Trabalho Prático 2
Fundamentos Teóricos da Computação

Gustavo Torres Bretas Alves, Maria Fernanda Oliveira Guimarães

[1]Pontifícia Universidade Católica de Minas Gerais (PUC Minas)

[2]Instituto de Ciência Exatas e Informática

Abstract. *One of the most famous solutions to solve the problem of determining whether a sentence can be generated by a context-free grammar is the Cocke-Younger-Kasami algorithm, but in addition to this, an algorithm known as the modified Cocke-Younger-Kasami was also discovered and in this work it is possible to understand how the implementation of both was made and the following results.*

Resumo. *Uma das soluções mais famosas para resolver o problema de determinar se uma sentença pode ser gerada por uma gramática livre de contexto é o algoritmo Cocke-Younger-Kasami, mas além deste, um algoritmo conhecido como Cocke-Younger-Kasami modificado também foi descoberto e neste trabalho é possível entender como foi feita a implementação de ambos e os seguintes resultados.*

1. Introdução

O presente trabalho tem como objetivo mostrar a implementação e os resultados de dois métodos para verificar a pertinência de uma sentença a uma determinada linguagem livre de contexto. Para isso, foi usado o algoritmo Cocke-Younger-Kasami original e uma outra versão modificada.

2. Linguagem Livre de Contexto

De acordo com a teoria de linguagens formais, uma linguagem livre de contexto (LCC) é uma linguagem gerada por uma gramática livre de contexto (GLC). Portanto, como arquivo de entrada, o programa recebe um arquivo contendo a descrição de uma GLC e um conjunto de sentenças que serão usadas para verificar e gerar um outro arquivo como saída contendo os resultados.

3. Algoritmo Cocke-Younger-Kasami Original

O Algoritmo Cocke-Younger-Kasami (CYK) é usado para determinar se uma sentença pode ser gerada por uma determinada gramática livre de contexto, e se puder, como pode ser gerada, além disso, esse algoritmo exige que a gramática da linguagem esteja na forma normal de Chomsky. Esse é um dos algoritmos mais famosos para resolver esse problema.

3.1. Forma Normal de Chomsky

Para verificar se a frase pertence à gramática, é necessário, que ela esteja na forma normal de Chomsky, ou seja, uma variável produz duas variáveis ou um terminal.

```
# Remover variavel que deriva ela mesma
for rule in new_gram.rules_dict:
    for rul in new_gram.rules_dict[rule]:
        if rule == rul:
            new_gram.rules_dict[rule].remove(rul)
            pass
```

Figure 2. Remove regra que deriva ela mesma

Se após fazer a verificação for confirmado que a gramática não está na forma normal de Chomsky ela é convertida, para isso foram seguidos os seguintes passos:

1- Introduzir nova variável de partida

O primeiro passo foi introduzir uma nova variável inicial, para isso, foi criado uma nova gramática, chamada de ”new_gram”, que contém todas as regras da gramática antiga e foi adicionada uma nova variável de partida.

2- Remover λ-produções

Para remover os λ foram implementados dois laços de repetição *for*, sendo o primeiro responsável por percorrer cada variável da gramática e segundo percorre cada regra dessa variável. Ao encontrar um λ essa regra é removida. A única variável que pode possuir uma regra com λ é a variável inicial.

```
while new_gram.have_lambda():

    for rule in new_gram.rules_dict: # Para cada Variavel
        for rul in new_gram.rules_dict[rule]: # Para cada regra da variavel
            if rul == "λ":
                # Remover as regras
                if debug:
                    print("Removendo regra: {} -> {}".format(rule, rul))
```

Figure 1. Remove as regras que contém λ

Após remover o λ é necessário criar novas regras, para não perder a lógica da gramática, sendo assim, todos os símbolos que tinham na sua regra o símbolo que antes possuía o λ, receberam novos λ como regra.

Esse processo será repetido até que se o λ aparecer será apenas na variável inicial.

3- Remover produções unitárias

O próximo problema a ser tratado se refere aos símbolos que produzem variáveis únicas ou produz a si mesmo.

Para isso, primeiramente, passamos em todas as variáveis e por suas respectivas regras, se a variável possuir ela mesma como regra, isso será removido.

Depois, percorremos as variáveis novamente para encontrar as regras unitárias, visto que, segundo a forma normal Chomsky de uma variável produz duas variáveis.

