Universidad del Valle de Guatemala

Facultad de Ingeniería

Departamento de Ciencia de la Computación

Teoría de la Computación

Sección 20

Proyecto 3 - Final

Simulador de Máquina de Turing con YAML, `graphviz` y UI en Streamlit

Grupo # 4
Integrante 1: Pablo Daniel Barillas Moreno, Carné No. 22193
Integrante 2: Hugo Daniel Barillas Ajin, Carné No. 23556

Profesor: Ing. Bidkar Alexander Pojoy Corzo

Índice

Resumen

Implementamos un simulador *determinista* de Máquinas de Turing (MT) con: (i) **parser YAML** manual tolerante a comentarios e identación, (ii) **modelo de datos** tipado (`@dataclass`) para transiciones y descripciones instantáneas, (iii) **motor de simulación** con cinta expandible y un *registro de memoria* auxiliar (`mem_cache`) como extensión del control, (iv) **validación estática** de la especificación (alfabetos, estados, duplicados), (v) **visualización** de autómata en `graphviz` y (vi) **UI** en Streamlit con tabla de transiciones, ejecución paso a paso y estadísticas. Documentamos decisiones de diseño (*wildcards* de blanco, prioridad de búsqueda de δ, modo estricto), complejidad, y probamos con un banco de ejemplos (A–G) que cubren aceptación, rechazo, *erasers*, permutas $a \leftrightarrow b$ y una simulación determinista de un OR no determinista (paridad/último símbolo).

1. Introducción

Una Máquina de Turing (MT) modela cómputo mediante una cinta potencialmente infinita, un cabezal de lectura/escritura y un *control finito*. Nuestro objetivo fue construir un **simulador docente** de MT con experiencia de uso moderna:

- Cargar definiciones desde YAML (archivo o editor in–app).
- Verificar consistencia de la MT antes de simular.
- Ejecutar *paso a paso*, ver ID (descripción instantánea) y diagrama de estados.
- Mostrar métricas (aceptadas/rechazadas y pasos).

Alcance. El simulador es determinista (una regla válida por configuración), pero admite **comodines de blanco** (`B`/`None`) para facilitar definiciones; puede desactivarse con `strict_mode`. La cinta se expande dinámicamente por ambos extremos y el control incluye un *registro de memoria de 1 símbolo* (`mem_cache`), útil para expresar MTs con un "marcador" interno adicional.

2. Fundamento teórico

2.1. Definición y extensiones

Una MT clásica es una 7–tupla $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$. Nuestro simulador usa una variante aceptadora por *estado final* q_{acc} (no se modela explícito q_{rej}) y añade un **registro de memoria** $m \in \Gamma \cup \{B\}$.

$$\delta : Q \times (\Gamma \cup \{B\})_m \times (\Gamma \cup \{B\})_{\text{tape}} \;\to\; Q \times (\Gamma \cup \{B\})_m \times (\Gamma \cup \{B\})_{\text{tape}} \times \{L, R, S\}.$$

Una **ID** se registra como $(q,\ m,\ \ldots a\underline{x}b\ldots)$ con el cabezal sobre x. La máquina *acepta* cuando $q = q_{\text{acc}}$; *rechaza* al quedar sin regla aplicable o por límite de pasos.

2.2. Criterios de validación

Antes de simular se verifica: (i) estados inicial/final válidos y sin duplicados; (ii) símbolos de cinta en Γ (blanco admitido como `-`/`None`/`B`); (iii) $\Sigma \subseteq \Gamma$; (iv) no hay δ duplicadas; (v) cadenas de prueba consistentes con Σ; (vi) coherencia de uso del separador `#`.

3. Arquitectura del programa

La solución se divide en **cinco capas** (Figura 1).

