Minimización de AFDs

Para la realización de la práctica 1 de la asignatura Teoría de Lenguajes, hemos definido una clase "AFD" que encapsula todas las estructuras de datos y métodos relacionados con los autómatas finitos deterministas. Esta clase será ampliada en la práctica 2 para que permita calcular el autómata universal de un autómata finito determinista.

La clase AFD

Como ya se ha comentado, la clase AFD es la clase que encapsula todos los datos y métodos relacionados con los autómatas finitos deterministas.

Un autómata finito determinista es una tupla $A = (Q, \Sigma, \delta, q_0, F)$. Vamos a describir cómo hemos implementado cada uno de estos elementos:

- Q: representa un conjunto de estados, donde cada estado está representado por un símbolo. En nuestro caso, hemos obligado a que este conjunto sea el conjunto {0..n-1}, donde *n* es la cardinalidad de Q. De esta forma, sólo tenemos que guardar *n*, lo que reduce el coste espacial a constante.
- Σ: representa el alfabeto de entrada, es decir, un conjunto de símbolos. En nuestro caso, la clase traduce los símbolos de entrada a números, opera y luego traduce el resultado de vuelta a los símbolos del alfabeto. Para ello necesitamos guardar:
    - Un entero: indica la cardinalidad de Σ. Los números usados en la implementación pertenecerán al conjunto {0..n-1}
    - Un mapa char->int: permite traducir los símbolos de entrada a números.
    - Un vector de char: permite realizar la traducción inversa, de números a símbolos, para devolver el resultado de las operaciones.
- δ: representa una función de transición sobre $Q \times \Sigma \to Q$. La implementación se ha realizado como un mapa (int,int)->int. Esto nos proporciona un acceso en tiempo logarítmico.
- q_0: dado que los estados van del 0 al n-1, el estado inicial se puede guardar simplemente como un entero.
- F: es el conjunto de estados finales. Para su implementación, hemos optado por un vector de booleanos de talla igual al número de estados. De esta forma tenemos un coste espacial lineal (en el peor de los casos, con cualquier implementación tendríamos este coste) y un coste temporal de acceso constate.

Dado que el lenguaje escogido para la realización de la práctica ha sido C++, todos los tipos no básicos usados, como los mapas, vectores, listas y demás, son los que proporciona la librería estándar STL. Esto nos asegura disponer de una implementación correcta y eficiente.

El método de minimización

El método de minimización implementado se basa en un algoritmo del libro “Introduction to Automata Theory, Languages, and Computation” de Hopcroft [1]. Decidimos implementar ese algoritmo tras leer el artículo “On the performance of automata minimization algorithms” de Marco Almeida, Nelma Moreira y Rogério Reis [2].

Este método se basa en el refinamiento de particiones de los estados de forma iterativa. En el momento en que el algoritmo para, aquellos estados que estén en la misma partición serán equivalentes, y podrán ser sustituidos por un solo estado.

Al empezar, el método crea una partición de los estados del autómata, dividiéndolos entre finales y no finales. A partir de ahí empieza a refinar la partición, viendo para cada estado, si al transitar con cada símbolo se llega a el mismo conjunto de la partición o no.

Llega un momento en que a pesar de que iteremos, la partición ya no se refina más, y por tanto podemos parar el algoritmo. En este momento, tenemos garantizado que todos los estados que pertenezcan al mismo conjunto de la partición serán equivalentes, y por lo tanto podremos fusionarlos en un nuevo estado.

