# CS142 - Spring 2020:
# Computability and Complexity
# Mid Semester Project.
# Language Parsers and Translators

Link to the GitHub Repository: https://github.com/arhamhameed/Language-Parsers-and-Translators

Question 1: An English Language Limited Parser [#languages, #ComputerSimulations].

**1.a. Provide a formal definition of your grammar.**

My grammar includes nouns, verbs, pronouns, adjectives, prepositions, and conjunction. The formal definition of my grammar can be described as a 4- tuple.

$$G_1 = \{T, \sum, S, R\}$$

Here:

1. $T = \{S, VP, NP, PP, NM, N, V, A, P, C, PR, PRN, AR\}$

Where some of them stand for:

1. N are nouns
2. V are verbs
3. A are adjectives
4. P are pronouns
5. C are conjunctions
6. PR are prepositions
7. PRN are proper nouns
8. AR are articles

2. $\sum$ = {Argentina, Pakistan, Arham, laptop, tea, camera, mountains, Himalayas, bike, lake, rode, drank, jumped, ran, fought, fast, hot, magnificent, aesthetic, dirty, me, I, we, you, they, it, us, from, to, over, under, and, but, or, a, the, an}

Where:

1. **Nouns** = {Argentina, Pakistan, Arham, street, tea, laptop, mountains, Himalayas, Bike, Lake}
2. **Verbs** = { drove, drank, jumped, ran, fought}
3. **Adjectives** = { fast, hot, magnificent, aesthetic, dirty}
4. **Pronouns** = {me, I, we, you, they, it, us, he}
5. **Proposition** = {from, to, over, down, in}
6. **Conjunction** = {and, but, or}
7. **Articles** = {a, the, an}

3. S is the sentence given by our grammar

4. R is basically the rules of the grammar:[1]

S → NP | VP
VP → VP PP | V NP | V NP PP | V PP
PP → P NP
NM → NM PP | N NM |vwegvet N
NP → P | N | N C N | P C N | N C P | P C P | AR A NM | AR NM
P → 'from' | 'to' | 'over' | 'down' | 'with'

[1] Took help from this lecture slide deck while writing grammar rules and parse trees:
6.864: Lecture 2, Parsing and Syntax I. (2005). Retrieved from https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-864-advanced-natural-language-processing-fall-2005/lecture-notes/lec2.pdf

AR → 'a' | 'an' | 'this' | 'the'
N → 'Argentina' | 'Pakistan' | 'Arham' | 'street' | 'tea' | 'laptop' | 'mountains' | 'Himalayas' | 'bike' | 'lake'
V → 'drove' | 'drank' | 'jumped' | 'ran' | 'fought'
A → 'fast' | 'hot' | 'magnificent' | 'aesthetic' | 'dirty'
P → 'me' | 'I' | 'we' | 'you' | 'they' | 'it' | 'us' | 'he'
C → 'and' | 'but' | 'or'
PR → 'from' | 'to' | 'over' | 'down' | 'in'

**1.b. Is your grammar ambiguous? Either way, justify your answer.**

A grammar is called ambiguous if there is more than one derivation for the given sentence (input string). This can be seen in the example provided by MIT, on parsing trees, where they give two parse trees for the sentence "*he drove down the street in the car"*.

