Universidad del Valle de Guatemala
Teoría de la Computación
Gabriel Brolo

PROYECTO 2

Integrantes:

- Diego Andres Alonzo Medinilla 20172
- Samuel Argueta 211024

Discusión

Para el desarrollo de este proyecto, se trató de hacerlo lo más ajustado a las instrucciones, pero cabe resaltar que a pesar de ello, se considera que no todo salió bien. En este caso, se implementaron varias gramáticas tanto de las clases presentadas, como de cortos como de presentaciones que se encontraron

Cabe resaltar que entre los obstáculos que se presentaron en la resolución del proyecto, fueron con respecto a la implementación del CFG, porque la implementación fue muy compleja, y se tuvieron estos problemas:

a. La eliminación de los epsilón debido a que esta generaba una especie de permutación, suponiendo una regla que sea A->ε|a y suponiendo algo como S->ASASA|ε se tendría que generar algo como SASA, ASSA, ASS…A,S.
b. Eliminar las producciones inútiles dado que se tenía que determinar si generaban y hay algunas que tenían producciones dependientes de otras reglas que no se habían determinado si generaban o no.
c. Finalmente, la última dificultad afrontada fue la recursividad ya que fue de las últimas cosas en implementarse y es debido a que se debía de eliminar que se hacía un poco complejo el programa sin embargo, menos complejo que los anteriores.

El mayor problema que se presentó fue el CYK y es que a pesar que se implementó y se trató realmente presentaba varios problemas. Así mismo, hubo casos en donde cumplía con la identificación de ciertas oraciones, pero había otras, que aún perteneciendo a tal gramática, no las reconocía. Con esto se tuvieron varias formas de implementar cyk, entre ellas, los resultados persistían, pues había veces donde si reconocía ciertas gramáticas y sus oraciones, pero otra vez, no reconocía todas las oraciones, aún perteneciendo y, tampoco funcionaba con todas las gramáticas.

Entre las funcionalidades que posee el proyecto, está la graficación de las derivaciones, o del parse tree. En este caso se utilizó graphviz para poder tenerlo de

manera visual. En este caso, se pudo implementar de una mejora manera el cyk junto al parse tree, resolviendo los problemas antes planteados.

Entre las recomendaciones, se debe destacar el conocimiento perfecto de los algoritmos y de cómo funciona la simplificación de gramáticas. De manera que es imprescindible que se entienda bien cómo realizar una simplificación de gramáticas, la eliminación de epsilons o en qué consiste Chomsky. A su vez se recomendaría que para futuros proyectos se logrará modificar el código para que en vez de que sea recursivo se utilice una pila para convertir todo lo recursivo a algo lineal en el caso de la conversión a Chomsky y todo lo que esto implica. Finalmente, se recomienda que se entienda a la perfección los pasos y en qué consiste el algoritmo de CYK.

Diseño de la aplicación

- **main.py**
  - *main:*
    - Método donde se ejecuta el programa.
- **preparation.py**
  - *validacion_gramatica:*
    - Método para validar si se ingresa una gramática válida, tanto de manera manual, como una gramática directa
  - *arreglar_gramática:*
    - Es un método que tiene como objetivo convertir una lista de reglas en la gramática a trabajar en formato de diccionario.
  - *cargar_gramatica:*
    - Lectura del archivo donde se encuentra la gramatica
- **CFG.py**
  - *eliminar_recursividad:*
    - Es un método que elimina la recursividad dada una gramática y unos no terminales brindados
  - *recursividad_parejas_unarias:*
    - Es un método que determina las parejas unarias de una gramática pero aquellas que se obtienen mediante la transitividad.
  - *crear_parejas_unarias:*
    - Es un método que crea todas las posibles parejas unarias para eliminar las producciones unitarias.
  - *remover_producciones_unitarias:*
    - Método que sirve para eliminar producciones unarias.
  - *index_ocurrencia_elemento:*

