Ciencias computacionales

Propedeutico: Autómatas

Contents

1 Una visión informal del autómata finito

Esta sección introduce a los lenguajes regulares, mismos que pueden ser descritos por un autómata finito. Como se mencionó anteriormente un autómata tiene un conjunto de estados y su control se mueve de un estado a otro en respuesta a una entrada. Un autómata finito puede ser:

- Determinista: si el autómata no puede estar en más de un estado al mismo tiempo.
- No determinista: si el autómata puede estar en varops estados al mismo tiempo.

Es importante notar que los autómatas no deterministas no son capaces de describir ningún lenguaje que no pueda ser descrito con un autómata determinista, pero si permiten programar soluciones a problemas usando lenguajes de alto nivel.

Para ilustrar a los autómatas finitos se usará como ejemplo de acompañamiento un problema de la vida real: el dinero electrónico. El dinero electrónico se compone de archivos electrónicos que un cliente puede usar para pagar por internet. Haciendo a un lado los aspectos de encriptación y almacenamiento de los archivos es necesario desarrollar protocolos que permitan la manipulación del dinero en las diversas formas que sean requeridas por lo usuarios, así como detectar "patrones" extraños o sospechosos. En esta sección se mostrarán algunos ejemplos al respecto.

1.1 Las reglas base

Siguiendo el ejemplo del comercio el dinero electrónico hay tres actores: el cliente, la tienda y el banco, y 5 cosas pueden pasar:

- El cliente paga: el cliente envía dinero a la tienda.
- El cliente cancela: el dinero es enviado de regreso al banco con la indicación de ser abonado a la cuenta del cliente.
- La tienda envía el producto al cliente.
- La tienda cobra el dinero: se envía dinero al banco con la indicación de que sea abonado a la cuenta del ciente.
- El banco transfiere el dinero a la tienda creando un nuevo archivo electrónico de dinero para enviarlo a la tienda.

1.2 El protocolo

Los tres participantes deben diseñar sus comportamientos de forma razonable:

- Cliente: podría tratar de copiar el dinero o usar el mismo certificado electrónico para pagar en dos tiendas, o pagar y cancelar. Cualquier cosa que le permita tener algo gratis.
- Banco: debe verificar qe dos tiendas no acrediten el mismo dinero electrónico y no debe permitir una acreditación después de una cancelación.
- La tienda: debe tener cuidado en mandar los productos solo después de validar que recibió un pago válido.

Los comportamientos pueden modelarse como un autómata finito como los mostrados en las figuras 1, 3 y 2.

1.3 Habilitación el autómata para ignorar acciones

Hay algunos problemas con el autómata de las figuras anteriores. Por ejemplo, la tienda no se ve afectada por un mensaje de cancelación que le permitiera quedarse en el mismo estado al recibirlo. Si se usara el autómata de la figura 2 el autómata dejaría de funcionar al recibir un mensaje de cancelación. En general, si el autómata recibe un mensaje no esperado dejará de funcionar. Por este motivo es necesario agregar algunos arcos a los autómatas anteriores que le permitan ingnorar ciertas acciones. Las siguientes figuras muestran los autómatas corregidos.

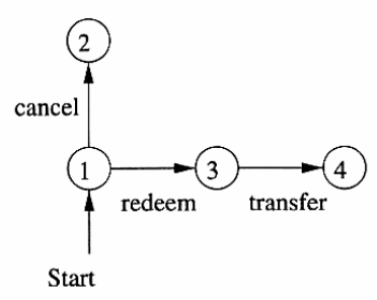

Figure 1: Autómata finito de banco.

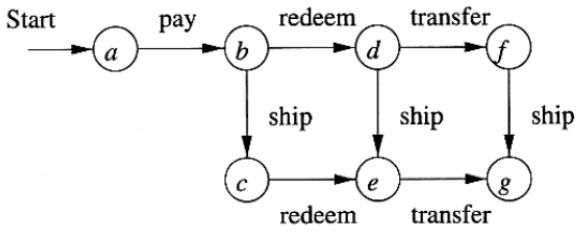

Figure 2: Autómata finito de la tienda.

