Universidad Rafael Landivar

Facultad de Ingenieria

Lenguajes formales y autónomas

Sección: 1

Ing. Julio David Requena Duarte

Proyecto 03

Angel Samuel Pérez Cruz

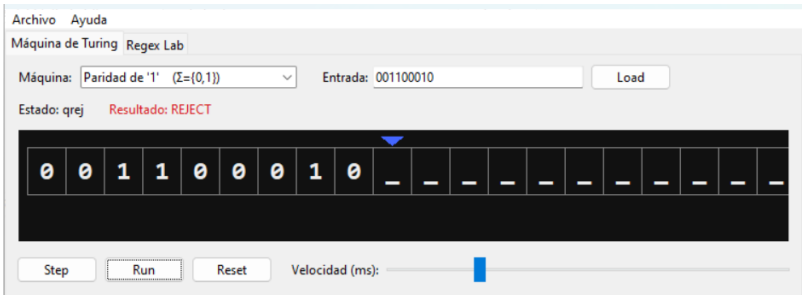
Carné: 1135323

Guatemala, 10 de noviembre del 2025

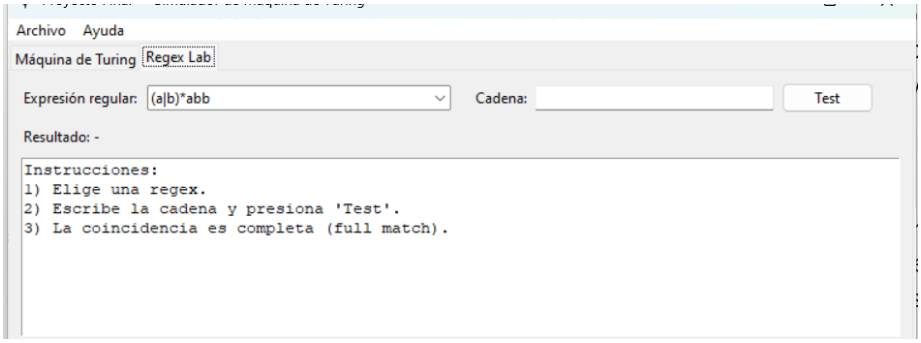
Descripción del programa

Este programa fue desarrollado como parte de un proyecto académico de Ingeniería en Sistemas. Su objetivo principal es simular el comportamiento de una Máquina de Turing y permitir la experimentación con expresiones regulares dentro de una interfaz visual sencilla creada con Python y Tkinter. La idea es que los estudiantes podamos visualizar paso a paso cómo funciona una máquina de Turing, cómo procesa una cadena de entrada, y cómo determina si la acepta o la rechaza, según las reglas de transición definidas.

El programa tiene dos secciones principales: la pestaña **“Máquina de Turing”** y la pestaña **“Regex Lab”.**
En la primera, se puede seleccionar una de las máquinas ya implementadas, escribir una cadena de entrada y observar cómo la cinta se va modificando mientras el cabezal avanza y cambia de estado. Esto ayuda a entender la lógica de los autómatas y cómo se resuelven problemas básicos como verificar si una cadena termina en cierto patrón, si tiene un número par de unos o si cumple con la forma a^n b^n.



En la segunda pestaña, el laboratorio de expresiones regulares, se puede elegir entre diez expresiones predefinidas y probar diferentes cadenas para comprobar si cumplen o no con el patrón. Esto sirve para comprender mejor el uso de los operadores y cuantificadores de las expresiones regulares.



El proyecto es completamente funcional y no necesita instalar librerías externas, ya que Tkinter viene incluido con Python. Además, puede ejecutarse tanto en Windows, macOS o Linux, lo que lo hace accesible para cualquier estudiante. Su propósito es facilitar la comprensión de temas de teoría de autómatas, lenguajes formales y expresiones regulares, permitiendo una visualización práctica y didáctica de conceptos que normalmente se estudian solo de forma teórica.

Instrucciones de instalación y ejecución

Para poder utilizar el programa, es necesario tener instalado Python 3.8 o una versión superior.
El proyecto está hecho únicamente con librerías estándar, por lo que no requiere instalaciones adicionales, excepto en sistemas Linux, donde puede ser necesario instalar Tkinter, que es la librería usada para la interfaz gráfica.