Então, ao encontrar uma regra que deriva apenas uma variável, essa regra é substituída pela regra dessa variável. Para visualizar melhor esse cenário, segue abaixo um

exemplo:

Exemplo:

$A \rightarrow B$

$B \rightarrow DC \mid b$

$C \rightarrow c$

$D \rightarrow d$

Podemos observar que a variável A produz apenas uma variável, no caso o B, após converte-lá para a forma normal de Chomsky temos os seguinte resultado:

$A \rightarrow DC \mid b$

$B \rightarrow DC \mid b$

$C \rightarrow c$

$D \rightarrow d$

4- Converter regras remanescentes

Apenas remover as produções unitárias não é o suficiente para realizar a conversão completa, ainda há outros problemas que podem precisar ser resolvidos.

Um outro caso possível, é quando uma variável deriva mais de dois elementos, no caso a seguir três variáveis, então segue abaixo um exemplo deste caso:

Exemplo:

$A \rightarrow ASA \mid aB \mid a$

$S \rightarrow ASA \mid Ab$

$B \rightarrow b$

Após fazer a conversão para a forma normal de Chomsky as regras ficaram assim:

$S \rightarrow AX \mid UB \mid A$

$A \rightarrow b \mid AX \mid a$

$X \rightarrow SA$

$U \rightarrow a$

Então dessa forma, foi possível converter todas as gramáticas para a forma normal de Chomsky.

3.2. Implementação do algoritmo CYK

Para a implementação do algoritmo Cocke-Younger-Kasami (CYK), foi usado o pseudo código da imagem 3 disponível no artigo *"To CNF or not to CNF? An Efficient Yet Presentable Version of the CYK Algorithm"*. Esse algoritmo recebe como entrada uma gramática livre de contexto G na forma normal de Chomsky, que neste ponto, já foi tratada, e uma palavra W sobre o alfabeto G e como saída teremos uma tabela de reconhecimento T que contém, para cada sub palavra v de w, o conjunto de não terminais que derivam v, ou seja, suas propriedades sintáticas.

```
input: a CFG G = (N,Σ,S,→) in CNF, a word w = a1...an ∈ Σ+

CYK(G,w) =
1    for i=1,...,n do
2       Ti,i : = {A ∈ N | A→ ai}
3    for j=2,...,n do
4       for i=j–1,...,1 do
5          Ti,j : = ∅;
6          for h=i,...,j–1 do
7             for all A → BC
8                if B ∈ Ti,h and C ∈ Th+1,j then
9                   Ti,j : = Ti,j ∪{ A }

10   if S ∈ T1,n then return yes else return no
```

Figure 3. Pseudo código do algoritmo CYK

Após receber a gramática e a palavra que será testada, criamos uma tabela que será preenchida com as informações recebidas.

```python
# 1. Inicializar a tabela
n = len(word)
if n == 0:
    word = "λ"
    n = 1
table = [[[] for i in range(n)] for j in range(n)]
```

Figure 4. Inicializar a tabela

A tabela é preenchida da seguinte forma:

Supondo que recebemos a seguinte gramática G, já na forma normal de Chomsky:

S → AB

A → BB | a

C → AB | b

E queremos saber se a palavra w = aabba existe na linguagem L(G), então para isso, o algoritmo completa a tabela inicializada anteriormente.

Sabemos que o tamanho de w é 5 (|w| = 5), então a tabela triangular terá 5 colunas e 5 linhas.

```
interation = 0
for j in range(n):
    for rule in gram.rules_dict:
        for rul in gram.rules_dict[rule]:
            interation += 1
            # print("Interation: {} of {}".format(interation, maxInterations))
            if len(rul) == 1:
                if rul == word[j]:
                    table[j][j].append(rule)
```

Figure 5. Preencher a tabela

Aplicando o código acima, a tabela preenchida ficará da seguinte forma:

