

INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

TÍTULO DEL REPORTE

P R O G R A M A
A U T Ó M A T A
D E
P I L A

PARA OBTENER UN 10 EN EL REPORTE:

PRESENTA:

CONNOR URBANO MENDOZA

DOCENTES:

JUÁREZ MARTÍNEZ GENARO

INSTITUTO POLITÉCNICO NACIONAL



Estados Unidos Mexicanos
Ciudad de México
2023

Índice general

| | |
|--|------------|
| Introducción | III |
| 1 Desarrollo | 1 |
| 1.1 Análisis del problema principal | 1 |
| 1.2 Límites del problema | 2 |
| 1.3 Estrategia para atacar el problema | 3 |
| 1.4 Implementación | 4 |
| 2 Análisis de Resultados | 22 |
| 2.1 Capturas del programa en ejecución | 22 |
| 3 Conclusión | 40 |
| 3.1 Problemas iniciales | 40 |
| 3.1.1 Soluciones | 41 |
| 3.2 Complejidades | 42 |
| 4 Bibliografías | 43 |
| 5 Anexos | 44 |
| 5.1 PDA.py | 44 |

Índice de figuras

| | |
|--|---|
| 1.1 Funciones básicas del código incluyendo el main. | 8 |
|--|---|

| | | |
|------|--|----|
| 1.2 | 2 funciones de recorrido de la pila. | 19 |
| 2.1 | Inicio del programa en terminal. | 23 |
| 2.2 | Animación primera parte. | 24 |
| 2.3 | Segunda parte de animación y terminal. | 25 |
| 2.4 | Tercera parte de animación y terminal. | 26 |
| 2.5 | Cuarta parte de animación y terminal. | 27 |
| 2.6 | Quinta parte de animación y terminal. | 28 |
| 2.7 | Sexta parte de animación y terminal. | 29 |
| 2.8 | Séptima parte de animación y terminal. | 30 |
| 2.9 | Octava parte de animación y terminal. | 31 |
| 2.10 | Novena parte de animación y terminal. | 32 |
| 2.11 | Décima parte de animación y terminal. | 33 |
| 2.12 | Onceava parte de animación y terminal. | 34 |
| 2.13 | Doceava parte de animación y terminal. | 35 |
| 2.14 | Archivo de salida de transiciones.txt. | 36 |
| 2.15 | Nueva entrada aleatoria. | 37 |
| 2.16 | Inicio de archivo de trancisiones.txt. | 38 |
| 2.17 | Final de archivo de trancisiones.txt | 39 |

Introducción

Me complace presentarle el informe de mi práctica individual sobre la implementación de un autómatas de pila (PDA, por sus siglas en inglés) para reconocer el lenguaje libre de contexto $[0^n 1^n \mid n \geq 1]$. En esta práctica, se nos proporcionaron las instrucciones necesarias para resolver el problema planteado y se nos solicitaron ciertas características específicas para el programa.

El objetivo principal de esta práctica fue diseñar y desarrollar un autómatas de pila que pudiera reconocer correctamente las cadenas pertenecientes al lenguaje $[0^n 1^n \mid n \geq 1]$. Además, se nos pidió implementar ciertas funcionalidades adicionales para enriquecer la experiencia del usuario y facilitar la evaluación del autómatas.

Las características solicitadas para el programa fueron las siguientes:

1. La posibilidad de que el usuario ingrese la cadena manualmente o que sea generada automáticamente. En el segundo caso, se estableció como requisito que la cadena no pudiera exceder los 100,000 caracteres.
2. La generación de un archivo y la visualización en pantalla de las descripciones instantáneas (IDs) correspondientes a la evaluación del autómatas. Esta función permitirá analizar paso a paso el proceso de reconocimiento de la cadena ingresada.
3. La animación del autómatas de pila, siempre y cuando la cadena tenga una longitud menor o igual a 10 caracteres. Esta característica adicional brindará una representación visual más interactiva del funcionamiento del autómatas.

4. La inclusión de pantallas del programa en ejecución en el informe, mostrando todas las características solicitadas. Estas capturas de pantalla servirán para ilustrar y respaldar los resultados obtenidos durante la implementación.
5. La presentación del código de la implementación en formato LaTeX, en lugar de utilizar imágenes. Esto permitirá una visualización más clara y facilitará la revisión del código.

A lo largo de este informe, se detallarán los pasos seguidos para cumplir con los requisitos mencionados, incluyendo el diseño del autómata, la implementación del programa en Python, el registro de las descripciones instantáneas, la animación cuando corresponda y la generación del informe en LaTeX.

Capítulo 1

Desarrollo

1.1. Análisis del problema principal

El análisis del problema principal para resolver el problema del autómata de pila se centra en comprender y abordar los siguientes aspectos clave:

1. Lenguaje libre de contexto: El problema implica reconocer el lenguaje libre de contexto $[0^n 1^n \mid n \geq 1]$, que consiste en cadenas formadas por una secuencia de ceros seguida de una secuencia igual de unos, con al menos un par de ceros y unos. Es fundamental comprender las reglas y características de este lenguaje para diseñar un autómata de pila adecuado.
2. Autómata de pila: El autómata de pila es un modelo computacional que utiliza una pila para almacenar y procesar información durante la evaluación de una cadena. Es necesario estudiar y comprender el funcionamiento de este tipo de autómata, así como su estructura y las transiciones entre estados, con el fin de diseñar un autómata de pila que reconozca correctamente el lenguaje planteado.
3. Características adicionales: Además de la implementación básica del autómata de pila, se solicitan características adicionales, como la posibilidad de ingreso manual o automático de la cadena, la generación de descripciones instantáneas (IDs) para el seguimiento del proceso de evaluación, la animación del autómata en caso de cadenas cortas y la presentación del informe

en formato LaTeX. Estas características requieren un análisis adicional para determinar cómo implementarlas de manera eficiente y satisfactoria.

4. Complejidad del problema: Se menciona que la cadena generada aleatoriamente no puede exceder los 100,000 caracteres. Esto implica considerar la complejidad y eficiencia del algoritmo implementado, ya que se debe garantizar que el programa sea capaz de manejar cadenas de tamaño considerable sin afectar significativamente el rendimiento.

1.2. Límites del problema

En el problema del autómatas de pila, existen ciertos límites y consideraciones que debemos tener en cuenta para su resolución adecuada. Estos límites son fundamentales para garantizar el correcto funcionamiento del programa y la eficiencia en la evaluación de las cadenas ingresadas.

En primer lugar, se establece un límite máximo para la longitud de la cadena. Si la cadena es generada automáticamente, se determina que no puede exceder los 100,000 caracteres. Esta restricción nos permite gestionar adecuadamente cadenas de gran tamaño, evitando posibles problemas de rendimiento y optimizando el procesamiento de la información.

Por otro lado, es importante considerar el número mínimo de caracteres necesarios en la cadena. Dado que el lenguaje libre de contexto $[0^n 1^n \mid n \geq 1]$ requiere que haya al menos un par de ceros seguido de un par de unos, se establece un límite mínimo de 4 caracteres. Esto garantiza que la cadena tenga la estructura necesaria para formar los pares correspondientes y cumpla con las reglas del lenguaje.

Además, se solicita la animación del autómatas de pila en la implementación del programa. Sin embargo, esta animación se limita a cadenas de hasta 10 caracteres. Esta restricción nos permite visualizar de manera clara y comprensible el funcionamiento del autómatas, evitando la saturación de información en casos de cadenas demasiado largas.

1.3. Estrategia para atacar el problema

La estrategia para resolver el problema del autómata de pila consiste en diseñar e implementar un programa que simule el comportamiento de un autómata de pila para reconocer el lenguaje libre de contexto $[0^n 1^n \mid n \geq 1]$. A continuación, se presenta una descripción general de la estrategia a seguir:

1. Diseño del autómata de pila: Se debe definir la estructura y el comportamiento del autómata de pila. Esto implica determinar los estados, las transiciones, los símbolos de entrada, los símbolos de pila y las reglas de transición necesarias para reconocer el lenguaje $[0^n 1^n \mid n \geq 1]$. Es importante tener en cuenta las reglas específicas de este lenguaje, que requieren que el número de ceros sea igual al número de unos.
2. Implementación del programa: Se debe desarrollar el programa que simule el autómata de pila. Esto implica escribir el código en el lenguaje de programación elegido, utilizando las estructuras de datos adecuadas y siguiendo las reglas de transición definidas en el diseño del autómata.
3. Entrada de la cadena: El programa debe permitir al usuario ingresar una cadena o generarla aleatoriamente, dependiendo de las características solicitadas. En el caso de la generación aleatoria, se debe verificar que la cadena generada no exceda el límite máximo establecido.
4. Evaluación del autómata: El programa debe evaluar la cadena ingresada utilizando el autómata de pila implementado. Para ello, se aplicarán las reglas de transición correspondientes y se realizará un seguimiento de las descripciones instantáneas del autómata, que mostrarán el estado actual, los símbolos de entrada y los símbolos en la pila en cada paso.
5. Resultados y visualización: El programa debe mostrar en pantalla y guardar en un archivo las descripciones instantáneas del autómata durante la evaluación de la cadena. Además, si la longitud de la cadena es menor o igual a 10, se puede animar el autómata para una visualización más interactiva y

comprensible.

1.4. Implementación

Para desarrollar la implementación del problema, al autómata de pila determinista (PDA) en Python utilizando la biblioteca PIL (Python Imaging Library) para crear imágenes. El PDA en el código tiene una serie de transiciones definidas en forma de diccionario.

El código se divide en dos funciones principales: `proceso_recorrido` y `proceso_recorrido2`. Ambas funciones reciben una cadena, un estado inicial y un índice como argumentos.

La función `proceso_recorrido` realiza un recorrido de la cadena en el PDA sin graficar el proceso. Muestra información relevante en la consola y escribe las transiciones en un archivo de texto.

La función `proceso_recorrido2` realiza un recorrido de la cadena en el PDA y grafica el proceso en una imagen utilizando la biblioteca PIL. Dibuja rectángulos, texto y flechas para representar los diferentes estados y transiciones del PDA. También guarda la imagen generada en un archivo llamado `.animacion.png`.

