Autómatas y Lenguajes Formales
Proyecto

Raúl Daniel García Ramón
rauld.garcia95@gmail.com
zs22000520@estudiantes.uv.mx

22 de diciembre de 2022

Índice

1. Lenguaje.

A partir del siguiente lenguaje generar su autómata, su tabla parse y programar el algoritmo LR(1) para comprobar que cadenas son aceptadas o no por el lenguaje dado.

$$S \to xSy \tag{1}$$

$$S \to xyTyz \tag{2}$$

$$T \to \lambda \tag{3}$$

Primero se introduce un nuevo símbolo de inicio S':

$$S' \to S \tag{4}$$

Después se agregan las cerraduras del set de reglas reescritas con el marcador:

$$S \to x_{\wedge}Sy \tag{5}$$

$$S \to_{\wedge} xSy \tag{6}$$

$$S \to_{\wedge} xyTyz \tag{7}$$

$$S \to xy_{\wedge}Tyz \tag{8}$$

$$T \to_{\wedge} \lambda \tag{9}$$

A continuación empezamos a crear el autómata, primero definiendo el estado inicial 0 como sigue:

$$S' \to_{\wedge} S$$
$$S \to_{\wedge} xSz$$
$$S \to_{\wedge} xyTyz$$

Posteriormente continuamos creando los demás estados del autómata:

Estado 1
$$s = S$$
$$X = \{S' \to_{\wedge} S\}$$
$$Y = \{S' \to S_{\wedge}\}$$
$$Y^c = \{S' \to S_{\wedge}\}$$

Estado 2

$$s = x$$
$$X = \{S \to_\wedge xSz, S \to_\wedge xyTyz\}$$
$$Y = \{S \to x_\wedge Sz, S \to x_\wedge yTyz\}$$
$$Y^c = \{S \to x_\wedge Sz, S \to x_\wedge yTyz, S \to_\wedge xSz, S \to_\wedge xyTyz\}$$
$$s = x$$
$$X = \{S \to_\wedge xSz, S \to_\wedge xyTyz\}$$
$$Y = \{S \to x_\wedge Sz, S \to x_\wedge yTyz\}$$
$$Y^c = \{S \to x_\wedge Sz, S \to x_\wedge yTyz, S \to_\wedge xSz, S \to_\wedge xyTyz\}$$

El estado 2, con transición x lleva a si mismo.

Estado 3

$$s = S$$
$$X = \{S \to x_\wedge Sz\}$$
$$Y = \{S \to xS_\wedge z\}$$
$$Y^c = \{S \to xS_\wedge z\}$$

Estado 4

$$s = z$$
$$X = \{S \to xS_\wedge z\}$$
$$Y = \{S \to xSz_\wedge\}$$
$$Y^c = \{S \to xSz_\wedge\}$$

Estado 5

$$s = y$$
$$X = \{S \to x_\wedge yTyz\}$$
$$Y = \{S \to xy_\wedge Tyz\}$$
$$Y^c = \{S \to xy_\wedge Tyz, T \to_\wedge \lambda\}$$

Estado6

$$s = \lambda$$
$$X = \{T \to_\wedge \lambda\}$$
$$Y = \{T \to \lambda_\wedge\}$$
$$Y^c = \{T \to \lambda_\wedge\}$$

Estado 7

$$s = T$$
$$X = \{S \to xy_\wedge Tyz\}$$
$$Y = \{S \to xyT_\wedge yz\}$$
$$Y^c = \{S \to xyT_\wedge yz\}$$

