INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

GARZÓN DOMÍNGUEZ GERARDO ISMAEL

2CV13

TEORÍA COMPUTACIONAL

PRÁCTICA 6:
GRAMÁTICA

INTRODUCCIÓN:

Intuitivamente, una gramática es un conjunto de reglas que generan un subconjunto L de Σ^*, para algún un alfabeto Σ, esto es: $L \subseteq \Sigma^*$. Estas reglas permiten reemplazar símbolos o cadenas de símbolos hasta que finalmente se puede probar que alguna cadena $w \in L$, a este proceso se le conoce como derivación. Bajo esta descripción, los símbolos que son reemplazados por otros símbolos son llamados *no terminales*, mientras que los símbolos que no pueden ser reemplazados por otros símbolos son referidos como *terminales*, por último, las reglas bajo las cuales está permitida la sustitución de símbolos son llamadas *producciones*.[1]

Formalmente, una gramática es una 4-tupla: $G = (N, \Sigma, S, P)$, donde:

N: Conjunto finito de símbolos *no terminales*, Σ: Conjunto finito de símbolos *terminales*,

S: Símbolo inicial, además $S \in N$, P: Conjunto finito de reglas de producción. [1]

Formalmente P es una relación en $(N \cup \Sigma)^*$, tal que cada elemento de P, es un par ordenado, donde el primer elemento pertenece a N, y al menos un par tiene como primer elemento a S. [1]

A continuación, se explicará brevemente que es una expresión regular, esto es útil, ya que la implementación de esta práctica puede explicarse mediante ciertas relaciones entre ciertos tipos gramáticas conocidas como gramáticas regulares, y los lenguajes regulares.

Las expresiones regulares pueden usarse para describir ciertos lenguajes. Si *r* es una expresión regular, entonces $L(r)$ denota un lenguaje asociado con *r* partiendo de las siguientes afirmaciones [2]:

1. ϕ es una expresión regular que denota el conjunto vacío.
2. λ es una expresión regular que denota el conjunto: $\{\lambda\}$.
3. Para todo $\alpha \in \Sigma$, α es una expresión regular que denota a $\{\alpha\}$.

Si r_1, r_2 son expresiones regulares, entonces [2]:

4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$,
5. $L(r_1 \cdot r_2) = L(r_1)\, L(r_2)$,
6. $L((r_1)) = L(r_1)$,
7. $L(r_1^*) = (L(r_1))^*$.

Son lenguajes regulares asociados con las expresiones regulares r_1, r_2.

La idea detrás de las expresiones regulares es describir un patrón o regularidad que cumplen las cadenas pertenecientes a cierto lenguaje, ya que, un lenguaje (que es básicamente un conjunto),

podría ser infinito o en el mejor de los casos muy grande, y como sabemos, las computadoras, no tienen memoria infinita (por ejemplo, para generar todo el conjunto de cadenas que cumplan ciertas condiciones, como (ab^*)), ni manera de "razonar" de manera directa o intuir afirmaciones como: $a \in \{a, b, c, \dots\}$. Por esta razón, se opta por usar las expresiones regulares, las cuales, en su mayoría, pueden ser verificadas mediante algún tipo de autómata no determinista, los cuales, están descritos de manera más algorítmica.

Las gramáticas son otra manera de describir lenguajes. Los tipos de gramática más sencillas son las gramáticas lineales por la derecha o lineales por la izquierda. [2]

Una gramática $G = (N, \Sigma, S, P)$ es lineal por derecha si todas sus producciones son de la forma [2]:

$$A \to xB,$$
$$A \to x,$$

Una gramática $G = (N, \Sigma, S, P)$ es lineal por la izquierda si todas sus producciones son de la forma [2]:

$$A \to Bx,$$
$$A \to x.$$

Para ambos casos: $A, B \in N, \quad x \in \Sigma$

Una gramática lineal es aquella que es solamente lineal por la izquierda o solamente lineal por la derecha, además, se debe notar que, a lo más, existe un *no terminal* que debe ser consistentemente el primer o ultimo símbolo de cualquier producción. [2]

Por último, se presentará la idea de que las gramáticas lineales por la derecha pueden generar lenguajes regulares. Como se explicó previamente, la relación de pertenencia de una cadena a un lenguaje regular asociado con una expresión regular puede ser probada mediante el uso de algún tipo de autómata no determinista. El hecho de que una gramática lineal por la derecha es capaz de generar un lenguaje regular, implica que se puede probar si una cadena pertenece o no al lenguaje regular asociado con una gramática lineal por la derecha, esto es: $w \in L(G)$ *o* $w \notin L(G)$.

La prueba de que las gramáticas lineales por la derecha generan lenguajes regulares se presenta como un teorema en la bibliografía [2]. Sin embargo, la intuición detrás de la prueba es que, cada paso de la derivación de una cadena mediante el uso de alguna regla de producción de la gramática puede ser imitado como una transición de un AFN. Esta idea es clave para el desarrollo de esta práctica, ya que el problema puede resolverse haciendo uso de esta conclusión sobre la relación entre los AFN, lenguajes regulares y gramáticas lineales por la derecha.