1. Instalación de dependencias

Si estás en Windows o macOS, normalmente no necesitas hacer nada más, ya que Tkinter viene incluido con Python.
En caso de usar Linux, puedes instalar Tkinter ejecutando los siguientes comandos en la terminal:

sudo apt-get update

sudo apt-get install -y python3-tk

2. Descargar el proyecto

Tienes dos opciones para obtener el código:

- **Opción 1: Descargar desde GitHub**
    1. Entra al repositorio del proyecto en GitHub.
    2. Presiona el botón verde que dice Code.
    3. Selecciona Download ZIP.
    4. Descomprime el archivo en una carpeta de tu preferencia (por ejemplo, en el Escritorio).
    5. Asegúrate de que dentro esté el archivo Proyectofinal.py.
- **Opción 2: Clonar con Git (opcional, si tienes Git instalado)**
- git clone https://github.com/angel-perez213/Proyecto3-lenguajee.git
- cd angel-perez213/Proyecto3-lenguajee.git

3. Ejecución del programa

Abre una terminal en la carpeta donde se encuentra el archivo Proyectofinal.py y ejecuta uno de los siguientes comandos según tu sistema operativo:

- En Windows:
- python Proyectofinal.py
- En macOS o Linux:
- python3 Proyectofinal.py

Si al escribir python ya se ejecuta la versión 3 en tu sistema, puedes usar indistintamente:

python Proyectofinal.py

4. Uso básico del programa

Una vez abierto, aparecerá una ventana con dos pestañas principales:

- Máquina de Turing: permite cargar una cadena, simular paso a paso o en modo automático, y observar el resultado (ACCEPT o REJECT).
- Regex Lab: permite seleccionar una expresión regular, escribir una cadena y comprobar si coincide con el patrón (MATCH o NO MATCH).

5. Solución de problemas comunes

- Si la ventana no se abre en Linux, asegúrate de haber instalado Tkinter con los comandos mencionados.
- Si en Windows el comando python no funciona, reinstala Python desde python.org y marca la opción “Add Python to PATH” durante la instalación.
- Si el programa no inicia o muestra errores, revisa que estés dentro de la carpeta correcta y que el archivo se llame exactamente Proyectofinal.py.

Ejemplo de ejecución con capturas de pantalla

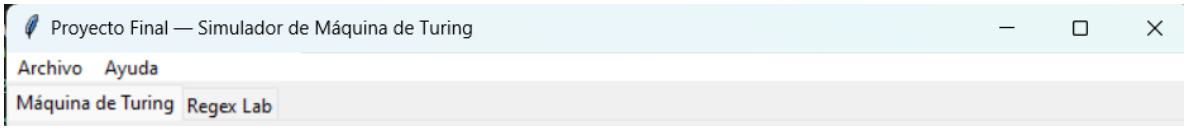
A continuación se describe cómo funciona el programa durante su ejecución y algunos ejemplos prácticos del uso de las dos secciones principales: la Máquina de Turing y el Regex Lab.

1. Pantalla inicial del programa

Al ejecutar el archivo Proyectofinal.py, aparece una ventana principal con dos pestañas:

- Máquina de Turing
- Regex Lab

En la parte superior se puede seleccionar la máquina deseada, escribir una cadena de entrada y controlar la simulación con los botones Load, Step, Run/Pause y Reset.

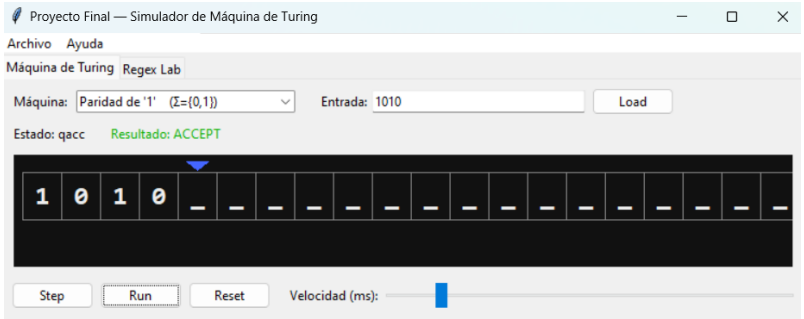


2. Ejemplo 1: Máquina de Turing – Paridad de 1 (Σ={0,1})

En esta máquina se evalúa si la cadena contiene una cantidad par de unos.

