

Minimización de AFD'S

Eduardo Salas

Jorge Nicho

Alvaro Aguirre

Teoría de la computación

UTEC

June 27, 2020

Introducción

Un autómata finito determinista permite evaluar una entrada de números o caracteres llamada cadena a través de estados que permiten comprobar si la cadena es válida o no. Todos los autómatas tienen un estado inicial, donde empieza la evaluación, y un estado final donde la cadena es aceptada. Los AFD son útiles para diseñar software y supervisar el comportamiento de circuitos digitales o para escanear largos textos y encontrar palabras o frases específicas dentro de ellos.

Definición del problema

Puede existir un AFD que tenga estados equivalentes. Esto quiere decir que es posible juntarlos y crear un AFD con menos estados. En este trabajo explicaremos dos maneras para poder minimizar un AFD. Es ideal que esté minimizado, ya que se reduce el tiempo de procesamiento de las cadenas recibidas, brindando mayor eficiencia.

1 Estado del Arte

Dado un AFD $M = (Q, \Sigma, \delta, q_0, F)$, se trata de encontrar un AFD M' con $L(M) = L(M')$ y tal que M' tenga el mínimo número de estados posible. El método consiste en encontrar todos los estados que son equivalentes, es decir, que son indistinguibles en el autómata. Por cada clase de estados equivalentes, el autómata mínimo necesitará un solo estado. Hemos hallado 3 algoritmos para lograr este objetivo: Hopcroft, Moore y Myhill Nerode.

1.1 Algoritmo de Moore

El método para minimizar un autómata mediante el algoritmo de moore consiste en encontrar todos los estados que son indistinguibles entre sí y sustituirlos por un único estado. Para ello lo principal es averiguar qué estados lo son y cuáles no. El tiempo de ejecución de este algoritmo en el peor caso es $O(n^2s)$, donde s es el número de sustituciones que se realizan en cada paso y n es el número de estados.

El método para saber qué estados son indistinguibles es el siguiente:

1. Estados separados en grupos que tienen la misma salida inmediata para la misma entrada.
2. Distinguir estados cuyos siguiente estado (s) están en diferentes grupos.
3. Reagrupar los estados y repetir el paso anterior hasta que no hay más estados son distinguibles.

1.1.1 Ejemplo

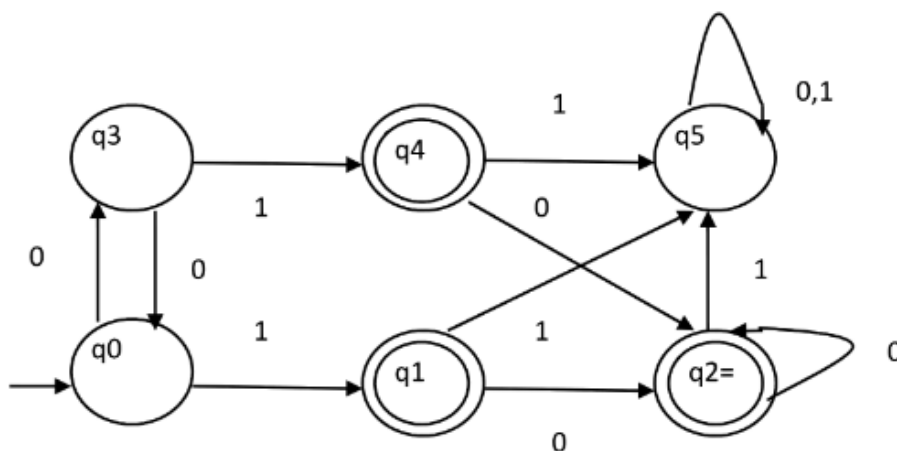


Figure 1: Autómata ejemplo

Paso 1: Separamos los estados de aceptación y los de no aceptación.

$$\mathbb{P}_0 = \{\{q_1, q_2, q_4\}, \{q_0, q_3, q_5\}\}$$

Figure 2: Paso 1

El conjunto de estados lo llamaremos P_o siendo $k = 0$; quiere decir que tienen 0 transiciones iguales, solo están divididos por aceptación y no aceptación.

Paso 2: Separamos aquellos estados en cada conjunto de P_o que tenga 1 estado de transición distinguible.

$$\delta(q_1, 0) = \delta(q_2, 0) = q_2 \text{ y } \delta(q_1, 1) = \delta(q_2, 1) = q_5$$

Figure 3: Paso 2

Por lo tanto en el primer conjunto, todos los valores son indistinguibles. Quiere decir que este no será particionado en P_1 . Ahora veamos el segundo conjunto:

$$\begin{aligned} \delta(q_0, 0) &= q_3 \text{ y } \delta(q_3, 0) = q_0 \\ \delta(q_0, 1) &= q_1 \text{ y } \delta(q_3, 1) = q_4 \end{aligned}$$

