



INSTITUTO POLITÉCNICO NACIONAL.
ESCUELA SUPERIOR DE CÓMPUTO.
TEORÍA COMPUTACIONAL.
PRÁCTICA NÚMERO TRECE.

ALUMNO: HERNÁNDEZ CASTELLANOS CÉSAR URIEL.

PROFESOR: ROSAS TRIGUEROS JORGE LUIS.

FECHA DE REALIZACIÓN: 6/DICIEMBRE/2017

FECHA DE ENTREGA: 10/DICIEMBRE/2017

```
content="width=device-width,  
icon" href="/favicon.ico" typ  
f="/favicon.ico" type="image
```

```
ss" rel="stylesheet" href="css/materialize.min.css" media="screen,projection" <  
heet" href="https://maxcdn.bootstrapcdn.com/font-awesome/4.4.0/css/font-awesome.min.css">  
heet" href="/css/animate.css">  
heet" href="css/theme.css">
```

```
ner">  
nav">  
="nav-wrapper">  
ass="container">  
ef="#" class="brand-logo hide-on-med-and-up"><div>  
s="container">  
m8 18 hide-on-small-only">  
ght:400">starter</span><span><div>  
up network and workspace tool for teams, it's not just  
span 100" banner-rotator" class="rotator">
```



Marco teórico

Una gramática formal es una estructura matemática con un conjunto de reglas de formación que definen las cadenas de caracteres admisibles en un determinado lenguaje formal o lengua natural. Las gramáticas formales aparecen en varios contextos diferentes: la lógica matemática, las ciencias de la computación y la lingüística teórica, frecuentemente con métodos e intereses divergentes.

En un lenguaje formal, a las cadenas formadas según las reglas de la gramática formal se las llama fórmulas bien formadas, y el conjunto de todas las fórmulas bien formadas constituye un lenguaje formal. Una gramática formal no describe el significado de las fórmulas bien formadas, sino solamente su forma. La teoría de los lenguajes formales estudia las gramáticas formales y los lenguajes formales, y es una rama de la matemática aplicada. Sus aplicaciones se encuentran en la ciencia computacional teórica, la lingüística, la semántica formal, la lógica matemática y otras áreas.

Una gramática formal es un conjunto de reglas para reescribir cadenas de caracteres, junto con un símbolo inicial desde el cual debe comenzar la reescritura. Por lo tanto, una gramática formal generalmente se piensa como una generadora de lenguajes. Sin embargo, a veces también puede ser usada como la base para un "reconocedor": una función que determina si una cadena cualquiera pertenece a un lenguaje o es gramaticalmente incorrecta.

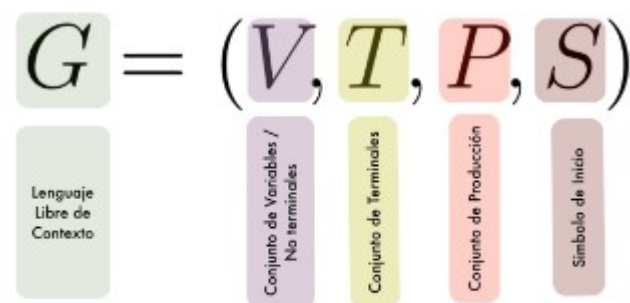


Imagen 1.0 - Gramática

Una gramática es:

$G = \{N, T, S, P\}$

donde:

N = Conjunto de no terminales.

T = Conjunto de terminales.

S = Símbolo inicial de la gramática

Y de aquí podemos partir a diferentes tipos de gramáticas:

1.1. Gramáticas lineales

Sólo tienen un no terminal en el lado derecho de una producción.

1.1.1. Gramática lineal por la derecha

Todas las producciones son de la forma:

$$A \rightarrow xB \text{ ó } A \rightarrow x$$

1.1.2. Gramática lineal por la izquierda:

Todas las producciones son de la forma:

$$A \rightarrow Bx \text{ ó } A \rightarrow x$$

Simplificación de una gramática

No se dispone de un método para construir la gramática equivalente **más sencilla** que genera un LIC.

Se pueden encontrar gramáticas equivalentes a una dada que no contengan cierto tipo de producciones que dificulten el trabajo:

- derivaciones vacías ($A \rightarrow \epsilon$)
- derivaciones unitarias ($A \rightarrow B$)
- símbolos inútiles (no utilizados en la derivación de ninguna cadena)

La gramática que resulta de realizar estas operaciones la denominaremos **gramática simplificada**.

Eliminación de producciones vacías

Una derivación es vacía si es del tipo $A \rightarrow \epsilon$

Si $A \xrightarrow{*} \epsilon$ se dice que A es **anulable**

Se puede dar una gramática equivalente a la dada sin producciones vacías (salvo cuando $\epsilon \in L(G)$).

