Implementing CKY Algorithm in Java

Dhrupad Kaneria
Graduate Student, UT Dallas
800 W Campbell Rd
Richardson TX 75080
dhrupad.kaneria@utdallas.edu

ABSTRACT

Within computer Science, the term **Parsing** is used in the analysis of computer languages, referring to the syntactic analysis of the input code. This report deals with one of the parsing algorithm for context-free grammar. The report provides an introduction to "Cocke, Younger, Kasami" algorithm. It gives the introduction to parsing along with steps to implement the Cocke-Younger-Kasami (CYK) algorithm in Java language. The report also gives the different experiments that have been conducted to test the correctness of the implemented software.

1. INTRODUCTION

1.1 Parsing

Parsing [1] (or *Syntactic analysis*) is the process of analyzing a string of symbols, either in natural language or in computer languages, conforming to the rules of a formal grammar. Within computational linguistics the term is used to refer to the formal analysis by a computer of a sentence or other string of words into its constituents, resulting in a parse tree showing their syntactic relation to each other, which may also contain semantic and other information.

1.2 CYK Algorithm

The Cocke-Younger-Kasami (alternatively called CYK, or CKY) [2] is a parsing algorithm for context-free grammars, named after its inventors, John Cocke, Daniel Younger and Tadao Kasami. It employs bottom-up parsing and dynamic programming. The importance of the CYK algorithm stems from its high efficiency in certain situations. The standard version of CYK algorithm operates only on context-free grammars given in Chomsky Normal Form. However there is a way to transform any context-free grammar to the equivalent grammar in Chomsky Normal Form.

1.3 Chomsky Normal Form

The standard version of CYK algorithm operates on context-free grammars given in Chomsky Normal Form (CNF) [3]. However any context-free grammar can be transformed into CNF which represents the same grammar. The CYK algorithm requires the context-free grammar to be rendered into CNF because it tests for possibilities to split the current sequence in two parts. Hence any context-free grammar that does not generate the empty string can be represented in CNF using the production rules:

$$A \rightarrow BC \text{ and}$$

$$A \rightarrow \alpha$$

Where A, B, C are non-terminal symbols and α is a terminal symbol

2. RELATED WORKS

Algorithms to parse context-free languages has been known for a long time. There are many notable algorithms namely Earley Parser, CYK Algorithm and Chart Parsing [4].

Earley Algorithm is implemented using dynamic programming and top-down approach. In this algorithm, a special symbol ● is used in the right-hand side of the production rule to indicate the progress that is being made.

The CYK algorithm is a classic example of the dynamic programming paradigm. This algorithm implements bottom-up approach.

In both CYK and Earley algorithms, the order in which different events occur is determined by the procedures that make up these algorithms. Chart Parsing facilitates dynamic determination of the order in which the entries are processed.

This report gives the implementation of the CYK algorithm present in most of the Natural Processing Language textbooks. There are many more efficient ways to implement this algorithm which can result in a faster execution [6].

3. APPROACH

The important part of CYK algorithm is parsing. At first basics of parsing was learnt and understood. After which, the working of CYK algorithm was studied. Once the working of the algorithm was clearly understood, the next step was to select the language to be used to implement the algorithm. Java language was selected since it is most widely used. The below section explains the implementation of the software and also provides pseudocode for the same.

3.1 Implementation

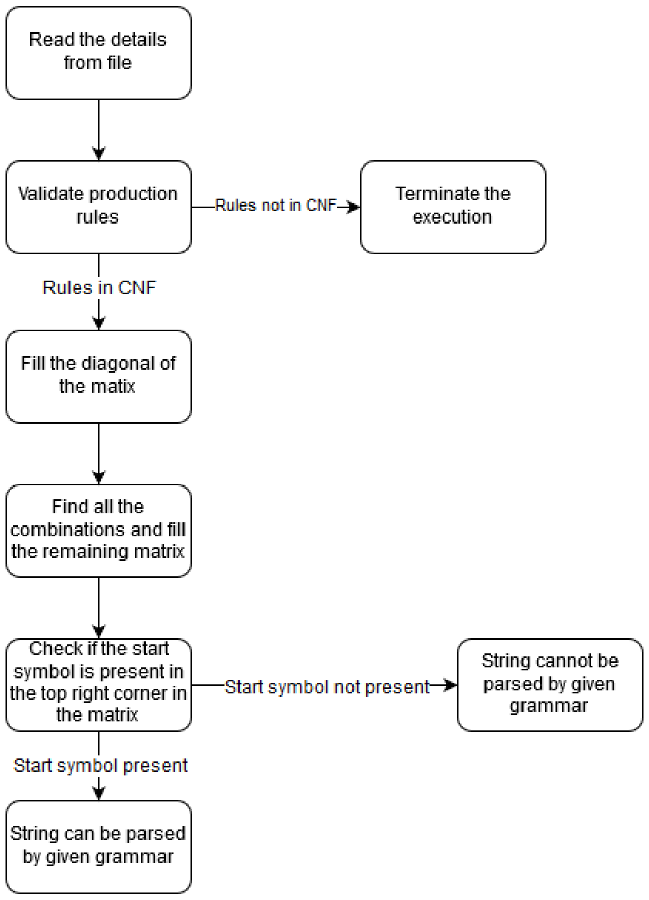
The implementation initially began by following the algorithm blindly and directly. At first the production rules are checked for correctness and check whether they belong to CNF. If they are not in CNF, the execution is terminated. Assuming the production rules abide by the given rules and are in CNF, we further proceed to fill the matrix.