Figure 4:

Los movimientos de q_0 con 0 y q_3 con 0 son q_3 y q_0 que están en el mismo conjunto de partición en P_o . Lo mismo sucede en q_0 con 1 y q_3 con 1, que son q_1 y q_4 , que también están en la misma partición en P_o . Caso contrario sucede con q_5 , veamos:

$$\delta(q_0, 0) = q_3 \text{ y } \delta(q_5, 0) = q_5 \text{ y } \delta(q_0, 1) = q_1 \text{ y } \delta(q_5, 1) = q_5.$$

Figure 5:

Vemos que no pertenecen al mismo conjunto en P_o . Es por ello que P_1 quedaría de la siguiente manera:

$$P_1 = \{\{q_1, q_2, q_4\}, \{q_0, q_3\}, \{q_5\}\}$$

Este es el caso final puesto que no se pueden realizar más particiones. El autómata quedaría de la siguiente manera.

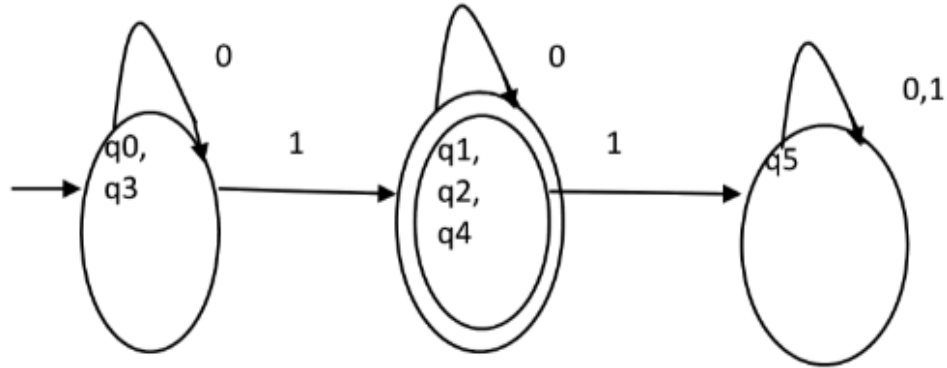


Figure 6: Aútomata final

1.1.2 Pseudocódigo:

```

1 MooreReductionProcedure(fsm first_fsm){
2     fsm new_fsm;
3     vector<vector<states>> groups
4     vector<states> acceptance;
5     vector<states> no_acceptance;
6     for i in first_fsm.states
7         if(i acceptance)
8             acceptance<=i;
9         else
10            no_acceptance<=i;
11     end for
12     groups<=acceptance and no_acceptance;
13
14     for i in groups.size
15         for j in groups[i].size = vec
16             if(distinguishable[j])
17                 groups<=distinguishable[j]
18             else
19                 continue;
20         end for
21     end for
22     append_states(new_fsm,groups)
23
24     return new_fsm;
25 }
```

1.2 Algoritmo de llenado de tabla

El objetivo es identificar a dos estados distintos A y B que puedan ser reemplazados por un solo estado C que se comporte igual a ambos. Se dice que A y B son estados distinguibles si un input 1 lleva al primer estado a un estado de aceptación y al otro a uno de no aceptación. Otra forma de darse cuenta es ver que un estado sea de aceptación y el otro no. En la Figura 1 podemos aplicar lo explicado anteriormente. El método de llenado de tabla consiste en juntar pares de estados distinguibles y los que sobren serán equivalentes.

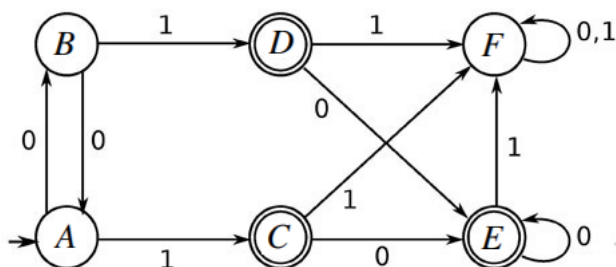


Figure 7: Ejemplo algoritmo de llenado de tabla

Primero nos damos cuenta de que los estados A , B y F son estados de no aceptación mientras que C , D y E sí. Marcamos todos los pares (ED, EA, EF, etc) . Luego nos damos cuenta de que $(A, 1)$ lleva a un estado de aceptación C mientras que $(F, 1)$ nos lleva a F , un estado de no aceptación. Por lo tanto no son equivalentes y marcamos este par en la tabla. Luego repetimos este análisis con todos los pares aún no marcados. El resultado final se aprecia en **Figure 8**.

B					
C	x	x			
D	x	x			
E	x	x			
F	x	x	x	x	x
	A	B	C	D	E

Figure 8: Tabla de pares

Después listamos los pares de estados no marcados: (A,B) , (C,D) , (C,E) , (D,E) . Gráficamente se muestra en **Figure 9**. Finalmente dibujamos el nuevo autómata fusionando los estados equivalentes agrupados en la Figura 3 y el resultado se muestra en **Figure 10**.

