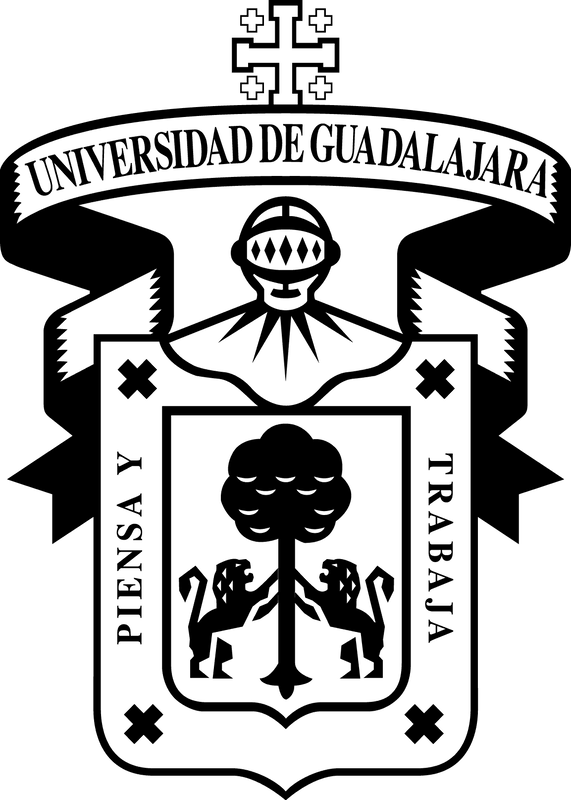
Seminario de Solución de problemas de Traductores de Lenguajes II

# Centro Universitario de Ciencias Exactas en ingenierías

# Universidad de Guadalajara



Maestro: [LUIS FELIPE MUNOZ MENDOZA](mailto:luis.munoz.m@academicos.udg.mx)

Juan Antonio Pérez Juárez

Código: 215660996

Carrera: INCO

Actividad 3 - Autómatas Finitos

Introducción:

Actividad: Validación de Notas y Acordes en Escalas Mayores y Menores

**Objetivo**

Implementar un Autómata Finito Determinista (AFD) que valide si una secuencia de notas y acordes pertenece a una escala mayor o menor elegida por el usuario.

Instrucciones

Desarrollar un programa que:

* Solicite una tonalidad (ej., C, A, D#).
* Solicite el tipo de escala (mayor o menor).
* Valide si una secuencia de notas o acordes pertenece a la escala elegida.

El autómata debe reconocer:

* Notas dentro de la escala.
* Acordes mayores, menores y séptimas según la tonalidad.
* Rechazar cualquier nota o acorde que no pertenezca a la escala seleccionada.

Ejemplo de uso:

Entrada: Tonalidad: C | Tipo: mayor | Secuencia: C E G F A Dm G7 Salida: Secuencia válida en C mayo

Entrada: Tonalidad: A | Tipo: menor | Secuencia: Am Dm Em G7 B7 Salida: Secuencia inválida en A menor.

Desarrollo:

Para implementar el Autómata Finito Determinista que valida secuencias musicales, seguimos los siguientes pasos fundamentales:

*Fase de Análisis y Diseño*

Primero analizamos los requerimientos del sistema y establecimos los elementos clave:

Definimos el alfabeto de entrada: notas musicales (C, D, E, F, G, A, B y sus alteraciones) y acordes (mayores, menores y séptimas).

Identificamos los estados necesarios: inicial (q0), validación (q1) y error (qError).

Establecimos las transiciones válidas entre estados basadas en las reglas musicales.

Determinamos las condiciones de aceptación: secuencia completa con elementos pertenecientes a la escala.

