Tecnológico nacional de México

Instituto Tecnológico de Iztapalapa

“Lenguajes y Autómatas 1”

Ávila Gómez Giovanni Arturo

181080041

García Pacheco Axel David

181080026

Rojas Pérez José Ramón

181080011

“REPORTE TEORICO”

INTRODUCCIÓN

En este documento analizaremos las 3 áreas centrales de la computación, que son autómata, computabilidad y complejidad.

TEORÍA DEL AUTÓMATA.

La teoría del autómata se ocupa de las definiciones y propiedades de los modelos matemáticos de cálculo. Estos modelos juegan un papel en varias áreas aplicadas de la informática. Un modelo, llamado autómata finito, se utiliza en procesamiento de texto, compiladores y diseño de hardware. Otro modelo, llamado gramática libre de contexto, se usa en lenguajes de programación e inteligencia artificial.

TEORÍA DE LA COMPUTABILIDAD.

Kurt Gödel, Alan Turing y Alonzo Church que ciertos problemas básicos que no se pueden resolver con computadoras. El desarrollo de ideas sobre modelos teóricos de computadoras que eventualmente ayudarían a conducir a la construcción de computadoras reales En la teoría de la computabilidad, la clasificación de los problemas es por aquellos que son solucionables y aquellos que no lo son.

TEORÍA DE LA COMPLEJIDAD.

Los problemas de la computadora vienen en diferentes complejidades, algunos son fáciles y otros son difíciles. Ésta es la cuestión central de la teoría de la complejidad. Sorprendentemente, no conocemos la respuesta. En un logro importante de la teoría de la complejidad hasta ahora, los investigadores han descubierto un elegante esquema para clasificar los problemas.

NOCIONES MATEMÁTICAS Y TERMINOLOGÍA.

CONJUNTOS.

Un conjunto es un grupo de objetos representados por una unidad. Los conjuntos contienen pueden contener cualquier tipo de objetos incluidos números, símbolos y otros conjuntos. Los objetos en el conjunto serán llamados elementos o números.

Su simbología:

- {} conjunto
- ∈ Es un elemento del conjunto o pertenece al conjunto.

- ∉ No es un elemento del conjunto o no pertenece al conjunto.
- I Tal que.
- n (C) Cardinalidad del conjunto C.
- U Conjunto Universo.
- Φ Conjunto Vacío.
- ⊆ Subconjunto de.
- ⊂ Subconjunto propio de.
- ⊄ No es subconjunto propio de.
- > Mayor que.
- < Menor que.
- ≥ Mayor o igual que.
- ≤ Menor o igual qe.
- ∩ Intersección de conjuntos.
- ∪ Unión de Conjuntos.

Secuencias y Matrices.

La secuencia es una lista de objetos en un orden específico. Cómo en el producto cartesiano, que consiste en la multiplicación de dos o más elementos, como ejemplo A X B.

Si A = {1, 2} y B = {x, y, z}

A X B = { (1, x), (1, y), (1, z), (2, x), (2, y), (2, z) }.

Las matrices son secuencias finitas.

Funciones y relaciones.

Una función es un objeto que establece una relación de entrada y salida. Una función toma una entrada y produce una salida. Si *f* es una función cuyo valor salida es B y cuando el valor de entrada es A.

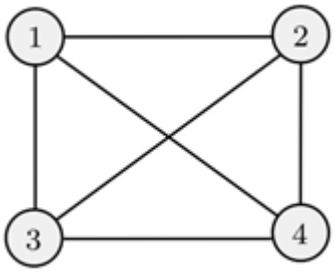
F(a) = b.

El conjunto de posibles entradas a la función de denomina dominio. La salidas de una función provienen de un conjunto llamado rango. La notación para decir que *f* es una función con dominio *d y* rango *r es:*

$f : D \rightarrow R.$

Grafos.

Un gráfico simple o de una dirección es un conjunto de puntos con algunas líneas conectadas. Los puntos son llamados nodos o vértices y las líneas son llamadas aristas.



El número de aristas es un nodo particular es un grado de ese nodo.

CADENAS Y LENGUAJES.

