Centro Universitario de Ciencias Exactas e Ingeniería

Sem. de Solución de Problemas de Traductores 2

Actividad 3.- Autómatas finitos

LUIS FELIPE MUNOZ MENDOZA

Juan Pablo Hernández Orozco

219294285

Objetivo:  
El objetivo principal es implementar una interfaz gráfica (GUI) que facilite la validación de entradas de datos mediante patrones definidos con expresiones regulares. Esto permite detectar y notificar al usuario cualquier error en la sintaxis de los datos introducidos, asegurando que estos cumplan con el formato requerido

# Introducción:

En el fascinante cruce entre la teoría de autómatas y la teoría musical, se abre un campo de estudio que une la precisión de la informática con la creatividad del arte. Este proyecto ilustra cómo un Autómata Finito Determinista (AFD) puede ser empleado para validar secuencias musicales, determinando si una serie de notas y acordes corresponde a una escala mayor o a alguna de sus variantes menores (natural, armónica o melódica).

Utilizando Python y la biblioteca Tkinter para la creación de interfaces gráficas, el programa ofrece una herramienta interactiva que permite al usuario ingresar una tonalidad, seleccionar el tipo de escala y probar secuencias musicales en tiempo real. Bajo la superficie, el AFD implementado opera de manera determinista: cada símbolo (nota o acorde) es evaluado de forma secuencial, y cualquier desviación de los conjuntos predefinidos provoca un rechazo inmediato de la secuencia.

# Desarrollo:

## Diseño del autómata

Tipo: Se trata de un Autómata Finito Determinista (AFD).

* Determinista: Porque, para cada entrada y en cada paso, la transición está definida de forma única. No hay ambigüedad en el camino a seguir.
* Finito: Debido a que el conjunto de estados y el alfabeto (notas y acordes) es finito.

Componentes del AFD

**a. Estados**

* **Estados Implícitos:**  
  Aunque el código no declara estados explícitos (como variables separadas), podemos interpretar la lógica de la siguiente manera:
  + **Estado Inicial/Activo (Aceptación):** Se asume que, al comenzar, la secuencia es válida. El autómata “se encuentra” en un estado aceptante.
  + **Estado de Rechazo:** Si en algún momento se detecta un símbolo (nota o acorde) que no pertenece al conjunto definido para la escala, el autómata transiciona a un estado de rechazo, y se interrumpe la validación devolviendo un mensaje de error.

**b. Alfabeto de Entrada**

* **Símbolos del Alfabeto:**  
  El alfabeto consiste en los **elementos musicales** (notas y acordes) definidos en los conjuntos asociados a cada escala. Por ejemplo, para la escala mayor de C, el alfabeto es:
  + **Notas:** {"C", "D", "E", "F", "G", "A", "B"}
  + **Acordes:** {"C", "Dm", "Em", "F", "G", "Am", "Bdim", "G7"}

**c. Función de Transición**

* **Definición Implícita:**  
  La función de transición se implementa en el método validar\_secuencia mediante la iteración sobre cada elemento de la secuencia:
  + **Para cada símbolo (elemento):**
    - Si el símbolo pertenece a alguno de los conjuntos ("notas" o "acordes") de la escala seleccionada, se “permanece” en el estado aceptante.
    - Si el símbolo **no pertenece** a ninguno de esos conjuntos, se produce una transición al estado de rechazo, y el proceso se detiene inmediatamente.

Este comportamiento se logra mediante la condición:

if elemento not in escala["notas"] and elemento not in escala["acordes"]:

return f"❌ Secuencia inválida..."

**d. Estado Inicial y Estados de Aceptación**

* **Estado Inicial:**  
  El proceso comienza asumiendo que la secuencia es potencialmente válida. No hay una verificación previa, lo que equivale a iniciar en el estado "aceptante".
* **Estados de Aceptación:**
  + **Aceptación:** Si, al procesar todos los símbolos de la secuencia, no se ha encontrado ningún elemento fuera del alfabeto, el autómata “permanece” en el estado de aceptación, retornando un mensaje de éxito:

return f"✅ Secuencia válida..."