1.4 El sistema completo como un autómata

Los autómatas de las figuras 4, 6 y 5 muestran el comportamiento individual de los tres participantes, pero no representan la interacción entre los tres participantes. Para poder entender esta interacción se usa el autómata producto. Los estados de este autómata representan un par de estados, el primero de la tienda y el segundo del banco. Por ejemplo el estado (3,d) del autómata producto, mostrado en la figura 7, representa la situación donde el banco está en el estado 3 y la tienda en el estado d. Las filas del autómata producto corresponden al estado del banco y las columnas corresponden al estado de la tienda. Para ahorrar espacio las letras P, S, C, R y T (por sus siglas en inglés) representan a las acciones pagar (pay), enviar (ship), cancelar (cancel), redimir (redeem) y transferir (transfer).

Algunos ejemplos del funcionamiento de este autómata producto son:

- Si el banco recibe un mensaje R cuando esta en el estado 1 se mueve al estado 3. Si lo recibe en los estados 3 o 4 no cambia de estado, y en el estado 2 el autómata deja de funcionar.
- Si el autómata está en el estado (1,b) el arco maracado con R llev al autómata al estado (3,d).
- La acción R lleva al autómata desde el estado (4,c) hasta el (4,e).

1.5 Uso del autómata producto para validar el protocolo

La figura 7 muestra algunos hechos importantes. Por ejemplo, de los 28 estados sólo 10 de ellos pueden ser accedidos desde el estado inicial (1,a). Los estados (2,e) y (4,d) no son accesibles desde el estado inicial, por lo que no es necesario incluirlos en el autómata.

Sin embargo, el propósito real de analizar un protocolo como el proceso de compra usando un autómata es responder preguntas como "es posible que ocurra determinado tipo de error?". Siguiendo el ejemplo una pregunta sería si es posible que la tienda envíe un producto sin recibir un pago. Esto es, si el autómata de producto puede llegar a un estado en el qe la tienda haya hecho el envío (estados de las columnas c,e o g) sin que se haya realizado o se pueda realizar una transición para la entrada T.

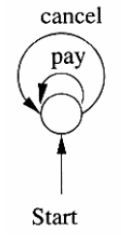

Figure 3: Autómata finito del cliente.

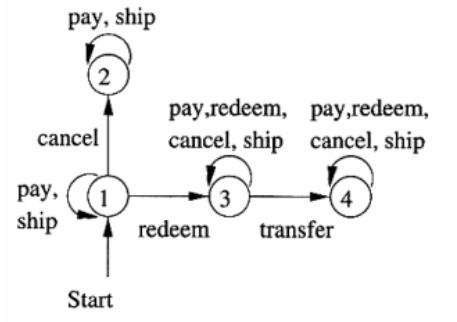

Figure 4: Autómata finito de banco aumentado.

2 Autómatas finitos deterministas

Ahora se presentará la noción formal de un autómata determinista. El término determinista se refiere al hecho de que para cada entrada hay uno y solo un estado al que el autómata puede cambiar. Por el contrario, un autómata no determinista, puede estar en varios estados al mismo tiempo. Para referirse a un autómata determinista se usará la abreviatura DFA.

2.1 Definición de un autómata finito determinista (DFA, por sus siglas en Inglés)

Un autómata determinista consiste de:

- Un estado finito de estados Q.
- Un estado finito de símbolos de entrada Σ.
- Una función de transición δ que toma como argumentos un estado y un símbolo de entrada y regresa un estado.
- Un estado inicial perteneciente a Q.
- Un conjunto de estados finales F, correspondientes a un subconjunto de Q.

