Prácticas de Autómatas y Lenguajes. Curso 2024/25

Práctica 1: Autómatas finitos

Duración: 6 semanas.

Fecha de entrega: Semana del 11 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 demostrarán, mediante su implementación práctica, los algoritmos de equivalencia y minimización de AF.

Los objetivos de esta práctica serán:

- Entender y saber implementar el proceso y aceptación de símbolos y cadenas por parte de AFnD, de los que los deterministas son un caso particular.
- 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, que no presente estados inaccesibles ni equivalentes.

Además, se pretende extender el conocimiento en el lenguaje de programación Python.

Se recuerda que los objetivos planteados deben ser adquiridos por **ambos** miembros de la pareja. En caso de realizarse una prueba y comprobar que este no es el caso, se podría suspender la práctica y exigir la entrega individual en la modalidad no presencial.

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 $\delta : Q \times \Sigma \rightarrow Q$ 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 las siguientes clases de Python para construir el autómata completo:

- **State**: Clase que define un estado del autómata.
- **Transitions**: Clase que define todo lo relacionado con las transiciones del autómata.
- **FiniteAutomaton**: Clase que define todo lo relacionado con el autómata completo.

Descripción de los ficheros suministrados:

En esta práctica se proporcionan los siguientes ficheros:

- **state.py**: Contiene todo lo relacionado con los estados del autómata. Un estado estará definido por un nombre y si el estado es final o no.
- **transitions.py**: Contiene todo lo relacionado con las transiciones del autómata. En la implementación de la práctica, se ha decidido implementar el conjunto de transiciones como un diccionario de diccionarios para poder hacer la siguiente búsqueda rápida:

  transitions[start_state][symbol] → {end_state_1, end_state_2, …}

- **automaton.py**: Contiene la clase que engloba todo el concepto de un AF, incluyendo sus estados, sus transiciones y el alfabeto. 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. Para ello se deben añadir más test por parte del estudiante.

Ejercicio 0: Comprensión del código proporcionado (0.0 puntos):

Como primer paso, el estudiante debe leer y entender las clases State, Transitions y FiniteAutomaton, así como la lógica que hay detrás del diseño del autómata. Como se puede leer en los ficheros suministrados, las funciones están documentadas detallando su funcionalidad, por tanto, no se volverán a redactar en este enunciado.

Ejercicio 1: Evaluación del autómata (3.0 puntos):

Se debe superar el **test_evaluator**. Se recomienda a los estudiantes que intenten también crear sus propios tests para comprobar todos los casos posibles. Para ello, se deben implementar los siguientes métodos:

- Todos los métodos de **transitions.py**, que son los siguientes:
    - state_has_transitions
    - state_get_symbols
    - state_has_any_transition_with_symbol
    - state_has_transition_to
    - goes_to
    - add_transition
- Los métodos de **automaton.py** en el bloque BEGIN-END relacionados con el procesamiento de cadenas, que son los siguientes:
    - process_symbol
    - accepts
    - _complete_lambdas

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

Se debe superar el **test_re_parser**. En este ejercicio se demostrará de forma práctica que para toda ER existe un AF equivalente. Los estudiantes deben discutir en clase cómo se realizaría la conversión en autómata de cada uno de los elementos que conforman una ER (operaciones y símbolos). Una vez puesto en común, los estudiantes deberán implementar en la clase REParser del fichero **re_parser.py** 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 deterministas (2.5 puntos):

Se debe superar el **test_to_deterministic**. 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.

Ejercicio 4: Minimización de autómatas finitos deterministas (3.0 puntos):

Se debe superar el **test_minimization**. 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.

Ejercicio opcional (0.5 puntos a añadir sobre la nota final de prácticas):

Diseña un conjunto exhaustivo de tests para generar a partir de las expresiones regulares de la práctica 0 los autómatas finitos deterministas mínimos. Comprueba con los test de la práctica 0 que se ha realizado correctamente.

Planificación de la práctica:

Ejercicio 1	Semanas 1 y 2
Ejercicio 2	Semana 2
Ejercicio 3	Semanas 3 y 4
Ejercicio 4	Semanas 5 y 6