**Rechazo:**

* Si en algún momento se detecta un símbolo inválido, se “transiciona” de manera inmediata a un estado de rechazo, retornando un mensaje de error que especifica el símbolo no reconocido.

Lógica Subyacente y Procesamiento de Entradas

* **Proceso Secuencial:**  
  El autómata procesa la secuencia de entrada de forma **lineal y secuencial**:

1. **Inicio:** El autómata recibe la tonalidad y el tipo de escala.
2. **Validación de la Configuración:** Se verifica que la tonalidad y el tipo de escala existen en el diccionario.
3. **Iteración sobre la Secuencia:** Cada símbolo de la secuencia se evalúa de manera determinista:
   * **Símbolo válido:** Se continúa con el siguiente elemento.
   * **Símbolo inválido:** Se detiene el proceso y se retorna un mensaje de error.

* **Determinismo:**  
  Cada símbolo tiene una única “decisión” (aceptado o rechazado) basada en su pertenencia a los conjuntos predefinidos. No existen bifurcaciones ni múltiples caminos posibles.

Ejemplos de Cadenas de Entrada

Aunque vamos a ver mas a delante corridas de escritorio, primero veremos dos ejemplos rápidos para poder comprender lo que esperamos en los resultados.

**Ejemplo 1: Secuencia Aceptada**

* **Entrada:**
  + Tonalidad: "C"
  + Tipo de escala: "mayor"
  + Secuencia: ["C", "E", "G", "F", "A", "Dm", "G7"]
* **Proceso:**  
  Cada elemento se comprueba contra los conjuntos:
  + "C", "E", "G", "F", "A" están en las notas válidas.
  + "Dm" y "G7" están en los acordes válidos.
* **Resultado:**  
  La secuencia es aceptada y se retorna:  
  "✅ Secuencia válida en C mayor."

**Ejemplo 2: Secuencia Rechazada**

* **Entrada:**
  + Tonalidad: "A"
  + Tipo de escala: "menor natural"
  + Secuencia: ["Am", "Dm", "Em", "G7", "B7"]
* **Proceso:**  
  Se comprueba cada elemento:
  + "Am", "Dm", "Em" están en la lista de acordes o notas válidas para la escala menor de A.
  + "G7" puede ser aceptado si estuviera definido, pero en este caso, la escala de A menor natural no lo incluye.
  + "B7" no pertenece a ninguno de los conjuntos definidos.
* **Resultado:**  
  Al encontrar "B7" (o incluso "G7" si no está definido), el autómata transiciona a un estado de rechazo y retorna:  
  "❌ Secuencia inválida en A menor natural. 'B7' no pertenece a la escala."

# Código

Para esta práctica vamos a utilizar la librería de tkinter para poder visualizar los estados en una GUI (Interfaz grafica de usuario)

import tkinter as tk

Se importa el módulo tkinter, que es la biblioteca estándar de Python para crear interfaces gráficas. Se renombra como tk para abreviar su uso en el código.

class AFDValidadorEscalas:

se define una clase que encapsula el autómata finito determinista (AFD) encargado de validar secuencias de notas y acordes según una escala musical seleccionada.

# Diccionario ampliado: escalas mayores y variantes de escalas menores.

self.escalas = {

Se crea un atributo escalas, un diccionario que almacena las definiciones de distintas escalas musicales. Cada clave de primer nivel corresponde al tipo de escala (por ejemplo, "mayor", "menor natural", etc.).

"mayor": {

"C": {"notas": {"C", "D", "E", "F", "G", "A", "B"},

"acordes": {"C", "Dm", "Em", "F", "G", "Am", "Bdim", "G7"}},

* **Notas:** Se definen las notas válidas para la escala mayor de C usando un conjunto (set) para permitir búsquedas rápidas.
* **Acordes:** Se listan los acordes válidos para la tonalidad, incluyendo acordes mayores, menores, disminuidos y séptimas.

