by NLP : grammar;
[global variable]
dialogue context system : dialogue context system :
    new dialogue context system;
knowledge database : knowledge database: new
  knowledge database;
[function]
$determiner$ $noun$ : return $noun phrase$ {
    let $noun phrase$ be (new noun phrase);
    (string list of $noun phrase$) increases by (string
        list of $determiner$);
    (string list of $noun phrase$) increases by (string
```

```
        list of $noun$);
    return $noun phrase$;
}
$quantity$ $adjective$ $noun$ : return $noun
        phrase$ {
    let $noun phrase$ be (new noun phrase);
    (string list of $noun phrase$) increases by (string
      list of $quantity$);
    (string list of $noun phrase$) increases by (string
        list of $adjective$);
    (string list of $noun phrase$) increases by (string
      list of $noun$);
    return $noun phrase$;
}
$noun phrase 1$ $possession$ $noun phrase 2$ :
return $noun phrase 3$ {
    let $noun phrase 3$ be (new noun phrase);
    (string list of $noun phrase 3$) increases by
          (string list of $noun phrase 1$);
    (string list of $noun phrase 3$) increases by
          (string list of $possession$);
    (string list of $noun phrase 3$) increases by
          (string list of $noun phrase 2$);

    if(((length of (string list of $noun phrase 1$))=2)
        and (((string list of $noun phrase 1$)1)="the")
        and (((string list of $noun phrase
        1$)2)="story")), then {
        new meaning $meaning$ {
            string : (string list of $noun phrase 3$);
            reference : get object by name (get string
                of $noun phrase 2$) from
                $$$knowledge database$$$;
            class name : "NLU story";
        }
        add $meaning$ to (meaning list of $noun
          phrase 3$);
    }
    return $noun phrase 3$;
}
$verb$ $noun phrase$ : return $verb phrase$ {
    let $verb phrase$ be (new verb phrase);
    (string list of $verb phrase$)
      increases by (string list of $verb$);
    (string list of $verb phrase$)
        increases by (string list of $noun
          phrase$);

    let (verb meaning of $verb phrase$) be ((meaning
        list of $verb$)1);
    let (object meaning of $verb phrase$) be
        ((meaning list of $noun phrase$)1);

    return $verb phrase$;
}
$noun phrase$ $tense$ $verb phrase$ : return
      $clause$ {
    let $clause$ be (new clause);
    (string list of $clause$) increases by (string list of
        $noun phrase$);
    (string list of $clause$) increases by (string list of
        $tense$);
    (string list of $clause$) increases by (string list of
        $verb phrase$);

    if((class name of $noun phrase$)="pronoun"),
```

```
        then {
        new meaning $pronoun meaning$ is (get pronoun reference of (get string of $noun phrase$) from $$$dialogue context system$$$);
        let (subject of $clause$) be $pronoun meaning$;
    }
    let (action of $clause$) be (verb meaning of $verb phrase$);
    let (object of $clause$) be (object meaning of $verb phrase$);
    let (tense of $clause$) be ((meaning list of $tense$)1);

    return $clause$;
}
$tense$ $comma$ $clause$ $dot$ : return $sentence$
    {
    let $sentence$ be (new sentence);
    (string list of $sentence$) increases by (string list of $tense$);
    (string list of $sentence$) increases by (string list of $comma$);
    (string list of $sentence$) increases by (string list of $clause$);
    (string list of $sentence$) increases by (string list of $dot$);

    let (subject of $sentence$) be (subject of $clause$);
    let (action of $sentence$) be (action of $clause$);
    let (object of $sentence$) be (object of $clause$);
    add(tense of $clause$) to (tense of $sentence$);
    add((meaning list of $tense$)1) to (tense of $sentence$);

    new NLU status $status$ {
        tense list : tense of $sentence$;
        action : action of $sentence$;
        object : object of $sentence$;
    }

    new NLU person $subject person$ is (<<NLU person>>(reference of (subject of $sentence$)));
    add $status$ to (status list of $subject person$);

