ICIS 409

Automata and Formal Theory

Professor Chinwe Ekenna

Homework 2 Process Report

Introduction:

In this report, we will cover the part 3 requirements to explain how we implemented and tested CNF form grammar from part 1 to test whether the given input string is part of that CNF string. We used the CYK algorithm to determine whether a given string can be implemented by specific CNF grammars.

First, we converted Context-Free Grammar G = (V, X, R, S) into Chomsky Normal Form by using techniques that we learned from class. Second, we used the Cocke-Younger-Kasami Algorithm (CYK algorithm).

Additional information regarding CFG: V is a finite set of variables, X is a finite set of terminal symbols, R is a finite set of rules, and S is an axiom or starting symbol.

Brief explanation of the CYK algorithm:

This algorithm is often used to determine the membership problems. This algorithm requires a Grammar G to be in Chomsky Normal Form. If the given input string consists of a string of length m, then the last (bottom) row of CYK table will have m number of cells, 2nd last row will have (m-1) number, 3rd last row will have (m-2) cells, etc. Eventually, the top row will have only 1 cell. This cell is used to determine whether the given input string is a member of CNF grammar G. This process is known as a bottom-up parsing tree, and if the top row accepts the starting symbol, then the given input string will be accepted by the grammar.

Overview of implemented algorithm and data structures

Our CYK Algorithm can be broken down into a 3 step process: Rule Parsing, CYK Triangle building, and String checking.

Rule Parsing:

The first step of our algorithm involves reading a given text file containing context-free grammar in Chomsky Normal Form. We have created and implemented our own data structures, called Rule and CNFParse, to read and store this grammar.

- A *Rule* is used to store a single rule of the given grammar. It has a String which stores the left-hand side of the rule, and a list of Strings which store the right-hand side of the rule.

- A *CNFParse* stores a list of all the Rules of the given grammar. It allows for implementation of functions such as getting a list of rules through which a string can be obtained.

A detailed structure of our class and data structure can be seen in the provided UML.

CYK Triangle Building

The CYK Triangle is necessary to determine whether the given string can be accepted by the grammar. We have used the bottom-up parsing tree method, but modified it to be able to implement it using a nested for loop. The resulting triangle formed through our code has the longest row (containing m cells) at the top, and the row with the single cell at the bottom, which we check for the presence of the starting state.

Challenges faced during the programming process:

There were 2 major challenges that we faced during the programming process. The first was on how to parse and store the given grammar from the text file in a way that they could be easily implemented as needed for the process of building the CYK Triangle. The second was on how to build and fill in the CYK Triangle itself.

- Our solution for our first challenge was to come up with our own unique data structures (called Rule and CNFParse) to store the given grammar. A Rule is a structure that stores each individual rule of the given grammar (in Chomsky Normal Form). It uses a string to store the left-hand side (variable) of the rule, and a list to store the strings that can be obtained through that variable. CNFParse contains an array of all the Rules of the given grammar. This structure was implemented to perform operations on the rules, such as getting a list of variables that can be used to obtain a specific string.

- Figuring out the algorithm to fill up the CYK Table for a given String using a specific set of rules was the greatest challenge. The indexing and methods that we used to produce the strings in each cell of the table had to be figured out so that they could be used generally for all the cells in the table. Furthermore, although the process would be easier using nested loops, the indexing resulting from using nested loops would not be the same as the indexing used when performing the algorithm by hand. To figure this out, I drew out 2 tables - one with indexes set up using nested loops, and one with indexes set up in the normal manner as we would use when solving the problem by hand. Then I went through filling out the table step by step, making note of the indexes I used and converting them into the indexes that would be used in the nested loop.