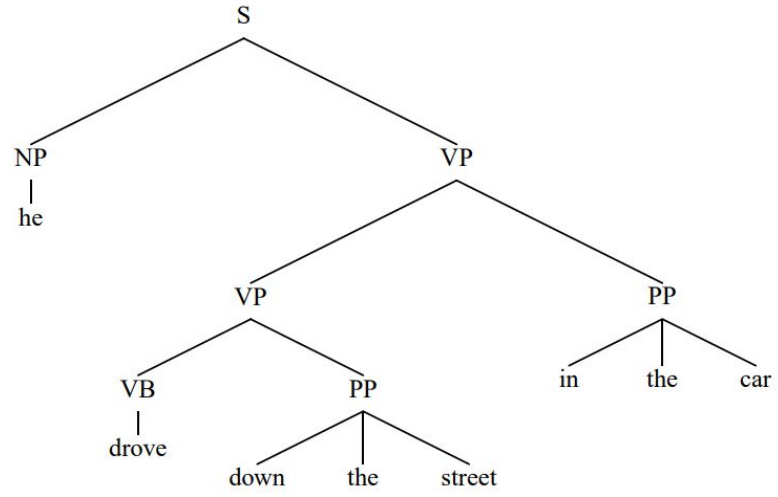


*Figure 1*

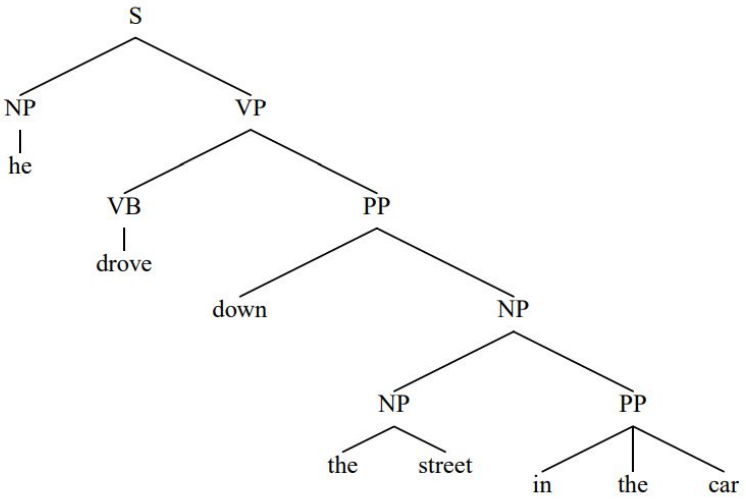


*Figure 2*

We can see that in both the figures above the parse tree constructed the same string. It is similar to our grammar where we can build the sentence *"he drove down teh street in the bike"*. This is an example of ambiguous grammar because we have more than one leftmost derivation for the same input string.

**1.c. Provide a formal definition of an equivalent machine of your choice that can recognize the strings in the language generated by your grammar. We call this Machine Parser1.**

Parser 1 would be a Turing Machine (TM) and it will function as a decider. This means that the machine at each node will either accepts or rejects before it will move on to the next node. Parser 1 will first check for string membership in the grammar defined in the part 1.a. The way TM will decide this is by firstly converting it to CNF (Chomsky Normal Form). The reason to do so is that it will allow us to check for all derivations exhaustively because every derivation of a string has 2n -1 steps. Since this is a finite number, our TM would be a decider rather than a recognizer that recognizers regular grammar where we can have potentially infinitely many derivations.

After we convert our string to CNF form we will use the Cocke-Younger-Kasami (CYK) algorithm which is a parsing algorithm for Context-Free Grammar and uses dynamic programming and bottom-up parsing to determine the potential derivations. CYK algorithm is easier to implement than a TM but the process and outputs are the same.

We can provide a formal definition of the machine based on the Theorem 4.7 from Sipser's Introduction to Theory of Computation (2013):

The TM $S$ = "On input $\langle G, w \rangle$, where $G$ is a CFG and w is Context-Free Grammar:

1. Convert $G$ to an equivalent grammar in Chomsky normal form.
2. List all derivations with $2n-1$ steps, where $n$ is the length of the input string; except if $n = 0$, then instead list all derivations with one step.
3. If any of these derivations equal to $w$, *accept*; if not, *reject*."

