Prácticas de Autómatas y Lenguajes. Curso 2025/26

Práctica 2: Autómatas finitos

Duración: 6 semanas

Fecha de entrega: Semana del 3 de noviembre, cada grupo antes de su clase

Peso: 45% de la nota de prácticas

Descripción del enunciado:

En esta práctica trabajaremos con autómatas finitos (AF), tanto deterministas (AFD) como no deterministas (AFnD). Se programará el algoritmo de aceptación de cadenas por parte de dichos autómatas y se implementarán los algoritmos de equivalencia y minimización de AF.

Los objetivos de esta práctica serán:

- Entender y saber implementar un AF, el proceso y aceptación de cadenas.
- Entender la equivalencia entre expresiones regulares (ER) y AF. Saber implementar la conversión de una ER a un AFnD equivalente.
- Entender la equivalencia entre AFnD y AFD. Saber implementar la transformación de un AFnD en un AFD equivalente.
- Entender y saber implementar la minimización de un AFD en su equivalente mínimo y que no presente estados inaccesibles.

Se recuerda que los objetivos planteados deben ser adquiridos por **ambos** miembros de la pareja. Se realizará un control para comprobarlo y, si este no es el caso en algún miembro de la pareja, se le suspenderá la práctica con una evaluación de NO APTO.

Definición de un AF:

Un autómata finito (AF) es una máquina abstracta muy utilizada en la teoría de la computación para reconocer lenguajes formales. Se define mediante la siguiente quíntupla:

$$A = (Q, \Sigma, \delta, q0, F)$$

Donde:

- Q: es un conjunto finito de **estados**.
- Σ: es el **alfabeto**, es decir, el conjunto de símbolos que el AF puede procesar.
- δ: es la función de transición $\delta : Q \times \Sigma \to Q$ que define el comportamiento del autómata, es decir, el **conjunto de transiciones** del autómata.
- $q0$: es el estado inicial $q0 \in Q$.

- F: es el **conjunto de estados finales** $F \subseteq Q$.

Por tanto, a lo largo de la práctica vamos a implementar la clase **FiniteAutomaton** en Python para construir el autómata completo.

Descripción de los ficheros suministrados:

En esta práctica se proporcionan los siguientes ficheros:

- **automaton.py**: Contiene la clase que engloba todo el concepto de un AF. Esta clase incluye también todos los métodos necesarios para comprobar si una cadena pertenece o no al lenguaje, la transformación de un AFnD a un AFD y la minimización de un AFD.
- **re_parser.py**: Contiene una clase que se utiliza para convertir una ER (usando la notación vista en clase) en un AF.
- **utils.py**: Contiene funciones de utilidad para trabajar con AF. El estudiante **no debe modificar** ni entregar este fichero.

Además de ello, se proporcionan varios test en formato unittest, útiles para comprobar que la funcionalidad implementada es correcta. El estudiante podrá añadir los test que considere necesarios.

Nota: Los test proporcionados son referencias para el desarrollo del proyecto, no tienen una relación directa con los mínimos requerimientos para aprobar la práctica.

Recomendación: Para implementar el AF en automaton.py se sugiere implementar la matriz de transiciones como un **diccionario de diccionarios** para evitar una matriz *sparse*, es decir, evitar que haya muchos "huecos vacíos". Así, el acceso a las transiciones desde el estado `start_state` con el símbolo `symbol` debería hacerse como:

```
end_states = self.transitions[start_state][symbol]
```

Nota: En automaton.py se proporciona, además, el método **draw** ya implementado, que servirá para visualizar el autómata que se está generando y poder depurar más fácilmente.

Ejercicio 1: Construcción del autómata y evaluación (3.0 puntos):

Se debe implementar la clase **FiniteAutomaton** del fichero automaton.py, sin modificar aún las funciones `to_deterministic` y `to_minimized`. Para la evaluación del código, se debe superar el **test_evaluator**.

Ejercicio 2: Conversión de ER en autómatas finitos (1.5 puntos):

Se debe implementar la clase **REParser** del fichero re_parser.py. En este ejercicio se demostrará de forma práctica que para toda ER existe un AF equivalente. Los estudiantes deben discutir en clase la conversión en autómata de cada elemento de una ER (operaciones y símbolos). Para la evaluación del código, se debe superar el **test_re_parser**. Concretamente, se deberán implementar los siguientes métodos:

- _create_automaton_empty: Crea el autómata que acepta el lenguaje vacío.
- _create_automaton_lambda: Crea el autómata que acepta la cadena vacía.
- _create_automaton_symbol: Crea el autómata que acepta un símbolo.
- _create_automaton_star: Crea el autómata para calcular la estrella de Kleene de un autómata dado.
- _create_automaton_union: Crea el autómata para calcular la unión de dos autómatas dados.
- _create_automaton_concat: Crea el autómata para calcular la concatenación de dos autómatas dados.

Ejercicio 3: Conversión de AFnD en AFD (2.5 puntos):

El profesor explicará en clase el algoritmo de conversión de AFnD con transiciones lambda a AFD. Los estudiantes deben implementar el método **to_deterministic** de la clase FiniteAutomaton, que realiza dicha conversión. Para la evaluación, se debe superar el **test_to_deterministic**.

Ejercicio 4: Minimización de AFD (3.0 puntos):

El profesor explicará en clase el algoritmo de minimización de AFD. Los estudiantes deben implementar el método **to_minimized** de la clase FiniteAutomaton, que realiza dicha minimización. Para la evaluación, se debe superar el **test_minimization**.

Planificación de la práctica:

Ejercicio 1	Semanas 1 y 2
Ejercicio 2	Semana 3
Ejercicio 3	Semana 4
Ejercicio 4	Semana 5
Dudas, conclusiones, etc.	Semana 6