Ambas funciones utilizan el diccionario PDA para determinar las transiciones del PDA. El diccionario PDA tiene como clave una tupla que representa el estado actual, el símbolo de entrada y el símbolo en la cima de la pila, y como valor una lista de transiciones posibles. Cada transición es una tupla que contiene el estado siguiente y el símbolo que se apunta en la pila.

Entrando más en detalles, iré mostrando paso a paso las partes del código:

1. La explicación de la parte básica del código empieza por las siguientes librerías:

a) `import random`

b) `from PIL import Image, ImageDraw, ImageFont`

c) `import sys`

Las librerías `random`, `Image`, `ImageDraw`, `ImageFont` y `sys` son módulos de Python que brindan funcionalidades adicionales para el desarrollo de aplicaciones.

La librería `random` proporciona funciones relacionadas con la generación de números aleatorios. Al importar esta librería, podemos utilizar métodos como `random.randint()` para generar números enteros aleatorios dentro de un rango específico. También podemos utilizar `random.choice()` para seleccionar aleatoriamente un elemento de una lista.

La librería `PIL` (Python Imaging Library) es una biblioteca popular para el procesamiento de imágenes en Python. Al importar los módulos `Image`, `ImageDraw` y `ImageFont` de `PIL`, obtenemos funcionalidades para crear, editar y manipular imágenes. Podemos cargar imágenes existentes, crear nuevas imágenes, dibujar formas y textos en ellas, aplicar filtros y efectos, y mucho más.

El módulo `sys` proporciona funciones y variables relacionadas con la funcionalidad del intérprete de Python y el sistema en el que se está ejecutando. Al importar este módulo, podemos acceder a variables como `sys.argv`, que contiene una lista de los argumentos pasados al script de Python desde la línea de comandos. También podemos utilizar `sys.exit()` para finalizar la ejecución del programa en cualquier momento.

Se definen algunas variables iniciales, como el tamaño de la imagen y el tamaño del cuadrado para la visualización gráfica (estas variables son utiliza-

das más adelante para crear una imagen en blanco), así como el diccionario PDA que define las transiciones del autómata de pila.

Se define la función `generar_cadena_aleatoria(tamano)` que genera una cadena aleatoria de tamaño compuesta por los caracteres "0" y "1". Esta función es utilizada más adelante para generar una cadena aleatoria si el usuario no proporciona una.

Se definen las funciones de recorrido 1 y recorrido 2. Estas funciones no están incluidas en el código que has proporcionado, por lo que no puedo proporcionar una explicación detallada sobre ellas.

Se crea una pila vacía y se agrega el símbolo 'Z0' como elemento superior de la pila. Luego se imprime el contenido de la pila.

Se solicita al usuario que ingrese una cadena de entrada. Si el usuario no ingresa ninguna cadena y simplemente presiona Enter, se genera una cadena aleatoria utilizando la función `generar_cadena_aleatoria()`.

Se realiza el proceso de recorrido utilizando la función `proceso_recorrido()` si se generó una cadena aleatoria, o utilizando la función `proceso_recorrido2()` si se ingresó una cadena manualmente. Estas funciones son responsables de validar si la cadena es válida según las reglas del autómata de pila.

Dependiendo del resultado obtenido, se imprime un mensaje indicando si la cadena es válida o no.

Luego, se inicia un ciclo `for` que recorre cada una de las palabras de la línea actual y se realiza una serie de acciones por cada palabra. En resumen, la función evalúa cada palabra en el DFA y registra las palabras encontradas en los archivos de salida, dependiendo de su categoría y su frecuencia en el texto de entrada. La parte del código que falta en el último `for` es probablemente la que contiene la lógica principal para evaluar cada palabra en el DFA y llevar un registro de las palabras encontradas.

A continuación se presentan las partes de código correspondientes a la explicación anterior: Observar Figura 1.1.

```
1 #Teoria de la computacion
2 #Automata de pila
3 #Alumno: Connor Urbano Mendoza
4
5 import random
6 from PIL import Image, ImageDraw, ImageFont
7 import sys
8
9 # Tama o de la imagen y tama o del cuadrado
10 image_size = (400, 400)
11 square_size = 100
12
13 # Crear una imagen en blanco
14 image = Image.new("RGB", image_size, "white")
15 draw = ImageDraw.Draw(image)
16
17
18
19 # Definir el PDA con sus transiciones
20 PDA = {
21     ('q', '0', 'Z0'): [('q', 'X')],
22     ('q', '0', 'X'): [('q', 'X')],
23     ('q', '1', 'X'): [('p', '')],
24     ('p', '1', 'X'): [('p', 'Z0')],
25     ('p', '', 'Z0'): [('f', 'Z0')]
26 }
27
28 def generar_cadena_aleatoria(tamano):
29     mitad = tamano // 2
30     ceros = '0' * mitad
31     unos = '1' * mitad
32     if tamano % 2 != 0:
33         ceros += random.choice(['0', '1'])
34     cadena = ceros + unos
35     return cadena
36
```

```
37 #-----
38 Funciones de recorrido 1 y 2 (no se anexan aqui ya que
    se explicaran con mayor detenimiento)
39 #-----
40 #Main
41 pila = []
42
43 elemento_superior = 'Z0'
44 pila.append(elemento_superior)
45 print("Contenido de la pila:", pila)
46 estado = 'q'
47 i = 0
48
49 cadena = input("Ingresa una cadena (o presiona Enter
    para generar una cadena aleatoria): ")
50
51 if not cadena:
52     tamano = random.randint(1, 10)
53     cadena = generar_cadena_aleatoria(tamano)
54     resultado = proceso_recorrido(cadena, estado, i)
55
56     if resultado:
57         print("La cadena es v lida.")
58     else:
59         print("La cadena no es v lida.")
60
61 else:
62     resultado = proceso_recorrido2(cadena, estado, i)
63
64     if resultado:
65         print("La cadena es v lida.")
66     else:
67         print("La cadena no es v lida.")
```

Figura 1.1: Funciones básicas del código incluyendo el main.

2. La siguiente parte del código corresponde a dos funciones: `proceso_recorrido` y `proceso_recorrido2`, las cuales son utilizadas para recorrer y procesar una cadena en un autómata de pila.

En la función `proceso_recorrido`, se realiza el recorrido de la cadena paso a paso en el autómata de pila. Se itera sobre cada símbolo de la cadena y se comprueba si hay una transición válida en el autómata para el estado actual, el símbolo actual y el tope de la pila. Si existe una transición válida, se actualiza el estado actual, se realiza alguna operación en la pila (como agregar o eliminar elementos) y se avanza al siguiente símbolo de la cadena. En cada paso, se imprimen mensajes en la consola para mostrar información sobre el símbolo actual, el contenido de la pila y el estado actual. Además, se escribe el estado actual, el símbolo actual y el contenido de la pila en un archivo llamado "transiciones.txt". Si en algún momento no se encuentra una transición válida, se escribe `.Error.`^{en} el archivo y se retorna `False`.

La función `proceso_recorrido2` tiene una funcionalidad similar a `proceso_recorrido`, pero además de realizar el recorrido de la cadena, genera una imagen visual del autómata de pila y las transiciones que se van realizando. Utiliza la biblioteca PIL (Python Imaging Library) para crear y modificar la imagen. En cada paso del recorrido, se dibuja un rectángulo que representa el estado actual, se muestran diferentes apartados de información en la imagen (como la cadena ingresada, la parte por ingresar, la entrada actual, etc.), se dibujan flechas y texto para indicar las transiciones y se guardan los cambios en un archivo de imagen llamado "animacion.png". Además, se espera a que el usuario presione Enter para avanzar al siguiente paso del recorrido.

Ambas funciones realizan un recorrido de la cadena en el autómata de pila, pero `proceso_recorrido2` proporciona una representación visual más elaborada y detallada del proceso.

