Laboratorio 2: Conversión de Expresiones Regulares a AFND-ϵ

Teoría de Lenguajes

1 de abril de 2025

1 Introducción

En este laboratorio, los estudiantes deberán implementar por un lado, una función que indique si una expresión regular cumple con un formato que será establecido más adelante y por otro lado, el algoritmo de conversión de una expresión regular a un autómata finito no determinista con transiciones epsilon (AFND-ϵ), tal como fue presentado en el curso.

En específico, el objetivo del laboratorio es que, a partir de una entrada (una expresión regular) que utilice únicamente los literales `a`, `b` y `c`, el símbolo `e` que representa la tira vacía y los operadores:

- **Concatenación** (representada por `.`)
- **Unión** (representada por `|`)
- **Clausura de Kleene** (representado por `*`)

los estudiantes implementen una función que chequee que la expresión leída solo contiene caracteres válidos y en caso de que lo sea, genere el AFND-ϵ correspondiente a dicha **expresión regular** utilizando el algoritmo visto en el curso. Observar que los paréntesis no son parte del lenguaje definido.

El material contará con una función que compare dos AFND-ϵ para verificar si el resultado generado es correcto.

2 Objetivos

- Implementar la funcion `is_entry_valid` que asegure que se detecten correctamente que las expresiones únicamente contiene caracteres válidos.
- Implementar la función `regex_to_afnde()` que, dada una expresión regular en formato `string` que previamente paso la validación mencionada en el item anterior, genere el AFND-ϵ correspondiente utilizando el algoritmo visto en el curso.
- Integrar la solución en el entorno de pruebas provisto y asegurar que la salida del programa sea **exactamente** la esperada.

3 Modo de Trabajo

- Se les entregará la implementación base de la clase `AFND_e` con funciones para:
  - Agregar estados.
  - Agregar transiciones
  - Establecer el estado inicial.
  - Marcar estados como finales.
- Su tarea consistirá en implementar dos funciones:
  - `is_entry_valid(expr)`: que verifique si la expresión regular dada cumple con las restricciones definidas. Para eso también deberán corroborar que la expresión regular de entrada usa solamente los símbolos válidos ("a","b","c", "e", "*", "|", ".") y no contiene ningún paréntesis u otro símbolo. La salida de esta función será un booleano: "True" si la expresión regular dada en la entrada cumple con estas restricciones; "False" en caso contrario.
  - `regex_to_afnde(expr)`: que, en caso de que la expresión sea válida, genere el AFND-ϵ correspondiente utilizando el algoritmo visto en el curso. Esta función retorna el AFND-ϵ generado, que

contará con el alfabeto `a`, `b`, `c` y usará el símbolo `e` para representar a las transiciones ϵ.

- El laboratorio se debe realizar en grupos de 2 a 4 estudiantes.

4 Descripción de la Tarea

La entrada de su programa será una expresión regular, por ejemplo:

```
a.b*.c
```

Esta expresión representa todas las cadenas que comienzan con el carácter `a`, seguidas por cero o más repeticiones del carácter `b`, y terminan con el carácter `c`.

Desglosada, la expresión contiene:

- El literal `a`.
- Concatenado con el literal `b` al que se le aplica el operador clausura de Kleene (`b*`).
- Concatenado con el literal `c`.

Aplicando el algoritmo del curso el AFND-ϵ que genera es el siguiente:

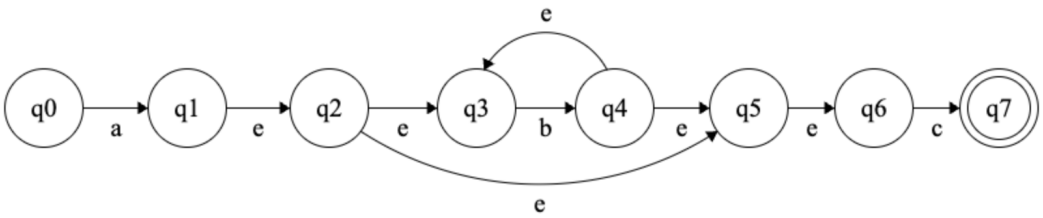

Figure 1: AFND-ϵ generado por la expresión regular `a.b*.c` donde el símbolo `e` se usa para representar las transiciones ϵ