Método:

1. buscar el conjunto de variables anulables;
2. añadir a las producciones otras en las que se hace explícita la posibilidad de producir ϵ^1 .

Ejemplo. Eliminar las producciones vacías de la siguiente gramática:

$$\begin{aligned} S &\rightarrow ABA \\ A &\rightarrow aA \mid \epsilon \\ B &\rightarrow bB \mid \epsilon \end{aligned}$$

¹Hay que eliminar todas las producciones duplicadas y de la forma $A \rightarrow A$ que aparezcan.

Eliminación de producciones unitarias

Son producciones en las que se deriva solamente un símbolo no terminal ($A \rightarrow B$).

El siguiente método es aplicable cuando no existen producciones vacías.

Método:

1. construir para cada variable $A \in V_N$ el conjunto V_A de variables accesibles desde A por derivaciones unitarias;
2. si $B \in V_A$ (y B tiene producciones no unitarias $B \rightarrow \alpha$), $\forall A$ tal que $B \in V_A$, añadir la regla $A \rightarrow \alpha$

Ejemplo. Eliminar las producciones unitarias de la siguiente gramática:

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow a \mid (E) \end{aligned}$$

Eliminación de los símbolos inútiles

Dada una GIC $G = (V_N, V_T, P, S)$, un símbolo $X \in V$ es **útil** si se utiliza en el proceso de generación de alguna cadena de $L(G)$.

Para que un símbolo X sea útil debe ser *generador* y *alcanzable*:

- X es *generador* si $X \xRightarrow{*} w$ para alguna cadena terminal w . (Todo símbolo terminal es generador, ya que w puede ser el mismo terminal, que se deriva en cero pasos).
- X es *alcanzable* si existe una derivación $S \xRightarrow{*} \alpha X \beta$ para algún α, β .

Material y equipo.

El material utilizado en la práctica es el siguiente:
Herramientas de software:

- * Oracle VM VirtualBox
- * Ubuntu 17.04
- * Python 3.6.2
- * eric6 Web Browser (QtWebKit)
- * JFlap

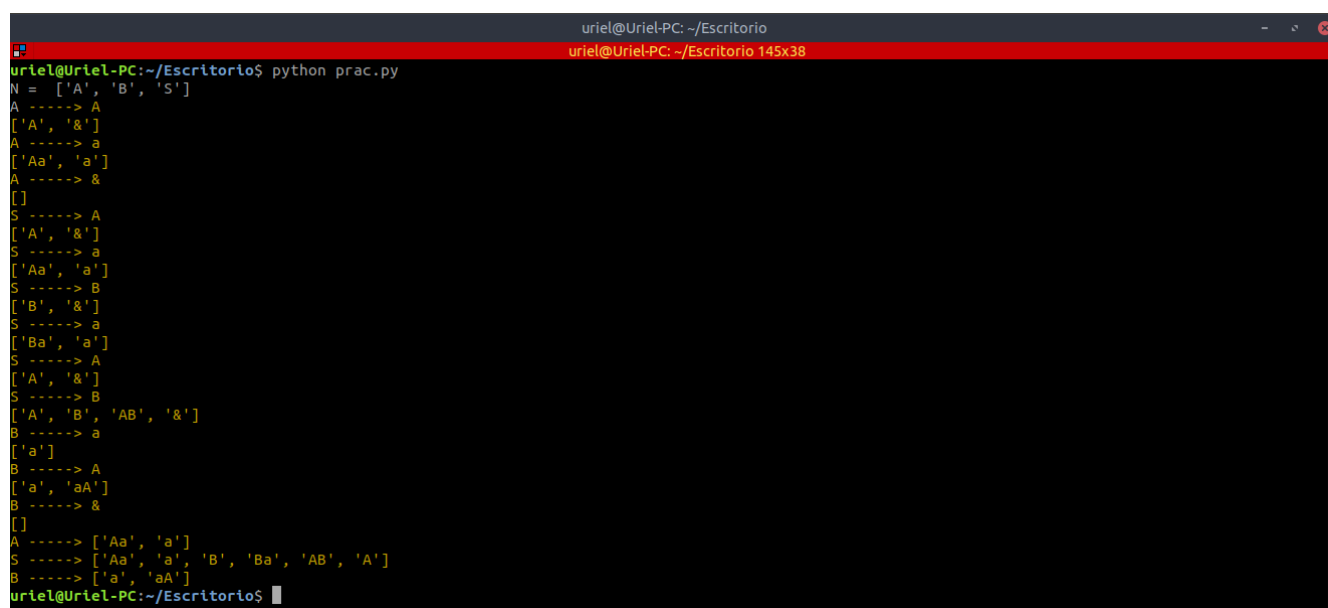
Herramientas de hardware:

- * Computadora personal.

