A Natural Language based Programming Engine

Submitted in partial fulfillment of the requirements

of the degree of

Bachelor of Engineering

by

Kaushal Bhogale (18)

Adheesh Juvekar (43)

Asutosh Padhi (56)

under the guidance of

Mrs. Smita Jangale

Department of Information Technology

Vivekanand Education Society's Institute of Technology

2018-19

Vivekanand Education Society's

Institute of Technology

Since 1962

(Affiliated to University of Mumbai, Approved by AICTE & Recognized by Govt. of Maharashtra)

Department of Information Technology

CERTIFICATE

This is to certify that **Mr. Kaushal Bhogale, Mr. Adheesh Juvekar, Mr. Asutosh Padhi** of Fourth Year Information Technology studying under the University of Mumbai have satisfactorily presented the project entitled **A Natural Language based Programming Engine** as a part of the PROJECT-I for Semester-VII under the guidance of **Mrs. Smita Jangale** in the year 2018-2019.

Date: 24/10/18

(Name and sign)	(Name and sign)	(Name and sign)
Head of Department	Supervisor/Guide	External

Abstract

It can be argued that it is better to program in a natural language such as English, instead of high-level programming languages like Python, C, etc. The reason for this is programming languages are tough, especially for beginners and non-programmers. A natural language interface for programming should result in greater readability, as well as making possible a more intuitive way of writing code. This application is aimed to convert natural logical statements into language independent code segments.

Keywords: Natural Language Processing, Programming Environment, Natural Language Interface

Table of Contents

Chapter 1

Introduction

"I love computer languages. In fact, I've spent roughly half my life nurturing one particular very rich computer language: Mathematica. But do we really need computer languages to tell our computers what to do? Why can't we just use natural human languages, like English, instead?"

- Stephen Wolfram, creator of Mathematica.

At present, programming languages are used to feed-in instructions to computers. This requires users to be educated in writing programs using these languages to make software or applications. This limits the endeavor of writing software to a specific community of programmers. The question remains, however, whether such enigmatic languages are needed in the first place. One answer as to why programming languages are ubiquitous is that they have structured, unambiguous syntaxes which make them an extremely efficient way to communicate with the computer. But this makes programming intriguing and boring, especially for beginners and entrepreneurs. This application aims to tackle this particular issue by developing a natural language interface to communicate with computers, which tries to simplify programming in general.

1.1 Problem Statement

There is a need to convert natural languages statements to code segments, for non-programmers and beginners, to code intuitively and naturally without going through the hassle of learning programming languages. The aim is to develop an application that does this without being dependent on a programming language or the user's pre-existing knowledge about programming. Users should be able to speak out instructions to the computer in natural languages like English. These instructions will be transformed into a high-level programming language like Python. Instructions can be spoken in numerous ways, even if all sentences imply an equivalent piece of logic.

1.2 Proposed System

The conversion of English sentences into a high-level language instead of executable code seems more preferable. By doing so, the job of compiling and creating the executable is left to the compiler.

The system is called a "Programming Engine". It is not a programming language. It allows free flow language, does not follow any fixed syntaxes, and is adaptive to the user. "Programming Engine" can be defined as follows - A **Programming Engine** is a software development environment designed to develop programs using natural language interface. Internally, the Programming Engine consists of several individual components(phases) through which the natural language sentences pass before generating the required output.

The Programming Engine works as follows. The system will provide a graphical interface to take speech input from the user. The natural language sentence is first tokenized into words. The tokenized words will be tagged by Part-of-Speech, for example, Noun, Verb, Pronoun, etc. The next phase involves finding the relationships between words and parts of sentences and form a dependency tree. Once the dependency tree is formed, the next phase involves parsing the tree to draw meaning out of the words. This can be done using a simple lookup, a pre-existing knowledge graph, or the rules can be learnt as well. The aim is to try all the three approaches and compare their results. The last phase is the code generating phase. When the interpretation of the sentence is singled out, the output code can be generated from a set of templates that would be fed into the system. As, every single phase is independent of each other, generating output code for a different programming language simple means that only the last phase needs to be changed. Rest all phases remain unaffected.