DESARROLLO E IMPLEMENTACIÓN

La meta de esta práctica es implementar la gramática descrita por:

$$E \longrightarrow aB \mid bA \mid e$$
$$A \longrightarrow aE$$
$$B \longrightarrow bE$$

Figura 1 Descripción de la gramática a implementar.

A partir de la figura 1, se puede deducir que:
$G = (N, \Sigma, S, P)$
Donde:
$N = \{E, A, B\}$,
$\Sigma = \{a, b, \lambda\}$,
$S = E$,
$P = \{E \to aB|bA|\lambda,\ A \to aE,\ B \to bE\}$.

Al analizar las reglas de producción dadas por P, se puede notar que ninguna producción por si misma da más de una opción de sustitución por no terminal, por ejemplo, una regla de la forma:

$$R \to aB|aA$$

Esto significa que la función de transiciones tendrá máximo un elemento como resultado para cualquier entrada de la forma (a, b).

Debido a lo mostrado en la introducción, podemos concluir que esta gramática es una gramática lineal por la derecha, ya que sus reglas de producción tienen un terminal al cual le precede un no terminal, o un terminal solo (en este caso la cadena vacía en la regla de producción E).

Para transformar las reglas de producción de esta gramática a un AFN, primero se separa la producción E en producciones individuales, esto es:

$$E \to aB, \qquad E \to bA, \qquad E \to \lambda.$$

Ahora, con todas las reglas de producción de manera individual, se puede suponer que los no terminales representan un estado del autómata, mientras que los terminales representan una transición, además, el no terminal E tiene las siguientes características especiales:

1. Representa el estado inicial del autómata.
2. Es el estado de aceptación del autómata.
3. Puede hacer transición λ a sí mismo.

Las reglas de producción de las gramáticas lineales por la derecha pueden ser reinterpretadas de la siguiente manera:

Si todo no terminal es un estado del autómata, entonces el no terminal del lado izquierdo de la producción, representa el primer elemento de algún par ordenado (a, b), y el terminal al lado derecho de la producción representa el segundo elemento de algún par ordenado (a, b), donde dicho par es la entrada que recibe la función δ del autómata como entrada. Por último, el no terminal al lado derecho de la producción representa a $\delta(a, b) = P_i$, donde:

(a, b) es un par de la forma $(estado, simbolo)$,

P_i es el resultado del mapeo δ con los argumentos (a, b), y P_i es un estado, o bien, desde el punto de vista de la gramática $P_i \in P$.

Aplicando este razonamiento, fácilmente se puede construir una tabla de transiciones para el autómata asociado al lenguaje regular que la gramática genera.

P, Σ	a	b	λ
$\rightarrow E *$	B	A	E
A	E	ϕ	ϕ
B	ϕ	E	ϕ

Figura 2 Función de transiciones para el autómata del lenguaje regular que genera la gramática de la figura 1.

Para implementar este autómata, cada estado será una función dentro del programa que decida a que estado saltar según el símbolo dado, además, cada estado regresa 0 en caso de que este reciba un símbolo que no está definido en su tabla de saltos [3], o en caso contrario, regresa lo que el estado al que se salta regresa, en adición a esto, los estados de aceptación deben regresar 1 cuando estos detecten que se ha terminado de iterar sobre la cadena, ya que esto ocurre solo cuando se ha recorrido un camino valido del autómata, pues bajo este planteamiento, todas las cadenas inválidas causaran que alguno de los estados del autómata regrese 0 en vez de llamar a otro estado.

Para clarificar la explicación anterior se mostrarán las definiciones de las funciones estado de este autómata. Nótese que estas funciones reciben como argumento un apuntador a un objeto *str_t*, y un índice, esto es debido a que las cadenas en los objetos *str_t* no están terminadas con el byte nulo, siendo la excepción cuando estas son una cadena vacía. Sin embargo, mediante las funciones para el tipo *str_t* es y el índice que estas funciones reciben es posible determinar cuando se ha llegado al final de una cadena.

```c
int S(str_t* str, size_t curr)
{
    if(str_len(str) == curr) return 1;
    switch(str_idx(str, curr))
    {
        case 'a': return B(str, curr+1);
        case 'b': return A(str, curr+1);
        default: break;
    }
    return 0;
}

int A(str_t* str, size_t curr)
{
    if(str_idx(str, curr) == 'a') return S(str, curr+1);
    else return 0;
}

int B(str_t* str, size_t curr)
{
    if(str_idx(str, curr) == 'b') return S(str, curr+1);
    else return 0;
}
```

Figura 3 Funciones que representan los estados del autómata para el lenguaje regular de la gramática de la figura 1.