In the first loop, the diagonal of the matrix is filled. This is done by checking the terminal symbol with their LHS. In the next loop, we iterate through all the possible combinations of the two strings and check if they can be produced. If they are produced, we make a note of it, save it and move to the next entry.

At the end, the entire matrix is filled. We then check if the start symbol is present in the right most index of first row. If it is present, the entered string can be parsed with the given grammar, if it does not exist, the string cannot be parsed by the given grammar.

3.2 Pseudocode

```
Read the details from the file
For (int I = 0; I < num_prod; ++I) //For each productions
{
Extract the LHS
Extract RHS
Validate LHS
Validate RHS
If the production rule is present in CNF
Add it to the grammar for future use
}
For (int I = 0; I < strLength; ++I)
Fill the diagonal elements of the matrix
For (int k = 1; k < strLength; ++k)
For (int j = k; j < strLength; ++j)
For (int l = j-k; l < j; ++l)
{
Generate different combinations of the intermediate string
If the generated string is a part of production rules
{
Save the symbol in the corresponding cell in the matrix
}
}
Display the upper half of the matrix
If the top right corner has the start symbol
{
The string can be parsed by the given grammar
}
Else
{
The string cannot be parsed by the give grammar
}
```


Read the details from file
Validate production rules
Rules not in CNF
Terminate the execution
Rules in CNF
Fill the diagonal of the matix
Find all the combinations and fill the remaining matrix
Check if the start symbol is present in the top right corner in the matrix
Start symbol not present
String cannot be parsed by given grammar
Start symbol present
String can be parsed by given grammar

4. SPECIAL INSTRUCTIONS

4.1 Input

The input for the software should be a text file called “inputGrammar.txt”. This input file should be in a required format given below:

- Line 1 : Number of production rules
- Line 2 : The start variable
- Line 3 : The string to be parsed
- Line 4 to end : The production rules for the grammar, one rule in a single line

4.2 Output

The output at the end of execution of the software is the CYK table (matrix). The upper half of the matrix is displayed in order to represent the CYK table. If there are more than one symbol in a single cell of the matrix, they are separated by a comma. At the end, the software also tells if the given string can be parsed by the given grammar.

If the validation of the input fails, corresponding error messages are shown and the execution is terminated at the very moment.

4.3 Note

There are special instructions to be followed in writing the production rules in the text file. Some of the rules are mentioned in the earlier section. Other rules are mentioned here.

In order to execute the program successfully, the rules should strictly be in Chomsky Normal Form (CNF). The LHS of the production rule should strictly be a non-terminal symbol. RHS can be of two non-terminal symbols (separated by a space) or a single terminal symbol. Also, care should be taken to make sure there are no space between the production rules except between the two non-terminal symbols in the RHS.

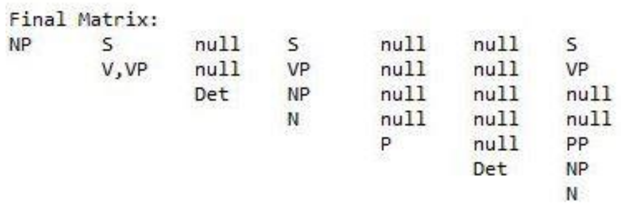
5. EXPERIMENTS

5.1 Test 1

The grammar used for this test is selected form Wikipedia. The grammar is given below:

S→NP VP	VP→VP PP
VP→V NP	VP→eats
PP→P NP	NP→Det N
NP→she	V→eats
P→with	N→fish
N→fork	Det→a

The string to be parsed is “she eats a fish with a fork”. The matrix generated by the software is:

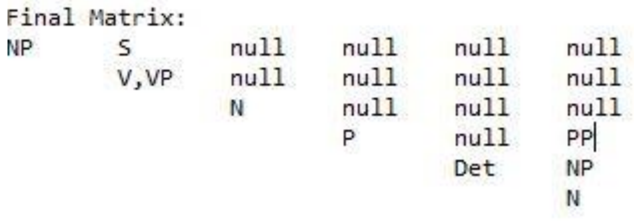


```
Final Matrix:
NP      S       null    S       null    null    S
        V,VP    null    VP      null    null    VP
                Det     NP      null    null    null
                        N       null    null    null
                                P       null    PP
                                        Det     NP
                                                N
```

Since the starting symbol i.e. S is at the top - right corner of the matrix, the string can be parsed by the given grammar.

5.2 Test 2

For testing the false case i.e. when the grammar is not able to parse the string, the grammar used is the same as above. The string to be parsed is changed to “she eats fish with a fork”. The matrix obtained for this case is:



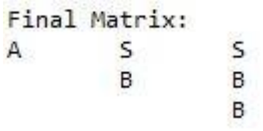
```
Final Matrix:
NP      S       null    null    null    null
        V,VP    null    null    null    null
                N       null    null    null
                        P       null    PP
                                Det     NP
                                        N
```

5.3 Test 3

Another experiment that has been carried out is not from English grammar. The grammar is given below

S→A B	A→a
A→S C	B→b
B→B B	C→c
C→C A	

The string to be parsed is “a b b”



```
Final Matrix:
A       S       S
        B       B
                B
```

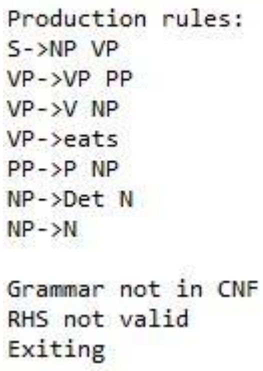
This grammar is taken for examining and testing only to check that the parser is not only made for English grammar. Hence proved that the software can be used to any language that is in Chomsky Normal Form.

5.4 Test 4

The next test gives an idea of what happens if the entered grammar is not in CNF. The grammar entered has a production rule that doesn’t comply with the rules of CNF.

S→NP VP	VP→VP PP
VP→V NP	VP→eats
PP→P NP	NP→Det N
NP→N	NP→she
V→eats	P→with
N→fish	N→fork
Det→a	

This grammar has a production rule “NP→N”. This rule violates the rules of CNF. The outcome that the program returns is:

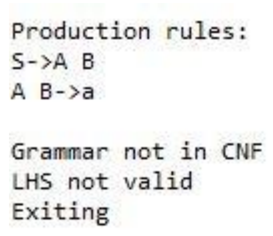


```
Production rules:
S->NP VP
VP->VP PP
VP->V NP
VP->eats
PP->P NP
NP->Det N
NP->N

Grammar not in CNF
RHS not valid
Exiting
```

5.5 Test 5

This testing is done to check if a faulty LHS is detected by the software. One of the grammar rule in this test case has the LHS with two non-terminal symbols. This should be caught by the software and corresponding error message should be displayed.



```
Production rules:
S->A B
A B->a

Grammar not in CNF
LHS not valid
Exiting
```

6. CONCLUSION

The successful completion of the software gives a clear vision and a better picture of the algorithm. It gives a clear understanding of the steps involved in CYK algorithm.

Implementing the algorithm, helps to learn the worst case running time complexity of CYK algorithm as θ (n^3. |G|) where n is the length of the parsed string and |G| is the size of the CNF grammar. The space complexity of CYK algorithm is θ (n^2) since there are only n^2 entries in the matrix for a given string of length n [5].

Even though this algorithm will run correctly, decisions about the size of the grammar, the length of the string to be parsed, early termination and caching can have a significant impact on the execution [6].

Another conclusion that is determined about the CYK algorithm is that it is a recognizer, not a parser [5]. For the algorithm to succeed it simply has to find the start symbol in the corresponding cell. For the algorithm to be a parser, it should return all the possible parses for a given input. There is a simple way to change this algorithm form recognizer to a parser. The table entries can be paired with pointers that keep track of the symbols from where they are derived. This will incur considerable cost to the software.

7. REFERENCES

[1] Parsing, https://en.wikipedia.org/wiki/Parsing

[2] CYK Algorithm, https://en.wikipedia.org/wiki/CYK_algorithm

[3] Chomsky Normal Form, https://en.wikipedia.org/wiki/Chomsky_normal_form

[4] Parsing with Context-Free Grammar, http://www.hnk.ffzg.hr/Jurafsky/12.pdf

[5] Li, Te, and Devi Alagappan. A Comparison of CYK and Earley Parsing Algorithms.

[6] Bondenstab, Nathan. Efficient Implementation of the CYK Algorithm (2009): Fall 2009.