Universidad del Valle de Guatemala
Teoría de la computación
Sección 20

Proyecto #2

Brian Anthony Carrillo Monzón - 21108
Carlos Edgardo López Barrera - 21666

Guatemala, 23 de octubre del 2023

Diseño de la aplicación

Método de modelado: Modelado de procesos.
Se modeló la serie de pasos que se llevan a cabo en el algoritmo de simplificación de gramáticas y en el algoritmo de CYK.

Función	Entrada	Salida
simbolosAnulables	Gramática	Set con todos los símbolos anulables de la gramática dada.
eliminarProdsEpsilon	Gramática	Array con la gramática sin producciones epsilon. Utiliza **simbolosAnulables**.
prodValida	Regex,producción	Valor booleano respecto a la coincidencia de la producción y el regex de una producción válida.
simbolosNoTerminales	Gramática	Set con todos los símbolos no terminales de la gramática dada.
simbolosTerminales	Gramática	Set con todos los símbolos terminales de la gramática dada. Utiliza **simbolosNoTerminales.**
prodsUnarias	Gramática, No terminales	Set con todas las producciones unarias de la gramática ingresada.
prodsNoUnarias	Gramática, No terminales	Set con todas las producciones no unarias de la gramática ingresada.
eliminarProdsUnarias	Gramática	Array con la gramática sin producciones unarias. Utiliza **simbolosNoTerminales, prodsUnarias, prodsNoUnarias.**
simbolosGeneran	Gramática	Set con todos los símbolos que generan en 0 o más pasos un símbolo terminal. Utiliza **simbolosTerminales.**
eliminarSimbolosNoGeneran	Gramática	Array con la gramática sin símbolos que no generan. Utiliza **simbolosGeneran.**

simbolosAlcanzables	Gramática, simboloInicial	Set con todos los símbolos que son alcanzables a partir del símbolo inicial.
eliminarSimbolosNoAlcanzables	Gramática, simboloInicial	Array con la gramática sin símbolos no alcanzables. Utiliza **simbolosAlcanzables.**
eliminarSimbolosInutiles	Gramática, simboloInicial	Array con la gramática sin símbolos inútiles. Utiliza **eliminarSimbolosNoGeneran, eliminarSimbolosNoAlcanzables.**
generarSimbolo	State	Produce un nuevo símbolo no terminal en base al símbolo no terminal ingresado como estado.
cnfA	Gramática	Array de gramática con nuevas producciones para símbolos terminales que se encuentran en cuerpos de producción de longitud 2 o más. Utiliza **simbolosTerminales.**
cnfB	Gramática	Array de gramática con nuevas producciones para símbolos no terminales que se encuentran en cuerpos de producción de longitud 3 o más. Utiliza **simbolosTerminales.**
cnf	Gramática	Array con la gramática en CNF. Utiliza **cnfA, cnfB.**
visualize_tree	Node	No retorna nada. Ejecuta la función view() de Graphviz.
build_graph	Dot, Node, Parent_name	No retorna nada. Construye el grafo necesario para ser utilizado por Graphviz. Es recursiva.
print_tree	Node, indentación	No retorna nada. Imprime el parse tree en la consola. Es recursiva.
convert_to_grammar	rules_list	Diccionario de la gramática ingresada como array en rules_list.

cyk	Grammar, w, simboloInicial	Valor del nodo en caso éste posea como símbolo al símbolo inicial. En caso contrario se retorna None.

Discusión

Obstáculos:

- Tiempo de modelado: El realizar el modelaje de todo el procedimiento correspondiente para realizar la simplificación de las gramáticas y el algoritmo cyk tomó un tiempo considerado.

Recomendaciones:

- Los símbolos de la gramática ingresada tienen que ir separados por espacios en blanco. De lo contrario, las letras se considerarán como parte de un mismo símbolo, por ejemplo DET.
- La cabeza de la primera producción de la gramática ingresada debe ser el símbolo inicial.
- Se debe de utilizar el caracter especial de flecha: →
- Los símbolos terminales deben ir en minúsculas y los símbolos no terminales en mayúsculas.
- Utilizar gramáticas que no poseen recursividad por la izquierda.