La cadena de caracteres son fundamentales para la creación de bloques en la ciencia de la computación. El alfabeto está sobre las cadenas que varían dependiendo la aplicación. Para nuestro propósito definimos un alfabeto es un conjunto finito no-vacío los miembros del alfabeto son símbolos del alfabeto. Generalmente usamos letras griegas Σ y Γ.

Lógica Booleana.

Es un sistema matemático construido alrededor de dos valores, verdadero y falso. Los valores verdadero y falso, son definidos valores booleanos y a menudo representados por los valores 1 y 0. Operadores lógicos.

Definiciones, teoremas y pruebas.

Son el corazón y el alma de las matemáticas y las definiciones son el espíritu.

Definiciones: describen los objetos y nociones que usamos.

Prueba: Es un argumento lógico convincente que una formación es verdadera.

Teorema: Es un enunciado matemático que su prueba es totalmente verdad.

Tipos de pruebas

Pruebas de construcción: Muchos teoremas establecen un tipo particular de objetos existentes.

Pruebas de contradicción: En una forma común, el argumento para probar un teorema asumimos que el problema es falso hasta que probamos lo contrario, eso es una contradicción.

Pruebas de inducción: Es un método avanzado que se usa para mostrar todos los elementos que tiene un conjunto infinito de propiedades específicas.

VIDEO 1 “INTRODUCCIÓN”

Modelos de computación este curso es principalmente para introducir y estudiar los modelos fundamentales de computación:

Autómatas finitos

Autómatas de empuje

Máquina de Turing

Lenguajes formales

Los problemas que pretendemos cumplir se describirán como "lenguajes formales".

Esperamos una computadora abstracta que pueda reconocer / aceptar el lenguaje (calcular / resolver el problema)

Clasificación

Los problemas más fáciles a los más difíciles y los respectivos idiomas

Luego, problemas insuperables / incuestionables y los respectivos idiomas.

● La computación puede ayudar a resolver problemas a través de la ejecución mecánica y programada de una serie de pequeños pasos inequívocos.

● El cálculo se puede realizar mediante una computadora.

● La teoría de la computación comprende las propiedades matemáticas fundamentales del hardware y software de la computadora.

● Las ciencias teóricas de la computación son estructuras de datos y algoritmos, teoría de la computación, y semántica de programas

● Para abordar varios aspectos se requieren modelos de cálculo, como un resumen a través de la cual podemos demostrar prueba matemática de las afirmaciones.

● Los modelos fundamentales de cómputo son Autómatas finitos, Autómatas pushdown y máquina de turing.

● Los problemas más comunes en la computación se dan independientemente de los idiomas.

● Todos los autómatas son básicamente dispositivos informáticos abstractos, modelos de computación que calculan y aceptan diferentes tipos de lenguajes, pueden resolver diferentes idiomas.

● Los autómatas finitos son los más simples de todos.

VIDEO 2 - ALFABETO, CADENAS Y LENGUAJES

Algunas propiedades de los idiomas

Las propiedades teóricas con respecto a la unión, intersección , complemento 0 , diferencia etc. incluso en el contexto de las lenguas.

Otras propiedades son con respecto a la concentración de operaciones introducidas,kleene, cierre y cierre positivo.

P1 Es la concatenación de las lenguas es asociativa

p2 Es el puesto que la concatenación de las cadenas no son conmutación. tenemos L1,L2 ≠L2,L1 en general

p3 L{e}={e}L=l

p4 LØ = ØL = Ø Proof

p5 propiedades de Distribución

Nos fijamos en los idiomas para que la finita representación sea posible

Dado un alfabeto los idiomas con solo cadena x ya pueden tener representación finita es decir x y 0, respectivamente.

Por lo tanto dando una representación finita para los finitos idiomas es un problema interesante no interesante, este contexto las operaciones en idiomas pueden ser útiles.

Usando la operación estrella kleene podemos tener representación de algunos lenguaje infinitos.

Para dar una representación finita para los idiomas uno puede primero mirar las lenguas indivisibles.

En una expresión regular mantenemos un mínimo número de paréntesis que se requieren para evitar ambigüedades en la expresión.

