Universidad Nacional de Ingeniería

CC321 Teoría de autómatas

Implementación de la gramática en Python

Profesor:
Victor Melchor Espinoza
Departamento de Ciencias de la Computación

Grupo:
Ronaldo Lopez
Davis Alderete
Cristhian Sánchez Sauñe

Índice

1. Introducción

Las gramáticas son un formalismo diseñado para la definición de lenguajes. Estas son un medio para generar las cadenas que pertenecen a un lenguaje. El elemento fundamental de las gramáticas es la regla.

1.1. Definición de una regla

Dado un alfabeto Σ , una regla definida sobre Σ es una par ordenado de cadenas (x, y) donde x, y $\in \Sigma^*$.

- x se dirá parte izquierda de la producción.
- y se dirá parte derecha de la producción.

Notación:

$$x ::= y$$

Una regla se llamará compresora si la longitud de su parte derecha es menor que la de la parte izquierda.

Ejemplo: Defina un conjunto de reglas que nos permita evaluar la correcta formación de frases. De tal forma que "el atleta corre velozmente" sea una frase correcta.

Sea el conjunto de producciones P:

$R_1 < oración >:=< sujeto >< predicado >$

$R_2 < sujeto >:=< artículo >< nombre >$

$R_3 < predicado >:=< verbo >< complemento >$

$R_4 < predicado >:=< verbo >$

$R_5 < articulo >:= el$

$R_6 < articulo >:= la$

$R_7 < nombre >:= atleta$

$R_8 < nombre >:= voleibolista$

$R_9 < verbo >:= corre$

$R_{10} < verbo >:= salta$

$R_{11} < complemento >:= velozmente$

$R_{12} < complemento >:= mucho$

Utilizando el conjunto P y partiendo del ítem <oracion> podemos obtener frases como:

- "el atleta corre velozmente".
- "la voleibolista salta mucho".
- "la voleibolista salta".

Sin embargo, nunca podríamos formar la frase:

- "mucho velozmente atleta".

1.2. Implementación en Python

El propósito de este informe es mostrar la construcción de un programa Python que genera cadenas aleatorias a partir de la gramática declarada con anterioridad.

2. Clases y métodos

El programa deberá implementar las siguientes clases para su correcto funcionamiento.

2.1. Clase *Regla*

Su instancia debe representar una regla en una gramática. Debe tener el siguiente constructor:

2.1.1. Metodo *__init__(self, left, right)*

Cada instancia de *Regla* debe tener tres variables internas. La variable **self.left** se inicializa con la cadena izquierda. La variable *self.right* se inicializa con la tupla de cadenas a la derecha. La variable *self.cont* se inicializa con el entero 1 (la clase *Gramatica* la usará para ayudar a elegir las reglas).

```
def __init__(self, left, right):
    self.left = left
    self.right = right
    self.cont = 1  # la clase gramática la usa para eligir las reglas
```

2.1.2. Método *__repr__(self)*

Devuelve una cadena de la forma $C\ L\ =>\ R_1R_2...R_n$, donde C está dado por $self.cont$, L está dado por $self.left$ y $R_1\ R_2...R_n$, son los elementos de $self.right$.

Por ejemplo, si crea una regla llamando al constructor:

$$Regla('Historia',\ ('Frase',\ 'y',\ 'Historia'))$$

entonces su método $__repr__$ debe devolver la cadena:

$$'1\ Historia => Frase\, y\, Historia'$$

Es decir, la cadena debe verse como una regla. Cuando se imprime una instancia de la clase Regla, Python llama automáticamente al método $__repr__$.

```
def __repr__(self):
    right = ''

    for i, r in enumerate(self.right):

        if(len(self.right) == 1):
            right = r

        else:
            right += r
            if i != (len(self.right) - 1):
                right += " "

    stringOutput = str(self.cont) + ' ' + self.left + " -> " + right
    return stringOutput
```

2.2. Clase *Gramatica*

Esta clase debe implementar una gramática usando reglas, como las descritas anteriormente.

2.2.1. Método __init__(self, semilla)

Inicializa una instancia de Gramática. Se crea una instancia del generador de números aleatorios **Random** que usa semilla (seed). También se define un diccionario que almacene las reglas.

El diccionario debe estar inicialmente vacío.

```
def __init__(self, semilla):
    import random as r1
    self.random = r1
    self.random.seed(semilla)
    self.dictOfRules = {}
```

2.2.2. Método *regla(self, left, right)*

Agrega una nueva regla a la gramática. Aqui, $left$ es una cadena. Representa el símbolo no terminal en el lado izquierdo de la regla. Además, right es una tupla cuyos elementos son cadenas. Representan los terminales y los no terminales que están en el lado derecho de la regla.

