

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**

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 mayor

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:

Python

```
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:

Python

```
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:

Python

```
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:

Python

```
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:

Python

```
def main():
    validator = ScaleValidator()
    tonic = input("Ingrese la tonalidad: ")
    scale_type = input("Ingrese el tipo de escala: ")
    sequence = input("Ingrese la secuencia: ")
```

Código:

Python

```
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)

```

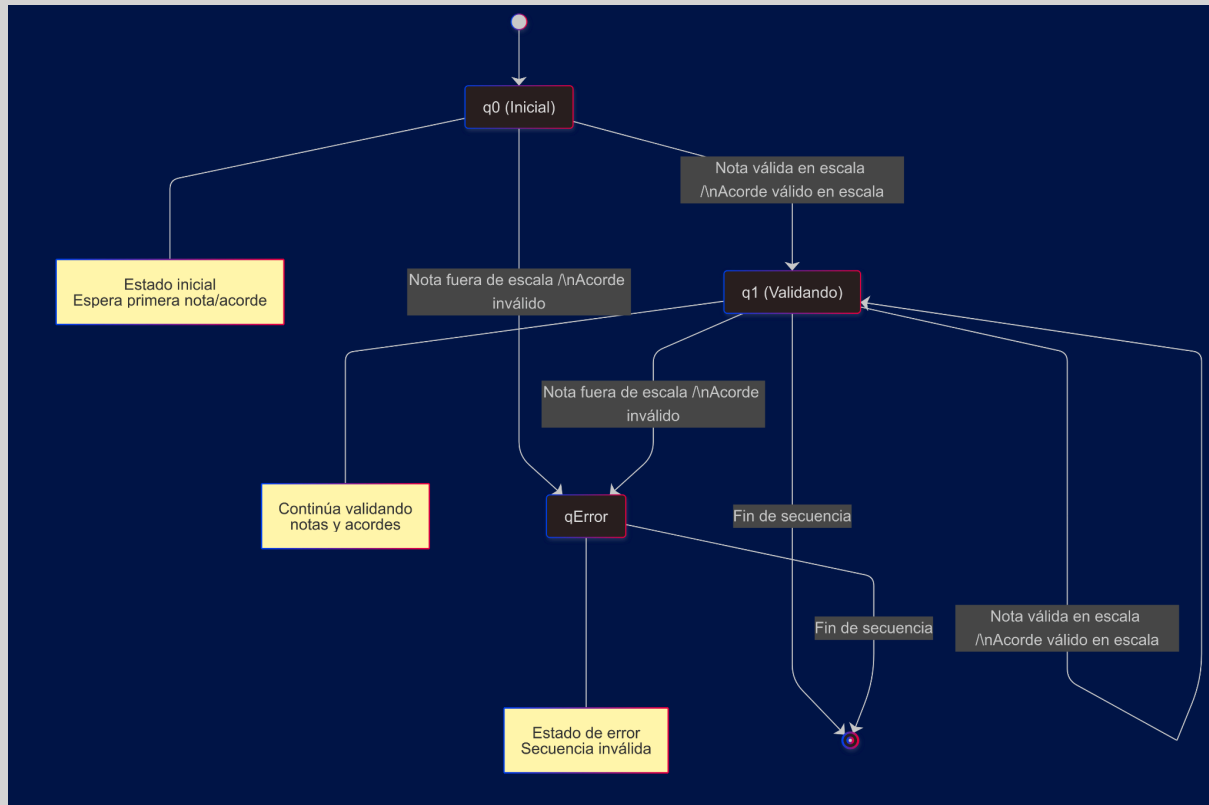
```

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

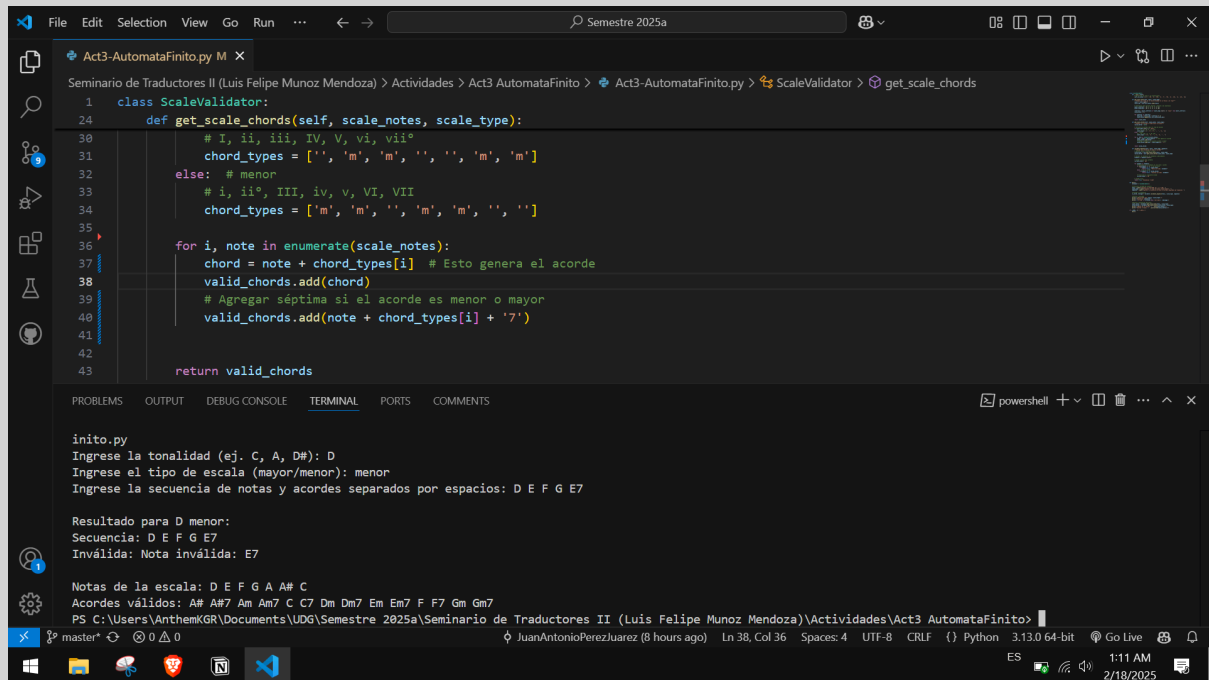
B7: q1 → qError (B7 no es un acorde válido en C mayor)

Fin de secuencia en qError

- Secuencia rechazada

Pruebas:

Con una Cadena válida:



The screenshot shows a VS Code editor with a file named 'Act3-AutomataFinito.py'. The code defines a 'ScaleValidator' class with a 'get_scale_chords' method. The method takes 'scale_notes' and 'scale_type' as input and returns a list of valid chords. The terminal output shows the execution of the script, where the user enters 'D' for the key, 'menor' for the scale type, and 'D E F G E7' for the sequence of notes and chords. The output shows the resulting sequence 'D E F G E7' and a message indicating that 'E7' is an invalid note.

