DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 1

Jueves 2 de julio de 2015 — Teoría de Lenguajes

Integrante	LU	Correo electrónico
Aleman, Damián Eliel	377/10	damian_8591@hotmail.com
Gauna, Claudio Andrés	733/06	gauna_claudio@yahoo.com.ar
Vargas Telles, Matías	318/12	mvt208@hotmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellon I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autonoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

1. Introducción

Para el presente trabajo se nos solicitó la implementación de diversos algoritmos para autómatas finitos determinísticos.

Este informe tiene como objetivo acompañar el código entregado para la resolución de los enunciados planteados, permitiendo así su mejor comprensión.

En las siguientes secciones describiremos la representación elegida para los autómata y luego agregaremos documentación sobre los algoritmos implementados.

2. Clase automaton

Representamos un automaton con una clase de java. La misma contiene (como atributos privados) un conjunto de caracteres σ, un diccionario de transiciones *transitions*, un arreglo de estados *states*, un estado inicial *initialState* y un conjunto de estados finales *finalStates*.
La interfaz pública permite:

- Crear un automaton
- Obtener el iésimo estado
- Obtener las transiciones
- Consultar si un estado es final
- Preguntar por una transición particular (desde un estado mediante un caracter)
- Completa un automaton con transiciones lambda.
- Obtener los estados (iniciales y finales) y saber si un string es aceptado.

2.1. Clase Automaton Operations

La interfaz pública permite :

- Crear el complemento de otro automata
- Crear el automata minimizado
- Crear la intersección de dos automatas
- Crear la unión de dos automatas
- Crear la concatenación de dos automatas
- Computa si dos automatas son equivalentes
- Determinizar un automata

2.2. Clase Automaton reader

En esta clase leemos un archivo que representa un automata con el formato del enunciado y devolvemos un automaton.

2.3. Clase Automaton writer

En esta clase escribimos el automata que tenemos como entrada y lo escribimos en un archivo. También se encarga de crear el archivo .dot para graficar automatas.

2.4. Clase Regular expression reader

En esta clase leemos un archivo que representa una expresión regular con el formato del enunciado y devolvemos un automaton.

2.5. Clase State

Mediante esta clase representamos al estado, que se crea con un nombre

2.6. Clase Transition

Mediante esta clase representamos una transición entre estados. Podemos calcular el origen de la transición, así como también su destino y símbolo correspondiente.

3. Algoritmos principales implementados

En esta sección describiremos brevemente los algoritmos empleados.

3.1. Minimización de AFD

Este algoritmo minimiza el automata clasificando a los estados en clases de estados indistinguibles. Se dice que dos estados son indistinguibles cuando para toda transición con origen en los dos estados tiene el mismo destino(o llegan a estados que son indistinguibles, es decir que pertenecen a la misma clase de equivalencia).

3.2. Decidir si una palabra pertenece al lenguaje

Esto lo implementamos en la función reconoce (en la clase Automaton). Básicamente, el algoritmo comienza con el estado inicial y va recorriendo desde allí mediante las transiciones, consumiendo los símbolos que forman la palabra. Si se consumieron todos los símbolos y se logró llegar a un estado final, la palabra pertenece al lenguaje, en caso contrario no pertenece al lenguaje.

3.3. Determinización de AFND

Para un mejor siguemiento de este algoritmo recomendamos tener a mano su pseudocódigo[1]. Observamos que en este algoritmo la explosión combinatoria puede generar autómatas con una cantidad de estados notoriamente mayor que los originales, lo cual prolonga considerablemente los tiempos de ejecución y el uso de memoria. Por ejemplo:

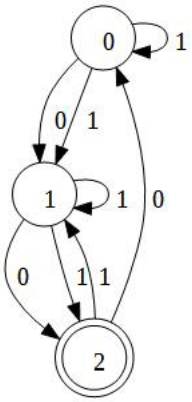

Figura 1: AFND original con 3 estados

[1] `http://web.cecs.pdx.edu/~harry/compilers/slides/LexicalPart3.pdf`

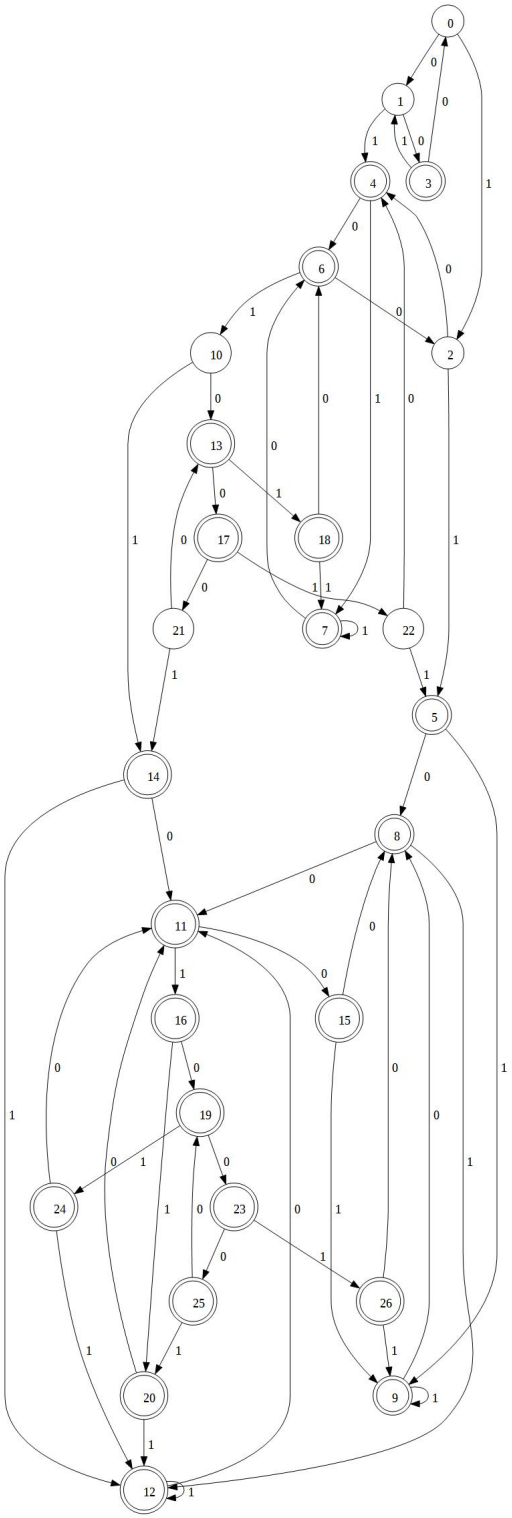

Figura 2: AFD resultante con 26 estados

3.4. Clausura λ

Para cada nuevo estado de la determinización es necesario realizar la clausura λ, es decir, obtener todos los estados alcanzables via transiciones vacías. Para ello procedimos con un algoritmo estilo BFS (con una cola) obteniendo los estados "vecinos"por transiciones λ.

4. Conclusiones

En la realización de este trabajo nos topamos con las dificultades que representan la implementación y la ejecución de los algoritmos vistos en clase para AFND (sobre todo aquellos de tiempo exponencial, como la determinización).