Intentaremos encontrar el valor de $left$ en el diccionario. Si no existe dicho valor, el valor de $left$ en el diccionario será una tupla cuyo único elemento es $Regla(left, right)$. Sin embargo, si existe tal valor, será una tupla. Añadimos una $Regla(izquierda, derecha)$ al final de esa tupla.

```
def regla(self, left, right):

    if left in self.dictOfRules.keys():

        r_mod = Regla(left=left, right=right)

        getRight = lambda r: r.right

        if right in list(map(getRight, self.dictOfRules[left])):
            r_mod.cont += 1

        n = len(self.dictOfRules[left])
        tup = ()

        for i in range(n):

            if right == self.dictOfRules[left][i].right:
                self.dictOfRules[left][i].cont += 1

            tup += (self.dictOfRules[left][i], )

        tup += (r_mod, )

        self.dictOfRules[left] = tup

    else:
        r_new = Regla(left, right)
        self.dictOfRules[left] = (r_new, )
```

2.2.3. Método genera(self)

Este método generará una cadena. Si hay una regla con el lado izquierdo igual a $'Inicio'$ en el diccionario, llamamos al método *generar* cuya entrada será la tupla $('Inicio',)$, y retornamos el resultado. Si no existe dicha regla, entonces lanzamos una excepción, porque no puede generar cadenas sin una regla para $'Inicio'$.

```
def genera(self):

    if 'Inicio' in self.dictOfRules.keys():
        r = self.generar(strings = ('Inicio', ))
        return r

    else:
        raise Exception("\nNo se puede generar cadenas sin una regla para 'Inicio'\n")
```

2.2.4. Método *generar(self, strings)*

El parámetro *strings* es una tupla cuyos elementos son cadenas. Las cadenas representan símbolos terminales y no terminales.

Inicializamos una variable llamada *resultado* como una cadena vacía. Esta almacenará el resultado que será devuelto por este método. Luego usaremos un bucle para visitar cada cadena en *strings*.

Si la cadena visitada no es una clave en el diccionario, entonces es un símbolo terminal. Lo agregamos al final del resultado, seguido de un espacio en blanco $''$.

Si la cadena visitada es una clave en el diccionario, entonces es un símbolo no terminal. Llamaremos al método *seleccionar* en la cadena para obtener una tupla de cadenas. Luego llamamos recursivamente a *generar* en esa tupla de cadenas, para obtener una nueva cadena. Agregamos la nueva cadena al final del resultado, sin un espacio en blanco al final.

Continuamos de esta manera hasta que se hayan visitado todas las cadenas en *strings*. En ese punto, el resultado será una cadena generada por la gramática. Al final retornamos la variable *resultado*.

```
def generar(self, strings):

    resultado = ''
    for s in strings:
        if s in self.dictOfRules.keys():
            tuplStrings = self.seleccionar(left = s)
            r = self.generar(tuplStrings)
            resultado += r

        else:
            resultado = resultado + s + " "


    return resultado
```

2.2.5. Método *seleccionar(self, left)*

Se elige una regla al azar cuyo lado izquierdo es la cadena $left$. Esto sucede en varios pasos:

- Establecemos la variable *reglas* para que sea una tupla de todas las reglas con $left$ en sus lados izquierdos (del diccionario). Establecemos la variable *total* como la suma de todas las variables *cont* en las reglas.
- Establecemos la variable *indice* en un entero elegido al azar. Debe ser mayor o igual que 0, pero menor que *total*.
- Visitamos cada regla en *reglas*, una a la vez. Restamos la variable *cont* de la regla del *indice*. Continuamos de esta manera hasta que la variable *indice* sea menor o igual a O. Establecemos la variable *elegido* para la regla que se estaba visitando cuando esto ocurrió. (Como resultado, es más probable que se elijan reglas con grandes variables de conteo que reglas con pequeñas variables de conteo).
- Agregamos 1 a las variables *cont* de todas las reglas en la variable *reglas*, excepto las elegidas (Esto hace que sea más probable que se seleccione una regla distinta a la elegida más adelante, lo que proporciona un rango más amplio de cadenas aleatorias).

Finalmente, devolvemos la variable *right* de *elegido*, como una tupla de cadenas.

```
def seleccionar(self, left):

    reglas = self.dictOfRules[left]
    total = sum(r.cont for r in reglas)
    indice = self.random.randint(0, total)

    for r in reglas:
        indice -= r.cont
        if indice <= 0:
            elegido = r

    for i, r in enumerate(reglas):
        if r != elegido:
            reglas[i].cont = r.cont + 1
    return elegido.right
```

3. Resultados

Una vez que ya tengamos las clases con sus respectivos métodos, ahora sólo nos queda crear nuestro *menu* y ejecutar el programa para la resolucion del ejemplo anterior.

```
def menu():

    print("-------------------------")
    print(f"{Fore.GREEN}Menú")
    print("1.- Agregar nueva regla")
    print("2.- Generar cadena")
    print("3.- salir")
    print(f'{Style.RESET_ALL}')

    op = int(input("Ingrese opción:  "))

    return op
```