Desarrollo de la práctica.

La práctica número trece y última del curso consistió en eliminar producciones vacías, para ello el profesor nos explicó previamente en el salón de clases, el como eliminar las producciones vacías, como las unitarios en el caso particular de esta práctica eliminaremos únicamente las producciones vacías, para esto se empleo el algoritmo explicado por el profesor.

Salida del programa.



```
uriel@Uriel-PC: ~/Escritorio
uriel@Uriel-PC: ~/Escritorio 145x38
uriel@Uriel-PC:~/Escritorio$ python prac.py
N = ['A', 'B', 'S']
A -----> A
['A', '&']
A -----> a
['Aa', 'a']
A -----> &
[]
S -----> A
['A', '&']
S -----> a
['Aa', 'a']
S -----> B
['B', '&']
S -----> a
['Ba', 'a']
S -----> A
['A', '&']
S -----> B
['A', 'B', 'AB', '&']
B -----> a
['a']
B -----> A
['a', 'aA']
B -----> &
[]
A -----> ['Aa', 'a']
S -----> ['Aa', 'a', 'B', 'Ba', 'AB', 'A']
B -----> ['a', 'aA']
uriel@Uriel-PC:~/Escritorio$
```

Código empleado.


```

from colorama import Fore, init

def eliminarProduccion(P, N):
    P1 = dict()
    D = []
    S = []
    j= ""
    for x in P:
        for y in P[x]:
            for l in y:
                print x,Fore.YELLOW+"----->",l
                if l == 'a'<=l<='z':
                    if not S:
                        S.append(l)
                    else:
                        S = anadirterminalEnmiLista(S,l);
                if l in N:
                    if not S:
                        S.append(l)
                        S.append('&')
                    else:
                        S = nanadirterminalEnmiLista(S,l);
                print S
                D = realizarUnion(D,S)
                S = []
            if '&' in D:
                D.remove('&')

        P1[x] = D
        D = []
    return P1

def nanadirterminalEnmiLista(miLista, Noterminal):
    variableAuxiliar = miLista
    miLista = anadirterminalEnmiLista(miLista,Noterminal)
    variableAuxiliar = realizarUnion(variableAuxiliar,miLista)
    return variableAuxiliar

def realizarUnion(A,B):
    variableAuxiliar= []
    for q in A:
        variableAuxiliar.append(q)
    for k in B:
        variableAuxiliar.append(k)

    return list(set(variableAuxiliar))

def anadirterminalEnmiLista(miLista, terminal):
    variableAuxiliar = []

```

```

s = ""
for q in miLista:
    for l in q:
        if l == '&':
            break
        s += l
    s += terminal
    variableAuxiliar.append(s)
    s = ""
return variableAuxiliar

def probadorDeAplicacion():
    P = {'S' : ['Aa', 'Ba', 'AB'], 'A' : ['Aa', '&'], 'B' : ['aA', '&']}
    NT = ['S', 'A', 'B']
    N= []
    variableAuxiliar = []
    for q in P:
        if '&' in P[q]:
            N.append(q)
    for q in NT:
        if q not in N:
            variableAuxiliar.append(q)
    for q in variableAuxiliar:
        for y in P[q]:
            for l in y:
                if l in N:
                    bandera = True
                else:
                    bandera = False
                    break
            if bandera:
                N.append(q)
    print "N = ", N
    P1 = eliminarProduccion(P,N)
    for q in P1:
        print q, "----->", P1[q]

probadorDeAplicacion()

```

Conclusión.

Las gramáticas se tratan de una estructura formal, la cual nos ayuda a especificar, de manera finita, el conjunto de cadenas de símbolos que constituyen el lenguaje de interés, por lo que resulta de gran ayuda que sean los más simples posibles, con el propósito de que estas sean programables, por esto es conveniente simplificar la gramáticas, eliminando producciones vacías, eliminación de símbolos inútiles, eliminación de producciones unitarias etc.

Referencias.

Es.wikipedia.org. (2017). Gramática formal. [online] Available at: https://es.wikipedia.org/wiki/Gram%C3%A1tica_formal [Accessed 11 Dic. 2017].

Scribd. (2017). Teoría de la Computación: Gramática libre de contexto. [online] Available at: <https://es.scribd.com/doc/97246521/Teoria-de-la-Computacion-Gramatica-libre-de-contexto> [Accessed 11 Dic. 2017].

Teoría de Lenguajes, Gramáticas y Autómatas Para Informáticos. (2000). [Place of publication not identified]: Digitalia, Inc.

Tello. (2017). *Simplificación de gramáticas independientes de contexto*. *Es.slideshare.net*. Retrieved 11 December 2017, from <https://es.slideshare.net/EduardoTello1/simplificacin-de-gramticas-independientes-de-contexto-26422757>