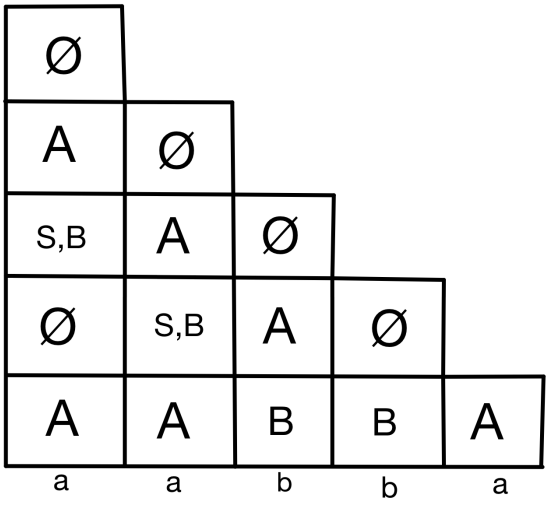

Figure 6. Tabela preenchida

Para a palavra w pertencer a L(G) é necessário que a variável inicial apareça na primeira posição da tabela, como isso não ocorreu, pode-se afirmar que essa palavra w não pertence a linguagem.

Um outro exemplo com uma nova palavra w = baaa, com a gramática abaixo:

$S \rightarrow AB \mid BC$

$A \rightarrow BA \mid a$

$B \rightarrow CC \mid b$

$C \rightarrow AB \mid a$

Como $|w| = 4$, teremos a seguinte tabela:

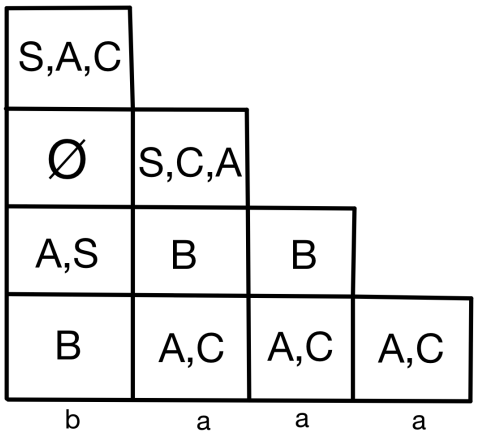

Figure 7. Tabela preenchida

Como a variável inicial, que neste caso é S aparece na primeira posição da tabela, podemos afirmar que a palavra w pertence a linguagem.

Para fazer isso, transformamos o pseudo código da figura 3 no seguinte algoritmo:

```
for l in range(1, n):
    for i in range(n-l):
        j = i + l
        for k in range(i, j):
            for rule in gram.rules_dict:
                for rul in gram.rules_dict[rule]:
                    tqdmProgress.update(1)
                    if len(rul) == 2:
                        for a in table[i][k]:
                            for b in table[k+1][j]:
                                if rul == a + b:
                                    table[i][j].append(rule)

tqdmProgress.set_description_str("`{}` aceita? {}".format(word, gram.start in table[0][n-1]))
tqdmProgress.close()

if gram.start in table[0][n-1]:
    return True
else:
    return False
```

Figure 8. Algoritmo CKY

Com isso, conseguimos afirmar, após receber uma palavra w e uma gramática livre de contexto, se w pertence ou não a linguagem.

4. Algoritmo Cocke-Younger-Kasami Modificado

O Algoritmo Cocke-Younger-Kasami Modificado foi uma proposta apresentado por Lange e Leiß em 2009, essa solução é aplicada a uma forma normal binária, ou seja, diferente do CYK original, este não precisa receber uma gramática na forma normal de Chomsky, esta pode receber uma gramática no formato da segunda forma normal (2NF).

4.1. Segunda Forma Normal - 2NF

A 2NF é baseada no conceito de dependência funcional completa, ela se aplica a relações com chaves compostas, ou seja, relações com uma chave primária composta por dois ou

mais atributos.

Para fazer a conversão de uma gramática qualquer para o formato 2NF, primeiramente, conferimos se a gramática não está neste formato, se não estiver, entramos em um loop até que esteja.

Então temos dois laços *for* que passam por cada regra e por cada produção dessa regra. Em seguida é conferido se a produção possui mais de dois símbolos, se sim, esses dois símbolos são guardados e a variável *"gerador"* recebe *"False"*.

Logo após, é feito a contagem de quantas produções tem na regra, e conferimos se a regra tem apenas uma produção e se a produção é igual aos dois primeiros símbolos, se sim, a variável geradora recebe essa regra.