    return $sentence$;
}
[use module]
NLU by NLP : grammar category;
NLU by NLP : basic structure;
NLU by NLP : NLU classes;
ennLP : language : meta : basic;
```

Note that these are just the attribute grammar[4] expressed in terms of Natural Language Programming. For the grammar rule

$clause$ -> $noun phrase$ $tense$ $verb phrase$

It is expressed by a function in Natural Language Programming

$noun phrase$ $tense$ $verb phrase$: return $clause$ {...}

And the function body is just the compositional semantics in attribute grammar. This is the key issue of Attribute Grammar[4] by Natural Language Programming. Because other programming languages can not have such a match between grammar syntax and Natural Language Programming functions. And the compositional semantics in attribute grammar is naturally put in the bodies of functions in Natural Language Programming. So, it is easy and natural to write attribute grammar by Natural Language Programming functions.

V. PARSE INPUT SENTENCE BY PARSER

The input and output of a parser(such as CYK algorithm[5], Chart Parser[6]) is in the following.

Input : sentence, lexicon categories, grammar rules
Output : parse tree of the sentence

For example : input is the sentence prepared by step 1

{"next", "comma", "i", "will", "tell", "the", "story", "of", "three", "little", "pigs"}

And the lexicon categories are read from the functions in the lexicon code book "lexicons.codeBook". For example the function

next : return $tense$ {...}

will be read to be : the grammar category of the lexicon "next" is $tense$. i.e. in context free grammar,' it is expressed in the following :

$tense$ -> next

Similarly, the grammar rules are read from the functions in the grammar code book "grammars.codebook". For example, the function

$noun phrase$ $tense$ $verb phrase$: return $clause$ {...}

will be read to be : grammar rule

$clause$ -> $noun phrase$ $tense$ $verb phrase$

So, given the input of sentence, lexicon categories, grammar rules, the parser can parse the input string to be a parse tree. For example

"Next, I will tell the story of three little pigs."

will be parsed to be a parse tree :
"((next) (comma) ((i) (will) ((tell) (((the) (story)) (of) ((three) (little) (pigs))))) (dot))".

VI. INTRODUCTION TO DYNAMICALLY RUN [2]

If we would like to write a calculator that can evaluate any arithmetic expression.

"(23*3+5)/2"

"(5+20)*(2+3)"

"5*(2^3+1)"

"9^(1/2)*(3+2*5)"

Generally speaking, we should write a compiler for the arithmetic expressions. Fortunately, Natural Language Programming provides a function "dynamically evaluate"; it can evaluate any Natural Language Programming function expressions. Because mathematical expressions are also Natural Language Programming expressions, so we can use "dynamically evaluate" to evaluate mathematical expressions inside a string.

The syntax is dynamically evaluate $string$: return $object$

If the string contains a procedure statement, not a function statement, we can use "dynamically run". The following are two examples of "dynamically run".

dynamically run "write a row 66;";

dynamically run "write a row (6+7); write a row(6*7);
";

This is a brief introduction to "dynamically run". In next section we will use it to run the parsed string and their composition semantics.

VII. RUNNING THE COMPOSITIONAL SEMANTICS

From the input "Next, I will tell the story of three little pigs." We can parse the sentence by the grammar rules in section 3 and section 4, and then get the parsed tree in section 5 in the following:

"((next) (comma) ((i) (will) ((tell) (((the) (story)) (of) ((three) (little) (pigs))))) (dot))"

Because each word is a function defined in section 3, it will be run to be the grammar category which is the same as the function return variable class. For example, i : return $pronoun$; which is defined in section 3. And the (the : return $determiner$) (story : return $noun$), the compositional semantics is ($determiner$ $noun$: return $noun phrase$), which is defined in section 5. By these functions defined, the whole sentence can actually be run by running the string by "Dynamically Run" mentioned in [2]. Therefore, the compositional semantics defined in the function bodies will be run at all.

dynamically run "((next) (comma) ((i) (will) ((tell) (((the) (story)) (of) ((three) (little) (pigs))))) (dot))";

Please note that in the Global Variable[3] section of "grammar rules.enCodebook", there are two new items defined :

$$$dialogue context system$$$

$$$knowledge database$$$

The first variable $$$dialogue context system$$$ will resolve "i" to a specific person "Weihan Huang". And this is the semantics of the $noun phrase$ -> $pronoun$ -> "i".

The second variable $$$knowledge database$$$ will help to search the entity of "(((the) (story)) (of) ((three) (little) (pigs)))" by key string "three little pigs" with class name "story".

Lastly, we update the knowledge database by adding a new status "I {next,will} tell the story of three little pigs" in the person "Weihan Huang"'s status list. A

status is defined in the following class.

```
[class] NLU status;
[module] NLU by NLP : NLU classes;
[parent class] NLU object;
[data]
class name : string : "NLU status";
tense list : meaning list : {};
action :    meaning;
object : meaning;
[constructor] new NLU status $NLU status$ {}
[function]
get string of $NLU status$ : return $string$ {
  return ((string of (tense list of $NLU status$))+" "+
  (string of (action of $NLU status$))+" "+
  (string of (object of $NLU status$)));
}
[use module]
NLU by NLP : NLU classes;
NLU by NLP : basic structure;
```

VIII. CONCLUSION

The match of compositional semantics of grammar rules and the Natural Language Programming(NLP) functions

shown above is unique, which means all other programming languages are not able to achieve this! The Definite Clause Grammar(DCG)[7] in programming language Prolog[8] which is also a realizer of attribute grammar, however, uses additional syntax that is not within the Prolog language. In sum, in this paper, I have constructed the grammar categories by NLP classes, and then lexicon syntax and semantics by NLP functions, and lastly the syntax and semantics of compositional semantics of grammar rules by NLP functions. And finally I run the compositional semantics by "dynamically run" in NLP, which updates the personal status in the knowledge database. i.e. "Next, I will tell the story of three little pigs.".

REFERENCES

[1] Natural Language Programming Weihan Huang, “Natural Language Programming in English”, LAP LAMBERT Academic Publishing Book, 2022-08-30

[2] Dynamically run [1] pp151

[3] Global Variable [1] pp142

[4] Attribute Grammar https://en.wikipedia.org/wiki/Attribute_grammar

[5] CYK algorithm https://en.wikipedia.org/wiki/CYK_algorithm

[6] Chart Parser https://en.wikipedia.org/wiki/Chart_parser

[7] Definite Clause Grammar https://en.wikipedia.org/wiki/Definite_clause_grammar

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