International Conference on Advances in Human Machine Interaction (HMI - 2016),
March 03-05, 2016, R. L. Jalappa Institute of Technology, Doddaballapur, Bangalore, India

Semi Natural Language Algorithm to Programming Language Interpreter

Sharvari Nadkarni
I.T. Department,
Sardar Patel Institute of Technology,
Mumbai, India
sharvari.nadkarni03@gmail.com

Parth Panchmatia
I.T. Department,
Sardar Patel Institute of Technology,
Mumbai, India
parthpanchmatia18@gmail.com

Tejas Karwa
I.T. Department,
Sardar Patel Institute of Technology,
Mumbai, India
tejaskarwa17@gmail.com

Prof. Swapnali Kurhade
I.T. Department,
Sardar Patel Institute of Technology,
Mumbai, India
swapnali.kurhade@spit.ac.in

Abstract – The conversion of an algorithm to code is still at an early stage. Effective conversion of algorithms mentioned in natural English language to code will enable programmers to focus on logic building and free them of syntactical worries, further it will also aid the visually impaired programmers. Although beneficial, implementation of such a converter encounters numerous challenges like limitations imposed due to semantics of the English language, case frames, etc. In this paper we have introduced an interpreter that is capable of converting algorithms in English to C code whose flexibility of interpretation has been enhanced by using synonyms and by the introduction of a personalised training model whose concept has been outlined below. We have defined the conceptual model along with a user scenario which demonstrates the functioning of our model.

Keywords – Case frames, NLP interpreter, Natural Language Processing, Personalized, Trigger words

I. INTRODUCTION

Today programming has become omnipresent; it is used in a wide variety of domains like astronomy, industrial automation, financial analysis, microbiology, etc. [1]. Solutions offered by programming are highly efficient and Information Technology has been playing a pivotal role in the rapid growth and transformation of today. Algorithms form the fundamental blocks of programming and are core to solution designing. Implementation of these algorithms using programming languages serves as a major hurdle.

People may possess good logical skills along with great algorithmic solution designing capabilities but the inadequate knowledge of programming languages makes them handicapped. Neophyte programmers may find it difficult to learn general programming skills and syntactical skills simultaneously. Visually impaired developers suffer as they spend double the time in eradicating syntactical errors as compared to any programmer with a normal vision. Thus there is a need to develop software that is capable of converting algorithms written in natural language to a programming language. This software will allow a person to focus on problem solving and free him from syntactical worries. Although the realization of such software may be very beneficial and capable enough to transform the industrial standards of development, its realization involves various challenges.

The rest of the paper is organized as follows: Section II discusses the literature survey; Section III highlights the challenges faced implementing natural language to code interpreter; Section IV puts forth the conceptual model; Section V gives a concluding remark and outlines the future scope.

II. LITERATURE SURVEY

Natural Language Processing (NLP) and Programming Languages share a mutual concept – "languages" and although both are well recognized domains in the field of Computer Science there has still been limited interaction between them [3, 4]. Certain proposals have been made to develop an interpreter that can generate a code in a programming language from the corresponding algorithmic specification but each of these have certain limitations. An implementation of such a translator, called ALGOSmart, translates pseudo code specification written in XML into C and Java [1]. But the aforementioned translator imposes an additional overhead on users by making it mandatory for them to have knowledge about a set of predefined tags and their correct usage. Additionally interpreters have been suggested that perform scanning, parsing, semantic handling, command execution and higher level processing by utilizing templates and a fixed structure and style of writing. Such interpreters have left users frustrated because of the difficulty encountered to adapt to the given set of constraints and rejected input sentences [5]. Also proposals have been made that have focused on the facets related to procedural programming and on observing the corpus of English descriptions have developed programming semantics, techniques that map linguistic constructs onto program structures [3]. There have been other proposals which have provided a natural-language based user interface which allows a user to input algorithms in natural English language and it generates the corresponding Java code. The prototypic implementation of the proposal mentioned above, called NaturalJava, contained three main modules as shown in Figure 1. PRISM allowed the user to input an algorithmic statement in English language which was in turn passed to Sundance. Sundance then generated the corresponding case frames which were analysed by PRISM by calling TreeFace. After each operation TreeFace generated the source code which was then presented to the user by PRISM [2].

978-1-4673-8810-8/16/$31.00 ©2016 IEEE

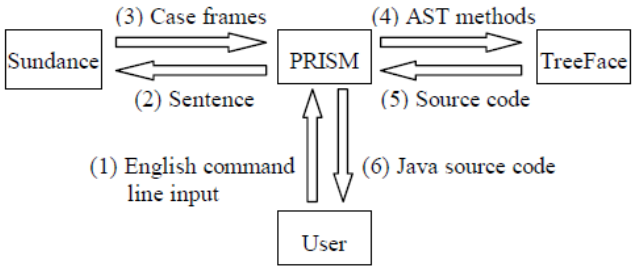

Figure 1. Architecture of NaturalJava [2].