Dicha sección la podemos ver en la siguiente parte. Observar figura 1.2.

```
1 #Funcion de recorrido sin graficar
2 def proceso_recorrido(cadena, estado, i):
```

```
3     ruta_archivo = "C:\\Users\\soyco\\OneDrive\\
4         Documents\\ESCOM\\sem4\\Teoria\\P2\\Prog4\\
5         output\\trancisiones.txt"
6
7     with open(ruta_archivo, "w") as archivo:
8         estado_actual=estado
9
10        while i < (len(cadena)+1):
11            if i == (len(cadena)):
12                symbol=""
13            else:
14                symbol = cadena[i]
15            cadena_pila = ''.join(pila)
16
17            cima_de_pila = pila[-1]
18            archivo.write("(" + estado_actual + ", " + symbol
19                + ", " + cadena_pila + ") | -")
20            if (estado_actual, symbol, cima_de_pila)
21                in PDA:
22                transiciones = PDA[(estado_actual,
23                    symbol, cima_de_pila)]
24                estado_siguiente = transiciones[0][0]
25                letra_apuntada = transiciones[0][1]
26                if(letra_apuntada == ''):
27                    cima_de_pila = pila.pop()
28                elif(letra_apuntada == 'Z0'):
29                    cima_de_pila = pila.pop()
30                else:
31                    pila.append(letra_apuntada)
32
33                estado_actual=estado_siguiente
34                i += 1
35            else:
36                archivo.write("Error")
37                return False
38            archivo.write("(" + estado_actual + ", " + symbol + ", "
39                + cadena_pila + ") ")
40            return True
41
42 #Funcion de recorrido para graficar
43 def proceso_recorrido2(cadena, estado, i):
44
```

```
37 ruta_archivo = "C:\\Users\\soyco\\OneDrive\\
    Documents\\ESCOM\\sem4\\Teoria\\P2\\Prog4\\
    output\\trancisiones.txt"
38 with open(ruta_archivo, "w") as archivo:
39     estado_actual=estado
40     cadena_grafica = cadena
41
42     while i < (len(cadena)+1):
43
44         # Coordenadas del rect ngulo
45         rect_width = 40
46         rect_height = 140
47         x1 = ((image_size[0] - rect_width) // 2)
48         desplazamientoa_abajo=60
49         y1 = ((image_size[1] - rect_height) // 2)
50             +desplazamientoa_abajo
51         x2 = x1 + rect_width
52         y2 = y1 + rect_height
53
54         # Dibujar el rect ngulo en la imagen
55         draw.rectangle([(x1, y1), (x2, y2)],
56             outline="black")
57         #-----Dibujamos apartados-----
58             -----
59
60         # Dibujamos apartados de cadena ingresada
61         text = "Cadena ingresada:"
62         font = ImageFont.truetype("arial.ttf", 16)
63
64         # Dibujar el texto en la imagen
65         draw.text((50, 40), text, fill="black",
66             font=font)
67
68         # Dibujar ecadenal texto en la imagen
69         draw.text((200, 40), cadena, fill="black",
70             font=font)
71
72         # Dibujamos apartados de por ingresar
73         text = "Por ingresar:"
```



```
70         # Dibujar el texto en la imagen
71         draw.text((50, 60), text, fill="black",
72                   font=font)
73
74         #Dibujamos primer caracter
75         j=0
76
77         cadena_grafica = cadena_grafica[1:] #
78         # Obtener la subcadena sin la primera
79         letra
80         draw.text((209, 61), cadena_grafica, fill=
81         "black", font=font)
82         # Dibujamos apartados de entrada
83         text = "Entrada:"
84         font2 = ImageFont.truetype("arial.ttf",
85         20)
86
87         # Dibujar el texto en la imagen
88         draw.text((50, 90), text, fill="black",
89         font=font2)
90
91         #Coordenada de incremento para cruces
92         cross_y = ((y1 + y2) // 2)+50) - ((i+1)*
93         19)
94         #Dibujamos caso base
95         draw.text((189, ((y1 + y2) // 2)+50)), "
96         Z0", fill="blue", font=font2)
97
98         if i == (len(cadena)):
99             symbol=""
100             #Dibujamos primer caracter y la
101             entrada
102             # Dibujar el texto en la imagen
103             draw.text((140, 91), "' '", fill="
104             green", font=font2)
105             draw.text((200, 61), "Vacio (E)", fill=
106             "red", font=font)
107             #Dibujamos felcha de conversion
108             draw.text((155, 91), ">", fill="
```

```

    black", font=font2)
99     else:
100         symbol = cadena[i]
101         number=int(symbol)
102         # Dibujar el texto en la imagen
103         draw.text((140, 91), symbol, fill="red",
104                  font=font2)
105         draw.text((200, 61), symbol, fill="red",
106                  font=font)
107         #Dibujamos felcha de conversion
108         draw.text((155, 91), " >", fill="black", font=font2)
109
110     cadena_pila = ''.join(pila)
111     cima_de_pila = pila[-1]
112
113     archivo.write("(" + estado_actual + "," + symbol +
114                  "+", "+cadena_pila+" ) | -")
115     if (estado_actual, symbol, cima_de_pila)
116     in PDA:
117         transiciones = PDA[(estado_actual,
118                               symbol, cima_de_pila)]
119         estado_siguiete = transiciones[0][0]
120         letra_apuntada = transiciones[0][1]
121         if(letra_apuntada == ''):
122             cima_de_pila = pila.pop()
123             #Dibujamos letra
124             draw.text((197, 91), "-X", fill="blue", font=font2)
125             draw.text((197, 125), "^", fill="red", font=font2)
126             draw.text((199, 141), "|", fill="red", font=font2)
127             if len(cadena) % 2 != 0:
128                 cross_y = (((y1 + y2) // 2) +
129                             50) - ((i+1)*19)-19)
130             else:
131                 cross_y = (((y1 + y2) // 2) +
132                             50) - ((i+1)*19)
133             # Coordenadas de la ltima cruz

```

```

127         agregada
128         last_cross_x = 200
129
130         last_cross_y = cross_y + ((38*i)-
131             (19*len(cadena)))
132         last_cross_size = 12
133
134         # Guardar las coordenadas y el
135         # tama o de la ltima cruz
136         last_cross_coords = [(
137             last_cross_x - last_cross_size)
138             -1, last_cross_y + 21), ((
139             last_cross_x + last_cross_size)
140             +1, (last_cross_y + 37)+1)]
141         # Eliminar la ltima cruz
142         draw.rectangle(last_cross_coords,
143             outline="white", fill="white")
144
145     elif(letra_apuntada == 'Z0' and number
146         ==1 and symbol!=''):
147         cima_de_pila = pila.pop()
148         #Dibujamos letra
149         draw.text((197, 91), "-X", fill="
150             blue", font=font2)
151         draw.text((197, 125), "^", fill="
152             red", font=font2)
153         draw.text((199, 141), "|", fill="
154             red", font=font2)
155         if len(cadena) % 2 != 0:
156             cross_y = (((y1 + y2) // 2)+
157                 50) - ((i+1)*19)-19)
158         else:
159             cross_y = (((y1 + y2) // 2)+
160                 50) - ((i+1)*19)
161         # Coordenadas de la ltima cruz
162         agregada
163         last_cross_x = 200
164         #print(cross_y)
165         last_cross_y = cross_y + ((38*i)-
166             (19*len(cadena)))

```

```
151         last_cross_size = 12
152
153         # Guardar las coordenadas y el
154         # tamaño de la ltima cruz
155         last_cross_coords = [(
156             last_cross_x - last_cross_size)
157             -1, last_cross_y + 21), ((
158             last_cross_x + last_cross_size)
159             +1, (last_cross_y + 37)+1)]
160
161         # Eliminar la ltima cruz
162         draw.rectangle(last_cross_coords,
163             outline="white", fill="white")
164     elif(letra_apuntada == 'Z0' and symbol
165          == ''):
166         cima_de_pila = pila.pop()
167         #Dibujamos letra
168         draw.text((197, 91), "Z0", fill="
169             blue", font=font2)
170         draw.text((197, 125), "^", fill="
171             red", font=font2)
172         draw.text((199, 141), "|", fill="
173             red", font=font2)
174         # Coordenadas de la ltima cruz
175         # agregada
176         last_cross_x = 200
177         #print(cross_y)
178         last_cross_y = cross_y + ((38*i)-
179             (19*len(cadena)))
180         last_cross_size = 12
181
182         # Guardar las coordenadas y el
183         # tamaño de la ltima cruz
184         last_cross_coords = [(
185             last_cross_x - last_cross_size)
186             -1, last_cross_y + 21), ((
187             last_cross_x + last_cross_size)
188             +1, (last_cross_y + 37)+1)]
189
190         # Eliminar la ltima cruz
191         draw.rectangle(last_cross_coords,
192             outline="white", fill="white")
```

```
173         else:
174             pila.append(letra_apuntada)
175             #Dibujamos letra
176             draw.text((197, 91),
177                       letra_apuntada, fill="blue",
178                       font=font2)
179             draw.text((201, 121), "|", fill="
180                       green", font=font2)
181             draw.text((197, 141), "v", fill="
182                       green", font=font2)
183             #Dibujamos cruz
184             draw.text((194, cross_y), "X",
185                       fill="black", font=font2)
186
187         estado_actual=estado_siguiete
188
189         # Guardar la imagen en la ruta
190         especificada
191         image.save("C:\\Users\\soyco\\OneDrive
192                   \\Documents\\ESCOM\\sem4\\Teoria\\
193                   P2\\Prog4\\output\\animacion.png")
194         if sys.stdin.isatty():
195             print("Presiona Enter para
196                   continuar...")
197             sys.stdin.readline()
198         else:
199             input("Presiona Enter para
200                   continuar...")
201         i += 1
202     else:
203         archivo.write("Error")
204         #Dibujamos letra
205         # Cadena invalida
206         draw.text((200, 40), cadena, fill="red
207         ", font=font)
208         draw.text((197, 91), "ERROR", fill="
209         blue", font=font2)
210
211         image.save("C:\\Users\\soyco\\OneDrive
212                   \\Documents\\ESCOM\\sem4\\Teoria\\
```

```

200         P2\\Prog4\\output\\animacion.png")
201     if sys.stdin.isatty():
202         print("Presiona Enter para
203             continuar...")
204         sys.stdin.readline()
205     else:
206         input("Presiona Enter para
207             continuar...")
208     return False
209
210     # Aqu se realiza el borrado del
211     contenido de la imagen
212     draw.rectangle([(11, 11), (389, 180)],
213                   fill="white")
214     image.save("C:\\Users\\soyco\\OneDrive\\
215               Documents\\ESCOM\\sem4\\Teoria\\P2\\
216               Prog4\\output\\animacion.png")
217
218     archivo.write("(" + estado_actual + ", " + symbol + ", "
219                 + cadena_pila + ")")
220
221     # Coordenadas del rect ngulo
222     rect_width = 40
223     rect_height = 140
224     x1 = ((image_size[0] - rect_width) // 2)
225     desplazamiento_a_abajo=60
226     y1 = ((image_size[1] - rect_height) // 2) +
227         desplazamiento_a_abajo
228     x2 = x1 + rect_width
229     y2 = y1 + rect_height
230
231     # Dibujar el rect ngulo en la imagen
232     draw.rectangle([(x1, y1), (x2, y2)], outline="
233                   black")
234
235     # Dibujamos apartados de cadena ingresada
236     text = "Cadena ingresada:"
237     font = ImageFont.truetype("arial.ttf", 16)
238
239
```

```
230     # Dibujar el texto en la imagen
231     draw.text((50, 40), text, fill="black", font=
        font)
232
233     # Dibujar ecadenal texto en la imagen
234     draw.text((200, 40), cadena, fill="green",
        font=font)
235
236     text = "Estado Final"
237
238     # Dibujar el texto en la imagen
239     draw.text((50, 60), text, fill="black", font=
        font)
240
241     # Dibujamos apartados de entrada
242     text = "Ya sacamos a Z0"
243     # Dibujar el texto en la imagen
244     draw.text((50, 92), text, fill="black", font=
        font)
245     #Dibujamos felcha de conversion
246     draw.text((178, 91), "      >", fill="black",
        font=font2)
247
248
249     draw.text((235, 91), "Z0", fill="blue", font=
        font2)
250
251     image.save("C:\\Users\\soyco\\OneDrive\\
        Documents\\ESCOM\\sem4\\Teoria\\P2\\Prog4\\
        output\\animacion.png")
252     if sys.stdin.isatty():
253         sys.stdin.readline()
254     else:
255         input("Presiona Enter para continuar...")
256     return True
```

Figura 1.2: 2 funciones de recorrido de la pila.