1.3 Objectives

- ❖ To create a programming engine that transforms natural language sentences into code segments.
- ❖ To test and analyze three different approaches to solve the problem.
- ❖ To design an uncomplicated user interface for students and entrepreneurs.
- ❖ To gather details on how non-programmers structure codes and give programming instructions.
- ❖ To test its effectiveness against all structures of programming(atomic statements, loops, conditional statements, etc).

Chapter 2

Literature Survey

2.1 Research Papers

1. **NLP (Natural Language Processing) for NLP (Natural Language Programming)**
    a. Abstract

    Natural Language Processing holds great promise for making computer interfaces that are easier to use for people since people will (hopefully) be able to talk to the computer in their own language, rather than learn a specialized language of computer commands. For programming, however, the necessity of a formal programming language for communicating with a computer has always been taken for granted. We would like to challenge this assumption. Modern Natural Language Processing techniques can make possible the use of natural language to (at least partially) express programming ideas, thus drastically increasing the accessibility of programming to non-expert users. To demonstrate the feasibility of Natural Language Programming, this paper tackles what are perceived to be some of the hardest cases: steps and loops. A corpus of English descriptions used as programming assignments, and develop some techniques for mapping linguistic constructs onto program structures, which are referred to as programmatic semantics.

b. Conclusion

One of the potential applications of such a natural language programming system is to assist those who begin learning how to program, by providing them with a skeleton of computer programs as required in programming assignments. Inspired by these applications, a collection a corpus of homework assignments as given in introductory programming classes is made, and attempt to automatically generate computer program skeletons for these programming assignments.

The corpus is collected using a Web crawler that searches the Web for pages containing the keywords programming and examples, and one of the keyphrases write a program, write an applet, create a program, create an applet. The result of the search process is a set of Web pages likely to include programming assignments. Next, in a post-processing phase, the Web pages are cleaned-up of HTML tags, and paragraphs containing the search keyphrases are selected as potential descriptions of programming problems. Finally, the resulting set is manually verified and any remaining noisy entries are thusly removed.

2. Programming With Unrestricted Natural Language

a. Abstract

We argue it is better to program in a natural language such as English, instead of a programming language like Java. A natural language interface for programming should result in greater readability, as well as making possible a more intuitive way of writing code. In contrast to previous controlled language systems, we allow unrestricted syntax, using wide-coverage syntactic and semantic methods to extract information from the user's instructions. We also look at how people actually give programming instructions in English, collecting and annotating a corpus of such statements. We identify differences between sentences in this corpus and in typical newspaper text, and the effect they have on how we process the natural language input. Finally, we demonstrate a prototype system, that is capable of translating some English instructions into executable code.

b. Conclusion

Programming is a very complicated task, and anyway in which it can be simplified will be of great benefit. The system we have outlined and prototyped aims to allow a user to describe their instructions in a natural language. For this, a user may be asked to clarify or rephrase a number of points, but will not have to correct syntax errors as when using a normal programming language. Using modern parsing techniques, and a better understanding of just how programmers would write English code, we have built a prototype that is capable of translating natural language input to working code. More complicated sentences that describe typical programming structures, such as if statements and loops are also understood. Indeed, much of the work to be done involves increasing the coverage of the system in a general manner, so that it is able to understand a wider variety of user input. Once we have built a complete system that can make some understanding of almost any input, we expect it to be usable by novice and experienced programmers alike.

3. Learning to Transform Natural to Formal Languages

a. Abstract

This paper presents a method for inducing transformation rules that map natural-language sentences into a formal query or command language. The approach assumes a formal grammar for the target representation language and learns transformation rules that exploit the non-terminal symbols in this grammar. The learned transformation rules incrementally map a natural-language sentence or its syntactic parse tree into a parse-tree for the target formal language. Experimental results are presented for two corpora, one which maps English instructions into an existing formal coaching language for simulated RoboCup soccer agents, and another which maps English U.S.-geography questions into a database query language. We show that our method performs overall better and faster than previous approaches in both domains.

b. Conclusion

Compared to previous ILP-based methods, SILT allows for a more global approach to semantic parsing that is not constrained to making correct local decisions after incrementally processing each word. However, it still lacks some of the robustness of statistical parsing. A more recent approach by Ge & Mooney (2005) adds detailed semantics to a state-of-the-art statistical parser. Their system learns a statistical parser that generates a semantically augmented parse tree, in which each internal node is given both a syntactic and a semantic label. By integrating syntactic and semantic interpretation into a single statistical model and finding the globally most probable parse, an accurate combined syntactic/semantic analysis can be obtained. However, this approach requires semantically augmented parse trees as additional training input.