Implementación de la Estructura Base

Desarrollamos la clase principal ScaleValidator con los componentes esenciales:

class ScaleValidator:

def \_\_init\_\_(self):

self.all\_notes = ['C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#', 'B']

Desarrollo de Funciones de Generación de Escalas

Implementamos el método para generar las notas de una escala específica:

def get\_scale\_notes(self, tonic, scale\_type):

tonic = tonic.upper()

start\_idx = self.all\_notes.index(tonic)

major\_intervals = [0, 2, 4, 5, 7, 9, 11]

minor\_intervals = [0, 2, 3, 5, 7, 8, 10]

intervals = major\_intervals if scale\_type.lower() == 'mayor' else minor\_intervals

Implementación de la Validación de Acordes

Creamos la funcionalidad para determinar los acordes válidos:

def get\_scale\_chords(self, scale\_notes, scale\_type):

valid\_chords = set()

chord\_types = ['', 'm', 'm', '', '', 'm', 'm'] if scale\_type.lower() == 'mayor' \

else ['m', 'm', '', 'm', 'm', '', '']

Desarrollo del Motor del Autómata

Implementamos la lógica central de validación:

def validate\_sequence(self, tonic, scale\_type, sequence):

scale\_notes = self.get\_scale\_notes(tonic, scale\_type)

valid\_chords = self.get\_scale\_chords(scale\_notes, scale\_type)

current\_state = 'q0'

Implementación de la Interfaz de Usuario

Desarrollamos la función main para la interacción con el usuario:

def main():

validator = ScaleValidator()

tonic = input("Ingrese la tonalidad: ")

scale\_type = input("Ingrese el tipo de escala: ")

sequence = input("Ingrese la secuencia: ")



Código:

class ScaleValidator:

def \_\_init\_\_(self):

# Definición de notas y sus alteraciones

self.all\_notes = ['C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#', 'B']

def get\_scale\_notes(self, tonic, scale\_type):

"""Genera las notas de la escala basada en la tónica y el tipo"""

tonic = tonic.upper()

start\_idx = self.all\_notes.index(tonic)

# Intervalos para escalas mayores y menores (en semitonos)

major\_intervals = [0, 2, 4, 5, 7, 9, 11]

minor\_intervals = [0, 2, 3, 5, 7, 8, 10]

intervals = major\_intervals if scale\_type.lower() == 'mayor' else minor\_intervals

scale\_notes = []

for interval in intervals:

note\_idx = (start\_idx + interval) % 12

scale\_notes.append(self.all\_notes[note\_idx])

return scale\_notes

def get\_scale\_chords(self, scale\_notes, scale\_type):

"""Genera los acordes válidos para la escala"""

valid\_chords = set()

# Patrones de acordes según el tipo de escala

if scale\_type.lower() == 'mayor':

# I, ii, iii, IV, V, vi, vii°

chord\_types = ['', 'm', 'm', '', '', 'm', 'm']

else: # menor

# i, ii°, III, iv, v, VI, VII

chord\_types = ['m', 'm', '', 'm', 'm', '', '']

# Agregar acordes básicos

for i, note in enumerate(scale\_notes):

chord = note + chord\_types[i]

valid\_chords.add(chord)

# Agregar séptimas

valid\_chords.add(chord + '7')

return valid\_chords

def validate\_sequence(self, tonic, scale\_type, sequence):

"""Valida una secuencia de notas y acordes"""

# Obtener notas y acordes válidos

scale\_notes = self.get\_scale\_notes(tonic, scale\_type)

valid\_chords = self.get\_scale\_chords(scale\_notes, scale\_type)

# Dividir la secuencia en elementos individuales

elements = sequence.split()

# Estado inicial del autómata

current\_state = 'q0'

for element in elements:

# Verificar si el elemento es una nota o acorde

if len(element) <= 2: # Es una nota

if element not in scale\_notes:

return False, f"Nota inválida: {element}"

else: # Es un acorde

if element not in valid\_chords:

return False, f"Acorde inválido: {element}"

# Transición al siguiente estado

current\_state = 'q1'

# Estado final

return True, "Secuencia válida"

def main():

validator = ScaleValidator()

