Universidad Nacional Autónoma de México

Facultad de Ciencias

Autómatas y Lenguajes Formales

Tarea examen 1: parte practica

Cadenas, lenguajes y autómatas finitos

Profesores:

M. en C. Fernando Abigail Galicia Mendoza
Luis Mario Escobar Rosales

Fecha de entrega
Viernes, 13 de septiembre del 2024.

Valor de la práctica: 3 puntos.

Objetivo: El objetivo de esta práctica es implementar un programa que verifique que una dirección de correo electrónico dado como entrada esté bien escrita. Para este fin deberás implementar un programa que reciba solamente una cadena de texto, e.g. `foo@fooooo.com`, y decida si dicha cadena de texto está bien escrita utilizando expresiones regulares y autómatas finitos no deterministas con transiciones épsilon (ε-AFNs).

Lenguaje de programación: Esta práctica tiene dos versiones, una implementación en Haskell y otra en Python. Puedes elegir solamente una de estas opciones.

En el archivo anexo a esta práctica encontrarás lo siguiente:

1. Una estructura de datos, llamada Symbol, que representa un símbolo. Un símbolo es un carácter ASCII o el símbolo épsilon (ε).
2. Una estructura de datos, llamada Alphabet, que representa un alfabeto.
3. Una estructura de datos, llamada Word, que representa una cadena.
4. Una estructura de datos, llamada Language, que representa un lenguaje.
5. Una estructura de datos, llamada RE, que representa una expresión regular.
6. Una estructura de datos, llamada DFA, que representa un autómata finito determinista.
7. Una estructura de datos, llamada NFAE, que representa un autómata finito no determinista con transiciones épsilon.
8. Una función, llamada `powerset`, que devuelve el conjunto potencia de un conjunto de estados.

Para poder lograr el objetivo de la práctica implementa las funciones que se te piden a continuación.
Símbolos y cadenas

1. `first`. La cual recibe una cadena y devuelve el símbolo más a la derecha. En caso de que reciba la cadena vacía, devolver ε.
2. `prefix`. La cual recibe una cadena y devuelve la subcadena sin el símbolo más a la derecha de la cadena original. Análogo a `first`, devuelve ε en caso de que la cadena de entrada sea vacía.
3. `membership`. La cual recibe una cadena y un lenguaje, verifica si dicha cadena pertenece al lenguaje dado como entrada.

ε-AFNs

1. `closure`. La cual recibe un ε-AFN y un estado de dicho autómata, y devuelve la cerradura épsilon del estado dado como entrada.
2. `accept`. La cual recibe una cadena y un ε-AFN, y decide si la cadena de entrada es aceptada por el ε-AFN.
3. `empty`. La cual devuelve el ε-AFN que acepta el lenguaje vacío.
4. `epsilon`. La cual devuelve el ε-AFN que acepta a ε.
5. `symbol`. La cual recibe un símbolo a, y devuelve el ε-AFN que acepta la cadena a.

6. `union`. La cual recibe dos ε-AFN, sean E_1 y E_2, y devuelve la unión de E_1 y E_2 utilizando transiciones ε.
7. `concat`. La cual recibe dos ε-AFN, sean E_1 y E_2, y devuelve la concatenación de E_1 y E_2 utilizando transiciones ε.
8. `kleene`. La cual recibe un ε-AFN, sea E, y devuelve la cerradura de Kleene aplicada a E utilizando transiciones ε.

Expresiones regulares

1. `toEAFN`. La cual recibe una expresión regular e, y devuelve el ε-AFN E tal que $\mathcal{L}(e) = \mathcal{L}(E)$.

Correo electrónico

1. `verify`. La cual recibe una cadena de texto, e.g. `foo@foooo.com`, y decide si la cadena de entrada es una dirección de correo electrónico bien escrita. Observa que la cadena de texto es de tipo distinto a las cadenas de tipo Word.

Punto extra: AFDs

1. `toAFD`. La cual recibe un ε-AFN E, y devuelve el AFD D tal que $\mathcal{L}(E) = \mathcal{L}(D)$.