Repositorio:

https://github.com/Gerax5/TC_Proyecto2.git

Documentación:

Diseño de la aplicación:

No se utilizó un método de modelación especifico simplemente se crearon clases las cuales cumplieran con las funciones. Fueron creadas tres clases que realizan tareas específicas para la ejecución del proyecto.

Las clases que existen son:

1. Normalización del CFG: Toma como parámetros para el constructor, una gramática y un símbolo inicial. Esta clase se encarga de Eliminar las producciones épsilon, las producciones unitarias y los caracteres inútiles.
2. Normalización de Chomsky: Toma como parámetro para el constructor, una gramática y un símbolo no terminal con el cual serán creadas las nuevas producciones. Esta clase se encarga de aplicar la normalización de Chomsky a la gramática ya normalizada. Las nuevas producciones nuevas se crean con el símbolo enviado.
3. Algoritmo CYK: Toma como parámetro una gramática normalizada de Chomsky y un símbolo inicial. Esta clase se encarga de determinar si una palabra pertenece a la gramática.
    a. Esta clase contiene una función llamada parser la cual se puede llamar con distintas palabras para verificar varias palabras si pertenecen a la gramática.
    b. Función generateTree:a

Todo esto es consumido por un archivo principal el cual lee un archivo del cual se obtiene la gramática. El primer símbolo encontrado en la gramática será tomado en cuenta como el símbolo inicial. Una vez aplicada la normalización de Chomsky se muestra la gramática y una vez aplicado el algoritmo de CYK se muestra la tabla con la respuesta.

Discusión

La parte más difícil de este proyecto fue la implementación de la normalización del CFG y la normalización de Chomsky. Para la construcción de este proyecto, fue tomada en cuenta solo gramáticas las cuales cuentan con solo un símbolo del lado izquierdo. Así se puede realizar un análisis sobre una gramática sencilla. Todos los algoritmos fueron hechos primero en papel para tener un mejor entendimiento de esto. Con la aplicación se agregaran los seudocódigos que se tomaron en cuenta para la elaboración.

Se tomo en cuenta la siguiente gramática:

S→0A0|1B1|BB

A→C

B→S|A

C→S|ε

Primero se tiene que encontrar símbolos anulables, en este caso se encuentra la C, luego mediante un while se vuelve a recorrer todo encontrando que A, C también son, como se agregaron nuevos anulable y así sucesivamente se vuelve a repetir el proceso hasta encontrar todos S,A,B,C.

Primero paso eliminar producciones épsilon:

Proceso:

Begin

Change = True

Nullables = {}

While Change:

    For NoTerminales

    Change = False

    Si NoTerminal.productions tiene ε

        Nullable.push(NoTerminal)

        Change = True

End

Una vez encontrados los anulables se revisan las producciones y se hacen casos donde los caracteres nullables estén o no estén. Haciendo combinaciones por cada producción.

Proceso:

Begin

For each NoTermianl

For each Producciones

    Si algún nullable está contenido en Producciones

        positionToRemove = Se encuentra la posición de los caracteres nullable

        Combinaciones = []

        For each PosiitonToRemove

            comb = Se hacen las combinaciones

            Se hacen mas combinaciones apartir de “comb” con todos los caracteres nullable

Combinaciones.push(nuevasCombinaciones)

Si alguna producción es ε eliminarla

End

Segundo Paso Eliminar producciones unitarias

Para las producciones unitarias. Empezando con la primera producción encontrada se empieza a buscar la forma: A → A. Desde la primera producción hasta la última.

Proceso:

Begin

For each NoTermianl

actualNoTerminal = current NoTerminal

For each Producciones

actualProd = current Produccion

Si alguna produccion de actualProd tiene length = 1 y es upper()

For each producción de este NoTerminal Encontrado

Agregar la producción a actualNoTerminal

// Ojo para cada producción se tiene que hacer una verificación. Primero, el No Terminal encontrado no tiene que haber sido visitado antes y Si el no Terminal genera otra producción Unaria este proceso se tiene que repetir.