Estado 8

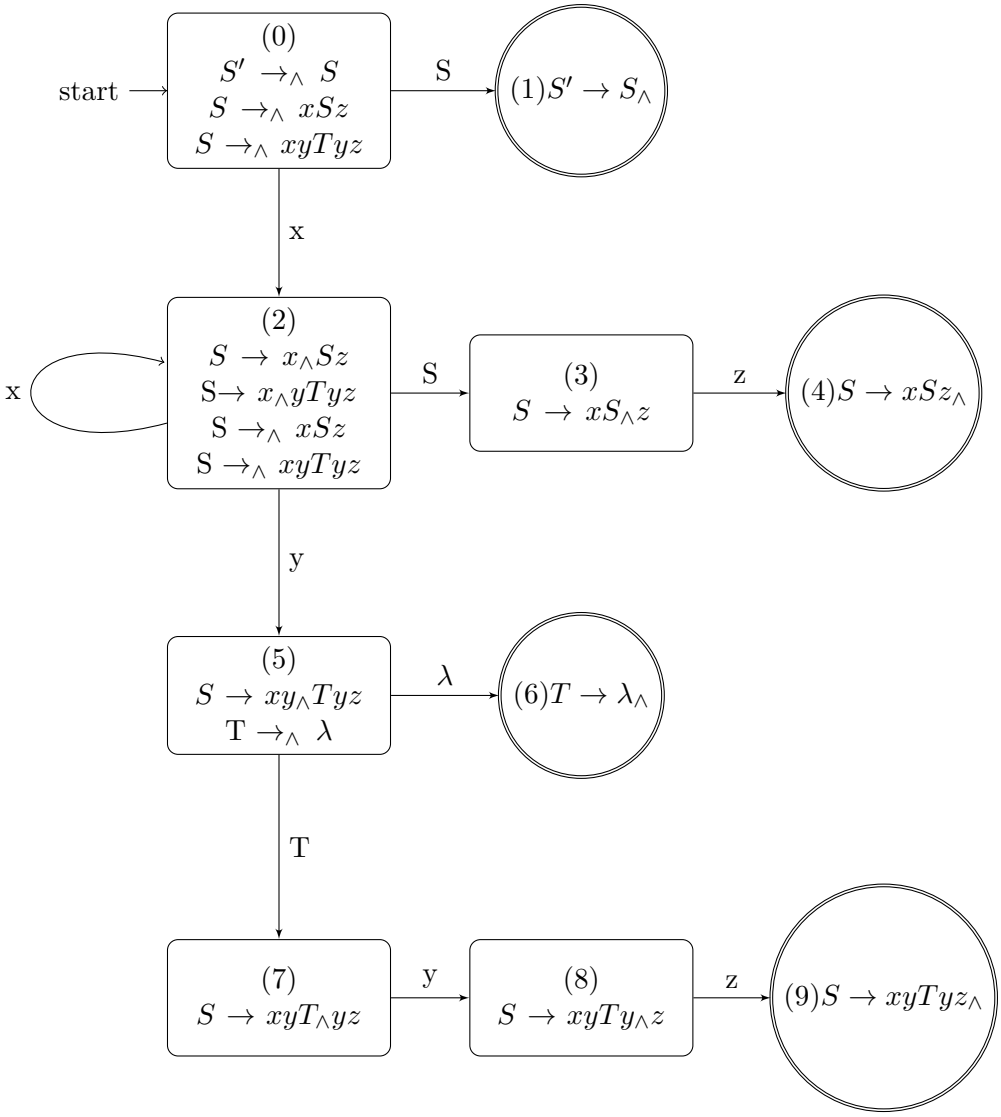
$$s = y$$
$$X = \{S \to xyT_{\wedge}yz\}$$
$$Y = \{S \to xyTy_{\wedge}z\}$$
$$Y^c = \{S \to xyTy_{\wedge}z\}$$

Estado 9

$$s = z$$
$$X = \{S \to xyTy_{\wedge}z\}$$
$$Y = \{S \to xyTyz_{\wedge}\}$$
$$Y^c = \{S \to xyTyz_{\wedge}\}$$

2. Autómata

El autómata generado por el lenguaje es el siguiente:



3. Parse Table

La tabla parse que se genera a partir del autómata anterior es la siguiente:

Token	x	y	z	EOS	S	T
0	shift 2				shift 1	
1				ACCEPT		
2	shift 2	shift 5			shift 3	
3			shift 4			
4			$S \rightarrow xSz$	$S \rightarrow xSz$		
5		$T \rightarrow \lambda$				shift 7
6		$T \rightarrow \lambda$				
7		shift 8				
8			shift 9			
9				$S \rightarrow xyTyz$		