Si r en una expresión regular, entonces el lenguaje representado por r se denota mediante L(r).

Se dice que un lenguaje L es regular si hay una expresión regular r tal que L-L(r)

Un lenguaje regular sobre un alfabeto es el que se puede obtener del cmptyset.

Glosario:

Teoría de la complejidad: Es el uso de la teoría de la complejidad en el campo de la gestión estratégica y los estudios organizacionales.

Ramificaciones: Parte secundaria de una cosa que nace o deriva de otra cosa principal.

Códigos secretos; Puede tratarse de una combinación de símbolos que, en el marco de un sistema ya establecido, cuente con un cierto valor.

Teorías de computabilidad: También denominada teoría de la recursión, es una de las cuatro partes que constituyen la lógica matemática.

Teoría autómata: Es una rama de la teoría de la computación que estudia las máquinas abstractas y los problemas que éstas son capaces de resolver.

Conjunto infinito: Es un conjunto que no es finito.

Diagrama de Venn: Son esquemas usados en la teoría de conjuntos.

Círculos superpuestos: Es cuando dos círculos se unen entre sí .

Secuencia de objetos: Es un tipo de diagrama usado para modelar la interacción entre objetos en un sistema según UML.

Función: Es un objeto que configura una relación insumo-producto.

Notificación infija: Es la notación común de fórmulas aritméticas y lógicas, en la cual se escriben los operadores entre los operandos en que están actuando.

Función unaria: Encontrar funciones que corresponden a estados de sí o no, o bien activo o inactivo.

Función k-ary: Es un anillo de k nodos. Sin pérdida de generalidad, colocamos los k nodos en una línea.

Relación binaria: Es un subconjunto del producto cartesiano X × Y ; es decir, que es un conjunto de pares ordenados.

Gráfico no dirigido: Un conjunto de objetos (llamados vértices o nodos) que están conectados entre sí, donde todos los bordes son bidireccionales.

Gráfico etiquetado: Es un gráfico cuyos vértices tienen asignado un elemento de un conjunto de símbolos (letras, por lo general, pero esto no es importante).

Ciclo simple: También llamado bucle en la programación.

Gráfico dirigido: Es un tipo de grafo en el cual las aristas tienen un sentido definido.

Lógica Booleana: es una lógica de conjuntos y nos sirve, principalmente, para definir formas de intersección entre conjuntos.

Valores Booleanos: tipo de datos que tiene uno de dos valores posibles (generalmente denotados como verdadero y falso).

K-tuple: Una lista de k objetos

Dirección contraria: Esta herramienta permite la traducción de direcciones IP a nombres. Cada dirección IP puede tener solo un nombre asociado. Este nombre es el registro inverso de la dirección

Paso de inducción: es un razonamiento que permite demostrar proposiciones que dependen de una variable {\displaystyle n\,}que toma una infinidad de valores enteros

Teorema de Ramsey's: establece que en cualquier esquema de color aplicado a un grafo de un grafo completo suficientemente grande.

REPRESENTACIÓN FINITA

Discutimos algunas propiedades de los lenguajes algunas de las propiedades que ya hemos discutido

Por ejemplo, las propiedades teóricas del conjunto habituales con respecto a la unión, intersección, el complemento, la diferencia, etcétera, se mantienen incluso en el contexto de algunas lenguas así que estamos interesados en algunas otras propiedades con respecto a la operación es recién instruidas

Autómata: Conjunto de estados + Control → Cambio de estados en respuesta a una entrada.

Tipo de Control:

• Determinístico: Para cada entrada, hay solo un estado al que el autómata puede ir desde el estado en que se encuentre.

• No determinístico: Un autómata finito es no-determinístico cuando se permite que el AF tenga 0 o más estados siguientes para cada par estado-entrada

Definición Formal de un Autómata

Un AF se representa como una 5-tupla: A = (Q, Σ, δ, q0, F). Donde:

1 Q: Un conjunto finito de estados.

2 Σ: Un conjunto finito de símbolos de entrada (alfabeto)

3 q0: El estado inicial/de comienzo.

4 F: cero o más estados finales/de aceptación.

5 δ: Función de transición. Esta función:

Toma un estado y un símbolo de entrada como argumentos.