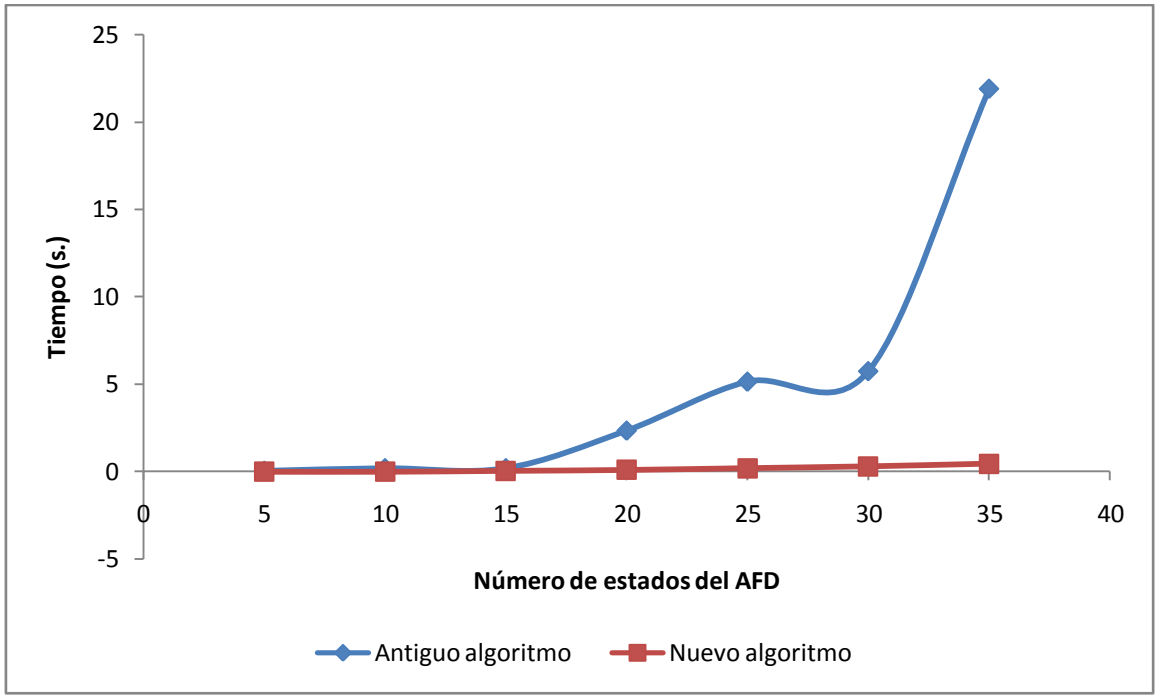
La parte final del algoritmo se encarga de crear el nuevo AFD a partir de la partición que nos devuelve la parte anterior, fusionando los estados y generando la nueva función de transición a partir de la anterior.

Medidas de complejidad de ejecución

El algoritmo de minimización que hemos implementado ha sufrido varias mejoras desde la primera versión que hicimos. Al principio nos basamos en el algoritmo de minimización visto en la asignatura “Teoría de Autómatas y Lenguajes”, usando una clase auxiliar para las particiones y otro tipo de recursos que al final se demostraron innecesarios.

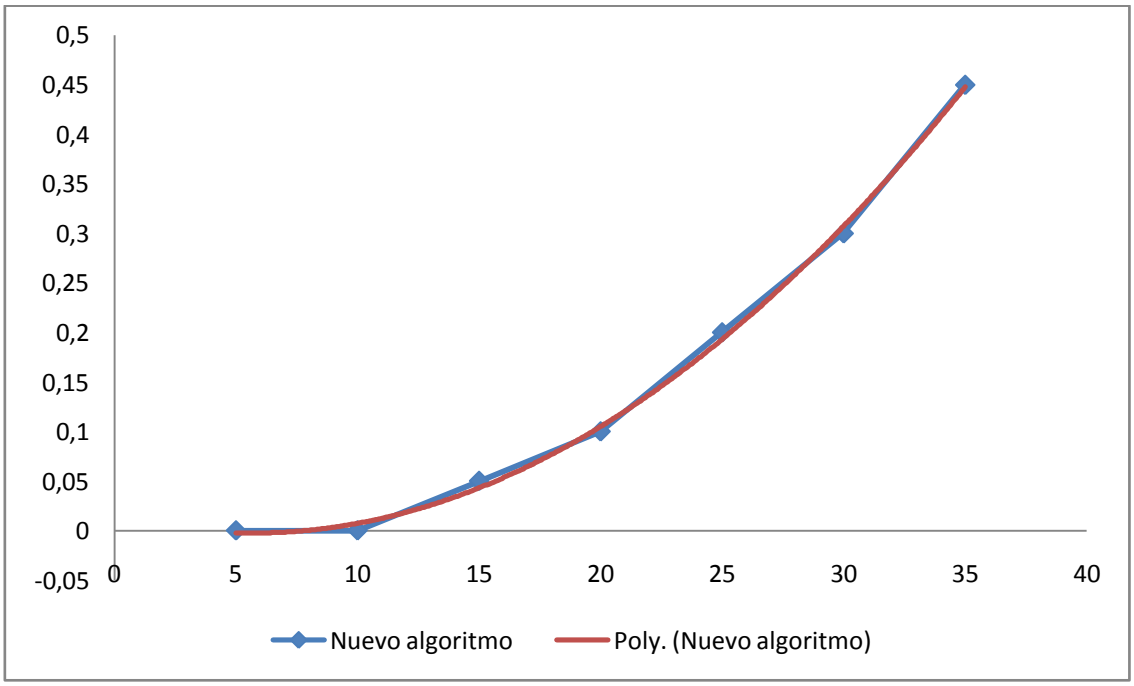
Finalmente acabamos cambiando de algoritmo a otro más eficiente que encontramos en la bibliografía de la asignatura, y estuvimos puliendo detalles de implementación hasta mejorar sustancialmente la eficiencia temporal del algoritmo.

