

Formal Languages and Automata Theory

Final Report

Boxiong Zhao

Iordanis Fostiropoulos

Weixin Wu

CSE 4083 Formal Languages and Automata Theory

Instructor: Debasis Mitra

April 6, 2013

Table of Contents

Overview

Algorithm

Program manual

Other observations

Project management

References

Acknowledgements

Appendices

Overview

The project is based on the requirements set by Dr. Mitra within his Syllabus. We were asked to “write a program, for an input CFG, and then a string, be ready to generate/reject the later”. The objective of the program was to create a string parser given a Context Free Grammar. A string parser is a program used by compilers to identify program syntax and be able to convert the given program to machine code. Application of string parsers can be found on Artificial Intelligence where the natural language is converted into digital commands. Furthermore the application of such parser was obviously an intriguing project with potential for expansion. After the research done on parsers, we concluded the scope of our program.

The scope of our program is defined by the inputs and the outputs. After research we concluded that the simplest grammar to use as input is CNF (Chomsky Normal Form).

$$\begin{aligned} A &\rightarrow BC \text{ or} \\ A &\rightarrow \alpha \end{aligned}$$

It is simple to understand and parse. We also realized that most of the Context Free Grammar that Dr. Mitra would ask us to input would be invalid or too hard to convert to CNF. Furthermore we made a grammar converter that would convert any Context Free Grammar to Chomsky Normal Form.

In order to make debugging easier for us and make the process of parsing more understandable to us, Dr. Mitra or any other future user, we output a *CYK table*. The *CYK Table* is the table created during the use of the CYK Algorithm. The CYK Table is really

similar to a parsing tree. The inputted string is on the top of the output window and through a series of steps it is converted to the starting symbol S . At the end of the table there is a conclusion of whether or not the string belongs to the language described by the Context Free Grammar.

Since no specific instructions on algorithm or approach of the project were given we chose to solve this problem with CYK algorithm using Java.

Algorithm

CYK [2]

1. **let** the input be a string S consisting of n characters: $a_1 \dots a_n$.
2. **let** the grammar contain r nonterminal symbols $R_1 \dots R_r$. This grammar contains the subset R_s which is the set of start symbols.
3. **let** $P[n,n,r]$ be an array of Booleans. Initialize all elements of P to false.
for each $i = 1$ to n
for each unit production $R_j \rightarrow a_i$
 set $P[i,1,j] = \text{true}$
4. **for each** $i = 2$ to n -- *Length of span*
for each $j = 1$ to $n-i+1$ -- *Start of span*
for each $k = 1$ to $i-1$ -- *Partition of span*
for each production $R_A \rightarrow R_B R_C$

if $P[j,k,B]$ and $P[j+k,i-k,C]$ **then** set $P[j,i,A] = \text{true}$

if any of $P[1,n,x]$ is true (x is iterated over the set s , where s are all the indices for R_s)

then S is member of language

else

S is not member of language

Chomsky Normal Form to Context Free Grammar [1]

1. Remove all nonterminal from the right hand side of all productions except the unit productions.
2. Replace any rule that has three or more nonterminals with the equivalent rules of size two.
3. Replace every rule S with S' . Add a new rule $S \rightarrow S'$
4. Remove all *epsilon* transitions by iterating their equivalent form. For each production that includes a terminal that is equal to *epsilon*, add another rule equal to the initial rule excluding the terminal that is equal to *epsilon*.
5. Remove all the *unit rules* of the form $A \rightarrow B$