Pruebas realizadas

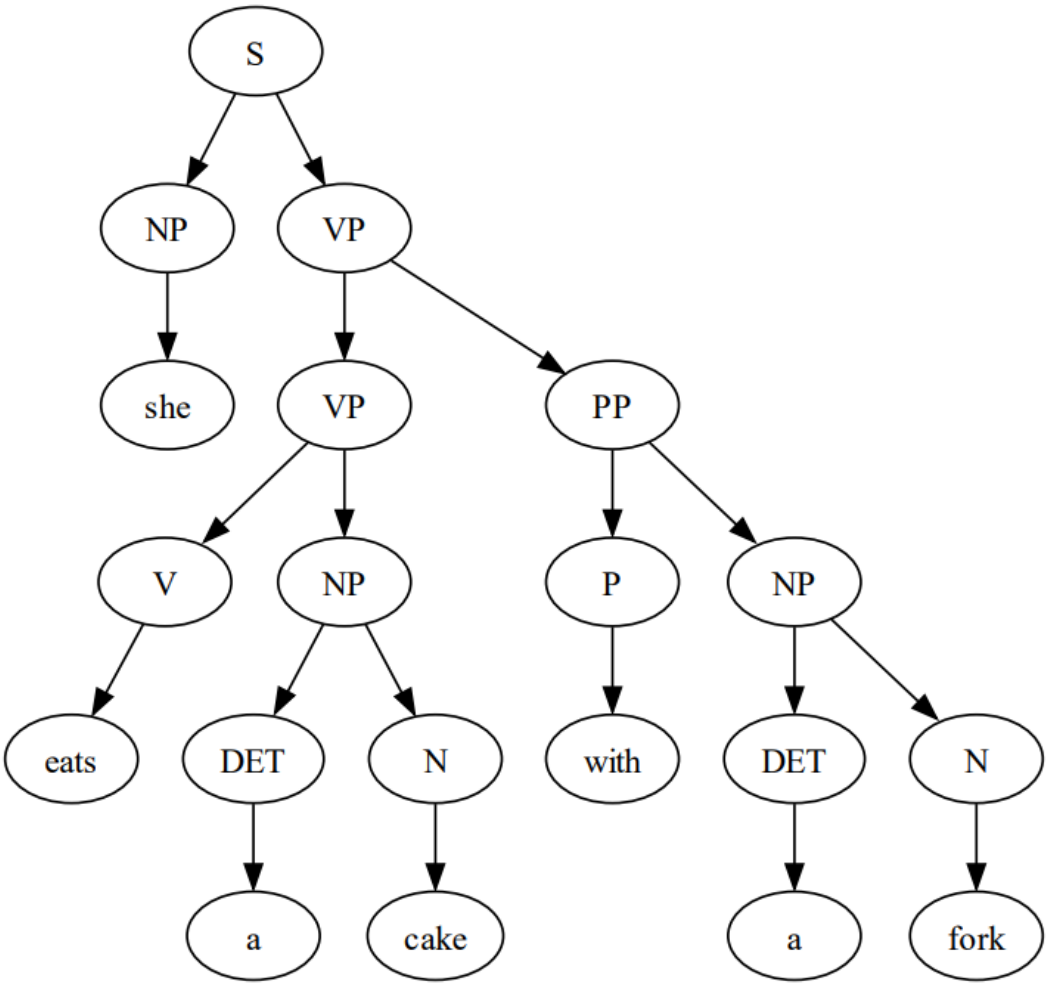
Primera gramática

w = she eats a cake with a fork

```
--------------------
GRAMATICA ORIGINAL:
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 | with
N → cat | dog
N → beer | cake | juice | meat | soup
N → fork | knife | oven | spoon
DET → a | the

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

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

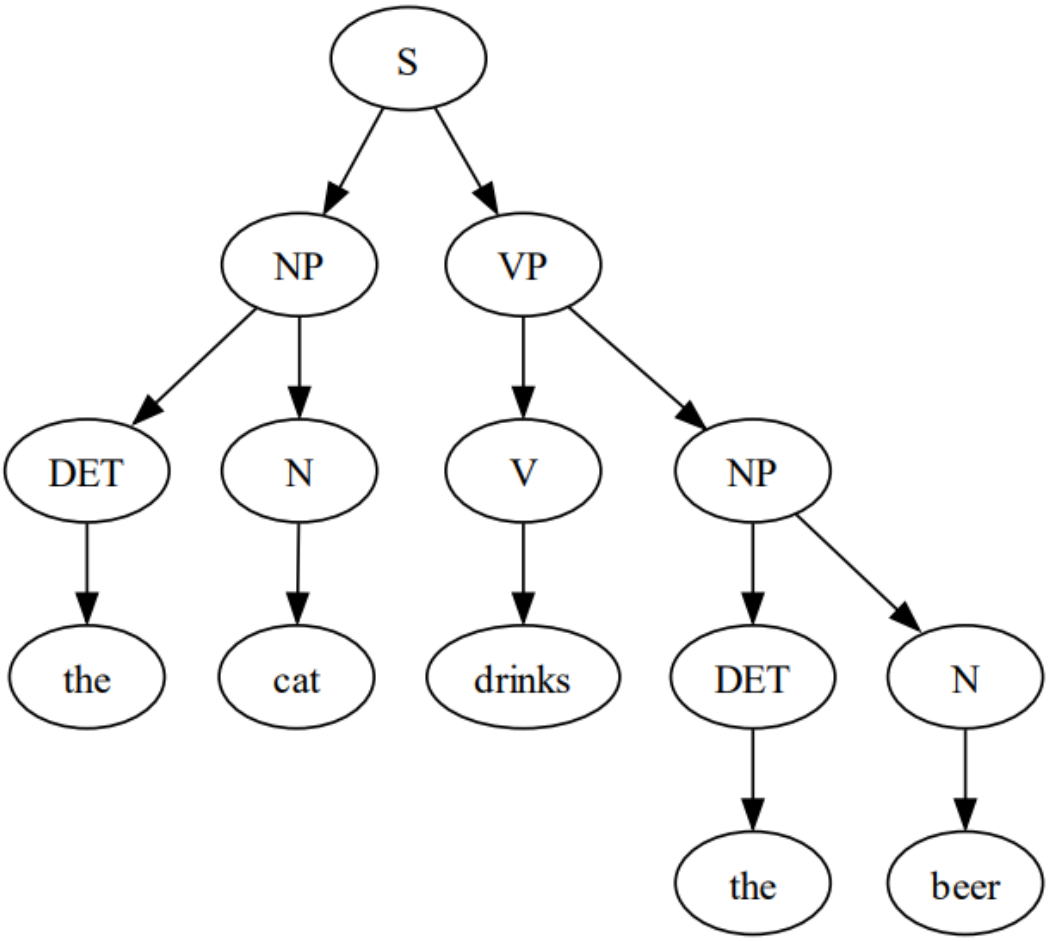
```
GRAMATICA SIN SIMBOLOS INUTILES:
DET → a | the
NP → DET N | he | she
V → eats | cuts | drinks | cooks
VP → drinks | VP PP | cuts | V NP | eats | cooks
N → spoon | fork | oven | cat | cake | knife | beer | juice | meat | dog | soup
P → in | with
S → NP VP
PP → P NP

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

```
Ingrese la cadena w. Separe con espacios en blanco cada no terminal. Ejemplo: ( id * id ) + id
she eats a cake with a fork

La expresión w SI pertenece al lenguaje descrito por la gramática.
El algoritmo tardó 0.3086 segundos en realizar la validación.
Presione una tecla para continuar . . .
```



Primera gramática
w = the cat drinks the beer

```
Ingrese la cadena w. Separe con espacios en blanco cada no terminal. Ejemplo: ( id * id ) + id
the cat drinks the beer

La expresión w SI pertenece al lenguaje descrito por la gramática.
El algoritmo tardó 0.3323 segundos en realizar la validación.
Presione una tecla para continuar . . .
```