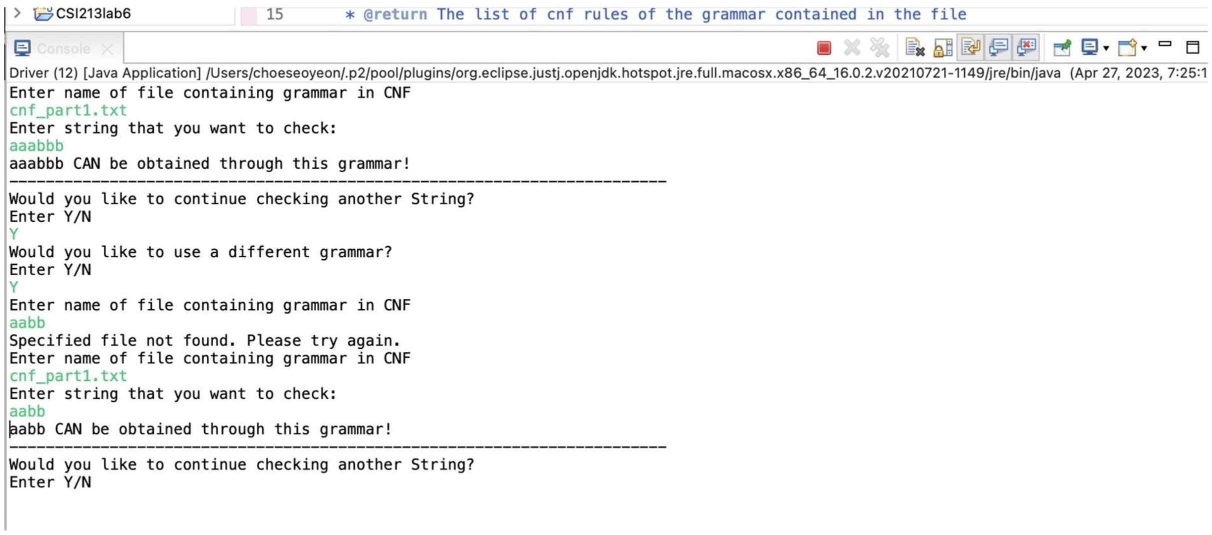
Testing process and result part:

It implements the CYK algorithm for a given input string by calling the CYK class by running the driver class to test whether a given input string w is acceptable by the CNF rules. The CYK class will call the CNFparse class and populate the CYK table. We have separately named four different CNFs in part 1. For example, if we want to test whether a given input string w is allowed in cnf_part1 grammar G, call "cnf_part1.txt" and enter the input you want to test.

Sample Testing Outputs:

For cnf_part1.txt

Based on part 1 CNG, you can generate an equal number of a and b or an empty string. Examples: aabb or aaabbb

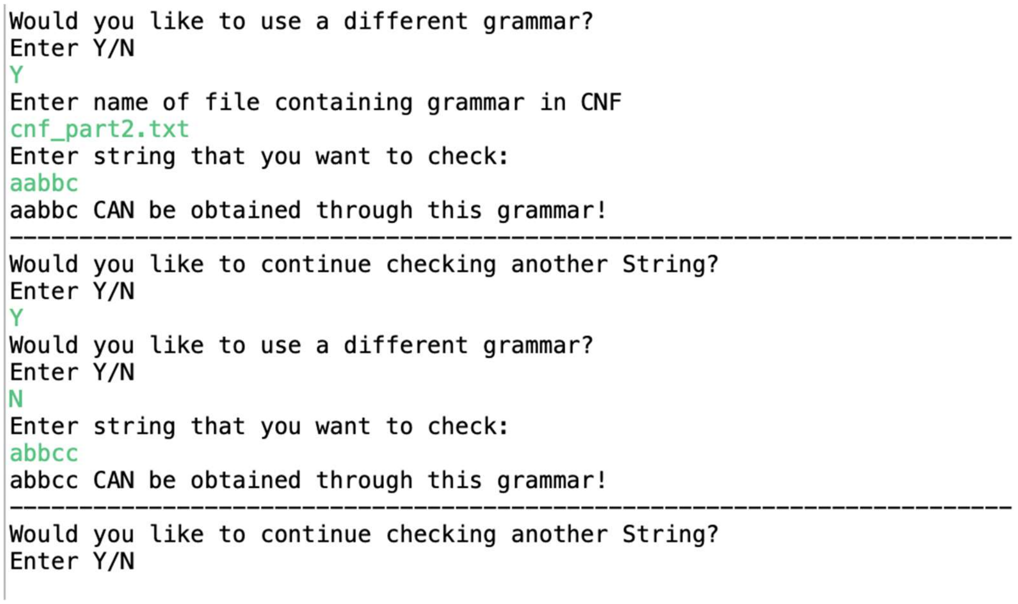


```
> CSI213lab6        15       * @return The list of cnf rules of the grammar contained in the file
Console
Driver (12) [Java Application] /Users/choeseoyeon/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_16.0.2.v20210721-1149/jre/bin/java  (Apr 27, 2023, 7:25:1
Enter name of file containing grammar in CNF
cnf_part1.txt
Enter string that you want to check:
aaabbb
aaabbb CAN be obtained through this grammar!
--------------------------------------------------------------------------
Would you like to continue checking another String?
Enter Y/N
Y
Would you like to use a different grammar?
Enter Y/N
Y
Enter name of file containing grammar in CNF
aabb
Specified file not found. Please try again.
Enter name of file containing grammar in CNF
cnf_part1.txt
Enter string that you want to check:
aabb
aabb CAN be obtained through this grammar!
--------------------------------------------------------------------------
Would you like to continue checking another String?
Enter Y/N
```