Combining everything

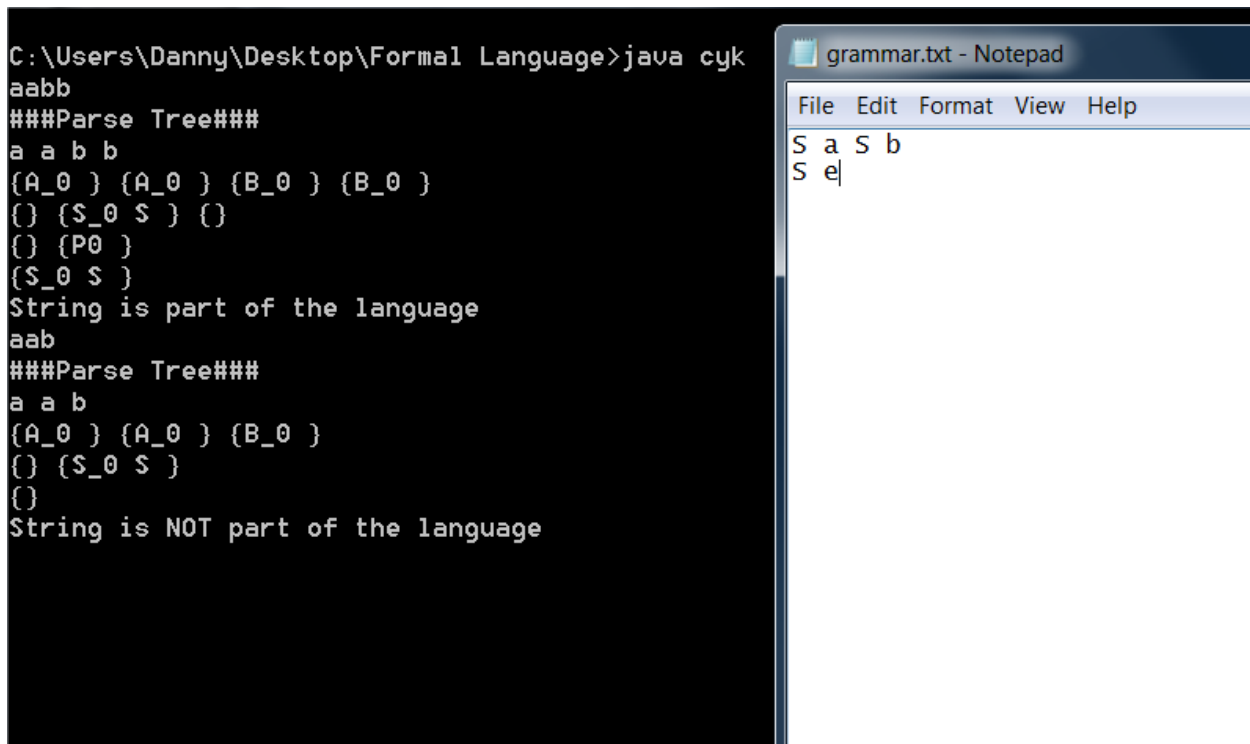
1. Read Grammar
2. Turn the CFG into Chomsky normal form

3. Iterate until *EOF* to read strings
4. for every string use CYK algorithm to determine if the string is a member of the language

Program Manual

The program can be run in any operating system as long as java is installed. Open the command line of the operating system and navigate (use `cd C:\folderpath` for windows) to the folder that the java files are. After being on the same folder execute the java command. The java command is

javacyk



The screenshot shows a Windows command prompt window and a Notepad window. The command prompt window displays the output of the `java cyk` command. It processes two input strings: `aabb` and `aab`. For `aabb`, it prints a parse tree and states "String is part of the language". For `aab`, it prints a parse tree and states "String is NOT part of the language". The Notepad window, titled "grammar.txt - Notepad", shows the grammar rules: `S a S b` and `S e`.

