Universidad del Valle de Guatemala
Teoría de la Computación, CC2019
Docentes: Alan Reyes; Tomás Gálvez P.
Semestre 2, 2022

Minimización de AFD: algoritmo de Hopcroft

El lenguaje descrito por una expresión regular es llamado **lenguaje regular**. Un lenguaje regular puede ser aceptado por diferentes AFD's, pero existe un único AFD para cada lenguaje regular cuyo número de estados es el mínimo posible. Este **AFD mínimo** se puede obtener a partir de cualquier otro AFD que acepte el mismo lenguaje regular por medio del **algoritmo de minimización de Hopcroft**. Este algoritmo agrupa los estados del AFD que no pueden ser distinguidos entre sí, pero ¿cómo se distinguen los estados? Se dice que un *string* x distingue al estado s del estado t si a partir del estado s se alcanza un estado de aceptación al seguir el camino descrito por x, mientras que desde t no. En general, cualquier estado s es **distinguible** de t si existe algún *string* que distinga uno de otro.

El algoritmo de Hopcroft es implementado de diferentes formas, pero la idea principal es separar los estados que son distinguibles entre sí. La versión en el libro del dragón usa **particiones**, aunque la versión más popular emplea **tablas**.

En la versión por particiones empezamos por separar los estados entre los de aceptación y los de no-aceptación, porque son distinguibles entre sí por medio del *string* vacío. Quedamos con una partición del conjunto de estados original.

Procedemos a tomar un subconjunto en la partición y, para cada uno de sus estados, determinamos los estados que se alcanzan a partir de ellos con cada símbolo del alfabeto, anotando a qué subconjuntos pertenecen. El resultado será un listado de los subconjuntos que se alcanzan con cada estado del subconjunto que tomamos.

El siguiente paso es reemplazar al subconjunto que tomamos de la partición original con el listado de subconjuntos que se alcanzaron a partir de él. Si la partición no cambió luego de este reemplazo procedemos a repetir el proceso con el siguiente subconjunto en la partición. Si la partición sí cambió volvemos a empezar todo tomando un subconjunto de la nueva partición. Todo este proceso debe repetirse hasta que en una pasada se revisen todos los subconjuntos en la partición y no haya cambio en ninguno.

A continuación, una presentación algorítmica del proceso recién descrito:

1. Separar los estados entre los que pertenecen al conjunto de estados de aceptación y los que no, y agregar estos grupos a un conjunto de partición P.
2. Para cada grupo g en la partición P:
    a. Para cada estado s en el grupo g:
        i. Para cada símbolo α en el alfabeto:
            1. Sea $t = \delta(s, \alpha)$.
            2. Para cada grupo h en la partición P:
                a. Si $t \in h$:
                    i. Agregar h al conjunto D_s.
            3. Agregar D_s a la lista L.
    b. Sea $i = 0$.
    c. Mientras que la lista L no esté vacía:
        i. Tomar un conjunto D_x en la lista L.
        ii. Copiar el estado x correspondiente a D_x a un conjunto K_i.
        iii. Sacar a D_x de L.

iv. Para cada conjunto D_y que queda en la lista L:
    1. Si $D_x = D_y$ (estrictamente igual; el orden de sus elementos sí importa):
        a. Meter el estado y correspondiente a D_y a K_i.
        b. Sacar a D_y de L.
v. $i = i + 1$.

d. Si $K_0 != g$ entonces:
    i. Remover g de la partición P.
    ii. Ingresar todos los K_i en lugar de g en la partición P.
    iii. Volver al paso 2, y reiniciar el ciclo (es decir, no siga con el siguiente grupo).

3. Inicializar un AFD vacío.
4. Para cada grupo J en la partición P agregar un estado nuevo al AFD.
    a. El grupo J que contenga al estado inicial del AFD siendo minimizado será el estado inicial del nuevo AFD.
    b. Cualquier grupo J que contenga estados de aceptación del AFD original será un estado de aceptación del AFD nuevo.
    c. Para cada símbolo α en el alfabeto habrá una transición del grupo J en la partición a otro grupo L en la partición si algún estado en J tiene transición con α a algún estado en L.

Nota: este algoritmo es una adaptación mía de lo que está en el libro del dragón junto con la presentación *partitioning.ppt* citada en las fuentes. Los últimos pasos (a partir del punto 4) son más indicaciones que pasos, pero confío en que con ellas se entienda cómo se debe proceder al alcanzar este punto en el algoritmo.