4. Feasibility Studies For Programming In Natural Language

a. Abstract

We think it is time to take another look at an old dream that one could program a computer by speaking to it in natural language. Programming in natural language might seem impossible, because it would appear to require complete natural language understanding and dealing with the vagueness of human descriptions of programs. But we think that several developments might now make programming in natural language feasible. First, improved broad coverage natural language parsers and semantic extraction techniques permit partial understanding. Second, mixed-initiative dialogues can be used for meaning disambiguation. And finally, where direct understanding techniques fail, we hope to fall back on Programming by Example, and other techniques for specifying the program in a more fail soft manner. To assess the feasibility of this project, as a first step, we are studying how non-programming users describe programs in unconstrained natural language. We are exploring how to design dialogs that help the user make precise their intentions for the program while constraining them as little as possible.

b. Conclusion

Programming directly in natural language, without the need for a formal programming language, has long been a dream of computer science. Even COBOL, one of the very early programming languages, and for a long time, the dominant business programming language, was designed to look as close as possible to natural language to enhance readability. Since then, very few have explored the possibility that natural language programming could be made to work.

In this paper, we have proposed an approach based on advances in natural language parsing technology, mixed-initiative dialog, and programming by example. To assess the feasibility of such an approach we have analyzed dialogs taken from experiments where non-programmer users were asked to describe tasks, and it seems that many of the important features of these dialogs can be handled by this approach. We look forward to the day when computers will do what we say, if only we ask them nicely.

5. On the nature of natural language programming: generating transformation rules

a. Abstract

In this paper we address the basic difference between the text written in a natural language and the text written in a programming language, the “natural programming language.” We ponder the rules of transforming a human-directed text written in a natural programming language into a computer-directed text ready for the execution by a computer. We offer examples for the use of such rules and attempt to arrive at some generalizations.

b. Conclusion

In this paper we pondered the notion of the natural language programming and required transformations. We posit that the next step is to create a program that will automatically perform transformations we discussed, and to test this on a variety of programs and programmers. The objective of such a testing will be to discover the following: are the rules of transformation sufficient in all cases; can these be expanded; or is there no way to find a universal and limited set of transformation rules. Ultimately we would like to know if the notion of a natural language programming is just a dream and will remain only an unfulfilled dream.

6. Natural language programming: Styles, strategies, and contrasts

a. Abstract

Our objective in this study was to obtain detailed empirical information about the nature of natural language "programming" to bring to bear on the issues of increasing the usability of computer language interfaces. Although we expected numerous difficulties to be detected concerning the potential of actually implementing a system to interpret natural language programs, we were not prepared for the magnitude of what we see as being the three major obstacles: style, semantics, and world knowledge. Concerning the first, there is little way in which the vast differences in styles could be increased: programming-language style is simply alien to natural specification. With respect to semantics, we also were unprepared to find out the extent to which the selection of the appropriate "meaning" (to a word, phrase, or sentence) is dependent upon the immediate and prior context. And as for world knowledge, we suspect that the extent to which shared experiences and knowledge are critical to procedural communication and understanding among people has barely been hinted by our present data.

b. Conclusion

Our objective in this study was to obtain detailed empirical information about the nature of natural language "programming" to bring to bear on the issues of increasing the usability of computer language interfaces. Although we expected numerous difficulties to be detected concerning the potential of actually implementing a system to interpret natural language programs, we were not prepared for the magnitude of what we see as being the three major obstacles: style, semantics, and world knowledge. Concerning the first, there is little way in which the vast differences in styles could be increased: programming-language style is simply alien to natural specification. With respect to semantics, we also were unprepared to find out the extent to which the selection of the appropriate "meaning" (to a word, phrase, or sentence) is dependent upon the immediate and prior context. And as for world knowledge, we suspect that the extent to which shared experiences and knowledge are critical to procedural communication and understanding among people has barely been hinted at by our present data.