        - Es una función que sirve para determinar en qué posición de una lista se encuentra la enésima iteración sirve para la eliminación de epsilons.
    - *generar_producciones_epsilon:*
        - Es una función que crea todas las posibles permutaciones que se generan al eliminar una producción epsilon.
    - *eliminar_producciones_epsilón:*
        - Es una función que elimina las producciones epsilon.
    - *es_alcanzable:*
        - Determina si un no_terminal es alcanzable.
    - *prod_util:*
        - Determina si una producción es útil.
    - *produccion_es_util*
        - Identificar que la producción es util
    - *remover_producciones_inutiles:*
        - Función que determina y elimina las producciones utiles y alcanzables.
    - *conversion_chomsky:*
        - Convierte la gramática dada a chomsky.
    - *otro_formato:*
        - Método para cambiar la forma en que se presenta la gramática en particular.
- **cyk_parse_tree.py**
    - *cyk_with_parse_tree:*
        - Implementación del algoritmo cyk.
    - *build_tree:*
        - Función que se utiliza para construir el árbol del parser sobretodo los nodos.
    - *construct_parse_tree:*
        - Función para construir el árbol del parser dependiendo de si la tabla es aceptable.
    - *write_node:*
        - Función para escribir el nodo manualmente en lenguaje graphviz
    - *build_graphviz_tree:*
        - Genera el árbol en graphviz.
    - *validacion_sentencias:*
        - Este método tiene como objetivo la validación de oraciones, indicando si tal oración pertenece o no en la gramática.
- **print.py**
    - *imprimir_gramática:*

        - Imprime una gramática en un formato en específico.
    - *print_tree:*
        - Imprime el árbol sintáctico según los niveles
- **gramaticas**
    - Es una carpeta que almacena diversas gramáticas que probamos y que pasaron cada fase del proyecto, incluyendo aquellas que no pueden llegar a forma normal de chomsky.

Funcionamiento del programa

https://youtu.be/K9-8K5hEaeY

Pruebas

Gramática brindada en la hoja

```
Reglas a trabajar:
 S->NP VP
 VP->VP PP
 VP->V NP
 VP->cooks|drinks|eats|cuts
 PP->P NP
 NP->Det N
 NP->he|she
 V->cooks|drinks|eats|cuts
 P->in|width
 N->cat|dog
 N->beer|cake|juice|meat|soup
 N->fork|knife|oven|spoon
 Det->a|the
```

```
Gramatica arreglada:
    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|width
    N -> cat|dog|beer|cake|juice|meat|soup|fork|knife|oven|spoon
    Det -> a|the
No terminales:['PP', 'V', 'N', 'Det', 'S', 'NP', 'VP', 'P']
Terminales:['cuts', 'juice', 'fork', 'knife', 'beer', 'cat', 'she', 'eats', 'cooks', 'a', 'dog', 'spoon', 'he', 'oven', 'width', 'in', 'the', 'drinks', 'meat', 'soup', 'cake']
```

```
Gramatica sin producciones epsilon:
    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|width
    N -> cat|dog|beer|cake|juice|meat|soup|fork|knife|oven|spoon
    Det -> a|the
```

```
recursivos:{'VP': ['VP PP']}
betas:{'S': ['NP VP'], 'VP': ['V NP', 'cooks', 'drinks', 'eats', 'cuts'], 'PP': ['P NP'], 'NP': ['Det N', 'he', 'she'], 'V': ['cooks', 'drinks', 'eats', 'cuts'], 'P': ['in', 'width'], 'N': ['cat', 'dog', 'beer', 'cake', 'juice', 'meat', 'soup', '
fork', 'knife', 'oven', 'spoon'], 'Det': ['a', 'the']}
Gramatica sin recursividad:
    S -> NP VP
    VP -> cooks|drinks|eats|cuts|V NP VP1
    PP -> P NP
    NP -> Det N|he|she
    V -> cooks|drinks|eats|cuts
    P -> in|width
    N -> cat|dog|beer|cake|juice|meat|soup|fork|knife|oven|spoon
    Det -> a|the
    VP1 -> PP VP1|ε
No terminales:['PP', 'V', 'N', 'Det', 'S', 'NP', 'VP', 'P', 'VP1']
```

```
Gramatica sin producciones epsilon:
    S -> NP VP
    VP -> cuts|V NP VP1|eats|cooks|drinks|V NP
    PP -> P NP
    NP -> Det N|he|she
    V -> cooks|drinks|eats|cuts
    P -> in|width
    N -> cat|dog|beer|cake|juice|meat|soup|fork|knife|oven|spoon
    Det -> a|the
    VP1 -> PP VP1|PP
```