La tarea se divide en los siguientes módulos:

1. Parser de Expresiones Regulares

Un **parser** (o analizador) es un componente que toma como entrada una expresión regular en forma de `string` y la transforma en una estructura que facilita su interpretación y posterior procesamiento. En este laboratorio, las expresiones regulares estarán compuestas **únicamente** por los **literales** (`a`, `b`, `c`) y los **operadores** (`.`, `|`, `*`). No se permiten paréntesis ni ningún otro símbolo adicional. El parser debe respetar la precedencia de los operadores, ya que no habrá agrupamiento explícito mediante paréntesis.

Este paso es necesario para poder aplicar correctamente el algoritmo de construcción del AFND-ϵ, ya que permite entender la estructura interna de la expresión regular y evaluar si la entrada es valida para aplicar la función `regex_to_afnde(expr)` o no.

2. Construcción del AFND-ϵ

Utilizando la clase `AFND_e` proporcionada, deberán construir el autómata completo a partir de autómatas más simples, también llamados **autómatas parciales**, que representan los componentes individuales de la expresión regular.

En particular, deberán:

- Crear un autómata parcial para cada literal (`a`, `b`, `c`).
- Crear un autómata parcial para la tira vacía `e`, en caso de existir.
- Aplicar los operadores de concatenación, unión y clausura de Kleene para combinar dichos autómatas parciales y formar el AFND-ϵ final.

3. Integración y Pruebas

Se les entregará un script de pruebas (`test.py`) que:

- Lee un conjunto de expresiones regulares de entrada.

- Llama a la función `is_entry_valid()` para determinar si la expresión únicamente contiene caracteres válidos.
- Si la expresión es válida, llama a la función `regex_to_afnde()` para generar el AFND-ϵ correspondiente.
- Si la expresión no es válida, el programa de pruebas generará la salida `Entrada no valida`.
- Compara el resultado (ya sea un AFND-ϵ generado o el mensaje de error) con la salida esperada utilizando la función de comparación o una validación directa de texto, según corresponda.
- Muestra un mensaje indicando si la solución es correcta.

5 Ejemplo de Uso

Dada la expresión regular:

```
a.b*.c
```

la salida esperada (desplegada por el script de pruebas) deberá indicar que el AFND-ϵ generado es **igual** al AFND-ϵ de referencia. Se considera que dos AFND-ϵ son iguales si la estructura de ambos ANFD-ϵ es la misma, sin tener en cuenta los nombres de sus estados. Se evaluará la correcta construcción del autómata a través de la función de comparación proporcionada.

6 Entregables

- El archivo `regex_to_afnde.py` con la implementación de la función `regex_to_afnde(expr)`
- El archivo `is_entry_valid.py` con la implementación de la función `is_entry_valid(expr)`
- El archivo `integrantes.txt` con los datos de los miembros del grupo.
- Se debe mantener el formato y nombres de archivos indicados.

Ejemplo de archivo integrantes.txt:

```
3
7123456, ApellidoA1 ApellidoA2, NombreA1 NombreA2
8654321, ApellidoB1 ApellidoB2, NombreA1 NombreB2
9876543, ApellidoC1 ApellidoC2, NombreC1 NombreC2
```

7 Fecha de Entrega

La entrega se realizará a través de una tarea en EVA antes de las **23:59 del 24 de abril**.

8 Referencias y Herramientas

- Python 3 (se recomienda Python 3.13 o superior) [1].
- Documentación de Python sobre expresiones regulares: [2].
- Material teórico sobre la construcción de autómatas en el eva [3].

Contenido del Archivo `lab.zip`

El archivo `lab.zip` incluirá:

- La implementación base de la clase `AFND_e`.
- Archivos de entrada con expresiones regulares y los correspondientes AFND-ϵ de referencia.
- El script `test.py` para la ejecución de pruebas.

1. Python 3 Documentation
2. Módulo `re` de Python
3. Material sobre pasaje de e.r. a AFND-ϵ