El proceso `proceso_recorrido2` realiza un recorrido y dibuja una representación gráfica de cada paso de un proceso en un autómata de pila. Aquí se explica detalladamente su funcionamiento:

- a) Se establece la ruta del archivo de salida donde se guardarán las transiciones del autómata en la variable `ruta_archivo`.
- b) Se inicializan las variables `estado_actual` y `cadena_grafica` con los valores de estado y cadena respectivamente.
- c) Se inicia un bucle `while` que se ejecutará mientras `i` sea menor que la longitud de la cadena más uno (para incluir el caso base de cadena vacía).
- d) Se definen las coordenadas (`x1`, `y1`, `x2`, `y2`) del rectángulo que se dibujará en la imagen. Este rectángulo representa el estado actual del autómata.
- e) Se dibujan diferentes apartados de texto en la imagen, como la cadena ingresada, la por ingresar, la entrada y el estado actual.
- f) Se comprueba si se ha llegado al final de la cadena (caso base). En caso afirmativo, se dibuja el último símbolo de la cadena y se muestra la transición hacia el estado final.
- g) En caso contrario, se obtiene el símbolo actual de la cadena y se verifica si existe una transición en el autómata desde el estado actual, utilizando el símbolo actual y el símbolo en la cima de la pila.
- h) Si existe una transición válida, se obtiene el estado siguiente y el símbolo a apuntar en la pila. Dependiendo de la situación, se realiza algu-

na de las siguientes acciones:

- 1) Si el símbolo a apuntar es una cadena vacía (""), se saca el símbolo de la cima de la pila y se dibuja en la imagen. También se elimina la última cruz dibujada en la posición correspondiente.
 - 2) Si el símbolo a apuntar es 'Z0' y el número del símbolo actual es 1, se saca el símbolo de la cima de la pila y se realiza las mismas acciones que en el caso anterior.
 - 3) Si el símbolo a apuntar es 'Z0' y el símbolo actual es una cadena vacía, se saca el símbolo de la cima de la pila y se dibuja en la imagen.
 - 4) En cualquier otro caso, se agrega el símbolo a apuntar a la pila y se dibuja en la imagen.
 - 5) Se actualiza el estado actual con el estado siguiente obtenido.
 - 6) Se guarda la imagen en la ruta especificada.
 - 7) Se espera la interacción del usuario presionando Enter para continuar.
 - 8) Se incrementa el contador i para pasar al siguiente símbolo de la cadena.
- i)* Si no existe una transición válida, se escribe `.Error.` en el archivo de salida y se muestra un mensaje de error en la imagen.
 - j)* Se borra el contenido de la imagen, preparándola para el siguiente paso del recorrido.
 - k)* Se escribe la última transición en el archivo de salida, que corresponde al estado actual, el símbolo actual y la cadena de la pila.

- l)* Se vuelven a dibujar los apartados de texto en la imagen para representar el estado final.
- m)* Se guarda.

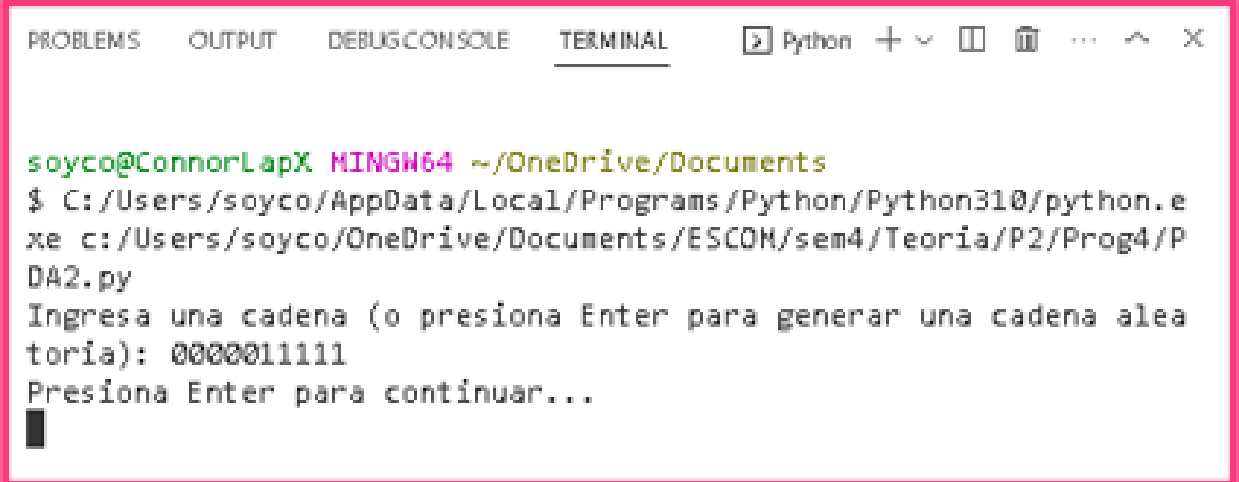
Capítulo 2

Análisis de Resultados

2.1. Capturas del programa en ejecución

A continuación se presenta en orden el proceso de ejecución del programa, donde primeramente se muestra el código en ejecución con un ejemplo chico.

- a) Iniciamos el programa, donde nos pide que introduzcamos una cadena o de lo contrario al presionar enter se generara una aleatoria, para nuestro ejemplo digitamos 0000011111 que vienen siendo 5 ceros y 5 unos. Observar la Figura 2.1.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL Python + - [ ] [X] ... ^ X

soyco@ConnorLapX MINGW64 ~/OneDrive/Documents
$ C:/Users/soyco/AppData/Local/Programs/Python/Python310/python.exe
c:/Users/soyco/OneDrive/Documents/ESCON/sem4/Teoria/P2/Prog4/P
DA2.py
Ingresa una cadena (o presiona Enter para generar una cadena alea
toria): 0000011111
Presiona Enter para continuar...
█
```

Figura 2.1: Inicio del programa en terminal.

- b) Aquí se puede visualizar la animación del autómata pila. Se inserta X. Observar la Figura 2.2.

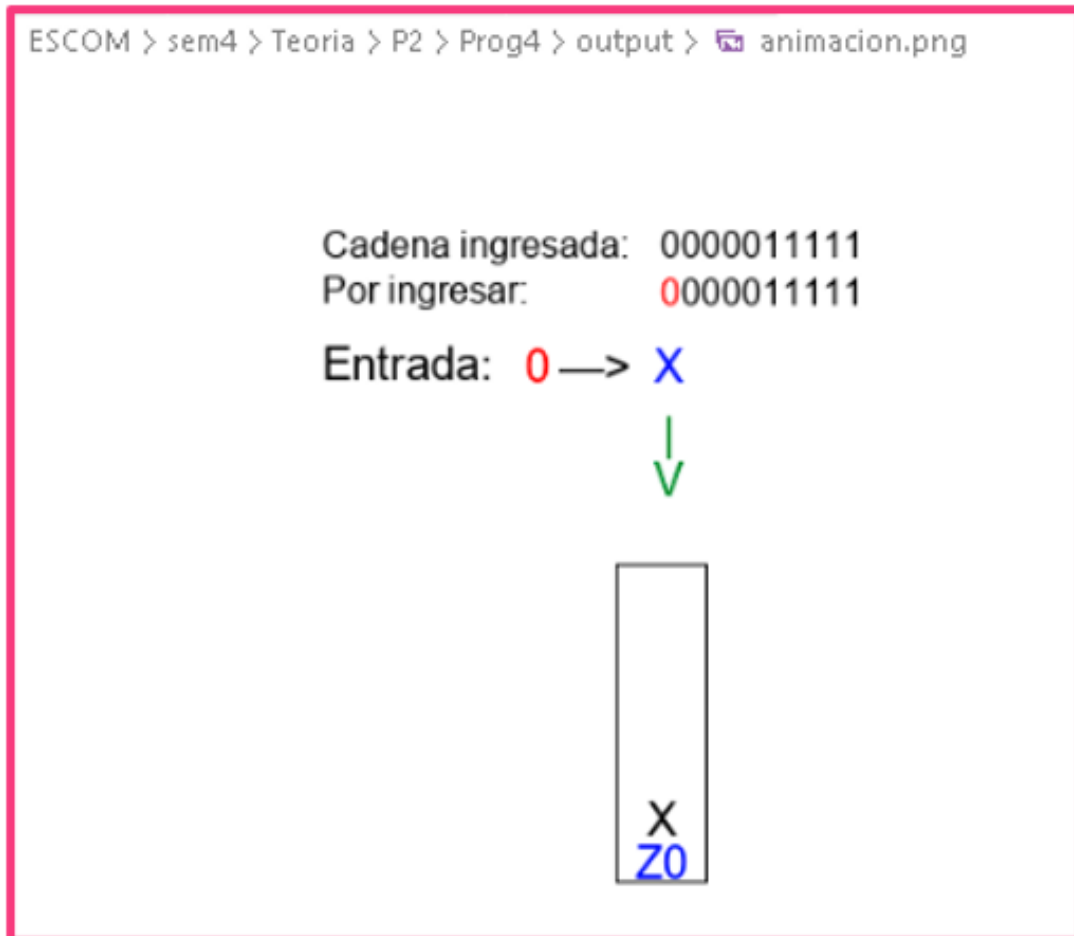


Figura 2.2: Animación primera parte.

- c) Segunda parte de animación y terminal, se inserta X. Observar la Figura 2.3.

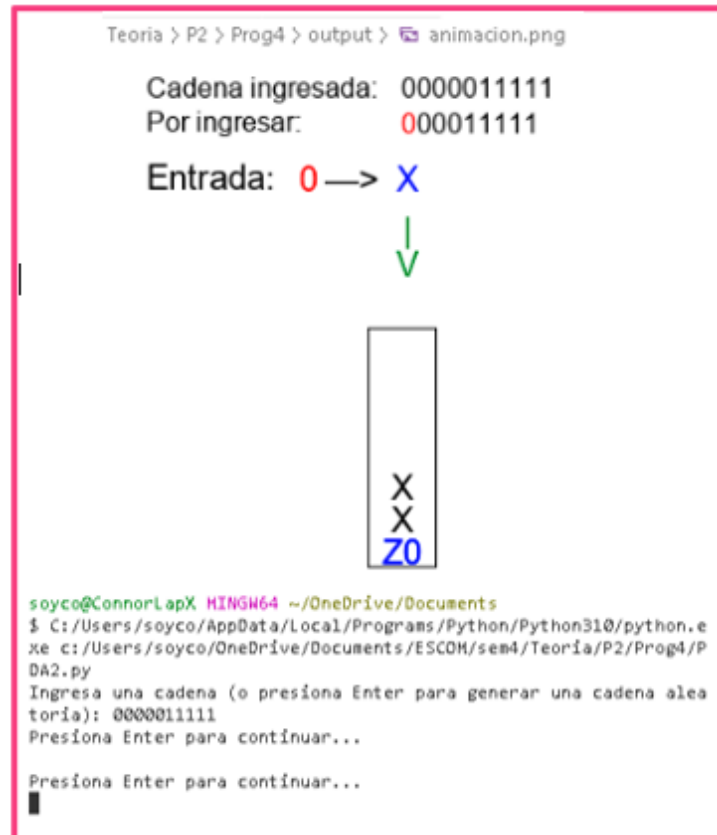


Figura 2.3: Segunda parte de animación y terminal.

- d) Tercera parte de animación y terminal, se inserta X. Observar la Figura 2.4.