7. **Ain't Nobody Got Time For Coding : Structure-aware Program Synthesis From Natural Language**

    a. Abstract

Program synthesis from natural language (NL) is practical for humans and, once technically feasible, would significantly facilitate software development and revolutionize end-user programming. We present SAPS, an end-to-end neural network capable of mapping relatively complex, multi-sentence NL specifications to snippets of executable code. The proposed architecture relies exclusively on neural components, and is built upon a tree2tree autoencoder trained on abstract syntax trees, combined with a pretrained word embedding and a bi-directional multi-layer LSTM for NL processing. The decoder features a doubly-recurrent LSTM with a novel signal propagation scheme and soft attention mechanism. When applied to a large dataset of problems proposed in a previous study, SAPS

performs on par with or better than the method proposed there, producing correct programs in over 90% of cases. In contrast to other methods, it does not involve any non-neural components to post-process the resulting programs, and uses a fixed-dimensional latent representation as the only link between the NL analyzer and source code generator.

b. Conclusion

We have shown that end-to-end synthesis of nontrivial programs from natural language is possible with purely neural means, and using exclusively gradient descent as the learning mechanism. Given that (i) capturing the semantics of the NL is in general difficult, (ii) there are deep differences between NL and AST trees on both syntactic and semantic level, and (iii) the correspondence between the parts of the former and the latter is far from trivial, we find this result yet another promising demonstration of the power dwelling in the neural paradigm. In future works, we plan to perform ablation studies on SAPS to determine which components are critical for its performance and focus on the compositionality of both specifications and program code.

2.2 Articles

[1] The domain of our project; Natural language programming - *https://en.wikipedia.org/wiki/Natural-language_programming*

[2.1] Wolfram alpha blogs (On the future of natural language programming) - *http://blog.wolfram.com/2010/11/16/programming-with-natural-language-is-actually-going-to-work/*

[2.2] Wolfram alpha blogs (How wolfram alpha was built) - *http://blog.wolframalpha.com/2009/05/01/the-secret-behind-the-computational-engine-in-wolframalpha/*

[3] Natural language processing for Natural language Programming - *http://web.eecs.umich.edu/~mihalcea/papers/mihalcea.cicling06a.pdf*

[4] Program synthesis using natural language
https://arxiv.org/pdf/1509.00413.pdf

[5] Learning to Mine Aligned Code and Natural Language Pairs from Stack Overflow - *https://arxiv.org/pdf/1805.08949.pdf*

[6] Natural Language Processing and Program Analysis for Supporting Todo Comments as Software Evolves -
http://www.cs.utexas.edu/~ml/papers/nie.nlse18.pdf

[7] Dijkstra's comments on why Natural language Programming is not possible -
1. *https://www.cs.utexas.edu/users/EWD/transcriptions/EWD06xx/EWD667.html*
2. *https://www.quora.com/Is-it-possible-to-code-programs-with-natural-language-If-it-is-when-will-it-be-possible*

[8] This post speaks about something similar to building websites using natural language -
https://stackoverflow.com/questions/2750931/is-it-better-to-use-a-natural-language-to-write-code

[9] David Price, Ellen Rilofff, Joseph Zachary, Brandon Harvey, NaturalJava: a natural language interface for programming in Java, Proceedings of the 5th international conference on Intelligent user interfaces, p.207-211, January 09-12, 2000, New Orleans, Louisiana, USA

[10] A Patent by Stephan Wolfram
https://patentimages.storage.googleapis.com/53/3a/d3/c46eb673beaa4f/US20180121173A1.pdf

[11] "Natural language is a programming language: Applying natural language processing to software development" by Michael D. Ernst. In SNAPL 2017: the 2nd Summit on Advances in Programming Languages, (Asilomar, CA, USA), May 2017, pp. 4:1-4:14.

Chapter 3

Requirements and Analysis

3.1 Functional requirements

- ❖ The system should convert the logical natural statements to programming language segments (speech to text)
- ❖ The code should be visible to the user.
- ❖ The functionality of the code should be descriptive in natural English for the user to understand what the code actually means
- ❖ Various modes should be provided for the user to either insert code segments, update portions of code(cut, copy, paste) and restructure the code.

3.2 Non-functional requirements

- ❖ The code should be converted in real time.
- ❖ There should be absolutely no difference in what the user intended and the converted code.
- ❖ The user should be given the same freedom a coder in a regular programming platform has.