Primera gramática

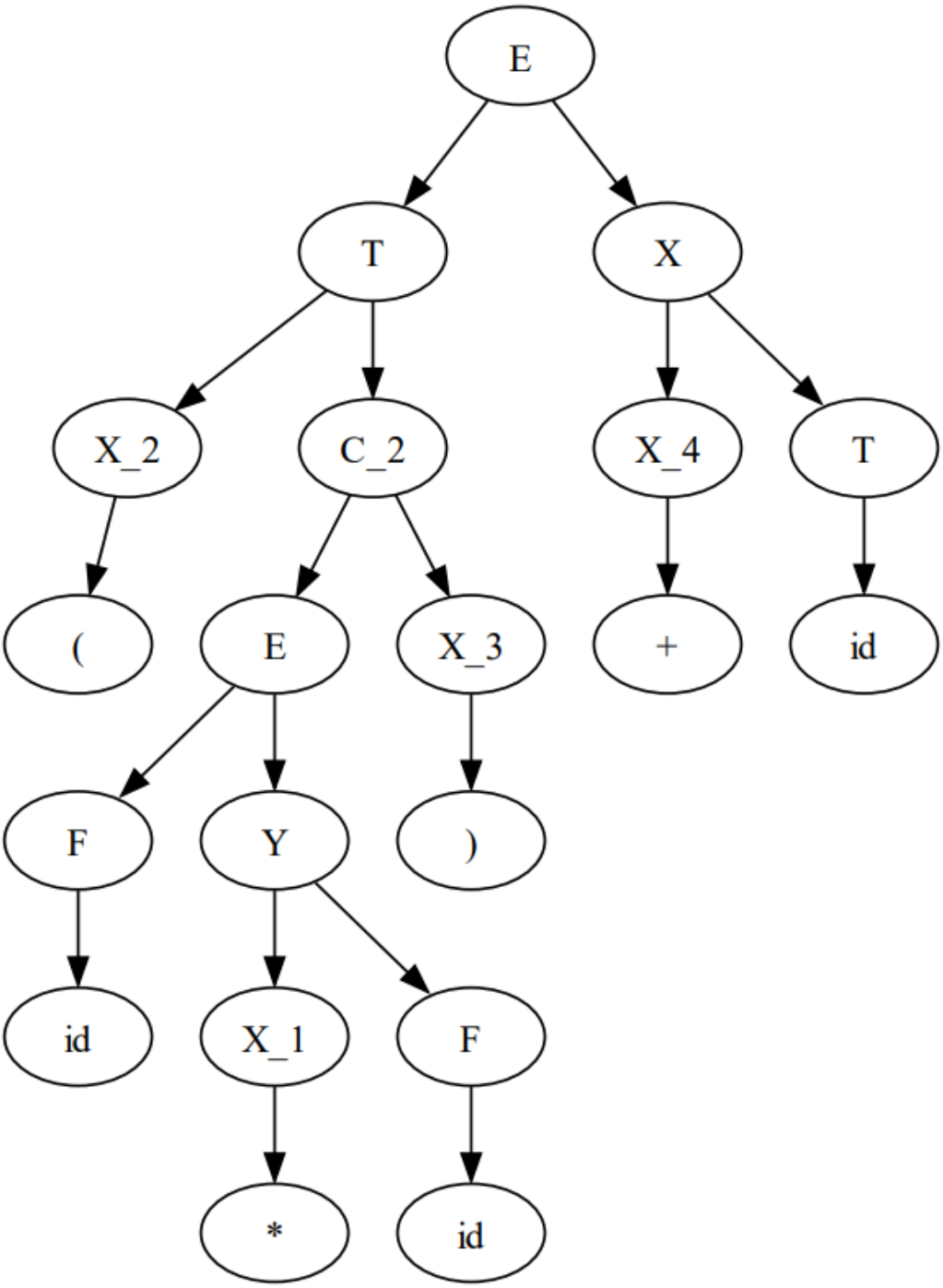
w = she eats a cake a fork with

```
Ingrese la cadena w. Separe con espacios en blanco cada no terminal. Ejemplo: ( id * id ) + id
she eats a cake a fork with
La expresión w NO pertenece al lenguaje descrito por la gramática.
El algoritmo tardó 0.0000 segundos en realizar la validación.
Presione una tecla para continuar . . .
```

Segunda gramática
w = (id * id) + id

```
--------------------
GRAMATICA ORIGINAL:
E → T X
X → + T X | ε
T → F Y
Y → * F Y | ε
F → ( E ) | id

GRAMATICA SIN PRODUCCIONES ε:
E → T | T X
X → + T X | + T
T → F Y | F
Y → * F | * F Y
F → ( E ) | id

GRAMATICA SIN PRODUCCIONES UNARIAS:
Y → * F | * F Y
E → ( E ) | F Y | T X | id
T → id | F Y | ( E )
F → ( E ) | id
X → + T X | + T

GRAMATICA SIN SIMBOLOS INUTILES:
Y → * F | * F Y
E → id | F Y | T X | ( E )
T → F Y | ( E ) | id
F → ( E ) | id
X → + T X | + T

GRAMATICA EN CNF:
C_1 → F Y
C_3 → T X
C_2 → E X_3
X_2 → (
X_4 → +
X_1 → *
X_3 → )
Y → X_1 F | X_1 C_1
E → F Y | X_2 C_2 | T X | id
T → F Y | X_2 C_2 | id
F → X_2 C_2 | id
X → X_4 T | X_4 C_3
```

```
Ingrese la cadena w. Separe con espacios en blanco cada no terminal. Ejemplo: ( id * id ) + id
( id * id ) + id

La expresión w SI pertenece al lenguaje descrito por la gramática.
El algoritmo tardó 0.2986 segundos en realizar la validación.
Presione una tecla para continuar . . .
```



Segunda gramática
w = ((id)