Cocke-Kasami-Younger ALGORITHM

A Python Implementation

University of Idaho

11/20/18

Koffi Anderson Koffi,

Matthew Mills

Contents

[Introduction 2](#_Toc530500663)

[Implementation 2](#_Toc530500664)

[Results 3](#_Toc530500665)

[Conclusion 4](#_Toc530500666)

[Appendix 5](#_Toc530500667)

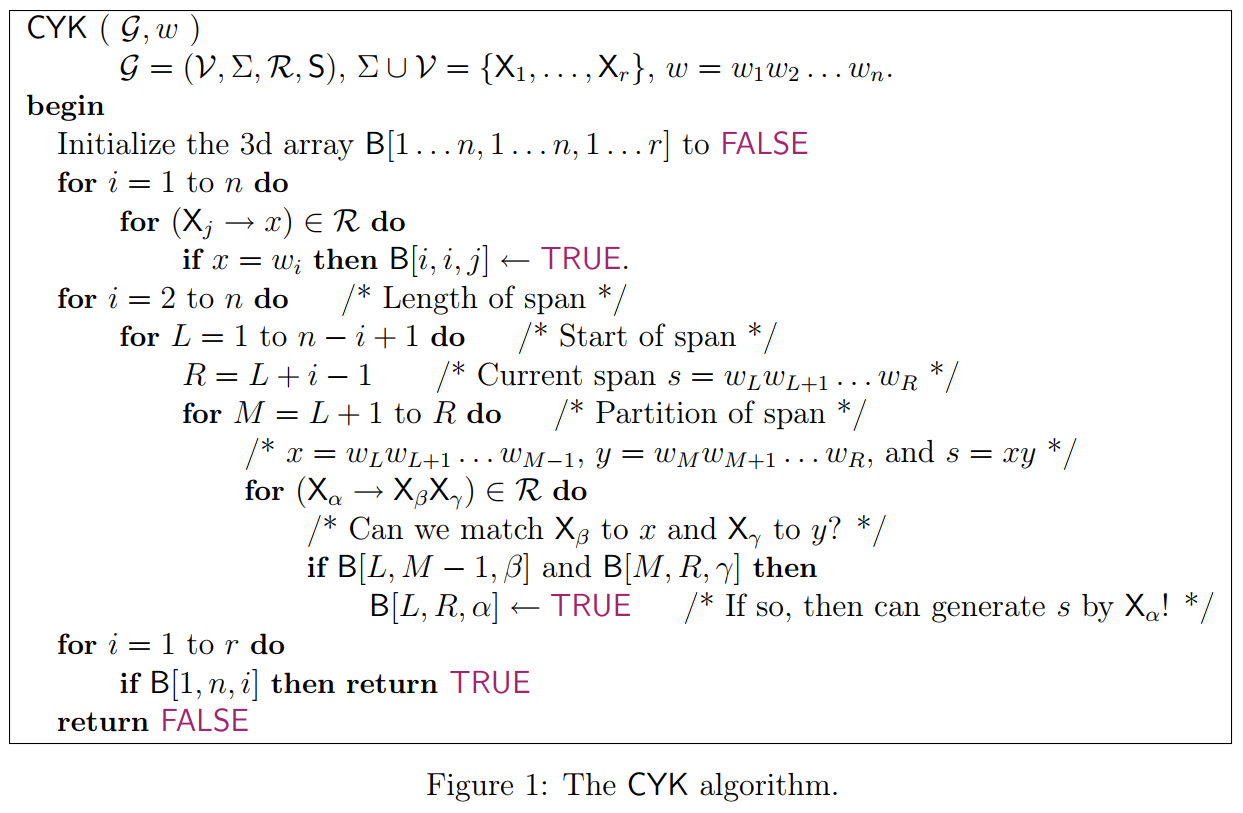
[References: 8](#_Toc530500668)

# Introduction

In this project, we will implement the Cocke-Kasami-Younger (CYK) Algorithm using the Python programming language. In fact, the CYK Algorithm is a membership algorithm for context-free grammars. Thus, using the CYK algorithm, it is possible to check whether a string is generated by the grammar of a context-free language.

# Implementation

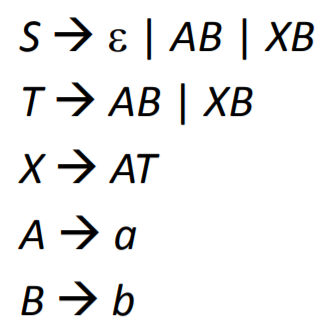
We implemented a version of the CYK algorithm using the python programming language and hosted the code source on github (see repository: <https://github.com/kandersonko/cyk_algorithm>). Our implementation required the grammar to be in Chomsky Normal Form. The version of the algorithm we implemented is the following:



Our code sources can be found in the appendix.