3.3 Constraints

Constraints can be divided into two parts:

- Hard constraints
- Soft constraints

3.3.1 Hard Constraints

Following are the constraints which needs to be fulfilled necessarily.

- A stable internet connection for real-time speech recognition.
- Sentences must be spoken as instructions, i.e. in present tense and in third person.
- Sentences should be grammatically correct.
- The generated code should have human-readable comments.
- The generated code should have syntax highlighting.

3.3.1 Soft Constraints

Following are the constraints that should be satisfied as much as possible but not at the expense of hard constraints. It is generally considered good enough if large numbers of them are taken care of.

- Availability of noise-canceling microphones result in better speech recognition.
- The environment should not be too noisy.
- Awareness of writing logical statements and algorithms.
- Sentences are preferred to be explicit in meaning so as to get the desired output.

3.4 Software requirements for Server

1. Operating system: Any Debian-based Linux OS
2. Compilers/Interpreters: GNU GCC, Python
3. Python packages: Spacy, NLTK, AST
4. Web Frameworks: Flask, SocketIO

3.5 Software requirements for User

1. Operating System: Windows/Linux/MacOS
2. Browser: Google Chrome
3. Audio Drivers

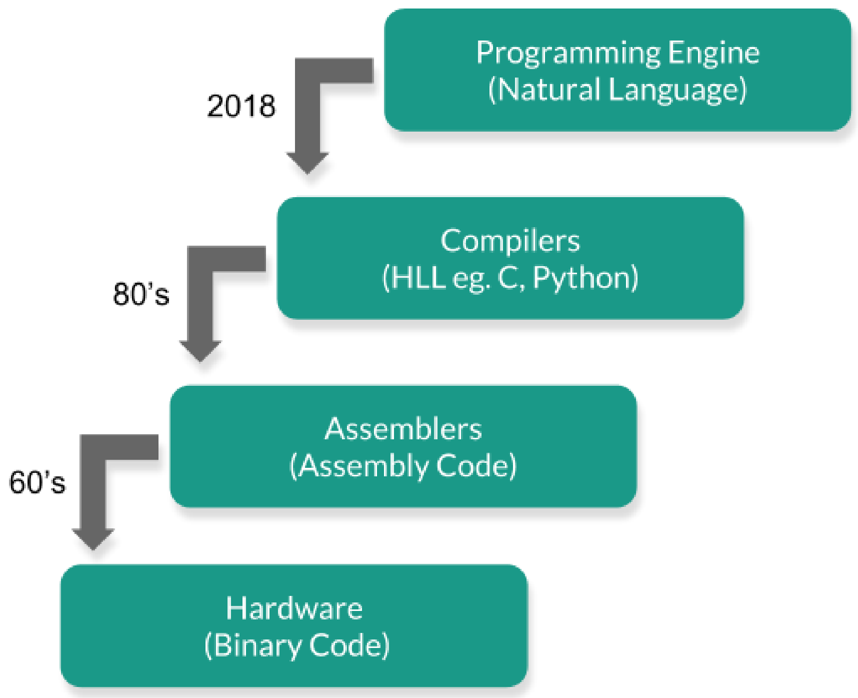
3.6 Hardware requirements for User

1. Microphone
2. Output audio device

Chapter 4

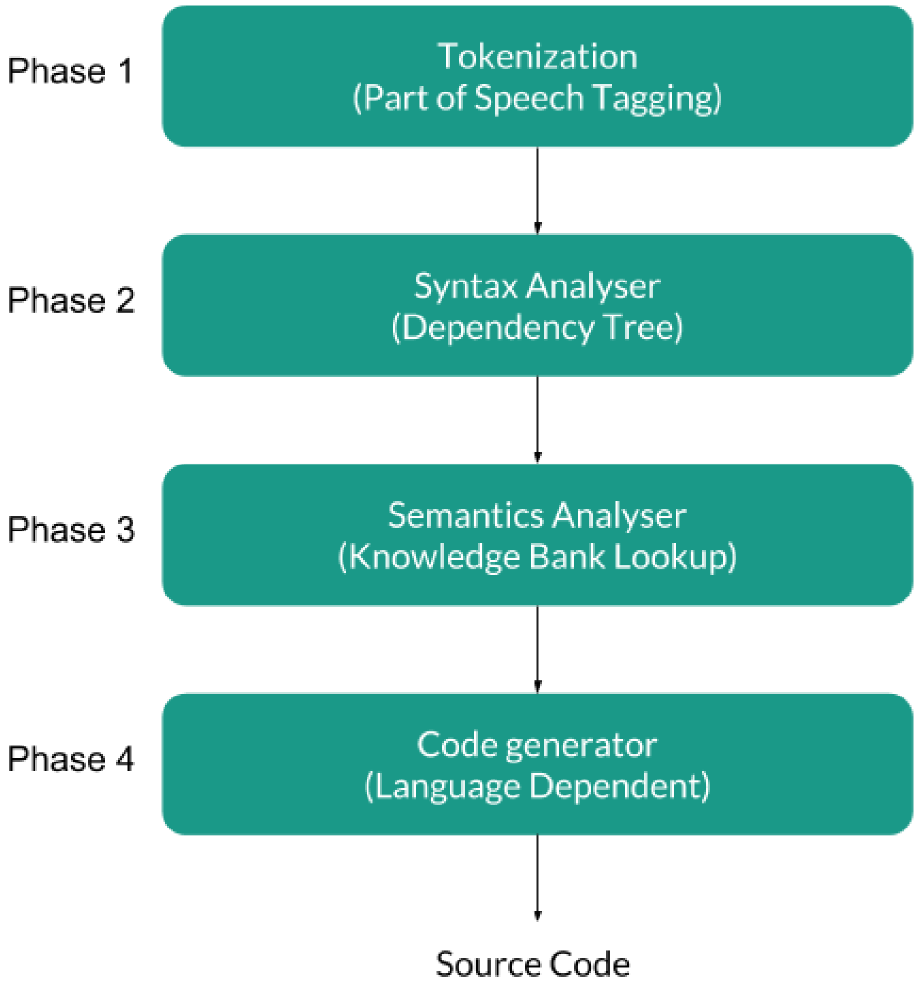
Proposed Design

4.1 Architecture



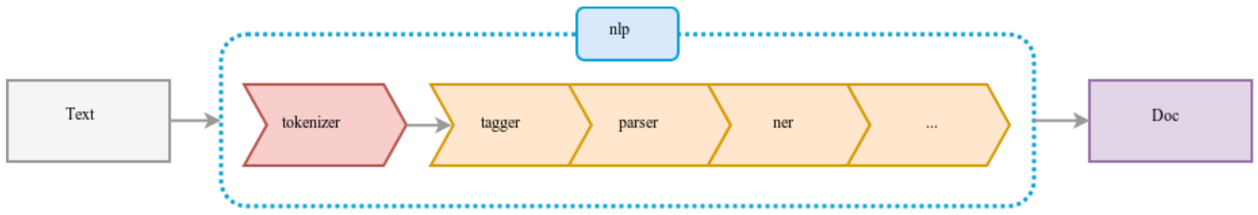
Programming Engine (Natural Language)
2018
Compilers (HLL eg. C, Python)
80's
Assemblers (Assembly Code)
60's
Hardware (Binary Code)