Un DFA se representa por la tupla:

$$A = (Q, \Sigma, \delta, q_0, F) \tag{1}$$

2.2 Como un DFA procesa cadenas

Lo primero que necesitamos entender sobre n DFA es cómo decide si acepta o no una cadena de símbolos de entrada. El lenguaje de un DFA es el conjunto de todas las cadenas que acepta. Supongamos que $a_1, a_2...a_n$ es una secuencia de símbolos de entrada pertenecientes a Σ. Comenzamos por el estado inicial q_0 y consultamos la función de transición δ, digamos $\delta(q_0, a_1) = q_1$ para conocer el estado al que el DFA

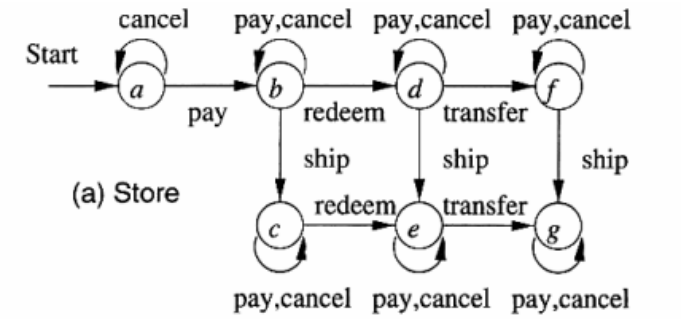

Figure 5: Autómata finito de la tienda aumentado.

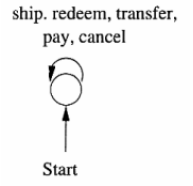

Figure 6: Autómata finito del cliente aumentado.

cambia después de procesar el símbolo de entrada a_1. Continuamos procesando $\delta(q_{i-1}, a_i) = q_i$ para cada i. Si el estado final $q_n \in F$ entonces la cadena $a_1, a_2...a_n$ es aceptada, de lo contrario es rechazada.

2.3 Notaciones simples para DFA

Especificar los DFA como un tupla es tedioso y poco intuitivo, por eso hay mejores alternativas:

- Diagramas de transición, como el de la figura 4.
- Tablas de transición, que es un listado

Diagramas de transición
Un diagrama de transición para un DFA $A = (Q, \Sigma, \delta, q_0, F)$ es un grafo definido de la siguiente forma:

- Para cada estado en Q hay un nodo.
- Para cada estado $q \in Q$ y para cada símbolo de entrada $a \in \Sigma$ existe $\delta(q, a) = p$. Entonces el diagrama de transición tiene un arco del nodo q al nodo p etiquetado a.
- Hay una flecha en el estado inicial q_0 llamada Start que no se origina en ningún nodo.
- Los nodos correspondientes a los estados finales F están marcados con un doble círculo.

Tablas de Transición
Una tabla de transición es una representación tabular de una función como δ que tiene dos argumentos de entrada y devuelve un valor. Las filas de la tabla corresponden a los estados y las columnas a las entradas. El estado inicial se indica con una flecha y los estados finales con un asterisco.

Table 1: Ejemplo de tabla de transición.

	0	1
$\rightarrow q_0$	q_2	q_0
$*q_1$	q_1	q_1
q_2	q_2	q_1

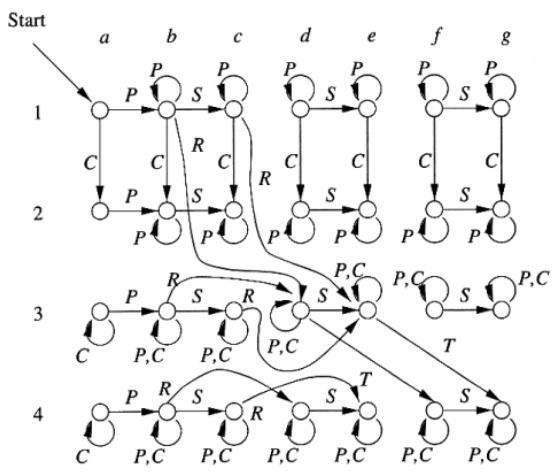

Figure 7: El autómata producto para la tienda y el banco.

2.4 Extensión de la función de transición a cadenas

Se ha explicado informalmente que un DFA define un lenguaje: el conjunto de todas las cadenas que resultan en una secuencia de transiciones de estado desde el estado inicial a un estado final. En términos de un diagrama de transición, el lenguaje de un DFA es el conjunto de todas las etiquetas a través de todos los caminos que van desde el estado inicial al estado final. Ahora definiremos una función de transición extendida $\hat{\delta}$ que acepta un estado q y una cadena w y regresa un estado p; el estado alcanzado después de procesar toda la cadena w. Ahora definiremos $\hat{\delta}$ por inducción en función del tamaño de la cadena.