```
1 class ScaleValidator:
24 def get_scale_chords(self, scale_notes, scale_type):
25     # I, ii, iii, IV, V, vi, vii°
26     chord_types = ['I', 'ii', 'iii', 'IV', 'V', 'vi', 'vii°']
27     else: # menor
28         # i, ii°, III, iv, v, VI, VII
29         chord_types = ['i', 'ii°', 'III', 'iv', 'v', 'VI', 'VII']
30
31     for i, note in enumerate(scale_notes):
32         chord = note + chord_types[i] # Esto genera el acorde
33         valid_chords.add(chord)
34         # Agregar séptima si el acorde es menor o mayor
35         valid_chords.add(note + chord_types[i] + '7')
36
37     return valid_chords
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

inito.py
Ingrese la tonalidad (ej. C, A, D#): D
Ingrese el tipo de escala (mayor/menor): menor
Ingrese la secuencia de notas y acordes separados por espacios: D E F G E7

Resultado para D menor:
Secuencia: D E F G E7
Inválida: Nota inválida: E7

Notas de la escala: D E F G A A# C
Acordes válidos: A# A#7 Am Am7 C C7 Dm Dm7 Em Em7 F F7 Gm Gm7
PS C:\Users\AnthemKGR\Documents\UDG\Semestre 2025a\Seminarario de Traductores II (Luis Felipe Munoz Mendoza)\Actividades\Act3 AutomataFinito>

Con una cadena inválida:


```
1 class ScaleValidator:
24     def get_scale_chords(self, scale_notes, scale_type):
30         # I, ii, iii, IV, V, vi, vii°
31         chord_types = ['', 'm', 'm', '', '', 'm', 'm']
32     else: # menor
33         # i, ii°, III, iv, v, VI, VII
34         chord_types = ['m', 'm', '', 'm', 'm', '', '']
```

PS C:\Users\AnthemKGR\Documents\UDG\Semestre 2025a\Seminario de Traductores II (Luis Felipe Munoz Mendoza)\Actividades\Act3 AutomataFinito> python .\Act3-AutomataFinito.py

Ingrese la tonalidad (ej. C, A, D#): F

Ingrese el tipo de escala (mayor/menor): menor

Ingrese la secuencia de notas y acordes separados por espacios: A B C D E H

Resultado para F menor:

Secuencia: A B C D E H

Inválida: Nota inválida: A

Notas de la escala: F G G# A# C C# D#

Acordes válidos: A#m A#m7 C# C#7 Cm Cm7 D# D#7 Fm Fm7 G# G#7 Gm Gm7

PS C:\Users\AnthemKGR\Documents\UDG\Semestre 2025a\Seminario de Traductores II (Luis Felipe Munoz Mendoza)\Actividades\Act3 AutomataFinito>

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
```

PS C:\Users\AnthemKGR\Documents\UDG\Semestre 2025a\Seminario de Traductores II (Luis Felipe Munoz Mendoza)\Actividades\Act3 AutomataFinito> python .\Act3-AutomataFinito.py

Ingrese la tonalidad (ej. C, A, D#): F

Ingrese el tipo de escala (mayor/menor): menor

Ingrese la secuencia de notas y acordes separados por espacios: A B C D E H

Resultado para F menor:

Secuencia: A B C D E H

Inválida: Nota inválida: A

Notas de la escala: F G G# A# C C# D#

Acordes válidos: A#m A#m7 C# C#7 Cm Cm7 D# D#7 Fm Fm7 G# G#7 Gm Gm7

PS C:\Users\AnthemKGR\Documents\UDG\Semestre 2025a\Seminario de Traductores II (Luis Felipe Munoz Mendoza)\Actividades\Act3 AutomataFinito>

Con una cadena que no existe:

The image shows a Visual Studio Code editor window with a file named 'Act3-AutomataFinito.py'. The code defines a `ScaleValidator` class with a `get_scale_chords` method. This method takes `scale_notes` and `scale_type` as input and returns a list of valid chords. The code uses a dictionary to map scale notes to chord types (major, minor, etc.) and iterates through the notes to generate chords. The terminal output shows the program running in a PowerShell prompt, asking for the key signature (C), scale type (major), and scale notes (A B B D F H). It then displays the resulting chords: A B B D F H, and lists the notes of the scale (C D E F G A B) and valid chords (Am Am7 Bm Bm7 C C7 Dm Dm7 Em Em7 F F7 G G7).

```
1 class ScaleValidator:
24     def get_scale_chords(self, scale_notes, scale_type):
30         # I, ii, iii, IV, V, vi, vii°
31         chord_types = ['', 'm', 'm', '', '', 'm', 'm']
32         else: # menor
33             # i, ii°, III, iv, v, VI, VII
34             chord_types = ['m', 'm', '', 'm', 'm', '', '']
35
36         for i, note in enumerate(scale_notes):
37             chord = note + chord_types[i] # Esto genera el acorde
38             valid_chords.add(chord)
39             # Agregar séptima si el acorde es menor o mayor
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

PS C:\Users\AnthemKGR\Documents\UDG\Semestre 2025a\Seminario de Traductores II (Luis Felipe Munoz Mendoza)\Actividades\Act3 AutomataFinito> python .\Act3-AutomataFinito.py

Ingrese la tonalidad (ej. C, A, D#): C

Ingrese el tipo de escala (mayor/menor): mayor

Ingrese la secuencia de notas y acordes separados por espacios: A B B D F H

Resultado para C mayor:

Secuencia: A B B D F H

Inválida: Nota inválida: H

Notas de la escala: C D E F G A B

Acordes válidos: Am Am7 Bm Bm7 C C7 Dm Dm7 Em Em7 F F7 G G7

PS C:\Users\AnthemKGR\Documents\UDG\Semestre 2025a\Seminario de Traductores II (Luis Felipe Munoz Mendoza)\Actividades\Act3 AutomataFinito>

The terminal window shows the execution of the Python script. It prompts the user for the key signature (C), scale type (major), and scale notes (A B B D F H). It then displays the resulting chords: A B B D F H, and lists the notes of the scale (C D E F G A B) and valid chords (Am Am7 Bm Bm7 C C7 Dm Dm7 Em Em7 F F7 G G7).

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
```

PS C:\Users\AnthemKGR\Documents\UDG\Semestre 2025a\Seminario de Traductores II (Luis Felipe Munoz Mendoza)\Actividades\Act3 AutomataFinito> python .\Act3-AutomataFinito.py

Ingrese la tonalidad (ej. C, A, D#): C

Ingrese el tipo de escala (mayor/menor): mayor

Ingrese la secuencia de notas y acordes separados por espacios: A B B D F H

Resultado para C mayor:

Secuencia: A B B D F H

Inválida: Nota inválida: H

Notas de la escala: C D E F G A B

Acordes válidos: Am Am7 Bm Bm7 C C7 Dm Dm7 Em Em7 F F7 G G7

PS C:\Users\AnthemKGR\Documents\UDG\Semestre 2025a\Seminario de Traductores II (Luis Felipe Munoz Mendoza)\Actividades\Act3 AutomataFinito>

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>