iJournals: International Journal of Software & Hardware Research in Engineering
ISSN:2347-4890, ijournals.in/ijshre
Volume 13 Issue 4, April 2025

Natural Language Understanding 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

DOI: 10.26821/IJSHRE.13.04.2025.130401

ABSTRACT

The main purpose of this paper is to introduce an advanced programming technique "dynamically run" in Natural Language Programming. And I will show how this technique is used in natural language understanding. This technique not only greatly simplifies the source codes, but also reduces the machine code loading time complexity initially. Meanwhile, I design a parsing method that scans the sentence once and which uses additional list and stack possibly for the context free structure. To get the syntax and semantics of the sentence, each word is run as a function which is defined inside one file already. And the file of this function is loaded right before it is run dynamically. Lastly, to complete the understanding of natural language sentences, I introduce the concepts of knowledge representation and information flow.

Keywords: Language, Natural, Programming, Understanding

1. INTRODUCTION TO "DYNAMICALLY RUN"[1]

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[2] 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); ";

Because the function call inside the string of

"dynamically run" could use function that has not been loaded from class files or code books, we can use the function "load natural language programming file $string$" to load the file firstly. Examples are in the following.

load natural language programming file "enNLP : public codebooks : math : basic : integer.codeBook";

load natural language programming file "enNLP : public codebooks : math : basic : other.codeBook";

This is a brief introduction to "dynamically run". Later we will use it for natural language understanding to run the function defined for an arbitrary word given.

2. INTRODUCTION TO NATURAL LANGUAGE UNDERSTANDING

2.1 Segment Words in Chinese Sentence

The Chinese sentence example is

"firstlywewillwriteourfirstchineseprogram"

Assume we have a dictionary of words in lexicon order :

"Chinese", "ChineseProgram" "first" "firstly", "our", "program", "we", "will", "write".

Then we can use a recursive function[3] to segment the Chinese sentence. The problem is that there are two possible parses :

"Firstly, we will write our first Chinese program."
"Firstly, we will write our first ChineseProgram."

Here I will choose the segmentation with smaller number of words, i.e. supporting longer words than shorter words, because longer words could have more special intended meanings. So I will choose the second segmentation here :

"Firstly, we will write our first ChineseProgram."

2.2 Parse by Scanning From Left to Right One Time

Instead of using a chart parser[4] or attribute grammar[5], my parse method is to scan from the leftmost of the sentence to the rightmost of the sentence. Since the scan is only one time, lists and stacks[6] are possibly used while parsing the sentence for its context free[7] feature. In my parse method, each word of the sentence is associated with a programming function "word function".

An intuitive approach is to use if-then-else to match the word and run the associated "word function". But think of how many words are in the dictionary, possibly 3000-6000 commonly used words, so the logical tree must be very long. Therefore, reading in the long logical tree if-then-else should require a lot of time initially. And note that most of words are unnecessary to load for the sentence we are reading. So the problem is how to load functions of the words in use only but not those unnecessary words.

My solution is to use "dynamically run" which is mentioned in the first section. And because the "word function" is defined in an independent file (code book file) intentionally, we can firstly load code book file and then run the "word function". Therefore, we can parse the sentence by loading and then running all the "word functions".

for each $word$ in $sentence$, do {

```
    load natural language programming file
                ($word$+" function.code book");
    dynamically run ($word$+" function");
}
```

2.3. Result of the Parsing

The sentence syntactic-semantic structures are parsed to be shown below

input : "Firstly we will write our first ChineseProgram."

output :

Event : we(Subject) write(Verb) ChineseProgram(Object)

Event modifier : firstly(number 1 goal to achieve, Event)

Event modifier : will(merged with firstly)

Subject : we(You and I)

Verb : write(how? unknown in knowledge database)

Object : ChineseProgram(program written in Chinese)

Object modifier : our(possession of we)

Object modifier : first(number 1)

Note that the meaning of the verb "write" is unknown yet. It is missing in the knowledge database and this problem of know-how will be proposed in the information flow. The concepts of knowledge database (knowledge representation) and information flow will be discussed in the following two sections to complete the understanding of natural language sentences.

3. INTRODUCTION TO KNOWLEDGE REPRESENTATION

To understand deeper the words in the natural language sentences like "I" "You" "Write", we may need to query a knowledge database such as

"Who am I?"

"Who are you?"

"How do we write a ChineseProgram?"

So we can actually prepare some facts in the database to answer these questions. For example :

I am Weihan.

You are Little Prince.

Weihan is the author of the book Natural Language Programming.

Little Prince is the reader of the book Natural Language Programming.

Hence these answer the first two questions "Who am I?" and "Who are you?". But the knowledge database currently does not have knowledge about the answer of "How do we write a ChineseProgram?". This problem will result in two issues, firstly we will need to propose this question in the "information flow" of the sentence "Firstly we will write our first program". And secondly, we need to think of a representation for procedural semantics[8] for "how do we write?" And these will be answered in the next section "Information flow".