BASE: $\hat{\delta}(q, \epsilon)$. Lo que significa que si estamos en un estado q y no tenemos ninguna entrada, entonces nos mantenemos en el estado q.

INDUCCIÓN: suponemos que w es una cadena de la forma xa; donde a es el último símbolo de w, y x es la cadena consistente de todos los símbolos menos el último. Por ejemplo, $w = 1101$ se separa en $x = 110$ y $a = 1$. Entonces

$$\hat{\delta}(q, w) = \delta(\hat{\delta}(q, x), a) \tag{2}$$

Ahora, según 2 para calcular $\hat{\delta}(q, w)$, primero calculamos $\hat{\delta}(q, x)$ correspondiente al estado en el que el autómata se encuentra después de procesar todos los símbolos a excepción de w, el último de la cadena. Supongamos que este estado es p; que corresponde a, $\hat{\delta}(q, x) = p$. Entonces $\hat{\delta}(q, w)$ es lo que obenemos al realizar una transición del estado p con una entrada a, que es el último símbolo de w. Esto es, $\hat{\delta}(q, w) = \delta(p, a)$.

2.5 El lenguaje de un DFA

Ahora podemos definit el lenguaje de un DFA $A = (Q, \Sigma, \delta, q_0, F)$. Este lenguaje se denomina $L(A)$ y está definido como

$$L(A) = \{w | \hat{\delta}(q_0, w) \in F\} \tag{3}$$

Lo que significa que el lenguaje de A es el conjnto de las cadenas w que pueden llevan a un estado inicial q_0 a un estado final. Si L es $L(A)$ para algún DFA A entonces decimos que L es un lenguaje regular.

3 Autómata finito no determinista (NFA, por sus siglas en Inglés)

Ahora definiremos un autómata finito no determinista y demostraremos que cada uno acepta un lenguaje que es también aceptado por algún DFA. Un NFA es más fácil de diseñar que un DFA,y por otro lado,

Table 2: Representación tabular de un NFA que acepta todas las cadenas que terminan en 01.

	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
$*q_1$	$\emptyset$	$\{q_2\}$
q_2	$\emptyset$	$\emptyset$

aunque siempre podemos convertir un NFA en un DFA, un DFA puede tener una cantidad exponencial mayor de estados.

3.1 Definición del NFA

Un NFA se representa esencialmente de la misma manera que un DFA:

$$A = (Q, \Sigma, \delta, q_0, F). \tag{4}$$

Donde:

- Q es un conjunto finito de estados.
- Σ es un conjunto finito de símbolos.
- $q_0 \in Q$ es el estado inicial.
- F, un conjunto de Q, es conjunto de estados finales.
- δ es la función de transición que toma un estado en Q y un símbolo de entrada de Σ y regresa un subconjunto de Q, a diferencia de un DFA, que regresa un estado en lugar de un conjunto de estados.

El NFA de la figura 8 se especifica formalmente con: $(\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_2\})$. La representación tabular del mismo autómata se muestra en la tabla 2. La única diferencia que hay entre una tabla de transición para un DFA y una de un NFA es que cada celda contiene un conjunto de estados con al menos un elemento.

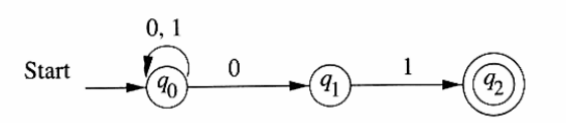

Figure 8: Un ejemplo de un NFA que acepta todas las cadenas que terminan en 01.

3.2 La función de transición extendida

De la misma forma que con un DFA, es necesario extender la función de transición δ de un NFA a una función $\hat{\delta}$ que tome un estado q y una cadena w y devuelva un conjunto de estados en los que el NFA estará si empieza en el estado q y procesa la cadena w. Formalmente definimos la función $\hat{\delta}$ a partir de la función δ de un NFA mediante inducción de la siguiente forma:

BASE: $\hat{\delta}(q, \epsilon) = \{q\}$. Esto es, sin leer ningún símbolo de entrada el autómata no se mueve del estado en el que empezó.