End

Tercer Paso Eliminar símbolos inútiles

Para eliminar los símbolos inútiles, primero se tiene que verificar que sea un símbolo generador es decir que pueda generar un terminal

Proceso:

Begin

Generadores = []

For each NoTerminal

For each Producciones

Si en alguna de la producciones tiene solo lower() o number()

Generadores.push(NoTerminal)

Luego desechar de la gramática todos los no terminales que no estén en generadores

Posteriormente se tiene que buscar los símbolos que sean alcanzables desde el no terminal inicial

Alcanzables = []

Símbolo inicial “S”

For each producción de S

Si la producción contiene un Upper

Alcanzable.push(del No Terminal encontrado)

// Ojo para cada no Terminal encontrado también se tienen que visitar los otros para obtener todos los alcanzables

END

Por ultimo se desecha de la gramática todos los no Terminales y producciones que no estén en el array alcanzables

Luego de la normalización, la gramática modelo llega a esta configuración:

S→0A0|1B1|BB|11|00

A→0A0|1B1|BB|11|00

B→0A0|1B1|BB|11|00

Para la forma normal de Chomsky se tienen que tomar en cuenta 3 Casos.

1. Cuando la producción es de longitud 2
    a. Cuando longitud = 2 se revisa carácter de la izquierda y de la derecha si uno de los dos es un terminal se cambia por el nuevo generado
2. Cuando la producción tiene longitud > 2.
    a. Para los de longitud > 2, se tiene que ver el primer símbolo, si es terminal se cambia por un nuevo no terminal y si es no terminal se deja así y luego se procesa el resto.
        i. Si el resto es longitud = 2 eso significa que el proceso ya acabo, solo se tiene que hacer la misma verificación que en longitud 2.

    ii. Si el resto es longitud > 2 se tiene que hacer nuevamente el proceso de obtener el primero y luego revisar el resto
3. Cuando la producción tiene longitud 1
    a. son producciones terminales que cumplen con la forma normal de Chomsky. Debido a que previamente se arreglaron las producciones unitarias

En el caso de esta gramática se encuentra 0A0 es longitud > 0

Proceso:

Begin

First = 0

Al ser un no terminal se tiene que cambiar por un nuevo no terminal X0 → 0

First = X0

Rest = resto del string

Para ir gurdando el resto se crea un nuevo no terminal y hacemos que apunte al resto

NewNoTerminal = X1

X1 → A0

Como es longitud 2, se revisan los dos caracteres:

1. A es no terminal se sigue
2. 0 es terminal, se verifica si ya existe un nuevo no terminal que apunte a este terminal y como si existe se cambia por X0

Por lo que,

Entrada: S → 0A0

Resultaría en:

S → X0X1

X1 → AX0

END

Repitiendo este proceso se llegaría a esta nueva configuración la cual ya está normalizada con Chomsky.

X0 → 0

X1→1

X2→BX1

X3→AX0

S→ X0X3|X1X2|BB|X0X0|X1X1

A→X0X3|X1X2|BB|X0X0|X1X1

B→ X0X3|X1X2|BB|X0X0|X1X1

CYK

Para la construcción del CYK de igual manera se realizo a mano pero este tuvo que ser modificado para poder monejarlo de mejor forma con arrays. Se crea una tabla de longitud de la palabra que se quiere y esta se comienza a llenar fila por fila hasta encontrar que en la primera fila se encuentra S, si se encuentra es que es una palabra que pertenece al lenguaje.

La primera fila de esta tabla son las letras separadas y cada fila en adelante son los no terminales que producen combinaciones de estos en orden, fila 1 son los no terminales que producen el no terminal de la fila. Fila 2 es combinación de estos, cada fila el largo que hay que ir agarrando es de longitud n.

Construcción

Begin

Palabra = “0 0 1 1”

La gramática usada es la creada anteriormente

Se crea una tabla del tamaño de la palabra

0	0	1	1

En la fila 1 van los no terminales que producen el terminal

0	0	1	1
X0	X0	X2	X2

En la fila 2 los van los no terminales que producen parejas seguida de estos es decir: (0,0),(0,1),(1,1)

0	0	1	1
X0	X0	X2	X2
B,S,A		B,S,A	

Si no existen no terminales que puedan producir estas parejas solo se deja un espacio en blanco o un “-“ para indicar que es una posición vacia. La siguiente fila son los no terminales que puedan producir tríos

0	0	1	1
X0	X0	X2	X2
B,S,A		B,S,A	

X3			

En la fila 4 son los no terminales que pueden hacer la construcción completa, tomando en cuenta los no terminales ya puestos con anterioridad.

0	0	1	1
X0	X0	X2	X2
B,S,A		B,S,A	
X3			
B,S,A			

Por último, si S está en esta última iteración, quiere decir que se puede llegar a obtener esta palabra a partir de S, lo que hace que la palabra pertenezca al lenguaje.

END

Esta es una construcción básica con una gramática sencilla. Ahora solo se tienen que aplicar para gramáticas más complejas con mas caracteres del lado izquierdo.