Una implementación distinta de esta misma versión del algoritmo de Hopcroft (encontrada en Wikipedia) es la siguiente:

```
P := {F, Q \ F}; // '\' denota la operación de diferencia entre conjuntos
W:= {F, Q \ F};

while (W no está vacío):
   tomar and remover un conjunto A de W

  for each c in ∑:
      sea X el conjunto de estados que llevan a un estado en A por medio de una transición con c

      for each Y in P tal que X ∩ Y no es vacío:
         reemplazar Y en P por los conjuntos X ∩ Y y Y \ X

         if Y pertenece a W:
            reemplazar Y en W por los mismos los conjuntos X ∩ Y y Y \ X
         else:
            if |X ∩ Y| <= |Y \ X|:
               agregar X ∩ Y a W
            else:
               agregar Y \ X a W
```

La idea de esta versión (en ambas implementaciones) del algoritmo es ir refinando la partición original en subconjuntos cada vez más pequeños, hasta que ya no se pueda refinar más. Diríamos que W son los grupos en la partición "bajo refinamiento". Esa es la razón para la última condición en la segunda versión del algoritmo, pues al colocar en W el subconjunto más pequeño de la separación estamos procurando refinarlo lo más posible. W se va a ir quedando sin grupos porque cuando un grupo de la partición ya no se pueda refinar más será separado en sí mismo y el conjunto vacío, de los cuales el conjunto vacío irá a parar a W (**Nota**: no hay que olvidar que cualquier grupo en W será también un grupo de la partición P). ¿Y hacia qué se está refinando la partición? Hacia conjuntos de estados indistinguibles entre sí, pero distinguibles de los estados en los demás conjuntos. Esa es la idea principal del algoritmo de Hopcroft.

Se puede observar, a partir de nuestra partición inicial entre estados de aceptación y de no-aceptación, que estos estados son distinguibles por medio del *string* vacío, como se mencionó. Luego, el algoritmo nos dice que un par de estados s y t estarán en grupos diferentes si sus transiciones con un símbolo α llevan a estados p y q distinguibles entre sí. Pero para que estos estados p y q sean distinguibles tiene que existir un *string* x que los haga tales, de forma que el par de estados s y t será distinguible por medio del *string* αx. Esta inducción se puede aplicar también a los estados indistinguibles.

La versión del algoritmo de Hopcroft que usa una tabla permite visualizar directamente esta inducción. El rollo está en crear una tabla sin repeticiones que relacione cada par de estados en el AFD original. Entonces marcamos las celdas de parejas de estados que sabemos que son distinguibles, que inicialmente son parejas donde un estado es de aceptación y el otro no. Luego, para cada posible pareja de estados p y q que no esté marcada en la tabla debemos revisar sus transiciones con cada símbolo del alfabeto. Si alguna de esas transiciones lleva a dos estados cuya celda en la tabla ya está marcada entonces marcamos la celda de la pareja p, q. Si ninguna de las transiciones lleva a estados con celda marcada entonces procedemos a crear una lista para cada pareja de estados que alcanzamos a partir de p y q con cada símbolo del alfabeto (a menos que los estados alcanzados sean iguales). A cada lista agregamos la pareja (p, q).

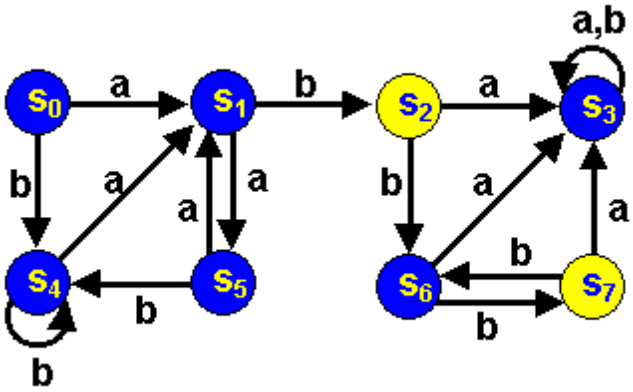
La utilidad de esto último está en que cada vez que marcamos una celda para una pareja de estados s, t en la tabla marcamos también las celdas de cada par de estados en la lista de s, t. Cuando terminamos de revisar todas las posibles parejas, las celdas que queden sin marcar serán estados indistinguibles que podemos unir en un mismo estado en el AFD mínimo.

El algoritmo es el siguiente:

```
for p in F y q in Q - F:
  marcar (p,q)
for cada pareja de estados (p,q) en F × F o (Q − F) × (Q − F):
  if (δ(p,a),δ(q,a)) está marcado para algún símbolo a:
    marcar (p,q) y todas las parejas en su lista, recursivamente
  else:
    for each símbolo a en la expresión regular:
      agregar (p,q) a la lista de (δ(p,a),δ(q,a)) a menos que δ(p,a) = δ(q,a)
```

A continuación, ejemplos.

