Universidad Rafael Landívar

Facultad de Ingeniería

Ingeniería en Informática y Sistemas

Lenguajes formales y autómatas

Docente: Ing. Julio Requena

TERCER PROYECTO

Simulador de Máquina de Turing

Javier Alessandro Rivera Lemus - 1241224

Guatemala, Ciudad de Guatemala, 07 de noviembre de 2025

I. MANUAL DE USO

1) Arquitectura general

El programa es un solo archivo con tres capas:

Modelo (lógica de MT): clase TuringMachine.

Construcción de máquinas: utilitario tm_from_dfa y build_predefined_machines().

Interfaz: clase App (Tkinter) que orquesta los pasos y dibuja la cinta.

La MT se ejecuta en memoria (sin hilos). La interfaz dispara los pasos mediante callbacks y el bucle de eventos de Tk.

2) Modelo: TuringMachine

2.1 Estructura

```
@dataclass
class TuringMachine:
    states: Set[str]
    input_alphabet: Set[str]
    tape_alphabet: Set[str]
    blank: str
    transitions: Dict[Tuple[str,str], Tuple[str,str,str]]  # (next_state, write, move)
    start_state: str
    accept_states: Set[str]
    reject_states: Set[str] = field(default_factory=set)

    tape: Dict[int, str] = field(default_factory=dict)  # índice -> símbolo
```

head: int = 0

state: str = ""

halted: bool = False

accepted: Optional[bool] = None

Cinta: dict disperso pos -> símbolo (eficiente para extremos en blanco).

Cabezal: índice entero head (puede ser negativo).

Transiciones: clave (estado, símbolo_leído) → valor (nuevo_estado, símbolo_a_escribir, movimiento).

Movimientos: 'L', 'R', 'N'.

2.2 Ciclo de vida

reset(cadena): carga la entrada en tape[0..n-1], coloca head=0, state=start_state.

read(): retorna tape[head] o blank si no hay símbolo.

write(s): escribe s en la celda actual.

step():

Lee (state, read()).

Si no hay transición:

marca halted=True y accepted = (state in accept_states).

devuelve False.

Si hay transición: escribe, mueve el cabezal, actualiza state. Devuelve True.

Criterio de aceptación: cuando ya no existe transición para el par (estado, símbolo). Si el estado actual está en accept_states, la palabra se acepta.

3) De DFA a MT “lectora”: tm_from_dfa(...)

Convierte un DFA a una MT que solo lee y avanza a la derecha:

```
def tm_from_dfa(states, alphabet, start, accepts, delta, blank='□') -> TuringMachine:
    transitions = {}
    for (q, a), p in delta.items():
        transitions[(q, a)] = (p, a, 'R')   # no cambia el símbolo; mueve R
    return TuringMachine(..., transitions=transitions, accept_states=accepts)
```

La detención ocurre al llegar al blanco (no existe transición con blank), por lo que se decide con el estado en ese punto.

Útil para lenguajes regulares (casos de prueba).

4) Máquinas predefinidas

build_predefined_machines() crea 5 máquinas desde DFA:

(ab)* sobre {a,b}

0*1* sobre {0,1}

(a|b)*abb sobre {a,b}

1(01)*0 sobre {0,1}

(a|b)*a(a|b)* sobre {a,b}

Cada una define:

states, alphabet, start, accepts, delta (función de transición DFA).

Se llama tm_from_dfa(...) y se añade al dict PREDEFINED_MACHINES.

Para agregar otra máquina regular, copie un bloque, defina su delta y añada al diccionario.

5) Módulo de pruebas con expresiones regulares

REGEX_TESTS es una lista de 10 patrones para evaluar con re.fullmatch:

REGEX_TESTS = [(r'(a|b)*abb','a/b'), ..., (r'ε|','a/b')]

La UI los ejecuta con eval_regex().

load_mt_for_regex() mapea 5 de esas regex a su MT equivalente para ver el paso a paso.

6) Interfaz (App)

6.1 Componentes relevantes

Entrada (cadena de la MT) + Cargar → llama on_load() ⇒ tm.reset(...).

Paso → on_step() ⇒ tm.step() + refresh_view().

Auto → on_run_toggle() activa un lazo con after(step_delay_ms).

Velocidad fija: step_delay_ms = 300 ms (se eliminó el control en UI).

Cinta: se muestra una ventana de CELL_SPAN celdas centrada en head. La celda del cabezal se resalta.

6.2 Flujo principal

on_load() -> reset -> refresh_view

on_step() -> step -> refresh_view

_run_loop(): while running and not halted -> step -> refresh_view -> after(300)

6.3 Refresco de vista

Estado/Cabezal: texto con state y head.

Resultado: si halted, muestra “ACEPTADA” o “RECHAZADA”.

Cinta: lee tape.get(idx, blank) para cada índice visible y actualiza etiquetas.

7) Alfabetos y blancos

input_alphabet y tape_alphabet se declaran al construir la MT; en las lectoras tape_alphabet = input_alphabet ∪ {blank}.

blank por defecto es '□' (carácter visible en UI). En la cinta lógica es el mismo símbolo.

Si la entrada incluye símbolos fuera del alfabeto, al llegar a ese símbolo no habrá transición y se terminará en rechazo (comportamiento intencional).

8) Extender el simulador

8.1 Añadir un nuevo DFA (lenguaje regular)

```
states  = {...}
alphabet= {...}
start   = 'q0'
accepts = {...}
delta   = { ('q', 'a'): 'p', ... }
machines['NOMBRE'] = tm_from_dfa(states, alphabet, start, accepts, delta)
```

8.2 Definir una MT general (que escribe/mueve a la izquierda)

Construya directamente la tabla transitions con tuplas (next_state, write_symbol, move):

```
tm = TuringMachine(
    states={...},
    input_alphabet={'0','1'},
    tape_alphabet={'0','1','X','□'},
    blank='□',
    transitions={
        ('q0','1'): ('q1','X','R'),
```

```
        ('q1','1'): ('q1','1','R'),
        ('q1','0'): ('q2','0','L'),
        # ...
    },
    start_state='q0',
    accept_states={'q_accept'},
    reject_states={'q_reject'}
)
```

Cárguela en interfaz: añádala a PREDEFINED_MACHINES['nombre'] = tm.

8.3 Cambiar retardo del modo automático

Modificar constante:

self.step_delay_ms = 300

9) Errores y casos límite

Sin transición para (estado, símbolo) ⇒ se detiene:

accepted = True si estado ∈ accept_states, de lo contrario False.

Entrada vacía: reset("") coloca cinta vacía y se decide al primer step() (leerá blank).

Cinta infinita: el dict disperso permite índices negativos/positivos sin tamaño fijo.

10) Pruebas unitarias sugeridas (rápidas)

Aceptación/rechazo: para cada MT predefinida, validar un set de cadenas típicas.

Determinismo de lectoras: verificar que nunca escriben símbolos distintos (write == read).

Bordes: "", un solo símbolo, símbolos fuera del alfabeto, movimiento hacia índices negativos.

Ejemplo mínimo (pytest):

```
def run(tm, s):
    tm.reset(s)
    while tm.step(): pass
    return tm.accepted
def test_ab_star():
    tm = PREDEFINED_MACHINES['(ab)*']
    assert run(tm, "") is True
    assert run(tm, "ab") is True
    assert run(tm, "aba") is False
```

11) Convenciones y constantes

CELL_SPAN = 31 (ancho visible de cinta).

CENTER = CELL_SPAN // 2.

blank = '□'.

Única dependencia de GUI: tkinter y ttk.