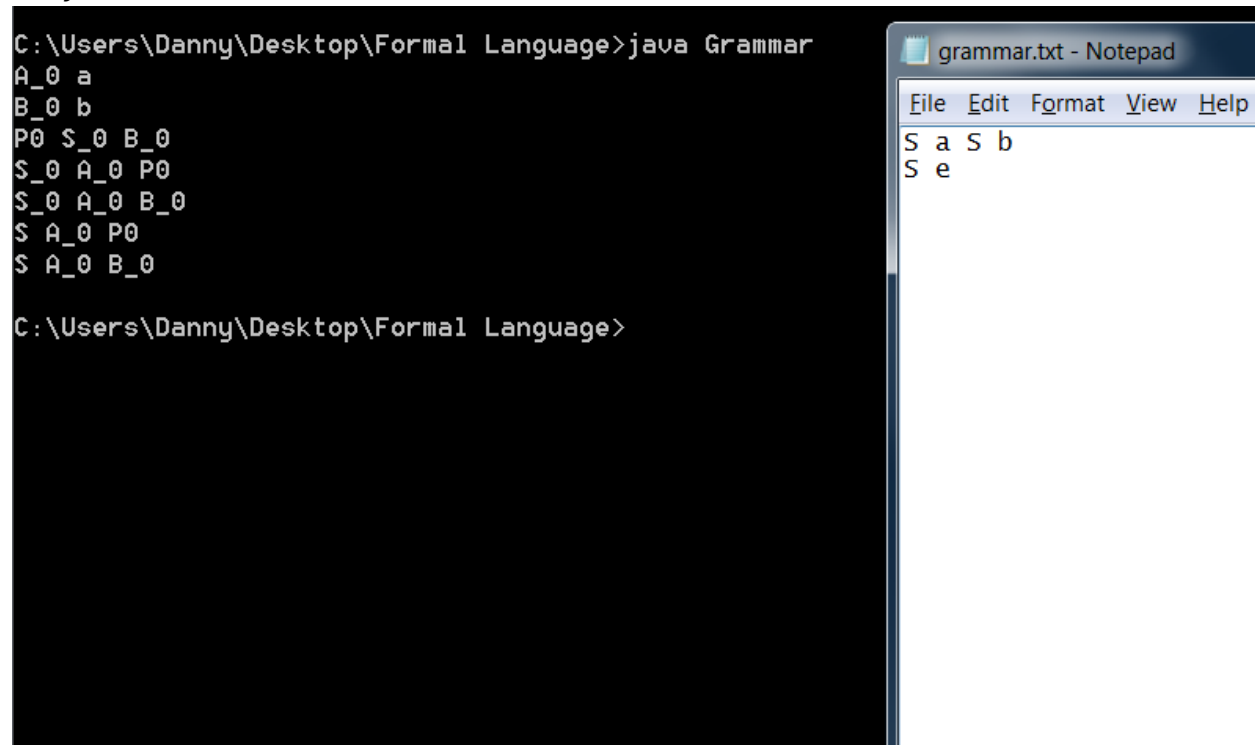
```
C:\Users\Danny\Desktop\Formal Language>java cyk
aabb
####Parse Tree###
a a b b
{A_0 } {A_0 } {B_0 } {B_0 }
{ } {S_0 S } { }
{ } {P0 }
{S_0 S }
String is part of the language
aab
####Parse Tree###
a a b
{A_0 } {A_0 } {B_0 }
{ } {S_0 S }
{ }
String is NOT part of the language
```

grammar.txt - Notepad

```
File Edit Format View Help
S a S b
S e
```

After that a loop will be accepting and evaluating string of the given Grammar till the user terminates the program by signaling EOF. (This can be done in windows by holding CTRL+Z)

You can convert any Grammar to CNF by inputting it on the Grammar file and running
java Grammar



