Práctica 1: Transformación a Determinista

G1391 P4
Daniel Mohedano
Silvia Sopeña

7 de noviembre de 2020

1. Estructura Intermedia

La estructura intermedia que decidimos utilizar para la implementación del algoritmo fue la siguiente:

```
struct _TransformData {
    int ** f;
    int ** contenido;
    int num_estados;
};
```

Una de los primeros problemas que encontramos al comenzar a pensar como implementar el algoritmo era la forma de identificar los estados. En cuanto a nomenclatura, estaba ya decidido que simplemente ibamos a nombrar los estados con la concatenación de los nombres de los estados que lo "formaban". Como idea resulta sencilla y además es a lo que estamos acostumbrados al ver los ejemplos de ejecución del algoritmo en las diapositivas. A la hora de implementarlo en código iba a resulta más engorroso. En primer lugar porque el hecho de concatenar cadenas en C no es algo trivial y además podía dar problemas el hecho de tener que comparar estados de la forma *q0q1* y *q1q0*.

La solución que ideamos para este problema fue simplemente almacenar para cada estado creado qué estados del autómata original lo formaban. La forma más sencilla de almacenar esta información era en una tabla, llamada **contenido**, donde las filas indican cada nuevo estado creado (se identifican simplemente por el índice) y las columnas son cada estado del autómata original. El contenido de la tabla para la fila i y la columna j es entonces un número que vale **0** si el estado original q_i no forma parte del estado nuevo creado q'_j y **1** si si que lo hace. De esta forma, resolvíamos ambas dificultades a la vez: por un lado no nos tenemos que preocupar de concatenar cadenas hasta más tarde cuando realmente sean necesarias, y por el otro, el "orden" en el que se incluyen los estados dentro del nuevo estado ya no tiene importancia. A la hora de comparar estados (para ver si un estado ha sido ya creado o no) simplemente tenemos que comparar que las filas tengan los 1s y los 0s en las mismas posiciones exactamente.

Por ejemplo, el estado **q0q1q3** tendría la siguiente representación:

	q0	q1	q2	q3
q0q1q3	1	1	0	1

Por lo tanto, a la hora de identificar los estados dentro del propio código simplemente se puede hacer con el índice de la fila que ocupan en la tabla.

Para almacenar la tabla de transición la solución resulta directa después de haber ideado la tabla de contenido. Simplemente es una tabla, llamada **f**, con un número de columnas igual al número de símbolos de entrada. El número de filas va cambiando a medida que se ejecuta el algoritmo puesto que cada vez que creamos un nuevo estado añadimos una nueva fila a la tabla (de esta forma también el índice de fila para dicho estado es el mismo tanto en la tabla de transición como en la tabla de contenido). Además, ahora solo necesitamos almacenar en cada transición el índice del estado al que transita.

Por último, otra ventaja de construir esta estructura intermedia es que podemos implementar el algoritmo de una pasada en vez de dos. Como se ha comentado antes, cada vez que se crea un nuevo estado se añade una fila a ambas tablas. Y la forma de saber cuando se ha acabado es cuando ya no queden más filas de las que calcular transiciones en la tabla de transiciones.

2. Resultados de Tests

El programa para testear *s1_main* tiene dos configuraciones cargadas que se seleccionan a través de argumentos al ejecutar el programa. A continuación se muestran los resultados obtenidos con ambas configuraciones:

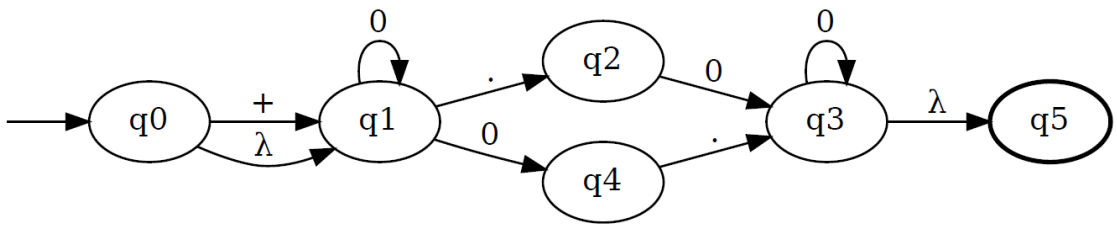

Figura 1: Autómata no determinista 1.

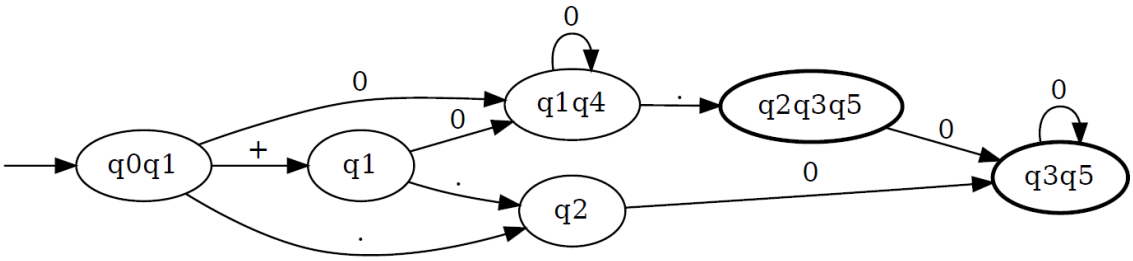

Figura 2: Autómata determinista 1.

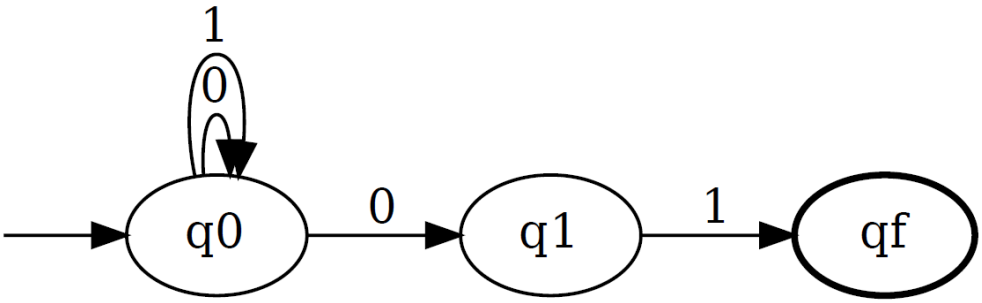

Figura 3: Autómata no determinista 2.

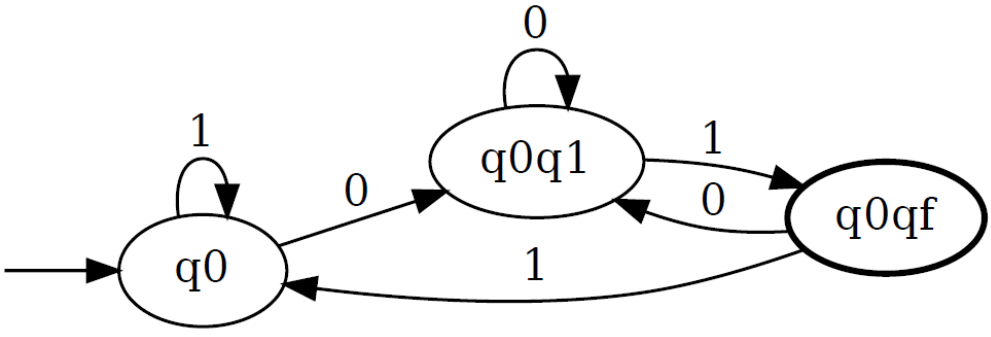

Figura 4: Autómata determinista 2.