

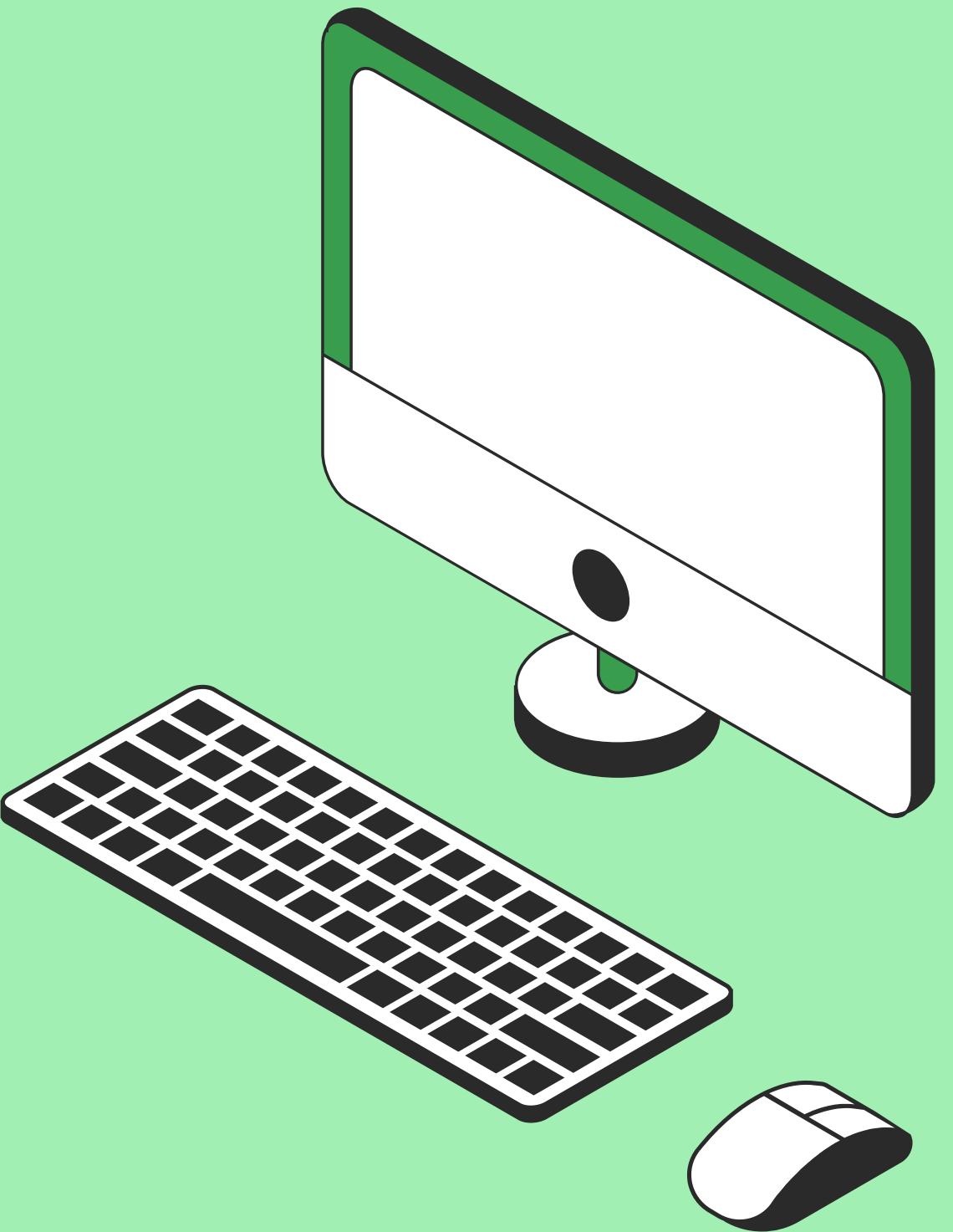
Chomsky Classifier AI

Angel Samuel Perez Cruz 1135323

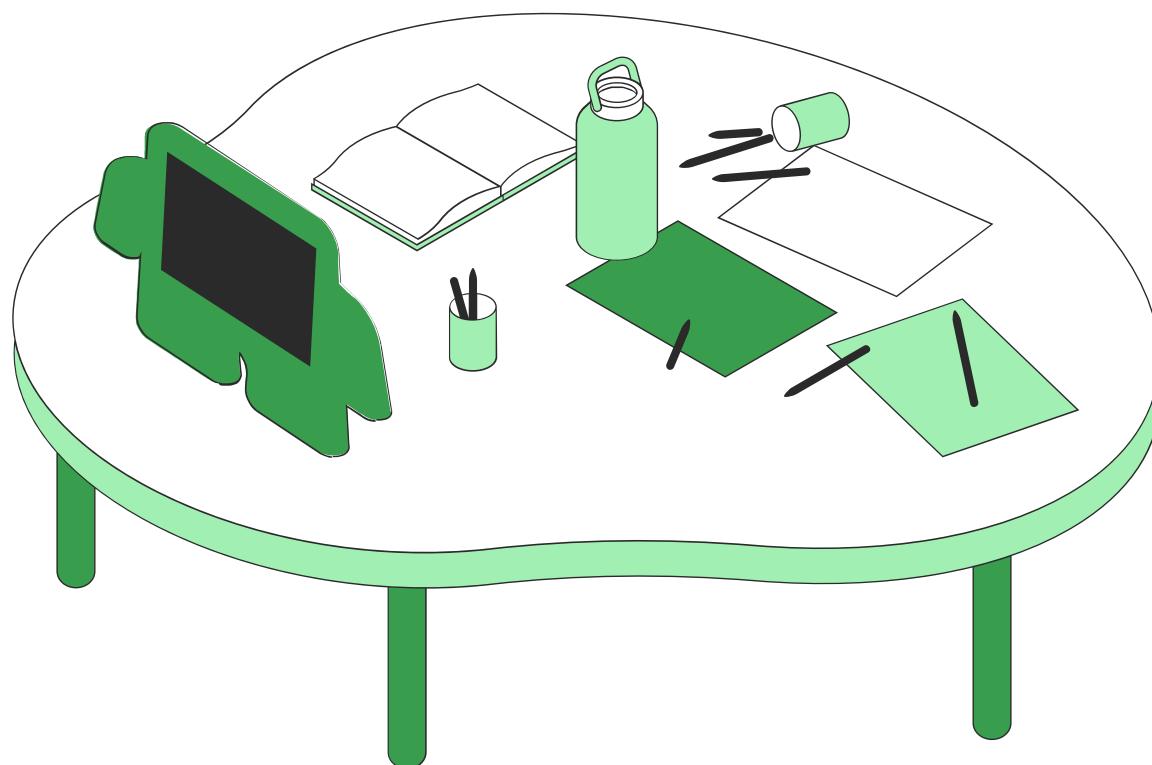


¿Por qué este proyecto?

- En el curso vemos teoría de gramáticas y autómatas, pero a veces cuesta visualizarla.
- La clasificación de una gramática en tipo 0, 1, 2 o 3 normalmente se hace a mano.
- Es fácil cometer errores al revisar las producciones una por una.
- Objetivo general: crear una herramienta que:
- Reciba una gramática o un autómata.
- Lo clasifique dentro de la Jerarquía de Chomsky.
- Explique el razonamiento y genere diagramas y reportes.



Resumen teórico: Jerarquía de Chomsky



- Tipo 3 – Lenguajes regulares
- Gramáticas regulares, autómatas finitos, expresiones regulares.
- Tipo 2 – Lenguajes libres de contexto
- Gramáticas libres de contexto, autómatas con pila.
- Tipo 1 – Lenguajes sensibles al contexto
- Gramáticas con producciones que cumplen $|\alpha| \leq |\beta|$.
- Tipo 0 – Lenguajes recursivamente enumerables
- Gramáticas generales, Máquinas de Turing.
- Inclusión:
- $\text{Tipo 3} \subset \text{Tipo 2} \subset \text{Tipo 1} \subset \text{Tipo 0}$

Objetivos del sistema



- Recibir gramáticas en formato texto y analizarlas.
- Recibir autómatas (AFD, AP, MT) en formato JSON.
- Determinar automáticamente el tipo (0, 1, 2 o 3).
- Mostrar un modo explicativo:
 - Qué se revisó en cada producción.
 - Qué condiciones de la jerarquía se cumplen o se violan.
- Generar:
 - Diagramas de gramáticas y autómatas.
 - Reportes PDF con todo el análisis.
 - Ejercicios tipo quiz para practicar.

Arquitectura del proyecto

- main.py
- Ventana principal con Tkinter y pestañas.
- clasificador.py
- Analiza las producciones y clasifica la gramática por tipo.
- automatas.py
- Carga autómatas desde JSON y los clasifica (AFD, AP, MT).
- visualizador.py
- Genera diagramas con Graphviz (gramáticas y autómatas).
- reportes.py
- Crea reportes en PDF con texto, explicación y diagramas.
- ejemplos.py y tutor.py



Funcionamiento: Clasificación de gramáticas

El usuario escribe la gramática, por ejemplo:

- $S \rightarrow aA$
- $A \rightarrow b \mid aA$

El módulo de clasificación:

- Separa lado izquierdo y lado derecho de cada producción.

Verifica:

- Si el lado izquierdo es un solo no terminal.
- Si las producciones siguen la forma regular ($A \rightarrow aB, A \rightarrow a$).
- Si se cumple o no $|α| \leq |β|$.

El sistema determina el tipo más restrictivo que cumple:

- Tipo 3 si es regular.
- Si no, intenta Tipo 2, luego Tipo 1, y finalmente Tipo 0.

En la interfaz:

- Muestra el tipo detectado.
- Muestra una lista de mensajes explicando la decisión.
- Permite ver el diagrama y generar un PDF.

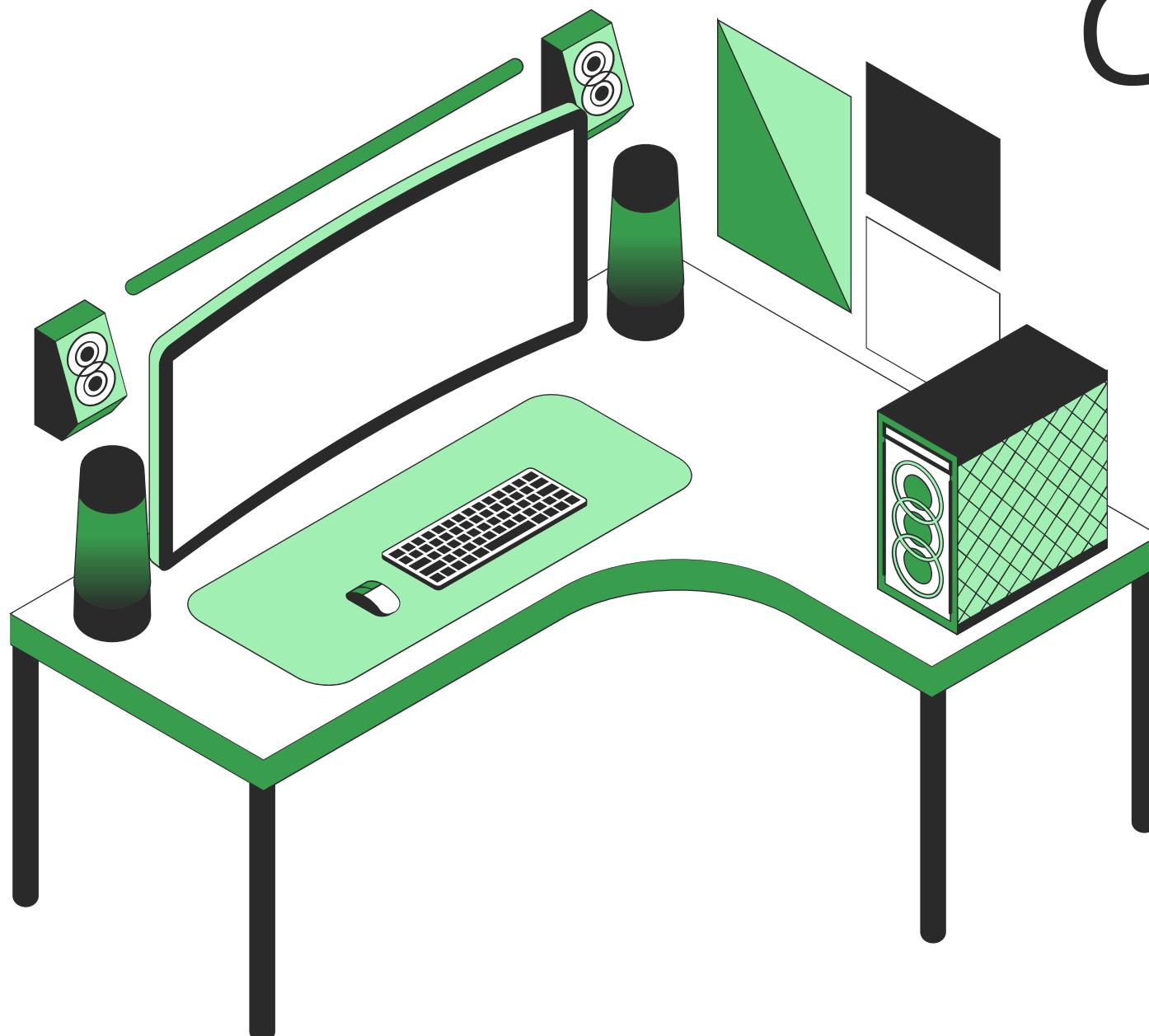


Funcionamiento: Autómatas

- El usuario ingresa un autómata en JSON, por ejemplo un AFD:
- Lista de estados, alfabeto, estado inicial, estados finales, transiciones.
- El módulo de autómatas:
- Lee el campo tipo (AFD, AP, MT).
- Clasifica según el modelo:
 - AFD/AFN → Lenguaje regular (Tipo 3).
 - AP → Lenguaje libre de contexto (Tipo 2).
 - MT → Lenguaje tipo 0.
- El visualizador:
- Dibuja el grafo de estados y transiciones con Graphviz.
- Desde la interfaz:
- Se puede ver el diagrama.
- Se puede generar un PDF de reporte con toda la información.



Módulos extra: Generador, Comparador y Tutor



- Generador de gramáticas:
- Permite generar gramáticas aleatorias por tipo (0, 1, 2, 3).
- Sirven como casos de prueba y para practicar.
- Ejemplos de autómatas:
- Muestra definiciones en JSON de AFD, AP y MT de ejemplo.
- Comparador de gramáticas:
- Recibe dos gramáticas.
- Genera cadenas hasta longitud n.
- Compara qué cadenas acepta cada una (heurística de equivalencia).
- Modo Tutor:
- Genera una gramática o autómata de forma aleatoria.
- El estudiante selecciona el tipo de lenguaje.
- El sistema indica si la respuesta es correcta y explica el por qué.



Resultados, conclusiones

Resultados:

- El sistema clasifica correctamente varios ejemplos clásicos de cada tipo.
- Genera explicaciones paso a paso que ayudan a entender la jerarquía.
- Permite visualizar gramáticas y autómatas con diagramas.

Conclusiones:

- El proyecto integra teoría de lenguajes formales con programación en Python.
- Sirve como herramienta de apoyo para estudiantes y como demostración en clase.