4.2 Phases of the Programming Engine

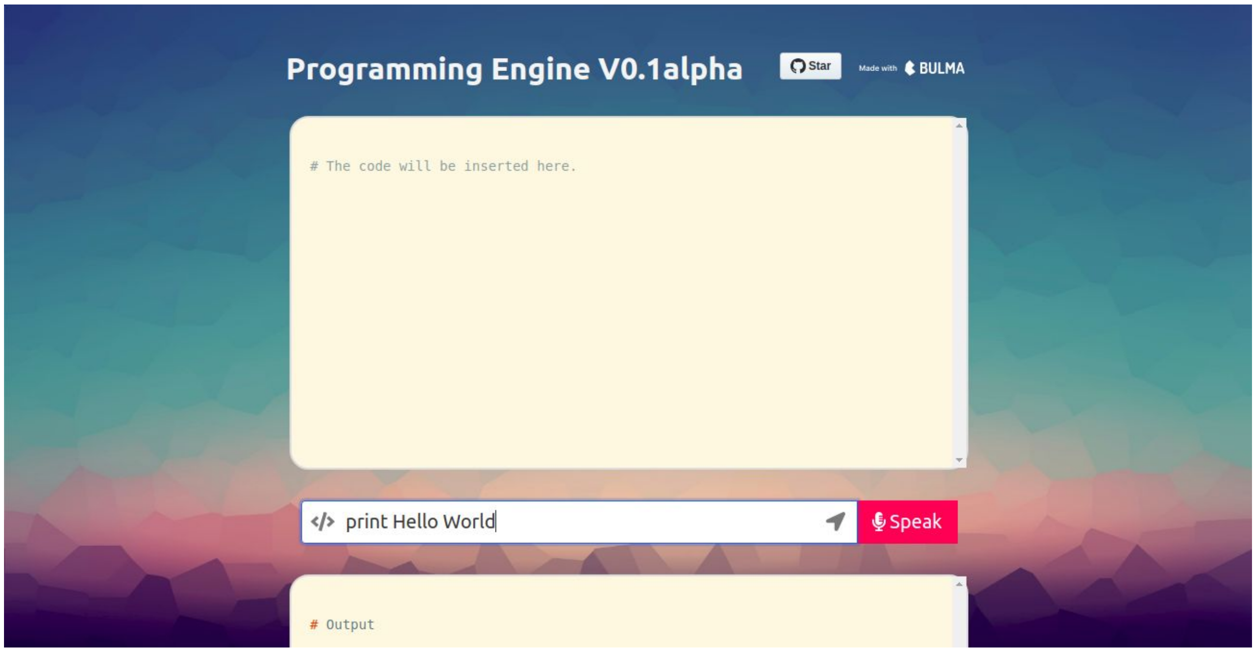


4.3 Pipeline for Syntax tree generation

The following image shows the steps to be taken to convert the natural language sentence into a document, which is used later to convert into source code.



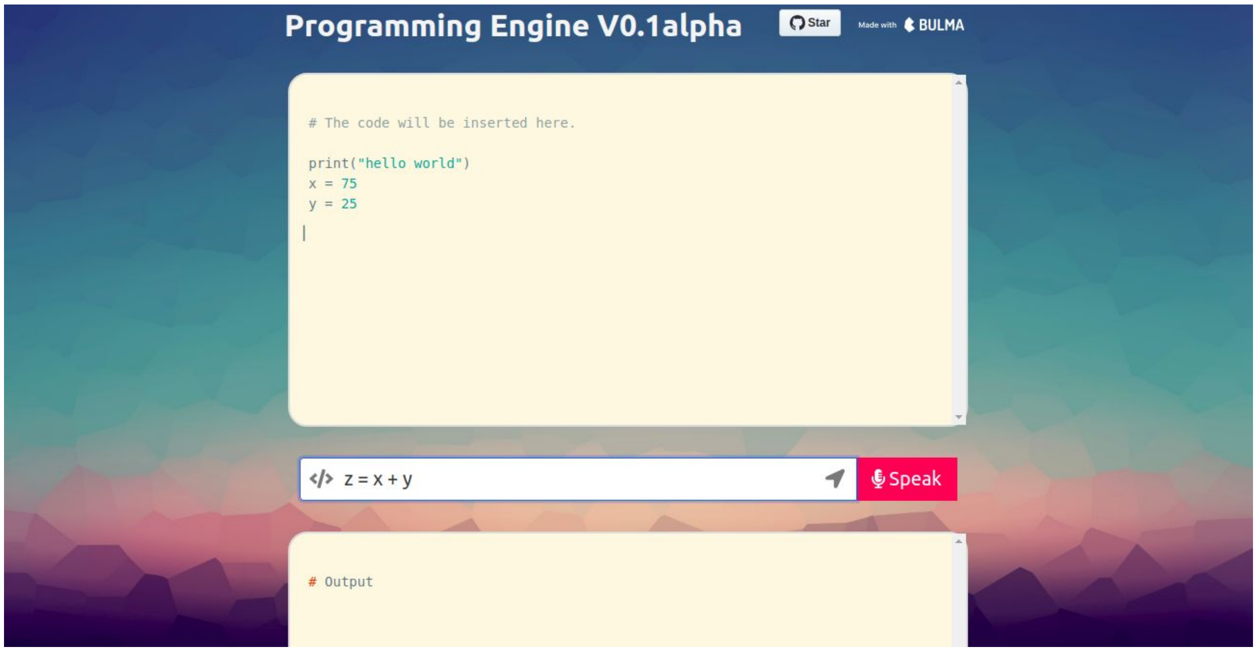
4.4 Prototype Snippets



```
# The code will be inserted here.
```