INDUCCIÓN. Suponer que w es de la forma $w = xa$, donde a es el estado final de w y x es el resto de w. De igual forma suponer que $\hat{\delta}(q, x) = \{p_q, p_2, ..., p_k\}$. Entonces:

$$\cup_{i=1}^{k} \delta(p_i, a) = \{r_1, r_2, ..., p_k\} \tag{5}$$

Así $\hat{\delta}(q, w) = \{r_1, r_2, ..., r_m\}$. Lo que quiere decir que calculamos $\delta(\hat{q, w})$ calculando primero $\hat{\delta}(q, x)$ y después siguiendo calquier transición de estos estados que tenga una etiqueta a.

Table 3: La tabla completa de transiciones del autómata de la figura 8

	0	1
$\emptyset$	$\emptyset$	$\emptyset$
$\rightarrow \{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1\}$	$\emptyset$	$\{q_2\}$
$*\{q_2\}$	$\emptyset$	$\emptyset$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$*\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
$*\{q_q, q_2\}$	$\emptyset$	$\{q_2\}$
$*\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

3.3 El lenguaje de un NFA

Como se ha sugerido anteriormente, un NFA acepta una cadena w si es posible hacer cualquier secuencia de elecciones para el siguiente estado, mientras se leen los caracteres de w, y se va desde el estado de inicio a un estado final. El hecho de que alguna de las alternativas que usen los sómbolos de w lleven al autómata a un estado no final, no implica que w sea aceptado por el NFA como un todo. Formalmente si $A = (Q, \Sigma, \delta, q_0, F)$ es un NFA, entonces:

$$L(A) = \{w | \hat{\delta}(q_0, w) \cap F \neq \emptyset\} \tag{6}$$

Y $L(A)$ es el conjunto de cadensa $w \in \Sigma*$ tal que $\hat{\delta}(q_0, w)$ contiene al menos un estado final.

3.4 Equivalencia entre autómatas finitos deterministas y no deterministas

Aunque hay muchos lenguajes para los que un NFA es más fácil de construir que un DFA, es sorprendente el hecho de que cada lenguaje que puede ser descrito por un NFA puede ser descrito por un DFA también. En el peor de los casos el DFA más pequeño tendrá 2^n estados, mientras que su correspondiente NFA tendría n estados.

La prueba de que un DFA puede cumplir la misma tarea que un NFA incluye una "construcción" llamada construcción de subconjuntos porque implica la construcción de todos los subconjuntos de un conjunto de estados del NFA. Es importante notar que la construcción de un subconjunto es un ejemplo cómo formalmente describir un autómata en términos de los estados y las transiciones de otro, sin conocer los detalles específicos de este último.

La construcción de un subconjunto comienza con un NFA $N = (Q_n, \Sigma, \delta_n, q_0, F_N)$. El objetivo es la descripción de un DFA $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ tal que $L(D) = L(N)$. Es importante notar que los alfabetos de entrada de los dos autómatas son iguales, y que el estado inicial de D es el conjunto que contiene únicamente el estado inicial de N. Los otros componentes de D son los siguientes.

- Q_D es el conjunto de subconjuntos de Q_N i.e, Q_D es el conjunto potencia de Q_N. Los estados inaccesibles pueden ser eliminados, así efectivamente el número de estados en D puede ser mucho menor que 2^n.
- F_D es el conjunto de subconjuntos S de Q_n tal que $S \cap F_N \neq \emptyset$. That is, F_D son todos los conjuntos de los N estados que incluyen al menos un estado final de N.
- Para cada conjunto $S \subseteq Q_N$ y para cada símbolo de entrada $a \in \Sigma$, $\delta_D(S, a = \cup_{p \in S} \delta_N(p, a)$. Esto significa que para calcular $\delta_D(S, a)$ buscamos todos los estados p en S, analizamos cuáles son los N estados que van desde p con una entrada a, y tomamos la unión de todos esos estados.

