DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

TP1

29/4/2015 Teoría de Lenguajes

PLD

Integrante	LU	Correo electrónico
Pablo Herrero	332/07	pablodherrero@gmail.com
Diego Sueiro	75/90	dsueiro@gmail.com
Leandro Tozzi	-	leandro.tozzi@gmail.com

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

Contents

1 Introducción

El Trabajo Práctico consiste en implementar un programa que permita la construcción y ejecución de Autómatas Finitos Determinísticos (AFD's).

2 Implementación.

La implementación se realizó en Python v3 utilizando el esqueleto provisto por la cátedra. Se proveen los test de unidad y los archivos de entrada de datos utilizados para verificar el correcto funcionamiento del programa.

2.1 Representación Autómata

Se utiliza la clase **Automata** para representar la estructura y los algoritmos necesarios para procesar los autómatas. Mediante conjuntos almacenamos los estados totales, los estados finales y el lenguaje a utilizar. La tabla de transiciones se representa mediante un diccionario.

2.2 Lectura Expresion Regular

Mediante la función **ReadFromFile** se procesa el archivo de entrada que contiene la expresión regular parseada. Se chequea que solo contenga las operaciones y los caracteres aceptados por el enunciado, así como también, el indentado de las mismas. La función **ReadFromFile** devuelve una lista que contiene las operaciones de la ER a realizar, así como también el numero de operandos a utilizar por cada operación

2.3 Conversion AFND a AFD

El primer punto del TP plantea leer una ER y generar un AFD mínimo. Una vez procesado el archivo de entrada que contiene la expresión regular, se genera un AFND-λ mediante la clase **AFNDfromER**. Esta clase utiliza como soporte la clase **BuilderAF** que recibe simplemente agrega los estados y las transiciones λ necesarias para construir el autómata que representa la ER que le pasamos. En base a este AFND resultante, se construye un AFD mediante la clase **AFDfromAFN**

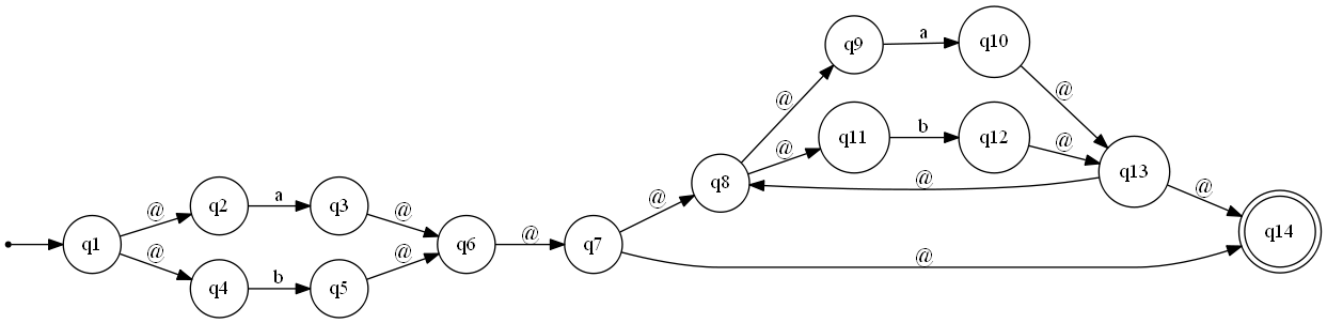

Figure 1: Conversion AFND a AFD: Autómata de Entrada AFND

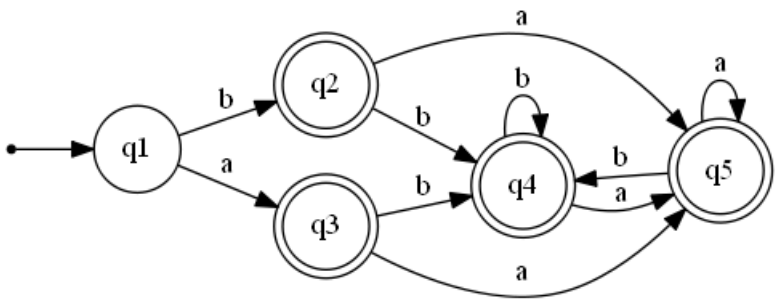

Figure 2: Conversion AFND a AFD: Autómata resultante AFD

2.4 Lectura de AFD desde archivo de entrada

Se procesa el archivo de entrada y se genera un AFD mediante la clase **AFDfromFile**. No se implementó chequeo de errores. Asuminos que el archivo de entrada cumple con la estructura planteada en el enunciado.