- Entrada: 1010
- Proceso: el cabezal recorre la cinta leyendo y moviéndose a la derecha.
- Resultado final: ACCEPT (porque hay dos '1').

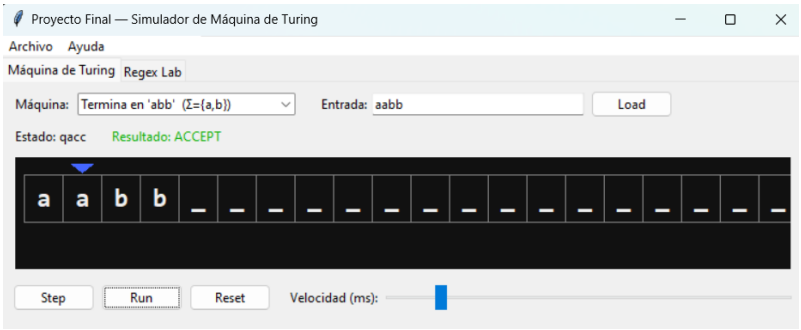
En la interfaz, la palabra "ACCEPT" aparece en color verde, indicando que la cadena fue aceptada.



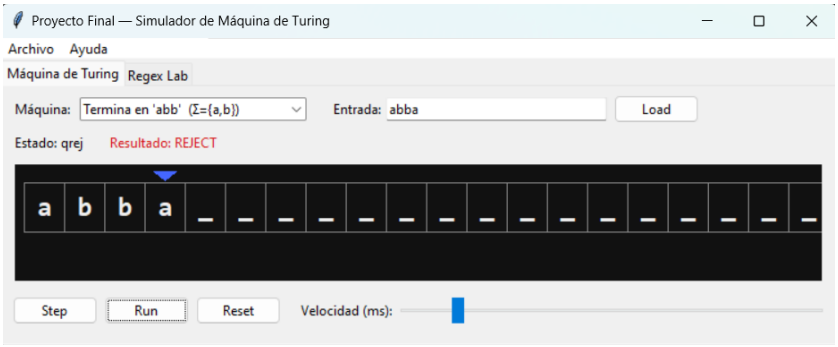
3. Ejemplo 2: Máquina de Turing – Termina en ‘abb’ (Σ={a,b})

Esta máquina verifica si la cadena termina exactamente con el patrón abb.

- Entrada: aabb
- Proceso: el cabezal se mueve hasta el final de la cinta y luego verifica las últimas letras.
- Resultado: ACCEPT (porque la cadena termina en ‘abb’).



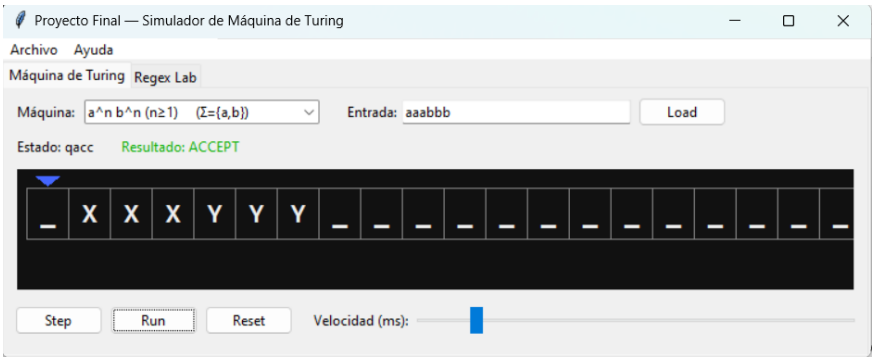
Si se ingresara una cadena como abba, el resultado sería REJECT (rojo), porque no cumple el patrón.



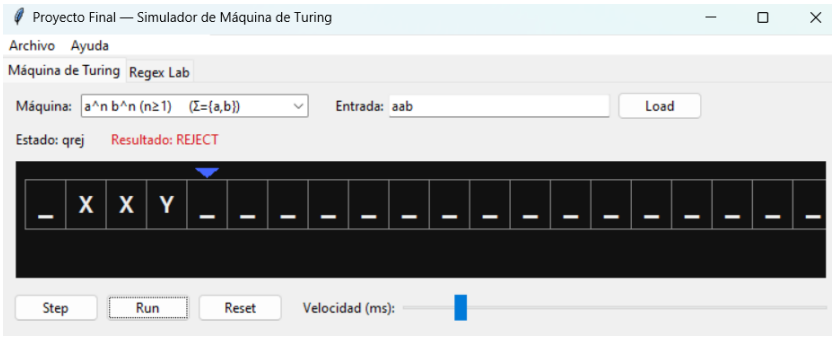
4. Ejemplo 3: Máquina de Turing – a^n b^n (n≥1)

Esta máquina valida que la cadena tenga una cantidad igual de letras ‘a’ seguidas de la misma cantidad de ‘b’.