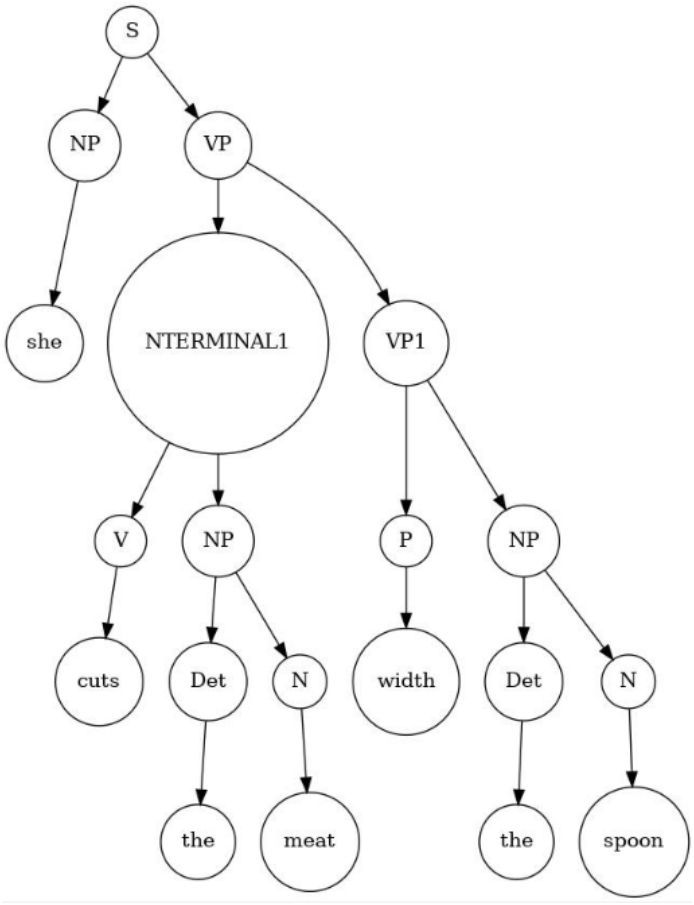
```
Parejas unitarias:[('VP1', 'PP')]
Gramatica sin producciones unitarias:
    S -> NP VP
    VP -> cuts|V NP VP1|eats|cooks|drinks|V NP
    PP -> P NP
    NP -> Det N|he|she
    V -> cooks|drinks|eats|cuts
    P -> in|width
    N -> cat|dog|beer|cake|juice|meat|soup|fork|knife|oven|spoon
    Det -> a|the
    VP1 -> PP VP1|P NP
```

```
Gramatica sin producciones inutiles:
    S -> NP VP
    VP -> cuts|V NP VP1|eats|cooks|drinks|V NP
    PP -> P NP
    NP -> Det N|he|she
    V -> cooks|drinks|eats|cuts
    P -> in|width
    N -> cat|dog|beer|cake|juice|meat|soup|fork|knife|oven|spoon
    Det -> a|the
    VP1 -> PP VP1|P NP
```

```
Gramatica en forma normal de chomsky:
    S -> NP VP
    VP -> cuts|NTERMINAL1 VP1|eats|cooks|drinks|V NP
    PP -> P NP
    NP -> Det N|he|she
    V -> cooks|drinks|eats|cuts
    P -> in|width
    N -> cat|dog|beer|cake|juice|meat|soup|fork|knife|oven|spoon
    Det -> a|the
    VP1 -> PP VP1|P NP
    NTERMINAL1 -> V NP
```

```
Ingrese una oracion: she cuts the meat width the spoon
SI
S
  NP
    she
  VP
    NTERMINAL1
      V
        cuts
      NP
        Det
          the
        N
          meat
    VP1
      P
        width
      NP
        Det
          the
        N
          spoon

Tiempo de ejecucion:
0.00034381018744574654 hrs: 0.020628611246744793 min: 1.238 seg
```



