Prácticas Programación Modelos Computación

1 Introducción

Se trata de hacer programas en Python de algunos algoritmos y procedimientos vistos en clase. Se seguirá (con variaciones) el modelo del libro:

J. MacCormick (2018) *What Can Be Computed? A Practical Guide to the Theory of Computation.* Princeton University Press.

Se pueden descargar los programas asociados a este libro de su página web `https://whatcanbecomputed.com/`. También se puede usar el programa afd.py que se proporciona y que será la base de los cálculos.

2 Ejercicios que se pueden realizar

2.1 Complementar afd.py

Se pueden realizar algunas funciones complementarias como:

- `checkDeterminist`: Comprobar que el autómata es realmente determinista.
- `organizeTable`: Poner una transición para cada estado y cada símbolo. Eliminar símbolos comodín.
- `addError`: Completar un AFD añadiendo un estado de error para las transiciones no definidas.

- `toDeterminist`: Transformar un autómata que puede ser no-determinista (incluso con transiciones nulas, representadas por '-') en uno determinista.
- `testFinite`: Comprueba si el lenguaje aceptado es finito.
- `testEmpty`: Comprueba si el lenguaje aceptado es vacío.

2.2 Realizar una Clase para las Gramáticas Tipo 3

Las gramáticas de tipo 3 se representarían de la siguiente forma:

- Una línea con las variables (símbolos de los que el primero es la variable inicial)
- Una línea de símbolos terminales
- Una línea for cada producción con la siguiente estructura: 'A `->` α', la cadena vacía se representa por '-'

Se deben de implementar las siguientes funciones:

- `readFromFile`: Leer de un fichero.
- `writeToFile`: Salvar en un fichero.
- `checkRegular`: Comprueba que es una gramática regular (lineal por la izquierda o por la derecha).
- `toAFND`: Transforma la gramática en una AFND (posiblemente con transiciones nulas).
- `fromAFND`: Coge una autómata finito (determinista o no) y lo transforma en gramática.

2.3 Realizar una Clase para las Expresiones Regulares

Esta clase debe de gestionar expresiones regulares. Estas se leen en un formato en el que se usan solo las operaciones básicas. Los símbolos especiales ('()*+') tienen el mismo significado que el visto en clase. La concatenación se deja implícita. La cadena vacía se representa mediante '-'. El símbolo vacío se represetna por '#'.

Se deben implementar las siguientes funciones:

- `readFromFile`: Leer de un fichero.
- `writeToFile`: Salvar en un fichero.
- `toAFND`: Transforma una expresión regular en una AFND (posiblemente con transiciones nulas).
- `fromAFND`: Coge una autómata finito (determinista o no) y lo transforma en una expresión regular.

2.4 Operaciones con Autómatas

Se deben implementar las siguientes funciones:

- `intersection:` Calcula el AFD que acepta la intersección de los lenguajes de dos ADFs.
- `union:` Calcula el AFD que acepta la unión de los lenguajes de dos ADFs.
- `complementary:` Calcula el AFD que acepta el complementario del lenguaje de un AFD dado.
- `minimize:` Calcula un AFD minimal que acepte el mismo lenguaje.

2.5 Gramáticas Independientes del Contexto

Las gramáticas independientes del contexto se representarán como las regulares.

Habrá que implementar las siguientes funciones:

- `readFromFile`: Leer de un fichero.
- `writeToFile`: Salvar en un fichero.
- `simplify`: Eliminar producciones inútiles.
- `removeEmpty`: Eliminar restricciones nulas.
- `removeUnitary`: Eliminar restricciones unitarias.
- `toChomsky`: Pasar a forma normal de Chomsky.
- `to Greibach`: Pasar a forma normal de Greibach.

2.6 Autómatas con Pila

Los autómatas con pila se representa de forma similar a los autómatas finitos deterministas, con la diferencia de que una transición se escribe de la forma:

qi `->` qj: entrada, pila; pilanuevo

Si la fila de estados finales está vacía, se supone que se acepta por pila vacía.

Se pueden usar comodines como '~' tanto para la entrada como para la pila y la substitución en la pila (pilanuevo). Cuando se usa '~'en pila nuevo, se supone que la pila se deja tal y como estaba.

e implementar las siguientes funciones:

- `readFromFile`: Leer de un fichero.
- `writeToFile`: Salvar en un fichero.
- `checkDeterministic:` Comprueba si el autómata es determinístico.
- `run`: ejecuta el autómata para una palabra de entrada dada.
- `toGrammar:` transformar un autómata con pila en una gramática independiente del contexto

2.7 Algoritmos de Parsing

Determinar si una palabra es generada por una gramática independiente del contexto con los siguientes algoritmos:

- `checkGeneral`: Algoritmo de búsqueda genérico
- `checkGreibach`: Algoritmo de búsqueda para gramáticas en forma normal de Greibach
- `checkCYK`: Algoritmo de Cocke-Younger-Kasami
- `checkEarly`: Algoritmo de Early