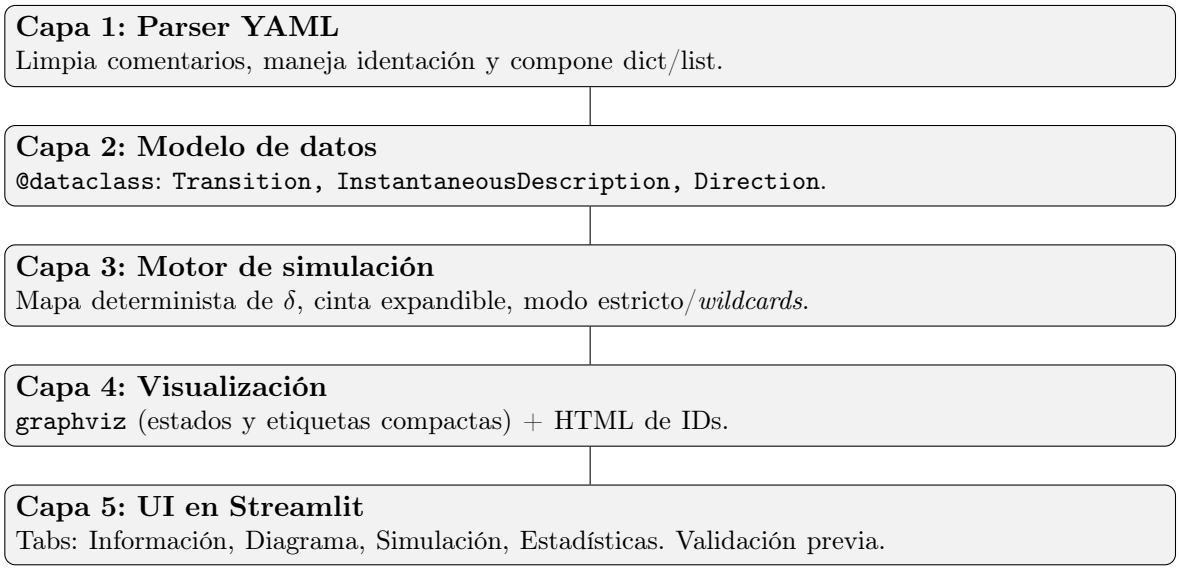

Figura 1: Capas principales de la arquitectura.

3.1. Parser YAML manual

Archivo: dentro del mismo módulo, clase `YAMLParser`. Soporta:

- **Comentarios** fuera de cadenas: se ignoran tras '#'.
- **Cadenas** entrecomilladas simples/dobles (se conserva el contenido).
- **Indentación** en espacios: construye árboles dict/list con una pila (_Node).
- **Escalares**: convierte '', ~, `null` en `None`.

Fragmento clave.

```
class YAMLParser:
    @staticmethod
    def parse(yaml_content: str) -> Dict[str, Any]:
        # strip_comment(), indent_of(), parse_scalar(), next_significant()
        # Pila de _Node(container, indent) para construir dict/list anidados
        ...
```

3.2. Modelo de datos (`@dataclass`)

- `TransitionParams(initial_state, mem_cache_value, tape_input)`
- `TransitionOutput(final_state, mem_cache_value, tape_output, tape_displacement)`
- `Transition(params, output)`: imprime δ en notación formal.
- `InstantaneousDescription`: serializa la cinta e ID a HTML estilizado.
- `Direction`: {L,R,S}.

3.3. Motor de simulación

Índice determinista de transiciones. Se construye `transition_map[(q,m,t)] = Transition` y se marcan duplicados. La búsqueda aplica **prioridad de comodines**: exacta $\rightarrow (m, B) \rightarrow (B, t) \rightarrow (B, B)$ (desactivable con `strict_mode`).

```
def _candidates(self, q, m, t):
    m = None if _is_blank(m) else str(m)
    t = None if _is_blank(t) else str(t)
    yield (q, m, t); yield (q, m, None)
    yield (q, None, t); yield (q, None, None)
```

Ejecución.

1. Inicializa cinta: $[B]$ + `list(input)` + $[B]$ y cabezal al primer símbolo (o celda 0 si vacía).
2. En cada paso: busca δ, escribe, actualiza estado y `mem_cache`, mueve $L/R/S$.
3. **Extiende** la cinta si el cabezal sale por los extremos.
4. **Detiene** si alcanza q_{final} o no hay δ aplicable.

Complejidad. Búsqueda $O(1)$ por paso (dict), por lo que el costo es $O(\#\text{pasos})$.

3.4. Visualización

`to_graphviz()` agrupa etiquetas por par ($q\_\text{src}, q\_\text{dst}$) y formatea [cache_in], tape_in $\rightarrow$ [cache_out], tape_out, dir. El HTML de ID resalta la celda del cabezal y muestra **Estado**, **Cache** y **Posición**.

3.5. Interfaz (Streamlit)

Barra lateral: modo (Ejemplos, Cargar, Editor), `max_steps`, `strict_mode`, `show_graph`, entrada personalizada. **Tabs:**

- *Información*: estados, alfabetos, tabla de transiciones (`pandas`).
- *Diagrama*: `graphviz` + leyenda.
- *Simulación*: resultado por cadena, última transición, IDs completas (expander).
- *Estadísticas*: métricas (`st.metric`) y tabla resumen.

