Runtime analysis of the Java implementation of the CYK-algorithm

Pina Kolling
piko0011@student.umu.se

September 26, 2022

Contents

1 Introduction

My project was about . . .
I developed a system to . . .
We did some experiments to find out . . .
The main results were . . .
[1]

2 Background

I did some background reading in the following areas . . .

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 . . .

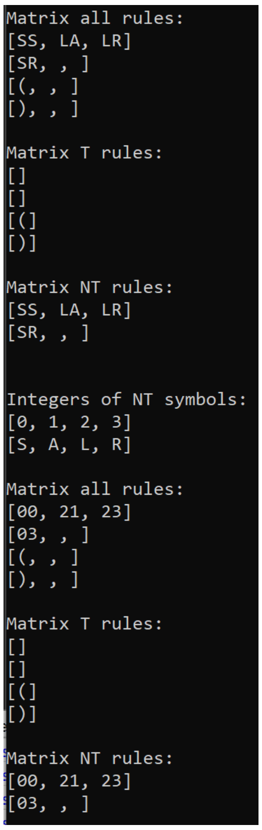
A How to use the code?

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.



```
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, , ]
```

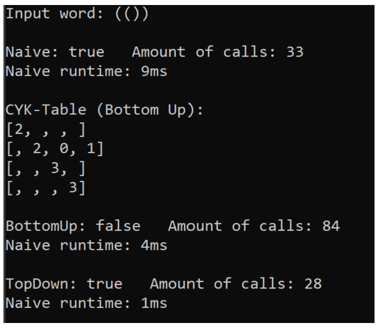
The first array contains all rules.

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.



```
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] Janet Chen. *Group Theory and the Rubik's Cube.* Online erhältlich unter `http://people.math.harvard.edu/~jjchen/docs/Group_Theory_and_the_Rubik's_Cube.pdf`; abgerufen am 05.01.2021.