**Se definen otras tonalidades y escalas:**  
Para "A", "G" y "F" en la categoría "mayor", y se incluyen escalas menores (natural, armónica y melódica) para tonalidades específicas (por ejemplo, "A" y "C"). Cada entrada contiene dos claves: "notas" y "acordes", que representan el conjunto de símbolos aceptados para esa escala.

def validar\_secuencia(self, tonalidad, tipo\_escala, secuencia):

Recibe tres parámetros:

* tonalidad: La clave musical (por ejemplo, "C", "A", etc.).
* tipo\_escala: El tipo de escala (por ejemplo, "mayor", "menor natural", etc.).
* secuencia: Una lista de elementos (notas o acordes) que se desean validar.

if tipo\_escala not in self.escalas or tonalidad not in self.escalas[tipo\_escala]:

return "⚠️ Tonalidad o tipo de escala no válidos."

* Comprueba si el tipo\_escala existe en el diccionario self.escalas.
* Luego, verifica que la tonalidad esté definida dentro del tipo de escala seleccionado.
* Si alguna de las condiciones falla, se retorna un mensaje de advertencia indicando que la entrada es inválida.

escala = self.escalas[tipo\_escala][tonalidad]

Extrae del diccionario la estructura (conjuntos de "notas" y "acordes") que corresponde a la tonalidad y tipo de escala seleccionados.

for elemento in secuencia:

if elemento not in escala["notas"] and elemento not in escala["acordes"]:

return f"❌ Secuencia inválida en {tonalidad} {tipo\_escala}. '{elemento}' no pertenece a la escala."

* Itera sobre cada elemento de la secuencia (cada símbolo ingresado por el usuario).
* Para cada elemento, comprueba si no se encuentra en ninguno de los dos conjuntos ("notas" o "acordes") de la escala seleccionada.
* Si se detecta un elemento inválido, se retorna inmediatamente un mensaje indicando el error y especificando el elemento que no pertenece.

return f"✅ Secuencia válida en {tonalidad} {tipo\_escala}."

Si todos los elementos de la secuencia son válidos (es decir, pertenecen a alguno de los conjuntos de la escala), se retorna un mensaje confirmando la validez de la secuencia.

def validar():

Define la función que se ejecutará al presionar el botón "Validar" en la interfaz. Esta función orquesta la obtención de datos, la validación y la actualización de la interfaz.

tonalidad = tonalidad\_entry.get().strip()

* Recupera el contenido del campo de entrada para la tonalidad usando el método .get().
* El método .strip() elimina espacios en blanco iniciales y finales para evitar errores en la validación.

tipo\_escala = tipo\_escala\_var.get().strip().lower()

* Extrae el valor seleccionado del menú desplegable para el tipo de escala.
* Se aplica .strip() para limpiar espacios y .lower() para convertir el texto a minúsculas, asegurando la coincidencia con las claves del diccionario (por ejemplo, "mayor").

secuencia\_text = secuencia\_entry.get().strip()

Aquí se recupera la secuencia ingresada (notas/acordes) y limpia espacios adicionales.

secuencia = secuencia\_text.split()

Convierte la cadena de entrada en una lista de elementos utilizando .split(), asumiendo que cada símbolo está separado por un espacio.

resultado = afd.validar\_secuencia(tonalidad, tipo\_escala, secuencia)

* Llama al método validar\_secuencia del objeto afd (instancia de AFDValidadorEscalas) pasando los parámetros capturados.
* El resultado (mensaje de validación) se almacena en la variable resultado.

resultado\_label.config(text=resultado)

Actualiza la etiqueta resultado\_label de la GUI para mostrar el mensaje obtenido (ya sea de éxito o de error).

afd = AFDValidadorEscalas()

Se crea una instancia de la clase AFDValidadorEscalas y se asigna a la variable afd para utilizarla en la validación

A partir de ahora solo se toma en cuenta la configuración para poder desplegar la GUI con thinker

root = tk.Tk()

root.title("Actividad 3.- Autómatas finitos")

tk.Label(root, text="Tonalidad (ej. C, A, D#):").grid(row=0, column=0, padx=5, pady=5, sticky="w")

tonalidad\_entry = tk.Entry(root)

tonalidad\_entry.grid(row=0, column=1, padx=5, pady=5)

tk.Label(root, text="Tipo de escala:").grid(row=1, column=0, padx=5, pady=5, sticky="w")

tipo\_escala\_var = tk.StringVar(root)