# Solicitar entrada del usuario

tonic = input("Ingrese la tonalidad (ej. C, A, D#): ")

scale\_type = input("Ingrese el tipo de escala (mayor/menor): ")

sequence = input("Ingrese la secuencia de notas y acordes separados por espacios: ")

# Validar la secuencia

is\_valid, message = validator.validate\_sequence(tonic, scale\_type, sequence)

# Mostrar resultado

print(f"\nResultado para {tonic} {scale\_type}:")

print(f"Secuencia: {sequence}")

print(f"{'Válida' if is\_valid else 'Inválida'}: {message}")

# Mostrar información adicional

scale\_notes = validator.get\_scale\_notes(tonic, scale\_type)

valid\_chords = validator.get\_scale\_chords(scale\_notes, scale\_type)

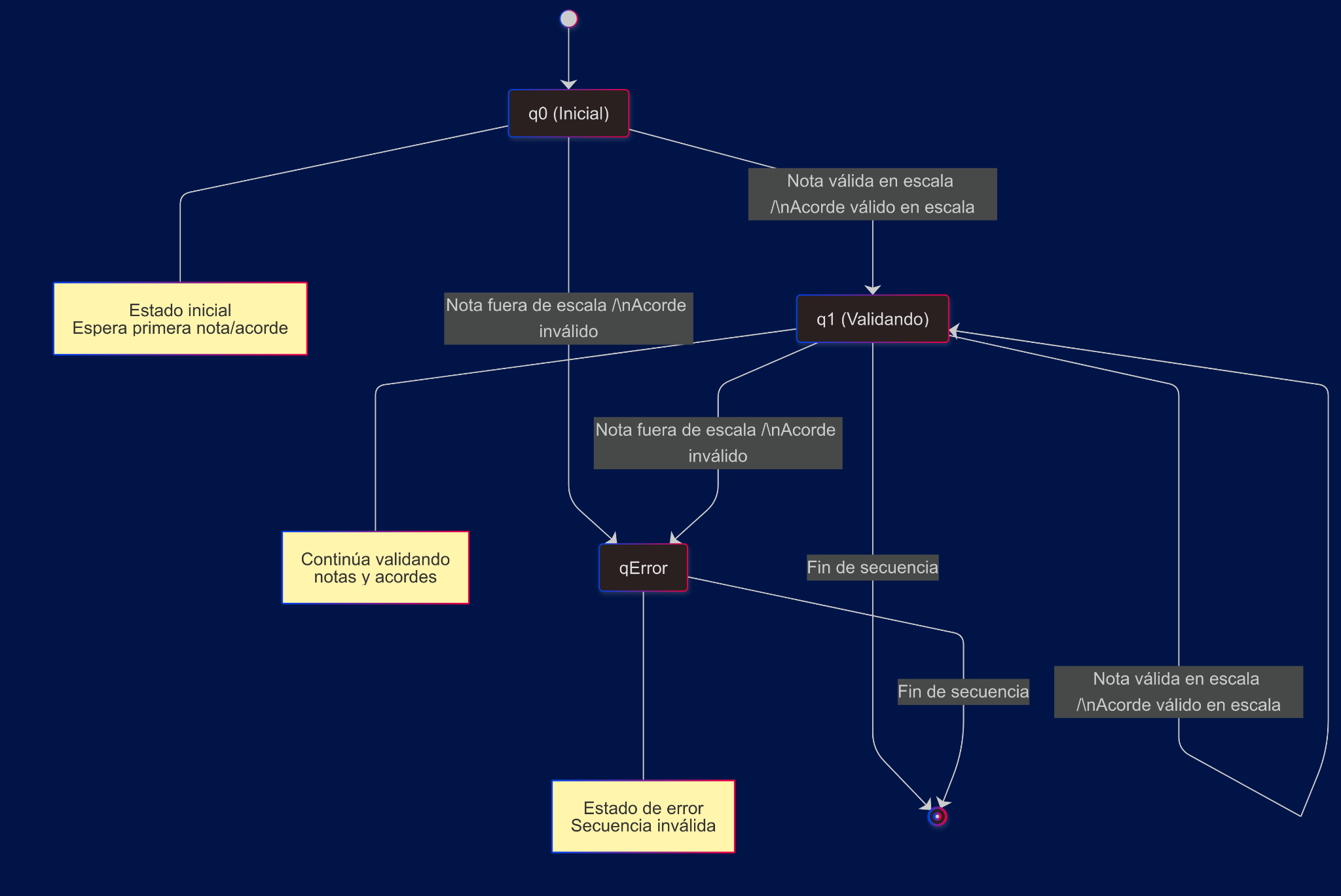
print(f"\nNotas de la escala: {' '.join(scale\_notes)}")