**1.d. Write a simulator in Python for your Parser1. You can define your own machine or use any of the available machines in the Python automata-lib.**

In order to run the program:

1. Clone repository to your local computer
2. Download all the required modules. Hopefully, most of them will be already installed, just look out for the typing module. You can install it using 'pip3 install typing'
3. Enter the file folder through the terminal and once inside the CS142_MidSemesterProject folder type 'python parser1.py' or 'python3 parser1.py'

Question 2 Word by Word Translator [#languages, #computability]

**2.1 Design and give a formal definition of your automaton that carries the word by word translations into either English-Spanish, or English-Telugu depending on your rotation.**

Since we need to translate word to word, the task requires an enumerator. That means we are going to build a Turing machine that is not only a decider but also prints as well. The formal definition would be the 7 tuples, as this is a Turing machine, $T = \{Q, \sum, \gamma, \delta, q_0, q_{print}, q_{reject}\}$, *here* :

1. $Q$ is the set of states
2. $\sum$ is the input alphabet not containing the blank space
3. $\gamma$ is the tape alphabet where the blank symbol is also contained in $\gamma$ and $\gamma$ is a subset of our input alphabet
4. $\delta$ is the transition function for all the English words to Spanish
5. $q_0$ is the start state
6. $q_{print}$ is the print state where the enumerator reads the $\gamma$
7. $q_{reject}$ is the reject state

The way this Turing Machine is designed is that it has 3 tapes:

1. The first tape contains the input string
2. The second tape contains the terminals in the alphabets that corresponds to their Spanish translation
3. The last tape is the printing tape that prints out the word-by-word translation

In order to run the translator:

1. Clone repository to your local computer
2. Download all the required modules. Hopefully, most of them will be already installed!
3. Enter the file folder through the terminal and once inside the CS142_MidSemesterProject folder type ‘python translator.py’ or ‘python3 translator.py’

**2.2 What are the main computational drawbacks and limitations of this word-by-word translator?**

The main drawback would be its ability limit to recognize complicated sentence structures and how the same word can have different meanings in different contexts. The translator will always translate a single word to a single translated word disregarding in which context that word is being used. Also, since it also first recognizes if the sentence is recognized by the machine, there are very few sentence structures that can be recognized by the machine and translated. The limitations would involve the construction of a translation key and dictionary where we need to have every word that ever existed to be present in the translation key with its translation in order for the translator to translate everything.

Question 3: Auto-Code Function Generator [#languages, #computability, #ComputerSimulations]

**3.a. Provide a formal definition of your language and grammar.**

Question 3 is very similar to question 1 when it comes to providing the formal definition of the language, grammar, and machines. Just like question 1, here we have again 4 tuples:

$$G_1 = \{T, \sum, S, R\}$$

1. $T = \{I, S, G, D, U, E, F, C, V, B, P, K, N\}$
2. $\sum$ = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, *, /, =, x, y}
3. S is the sentence given by our grammar
4. R are the rules of our grammar:

U → F '(' B ')' E G
B → V | V B
B → ',' B
G → D | V | K | G S G | '(' G ')' | G G
K → I N '.' N | I N | N '.' N | N
Z → D | D K
E → '='
V → 'x' | 'y' | 'z'
F → 'f' | 'g'
I → '+' | '-'
S → '+' | '-' | '*' | '/'
G → 'sin' | 'cos' | 'tan' | 'sqrt'
D → '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

**3.b. Provide a formal definition of an equivalent machine of your choice that can recognize the language described by your grammar. We call this Machine ParseFunc.**

The formal definition of this machine will be similar to that of teh machine we described above in question 1.c. The only difference is that our previous machine recognized a different grammar $G_1$ whereas, this machine will recognize teh grammar $G_2$.

**3.c. Write a simulator in Python for your *ParseFunc*.**

The *ParseFunc* is included in the python repository. In order to run it enter the file folder through the terminal and once inside the CS142_MidSemesterProject folder, type ‘python ParseFunc.py’ or ‘python3 ParseFunc.py’ to run the program.

**3.d. Write Python code that calls ParseFunc to determine if a given input is the language of your Python function generator. Then, if the input string is accepted, your program outputs back the equivalent Python function.**

The Python function generator is included in the python repository. In order to run it enter the file folder through the terminal and once inside the CS142_MidSemesterProject folder, type 'python fun_gen.py' or 'python3 fun_gen.py' to run the program.

Question 4: Concluding Remarks [#languages, #computability].