The screenshot shows a Windows command prompt window with the following text:

```
C:\Users\Danny\Desktop\Formal Language>java Grammar
A_0 a
B_0 b
P0 S_0 B_0
S_0 A_0 P0
S_0 A_0 B_0
S A_0 P0
S A_0 B_0

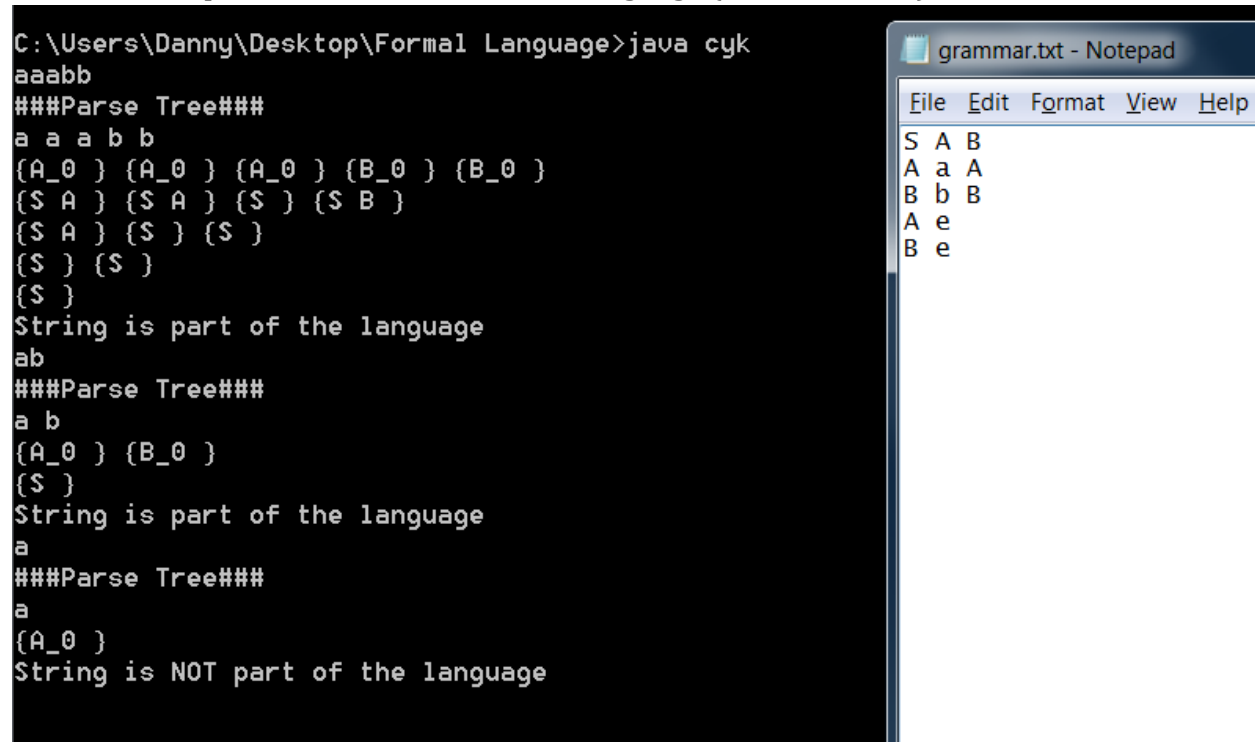
C:\Users\Danny\Desktop\Formal Language>
```

Next to the command prompt is a Notepad window titled "grammar.txt - Notepad" with the following text:

```
File Edit Format View Help
S a S b
S e
```

This is automatically done by CYK in case you have entered a non-CNF grammar so you won't have to copy paste any output.

Another example for different CFG of the Language $\{a^n b^m, n, m \geq 1\}$



The screenshot shows a Windows command prompt window with the following text:

```
C:\Users\Danny\Desktop\Formal Language>java cyk
aaabb
###Parse Tree###
a a a b b
{A_0 } {A_0 } {A_0 } {B_0 } {B_0 }
{S A } {S A } {S } {S B }
{S A } {S } {S }
{S } {S }
{S }
String is part of the language
ab
###Parse Tree###
a b
{A_0 } {B_0 }
{S }
String is part of the language
a
###Parse Tree###
a
{A_0 }
String is NOT part of the language
```

Next to the command prompt is a Notepad window titled "grammar.txt - Notepad" with the following text:

```
File Edit Format View Help
S A B
A a A
B b B
A e
B e
```

Other observations

The program's scope is to solve a parsing problem. Furthermore, detail to user friendliness was disregarded. We strongly believe that every program should be user friendly and usable, but for this specific assignment it is out of scope.

The interface is a command line terminal. A file named grammar.txt is expected to be next to the executable (If the program is run in eclipse, grammar.txt should be on the project folder). The expected format of the grammar is CNF. Only rules of the form $A \rightarrow BC$ and $A \rightarrow a$ are accepted. If it is hard to manually convert a Grammar to CNF then and only then you should resort to inputting a CFG. In order to convert any rule $A \rightarrow BCDEF..N$ to a readable, by the computer, format, the user should remove the symbol \rightarrow and add a space between each nonterminal or terminal. If you must use an epsilon transition you will have to use a non-CNF input which is not encouraged. Epsilon transition is defined as "e", and the use of e as terminal on CNF is prohibited due to confusion. Nonterminals are Capital alphabet letters, terminals are lower case alphabet letters.