print(f"Acordes válidos: {' '.join(sorted(valid\_chords))}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

En el siguiente diagrama podemos ver claramente como funciona este autómata.



El autómata tiene **tres estados principales:**

**Estado q0 (Inicial):**

Es el estado donde comienza toda validación

Espera recibir el primer elemento de la secuencia

Solo permite dos transiciones:

* Hacia q1 si el elemento es válido
* Hacia qError si el elemento es inválido

**Estado q1 (Validando):**

Estado que indica que la secuencia va correcta hasta el momento

Puede permanecer en sí mismo mientras reciba elementos válidos

Transiciona a qError si recibe un elemento inválido

Puede finalizar la secuencia (es un estado de aceptación)

**Estado qError:**

Estado que indica que la secuencia es inválida

Una vez que se llega a este estado, no hay salida

Es un estado final (aunque de rechazo)

Ejemplo de uso:

Supongamos que tenemos la siguiente entrada:

Tonalidad: C

Tipo: mayor

Secuencia: C Em G7 B

La escala de C mayor contiene las notas: C D E F G A B

Los acordes válidos incluyen: C, Dm, Em, F, G7, Am, Bdim

**Paso a paso:**

Inicio → q0

Lee "C" (primer acorde)

C es válido en C mayor

q0 → q1

Lee "Em" (segundo acorde)

Em es válido en C mayor

q1 → q1

Lee "G7" (tercer acorde)

G7 es válido en C mayor

q1 → q1

Lee "B" (cuarta nota)

B es válida en C mayor

q1 → q1

Fin de secuencia en q1

Secuencia aceptada

Si cambiáramos el último elemento a "B7":