- Entrada: aaabbb
- Proceso: la máquina marca las ‘a’ y las ‘b’ que empareja y avanza hasta verificar toda la cadena.
- Resultado final: **ACCEPT**, ya que hay tres ‘a’ y tres ‘b’.



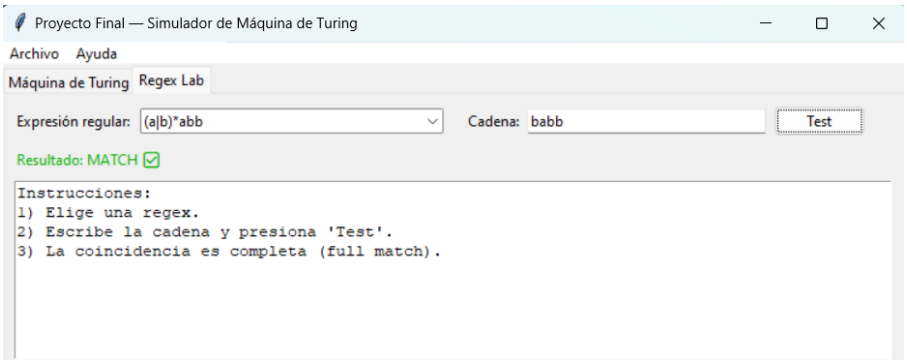
Si se ingresara aab, el resultado sería **REJECT**, ya que no se cumple la correspondencia.



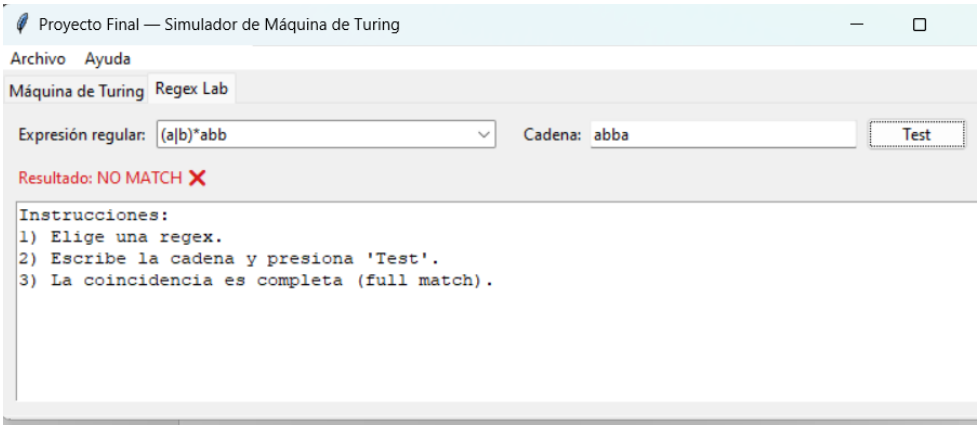
5. Ejemplo 4: Uso del Regex Lab

En la segunda pestaña del programa se encuentra el Regex Lab, un espacio para probar expresiones regulares.

- Expresión seleccionada: (a|b)*abb
- Cadena ingresada: babb
- Resultado: MATCH, porque la cadena termina con ‘abb’.



Si se ingresara la cadena abba, el resultado sería NO MATCH, ya que no cumple el patrón.



6. Visualización de resultados

En todos los casos, el programa muestra:

- El estado actual de la máquina.
- El resultado (ACCEPT o REJECT en la máquina de Turing, MATCH o NO MATCH en Regex Lab).
- La cinta de entrada con el movimiento del cabezal, lo cual ayuda a visualizar cómo la máquina procesa cada símbolo.

Explicación de las expresiones regulares utilizadas

En el laboratorio de expresiones regulares se implementaron diez patrones distintos. Cada uno fue escogido para representar un tipo de estructura o concepto importante dentro del uso de expresiones regulares. A continuación se explica de forma sencilla qué hace cada una y qué tipo de cadenas acepta o rechaza:

1. (a|b)abb
Esta expresión acepta cualquier cadena formada por las letras a y b que termine en “abb”.

