Prácticas de Autómatas y Lenguajes. Curso 2023/24

Práctica 1: Autómatas finitos

Duración: 6 semanas.

Entrega: Semana del 13 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, tanto deterministas como no deterministas. 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 autómatas finitos.

Los objetivos de esta práctica serán:

- Entender y saber implementar el proceso y aceptación de símbolos y cadenas por parte de autómatas finitos no deterministas, de los que los deterministas son un caso particular.
- Entender la equivalencia entre expresiones regulares y autómatas finitos. Saber implementar la conversión de una expresión regular a un autómata finito no determinista equivalente.
- Entender la equivalencia entre autómatas finitos no deterministas y autómatas finitos deterministas. Saber implementar la transformación de un autómata finito no determinista en un autómata finito determinista equivalente.
- Entender y saber implementar la minimización de un autómata finito determinista en uno equivalente que no presente estados inaccesibles ni equivalentes.

Además, se pretenden lograr también los siguientes objetivos de programación:

- Entender y poder implementar código en el lenguaje de programación Python.
- Entender y saber implementar tests usando el módulo unittest, similar a Junit.

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.

Descripción de los ficheros suministrados:

En esta práctica se proporcionan los siguientes ficheros:

- automaton.py: Contiene las clases básicas para definir un autómata finito no determinista, incluyendo sus estados y transiciones. El estudiante deberá añadir los métodos solicitados en la clase Transitions. Opcionalmente podrá añadir nuevos atributos y métodos en las clases State, Transitions y FiniteAutomaton, pero siempre respetando la lógica de las clases propuestas.
- dfa.py: Contiene una única clase DeterministicFiniteAutomaton que hereda de FiniteAutomaton y contiene los métodos necesarios para crear un automata finito determinista y mínimo a partir de un automata finito no determinista.
- automaton_evaluator.py: Contiene una única clase que se usa para evaluar el autómata y comprobar si se aceptan o rechazan símbolos y cadenas. El estudiante deberá completar los métodos que faltan. Opcionalmente podrá añadir nuevos atributos y métodos, pero siempre respetando la interfaz suministrada.
- re_parser.py: Contiene una clase que se utiliza para convertir una expresión regular (usando la notación vista en clase) en un autómata finito. El estudiante deberá completar los métodos que faltan. Opcionalmente podrá añadir nuevos atributos y métodos, pero siempre respetando la interfaz suministrada.
- utils.py: Contiene funciones de utilidad para trabajar con autómatas finitos. El estudiante no debe modificar ni entregar este fichero.

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

Ejercicio 1: Evaluación del autómata (2.5 puntos):

Como primer paso el estudiante debe leer y entender las clases State, Transition, FiniteAutomaton y FiniteAutomatonEvaluator, así como los atributos y métodos que heredan de sus respectivas superclases.

a. Se debe completar el fichero automaton.py para que la clase Transitions esté completa. Para ello se deben implementar los siguientes métodos:
    - has_transition: Comprueba si existe en el diccionario de transiciones una transición dado un estado inicial y un símbolo. Devuelve True o False.
    - has_transition_to: Comprueba si existe en el diccionario de transitiones una transición dado un estado inicial, un símbolo y un estado final. Devuelve True o False.

- get_transition: Devuelve un set de estados correspondiente a un estado inicial y un simbolo, si el estado-simbolo pertenece al diccionario de transiciones.

b. Se debe completar el fichero automaton_evaluator.py para permitir la validación de cadenas usando un autómata finito. Se deben implementar los siguientes métodos de la clase FiniteAutomatonEvaluator:

- process_symbol: Procesa un símbolo de la cadena (y cualquier número de transiciones lambdas inmediatamente después, mediante la llamada a _complete_lambdas).
- _complete_lambdas: Añade al conjunto de estados pasado como argumento todos los estados que sean alcanzables mediante un número arbitrario de transiciones lambda.
- is_accepting: Indica si la cadena que se ha procesado hasta el momento se acepta o no.

Al completar esta función, los tests en test_evaluator deberían pasar. Se recomienda a los estudiantes que intenten también crear sus propios tests para comprobar todos los casos.

Nota: para ejecutar test_evaluator es necesario incluir la ruta de automata (el cual importa) de la siguiente forma:

export PYTHONPATH=$PYTHONPATH:.

Ejemplo:

```
/P1$ export PYTHONPATH=$PYTHONPATH:.
/P1$ python automata/tests/test_evaluator.py
```

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

En este ejercicio se demostrará de forma práctica que para toda expresión regular existe un autómata finito equivalente.

Los estudiantes deben discutir en clase cómo se realizaría la conversión en autómata de cada uno de los elementos que conforman una expresión regular (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.