4. Validación de la especificación

`validate_machine(...)` realiza:

1. Estados inicial/final válidos; duplicados; estados no usados.
2. Símbolos de cinta en Γ (incluye `B/None`); coherencia de $\Sigma \subseteq \Gamma$.
3. Detección de **transiciones duplicadas** por clave (q, m, t) (con blancos normalizados).
4. Cadenas de simulación consistentes con Σ (advierte si aparece '#' fuera del alfabeto).

Si hay *issues*, la UI los muestra y **no** ejecuta simulaciones.

5. Ejemplos y resultados

El repositorio integra ejemplos A–G (definidos en `EXAMPLES`):

- **A:** Aceptador universal cíclico (3 fases).
- **B:** Borrador (marca con X → limpia → acepta).
- **C:** Intercambio $a \leftrightarrow b$ y aceptación.
- **D–F:** Rechazadores/sumideros; HALT no aceptante.
- **G:** OR (termina en a *o* paridad de a par) simulando un NDT con ramas serializadas.

Lecturas típicas de la UI. Para cada cadena se reporta:

- **ACEPTADA/RECHAZADA**, # de pasos, estado final textual (*"SIN δ"* si no hay regla).
- Última transición aplicada en notación δ.
- IDs completas (opcional), confirmando escritura y movimiento del cabezal.

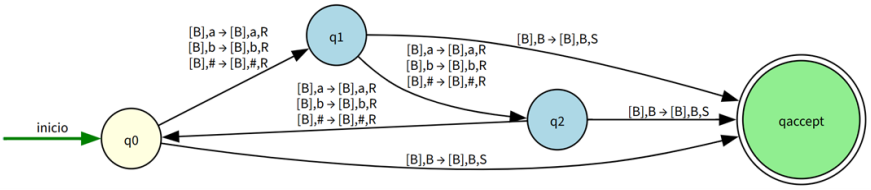

Figura 2: Diagrama de estados generado para un ejemplo (placeholder).

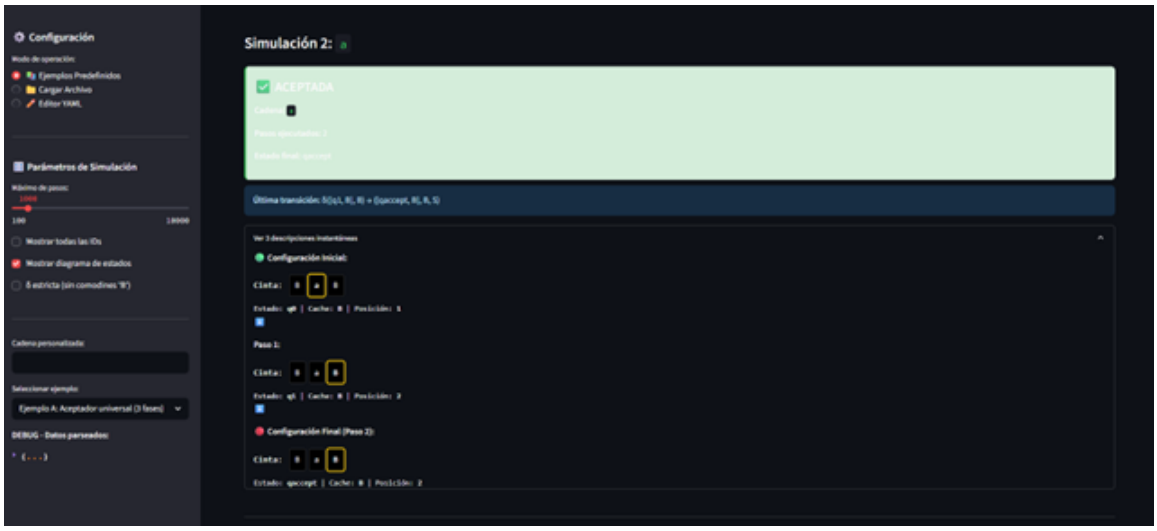

Figura 3: Simulación paso a paso con ID en HTML (placeholder).

6. Manual técnico: API y reproducibilidad

6.1. Construcción desde YAML

```
tm, simulation_strings, dup_msgs = build_turing_machine_from_yaml(yaml_content,
↪  strict_mode=False)
```

Devuelve la MT, las cadenas de prueba y mensajes de duplicados (si los hay).