Moreover, the knowledge database can have the ability of inference to deduce new facts. For example, from the two sentences above "I am Weihan" and “Weihan is the author of the book Natural Language programming", we can infer that "I am the author of the book Natural Language Programming". Similarly we can infer the fact "You are the reader of the book Natural Language Programming”.

To represent the knowledge in the knowledge database uniformly, I will use a directed graph[9] as the knowledge representation.

Firstly, we need to distinguish two types of nodes "unique node" and "separate node". A "unique node" is always referred to the unique one node in all context. Examples of unique nodes are like "I", "You", “Weihan", "Little Prince", "Book of Natural Language Programming". And a "separate node" is a node that is not a unique node. Examples of "separate node" are "author", "reader", "name".

A node has links fanned out and links fanned in. And a link contains a string and one from-node and one to-node.

Example of "I am Weihan" is represented in the following graph :

"I" --be --> “Weihan”
<--be –

Example of "Book of Natural Language Programming" is

“Book -own-> “name” -be-> “Natura
Natural <-belong to- <-be- Language
Language Programming”
Programming”

“Book -own-> “author” -be-> “Weihan”
Natural <-belong to- <-be-
Language -own-> “reader” -be->” Little
Programming” <-belong to- <-be- Prince”

We can read from the left to the right that "The author of Book Natural Language Programming is Weihan.” and read from right to left that “Weihan is the author of Book Natural Language Programming". Lastly for the representation of procedural word like "write", we will talk about it in the next section of Information Flow.

4. INTRODUCTION TO INFORMATION FLOW

A sentence is not independent, its appearance will possibly affect the future reading of a paragraph. So we will say that the sentence carries some information to be transferred toward next reading. For the sentence "Firstly, we will write our first ChineseProgram", let's review the result of the parse :

Event : we(Subject) write(Verb) ChineseProgram(Object)
Event modifier : firstly(number 1 goal to achieve, Event)
Event modifier : will(merged with firstly)
Subject : we(You and I)
Verb : write(how? unknown in knowledge database)
Object : ChineseProgram(program written in Chinese)
Object modifier : our(possession of we)
Object modifier : first(number 1)

We can read from these outputs that there are 2 expectations to fulfill and one question to answer.

1. Event is number 1 goal to achieve
2. Number 1 ChineseProgram will be written.
3. How do we write a ChineseProgram?

These 3 points are just the information flow that the current sentence carries to move on to the future. Point 2 is important because this program will be the first program we will write. Point 3 points out a question that the knowledge database does not have an answer to this question yet.

To fulfill the expectations in Information Flow,

following the sentence, the book just shows a first ChineseProgram to write.

********** first.enProgram **********

write a row "Hello world!";

****

If we have done writing the first program, then the expectations of the information flow will be fulfilled. To do this, we need to know how to write a ChineseProgram, which is the point 3 of the information flow. The procedure of how to write is provided by the author to the reader by the following knowledge representation for procedural semantics.

"How" -type-> "verb" -name-> "write"

“write” -type-> "we write ChineseProgram"
-procedure->{decide the directory name;

decide the program name;
decide the program codes;
edit the codes;
compile the codes;
run the codes;
}

Here each statement of the procedure is corresponding to a Natural Language Programming function defined. So after running these functions the ChineseProgram is written, compiled and run automatically. And this is just the power of information flow.

5. CONCLUSION

I have shown you an advanced programming technique "dynamically run" in Natural Language Programming. And I used it for natural language understanding to run the function defined for an arbitrary word given. This technique not only greatly simplifies the source codes, but also reduces the machine code loading time complexity initially. The result of sentence syntax and semantics is given by running the word function of each word inside the sentence from left to right.

And to complete the understanding of natural language sentences, I introduced the concepts of knowledge representation and information flow. In knowledge representation I gave a uniform directed graph representation for the knowledge system. And for information flow I showed how to fulfill the expectations and answer the question of how to write a ChineseProgram. In the future I look forward to automatically reading the book tutorial and making automation of writing a program. Lastly, I will need to build a run time console, comparing the codes and the output, to learn the cause-effect of the program and its output automatically.

6. REFERENCES

[1] Dynamically run Natural Language Programming in English, LAP LAMBERT Academic Publishing Book, Weihan Huang, 2022-08-30, Chapter 5 Advanced Programming, Section 8 Dynamically Run

[2] Syntax
https://en.wikipedia.org/wiki/Syntax

[3] Recursive function
https://en.wikipedia.org/wiki/Recursion_(computer_science)

[4] Chart parser
https://en.wikipedia.org/wiki/Chart_parser

[5] Attribute grammar
https://en.wikipedia.org/wiki/Attribute_grammar

[6] Stack

https://en.wikipedia.org/wiki/Stack_(abstract_data_type)

[7] Context free

https://en.wikipedia.org/wiki/Context-free_grammar

[8] Semantics

https://en.wikipedia.org/wiki/Semantics

[9] Directed graph

https://en.wikipedia.org/wiki/Directed_graph