Obstáculos encontrados, al construir la forma normal del Chomsky es complicada la aplicación cuando las cadenas son de largo > 2. Porque es un proceso cíclico con el cual hay que tener cuidado de no generar nuevas producciones inútiles por lo que hay que estar en constante revisión. Esta transformación me dio mucho problema porque se generaban muchas producciones inútiles y sin sentido. Además, se generaban muchas producciones repetidas. Por lo que, hay que tener en consideración estas cosas para poder aplicar bien este algoritmo.

También, hubo obstáculos en las lecturas más avanzadas como VP → VP P P, se tuvo que cambiar la lectura y se asumió que las gramáticas venían en esa forma con un espacio de por medio.

Se Recomienda que los algoritmos se hagan primero a papel para tener un mejor entendimiento sobre el tema. Además, si se quiere utilizar inteligencia artificial para solucionar cosas, es recomendable tener un modelo propio ya hecho con las especificaciones porque usualmente esta falla y no da un buen resultado. También formular bien los props para que esta no se equivoque y haga algo que no tiene.

Ejemplos Realizados:

Ejemplo 1:

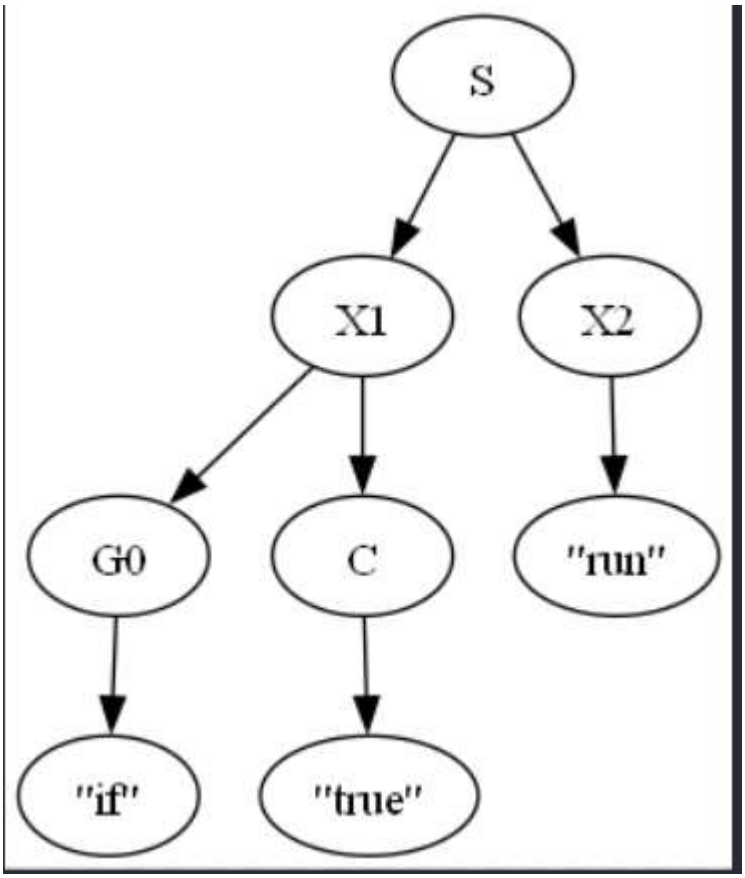
Entrada:

```
S    → X1 X2
X1   → if C
X2   → X3 else S | ACTION
X3   → then S
C    → true | false
ACTION → A
A    → run | stop
```

Palabra:

```
string = "if true stop"
```

Resultado:



Ejemplo 2:

Entrada:

```
S   → NP VP
VP  → VP PP | V NP | cooks | drinks | eats | cuts
PP  → P NP
NP  → DET N | he | she
V   → cooks | drinks | eats | cuts
P   → in | with
N   → cat | dog | beer | cake | juice | meat | soup | fork | knife | oven | soup
DET → a | the
```