Figura 1: Función *menu*

```
if __name__ == '__main__':

    ke = Gramatica(semilla = 144)
    #print(ke.generar('Inicio'))

    while True:
        ope = menu()

        if ope == 1:
            izq = input("ingrese parte izq: ")
            der = input("ingrese parte der: ")

            tuplaDer = tuple(der.split(" "))

            ke.regla(left=izq, right=tuplaDer)

            print(f"{Fore.YELLOW}\nmostrando reglas...\n")
            for rules in ke.dictOfRules:
                if len(ke.dictOfRules[rules]) == 1:
                    print(ke.dictOfRules[rules][0])

                else:
                    for r in ke.dictOfRules[rules]:
                        print(r)
            print(f'{Style.RESET_ALL}')

        if ope == 2:
            print(f"\n{Fore.YELLOW}mostrando reglas disponibles...\n")
            for rules in ke.dictOfRules:
                if len(ke.dictOfRules[rules]) == 1:
                    print(ke.dictOfRules[rules][0])

                else:
                    for r in ke.dictOfRules[rules]:
                        print(r)
            print(f'{Style.RESET_ALL}')

            print("\ngenerando cadena...\n")
            print(f"\t{ke.genera()}\n")

        elif ope == 3:
            break
```

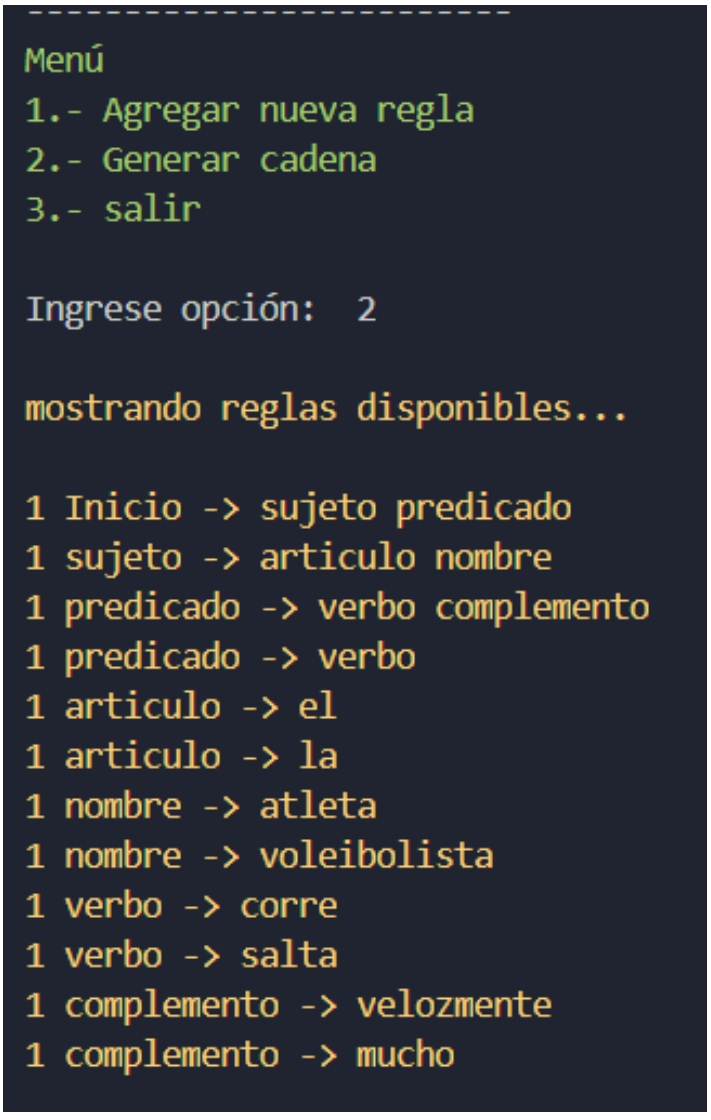

Figura 2: Declaración de las reglas

```
1 complemento -> velozmente
1 complemento -> mucho


generando cadena...


Reglas: (1 Inicio -> sujeto predicado,)
Regla elegida: 1 Inicio -> sujeto predicado


Reglas: (1 sujeto -> articulo nombre,)
Regla elegida: 1 sujeto -> articulo nombre


Reglas: (1 articulo -> el, 1 articulo -> la)
Regla elegida: 1 articulo -> la


Reglas: (1 nombre -> atleta, 1 nombre -> voleibolista)
Regla elegida: 1 nombre -> voleibolista


Reglas: (1 predicado -> verbo complemento, 1 predicado -> verbo)
Regla elegida: 1 predicado -> verbo


Reglas: (1 verbo -> corre, 1 verbo -> salta)
Regla elegida: 1 verbo -> salta

        la voleibolista salta

-------------------------
```

Figura 3: Cadena aleatoria obtenida