Por fim, se a variável geradora não for encontrada, pegamos a próxima letra do alfabeto, mas se for encontrada, o terminal é substituído por ela.

```python
# Converter gramática para 2NF
def convertTo2NF(gram):
    if gram.is_2nf():
        return gram

    new_gram = gram.copy()
    loop = 0
    while not new_gram.is_2nf():
        loop += 1
        if loop > 100:
            print("Erro ao converter para 2NF")
            break
        for rule in list(new_gram.rules_dict):
            for rul in new_gram.rules_dict[rule]:
                if len(rul) > 2:
                    if debug:
                        print("Removendo regra: {} -> {}".format(rule, rul))

                    twoFirst = rul[:2]
                    gerador = False
                    for rule2 in new_gram.rules_dict:
                        rulesInRule2 = len(new_gram.rules_dict[rule2])
                        if rulesInRule2 == 1:
                            if new_gram.rules_dict[rule2][0] == twoFirst:
                                gerador = rule2
                                break

                    if not gerador:
                        gerador = getNextAlphabet(new_gram)

                        new_gram.add_variable(gerador)
                        new_gram.add_rule([gerador, twoFirst])

                    if gerador:
                        if debug:
                            print("Removendo regra: {} -> {}".format(rule, rul))
                        # replace terminal to gerador in rule
                        new_gram.rules_dict[rule].remove(rul)
                        rulTmp = rul.replace(twoFirst, gerador)

                        new_gram.add_rule([rule, rulTmp])
```

Figure 9. Função que faz a conversão da gramática para 2NF

4.2. Implementação do algoritmo CYK Modificado

Assim como no CYK original, para a implementação do algoritmo CYK modificado, foi usado o pseudo código da imagem 10 disponível no artigo *"To CNF or not to CNF? An Efficient Yet Presentable Version of the CYK Algorithm"*.

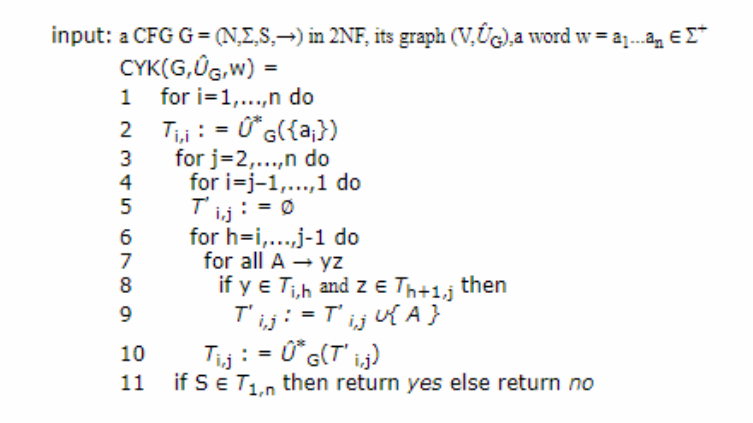

Figure 10. Pseudo código do algoritmo CYK Modificado

Comparando as figuras 3 e 10 podemos afirmar que os dois algoritmos se diferem minimamente, apenas duas operações são diferentes, a primeira é a entrada da tabela e a segunda é que diferente do CYK original o modificado pressupõe que a palavra de entrada w não está vazia.

Após fazer a conversão para 2NF e a implementação do pseudo código acima, foi possível determinar se a sentença pertence ou não a linguagem.

O algoritmo é iniciado com a entrada de uma gramática, um símbolo inicial e uma sequência de símbolos (palavra). Ele cria uma tabela bidimensional de tamanho N x N, onde N é o tamanho da palavra de entrada. Em seguida, preenche a diagonal principal da tabela com os símbolos terminais da palavra de entrada. Em seguida, itera sobre os elementos da tabela, começando pela segunda coluna e trabalhando sua forma para a primeira coluna, preenchendo os elementos restantes da tabela com os símbolos não-terminais da gramática que podem ser derivados das combinações de símbolos nas posições adjacentes da tabela. Por fim, o algoritmo verifica se o símbolo inicial está presente na última linha da tabela e retorna **True** se estiver, indicando que a palavra pertence à gramática, ou **False** caso contrário.

5. Experimentos

Ao compilar o programa, é necessário escolher um dos arquivos de entrada para testar se a palavra pertence a gramática dada. Já foram disponibilizados 7 arquivos de entradas, mas o usuário também pode criar um novo e adicionar a pasta inputs, sendo necessário, apenas seguir a seguinte estrutura:

```
VARIAVEL1(START),VARIAVEL2,VARIAVEL3...
TERMINAL1,TERMINAL2,TERMINAL3...
VARIAVEL:PRODUCAO1|PRODUCAO2|PRODUCAO3...
ENTRADAS
SENTENCA1
SENTENCA2
SENTENCA3...
FIM
```

Figure 11. Estrutura necessária para criar um arquivo de entrada

Nas imagens abaixo é possível encontrar alguns exemplos de arquivos de entrada que foram utilizados na implementação.

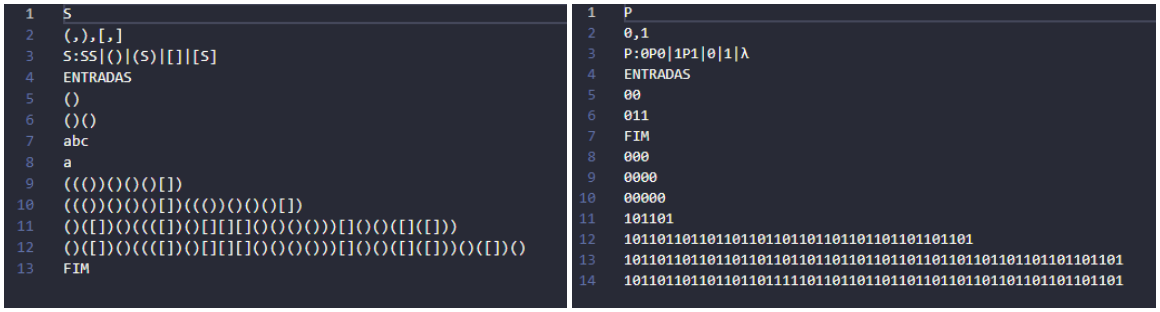

Figure 12. Arquivos de Entrada 1 e 2, respectivamente

Todas entradas recebidas são convertidas para a forma normal de Chomsky e para a Segunda Forma Normal, e consequentemente, usamos os algoritmos CYK original e o CYK modificado para testar se a palavra presente no arquivo de entrada existe na gramática, juntamente com isso, também informamos os tempos de conversão e de execução.

Resultados obtidos por meio do arquivo de entrada 1, presente na figura 12:

Algoritmo CYK original:

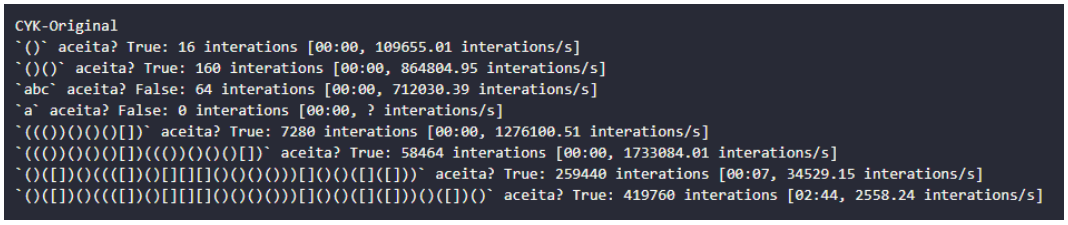

Figure 13. Resultado do CYK original

Algoritmo CYK modificado:

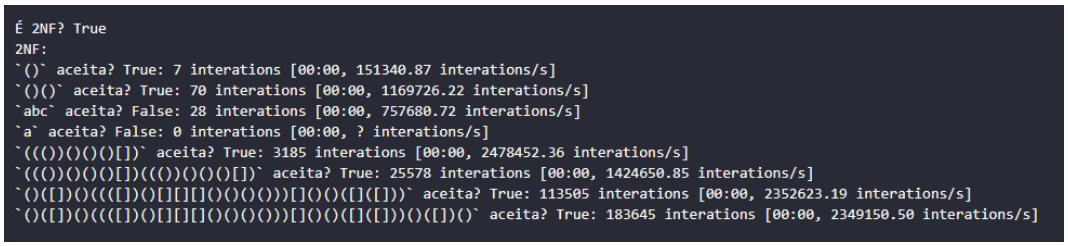

Figure 14. Resultado do CYK modificado

Tempo de conversão e execução:

