BLG 336E – ANALYSIS OF ALGORITHMS II

Project 2

Muhammed Burak Buğrul
150140015

Introduction

We are wanted to implement a language parser for the given context free grammar.

Environment

Project implemented in an **Ubuntu 16** machine with **C++**.

Classes

Parser: Contains necessary functions an d variables for algorithm.

- **Parser::Parser():** Initialing grammar rules.
- **Parser::split(string s)**: Splits words of string s and stores them in a string vector
- **Parser::toLower(string s):** Turns letters of string s to lower
- **Parser::f(vector<string> &input, string state, int beg, int end):** Main part of the algorithm. It is a divide and conquer logic parser function with memoization.

Code

In the code there are some explanations(comments).

Analysis

- If the advantages of divide and conquer methodology were not to be used in your implementation, how would you formulate the problem?

  \- By analysing the parse tree we can see a solution of a node(subproblem) only depends on children(subproblems) of its own. So there can be a solution by combining little subproblems to solve bigger ones. Same algorithm can be implemented by a bottom up solution(nested for loops) from little subproblems to bigger ones with help of a memoization.

- What parameters does the complexity of your implementation depend on? How would you represent the worst case complexity in big O notation?

  - For the sake of simplicity, let **N** be the number of words of an input and **M** be the number of state transitions of grammar. Time Complexity of the algorithm is **$O(MN^3)$.** Because there are **$O(N^2)$** (beg,end) intervals in the input. For every interval, the algorithm tries to divide it into two distinct intervals. There are **$O(N)$** different division for an interval. So we have **$O(N^3)$** now. But division has no meaning without grammar rules. For every division, algorithm tries to give a meaning to a division by simply trying every state transition of that state. There are **$O(M)$** state transitions. In conclusion, time complexity of the algorithm is **$O(MN^3)$.** It could be more slower but occurrence of overlapping subproblems gave the idea of **memoizitaion** for this problem. So in the algorithm, there are no additional process.

- If the rule VP → NP VP were to be added to current set of grammar rules, this would cause ambiguity because it would overlap with the rule S → NP VP. Briefly explain, how you would change your implementation to overcome this problem. Would it change the worst case complexity? If so, what would the complexity be in big O notation?

  \- In my solution. First state is always S. So there are no overlapping in the beginning. If we think state transitions of non terminals as a graph, once we leave state S, there is no way to reach state S again. There will be no overlapping in the tree because if we encounter a ‘NP VP’ situation because it can not be a S state, it will be a VP state.