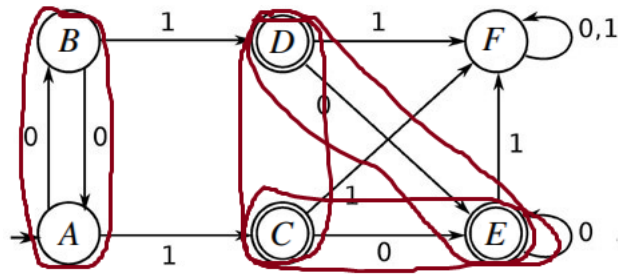


Figure 9: Agrupación visual de los estados equivalentes

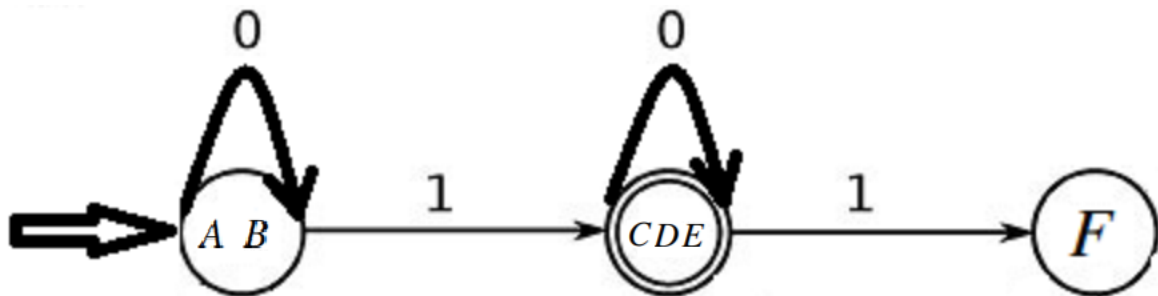


Figure 10: AFD minimizado con el algoritmo de llenado de tabla

Los pasos del algoritmo son:

1. Dar como input un AFD (P, Q) , indicando sus transiciones, estado inicial y estado final.
2. Construir una matriz $n \times n$, con una fila para cada estado y una columna para cada estado, donde n es la suma del número de estados en P y el número de estados en Q . Solo necesitamos considerar el área debajo de la diagonal de la matriz.
3. Marcamos con 0 (sin marcar) o 1 (marcado) todos los pares donde $P \neq F$ y $Q = F$, donde F es el conjunto de estados finales.
4. Si hay pares sin marcar (P, Q) de modo que $[\delta(P,x), \delta(Q,x)]$ esté marcado, marque $[P, Q]$, donde 'x' es un símbolo de entrada. Repita esto hasta que no se puedan hacer más marcas.
5. Combinamos todos los pares de estados que no han sido marcados y los juntamos en un solo estado correspondiente para formar el AFD minimizado.
6. Creamos los nuevos estados para el AFD minimizado.
7. Relacionamos cada estado para formar el AFD minimizado.

1.2.1 Pseudocódigo:

```
1 Table_Filling_Algorithm(A,P){
2   InitializeDFA(A);
3
4   belowDiagonal=length[A]-1;
5   for i=A[1] to length[A]-1
6     for j=A[0] to length[A]-1-belowDiagonal
7       if Distinguishable(A[i],A[j])
8         A[i][j]=1;
9     belowDiagonal--;
10
11   belowDiagonal=length[A]-1;
12   for i=A[1] to length[A]-1
13     for j=A[0] to length[A]-1-belowDiagonal
14       if A[i][j]=0;
15         if Check(A[i],A[j])
16           A[i][j]=1;
17
18     belowDiagonal--;
19
20   n=0;
21   ArrayOfNewStates(P);
22   belowDiagonal=length[A]-1;
23   for i=A[1] to length[A]-1
24     for j=A[0] to length[A]-1-belowDiagonal
25       if A[i][j]=0;
26         P[n]=A[i][j];
27         n++;
28
29     belowDiagonal--;
30
31   Print(P);
32 }
33 }
```

Llenar la tabla, analizar si dos estados son distinguibles y combinar los estados que podrían minimizarse lleva un tiempo polinómico. Debido a que se necesita una cantidad de pares igual a $n(n-1)/2$, tomando en cuenta que existen n estados podemos determinar que la complejidad es de al menos de $O(n^2)$.

1.3 Referencias

[1] John E. Hopcroft, Rajeev Motwani Jeffrey D. Ullman. (1979). Introduction to automata theory, languages, and computation. USA: Addison-Wesley.

[2] Daphne A. Norton (2009). Algorithms for Testing Equivalence of Finite Automata, with a Grading Tool for JFLAP (Master's thesis). Rochester Institute of Technology, USA.