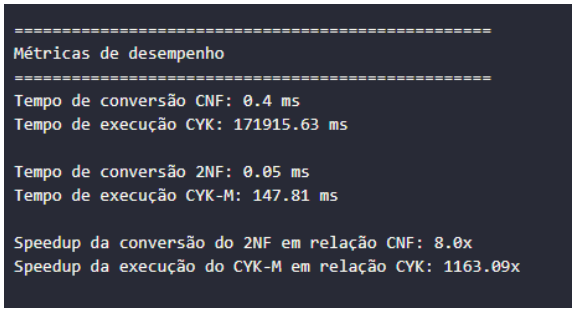

Figure 15. Resultado dos tempos gastos

Resultados obtidos por meio do arquivo de entrada 2, presente na figura 12:

Algoritmo CYK original:

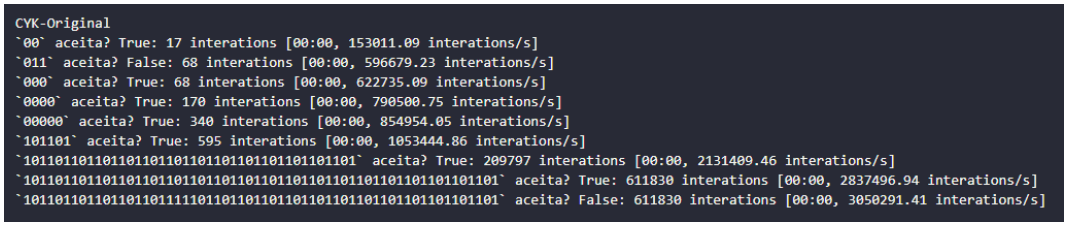

Figure 16. Resultado do CYK original

Algoritmo CYK modificado:

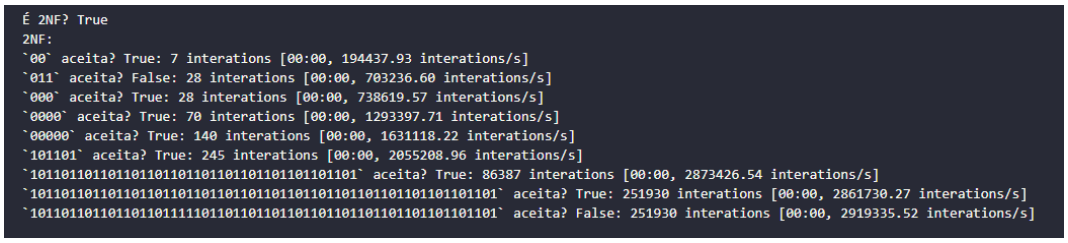

Figure 17. Resultado do CYK modificado

Tempo de conversão e execução:

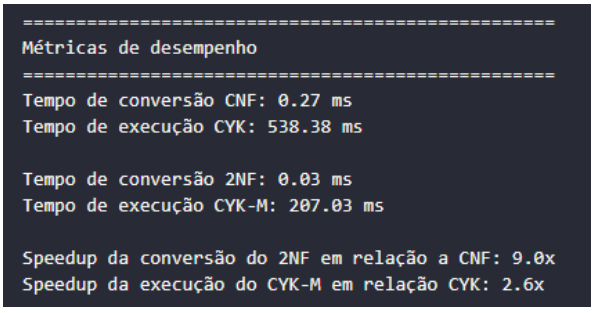

Figure 18. Resultado dos tempos gastos

6. Resultados

Após realizar vários testes, nos algoritmos Cocke-Younger-Kasami Original e no Cocke-Younger-Kasami Modificado com várias gramáticas e palavras diferentes, pode-se afirmas que ambos são formar eficazes de determinar de uma sentença pode ser gerada por uma certa gramática livre de contexto, no caso do original, na forma normal de chomsky e a modificada usando a segunda forma normal, porém, após analisar os resultados, percebemos que converter a gramática para o formato 2NF é mais rápido do que converter para a forma normal de chomsky, além disso, o algoritmo modificado, proposto em 2009, também é mais eficiente que o original.

7. Referencias

[Martin Lange 2009], [Kymberlly Melo 2018], [EducationAboutStuff 2016], and [Dimitroff 2013].

References

Dimitroff, N. (2013). Cfg solver. github.

EducationAboutStuff (2016). *CYK Algorithm Made Easy (Parsing)*. YouTube.

Kymberlly Melo, H. J. (2018). Cyk modificado. github.

Martin Lange, H. L. (2009). *To CNF or not to CNF? An Efficient Yet Presentable Version of the CYK Algorithm*. Ludwig-Maximilians-Universität München, Germany.