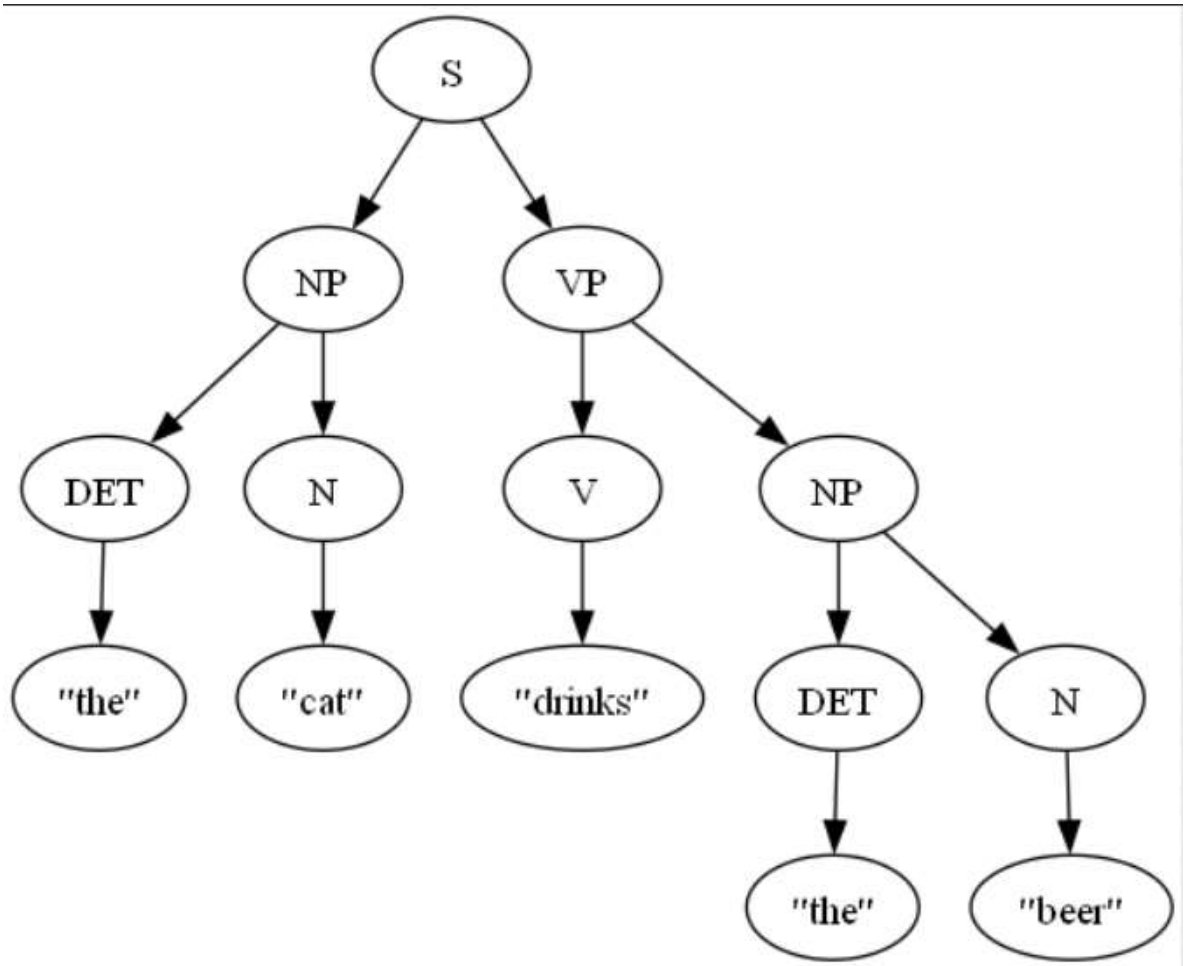
Palabra:

```
string = "the cat drinks the beer"
```

Resultado:

```
Gramática I:
{'S': [['NP', 'VP']], 'VP': [['VP', 'PP'], ['V', 'NP'], ['cooks'], ['drinks'], ['eats'], ['cuts']], 'PP': [['P', 'NP']], 'NP': [['DET', 'N'], ['he'], ['she']], 'V': [['cooks'], ['drinks'], ['eats'], ['cuts']], 'P': [['in'], ['with']], 'N': [['cat'], ['dog'], ['beer'], ['cake'], ['juice'], ['meat'], ['soup'], ['fork'], ['knife'], ['oven'], ['soup']], 'DET': [['a'], ['the']]}
Gramatica Transformada:
S → NP VP
VP → V NP | drinks | cuts | eats | VP PP | cooks
PP → P NP
NP → he | DET N | she
V → drinks | cooks | cuts | eats
P → with | in
N → juice | knife | beer | cat | meat | oven | dog | cake | fork | soup
DET → the | a
Tiempo: 0.0614 ms
('S', ('NP', ('DET', 'the'), ('N', 'cat')), ('VP', ('V', 'drinks'), ('NP', ('DET', 'the'), ('N', 'beer'))))
Tabla CYK:
   the       cat       drinks      the       beer
  -----     -----      -----      -----     -----
   DET       N         V,VP       DET       N
    NP                              NP
    S                   VP

    S
La cadena 'the cat drinks the beer' pertenece al lenguaje.
```