Básicamente esa es la solución para este problema, además note que cada salto en el autómata consume un símbolo. Cabe mencionar que se implementaron algunas otras minuciosidades en el programa, como una función que espera recibir como argumentos, una cadena a validar, un conjunto al cual se agregan cadenas validadas y la función que representa el estado inicial del autómata. Así, se podrá comprobar si ciertas cadenas son válidas para la gramática de la figura 1.

```
Introduzca cadenas.
Para salir escriba 'exit'
>> a
No aceptado por la gramatica!
>> ab
Aceptado por la gramatica!
>> abbaab
Aceptado por la gramatica!
>> babaabb
No aceptado por la gramatica!
>> exit
Cadenas en L(G) = {"abbaab", "ab", }
```

Figura 4 Resultados para las cadenas de prueba dadas en la asignación

Como paso extra, se mostrará las soluciones para determinar si estas cadenas pertenecen al lenguaje regular generado por la gramática de la figura 1, para cada ejemplo se supone que si la cadena pertenece al lenguaje de la gramática, entonces debe existir una derivación a partir de E, tal que esta lleve a la cadena en cuestión y que en el último paso de la derivación, el no terminal restante sea sustituido por una producción que no contenga variables, en este caso esa sustitución se realiza mediante la cadena vacía.

1. Supongamos que, $a \in L(G)$

$$a \Rightarrow E \Rightarrow aB \Rightarrow abE \Rightarrow ab\lambda$$

Se determina que, $a \notin L(G)$

2. Supongamos que, $ab \in L(G)$

$$ab \Rightarrow E \Rightarrow aB \Rightarrow abE \Rightarrow ab\epsilon$$

Se confirma que, $ab \in L(G)$

3. Supongamos que, $abbaab \in L(G)$

$$abbaab \Rightarrow E \Rightarrow aB \Rightarrow abE \Rightarrow abbA \Rightarrow abbaE \Rightarrow abbaaB \Rightarrow abbaabE \Rightarrow abbaab\lambda$$

Se confirma que, $\in L(G)$

4. Supongamos que, $babaabb \in L(G)$

$$babaabb \Rightarrow E \Rightarrow bA \Rightarrow baE \Rightarrow babA \Rightarrow babaE \Rightarrow babaaB \Rightarrow babaabE \Rightarrow babaabbA \Rightarrow babaabbaE \Rightarrow babaabba\lambda$$

Se determina que, $babaabb \notin L(G)$

5. Ejemplo extra de una cadena que es generable por la gramática. Esta derivación comienza desde el penúltimo paso del ejemplo 3.

$$abbaabE \Rightarrow abbaabbA \Rightarrow abbaabba$$

CONCLUSIÓN

Considero que esta práctica ayudó más que nada a profundizar sobre el conocimiento previo adquirido mediante las clases y prácticas anteriores. Lo menciono por el hecho de que gracias a la bibliografía pude notar ciertas conexiones entre temas ya conocidos, el ejemplo es bastante claro, ya que la implementación de esta práctica se basó mayormente en la relación que existe entre las gramáticas lineales de lado derecho, los lenguajes regulares y los autómatas finitos no deterministas.

En sí, me parece que algunas de las ideas centrales de esta práctica fueron:

1. Notar que hay maneras distintas de representar conjuntos de cadenas que cumplen ciertas condiciones, o sea, los lenguajes regulares.
2. Mostrar la capacidad que tienen las computadoras para "probar" ciertas afirmaciones, en este caso estas "afirmaciones" fueron si una cadena pertenece o no a un conjunto de cadenas, sin usar un enfoque de "fuerza bruta", el cual sería algo como generar un posible conjunto enorme de cadenas solo para saber si una afirmación es cierta o no.

En cierta parte los lenguajes regulares me recordaron a la manera en la que se definen los pares ordenados que "genera" una función sobre números. Por ejemplo, tomemos el producto cartesiano de los naturales con los naturales, y posteriormente tomar aquellos pares que cumplan con cierta condición como que el segundo elemento del par sea el cuadrado del primero, ahora en efecto hemos formado un subconjunto del producto cartesiano de los naturales con los naturales. La analogía mediante el uso de cadenas sería, tener un alfabeto y aplicar la cerradura de Kleene (análogo al producto cartesiano), posteriormente, seleccionar aquellas cadenas que cumplan cierta condición (análogo a formar un subconjunto de números). Ahora imaginando que usamos computadoras; si se quisiese saber si un número natural cualquiera pertenece al conjunto imagen del primer ejemplo, tendríamos que hacer pasar a tal número mediante una función que determine si la raíz cuadrada de este es un entero o no. La analogía para el ejemplo de las cadenas sería procesar la cadena mediante un autómata para determinar si esta cumple con una serie de condiciones impuestas por una expresión regular.

REFERENCIAS

[1] J. Anderson, Automata theory with modern applications, 1ra ed. Cambridge, New York, Melbourne: Cambridge University Press, 2006, p. 114

[2] P. Linz, An Introduction to Formal Languages and Automata, 6ta ed. Davis, California: Jones & Bartlett, 2017, pp. 75-103

[3] J. Bacon, Lectura de clase, tema: “Jump tables”, CS315, Computer Science Department, College of Engineering & Applied Science, University of Wisconsin, Milwaukee., 2011.