Ejemplo por Tablita



En este AFD los estados amarillos son estados de aceptación y el estado inicial es S_0. Vamos a resolverlo con el algoritmo por tabla.

S1							
S2							
S3							
S4							
S5							
S6							
S7							
	S0	S1	S2	S3	S4	S5	S6

Marcamos los estados distinguibles. Es decir, marcamos todas las celdas que aparejan un estado de aceptación y uno que no es de aceptación.

S1							
S2	X	X					
S3			X				
S4			X				
S5			X				
S6			X				
S7	X	X		X	X	X	X
	S0	S1	S2	S3	S4	S5	S6

Luego recorremos cada posible pareja de estados en los conjuntos de estados de aceptación y de no-aceptación. En resumen, todas las celdas vacías. Vamos a hacer otra tabla que contenga la información sobre transiciones por pareja.

Celda en revisión	Transición con a	Transición con b
$(S0, S1)$	$(S1, S5)$	$(S4, S2)$
$(S0, S3)$	$(S1, S3)$	$(S4, S3)$
$(S0, S4)$	$(S1, S1)$	$(S4, S4)$
$(S0, S5)$	$(S1, S1)$	$(S4, S4)$
$(S0, S6)$	$(S1, S3)$	$(S4, S7)$

Vemos entonces que podemos marcar las celdas para $(S0, S1)$ y $(S0, S6)$, mientras que producimos las siguientes listas:

- $(S1, S3) = \{(S0, S3)\}$
- $(S4, S3) = \{(S0, S3)\}$

Ignoramos las listas para $(S1, S1)$ y $(S4, S4)$ porque sabemos que los estados en estas tuplas son indistinguibles por ser iguales.

La nueva tabla es:

S1	X						
S2	X	X					
S3			X				
S4			X				
S5			X				
S6	X		X				
S7	X	X		X	X	X	X
	S0	S1	S2	S3	S4	S5	S6

Siguiente paso, revisar las parejas (celdas vacías) con *S1*:

Celda en revisión	Transición con a	Transición con b
$(S1, S3)$	$(S5, S3)$	$(S2, S3)$
$(S1, S4)$	$(S5, S1)$	$(S2, S4)$
$(S1, S5)$	$(S5, S1)$	$(S2, S4)$
$(S1, S6)$	$(S5, S3)$	$(S2, S7)$

Ejercicio de práctica A: marque en la tabla siguiente las celdas que corresponda, según con la revisión anterior. Cree también las listas para las parejas de estados cuyas transiciones hayan llevado únicamente a celdas sin marcar.

S1	X						
S2	X	X					
S3			X				
S4			X				
S5			X				
S6	X		X				
S7	X	X		X	X	X	X
	S0	S1	S2	S3	S4	S5	S6

En el caso de las parejas con $S2$:

$(S2, S7)$	$(S3, S3)$	$(S6, S6)$

No podemos marcar nada porque los estados en cada tupla son indistinguibles.

Ahora las parejas con *S3*:

Celda en revisión	**Transición con a**	**Transición con b**
$(S3, S4)$	$(S3, S1)$	$(S3, S4)$
$(S3, S5)$	$(S3, S1)$	$(S3, S4)$
$(S3, S6)$	$(S3, S3)$	$(S3, S7)$

Marcamos la celda para $(S3, S6)$ y para $(S3, S4)$. Nótese que entonces debemos marcar también $(S3, S5)$ por sus transiciones en b. Es decir, lo marcamos todo. Las listas que teníamos hasta el momento eran

- $(S2, S7) = \{(S1, S6)\}$
- $(S5, S3) = \{(S1, S6)\}$

Pero al marcar $(S5, S3)$ marcamos también $(S1, S6)$.

