TEORÍA DE LA COMPUTACIÓN
Prof: Dr. HIram Calvo
Alumno: José Antonio M Faustinos
Implementación de un analizador LR(0)

El ejercicio propuesto consiste en que a partir de una gramática libre de contexto, el programa sea capaz de generar la tabla LR0 por medio del procedimiento que implica utilizar un AFD para la construcción de la misma y adicionalmente, utilizar dicha tabla para verificar si una cadena es aceptada por la gramática o no.

Para la implementación de este proceso, tenemos lo siguiente:

- La gramática libre de contexto será almacenada en un archivo de texto *.cfg*

```
S->zMNz
M->aMa
N->bNb
M->z
N->z
```

- Las reglas son modeladas por una lista como la siguiente:

```
['S',"zMNz",0]
```

En la que el primer elemento representa el símbolo no terminal a la izquierda de la producción, el segundo el conjunto de terminales y no terminales a la derecha de la producción y el tercero representa la posición del apuntador que nos permitirá ir produciendo las reglas que conformarán los estados, por ejemplo:

```
['S',"zMNz",0] Representa S->^zMNz
['S',"zMNz",1] Representa S->z^MNz
['S',"zMNz",2] Representa S->zM^Nz
etc...
```

- La estructura del autómata está conformado por una lista de estados. Cada estado está conformado por 4 elementos representados en una lista: Un identificador (que más tarde se convertirá en el id de la fila de la tabla LR0, un conjunto de reglas (lista de reglas), un conjunto de transiciones (diccionario con llave=caracter de transición y valor= id del estado al que se llega) y un booleano que indica si el estado es de aceptación.

```
[[1, [['$', 'S', 0], ['S', 'zMNz', 0]], {'S': 2, 'z': 3}, False],
[2, [['$', 'S', 1]], {}, True]...]
```

- Se implementaron funciones que nos permiten obtener las derivaciones de las reglas basadas en la posición del apuntador, esto con el fin de obtener el conjunto

de reglas que puede formar un nuevo estado. Estas reglas se someten a la prueba DK y si la pasan se manda llamar recursivamente la función generadora para cada regla hasta que se encuentra con el caso base de que sea una regla que genere un estado de aceptación. En caso de que la gramática no pase a prueba DK, la ejecución del programa termina.

- Haciendo uso del modelo del autómata ya completo, podemos completar la tabla LR0, la cual se implementa con una lista de listas.
- Hacemos uso de un vector “encabezado” auxiliar que contiene los símbolos terminales, los no terminales y el símbolo especial de “FDC” que posteriormente servirá tanto para presentar la tabla como para realizar la prueba de una cadena.
- Posteriormente, siguiendo el algoritmo de prueba de la tabla LR0 propuesto por Brookshear, se utiliza la tabla calculada para propagar una cadena y verificar si esta es aceptada por la gramática.
- Se hace uso de ciertas funciones auxiliares en el desarrollo del programa, como aquella para eliminar reglas repetidas de una lista, aquella que indica si un símbolo o una regla son terminales, etc. Todos ellos abstracciones de procedimientos necesarios para la elaboración de la tabla.
- El programa es capaz de manejar gramáticas que contienen lambda , representada por una diagonal / en el archivo .cfg. Se calculan las derivaciones de todos los casos donde aparece lambda y la transición S->/ se contempla para el correcto funcionamiento del programa.
- Se utilizo la libreria `prettytable` para la visualización amable de la tabla LR0, para la correcta ejecución del programa es necesario instalar este paquete previamente, lo que se puede hacer facilmente con l instrucción:

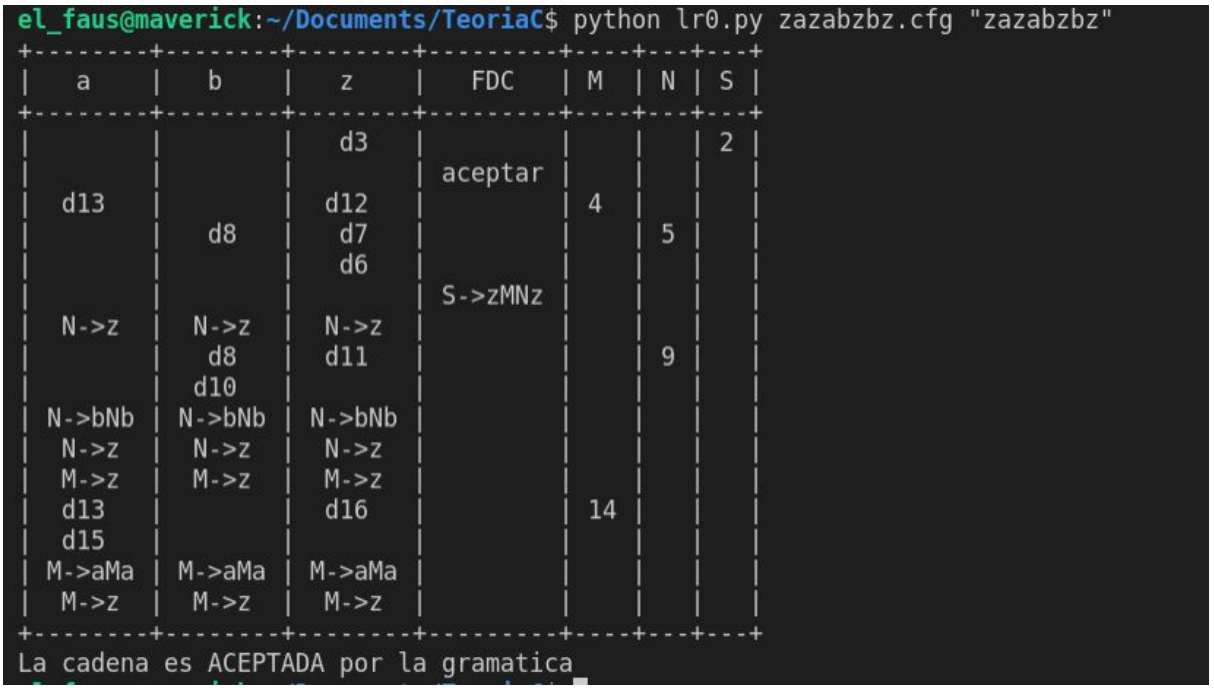
  pip install prettytable

Ejemplos de ejecución:

- Usando el archivo “zazabzbz.cfg” se realizaron las siguientes pruebas:

  zazabzbz.cfg

```
S->zMNz
M->aMa
N->bNb
M->z
N->z
```



```
el_faus@maverick:~/Documents/TeoriaC$ python lr0.py zazabzbz.cfg "zazabzbz"
+--------+--------+--------+---------+----+---+---+
|   a    |   b    |   z    |   FDC   | M  | N | S |
+--------+--------+--------+---------+----+---+---+
|        |        |   d3   |         |    |   | 2 |
|        |        |        | aceptar |    |   |   |
|  d13   |        |  d12   |         | 4  |   |   |
|        |   d8   |   d7   |         |    | 5 |   |
|        |        |   d6   |         |    |   |   |
|        |        |        | S->zMNz |    |   |   |
|  N->z  |  N->z  |  N->z  |         |    |   |   |
|        |   d8   |  d11   |         |    | 9 |   |
|        |  d10   |        |         |    |   |   |
| N->bNb | N->bNb | N->bNb |         |    |   |   |
|  N->z  |  N->z  |  N->z  |         |    |   |   |
|  M->z  |  M->z  |  M->z  |         |    |   |   |
|  d13   |        |  d16   |         | 14 |   |   |
|  d15   |        |        |         |    |   |   |
| M->aMa | M->aMa | M->aMa |         |    |   |   |
|  M->z  |  M->z  |  M->z  |         |    |   |   |
+--------+--------+--------+---------+----+---+---+
La cadena es ACEPTADA por la gramatica
```

La gramática pasa la prueba DK y la cadena es aceptada.

```
el_faus@maverick:~/Documents/TeoriaC$ python lr0.py zazabzbz.cfg "zazabzbzz"
+--------+--------+--------+---------+---+---+---+
|   a    |   b    |   z    |   FDC   | M | N | S |
+--------+--------+--------+---------+---+---+---+
|        |        |   d3   |         |   |   | 2 |
|        |        |        | aceptar |   |   |   |
|  d13   |        |  d12   |         | 4 |   |   |
|        |   d8   |   d7   |         |   | 5 |   |
|        |        |   d6   |         |   |   |   |
|        |        |        | S->zMNz |   |   |   |
|  N->z  |  N->z  |  N->z  |         |   |   |   |
|        |   d8   |  d11   |         |   | 9 |   |
|        |  d10   |        |         |   |   |   |
| N->bNb | N->bNb | N->bNb |         |   |   |   |
|  N->z  |  N->z  |  N->z  |         |   |   |   |
|  M->z  |  M->z  |  M->z  |         |   |   |   |
|  d13   |        |  d16   |         | 14|   |   |
|  d15   |        |        |         |   |   |   |
| M->aMa | M->aMa | M->aMa |         |   |   |   |
|  M->z  |  M->z  |  M->z  |         |   |   |   |
+--------+--------+--------+---------+---+---+---+
La gramatica no puede procesar la cadena
```

En este caso, la cadena de prueba no pertenece al lenguaje generado por la gramática.

Para el caso de la gramática:

```
S->A
S->B
A->xA
B->xBy
A->/
B->/
```

La cual no pasa la prueba DK, el programa lo indica y proporciona las reglas que están causando conflicto en la generación de un nuevo estado.

```
el_faus@maverick:~/Documents/TeoriaC$ python lr0.py Tarea.cfg "xxyy"
La gramatica no cumple la regla DK por conflicto con las siguientes reglas en el mismo estado:['B', 'xy', 1]['A', 'x', 1]
```

Se proporciona un pequeño conjunto de gramáticas con el que fue probado el programa, estas incluyen ejemplos tanto que pasan como no pasan la prueba DK.