4. Código

El algoritmo se programo en el lenguaje python, se utilizo la librería pandas para el manejo de la tabla parse y numpy para manejar el stack y la cadena de entrada.

```
import numpy as np
import pandas as pd

tablaparse=np.array([
    ["shift 2", False, False, False, "shift 1", False],
    [False, False, False, True, False, False],
    ["shift 2", "shift 5", False, False, "shift 3", False],
    [False, False, "shift 4", False, False, False],
    [False, False, "S_xSz", "S_xSz", False, False],
    [False, "T_lambda", False, False, False, "shift 7"],
    [False, "T_labmda", False, False, False, False],
    [False, "shift 8", False, False, False, False],
    [False, False, "shift 9", False, False, False],
    [False, False, "S_xyTyz", "S_xyTyz", False, False]
])

def tableentry (token,symbol):
```

```
18      return parsetable.loc[token,symbol]
19
20  parsetable=pd.DataFrame(tablaparse)
21  parsetable.columns=["x","y","z","EOS","S","T"]
22
23  #se inicializa el token en 0
24  token=0
25  #push del token al stack
26  stack=np.array([token])
27  #se lee la cadena y se le agrega EOS al final
28  cadena=list(input("Ingrese la cadena deseada:"))
29  cadena=np.append(cadena,"EOS")
30  #se lee el primer simbolo de la cadena y se borra de la misma
31  symbol=cadena[0]
32  cadena=np.delete(cadena,[0])
33  #se llama al valor de tabla parse
34  table=tableentry(token,symbol)
35  #mientras la tabla no de un true se ejecutara
36  while table != "True":
37      x=table.split()
38      #si la tabla regresa un shift se hace push al simbolo
39      #el token ahora es el indicado en el shift y se hace push de el
40      #y se lee el siguiente simbolo de la cadena
41      if x[0]=="shift":
42          stack=np.append(symbol,stack)
43          stack=np.append(x[1],stack)
44          token=int(x[1])
45          symbol=cadena[0]
46          cadena=np.delete(cadena,[0])
47
48      #si la tabla regresa False imprime que la cadena no se acepta
49      #y se detiene el programa
50      elif table=="False":
51          print("Cadena no aceptada.")
52          quit()
53
54      else:
55          #si la tabla regresa reduccion
56          #se hace pop a lo que esta del lado derecho de la regla
57          #token toma el valor del top del stack
```

```
58          #se hace push del lado izquierda de la regla
59          #token toma el valor de la tabla
60          #para el token que tenia con el lado izquierdo de la regla
61          #se hace push del nuevo token
62          if token==4:
63              stack=np.delete(stack,[0,1,2,3,4,5])
64              token=int(stack[0])
65              stack=np.append("S",stack)
66              table=tableentry(token,"S")
67              x=table.split()
68              token=x[1]
69              stack=np.append(token,stack)
70
71          elif token==9:
72              stack=np.delete(stack,[0,1,2,3,4,5,6,7,8,9])
73              token=int(stack[0])
74              stack=np.append("S",stack)
75              table=tableentry(token,"S")
76              x=table.split()
77              token=x[1]
78              stack=np.append(token,stack)
79
80          elif token==5:
81              token=int(stack[0])
82              stack=np.append("T",stack)
83              table=tableentry(token,"T")
84              x=table.split()
85              token=x[1]
86              stack=np.append(token,stack)
87
88      #se busca el nuevo valor en la tabla y se inicia otra vez el proceso
89      table=tableentry(int(token),symbol)
90
91  #si al terminar el while lo que queda de la cadena es EOS se acepta la cadena
92  #si no es asi se rechaza la cadena
93  if symbol=="EOS":
94      print("Cadena Aceptada.")
95
96  else:
97      print("Cadena no aceptada.")
```

A continuación se muestran unas pruebas realizadas con diferentes cadenas para ver el comportamiento general del programa:

```
Ingrese la cadena deseada:xxyyzz
Cadena Aceptada.

Ingrese la cadena deseada:xxxyyzzz
Cadena Aceptada.

Ingrese la cadena deseada:xyz
Cadena no aceptada.

Ingrese la cadena deseada:xyyz
Cadena Aceptada.

Ingrese la cadena deseada:xxyz
Cadena no aceptada.

Ingrese la cadena deseada:xxyyzzz
Cadena no aceptada.

Ingrese la cadena deseada:xxxyyzz
Cadena no aceptada.
```

Como se logra ver se aceptan únicamente las cadenas que si pueden ser generadas por el lenguaje, mientras que en el momento en que se detecta que una cadena no puede ser generada por el lenguaje, se imprime que la cadena no es aceptada y se detiene el computo sin necesidad de revisar el resto de la cadena puesto que ya se sabe que no es aceptada.

Referencias

Brookshear, J. G. (1989). *Theory of Computation, Formal Languages, Automata, and Complexity*. The Benjamin/Cummings Publishing Company, Inc.

Sipser, M. (2006). *Introduction to the Theory of Computation*. Thomson Course Technology, 2nd edition.