6.2. Clases y funciones clave

`class TuringMachine`

- `find_transition(q, m, t) ->Transition|None`: aplica prioridad de comodines o *estricto*.
- `simulate(input_string, max_steps) ->(bool, [ID], last_transition)`: ejecuta y retorna aceptada/rechazada, la traza de ID y la última transición.
- `to_graphviz() ->Digraph`: diagrama agrupado por aristas.

`export_transitions_table(tm) ->pd.DataFrame` Tabla amigable para `st.dataframe`.

6.3. Contrato de blancos (B)

El símbolo blanco puede especificarse como `None`, cadena vacía o `'B'` en YAML. Internamente se normaliza a `None` para clave de δ, y se renderiza como `B` al mostrar.

6.4. Modo estricto vs. comodines

- **Estricto**: sólo coincide (q, m, t) exacto.
- **Flexible (por defecto)**: prueba (q, m, t), (q, m, B), (q, B, t), (q, B, B).

Esto permite definir reglas "por defecto" con blanco como *wildcard*.

6.5. Buenas prácticas de YAML (plantilla mínima)

```
---
q_states: { q_list: ['q0','q1','qaccept'], initial: 'q0', final: 'qaccept' }
alphabet: ['a','b','#']
tape_alphabet: ['#', -]   # el guion (-) representa el blanco
delta:
  - params: { initial_state: 'q0', mem_cache_value: , tape_input: a }
    output: { final_state: 'q1', mem_cache_value: , tape_output: X, tape_displacement:
    ↪  R }
  - params: { initial_state: 'q1', mem_cache_value: , tape_input:  }
    output: { final_state: 'qaccept', mem_cache_value: , tape_output: ,
    ↪  tape_displacement: S }
simulation_strings: ['a','aa','']
```

7. Decisiones de diseño y discusión

(D1) δ indexada y prioridad de búsqueda. Elegimos dict $O(1)$ para aplicar reglas. La prioridad con comodines minimiza verbosidad en YAML y emula reglas "por defecto". `strict_mode` garantiza lectura *exacta* cuando se requiera formalismo fuerte.

(D2) Registro `mem_cache`. Modela un *marcador interno de 1 símbolo* que simplifica varias construcciones (p.ej., recordar un símbolo al cruzar una región). Es equivalente a expandir Q, pero más legible en YAML.

(D3) Cinta expandible inmediata. Al mover fuera de los extremos, la cinta se *expande* y el cabezal se reubica. Esto evita *IndexError* y respeta la MT infinita teórica.

(D4) Validación *antes* de simular. Frena errores comunes: olvidar el blanco en Γ, usar '#' sin declararlo, o introducir transiciones duplicadas.

(D5) UX con Streamlit. Tabs separan *definición* (tabla/diagrama) de *ejecución* (IDs) y *métricas*. El HTML de la ID hace evidente el estado y la posición del cabezal.

8. Conclusiones

El simulador cumple objetivos docentes: especificación legible (YAML), ejecución transparente (IDs y última δ), y visualización clara (Graphviz/Streamlit). La arquitectura por capas separa preocupaciones (parseo, modelo, motor, vista), lo que facilita extensión (p. ej., añadir *breakpoints*, exportar traza o soportar no determinismo con backtracking).

9. Referencias

Referencias

[1] Copeland, B. J. (2023). *Turing Machines*. Stanford Encyclopedia of Philosophy. Recuperado de https://plato.stanford.edu/entries/turing-machine/

[2] University of Illinois at Urbana–Champaign (s. f.). *Turing Machines and Computability* (apuntes en PDF). Recuperado de https://courses.engr.illinois.edu/cs498374/fa2011/lec/Turing.pdf

[3] University of Maryland (s. f.). *CMSC451: Turing Machines* (apuntes). Recuperado de https://www.cs.umd.edu/class/spring2018/cmsc451/TuringMachines.pdf

[4] University of California, Davis (s. f.). *ECS 120 — Theory of Computation* (material de curso). Recuperado de https://web.cs.ucdavis.edu/~peisert/teaching/ecs120/

[5] Wikipedia (2025). *Turing machine*. Recuperado de https://en.wikipedia.org/wiki/Turing_machine

[6] Streamlit, Inc. (s. f.). *API reference — Elements in Streamlit*. Recuperado de https://docs.streamlit.io/library/api-reference

[7] Streamlit, Inc. (s. f.). *Status elements (st.metric, st.spinner, etc.)*. Recuperado de https://docs.streamlit.io/library/api-reference/status

[8] Streamlit, Inc. (s. f.). *st.graphviz_chart — Display a Graphviz chart*. Recuperado de https://docs.streamlit.io/library/api-reference/charts/st.graphviz_chart