La siguiente gráfica es una comparativa entre el primer algoritmo que implementamos y el algoritmo final que hemos decidido entregar:



La irregularidad que se observa en la gráfica del antiguo algoritmo se debe a la hiperpaginación del sistema operativo.

Como se puede ver, el cambio de algoritmo nos permitió mejorar el comportamiento asintótico. De hecho, el antiguo algoritmo tenía un coste exponencial mientras que, como se puede ver en la siguiente gráfica, el nuevo algoritmo tiene un coste cuadrático:



La línea roja en la gráfica representa la función cuadrática que mejor se ajusta a los datos. Se ve claramente que el ajuste es prácticamente perfecto.

Algunos ejemplos de ejecución

$\Sigma = \{0,1\}$

Ejemplo 1

Entrada:

$Q = \{0,1,2,3\}$ $F = \{0\}$ $q_0 = 0$

$\delta(0,0) = 0$ $\delta(0,1) = 1$

$\delta(1,0) = 1$ $\delta(1,1) = 1$

$\delta(2,0) = 2$ $\delta(2,1) = 3$

$\delta(3,0) = 3$ $\delta(3,1) = 3$

Salida:

$Q = \{0\}$ $F = \{0\}$ $q_0 = 0$

$\delta(0,0) = 0$

Ejemplo 2

Entrada:

$Q = \{0,1,2,3\}$ $F = \{1,2\}$ $q_0 = 0$

$\delta(0,0) = 1$ $\delta(0,1) = 2$

$\delta(1,0) = 1$ $\delta(1,1) = 3$

$\delta(2,0) = 2$ $\delta(2,1) = 3$

$\delta(3,0) = 3$ $\delta(3,1) = 3$

Salida:

$Q = \{0,1\}$ $F = \{0\}$ $q_0 = 1$

$\delta(0,0) = 0$

$\delta(1,0) = 0$ $\delta(1,1) = 0$

Ejemplo 3

Entrada:

$Q = \{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16\}$ $F = \{15\}$ $q_0 = 0$

$\delta(0,0) = 1$ $\delta(0,1) = 8$ $\delta(2,0) = 16$ $\delta(2,1) = 3$

$\delta(1,0) = 2$ $\delta(1,1) = 16$ $\delta(3,0) = 16$ $\delta(3,1) = 4$

$\delta(4,0) = 5$ $\delta(4,1) = 16$ $\delta(11,0) = 12$ $\delta(11,1) = 16$

$\delta(5,0) = 6$ $\delta(5,1) = 16$ $\delta(12,0) = 13$ $\delta(12,1) = 16$

$\delta(6,0) = 16$ $\delta(6,1) = 7$ $\delta(13,0) = 16$ $\delta(13,1) = 14$

$\delta(7,0) = 16$ $\delta(7,1) = 15$ $\delta(14,0) = 16$ $\delta(14,1) = 15$

$\delta(8,0) = 9$ $\delta(8,1) = 16$ $\delta(15,0) = 16$ $\delta(15,1) = 16$

$\delta(9,0) = 10$ $\delta(9,1) = 16$ $\delta(16,0) = 16$ $\delta(16,1) = 16$

$\delta(10,0) = 16$ $\delta(10,1) = 11$

Salida:

$Q = \{0,1,2,3,4,5,6,7,8,9,10\}$ $F = \{0\}$ $q_0 = 8$

$\delta(1,1) = 10$ $\delta(2,0) = 1$

$\delta(3,0) = 2$ $\delta(4,1) = 3$

$\delta(5,0) = 4$ $\delta(6,1) = 4$

$\delta(7,0) = 6$ $\delta(8,0) = 7$

$\delta(8,1) = 9$ $\delta(9,0) = 5$

$\delta(10,1) = 0$

Referencias

[1] J. Hopcroft, R. Motwani, and J. D. Ullman: *Introduction to Automata Theory, Languages and Computation*. Addison Wesley, 2000

[2] Marco Almeida, Nelma Moreira and Rogério Reis*: On the performance of automata minimization algorithms.* Technical Report Series: DCC-2007-03. Departamento de Ciência de Computadores & Laboratório de Inteligência Artificial e Ciência de Computadores, Faculdade de Ciências da Universidade do Porto, 2007