The major limitation in the aforementioned model is imposed by the finite set of case frames and the ability to comprehend natural language input. We will deal will these challenges in detail in the next section.

III. CHALLENGES

Few attempts have been made before to interpret natural language algorithm into formal code. Most challenges revolve around the following aspects

- While it is easy to tag individual words using part of speech, semantics of the algorithm as a whole becomes difficult to interpret and process.
- Every programming language has its own speciality. Various programming language aspects become challenging to incorporate to be identified and interpreted by natural language processing.
- Flexibility of identifying and interpreting natural language is limited. This is because every individual has a different method of expressing a single idea.

NLP for NLP uses the concept of mapping an active verb to a programming action. [3] The NaturalJava system uses case frames. [1] This means, that if a particular set of words is not present in the algorithm line, a trigger to activate parsing and interpretation will not take place. In an attempt to tackle the aforementioned challenge, we are using a synonym finder to increase the vocabulary repertoire of the system. Also, users can input their individualistic writing style into the system. This input can be used to generate a personalized training model for every user to make natural language interpretation more flexible. The next section discusses this solution in detail.

Challenges
• Semantics • Programming language dependency • Flexibility

IV. CONCEPTUAL MODEL

In order to address the aforementioned challenge of flexibility, we have proposed a model consisting of an interpreter and related interacting modules. The system accepts an algorithm from the user. Basic Natural Language Processing is applied on the algorithm, sentence by sentence. Post this, the processed output is sent to the interpreter. At the interpreter module, first the statement type is identified and accordingly, it is parsed into formal C code. The code is hence forwarded from the interpreter module and displayed to the user. Hence, the conceptual model consists of four modules interacting with each other to accept an algorithm in natural language and interpret it

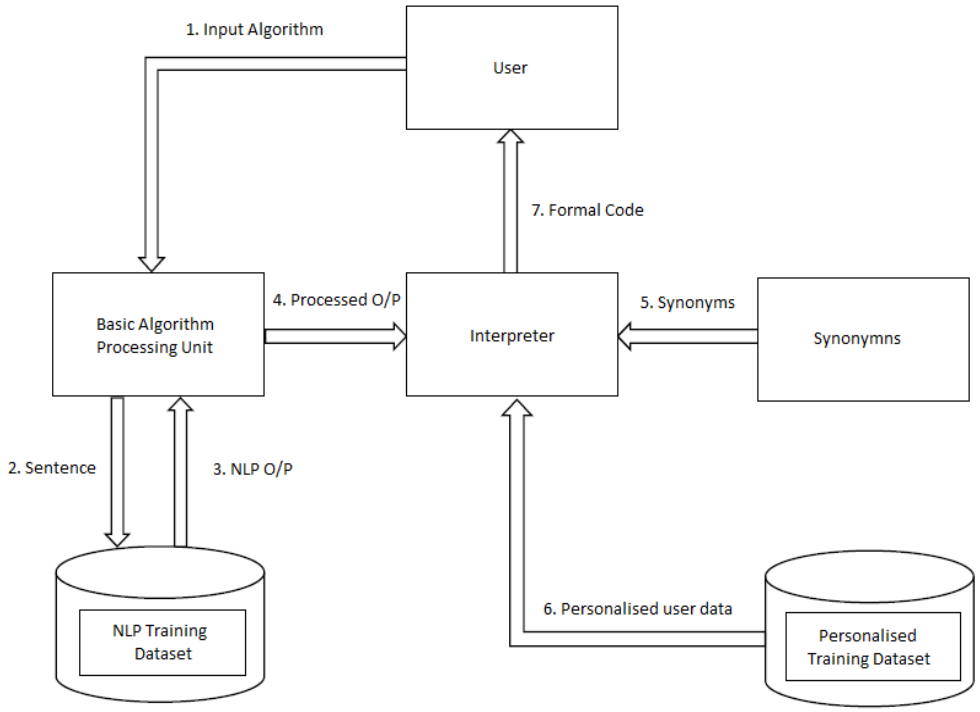

Figure 2. Conceptual model of the Semi Natural Algorithm to Programming Language Interpreter

978-1-4673-8810-8/16/$31.00 ©2016 IEEE

in formal language. The model is shown in Figure 2.
The modules are:

- User
- Basic Algorithm Processing
- Interpreter
- Synonyms
- Personalised Training Model

1. User module:

This module signifies the end user. Via a web based application, an algorithm is accepted into the system. The algorithm is processed by the other modules and a formal C language code is returned to the user.

2. Basic Algorithm Processing module:

After accepting the algorithm from the user, basic natural language processing is applied line wise. Lines and words are separated and Part Of Speech tagging is applied to the algorithm. This module sets stage for interpretation.
Consider statement- initialize integer i to 5
The output of this statement after applying basic algorithm processing would be - initialize_NN integer_NN i_NN to_TO 5_CD, where NN is noun, TO is to and CD is Cardinal Number.