Based on the output above, the CYK algorithm program accepts an input string and uses the CYK algorithm to check if the given string is a member of the part 1 language. The result was successful and aabb and aaabbb can be generated by the cnf_part1 rule.

cnf_part 2.txt

A given grammar can produce equal numbers of a and b *or* equal numbers of b and c. Example: aabbc and abbcc.

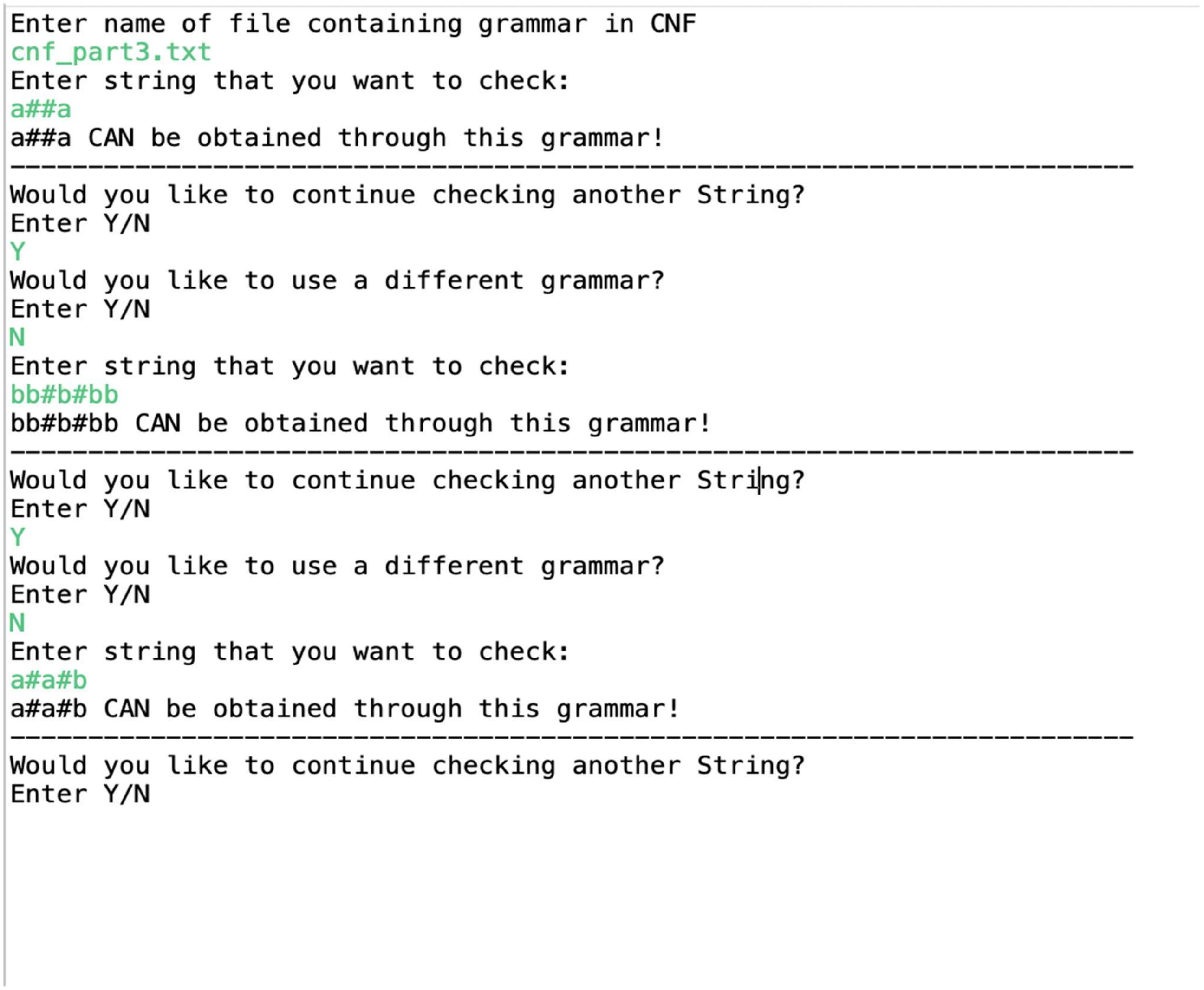


```
Would you like to use a different grammar?
Enter Y/N
Y
Enter name of file containing grammar in CNF
cnf_part2.txt
Enter string that you want to check:
aabbc
aabbc CAN be obtained through this grammar!
------------------------------------------------------------------------
Would you like to continue checking another String?
Enter Y/N
Y
Would you like to use a different grammar?
Enter Y/N
N
Enter string that you want to check:
abbcc
abbcc CAN be obtained through this grammar!
------------------------------------------------------------------------
Would you like to continue checking another String?
Enter Y/N
```