  Para todo DFA y su correspondiente NFA se cumplen los siguientes teoremas.

  Teorema 1 *Si $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ es el DFA construido a partir de un NFA $N = (Q_n, \Sigma, \delta_n, q_0, F_N)$ mediante la construcción de sunconjuntos, entonces $L(D) = L(N)$.*

  Teorema 2 *Un lenguaje L es aceptado por un DFA sólo y sólo si L es aceptado por algún NFA.*

4 Una aplicación: búsqueda en texto

En esta sección se utilizará la teoría de autómatas para la tarea práctica de encontrar palabras en un texto.

4.1 Encontrar cadenas en textos usando NFA

Un problema común que puede solucionarse con autómatas es la búsqueda de texto: dado un conjunto de palabras, encontrar todos los documentos que contienen una (o todas) las palabras en cuestión. Las características que hacen que una aplicación de búsqueda de texto sea apropiada para resolverse con autómatas son:

- Los valores en el repositorio debe estar cambiando rápidamente, como en el caso de los valores asociados a los "tickers" de alguna compañía en la bolsa de valores.
- Los documentos sobre los que se realizará una búsqueda no pueden ser catalogados. Por ejemplo Amazon hace fácil el poder indexar su catálogo mediante un crawler, pues las páginas se generan al vuelo en base a consultas. Pero podríamos generar una búsqueda específica y buscar las revisiones "excelentes" en la página de un producto.

4.2 Autómatas finitos no deterministas para búsqueda en texto

Queremos encontrar una serie de keywords en un texto. En una aplicación como esta una manera útil de proceder es diseñar un NFA que entra a un estado final al encontrar una keyword. Se puede usar un NFA de forma sencilla para reconocer un conjunto de keywords.

- Hay un estado inicial con una transición a sí mismo por cada símbolo de entrada (cada carácter ASCII imprimible si estamos examinando texto). Intuitivamente el estado inicial representa una suposición de que no hemos comenzado a ver una de las keywords, aún cuando ya hayamos visto alguna letra de estas palabras.
- Para cada keyword $a_1, a_2...a_k$, hay k estados, digamos $q_1, q_2, ...qk$. Hay una transición desde el estado inicial hacia q_1 en el símbolo a_1, otra transición en q_2 en el símbolo a_2 y así sucesivamente. El estado q_k es un estado final e indica que la palabra $a_1, a_2...a_k$ ha sido encontrada.

El NFA de la figura 9 encuentra las palabras web y ebay.

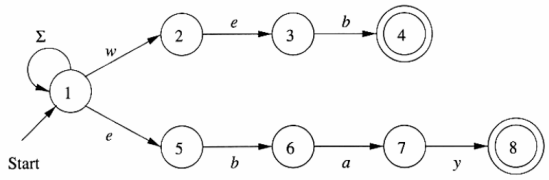

Figure 9: Un ejemplo de un NFA que acepta todas las cadenas ebay y web.

4.3 Un DFA para reconocer un conjunto de palabras clave

Podemos aplicar la construcción de sbconjuntos a cualquier NFA. Sin embargo, cuando aplicamos esa construcción a un NFA que fue diseñado desde un conjunto de keywords, de acuerdo a la estrategia descrita en la sección anterior, encontramos que el número de estados del DFA nunca es mayor qe el número de estados del DFA. Ya que en el peor caso el número de estados crece exponencialmente al hacer la transformación a un DFA, esta observación es una buena noticia y justifica por qué el método de diseñar un NFA para keywords para luego construir un DFA es una técnica usada frecuentemente. Las reglas para construir el DFA son las siguientes:

1. Si q_0 es el estado inicial del NFA, entonces $\{q_0\}$ es uno de los estados del DFA.

2. Si suponemos que P es uno de los estados del NFA, y se puede llegar a él desde el estado inicial a través de un camino cuyos símbolos son $a_1, a_2...am$. Entonces uno de los estados del DFA es el conjunto de estados del NFA que incluyen:

   - q_0.
   - p.
   - Todos los otros estados del NFA que son alcanzbales dede q_0 siguiendo un camino cuyas etiquetas son cualquier secuencia de símbolos de la forma $a_j a_{j+1}...a_m$.