1. Gramática brindada en el ejemplo

```
Reglas a trabajar:
 E->E + T|T
 T->T * F|F
 F->( E )|id
```

```
Gramatica arreglada:
    E -> E + T|T
    T -> T * F|F
    F -> ( E )|id
No terminales:['T', 'E', 'F']
Terminales:['(', ')', '*', 'id', '+']
```

```
Gramatica sin producciones epsilon:
    E -> E + T|T
    T -> T * F|F
    F -> ( E )|id
```

```
recursivos:{'E': ['E + T'], 'T': ['T * F']}
betas:{'E': ['T'], 'T': ['F'], 'F': ['( E )', 'id']}
Gramatica sin recursividad:
    E -> T E1
    T -> F T1
    F -> ( E )|id
    E1 -> + T E1|ε
    T1 -> * F T1|ε
No terminales:['T', 'E', 'F', 'E1', 'T1']
```

```
Gramatica sin producciones epsilon:
    E -> T E1|T
    T -> F|F T1
    F -> ( E )|id
    E1 -> + T E1|+ T
    T1 -> * F|* F T1
```

```
Parejas unitarias:[('E', 'T'), ('T', 'F'), ('E', 'F')]
Gramatica sin producciones unitarias:
    E -> T E1|F T1|id|( E )
    T -> id|F T1|( E )
    F -> ( E )|id
    E1 -> + T E1|+ T
    T1 -> * F|* F T1
```

```
Gramatica en forma normal de chomsky:
    E -> T E1|F T1|id|NTERMINAL1 TERMINAL2
    T -> id|F T1|NTERMINAL1 TERMINAL2
    F -> NTERMINAL1 TERMINAL2|id
    E1 -> NTERMINAL2 E1|TERMINAL3 T
    T1 -> TERMINAL4 F|NTERMINAL3 T1
    TERMINAL1 -> (
    TERMINAL2 -> )
    TERMINAL3 -> +
    TERMINAL4 -> *
    NTERMINAL1 -> TERMINAL1 E
    NTERMINAL2 -> TERMINAL3 T
    NTERMINAL3 -> TERMINAL4 F
```

```
Ingrese una oracion: ( id * id ) + id
SI
E
  T
    NTERMINAL1
      TERMINAL1
        (
      E
        F
          id
        T1
          TERMINAL4
            *
          F
            id
    TERMINAL2
      )
  E1
    TERMINAL3
      +
    T
      id

Tiempo de ejecucion:
9.748670789930557e-07 hrs: 5.8492024739583336e-05 min: 0.004 seg
```

Este no se puede ver como tal el grafo debido a que al tener producciones (produce errores dado el label.