Regresa un estado.

Una “regla” de δ se escribe como δ (q, a) = p, donde q y p son estados y a es un símbolo de entrada.

Intuitivamente: Si el AF está en un estado ´ q, y recibe una entrada a, entonces el AF va al estado p (nota: puede ser al mismo estado q = p).

Ejemplo:

(L_1 U L_2) L_3=L_1 L_3 UL_2 L_3

X€(L_1 UL_2)^(L_3)

X=X_1 X_2 X_1 €L_1 UL_2 X_2 €L_3

X=X_1 X_(2) X_1 €L_1 or X_1 €L_2 X_2 €L_3

X=X_1 X_(2) X_1 €L_1 and X_2 €L_3 or X_1 €L_(2) and 〖X〗_2 €L_3

X€L_1 L_3 or X€L_2 L_3

X€L_1 L_3 U X€L_2 L_3

X€L_1 L_3 U X€L_2 L_3

Lenguajes de un NFA

-NFA es una quíntupla (Q, Σ, δ, q0, F), donde δ es una función de Q × Σ ∪ {} al conjunto potencia de Q.

• Hay que evitar que sea parte del alfabeto Σ

El lenguaje aceptado por un NFA, A, es: L(A) = {w|δˆ (q0, w) ∩ F 6= ∅}.

Equivalencia entre un DFA y un NFA Un NFA es normalmente más fácil de definir, aunque al mismo tiempo, para cualquier NFA N existe un DFA D tal que L(D) = L(N) y viceversa. Para esto se usa la construcción de subconjunto que muestra un ejemplo de cómo un autómata se puede construir a partir de otro.

Construcción de Subconjunto

• Lo cual es un DFA (simplemente cambiando la notación). También es importante notar que no todos los estados pueden ser alcanzados. En particular, solo los estados B, E y F son accesibles, por lo que los demás los podemos eliminar.

• Una forma de no construir todos los subconjuntos para después encontrar que solo unos cuantos son accesibles, es construir la tabla solo para los estados accesibles (lazy evaluation).

EJEMPLO:

Para el ejemplo anterior: δD({q0}, 0) = {q0, q1}

δD({q0}, 1) = {q1}

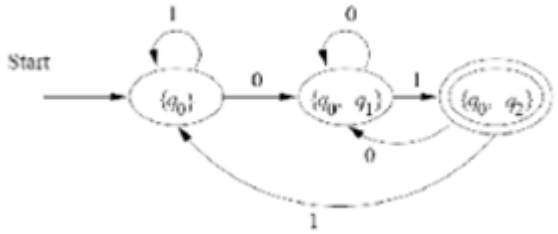
δD({q0, q1}, 0) = {q0, q1}

δD({q0, q1}, 1) = {q0, q2} = δN(q0, 1) ∪ δN(q1, 1)

δD({q0, q2}, 0) = {q0, q1}

δD({q0, q2}, 1) = {q0}

Lo que nos queda:



Veremos cómo se pueden representar los lenguajes usando información infinita y qué significa la representación finita de las lenguas, hay muchos interesados en una representación finita de un idioma, si una persona es competente en un idioma en particular, no significa que produce todas las oraciones de los lenguajes. Básicamente lo que esperamos es que utilizando una cantidad finita de información queremos poder validar o construir cosas de diferencia de los idiomas, esto significa que al dar una cantidad finita de información todas las cadenas de los idiomas se enumeran o validad.

Por ejemplo, sirve el caso del compilador, el compilador puede validar cualquier programa que no sea más que una cadena del lenguaje de programación usando sólo unas instrucciones finalmente válidas.

GRAMATICAS

Dado un conjunto de símbolos o alfabeto V, el conjunto de todas cadenas posibles sobre ese alfabeto es V* (el monoide libre sobre V mediante el operador concatenación) y V+ será el mismo conjunto sin la cadena vacía. Un lenguaje sobre el alfabeto V es un subconjunto de V*. Existen un número no enumerable de lenguajes, todos ellos representables mediante gramáticas.