Es importante notar que habrá un estado del DFA por cada estado p del NFA. Sin embargo, en el paso 2, dos estados pueden dar lugar el mismo conjunto de estados del NFA, y por lo tanto, convertirse en un estado del DFA.

5 Autómatas finitos con Transiciones Epsilon

Ahora introduciremos otra extensión del autómata finito. La nueva característica es que ahora se permiten transiciones en ϵ, correspondientes a la cadena vacía. De la misma forma que el no determinsmo, esta nueva característica no aumenta la clase de lenguajes que pueden ser aceptados por el autómata finito, pero si genera da lugar a cierta conveniencia.

5.1 Usos de las transiciones epsilon

Ahora se dará un tratamiento informal de un autómata finito con transiciones epsilon ($\epsilon - NFA$) mediante diagramas de transición con ϵ como etiqueta permitida. Se puede pensar que el autómata que recibe caracteres ϵ los considera como invisibles, es decir, no contribuyen en nada a la cadena a lo largo del camino desde el estado inicial al estado final.

En la figura 10 se muestra un ϵ-NFA que acepta números decimales consistentes de un signo + o − opcional, una cadena de dígitos, un punto decimal y otra cadena de dígitos. Es posible que alguna de las dos cadenas de dígitos sea vacía.

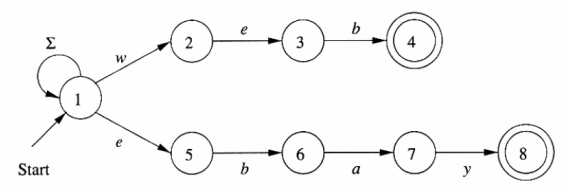

Figure 10: Un ϵ-NFA que acepta números decimales.

Es de particular interés la transición de q_0 hacia q_1 para cualquier valor ϵ, + o -. Así, el estado q_1 representa la situación en la que hemos visto el signo, si existe, pero no hemos visto el punto decimal. El estado q_2 representa la situación donde hemos visto el signo decimal, sin importar si hemos visto dígitos anteriormente. En q_4 hemos definitivamente visto por lo menos un dígito pero no el punto decimal. Por lo tanto, la interpretación de q_3 es que hemos visto el punto decimal y por lo menos un dígito, después o antes del punto decimal.

5.2 La notación formal para ϵ-NFA

Podemos representar un ϵ-NFA igual que un NFA pero con una excepción: la función de transición debe incluir información acerca de las transiciones ϵ. Formalmente representamos un ϵ-NFA mediante $A = (Q, \sigma, \delta, q_0, F)$, donde todos los componentes tienen la misma interpretación que en un NFA, a excepción de δ que es ahora una función que toma como argumentos:

1. Un estado Q y
2. Un miembro de $\Sigma \cup \{\epsilon\}$, correspondiente a un símbolo de entrada o el símbolo ϵ.

5.3 Cerraduras-epsilon

Ahora procederemos a dar una deficinición formal de una función de transición extendida para los ϵ-NFA, lo que llevará a la definición de las cadenas de aceptación y lenguajes para este autómata, y eventualmente definir porqué un ϵ-NFA no puede ser simulado mediante un DFA. Antes de eso hay que definit el concepto de cerradura-ϵ. Informalmente cerramos con ϵ un stado q siguiendo todas las ransiciones que salen de q etiquetadas con ϵ. Sin embargo, cuando llegamos a otros estados siguiendo ϵ, seguimos también las transiciones ϵ desde esos estados hasta encontrar todos los estados qe pueden ser alcanzados desde q a través de cualquier arco etiquetado con ϵ.

Formalmente definimos la cerradura ϵ ECLOSE(q) recursivamente de la siguiente forma:

BASE: el estado q es ECLOSE(q)

INDUCCIÓN: si el estado p está en ECLOSE(q), y hay una transición desde el estado p al estado r etiquetado como ϵ, entonces r está en ECLOSE(q). Más precisamente, si δ es la función de transición del ϵ-NFA involucrado, y p está en ECLOSE(q), entonces ECLOSE(q) también contiene todos los estados en $\delta p, \epsilon$.