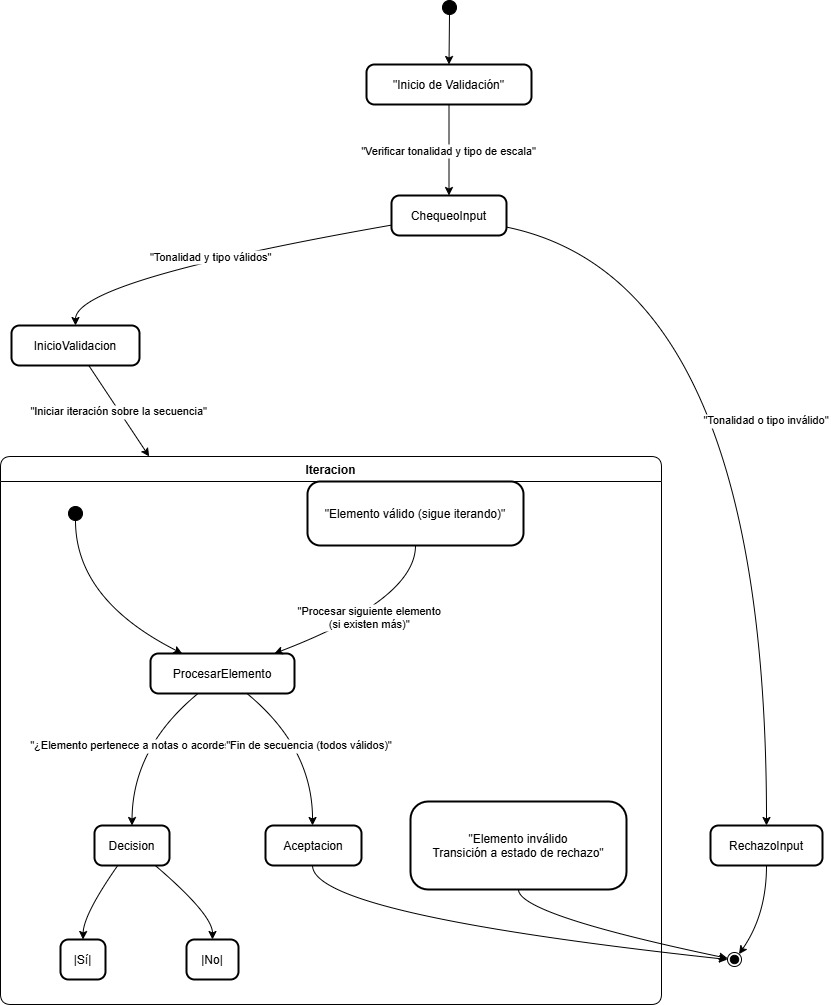
tipo\_escala\_var.set("mayor") # valor por defecto

tipo\_escala\_menu = tk.OptionMenu(root, tipo\_escala\_var, "mayor", "menor natural", "menor armónica", "menor melódica")

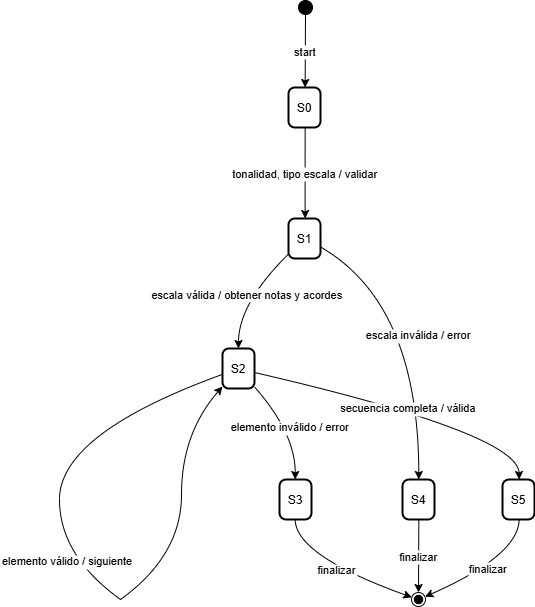
tipo\_escala\_menu.grid(row=1, column=1, padx=5, pady=5)

root.mainloop()

A grandes rasgos la practica funciona de esta manera



La lógica solo de la maquina de estado se ve algo así



# Conclusión