Una gramática se define como la cuádrupla G=(N,V,P,S), donde V es el alfabeto de G y N es un conjunto finito de símbolos no terminales, V[intersection]N=Ø. Se conoce a W=VuN como el vocabulario de G, una frase [zeta][propersubset]W* es una cadena de símbolos de W. PcW*NW*xW* es un conjunto de reglas de producción o reescritura, que transforman una frase (en la que por lo menos hay un no terminal) en otra frase del mismo vocabulario. Estas reglas se denotan como [zeta]1A[zeta]2->[zeta]3, donde [zeta]1,[zeta]2,[zeta]3[propersubset]W* y A[propersubset][Nu]. Finalmente, S[propersubset]N es el símbolo no terminal inicial o axioma de G.

Una gramática es una estructura algebraica formada por cuatro elementos fundamentales:

G = { NT, T, S, P }

Donde:

• NT es el conjunto de elementos No Terminales

• T es el conjunto de elementos Terminales

• S es el Símbolo inicial de la gramática

• P es el conjunto de Reglas de Producción

Al proceso de obtener una derivación de una cadena a partir del axioma de una gramática y aplicando reglas de ésta, se le denomina análisis sintáctico. Se dice que una gramática es ambigua si para alguna cadena del lenguaje existe más de una posible derivación para generarla.

Ejemplo:

Derivación en G: S ⇒ McF ⇒ aMcAF ⇒ aMcaF ⇒ aaMcAaF ⇒ aaMcaAF ⇒ aaMcaaF ⇒ aabMcBaaF ⇒ aabMcaBaF ⇒ aabMcaaBF ⇒ aabMcaabF ⇒ aabcXaabF ⇒ aabcaXabF ⇒ aabcaaXbF ⇒ aabcaabXF ⇒ aabcaab

Chomsky dividió las gramáticas (y por lo tanto los lenguajes) en una jerarquía que va del tipo 0 a 3, en orden decreciente de complejidad, y en base a la forma de sus reglas:

0 o “No restringida o recursivamente enumerables"

$$x \rightarrow y$$
$$x \in (NT/T)^{+}$$
$$y \in (NT/T)^{*}$$

• “x puede ser sustituido por y si x está, ya sea, en los símbolos No Terminales o los símbolos Terminales, sin incluir la cadena vacía e y está en los símbolos No Terminales o Terminales, incluyendo la cadena vacía.”

• Los lenguajes generados por este tipo de gramáticas se llaman "lenguajes sin restricciones"

• Nota: "+" significa "sin incluir la cadena vacía" y "*" significa "incluyendo la cadena vacía". "/" significa "o"

• Estos lenguajes también son denominados "recursivamente enumerables"

• Las máquinas que los aceptan son las máquinas de Turing (y equivalentes no deterministas.

1 o "Sensible al contexto"

$$\alpha \rightarrow \beta \,;\, |\alpha| \leq |\beta|$$
$$\alpha = z_1 x\, z_2$$
$$\beta = z_1 y\, z_2$$
$$z_1, z_2 \in T^{*}$$
$$x \in NT$$
$$y \in (NT/T)^{+}$$

• “α puede ser reemplazado por β si la longitud de α es menor o igual a la longitud de β, siendo α un símbolo Terminal o una cadena vacía z1, seguido de un símbolo No Terminal X, seguido de otro símbolo Terminal o una cadena vacía z2. En el caso de β, z1 debe ser el mismo símbolo z1 de α seguido de un símbolo No Terminal o Terminal sin ser la cadena vacía, seguido del símbolo z2.”

• Las máquinas que los aceptan son autómatas linealmente acotados(linear-bounded).

2 o "Libre de contexto"

$$x \rightarrow y$$
$$x \in NT$$
$$y \in (NT/T)^*$$

"x puede ser reemplazado por y si x pertenece a los símbolos No Terminales e y es un Terminal o No Terminal, incluyendo la cadena vacía."