Palabra mal ingresada:

```
string = "the cat drinks the the"
```

Resultado:

```
Tabla CYK:
   the        cat        drinks      the        the
  -----      -----      -----      -----      -----
   DET         N          V,VP       DET        DET
    NP
    S


La cadena 'the cat drinks the the' NO pertenece al lenguaje.
```

Ejemplo 3:

Entrada:

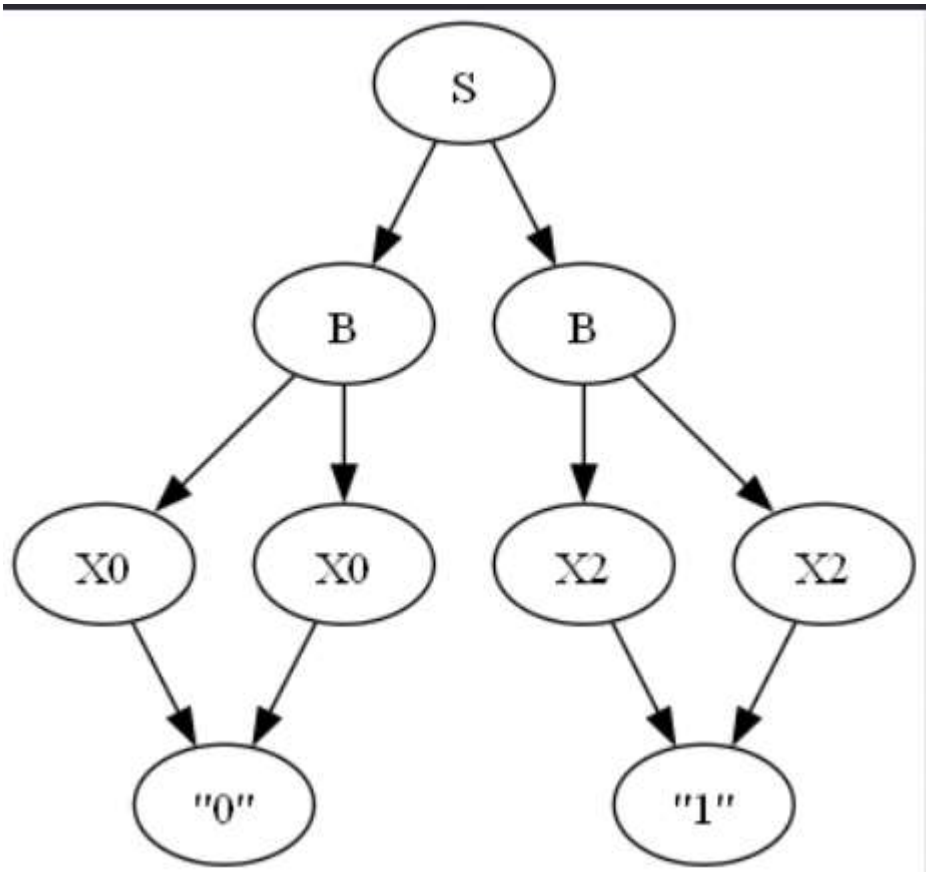
```
S   → 0 A 0 | 1 B 1| B B
A   → C
B   → S | A
C   → S | ε
```

Palabra:

```
string = "0 0 1 1"
```

Resultado:

```
Gramática 1:
{'S': [['0', 'A', '0'], ['1', 'B', '1'], ['B', 'B']], 'A': [['C']], 'B': [['S'], ['A']], 'C': [['S'], ['ε']]}
Gramatica Transformada:
X0 → 0
X1 → A X0
X2 → 1
X3 → B X2
S → X2 X2 | X0 X0 | X0 X1 | B B | X2 X3
A → X2 X2 | X0 X0 | X0 X1 | B B | X2 X3
B → X2 X2 | X0 X0 | X0 X1 | B B | X2 X3
Tiempo: 0.0690 ms
('S', ('B', ('X0', '0'), ('X0', '0')), ('B', ('X2', '1'), ('X2', '1')))
Tabla CYK:
    0          0          1          1
  -----      -----      -----      -----
    X0         X0         X2         X2
  B,S,A                 B,S,A
    X3
  B,S,A
La cadena '0 0 1 1' pertenece al lenguaje.
```


S
B
B
X0
X0
X2
X2
"0"
"1"