Umeå University
Efficient Algorithms

Assignment Step 2

Runtime analysis of the Java implementation of the CYK-algorithm

Pina Kolling

September 26, 2022

Contents

1 Introduction

Parsing in Computer Science is the process of analysing a string of characters to examine if the string is built according to the rules of a formal grammar. [1]
A formal grammar describes how to form strings with correct syntax from a language's alphabet. To examine if such a string follows the rules of a grammar the CYK-algorithm can be used. Therefore the grammar needs to be in a specific format, called CNF (explained in section 2).

The task for this assignment was to code three different parsing methods to execute the CYK-algorithm in *Java*. For this three different classes were implemented: `main.java, grammar.java` and `parser.java`. The `main`-class calls the methods and the `grammar`-class parses the input grammer ans string into a format that then can be processed in the `parser`-class. The function and implementation will be former described in section 3.

2 Background

- grammar
  A formal grammar describes how to form strings with correct syntax from a language's alphabet. A grammar does not describe the meaning of the strings or any semantics — only their syntax is defined.
- CNF
- CYK
- DP

3 System Design

My system worked as follows ...

4 Evaluation

We did some experiments ...

5 Conclusions and Future Work

From our experiments we can conclude that ...

A How to use the code?

The code can be run in the terminal and input is expected as Strings in quotation marks. The grammar needs to be in CNF. The first rule begins with the startsymbol of the grammar.

First: Rules without arrows (one rule as one String)
Last: The last argument is the input word

Input example (*Well-Balanced-Parantheses*):
`java Main "SSS" "SLA" "SLR" "ASR" "L(" "R)" "(())"`
for the grammar `S` $\rightarrow$ `SS | LA | LR`, `A` $\rightarrow$ `SR`, `L` $\rightarrow$ `(`, `R` $\rightarrow$ `)` and the input word `(())`.

Output example:
The first part of the output shows the arrays, which get generated in the `Grammar.java` class.

The first array contains all rules.

The second array contains only the terminal rules.

The third array contains only the nonterminal rules.

Then it is shown which nonterminal symbols are represented by which integers. Later the nonterminal symbols can be referred to with those integers.

After this the mentioned arrays are shown again but the nonterminal symbols got replaced with the according integers.

```
Matrix all rules:
[SS, LA, LR]
[SR, , ]
[(, , ]
[), , ]

Matrix T rules:
[]
[]
[(]
[)]

Matrix NT rules:
[SS, LA, LR]
[SR, , ]


Integers of NT symbols:
[0, 1, 2, 3]
[S, A, L, R]

Matrix all rules:
[00, 21, 23]
[03, , ]
[(, , ]
[), , ]

Matrix T rules:
[]
[]
[(]
[)]

Matrix NT rules:
[00, 21, 23]
[03, , ]
```

```
Input word: (())

Naive: true   Amount of calls: 33
Naive runtime: 9ms

CYK-Table (Bottom Up):
[2, , , ]
[, 2, 0, 1]
[, , 3, ]
[, , , 3]

BottomUp: false   Amount of calls: 84
Naive runtime: 4ms

TopDown: true   Amount of calls: 28
Naive runtime: 1ms
```

Then the results, counter and runtime in *ms* is shown for each parsing method.
For the `BottomUp` method is the CYK algorithm table printed.

References

[1] Fabio Massimo Zanzotto, Giorgio Satta, and Giordano Cristini. "CYK Parsing over Distributed Representations". In: *Algorithms* 13 (Oct. 2020), p. 262. DOI: 10.3390/a13100262.