La tabla queda así:

S1	X						
S2	X	X					
S3	X	X	X				
S4		X	X	X			
S5		X	X	X			
S6	X	X	X	X			
S7	X	X		X	X	X	X
	S0	S1	S2	S3	S4	S5	S6

Revisamos las parejas con *S4*:

Celda en revisión	**Transición con a**	**Transición con b**
$(S4, S5)$	$(S1, S1)$	$(S4, S4)$
$(S4, S6)$	$(S1, S3)$	$(S4, S7)$

Marcamos la celda $(S4, S6)$:

S1	X						
S2	X	X					
S3	X	X	X				
S4		X	X	X			
S5		X	X	X			
S6	X	X	X	X	X		
S7	X	X		X	X	X	X
	S0	S1	S2	S3	S4	S5	S6

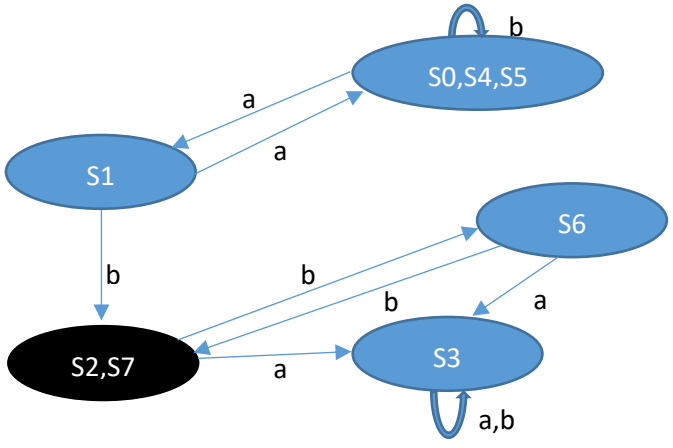
Por último, la celda vacía que no hemos revisado con $S5$:

Celda en revisión	Transición con a	Transición con b
$(S5, S6)$	$(S1, S3)$	$(S4, S7)$

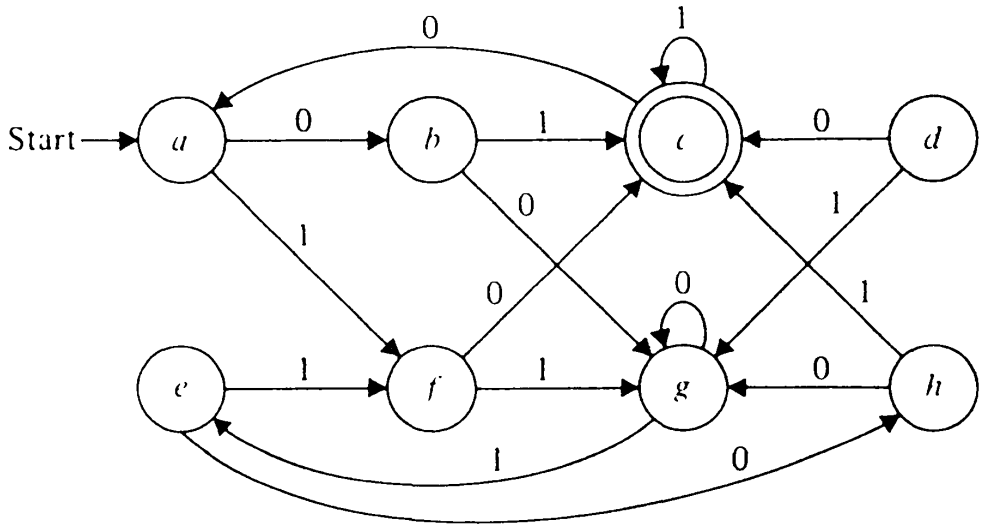
Marcamos la celda $(S5, S6)$:

S1	X						
S2	X	X					
S3	X	X	X				
S4		X	X	X			
S5		X	X	X			
S6	X	X	X	X	X	X	
S7	X	X		X	X	X	X
	S0	S1	S2	S3	S4	S5	S6

Ya completamos la revisión de las celdas vacías. Las celdas que quedaron vacías representan estados equivalentes. Nuestro autómata quedaría así:



Ejemplo por Particiones



Creamos la partición $P = \{\{a, b, d, e, f, g, h\}, \{c\}\}$

Tomamos el primer grupo de P y averiguamos los estados que se alcanzan con cada símbolo para cada estado en el grupo:

Estado	Transición con 0	Transición con 1
a	b	f
b	g	c
d	c	g
e	h	f
f	c	g
g	g	e
h	g	c

Determinamos a que grupo pertenece cada estado alcanzado:

Estado	Transición con 0	Transición con 1
a	G1	G1
b	G1	G2
d	G2	G1
e	G1	G1
f	G2	G1
g	G1	G1
h	G1	G2

Agrupemos los estados según la serie de grupos en su fila:

Estado	Transición con 0	Transición con 1
a	G1	G1
g	G1	G1
e	G1	G1
b	G1	G2
h	G1	G2
d	G2	G1
f	G2	G1