2. Gramática inventada recursiva y con permutaciones

```
Reglas a trabajar:
 S->A S A S A|c|ε
 A->b|ε
```

```
Gramatica arreglada:
    S -> A S A S A|c|ε
    A -> b|ε
No terminales:['A', 'S']
Terminales:['c', 'b']
```

```
Gramatica sin producciones epsilon:
    S -> A S S A|A S A S|A S S|A A S A|A S A A|c|A S A S A|S A S A|S A S|A A A|S S|S S A
    A -> b
```

```
recursivos:{'S': ['S A S A', 'S A S', 'S S', 'S S A']}
betas:{'S': ['A S S A', 'A S A S', 'A S S', 'A A S A', 'A S A A', 'c', 'A S A S A', 'A A A'], 'A': ['b']}
Gramatica sin recursividad:
    S -> A S A A|c|A S A S A|A A A|A S S A S1|A S A S S1|A S S S1|A A S A S1
    A -> b
    S1 -> A S A A S1|ε|A S S1|S S1|S A S1
No terminales:['A', 'S', 'S1']
```

```
Gramatica sin producciones epsilon:
    S -> A S S A|A S A S|A S A S S1|A A S A S1|A S S|A A S A|A S A A|c|A S A S A|A S S S1|A S S A S1|A A A
    A -> b
    S1 -> A S S1|A S A|S|S S1|S A|A S|S A S1|A S A S1
```