3. Interpreter module:

This module is the crux of the model. The interpreter works in two stages.

- Type identification:

The input sentence is identified as declaration, initialization, input, conditional, looping etc. statement. This is done by identifying a trigger word in the statement. The interpreter contains case frames that map a trigger words to a statement type. Consider the statement - initialize integer i to 5, the interpreter first looks for part of speech tags or keywords. Thus, initialise would be recognised as a keyword and the statement is forwarded to initialise module for parsing statement to code.

- Parsing into formal C code

Once the statement is correctly identified, it is sent to the Type specific module for parsing. Here, using POS tags and the sentence structure, the algorithmic line is converted to formal code.
Here, the key is to identify and address various styles of writing algorithms and correctly parsing them. For this, we implement a personalised training model that learns style of writing algorithms, therefore improving flexibility of writing and accuracy of interpretation. Thus statement - initialize integer i to 5, is interpreted to form - int i=5.

4. Synonyms:

To increase the flexibility of identifying a trigger for the interpreter module, the repertoire of trigger words is increased by not only feeding words manually but by using Synonyms as well. A large set of words increases the probability of a statement being correctly identified and parsed.

5. Personalised Training Model

Another powerful method to increase flexibility is by using a personalised training model. Users would be asked to input natural language statements for expressing their individualistic writing style. This style would be gauged and adapted to by the system. Thus, the next time the user would type a similar statement; it would be easily recognised and parsed by the system. This module is under implementation. This module would increase accuracy of interpreting algorithm to code by a great deal.

V. USER SCENARIO

The user will be provided with a front end interface wherein he can enter the algorithm in natural English language. An example input algorithm - to determine whether the given number is odd or even, is shown in the figure 3. This figure represents the input algorithm to our system which is step 1 in the conceptual model.

```
define integer rem
input an integer n
rem = n mod 2
if rem equal to 0 then
display 'Given number is even'
end if
else
display 'Given number is odd'
end else
```

Figure 3. Input Algorithm

This input will then be sent line by line to the NLP module which will carry out POS tagging and generate the corresponding output as shown in figure 4.

```
define_VB integer_NN rem_NN
input_NN an_DT integer_NN n_NN
rem_NN =_SYM n_NN mod_NN 2_CD
if_IN rem_NN equal_JJ to_TO 0_CD
then_RB
display_NN 'Given_JJ number_NN
is_VBZ even'_RB
end_NN if_IN
else_RB
display_NN 'Given_JJ number_NN
is_VBZ odd'_JJ
end_NN else_RB
```

Figure 4 POS Tagged Algorithm

The POS Tagged Algorithm will be generated line by line and each line will be subjected to interpretation by making

use of synonyms and dataset from the personalized model. The corresponding C code will be generated line by line and then it will be merged to produce the final formal code as shown in figure 5.

```
int rem;
int n;
scanf("%d", &n);
rem = n%2;
if( rem==0 ){
printf("Given number is even");
}
else {
printf("Given number is odd");
}
```

Figure 5. Formal C Code

VI. CONCLUSION AND FUTURE SCOPE

This paper explains how flexibility of a system performing algorithm to code interpretation maybe improved. The system consists of User, Basic Algorithm Processing, Interpreter, Synonyms and Personalised Training Dataset modules which interact to form formal code. Synonyms and Personalised Training Dataset are the two main modules adding flexibility to the system.

We have opened promising results using our current model and we plan to extend it and incorporate functions, arrays, declarations and pointers. This part can be covered by creating further modules with associated triggers and logic for the same. Further, we aim to overcome the challenge related to semantics as part of our future scope.

References

[1] Suvam Mukherjee and Tamal Chakrabarti, "Automatic Algorithm Specification to Source Code Translation", in *Indian Journal of Computer Science and Engineering, 2011 on,* Vol. 2, No. 2, pp. 146 – 159, April – May 2011.

[2] David Price, Ellen Riloff, Joseph Zachary and Brandon Harvey, "NaturalJava: A Natural Language Interface for Programming in Java", in *the Proceedings of the 2000 ACM on Intelligent User Interfaces Conference,* pp. 207 – 211, January 2000.

[3] Rada Mihalcea, Hugo Liu, Henry Lieberman, NLP (Natural Language Processing) for NLP (Natural Language Programming), in the 7th International Conference on Computational Linguistics and Intelligent Text Processing, LNCS, Mexico City, February 2006

[4] BALLARD, B., AND BIERMAN, A, "Programming in natural language: NLC as a prototype", In Proceedings of the 1979 annual conference of ACM/CSC-ER (1979).

[5] Biermann, A., Ballard, B., and Sigmon, A, "An Experimental Study of Natural Language Programming", International Journal of Man-Machine Studies, Vol. 18, pp. 71-87, 1983.

978-1-4673-8810-8/16/$31.00 ©2016 IEEE