**[a]. In this assignment, we have explored a very simple example of rule-based NLP, which was used in the early decades of NLP, including AI applications. Nowadays, natural language translators incorporate machine learning techniques to provide more accurate language translations (these are examples of statistical-based NPL).**

**Briefly, write a short summary of how language translators have evolved from ruled-based translations to statistical-based ones (see references below). Focus on the syntax part only, as the field of NLP has a vast number of ramifications. Your summary should address the following points:**

- **Give an example of a statistical-based language translator (e.g. Google translate [c] or your favorite language translator) and briefly explain how it works.**
- **Highlight some of the benefits and drawbacks for statistical NLP vs rule-based NLP**
- **In your opinion, what kind of language generators and recognizers (machines) are needed for rule-based and statistical-based translators.**

IBM first designed its 701 ML translator that successfully translated 49 sentences on the topic of chemistry from Russian into English (Nicole Post, 2018). The original behind MT was to build computers that would perform rule-based translations independently. Rule-based translators require huge human effort to prepare the rules and linguistic resources. Whereas data-driven approaches provide an alternative to rule-based MT systems and have been used extensively over the past decade. These approaches use a supervised or unsupervised statistical machine learning algorithm to build statistical models from the bilingual parallel corpora.

My favorite translator: Google translate, is powered by statistical MT. These translators work on the premise that the user feeds enough data to the computer's expansive memory in the shape of parallel texts in two languages that it will be able to identify and recreate the patterns that the machine finds statistically in the inputs. It becomes self-learning by feeding its machine to grow its corpus. Just like Google Translate has a feedback option from its users to help it get better.

**The differences between Rule-Based Machine Translators VS Statical Machine Translators**

| RBMT | SMT |
| --- | --- |
| Consistent and predictable translation quality | Unpredictable translation quality |
| High performance and robustness in translations | Requires high CPY and Memory to store the data |
| Knows grammatical rules | Does not know grammatical rules |
| Lack of fluency in translations | Fluent translations |
| Hard to handle complicated sentence structure and connotations | Better for handling complicated sentence structures |

**HC Tags:**

**#algorithms:** I used CYK algorithm instead of using ™ algorithm from automata-lib as it was easier to implement while getting the same functionality and output for the task at hand. Also created a github repository with readme to help the user launch the program successfully.

**#breakitdown:** I used this HC to decompose the problem into smaller parts where I first built the parser and then the translator. And after I build back these smaller pieces to build a whole translator that first parses the language for recognizability and then translates it.

**#composition:** I documented the code so that the reader can understand the processes. Named the variables in an intuitive way that while reading it is easier to keep track of the processes and write output functions in such a way that the reader can modify the input strings to check for testability of my algorithm.. Also, I used figures and tables to explain my reasoning better in the assignment prompt.

**References:**


6.864: Lecture 2, Parsing and Syntax I. (2005). Retrieved from https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-864-advanced-natural-language-processing-fall-2005/lecture-notes/lec2.pdf

McHardy, R.. (2017) CYK-Parser (Python Code) Retrieved from: https://github.com/RobMcH/CYK-Parser

Post, N. (2020, March 24). The Evolution Of Machine Translation. Retrieved from https://localizeblog.com/the-evolution-of-machine-translation/

Sipser, M.. (2013) Introduction to the theory of computation. Boston, MA: Cengage Learning.

Sreelekha, S. (2017). Statistical Vs Rule Based Machine Translation; A Case Study on Indian Language Perspective. Retrieved from https://arxiv.org/ftp/arxiv/papers/1708/1708.04559.pdf

Lemone, K.. (n.a.) Context-free Grammars for Natural Languages Retrieved from: https://web.cs.wpi.edu/~kal/PLT/PLT2.1.1.html

What is Machine Translation? Rule Based Machine Translation vs. Statistical Machine Translation. (n.d.). Retrieved from https://www.systransoft.com/systran/translation-technology/what-is-machine-translation/

Wikipedia. (n.a.) CYK algorithm Retrieved from https://en.wikipedia.org/wiki/CYK_algorithm

Wikipedia. (n.a.) Chomsky Normal Form Retrieved from https://en.wikipedia.org/wiki/Chomsky_normal_form