The output above shows that the given input can be generated by the cnf_part2 grammar and that the given input is a member of part 2 and the grammar.

cnf_part 3.txt

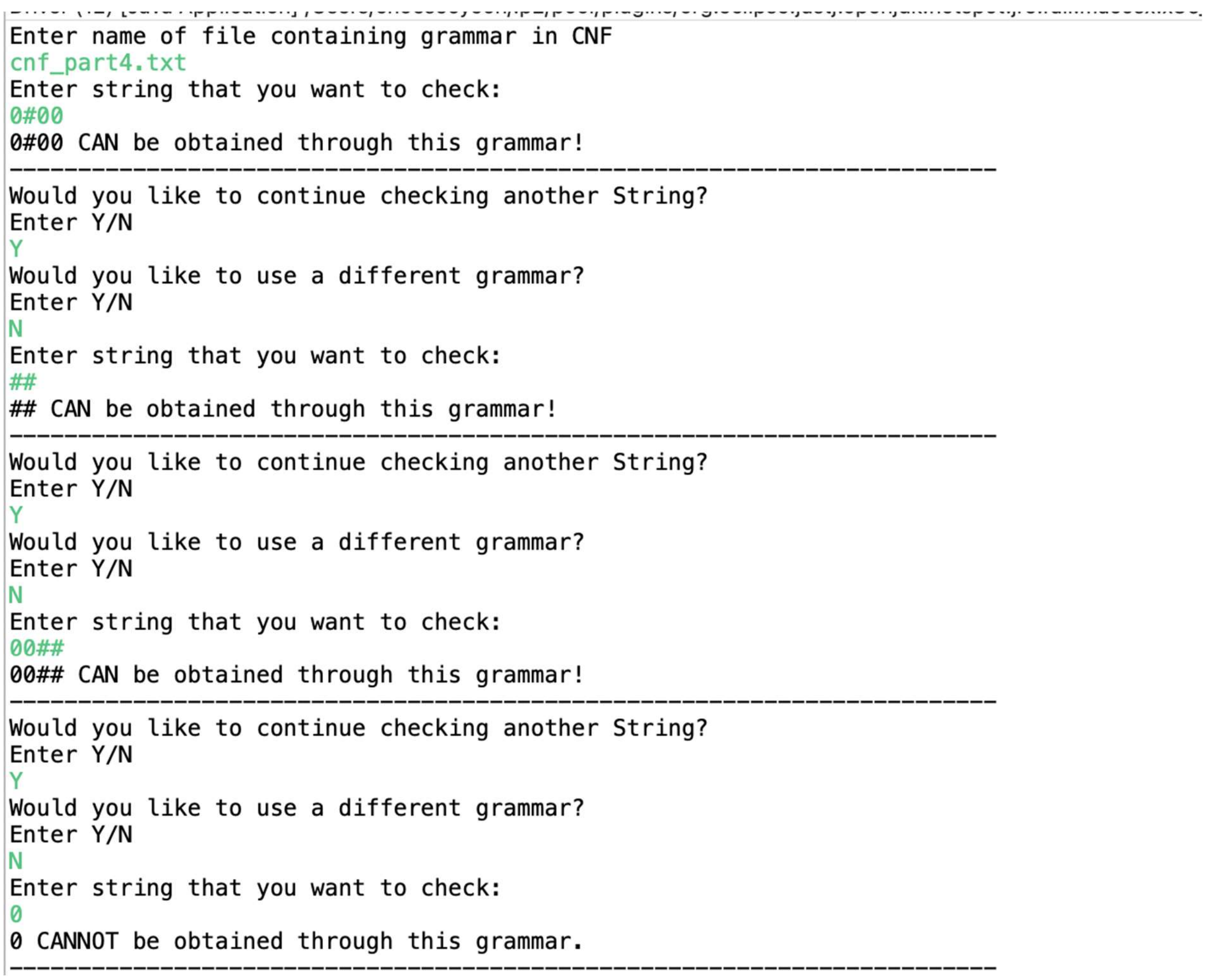
A given grammar can generate any number of a and b, and a and b are separated by #. For example, you should be able to generate a##a, bb#b#bb, and a#a#b. So this grammar produces the length of b and # of a.



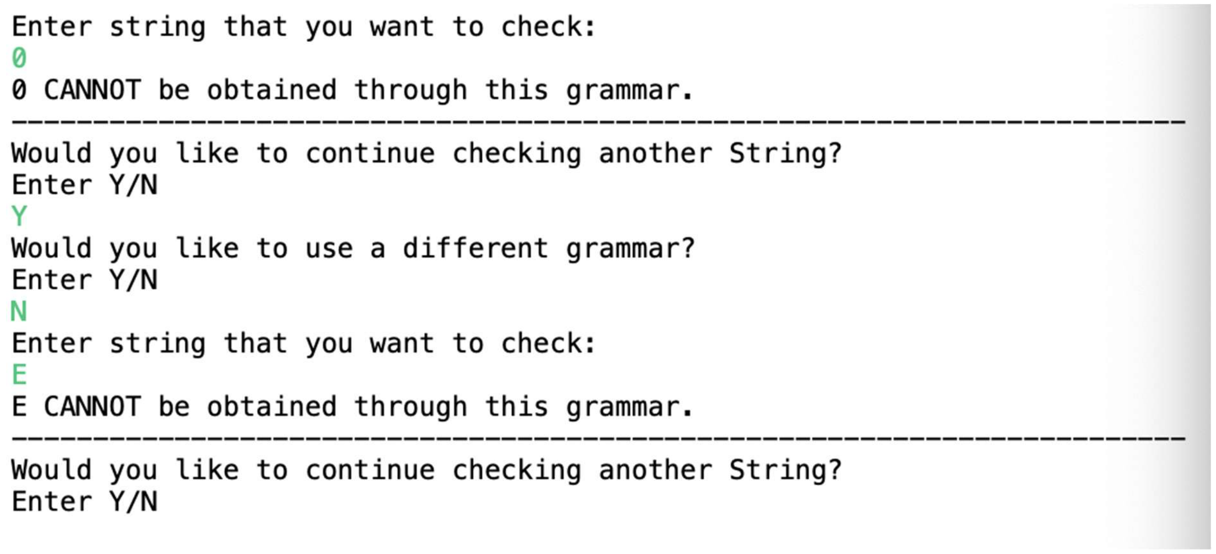
```
Enter name of file containing grammar in CNF
cnf_part3.txt
Enter string that you want to check:
a##a
a##a CAN be obtained through this grammar!
------------------------------------------------------------------------
Would you like to continue checking another String?
Enter Y/N
Y
Would you like to use a different grammar?
Enter Y/N
N
Enter string that you want to check:
bb#b#bb
bb#b#bb CAN be obtained through this grammar!
------------------------------------------------------------------------
Would you like to continue checking another String?
Enter Y/N
Y
Would you like to use a different grammar?
Enter Y/N
N
Enter string that you want to check:
a#a#b
a#a#b CAN be obtained through this grammar!
------------------------------------------------------------------------
Would you like to continue checking another String?
Enter Y/N
```