2.5 Generación de archivo .dot

Se generan correctamente los archivos .dot para ser visualizados mediante GraphViz. Adicionalmente se genera el archivo gráfico en formato png.

2.6 Minimización

Se implementó el algoritmo de minimización por particionado en clases equivalentes. Se comienza desde un particionado inicial de 2 grupos, estados finales y estados no finales. Luego se recorre cada estado y se chequea a que grupo iría mediante cada una de los símbolos del lenguaje. De esta manera vamos agrupando en clases equivalentes a los estados. El algoritmo termina cuando no se puede volver a particionar. Luego eliminamos los estados inalcanzables del resultado para poder rearmar el autómata M mínimo

Algorithm 1 Minimizar(automata M)

Require: M automata finito deterministico
Ensure: M minimo

```
1:  grupoA := {Estados Finales}
2:  grupoB := {Estados no Finales}
3:  repeat
4:    for all grupo do
5:      for all estado do
6:        Encontrar a que grupo nos llevan las entradas
7:        if Hay diferencias then
8:          Particionar el grupo en conjuntos cuyos estados
            vayan a los mismos grupos bajo esa entrada
9:        end if
10:     end for
11:   end for
12: until (no haya nuevo particionado)
13: RemoverEstadosInalcanzables(M)
14: M ← grupos como estados
```

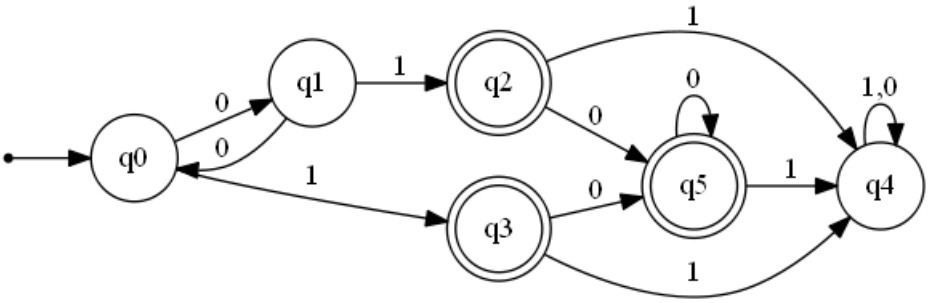

Figure 3: Ejemplo de Minimización: Autómata de Entrada - sin minimizar

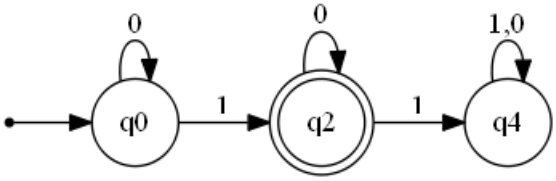

Figure 4: Ejemplo de Minimización: Autómata minimizado

2.7 Intersección de Autómatas

Se implementó la operación **Cross Product** entre autómatas para la realización de este item.

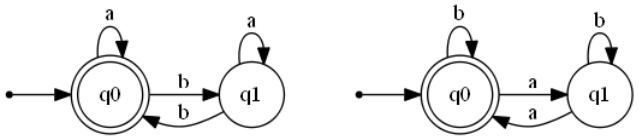

Figure 5: Ejemplo de Intersección: Autómatas a intersectar

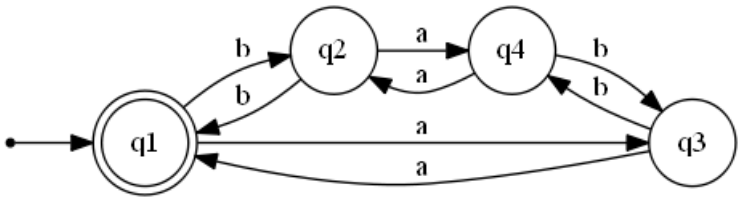

Figure 6: Ejemplo de Intersección: Resultado

2.8 Complemento

Primero completamos el autómata mediante el método **complete_all** de la clase **autómata**. Luego se llamá al método **complemento** de la misma clase para realizar el complemento de su lenguaje del autómata.

2.9 Equivalencia de Autómatas

En este punto se aprovecharon las funciones implementadas previamente, ya que calculamos que si la intersección del complemento es **NULL** en ambas direcciones, entonces los lenguajes generados por los autómatas son equivalentes

3 Conclusiones

La realización del trabajo práctico nos permitió famiarizarnos con los algoritmos vistos en la teórica para el manejo de expresiones regulares y autómatas.