Please note that step five, for the conversion between CNF to CFG is commented and will result in error. We advise uncommenting each version and comparing the resulting Grammar to see if it is correct. We require about 1 more day to debug the code and fix everything which we don't have due to finals. Moreover step 5 was the responsibility of Boxiong Zhao who was not able to finish it. Also keep in mind the default acceptable inputted language is in CNF format. We tried to work hard and provide you with an astonishing program that would automatically convert formats.

Example of a correct grammar

S A B

A A A

A a

B B B

B b

The grammar above is the equivalent of the grammar

$S \rightarrow AB$

$A \rightarrow AA$

$A \rightarrow a$

$B \rightarrow BB$

$B \rightarrow b$

Example of **incorrect** grammar

S->AB // symbol -> is not readable.

A AA // lack of spaces between nonterminals

Aa //lack of space between terminal and nonterminal

B→BB //symbol → is not readable.

B bB // Avoid if possible, not in CNF

A e // e represents epsilon, avoid if possible, not in CNF

Even if the conversion of CFG to CNF could be considered out of scope we implemented it regardless to cover any case and make the program more usable to the professor. The conversion isn't always right and there are cases for which the conversion fails to return the equivalent of a Grammar. Also there are cases that the equivalent Grammar is returned correctly. However due to multiple nonterminals that point to other nonterminal the final symbol might sometime be present on the last cell of the CYK algorithm. Because the CYK algorithm only checks for S at the last cell, it ignores any other symbols and returns false. That is why we encourage you to determine if the string belongs to the language or not by looking at the CYK table.

The Grammar should only have S as the starting symbol. The nonterminal S should always be present in the Grammar file. If the nonterminal S is not present the algorithm will assume as starting symbol the first nonterminal found.

Currently known bugs are related to the conversion of CFG to CNF. We have debugged our way through to step 4 of the algorithm and was found to be valid. Step 5 is too hard to be implemented due to the difficulty of covering all the possible cases of that the left hand side nonterminal can appear. Future debugging includes test cases of grammars that have one nonterminal on the right side of the rule of the starting symbol ($S \rightarrow A$) and multiple A definitions ($A \rightarrow a$ and $A \rightarrow e$)

Project management

The tasks of the project were split so that everyone is doing something simultaneously. We

did so that we would finish faster. Few parts were correlated with each other so we had to host meetings to work on them together. We have hosted many team meetings with all the team members present. All team members have contributed equally based on the area of their strengths.

Boxiong Zhao

Implementation of step 4* and step 5 for the CFG to Chomsky Normal Form algorithm

Creation of test cases for the program and debugging

**The version of step 4 of Boxiong Zhao could not pass few test cases. His code remained in the file turned in. Another version was implemented by Iordanis Fostiropoulos*

Iordanis Fostiropoulos

Implementation of CYK algorithm

Implementation of step 3 and step 4 for the CFG to Chomsky Normal Form algorithm

Creation of test cases for the program and debugging

Project management and combining individual code

Weixin Wu

Creation of the input functions and the storage of the Grammar

Implementation of step 1 and step 2 for the CFG to Chomsky Normal Form algorithm

References

- [1] Cole, Richard. "Converting CFGs to CNF (Chomsky Normal Form)." - *Chomsky Normal Form*. New York University, 17 Oct. 2007. Web. 6 Apr. 2013.

<<http://www.cs.nyu.edu/courses/fall07/V22.0453-001/cnf.pdf>>.

[2] Lange, Martin, and Hans Leiß. "To CNF or Not to CNF? An Efficient Yet Presentable Version of the CYK Algorithm." Thesis. Ludwig-Maximilians-Universität München, Germany, 2009. *To CNF or Not to CNF? An Efficient Yet Presentable Version of the CYK Algorithm*. [Http://www.informatika-didactica.de](http://www.informatika-didactica.de), 29 June 2009. Web. 6 Apr. 2013.

Acknowledgements

StackOverflow.com was widely used for any problems encountered.

Wikipedia.com was used to study topics related to Context Free Languages and CYK algorithm

Special Acknowledgements to the PhD student, Ivan Bogun(ibogun2010@my.fit.edu) who helped with the debugging of the CYK algorithm

No code was copied or used from any other project. We studied many other projects that implemented CYK algorithms but they have been proven unhelpful or too complicated to implement or get any help from.