cnf_part 4.txt

The given grammar must allow any combination of 0 and #, and this grammar must not accept either the empty string or just 0. So it tests with the input strings 00##, 0#00, and ##. This means the cnf grammar and CYK algorithm will work if that string is allowed. Also, the input string 0 must not be accepted.



```
Enter name of file containing grammar in CNF
cnf_part4.txt
Enter string that you want to check:
0#00
0#00 CAN be obtained through this grammar!
--------------------------------------------------------------------------
Would you like to continue checking another String?
Enter Y/N
Y
Would you like to use a different grammar?
Enter Y/N
N
Enter string that you want to check:
##
## CAN be obtained through this grammar!
--------------------------------------------------------------------------
Would you like to continue checking another String?
Enter Y/N
Y
Would you like to use a different grammar?
Enter Y/N
N
Enter string that you want to check:
00##
00## CAN be obtained through this grammar!
--------------------------------------------------------------------------
Would you like to continue checking another String?
Enter Y/N
Y
Would you like to use a different grammar?
Enter Y/N
N
Enter string that you want to check:
0
0 CANNOT be obtained through this grammar.
--------------------------------------------------------------------------
```



```
Enter string that you want to check:
0
0 CANNOT be obtained through this grammar.
------------------------------------------------------------------------
Would you like to continue checking another String?
Enter Y/N
Y
Would you like to use a different grammar?
Enter Y/N
N
Enter string that you want to check:
E
E CANNOT be obtained through this grammar.
------------------------------------------------------------------------
Would you like to continue checking another String?
Enter Y/N
```

The above result shows that the CYK algorithm only accepts strings that can be generated by CNF grammar.

Analysis of time and space complexity:

Detail of time complexity T(n) is included in a separate pdf file.

CYK class time complexity

- public CYK() Function takes $O(n^3)$
- fillTable() takes $n*n-1$, so $O(n^2)$
- fillCell() takes ~~O(1) or O(n)~~. $O(n^2)$
- ArrayList() takes $O(n^2)$
- checkInGrammer() take $O(1)$

$\longrightarrow T(n) = n^3+n^2+n^2+n^2+1$, so $O(n^3)$.

CNFParse class Time complexity

- CNFParse() takes $n*(n-1)$ so $O(n^2)$
- getStart() takes $O(1)$
- getRule() takes $O(n)$
- belongToRules takes $O(n)$

$\longrightarrow T(n) = n^2+1+n+n$, so $O(n^2)$.

Rule class Time complexity

- most of function takes just $O(1)$
- inRule() takes $O(n)$
- toString() takes $O(n)$

$\longrightarrow T(n) = 1+1+\ldots\ n+n$, so $O(n)$.

Driver Class T(n).

- ArrayList() takes O(n)
- main() takes. O(1) for printing values,
  main() class CYK class - this take $O(n^3)$
  checkInGrammer() Function is in CYK class and that takes O(1)
  to check grammer and printout outputs.

$T(n) = n+1 \ldots + n^3 + 1$, so $O(n^3)$.