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 **2**

2 Background **3**
2.1 Formal language . . . 3
2.2 Formal grammar . . . 3
2.3 Chomsky-Normal-Form . . . 3
2.4 CYK-algorithm . . . 4
2.5 Dynamic programming . . . 4

3 System Design **5**

4 Evaluation **6**

5 Conclusions and Future Work **7**

A How to use the code? **8**

Bibliography **10**

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.
A formal grammar describes how to form strings with correct syntax from a language's alphabet (section 2.2 and 2.1). To examine if such a string follows the rules of a grammar the *Cocke-Younger-Kasami*-algorithm (short: *CYK*) can be used. This algorithm is described in section 2.4. To use the *CYK*-algorithm the grammar needs to be in a specific format, that is called *Chomsky-Normal-Form* (*CNF*), which is explained in section 2.3. [2, 4]
The task for this assignment was to code three different parsing methods to execute the *CYK*-algorithm in *Java*. The different parsing methods will be described and presented as pseudo code in section 3. For the implementation 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

In this section background information on formal languages (section 2.1), formal grammars (section 2.2), Chomsky-Normal-Form (section 2.3), CYK-algorithm (section 2.4) and dynmaic programming (section 2.5) will be presented.

2.1 Formal language

Formal languages are abstract languages which define the syntax of the words that get accepted by that language. It consits of a set of words that get accepted by the language and a set of symbols that is called alphabet and contains the characters of the words. Those characters are calles nonterminal symbols. [**cnf**, 3]

Example (Formal language)**.** [1]
The language accepts words that contain the same number of as and bs, while the a has to be left of the b. The alphabet Σ *of this language looks like this:*

$$\Sigma = \{a, b\}$$

The language definition L is the following one:

$$L = \{(a^n b^n)^m\} \text{ with } n, m \in \mathbb{N}$$

2.2 Formal 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. The grammar is a set of rules which define which words are accepted by a formal language. Those rules consist of terminal and nonterminal symbols. The terminalsymbols are the characters of the alphabet of the language and the nonterminalsymbols are used to build the rules of the language – they get replaced by terminalsymbols. [1, 3]

Example (Formal grammar)**.** *The example grammar for the previous example language is the following:*

$$S \rightarrow SS \mid aSb \mid ab$$

2.3 Chomsky-Normal-Form

The *Chomsky-Normal-Form* (short: *CNF*) is a grammar which is formated in a specific way. If the startsymbol (nonterminal symbol) is not generating the empty word ($S \rightarrow \epsilon$) it either generates two nonterminal symbols or one terminal symbol for the grammar to be in *CNF*. [1]

[1] The following examples show the *Well-Balanced Parantheses* example from the assignment task sheet with the alphabet $\{a, b\}$ instead of $\{(,)\}$.

Example (Chomsky-Normal-Form)**.** *To change the previous example into CNF the rules have to be split up:*

$$S \rightarrow SS \mid LA \mid LR$$
$$A \rightarrow SR$$
$$L \rightarrow a$$
$$R \rightarrow b$$

2.4 CYK-algorithm

2.5 Dynamic programming

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 ...

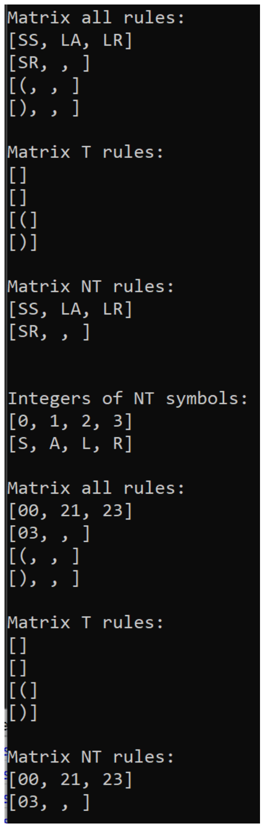
A How to use the code?

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] *Chomsky's Normal Form (CNF)*. Website. `https://www.javatpoint.com/automata-chomskys-normal-form`, opened on 26.09.2022.

[2] Glenn K. Manacher. "An improved version of the Cocke-Younger-Kasami algorithm". In: *Computer Languages* 3.2 (1978), pp. 127–133. ISSN: 0096-0551. DOI: `https://doi.org/10.1016/0096-0551(78)90029-2`. URL: `https://www.sciencedirect.com/science/article/pii/0096055178900292`.

[3] A.J. Kfoury Robert N. Moll Michael A. Arbib. *An Introduction to Formal Language Theory*. Springer-Verlag, 1988.

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