print Hello World

```
# Output
```

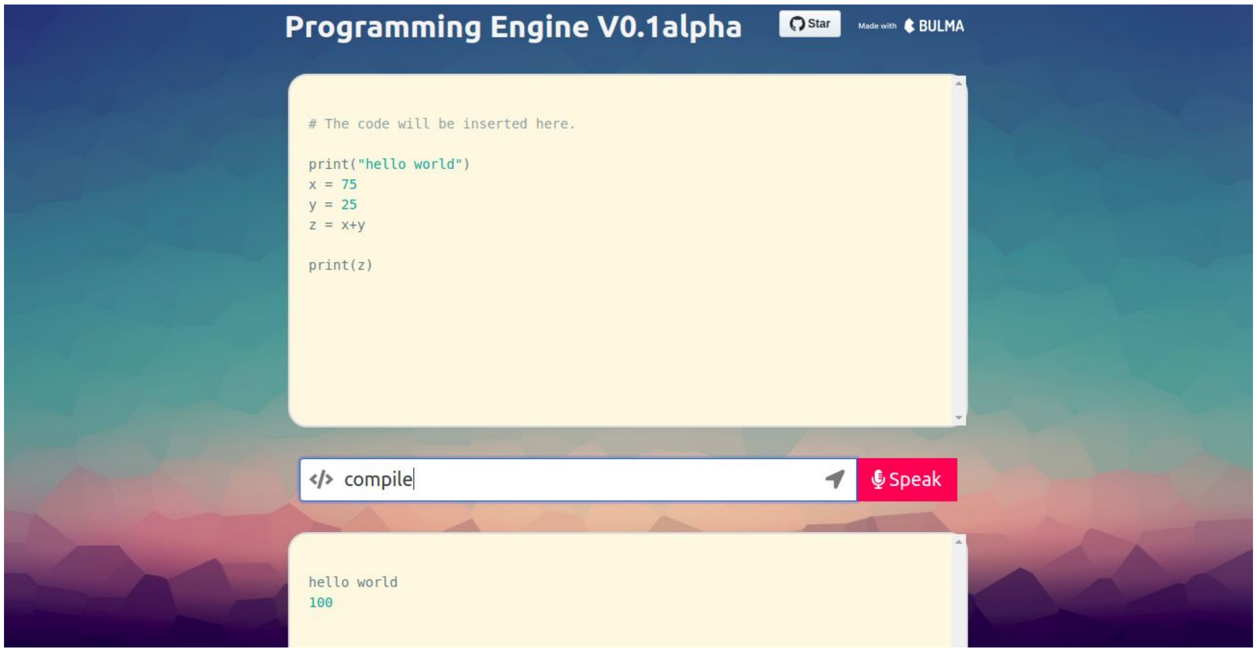


```
# The code will be inserted here.

print("hello world")
x = 75
y = 25
```

z = x + y

```
# Output
```



```
# The code will be inserted here.

print("hello world")
x = 75
y = 25
z = x+y

print(z)
```

```
hello world
100
```

Chapter 5

Results and Conclusion

5.1 Expected Results

The user will speak out, i.e. by voice, a specific task that he wants to achieve in natural language, for example

“Assign value 10 to a. Let b equals 20. Add a and b and store it in c. Print the value of c.”, will be converted to Python/C code which looks something like this.

```
python3.5/hello.py
a=10
b=20
c=a+b
print(c)
```

A single statement that a user speaks can be as simple as a print command like “Print hello world” or a statement which performs a complex function like “Define a function to reverse a string”. These two cases will be handled differently. The first statement can be handled by creating intents and mapping them to all major programming constructs like assignments, arithmetic operations, loops, functions and so on. The second case will depend heavily on large datasets of code. Large repositories of code like Github may have many snippets which can be used to directly to get results. An optimal data structure can be created which will make loading of these snippets faster and more accurate.

5.2 Conclusion

The intention of developing a Programming Engine is to remove the hurdles experienced by a naive user while programming, doing away with the need of having to learn the peculiar specifics of a particular programming language. The Engine is also flexible and adaptable due to the independent layered structure that it has, thus making it a well-engineered design for future generations of it.