Inicio → q0

C: q0 → q1

Em: q1 → q1

G7: q1 → q1

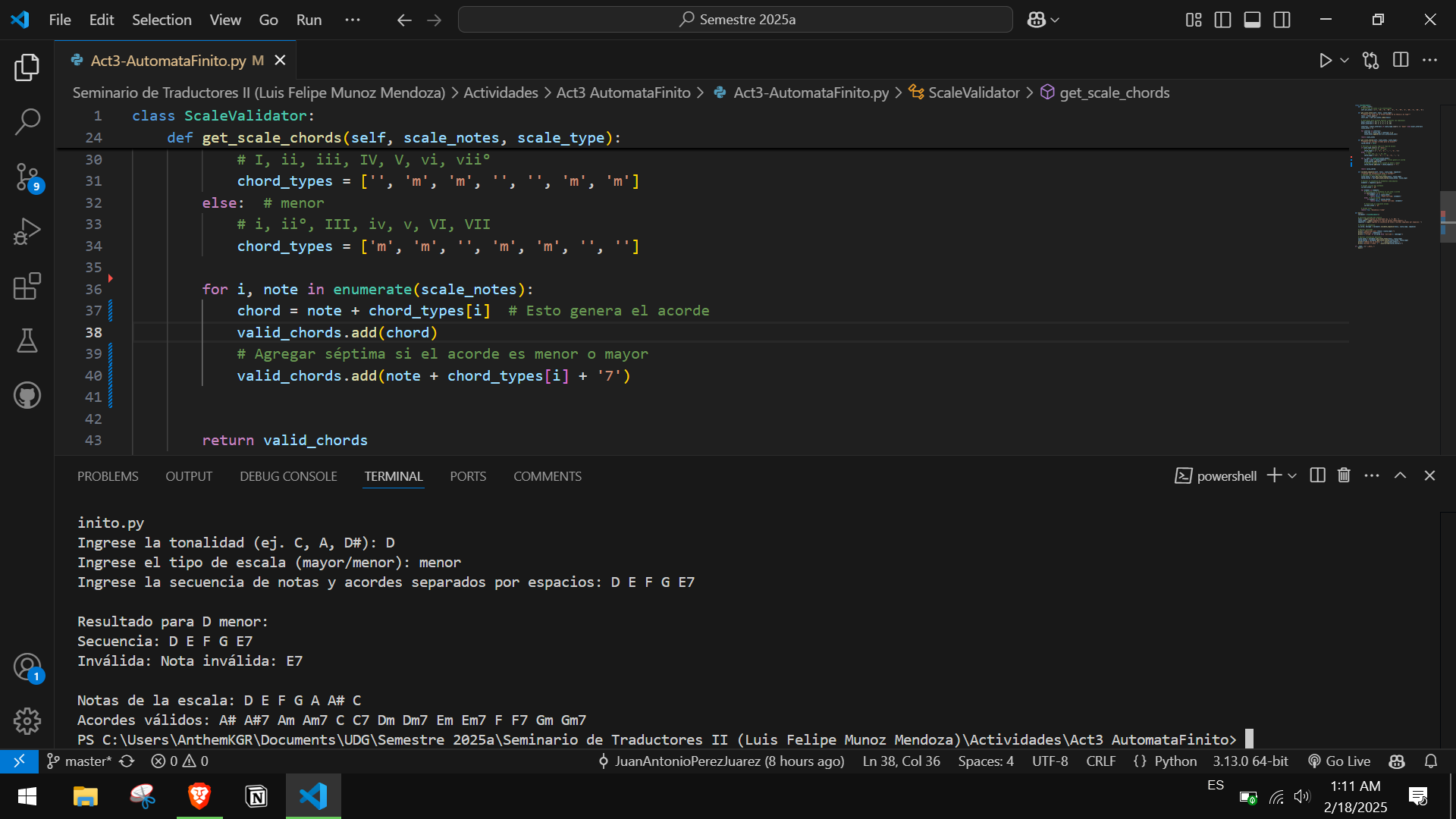
B7: q1 → qError (B7 no es un acorde válido en C mayor)