```
Teoria > P2 > Prog4 > output > animacion.png

Cadena ingresada: 0000011111
Por ingresar: 00011111
Entrada: 0 → X
          ↓
          X
          X
          X
          Z0

soyc@ConnorLapX MINGW64 ~/OneDrive/Documents
$ C:/Users/soyco/AppData/Local/Programs/Python/Python310/python.exe c:/Users/soyco/OneDrive/Documents/ESCOM/sem4/Teoria/P2/Prog4/PA2.py
Ingresa una cadena (o presiona Enter para generar una cadena aleatoria): 0000011111
Presiona Enter para continuar...

Presiona Enter para continuar...

Presiona Enter para continuar...
█
```

Figura 2.4: Tercera parte de animación y terminal.

- e) Cuarta parte de animación y terminal, se inserta X. Observar la figura 2.5.

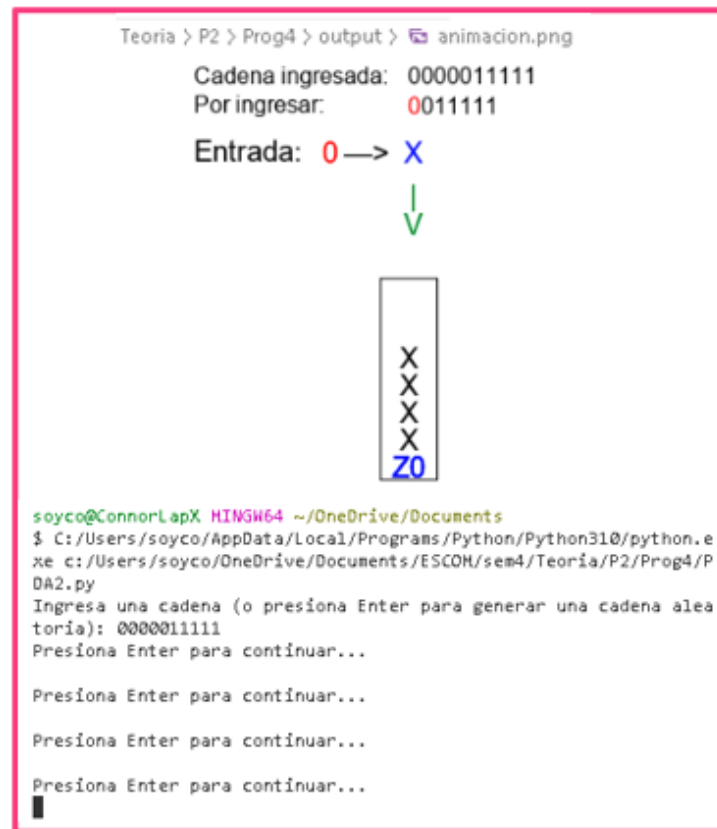


Figura 2.5: Cuarta parte de animación y terminal.

- f) Quinta parte de animación y terminal, se inserta X. Observar la figura 2.6.

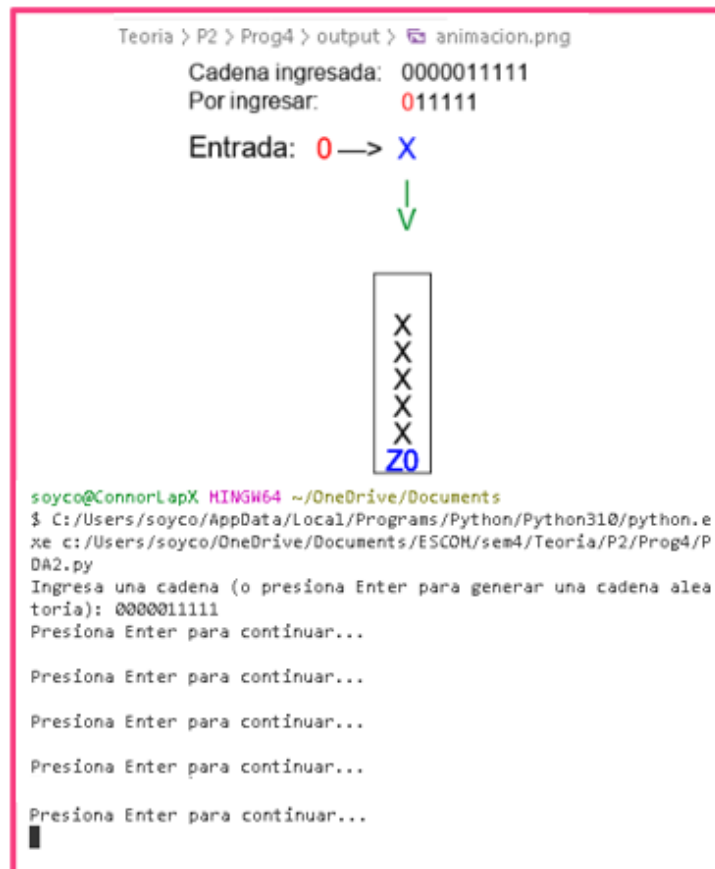
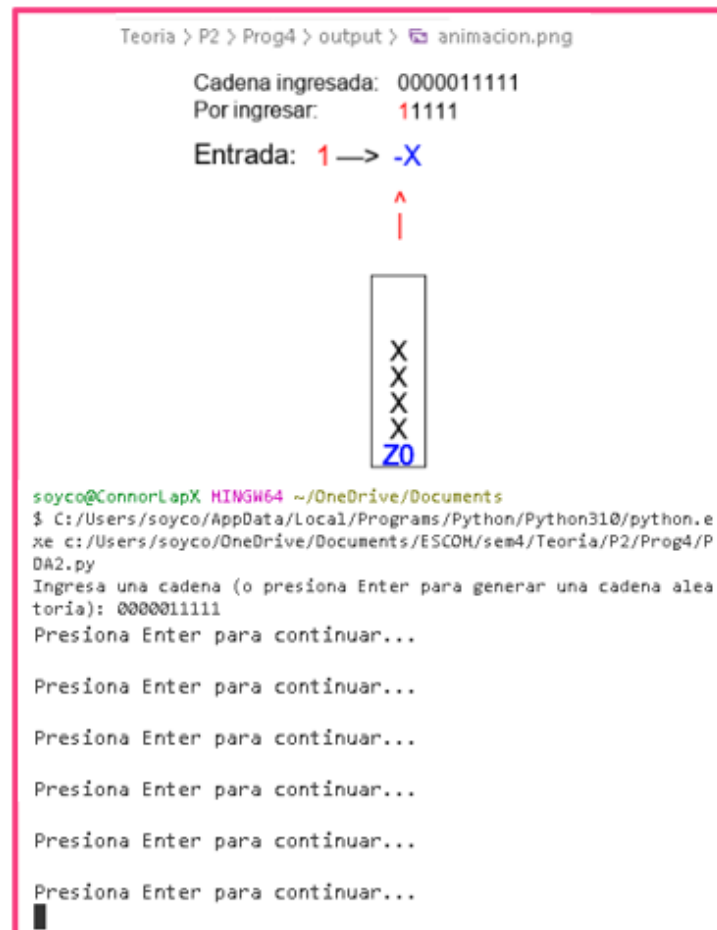


Figura 2.6: Quinta parte de animación y terminal.

- g) Sexta parte de animación y terminal, se elimina X. Observar la figura 2.7.



```
Teoria > P2 > Prog4 > output > animacion.png

Cadena ingresada: 0000011111
Por ingresar: 11111
Entrada: 1 -> -X
      ^
      |
      X
      X
      X
      X
      X
      Z0

soyco@ConnorLapX HINGW64 ~/OneDrive/Documents
$ C:/Users/soyco/AppData/Local/Programs/Python/Python310/python.exe c:/Users/soyco/OneDrive/Documents/ESCON/sem4/Teoria/P2/Prog4/PDA2.py
Ingresar una cadena (o presiona Enter para generar una cadena aleatoria): 0000011111
Presiona Enter para continuar...

Presiona Enter para continuar...

Presiona Enter para continuar...

Presiona Enter para continuar...

Presiona Enter para continuar...