- Ejemplo de coincidencia (MATCH): abb, babb, aabb, abababb.
- No coincide (NO MATCH): ab, abba, baab.

2. 0*1*
Permite cero o más 0 seguidos de cero o más 1, en ese orden. La cadena vacía también es válida.

- MATCH: "", 0, 1, 000111, 000, 111.
- NO MATCH: 10, 101, 1100.

3. (ab)*
Acepta repeticiones exactas del bloque “ab”. Puede estar vacío o repetirse varias veces.

- MATCH: "", ab, abab, abababb.
- NO MATCH: a, b, aba, abb.

4. 1(01)0
La cadena debe comenzar con 1, luego puede tener cero o más repeticiones del bloque “01”, y debe terminar con 0.

- MATCH: 10, 1010, 101010.
- NO MATCH: 1, 0, 100, 101, 1010101.

5. (a|b)*a(a|b)*
Acepta cualquier cadena de a y b que contenga al menos una “a”.

- MATCH: a, ba, ab, bab, aaab.
- NO MATCH: "", b, bbb.

6. a+
Representa una o más letras a seguidas.

- MATCH: a, aa, aaaa.
- NO MATCH: "", b, ab, ba.

7. (01|10)+
Acepta una o más repeticiones del bloque “01” o del bloque “10”.

- MATCH: 01, 10, 0110, 1001, 011001.
- NO MATCH: 0, 1, 0011, 010, 110.

8. (a|b){3,5}
Coincide con cadenas formadas solo por a y b con una longitud entre 3 y 5 caracteres.

- MATCH: aaa, abb, baba, ababa.
- NO MATCH: aa (2 caracteres), abababa (7 caracteres), abc (carácter inválido).

9. 0(0|1)0
Acepta cadenas que empiezan y terminan con 0, y pueden tener ceros o unos entre medio.

- MATCH: 00, 010, 0010, 011110.
- NO MATCH: 0, 1, 10, 01111.

10. (?!.*11)10*
Usa un lookahead negativo para asegurarse de que la cadena no contenga “11” en ningún lugar. Después, permite cero o más 1 seguidos de cero o más 0.

- MATCH: "", 0, 1, 10, 1000.
- NO MATCH: 11, 110, 1011, 1110.

Resumen general

Estas expresiones fueron elegidas porque cubren los conceptos básicos del uso de expresiones regulares, como:

- Agrupación y alternancia: (a|b), (01|10)
- Repetición: *, +, {3,5}
- Inicio y fin de patrones: 0(0|1)*0
- Restricciones mediante lookahead: (?!.*11)

De esta forma, el Regex Lab permite practicar y visualizar cómo los distintos símbolos y operadores de las expresiones regulares afectan el resultado final, ayudando a comprender su funcionamiento de manera más intuitiva y aplicada.

Conclusión

Este proyecto permitió aplicar de forma práctica los conceptos teóricos vistos en el curso sobre Máquinas de Turing y expresiones regulares, dos temas fundamentales dentro de la teoría de autómatas y lenguajes formales. A través del desarrollo de un programa en Python con interfaz gráfica (Tkinter), se logró crear una herramienta didáctica que facilita la comprensión visual del funcionamiento interno de una máquina de Turing, mostrando cómo el cabezal lee, escribe y se desplaza sobre la cinta para determinar si una cadena es aceptada o rechazada.

Además, el módulo de Regex Lab permitió reforzar el uso de las expresiones regulares, entendiendo cómo se construyen los patrones, qué significan los operadores y cómo se comportan ante diferentes cadenas de entrada. Esto hace que el proyecto no solo sea una práctica de programación, sino también un recurso educativo que ayuda a visualizar los procesos lógicos detrás de la computación teórica.

En conclusión, este trabajo fortaleció habilidades tanto técnicas como conceptuales: desde la lógica de autómatas hasta la implementación de interfaces gráficas y el manejo de estructuras en Python. Desarrollar este simulador permitió integrar teoría y práctica, demostrando que los principios de los lenguajes formales pueden representarse de manera interactiva, clara y accesible para cualquier estudiante de Ingeniería en Sistemas.

Link del repositorio de GitHub

https://github.com/angel-perez213/Proyecto3-lenguajee