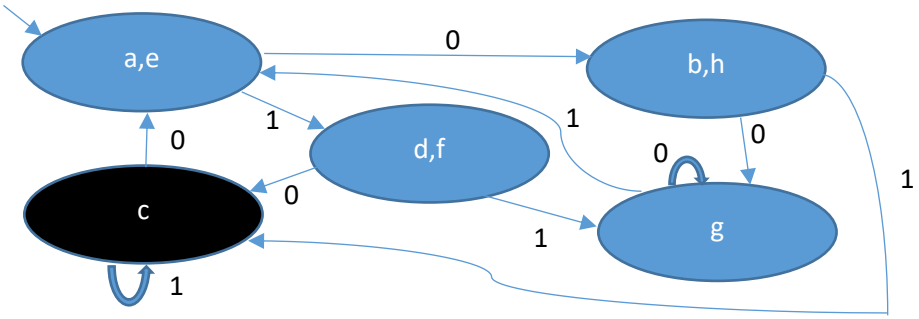
Como la primera de estas agrupaciones (los estados marcados en rojo) no es igual al grupo de donde salió, reemplazamos el grupo que tomamos de P por las nuevas agrupaciones creadas; luego reiniciamos el ciclo.

Tomamos el primer grupo, sacamos sus transiciones, y vemos a qué grupos pertenecen los estados alcanzados. Según estos resultados creamos nuevas particiones.

Ejercicio de práctica B: llene la tabla siguiente y clasifique cada estado según sus destinos con 0 y 1.

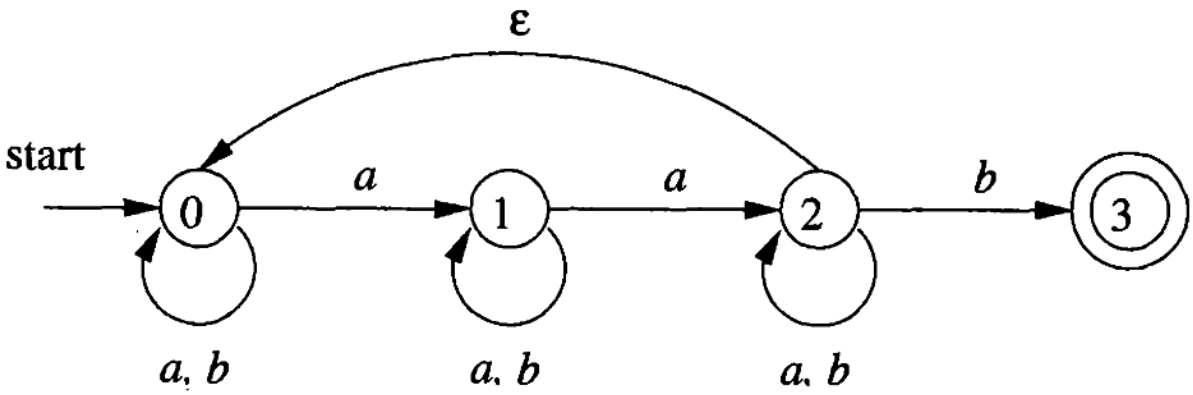
Estado	Transición con 0	Transición con 1

Para esta tabla resultante, el algoritmo tomará cada grupo en la partición y verá que la "partición" producida con él es igual al mismo grupo que tomó. Por tanto, no reinicia el ciclo, sino que sigue con el siguiente grupo. Al terminar, nuestra partición original tendrá los grupos especificados en la tabla anterior y el estado de aceptación c. Nuestro AFD se arma viendo a qué grupos de la partición transfiere cualquier estado de cada grupo bajo los símbolos del alfabeto. Queda así:



Obsérvese que c sigue siendo el estado de aceptación y simplemente mantiene sus transiciones originales. Además, (a, e) es el estado inicial porque a era el estado inicial en el AFD original.

Ejercicio de práctica C: recuerde la expresión regular del AFN y el AFD obtenido a partir de él en el ejercicio de práctica A del PDF sobre conversión de AFN a AFD y construcción directa de AFD (mostrado a continuación). Construya el AFD directamente a partir de la expresión regular obtenida. Luego minimice tanto este AFD como el AFD obtenido a partir del AFN. Compare los resultados.



Fuentes

- http://en.wikipedia.org/wiki/DFA_minimization
- http://web.cs.wpi.edu/~kal/courses/cs503/module4/partitioning.ppt
- Aho, A. V., Lam, M. S., Ravi, S., & Ullman, J. D. (2007). *Compilers: Principles, Techniques and Tools.* Pearson.
- Hopcroft, J. E., & Ullman, J. D. (1979). *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley.