Contents

1. Parsing with Context Free Grammars

1.1. LL and LR Parsers

LL grammars are a subset of context-free grammars and impose more restrictions to simplify parsing [1]. LL parsers operate by taking the input and processing it from left to right and yields a leftmost derivation of the taken input. Thus, they are classified as a type of top-down parsers. They are named as LL due to being left-to-right and yielding leftmost derivation.

On the other hand, LR parsers take the input and process from left to right but creates a rightmost derivation of taken input. To create a rightmost derivation, this parser starts from lexicons and tries to match an expansion rule. Therefore, LR parsers are classified as bottom-up parsers because they try to parse starting from a terminal-to-root approach.

When the variants of these parsers are considered, not only the current input but also the forward parts of inputs are used for deciding. These derivations are called as lookahead and when the parser is allowed to check k next symbols it is called k-lookahead. On the other hand, when k is equal to 1 these parsers are classified as without lookahead [3].

1.2. Recursive Descent Parser

Recursive descent parser tries to parse the input by creating sub-goals and achieving them. Starting from S, sub-goals are created recursively till to the leaf nodes and input words are tried to be matched. When the match occurs, parser goes one step up and this recursively continues until it reaches to root. Therefore, this parser is a top-down parsing approach implementation. In addition, this parser tries multiple possible branches at the same time so a backtracking is implemented in the case of failure at the interested branch [4].

Main drawbacks of this parser can be listed as:

- Left recursive grammar rules can go infinite loop.
- Parser wastes a lot of time for words that are not related to the input sentence
- Backtracking discards efforts of other branches and parse the same input partition again and again.

1.3. Shift Reduce Parser

Shift reduce parsers use a bottom-up approach with the help of a data structure. This method reads the input from left-to-right and incrementally creates the parse tree where this creation does not involve any backtracking or prediction and list of sub-trees are collected as scanning goes on. However, the collected sub-trees are joined at the end of parsing when the root is found. These operations mainly can be combined under the name of “shift” and “reduce”. In “shift” stage, the next input word is read and included into parse tree. Then, in the “reduce” stage, a possible grammar rule is found where all conditions are hold and these sub-trees related to conditions are combined [5]. When these operations resulted with the root at the end of input string, the input is correctly parsed and accepted. Otherwise, it cannot be parsed with the given grammar.

1.4. CKY and Earley Parsers

Cocke - Younger - Kasami (CYK) parsing is a bottom-up parsing method that can be applied on grammars that are in Chomsky Normal Form (CNF). However, since any grammar can be converted into CNF in linear time, CKY parsing can be applied to any grammar without a restriction. Efficiency of this parser is based on its worst-case complexity; although there are other methods which are better in average running time.

CKY method is a solution for blindly generating all parse trees without considering the string that needs to be parsed. This drawback is solved by using a data structure named as “chart” [6]. Using charts, input words are taken and possible expansion rules are saved into related sections of the chart. In other words, starting from leaves root of the tree is tried to be found, where the root is complete sentence in this sense.

On the other hand, Earley parsing is a top-down approach that can be applied to any form of grammar without any restriction. The main idea behind Earley parsing is keeping record of not only completed productions but also storing incomplete productions, which are partially parsed constituents. In other words, these incomplete parts are predictions about what might come next [7]. Principle of this parser can be summarized as starting from all possible rules, parallel construction of all incomplete rules are followed until input words are found and this approach is done until the whole sentence is parsed.

When CKY and Earley parsers are compared, the following points can be listed:

- Both use dynamic programming to create a table which includes all possible parsers.
- Space complexity for CYK is $O(n^2)$ since there are n^2 table entries; whereas space complexity of Earley parsing is $O(n)$ [8].
- Time complexity for both algorithms is $O(n^3)$; however, $O(n^3)$ is only an upper bound of Earley and on most grammars Earley does better [8].
- CKY is a bottom up approach whereas Earley is a top-down approach.
- CKY works only on CNF grammars whereas Earley can work on any arbitrary context free grammar.
- CKY keeps record of completed constituents whereas Earley keeps record of both completed and in-progress (predicted) constituents.

2. Implementation Idea

In this homework, a grammar for relative clauses in English is implemented. Since only the clauses start with “who” and “which” are asked, two main relative clause expansions are considered. Starting from the provided “base” context-free grammar, new lexicons, expansion rules and feature structures are added for correctly parsing and refusing relative clauses.

2.1. Analysis of Implementation

When the precision and recall of the grammar is calculated on the dataset provided in “try.py” file, the following tables are constructed:

		Actual Class	
		Positive	Negative
Grammar Output	Positive	10	1
	Negative	2	17

Confusion Matrix

Precision	Recall	Grade
10 / (10 + 1) = 0.91	10 / (10+2) = 0.83	0.87

2.2. Possible Problems and Improvements

When the drawbacks of the grammar is considered, there are few points to mention:

- Implemented grammar cannot differentiate the usage of object in some sentences and cannot correctly refuse the sentence “David ate pizza which Tom gave pizza” where repetition takes place. *(False positive)*

- Implemented grammar cannot handle correctly relative clauses which are relative clause of another clause and they are changing roles in sentences. For instance, grammar could not parse this sentence: “the men who went to the park see the stars with the telescope which the boy find” *(False negative)*

- In this implementation only the lexicons related to this homework are included, for a further development a more comprehensive list of lexicons must be included.

3. Conclusion

To conclude, in this homework a context-free grammar is implemented for relative clauses for English and a report about this implementation and parsers is prepared. In this report, firstly parsing methods are presented and then the implemented grammar is described.

4. References

[1] LL Grammars

http://www.cs.uaf.edu/~cs331/notes/LL.pdf

[2] Bottom-Up Parsing

http://www.cs.uky.edu/~lewis/essays/compilers/bu-parse.html

[3] The Honalee LR(k) Algorithm

http://david.tribble.com/text/honalee.html

[4] Natural Language Processing with Python, Steven Bird, Ewan Klein and Edward Loper

[5] Introduction to Shift-Reduce Parsing
http://www.cs.binghamton.edu/~zdu/parsdemo/srintro.html

[6] Lecture 15: CYK Parsing Algorithm, Sariel Har-Peled & Madhusudan Parthasarathy

http://courses.engr.illinois.edu/cs373/sp2009/lectures/lect_15.pdf

[7] Earley Parsing, Mirella Lapata

http://www.inf.ed.ac.uk/teaching/courses/inf2a/slides/2011_inf2a_L18_slides.pdf

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