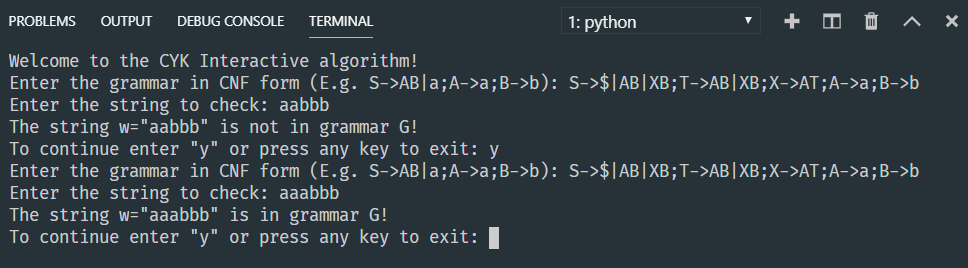
# Results

Using our implementation, we were able to check the membership of different string for the following grammar:



**Figure 2:** Grammar G

We found that the string w=”aabbb” is not in L(G), but the string w=”aaabbb” is in L(G) (see figure 3).



**Figure 3**: Output of an execution of the algorithm

This result agrees with our findings when we apply manually the algorithm.

This implementation can be downloaded on GitHub from the repository <https://github.com/kandersonko/cyk_algorithm> as a zip file and can be extracted into a folder “cyk\_algorithm”. Then, the code can be run on a computer with Python 3 installed by running on the following command on the command line “python main.py”. Also, it is necessary to be in the folder containing “main.py”. A simple “cd cyk\_algorithm” should suffice to set the working directory to the folder “cyk\_algorithm” containing the source code.

# Conclusion

In this project, we implemented the CYK Algorithm in Python programming language. Our implementation can correctly check the membership of a string in the language generated by a grammar. The CYK Algorithm is important because it can be applied to check whether a keyword in a programming language.

# Appendix

**main.py**

*from* grammar *import* Grammar

*from* CYKAlgo *import* CYKAlgo

def main():

print("Welcome to the CYK Interactive algorithm!")

command = "y"

*while*(command == "y"):

grammar\_text = input("Enter the grammar in CNF form (E.g. S->AB|a;A->a;B->b): ")

*if* grammar\_text == "": *break*

G = Grammar(grammar\_text.strip())

w = input("Enter the string to check: ")

cykAlgo = CYKAlgo(G)

*if* (cykAlgo.membership(w.strip())):

print("The string w=\"{}\" is in grammar G!".format(w))

*else*:

print("The string w=\"{}\" is not in grammar G!".format(w))

command = input("To continue enter \"y\" or press any key to exit: ")

print("Bye!")

*if* \_\_name\_\_ == '\_\_main\_\_':

main()

**gammar.py**

class Grammar(object):

def \_\_init\_\_(self, G):

""" \_\_init\_\_ takes a string G

and parses parses the productions into an array of productions

"""

*self*.rules = G.split(';');

*self*.productions = dict()

*for* rule in *self*.rules:

startVar = rule.split('->')[0]

varSet = rule.split("->")[1]

variables = [x *for* x in varSet.split('|') *if* x.isupper()]

terminals = [x *for* x in varSet.split('|') *if* x.islower()]

*self*.productions[startVar] = {"variables": variables, "terminals": terminals}

**CYKAlgo.py**

class CYKAlgo:

def \_\_init\_\_(self, G):

""" initilizes with the grammar G

"""

*self*.G = G

def membership(self, w):

B = dict()

X = dict()

V = [i *for* i in *self*.G.productions.keys()]

*for* k,v in enumerate(*self*.G.productions.keys()):

X[v]=k

n = len(w)

r = len(X)

*# case where S->a is the only production*

*if*(r==1 and w in *self*.G.productions[V[0]]["terminals"]):

*return* True

*# initialize all items in B to false*

*for* i in range(n):

*for* j in range(n):

*for* k in range(r):

B[i, j, k] = False

*# production A->a*

*for* i in range(n):

*for* j,v in enumerate(X):

*if* w[i] in *self*.G.productions[v]["terminals"]:

B[i, i, j] = True

*# production A -> BC*

*for* i in range(1, n):

*for* L in range(n-i+1):

R = L + i - 1

*for* M in range(L+1, R):

*for* v in range(r):

P=*self*.G.productions[V[v]]

variables = P["variables"]

*if*(len(variables)):

b, c = tuple(variables[0])

s, t = X[b],X[c]

*if*(B[L, M-1, s] and B[M, R, t]):

B[L, R, v] = True

r = n-1

*for* i in range(r):

*if*(B[r, n-1, i]):

*return* True

*return* False

# References:

“Lecture 15.” *Ethics and Engineering*, courses.engr.illinois.edu/cs373/sp2009/lectures/.