Presiona Enter para continuar...
```

Figura 2.7: Sexta parte de animación y terminal.

- h) Séptima parte de animación y terminal, se elimina X. Observar figura 2.8.

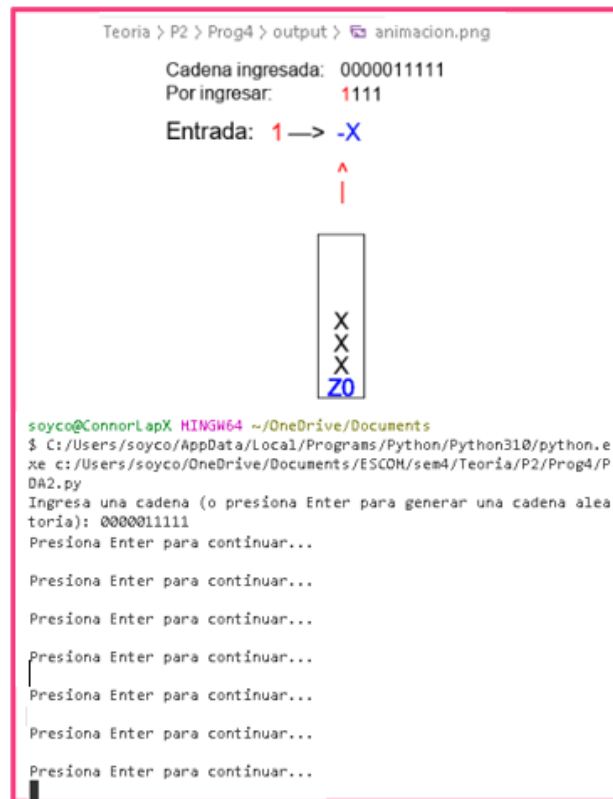


Figura 2.8: Séptima parte de animación y terminal.

- i) Octava parte de animación y terminal, se elimina X. Observar figura 2.9.

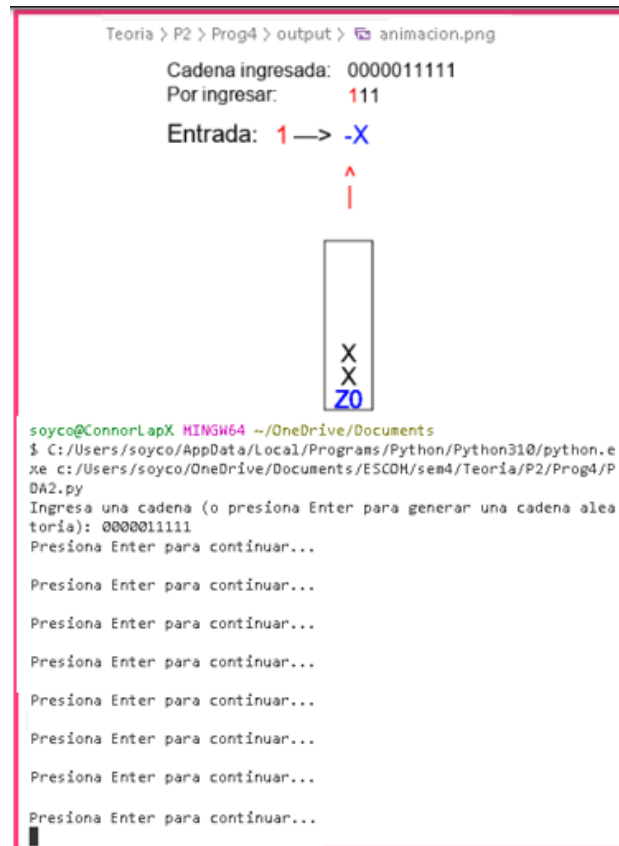


Figura 2.9: Octava parte de animación y terminal.

j) Novena parte de animación y terminal, se elimina X. Observar figura 2.10.

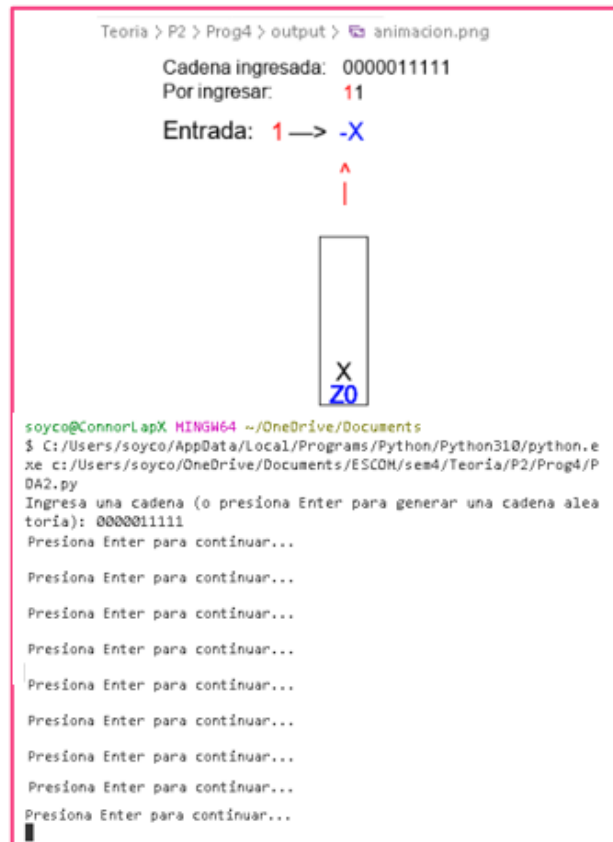


Figura 2.10: Novena parte de animación y terminal.

- k) Décima parte de animación y terminal, se elimina X. Observar la figura 2.11.

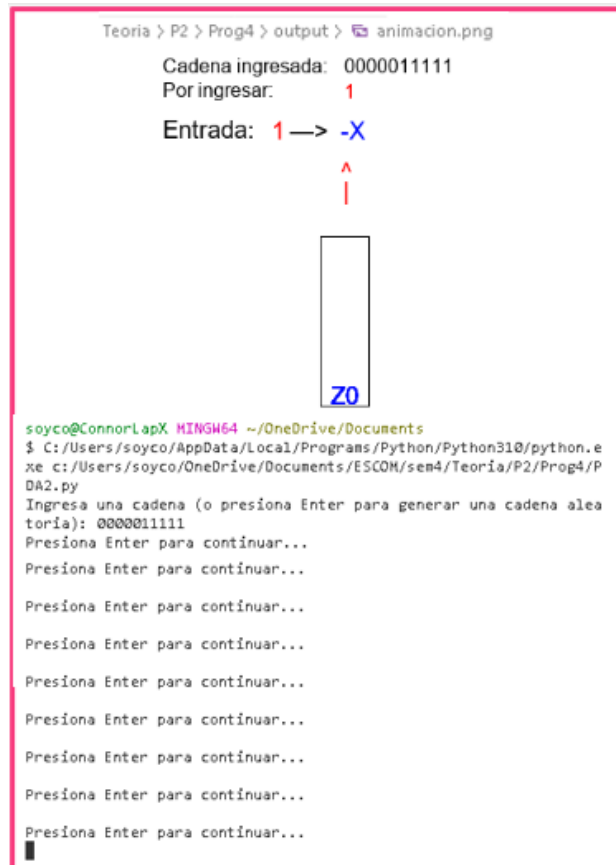


Figura 2.11: Décima parte de animación y terminal.

- l) Onceava parte de animación y terminal, se recibe el vacío y, por lo tanto, se saca lo que haya en la pila, en este caso Z0. Observar la figura 2.12.

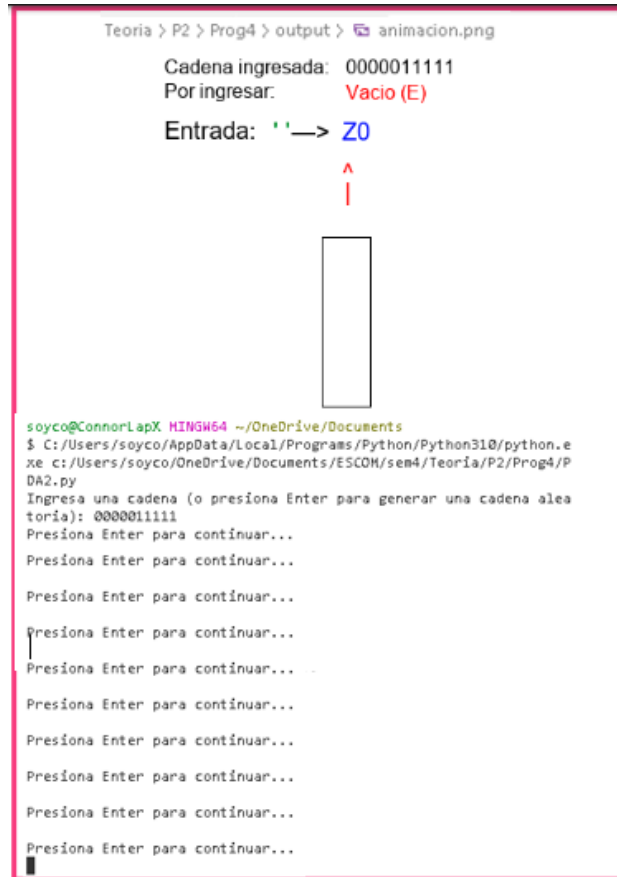


Figura 2.12: Onceava parte de animación y terminal.

- m) Doceava parte de animación y terminal, se válida la cadena. Observar la figura 2.13.



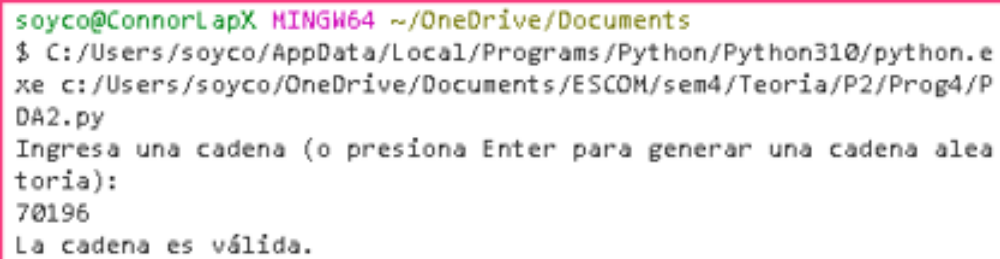
Figura 2.13: Doceava parte de animación y terminal.

- n) Aquí podemos ver el archivo de salida de transiciones.txt, y poder visualizar como se llevó a cabo la construcción de la pila. Observar la figura 2.14.

```
ESCOM > sem4 > Teoria > P2 > Prog4 > output > transiciones.txt
1 (q, 0, Z0) |-(q, 0, Z0X) |-(q, 0, Z0XX) |-(q, 0, Z0XXX) |-(q, 0, Z0XXXX) |-(q, 1, Z0XXXXX)
2 |-(p, 1, Z0XXXX) |-(p, 1, Z0XXX) |-(p, 1, Z0XX) |-(p, 1, Z0X) |-(p, , Z0) |-(f, , Z0)
3
```

Figura 2.14: Archivo de salida de transiciones.txt.

- ñ) En esta aparte pedimos al programa una cadena aleatoria presionando Enter, se digito aleatoriamente una cadena de tamaño 70,196. Observar la figura 2.15.