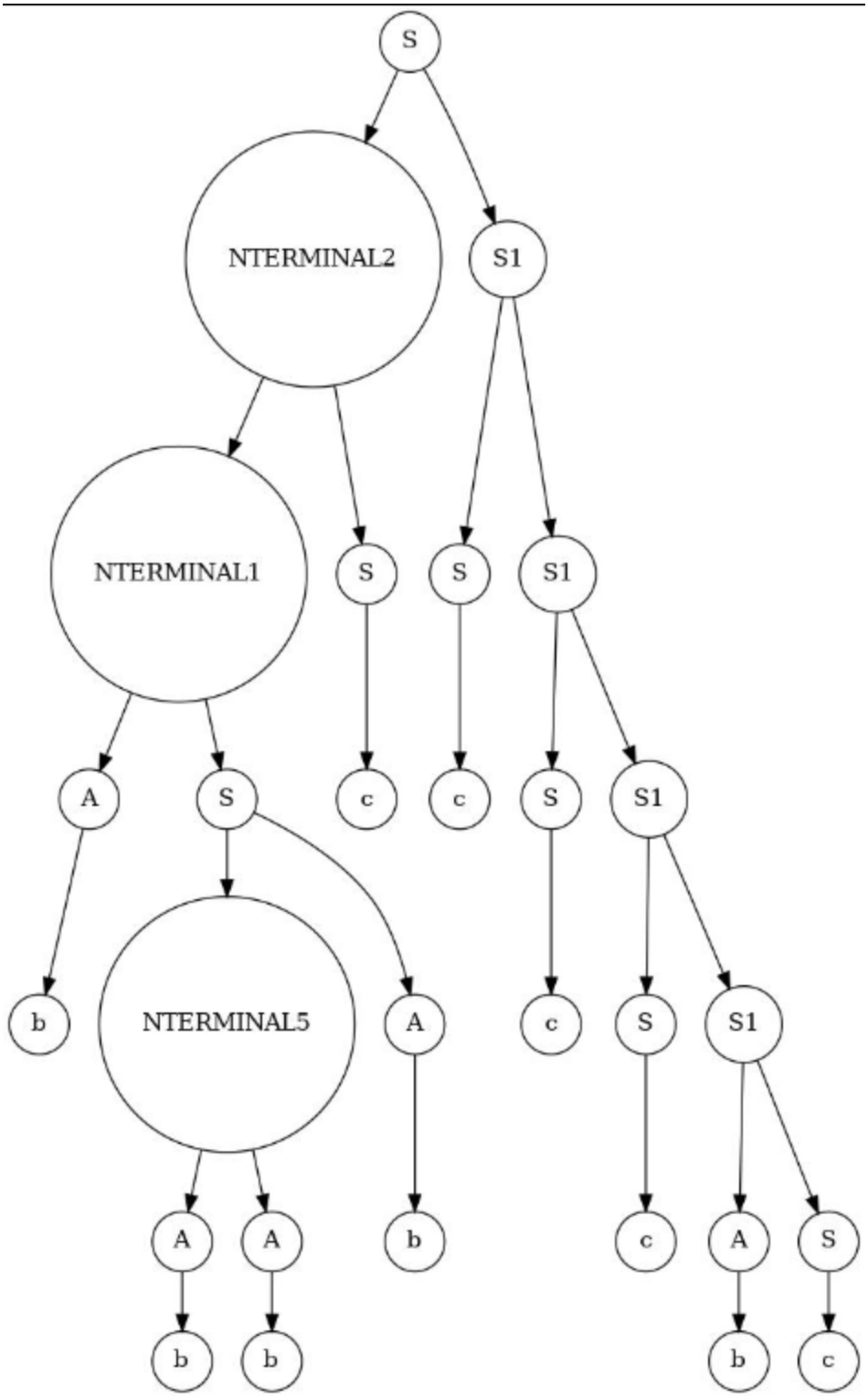
```
Parejas unitarias:[('S1', 'S')]
Gramatica sin producciones unitarias:
    S -> A S S A|A S A S|A S A S S1|A A S A S1|A S S|A A S A|A S A A|c|A S A S A|A S S S1|A S S A S1|A A A
    A -> b
    S1 -> A S A S|A A S A S1|A A S A|S A|A S|A S S1|S S1|S A S1|A S A S1|A S A S S1|A S S A S1|A A A|A S S A|A S A|A S S|A S A A|c|A S A S A|A S S S1
```

```
Gramatica sin producciones inutiles:
    S -> A S S A|A S A S|A S A S S1|A A S A S1|A S S|A A S A|A S A A|c|A S A S A|A S S S1|A S S A S1|A A A
    A -> b
    S1 -> A S A S|A A S A S1|A A S A|S A|A S|A S S1|S S1|S A S1|A S A S1|A S A S S1|A S S A S1|A A A|A S S A|A S A|A S S|A S A A|c|A S A S A|A S S S1
```

```
Gramatica en forma normal de chomsky:
    S -> NTERMINAL2 A|NTERMINAL3 S|NTERMINAL4 S1|NTERMINAL7 S1|NTERMINAL1 S|NTERMINAL6 A|NTERMINAL3 A|c|NTERMINAL4 A|NTERMINAL2 S1|NTERMINAL8 S1|NTERMINAL5 A
    A -> b
    S1 -> NTERMINAL3 S|NTERMINAL7 S1|NTERMINAL6 A|S A|A S|NTERMINAL1 S1|S S1|NTERMINAL9 S1|NTERMINAL3 S1|NTERMINAL4 S1|NTERMINAL8 S1|NTERMINAL5 A|NTERMINAL2 A|NTERMINAL1 A|NTERMINAL1 S|NTERMINAL3 A|c|NTERMINAL4 A|NTERMINAL2 S1
    NTERMINAL1 -> A S
    NTERMINAL2 -> NTERMINAL1 S
    NTERMINAL3 -> NTERMINAL1 A
    NTERMINAL4 -> NTERMINAL3 S
    NTERMINAL5 -> A A
    NTERMINAL6 -> NTERMINAL5 S
    NTERMINAL7 -> NTERMINAL6 A
    NTERMINAL8 -> NTERMINAL2 A
    NTERMINAL9 -> S A
```

```
Ingrese una oracion: b b b b c c c c b c
SI
S
  NTERMINAL2
    NTERMINAL1
      A
        b
      S
        NTERMINAL5
          A
            b
          A
            b
        A
          b
    S
      c
  S1
    S
      c
    S1
      S
        c
      S1
        S
          c
        S1
          A
            b
          S
            c

Tiempo de ejecucion:
2.095235718621148e-06 hrs: 0.00012571414311726887 min: 0.008 seg
```


S
NTERMINAL2
S1
NTERMINAL1
S
S
S1
A
S
c
c
S
S1
b
NTERMINAL5
A
c
S
S1
A
A
b
c
A
S
b
b
b
c