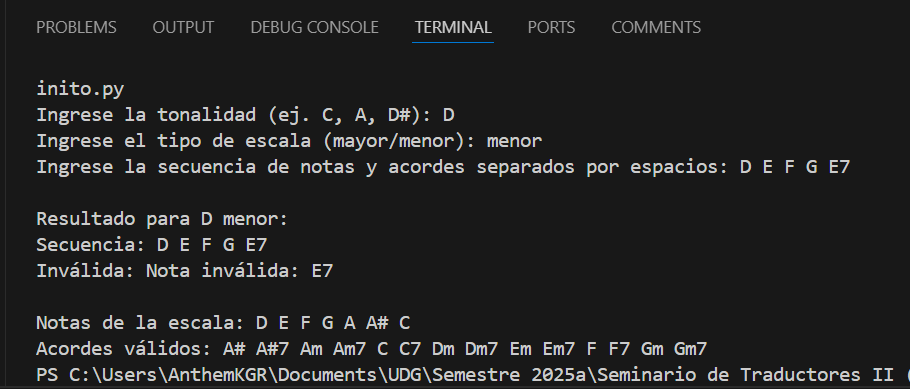
Fin de secuencia en qError

* Secuencia rechazada

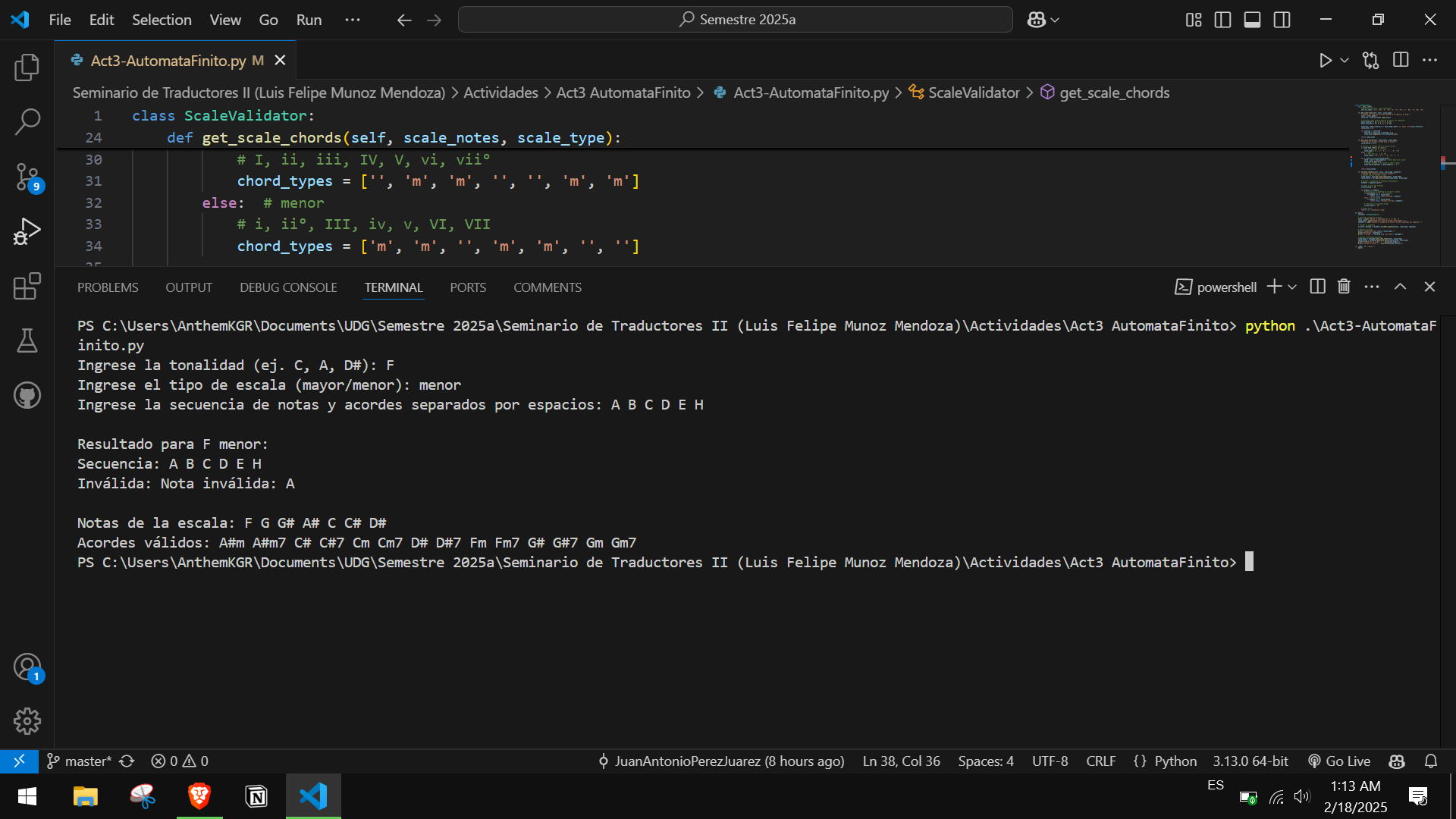
Pruebas:

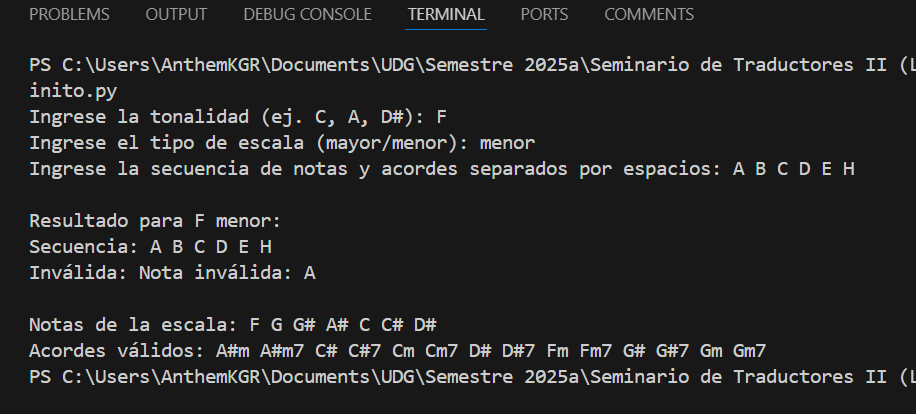
Con una Cadena válida:



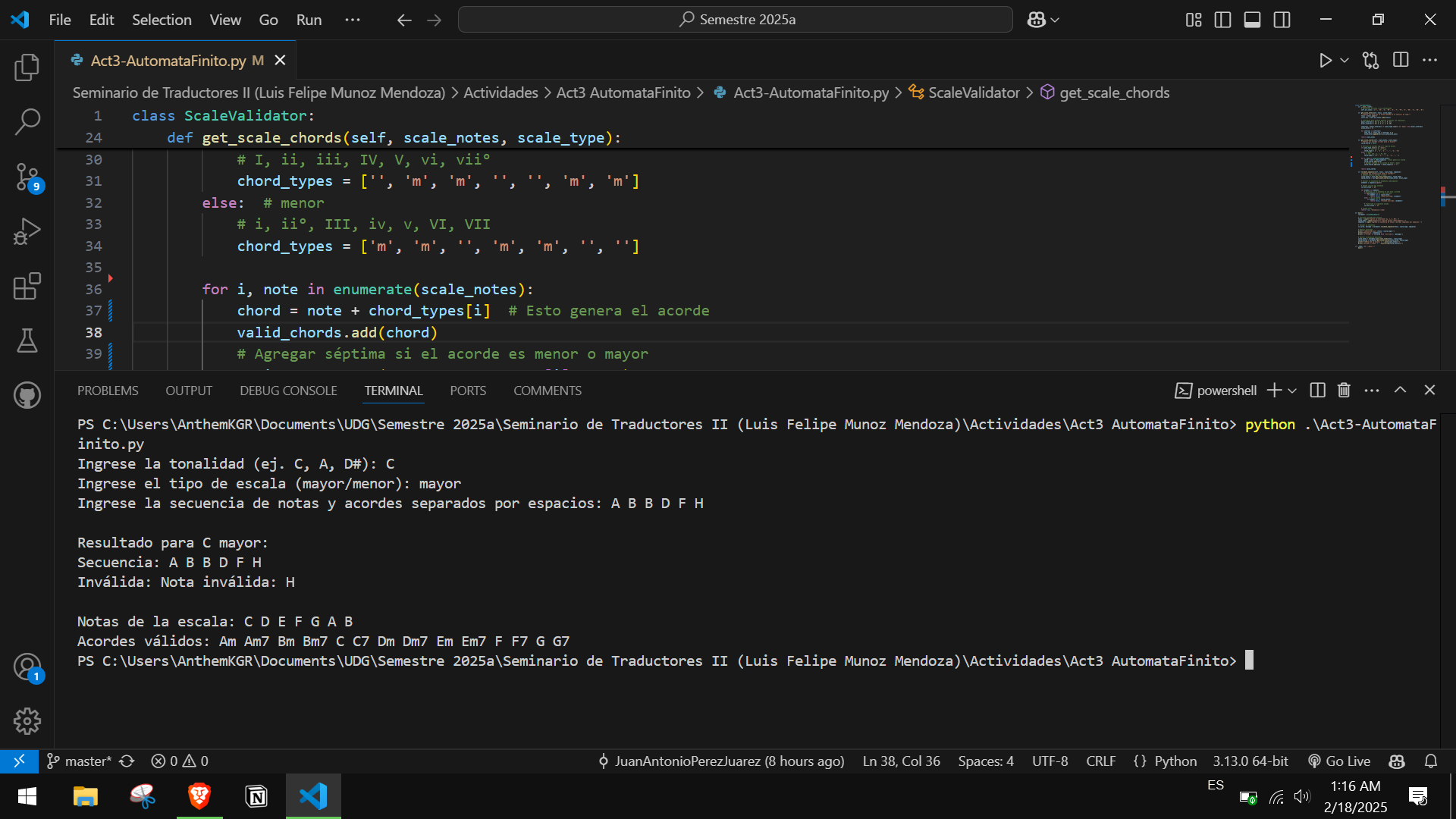


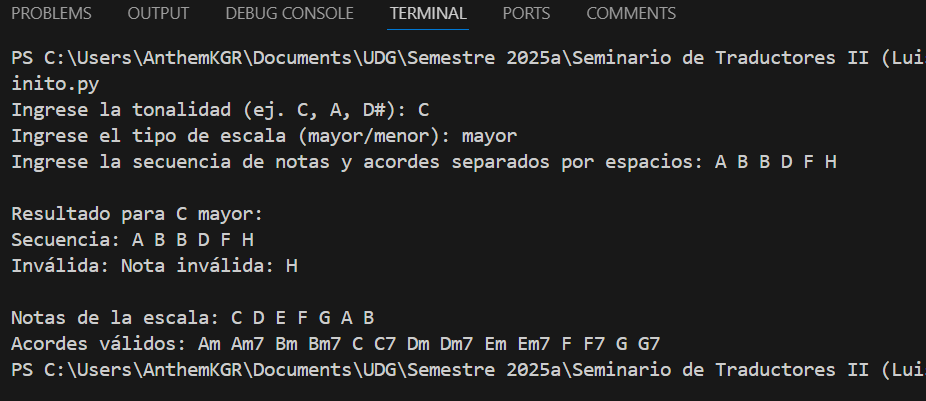
Con una cadena inválida:





Con una cadena que no existe:





Conclusión:

Joder, que buena actividad, el combinar los automatas con las escalas musicales la verdad sí me gustó muchísimo, desde hace tiempo que no tenía la oportunidad de clavarme en un tema tan a profundidad como hoy, que hasta videos de teoría musical me ví.

Y yo se tocar instrumentos, pero gracias a esta actividad pude tener un acercamiento nuevamente a algo que me encanta que es la música.

Al momento de diseñar el autómata finito me ayudé de varios repositorios de internet y de alguna ia que me ayudara para que el diseño fuera eficiente, por qué sigo siendo terrible para estas cosas.

Referencias:

VictorNarov/SimAutomata: Simulador de Autómatas Finitos Deterministas (AFD) y Autómatas Finitos No Deterministas (AFND). (n.d.). GitHub. <https://github.com/VictorNarov/SimAutomata>

MartinCastroAlvarez/automata-python: Implementation of automatas using Python. (n.d.). GitHub. <https://github.com/MartinCastroAlvarez/automata-python>

FcoManueel/deterministic-finite-automaton: Modelador de autómatas finitos deterministas (DFA) implementado en Python. (n.d.). GitHub. <https://github.com/FcoManueel/deterministic-finite-automaton>