```
soyco@ConnorLapX MINGW64 ~/OneDrive/Documents
$ C:/Users/soyco/AppData/Local/Programs/Python/Python310/python.exe c:/Users/soyco/OneDrive/Documents/ESCOM/sem4/Teoría/P2/Prog4/PDA2.py
Ingresa una cadena (o presiona Enter para generar una cadena aleatoria):
70196
La cadena es válida.
```

Figura 2.15: Nueva entrada aleatoria.

- o) Aquí podemos ver el inicio del archivo de salida de transiciones.txt, y poder visualizar como se llevó a cabo la construcción de la pila. Observar la figura 2.16.

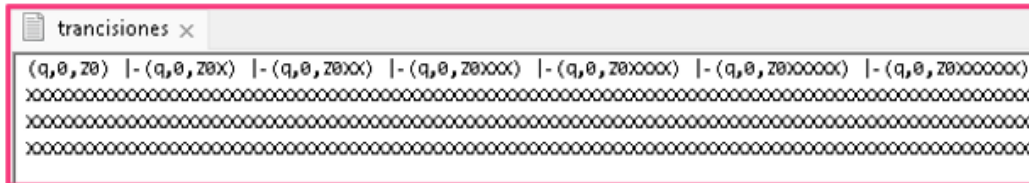


Figura 2.16: Inicio de archivo de transiciones.txt.

p) Aquí podemos ver el final del archivo de salida de transiciones.txt, y poder visualizar como se llevó a cabo la construcción de la pila. Observar la figura 2.17.

```

|-----|
|-----|
|-----|
|- (p1, 20x000000) |- (p1, 20x000000) |- (p1, 20x000000) |- (p1, 20x000000) |- (p1, 20x0000) |- (p1, 20x0000) |- (p1, 20x00) |- (fs, 28) |

```

Figura 2.17: Final de archivo de transiciones.txt .

Capítulo 3

Conclusión

Durante el desarrollo de este programa de autómata de pila, he logrado comprender en profundidad los conceptos relacionados con los lenguajes contextuales y su análisis. La implementación de este programa me ha permitido explorar la estructura y validación de lenguajes formales, así como comprender la importancia de las estructuras de datos en el análisis sintáctico.

A través de esta experiencia, he aprendido cómo utilizar los autómatas de pila como herramienta fundamental en la teoría de la computación. Estos autómatas tienen una amplia gama de aplicaciones prácticas, como la compilación de lenguajes de programación, el análisis de lenguajes naturales y la verificación de la corrección sintáctica en diversos contextos.

3.1. Problemas iniciales

Durante el desarrollo del problema del autómata de pila, se presentaron algunos desafíos iniciales que requirieron atención y resolución. Estos problemas iniciales incluyeron:

- a) Comprender el concepto de autómeta de pila: Al inicio, fue necesario comprender en detalle qué es un autómeta de pila y cómo funciona. Esto implicó estudiar su estructura, transiciones y comportamiento en la resolución de problemas de lenguajes formales.
- b) Diseñar la estructura de datos adecuada: Para implementar el autómeta de pila, fue necesario seleccionar una estructura de datos adecuada que permitiera representar y manipular la pila. Esto implicó evaluar diferentes opciones y elegir la más eficiente y conveniente para el problema en cuestión.
- c) Definir las reglas de transición: El siguiente desafío fue establecer las reglas de transición del autómeta de pila para cada símbolo de entrada. Esto requería comprender las reglas de la gramática y definir correctamente las transiciones en función de los símbolos de entrada y el estado actual del autómeta.
- d) Manejo de casos de error y ambigüedades: Durante la implementación, surgió la necesidad de manejar casos de error y ambigüedades en la entrada. Esto implicó considerar situaciones como símbolos no válidos, cadenas mal formadas o transiciones no definidas, y diseñar mecanismos para manejar estas situaciones de manera adecuada.

Enfrentar estos problemas iniciales y abordarlos de manera adecuada fue fundamental para el desarrollo exitoso del problema del autómeta de pila. Cada uno de estos desafíos brindó oportunidades para aprender y fortalecer los conocimientos sobre teoría de la computación y lenguajes formales.

3.1.1. Soluciones

Se llevó a cabo un estudio detallado del autómeta de pila para comprender su funcionamiento y aplicaciones relevantes. Posteriormente, se eligió una estructura de datos adecuada para representar la pila, garantizando eficiencia y facilidad de manipulación.

Se definieron reglas de transición precisas para el autómata, considerando todos los estados, símbolos de entrada y las acciones correspondientes. Además, se implementaron mecanismos para manejar errores y ambigüedades en la entrada, asegurando la validez de los símbolos y la corrección en el procesamiento de las cadenas.

Estas soluciones permitieron superar los desafíos iniciales y lograr una implementación exitosa del autómata de pila en el código proporcionado.

3.2. Complejidades

Supongamos que la longitud de la cadena de entrada es n . En el peor de los casos, el bucle `while` se ejecutará $n + 1$ veces, ya que también se ejecuta una vez más después de procesar todos los caracteres de la cadena.

Dentro del bucle, se realizan operaciones como comprobaciones de pertenencia en un diccionario (`((estado_actual, symbol, cima_de_pila) in PDA)`), acceso a elementos de diccionario (`transiciones = PDA[(estado_actual, symbol, cima_de_pila)]`), operaciones de pila (`pila.pop()`, `pila.append(letra_apuntada)`), concatenación de cadenas (`"".join(pila)`), y asignaciones de variables. Estas operaciones tienen una complejidad constante, es decir, no dependen del tamaño de la cadena de entrada.

Por lo tanto, la complejidad total de la función `proceso_recorrido` sin graficar es lineal, $O(n)$, donde n es la longitud de la cadena de entrada.

Capítulo 4

Bibliografías

- a)* Hopcroft, J. E., Motwani, R., Ullman, J. D. (2006). Introduction to Automata Theory, Languages, and Computation (3rd ed.). Addison-Wesley.
- b)* Sipser, M. (2012). Introduction to the Theory of Computation (3rd ed.). Cengage Learning.
- c)* Papadimitriou, C. H. (1997). Elements of the Theory of Computation. Prentice-Hall.
- d)* Linz, P. (2006). An Introduction to Formal Languages and Automata (4th ed.). Jones Bartlett Learning.

Capítulo 5

Anexos

5.1. PDA.py

Se presenta el código implementado para la solución al problema con extensión .py .

```
1 #Teoria de la computacion
2 #Automata de pila
3 #Alumno: Connor Urbano Mendoza
4
5 import random
6 from PIL import Image, ImageDraw, ImageFont
7 import sys
8
9 # Tama o de la imagen y tama o del cuadrado
10 image_size = (400, 400)
11 square_size = 100
12
13 # Crear una imagen en blanco
14 image = Image.new("RGB", image_size, "white")
15 draw = ImageDraw.Draw(image)
16
17
18
```

```

19 # Definir el PDA con sus transiciones
20 PDA = {
21     ('q', '0', 'Z0'): [('q', 'X')],
22     ('q', '0', 'X'): [('q', 'X')],
23     ('q', '1', 'X'): [('p', '')],
24     ('p', '1', 'X'): [('p', 'Z0')],
25     ('p', '', 'Z0'): [('f', 'Z0')]
26 }
27
28 #Funcion de recorrido sin graficar
29 def proceso_recorrido(cadena, estado, i):
30     ruta_archivo = "C:\\Users\\soyco\\OneDrive\\
31         Documents\\ESCOM\\sem4\\Teoria\\P2\\Prog4\\
32         output\\trancisiones.txt"
33     with open(ruta_archivo, "w") as archivo:
34         estado_actual=estado
35
36         while i < (len(cadena)+1):
37             if i == (len(cadena)):
38                 symbol=''
39             else:
40                 symbol = cadena[i]
41             cadena_pila = ''.join(pila)
42
43             cima_de_pila = pila[-1]
44             archivo.write("(" + estado_actual + ", " + symbol
45                 + ", " + cadena_pila + ") |-")
46             if (estado_actual, symbol, cima_de_pila)
47                 in PDA:
48                 transiciones = PDA[(estado_actual,
49                     symbol, cima_de_pila)]
50                 estado_siguiente = transiciones[0][0]
51                 letra_apuntada = transiciones[0][1]
52                 if(letra_apuntada == ''):
53                     cima_de_pila = pila.pop()
54                 elif(letra_apuntada == 'Z0'):
55                     cima_de_pila = pila.pop()
56                 else:
57                     pila.append(letra_apuntada)

```

```

54         estado_actual=estado_siguiente
55         i += 1
56     else:
57         archivo.write("Error")
58         return False
59     archivo.write("(" + estado_actual + ", " + symbol + ", "
60         + cadena_pila + ")")
61     return True
62 #Funcion de recorrido para graficar
63 def proceso_recorrido2(cadena, estado, i):
64     ruta_archivo = "C:\\Users\\soyco\\OneDrive\\
65         Documents\\ESCOM\\sem4\\Teoria\\P2\\Prog4\\
66         output\\trancisiones.txt"
67     with open(ruta_archivo, "w") as archivo:
68         estado_actual=estado
69         cadena_grafica = cadena
70
71         while i < (len(cadena)+1):
72
73             # Coordenadas del rect ngulo
74             rect_width = 40
75             rect_height = 140
76             x1 = ((image_size[0] - rect_width) // 2)
77             desplazamientoa_abajo=60
78             y1 = ((image_size[1] - rect_height) // 2)
79             +desplazamientoa_abajo
80             x2 = x1 + rect_width
81             y2 = y1 + rect_height
82
83             # Dibujar el rect ngulo en la imagen
84             draw.rectangle([(x1, y1), (x2, y2)],
85                 outline="black")
86             #-----Dibujamos apartados-----
87             # Dibujamos apartados de cadena ingresada
88             text = "Cadena ingresada:"
89             font = ImageFont.truetype("arial.ttf", 16)
90
91             # Dibujar el texto en la imagen

```