5.4 Transiciones extendidas y lenguajes para ϵ-NFA

Las cerraduras ϵ nos permiten explicar fácilmente cómo se ven las transiciones de un ϵ-NFA al recibir una secuencia de entradas no ϵ. Desde ahí es posible definir qué significa para un ϵ-NFA aceptar esa entrada.

Suponer que $E = (Q, \Sigma, \delta, q_0, F)$ es un ϵ-NFA, primero definimos $\hat{\delta}$, la función de transición extendida, para reflejar lo que sucede al recibir una secuencia de entrada. La ides es que $\hat{\delta}(q, w)$ es la secuencia de estados que puede ser alcanzada a través de un camino cuando sus etiquetas, concatenadas, forman la cadena w. Las ϵ a lo largo de esta camino no contribuyen a w. La definición recursiva de $\hat{\delta}$ es:

BASE: $\hat{\delta}(q, \epsilon) = ECLOSE(q)$. Lo que quiere decir que si la etiqueta del camino es ϵ, entonces podemos seguir sólo arcos etiquetados con ϵ saliendo desde q; pues exactamente lo que una cerradura ϵ hace.

INDUCCION: Suponer que w es de la forma xa, donde a es el último símbolo de w. Notar que a es un miembro de Σ; que no puede ser ϵ. Entonces calcularmos $\hat{\delta}()q, w$ de la siguiente forma:

1. Permitir que $\{p_q, p_2, ..., p_k\}$ sea $\hat{\delta}(q, x)$. Lo que quiere decir que las p_i sib tidis kis estais que podemos alcanzar desde q siguiendo un camino etiquetado como x. Este camino puede terminar con uno o más transiciones etiquetadas como ϵ, y puede tener otras transiciones ϵ también.

2. Dejar que $\cup_{i=1}^{k} \delta(p_i, a)$ sea el set $\{r_q, r_2, ..., r_m\}$. Esto corresponde a seguir todas las transiciones etiquetadas como a desde los estados que podemos alcanzar desde el estado q a través de todos los caminos etiquetados como x. Las r_j son algunos de los estados que podemos alcanzar desde q a través de los caminos etiquetados como w. Los estados adicionales que podemos alcanzar pueden ser encontrados desde las r_j siguiendo todos los arcos etiquetados como ϵ del siguiente paso.

3. Entonces $\hat{\delta}(q, x)(q, w) = \cup_{j=1}^{m} ECLOSE(r_j)$. Este paso adicional incluye todos los caminos desde q etiquetados como w, considerando la posibilidad de qe haya arcos etiquetados con ϵ que podemos seguir después de hacer una transición en el último símbolo real a.

5.5 Eliminando transiciones-ϵ

Dado cualquier ϵ-NFA E, podemos encontrar el DFA D que acepta el mismo lenguaje E. La construcción que usaremos es similar a la construcción de subconjuntos, dado que los estados de D son subconjuntos de los estados de E. La única diferencia es que debemos incorporar las transiciones ϵ, lo que se logra a través las cerraduras ϵ.

Si $E = (Q, \Sigma, \delta_E, q_0, F_E)$, entonces el DFA equivalente es $D = (Q_D, \Sigma, \delta_D, q_D, F_D)$ y se define de la siguiente forma:

1. Q_D es el conjunto de subconjuntos de Q_E.

2. $q_E = ECLOSE(q_0)$; lo que significa que empezamos del estado inicial de D cerrando el conjunto consistente únicamente del estado iniical de E.

3. F_D son los conjuntos de estados que contienen por lo menos un estado final de E.

4. $\delta_D(s, a)$ se calcla para todos los estados $a \in \Sigma$ y para todos los conjuntos S en Q_D.

[1]

References

[1] George B. Dantzig. Reminiscences about the origins of linear programming. *Operations Research Letters*, 1(2):43–48, 1982.