[9] Streamlit, Inc. (s. f.). *st.tabs — Tabbed layout*. Recuperado de https://docs.streamlit.io/library/api-reference/layout/st.tabs

[10] Streamlit, Inc. (s. f.). *st.expander*. Recuperado de https://docs.streamlit.io/library/api-reference/layout/st.expander

[11] Streamlit, Inc. (s. f.). *st.file_uploader*. Recuperado de https://docs.streamlit.io/library/api-reference/widgets/st.file_uploader

[12] Streamlit, Inc. (s. f.). *Session State API (st.session_state)*. Recuperado de https://docs.streamlit.io/library/api-reference/session-state

[13] The Graphviz Authors (s. f.). *Graphviz — Graph Visualization Software*. Recuperado de https://graphviz.org/

[14] Gansner, E. R., Koutsofios, L., & North, S. C. (s. f.). *The DOT Language* (documentación). Recuperado de https://graphviz.org/doc/info/lang.html

[15] The Graphviz Authors (s. f.). *Graphviz attributes reference*. Recuperado de https://graphviz.org/doc/info/attrs.html

[16] xflr6. (s. f.). *python-graphviz — Python interface to Graphviz* (GitHub). Recuperado de https://github.com/xflr6/graphviz

[17] Python Package Index (s. f.). *graphviz — PyPI*. Recuperado de https://pypi.org/project/graphviz/

[18] pandas development team (s. f.). *pandas User Guide*. Recuperado de https://pandas.pydata.org/docs/user_guide/index.html

[19] pandas development team (s. f.). *pandas.DataFrame — Reference*. Recuperado de https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

[20] Ben-Kiki, O., Evans, C., & döt Net, I. (2009/2021). *YAML Ain't Markup Language (YAML) Version 1.2*. Recuperado de https://yaml.org/spec/1.2/spec.html

[21] yaml.org (s. f.). *YAML 1.2 — FAQ y recursos*. Recuperado de https://yaml.org/

[22] Python Software Foundation (s. f.). *dataclasses — Data Classes*. Recuperado de https://docs.python.org/3/library/dataclasses.html

[23] Python Software Foundation (s. f.). *enum — Support for enumerations*. Recuperado de https://docs.python.org/3/library/enum.html

[24] Python Software Foundation (s. f.). *typing — Support for type hints*. Recuperado de https://docs.python.org/3/library/typing.html

[25] Python Software Foundation (s. f.). *typing.Optional*. Recuperado de https://docs.python.org/3/library/typing.html#typing.Optional

[26] Tantau, T. (2025). *The PGF/TikZ Manual*. Recuperado de https://ctan.org/pkg/pgf

[27] Oberdiek, H., et al. (s. f.). *hyperref package — CTAN*. Recuperado de https://ctan.org/pkg/hyperref

Apéndice A — Extractos del código

A.1. Índice determinista y duplicados

```
self.transition_map: Dict[Tuple[str, Optional[str], Optional[str]], Transition] = {}
self.duplicates: List[Tuple[str, Optional[str], Optional[str]]] = []
for t in transitions:
    cache_key = None if _is_blank(t.params.mem_cache_value) else
    ↪  str(t.params.mem_cache_value)
    tape_key  = None if _is_blank(t.params.tape_input)      else
    ↪  str(t.params.tape_input)
    key = (t.params.initial_state, cache_key, tape_key)
    if key in self.transition_map:
        self.duplicates.append(key)
    else:
        self.transition_map[key] = t
```

A.2. Prioridad de búsqueda de δ

```
def find_transition(self, state, mem_cache, tape_symbol):
    if self.strict_mode:
        m = None if _is_blank(mem_cache)  else str(mem_cache)
        t = None if _is_blank(tape_symbol) else str(tape_symbol)
        return self.transition_map.get((state, m, t))
    for k in self._candidates(state, mem_cache, tape_symbol):
        tr = self.transition_map.get(k)
        if tr is not None:
            return tr
    return None
```

A.3. Serialización de la ID a HTML

```
def to_html(self) -> str:
    base = ("background:#000;color:#fff;padding:6px 10px; ...;
    ↪  font-family:monospace;")
    head = base + "outline:3px solid #ffcc00;font-weight:bold;"
    tape_html = '<div style="display:flex;...">Cinta:</span>'
    for i, symbol in enumerate(self.tape):
        sym = symbol if symbol is not None else 'B'
        style = head if i == self.head_position else base
        tape_html += f'<span style="{style}">{sym}</span>'
    ...
```