```
88         draw.text((50, 40), text, fill="black",
89                   font=font)
90
91         # Dibujar ecadenal texto en la imagen
92         draw.text((200, 40), cadena, fill="black",
93                   font=font)
94
95         # Dibujamos apartados de por ingresar
96         text = "Por ingresar:"
97
98         # Dibujar el texto en la imagen
99         draw.text((50, 60), text, fill="black",
100                  font=font)
101
102         #Dibujamos primer caracter
103         j=0
104
105         cadena_grafica = cadena_grafica[1:] #
106         # Obtener la subcadena sin la primera
107         # letra
108         draw.text((209, 61), cadena_grafica, fill=
109                 "black", font=font)
110
111         # Dibujamos apartados de entrada
112         text = "Entrada:"
113         font2 = ImageFont.truetype("arial.ttf",
114                                     20)
115
116         # Dibujar el texto en la imagen
117         draw.text((50, 90), text, fill="black",
118                   font=font2)
119
120         #Coordenada de incremento para cruces
121         cross_y = (((y1 + y2) // 2)+50) - ((i+1)*
122                                     19)
123
124         #Dibujamos caso base
125         draw.text((189, (((y1 + y2) // 2)+50)), "
126                 Z0", fill="blue", font=font2)
```

```

118         if i == (len(cadena)):
119             symbol=""
120             #Dibujamos primer caracter y la
                entrada
121             # Dibujar el texto en la imagen
122             draw.text((140, 91), "' '", fill="
                green", font=font2)
123             draw.text((200, 61), "Vacio (E)", fill
                ="red", font=font)
124             #Dibujamos felcha de conversion
125             draw.text((155, 91), " >", fill="
                black", font=font2)
126         else:
127             symbol = cadena[i]
128             number=int(symbol)
129             # Dibujar el texto en la imagen
130             draw.text((140, 91), symbol, fill="red
                ", font=font2)
131             draw.text((200, 61), symbol, fill="red
                ", font=font)
132             #Dibujamos felcha de conversion
133             draw.text((155, 91), " >", fill="
                black", font=font2)
134
135         cadena_pila = ''.join(pila)
136         cima_de_pila = pila[-1]
137
138         archivo.write("(" + estado_actual + "," + symbol
                + "," + cadena_pila + ") |-")
139         if (estado_actual, symbol, cima_de_pila)
                in PDA:
140             transiciones = PDA[(estado_actual,
                symbol, cima_de_pila)]
141             estado_siguiete = transiciones[0][0]
142             letra_apuntada = transiciones[0][1]
143             if(letra_apuntada == ''):
144                 cima_de_pila = pila.pop()
145                 #Dibujamos letra
146                 draw.text((197, 91), "-X", fill="
                blue", font=font2)

```

```

147         draw.text((197, 125), "^", fill="
           red", font=font2)
148         draw.text((199, 141), "|", fill="
           red", font=font2)
149         if len(cadena) % 2 != 0:
150             cross_y = (((y1 + y2) // 2) +
                          50) - ((i+1)*19)-19)
151         else:
152             cross_y = (((y1 + y2) // 2) +
                          50) - ((i+1)*19)
153         # Coordenadas de la ltima cruz
           agregada
154         last_cross_x = 200
155
156         last_cross_y = cross_y + ((38*i)-
           (19*len(cadena)))
157         last_cross_size = 12
158
159         # Guardar las coordenadas y el
           tama o de la ltima cruz
160         last_cross_coords = (((
           last_cross_x - last_cross_size)
           -1, last_cross_y + 21), ((
           last_cross_x + last_cross_size)
           +1, (last_cross_y + 37)+1])
161         # Eliminar la ltima cruz
162         draw.rectangle(last_cross_coords,
           outline="white", fill="white")
163
164     elif(letra_apuntada == 'Z0' and number
           ==1 and symbol!=''):
165         cima_de_pila = pila.pop()
166         #Dibujamos letra
167         draw.text((197, 91), "-X", fill="
           blue", font=font2)
168         draw.text((197, 125), "^", fill="
           red", font=font2)
169         draw.text((199, 141), "|", fill="
           red", font=font2)
170         if len(cadena) % 2 != 0:

```

```

171         cross_y = (((y1 + y2) // 2) +
172                    50) - ((i+1)*19)-19)
173     else:
174         cross_y = (((y1 + y2) // 2) +
175                    50) - ((i+1)*19)
176     # Coordenadas de la ltima cruz
177     agregada
178     last_cross_x = 200
179     #print(cross_y)
180     last_cross_y = cross_y + ((38*i)-
181                             (19*len(cadena)))
182     last_cross_size = 12
183
184     # Guardar las coordenadas y el
185     tama o de la ltima cruz
186     last_cross_coords = [((
187         last_cross_x - last_cross_size)
188         -1, last_cross_y + 21), ((
189         last_cross_x + last_cross_size)
190         +1, (last_cross_y + 37)+1)]
191     # Eliminar la ltima cruz
192     draw.rectangle(last_cross_coords,
193                   outline="white", fill="white")
194 elif(letra_apuntada == 'Z0' and symbol
195      =='' ):
196     cima_de_pila = pila.pop()
197     #Dibujamos letra
198     draw.text((197, 91), "Z0", fill="
199     blue", font=font2)
200     draw.text((197, 125), "^", fill="
201     red", font=font2)
202     draw.text((199, 141), "|", fill="
203     red", font=font2)
204     # Coordenadas de la ltima cruz
205     agregada
206     last_cross_x = 200
207     #print(cross_y)
208     last_cross_y = cross_y + ((38*i)-
209                             (19*len(cadena)))
210     last_cross_size = 12

```

```
195
196         # Guardar las coordenadas y el
           tama o de la ltima cruz
197     last_cross_coords = [(
        last_cross_x - last_cross_size)
        -1, last_cross_y + 21), ((
        last_cross_x + last_cross_size)
        +1, (last_cross_y + 37)+1)]
198     # Eliminar la ltima cruz
199     draw.rectangle(last_cross_coords,
        outline="white", fill="white")
200 else:
201     pila.append(letra_apuntada)
202     #Dibujamos letra
203     draw.text((197, 91),
        letra_apuntada, fill="blue",
        font=font2)
204     draw.text((201, 121), "|", fill="
        green", font=font2)
205     draw.text((197, 141), "v", fill="
        green", font=font2)
206     #Dibujamos cruz
207     draw.text((194, cross_y), "X",
        fill="black", font=font2)
208
209     estado_actual=estado_siguiete
210
211     # Guardar la imagen en la ruta
        especificada
212     image.save("C:\\Users\\soyco\\OneDrive
        \\Documents\\ESCOM\\sem4\\Teoria\\
        P2\\Prog4\\output\\animacion.png")
213     if sys.stdin.isatty():
214         print("Presiona Enter para
            continuar...")
215         sys.stdin.readline()
216     else:
217         input("Presiona Enter para
            continuar...")
218     i += 1
```



```

219         else:
220             archivo.write("Error")
221             #Dibujamos letra
222             # Cadena invalida
223             draw.text((200, 40), cadena, fill="red",
224                       font=font)
225             draw.text((197, 91), "ERROR", fill="blue",
226                       font=font2)
227
228             image.save("C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM\\sem4\\Teoria\\P2\\Prog4\\output\\animacion.png")
229             if sys.stdin.isatty():
230                 print("Presiona Enter para continuar...")
231                 sys.stdin.readline()
232             else:
233                 input("Presiona Enter para continuar...")
234             return False
235
236         # Aqu se realiza el borrado del contenido de la imagen
237         draw.rectangle([(11, 11), (389, 180)], fill="white")
238         image.save("C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM\\sem4\\Teoria\\P2\\Prog4\\output\\animacion.png")
239
240         archivo.write("(" + estado_actual + ", " + symbol + ", " + cadena_pila + ")")
241
242         # Coordenadas del rect ngulo
243         rect_width = 40
244         rect_height = 140
245         x1 = ((image_size[0] - rect_width) // 2)
246         desplazamientoa_abajo=60
247         y1 = ((image_size[1] - rect_height) // 2) + desplazamientoa_abajo

```

```
247     x2 = x1 + rect_width
248     y2 = y1 + rect_height
249
250     # Dibujar el rectngulo en la imagen
251     draw.rectangle([(x1, y1), (x2, y2)], outline="
        black")
252
253     # Dibujamos apartados de cadena ingresada
254     text = "Cadena ingresada:"
255     font = ImageFont.truetype("arial.ttf", 16)
256
257     # Dibujar el texto en la imagen
258     draw.text((50, 40), text, fill="black", font=
        font)
259
260     # Dibujar ecadenal texto en la imagen
261     draw.text((200, 40), cadena, fill="green",
        font=font)
262
263     text = "Estado Final"
264
265     # Dibujar el texto en la imagen
266     draw.text((50, 60), text, fill="black", font=
        font)
267
268     # Dibujamos apartados de entrada
269     text = "Ya sacamos a Z0"
270     # Dibujar el texto en la imagen
271     draw.text((50, 92), text, fill="black", font=
        font)
272     #Dibujamos felcha de conversion
273     draw.text((178, 91), "      >", fill="black",
        font=font2)
274
275
276     draw.text((235, 91), "Z0", fill="blue", font=
        font2)
277
278     image.save("C:\\Users\\soyco\\OneDrive\\
        Documents\\ESCOM\\sem4\\Teoria\\P2\\Prog4\\
```

```
        output\\animacion.png")
279     if sys.stdin.isatty():
280         sys.stdin.readline()
281     else:
282         input("Presiona Enter para continuar...")
283     return True
284
285 def generar_cadena_aleatoria(tamano):
286     mitad = tamano // 2
287     ceros = '0' * mitad
288     unos = '1' * mitad
289     if tamano % 2 != 0:
290         ceros += random.choice(['0', '1'])
291     cadena = ceros + unos
292     return cadena
293
294
295 #Main
296 pila = []
297
298 elemento_superior = 'Z0'
299 pila.append(elemento_superior)
300 estado = 'q'
301 i = 0
302
303 cadena = input("Ingresa una cadena (o presiona Enter
                 para generar una cadena aleatoria): ")
304
305 if not cadena:
306     tamano = random.randint(1, 100000)
307     print(tamano)
308     cadena = generar_cadena_aleatoria(tamano)
309     resultado = proceso_recorrido(cadena, estado, i)
310
311     if resultado:
312         print("La cadena es v lida.")
313     else:
314         print("La cadena no es v lida.")
315
316 else:
```

```
317     resultado = proceso_recorrido2(cadena, estado, i)
318
319     if resultado:
320         print("La cadena es v lida.")
321     else:
322         print("La cadena no es v lida.")
```

Se presenta el código LaTeX de este archivo mediante el siguiente link:

Link overleaf: "<https://www.overleaf.com/read/tpnhwgwdrprq>"

Link github: "<https://github.com/Connor-UM-18/Teoria-computacional-PDA.git>"