Máquinas que los pueden leer:

Máquinas que los aceptan: Autómata a Pila (Pushdown Automaton)

3 o "Regular"

$$\alpha \rightarrow \beta$$
$$\alpha \in NT$$
$$\beta \in \begin{cases} aB \\ Ba \\ b \end{cases}$$
$$B \in NT$$
$$a \in T^+$$
$$b \in T^*$$

También llamada "De contexto regular"

"α puede ser reemplazado por β si α pertenece a los símbolos No Terminales y β es uno de estos 3:

• Un símbolo Terminal no nulo seguido de un No Terminal.

• Un símbolo No Terminal seguido de un símbolo Terminal no nulo.

• Un símbolo Terminal pudiendo ser la cadena vacía."

Máquinas que los aceptan: autómata finito, determinista o no determinista.

ÁRBOLES DE DERIVACIÓN

Un árbol de derivación permite mostrar gráficamente cómo se puede derivar cualquier cadena de un lenguaje a partir del símbolo distinguido de una gramática que genera ese lenguaje.

Un árbol es un conjunto de puntos, llamados nodos, unidos por líneas, llamadas arcos. Un arco conecta dos nodos distintos. Para ser un árbol un conjunto de nodos y arcos debe satisfacer ciertas propiedades:

• Hay un único nodo distinguido, llamado raíz (se dibuja en la parte superior) que no tiene arcos incidentes.

• Todo nodo c excepto el nodo raíz está conectado con un arco a otro nodo k, llamado el padre de c (c es el hijo de k). El padre de un nodo, se dibuja por encima del nodo.

• Todos los nodos están conectados al nodo raíz mediante un único camino.

• Los nodos que no tienen hijos se denominan hojas, el resto de los nodos se denominan nodos interiores.

El árbol de derivación tiene las siguientes propiedades:

• El nodo raíz está rotulado con el símbolo distinguido de la gramática;

• Cada hoja corresponde a un símbolo terminal o un símbolo no terminal;

• Cada nodo interior corresponde a un símbolo no terminal.

Para cada cadena del lenguaje generado por una gramática es posible construir (al menos) un árbol de derivación, en el cual cada hoja tiene como rótulo uno de los símbolos de la cadena.

Ejemplo 1

Árbol de derivación para

Figura 5: interpretado como

Figura 6: interpretado como

Ambigüedades

• Una gramática G es ambigua si existe $x \in L(G)$ con al menos dos árboles de derivación diferentes. Ej: La gramática $S \rightarrow SbS \mid ScS \mid a$ es ambigua

• Equivalentemente, una gramática G es ambigua si existe x∈L(G) con al menos dos derivaciones a la izquierda.

• Un lenguaje L es inherentemente ambiguo si todas las gramáticas para dicho lenguaje son ambiguas. Ej: El lenguaje siguiente es inherentemente ambiguo: L = { abjck | i=j o j=k }

Ejemplo 2

El siguiente lenguaje es inherentemente ambiguo:

Esta gramática es ambigua:

Figura 7:

Eso no garantiza que L sea ambiguo. Por eso debemos comprobar que toda GI que lo represente es ambigua. ¿Cómo? Es bastante complejo, no lo veremos.

Intuición

Siempre encontraremos cadenas de la forma que se puedan generar de dos formas distintas (pertenecen a L por dos "motivos" distintos no disjuntos que no podemos "mzclar" en uno).

GRAMÁTICAS REGULARES

Son una gramática formal (N, Σ, P, S) que pueden ser clasificadas como regular izquierda y regular dercha. Estas gramáticas solo pueden generar a los lenguajes regulares de manera similar a los autómatas finitos y las expresiones regulares.

Una gramática regular derecha es aquella cuyas reglas de producción P son de la siguiente forma:

1. A → a, donde A es un símbolo no-terminal en N y a uno terminal en Σ
2. A → aB, don de A y B pertenecen a N y a pertenece a Σ
3. A → ε, donde A pertenece a N.

Análogamente, en una gramática regular izquierda, las reglas son de la siguiente forma:

1. A → a, donde A es un símbolo no-terminal en N y a uno terminal en Σ
2. A → Ba, donde A y B pertenecen a N y a pertenece a Σ
3. A → ε, donde A pertenece a N.

Las gramáticas formales definen un lenguaje describiendo cómo se pueden generar las cadenas del lenguaje. Una gramática formal es una cuádrupla G = (N, T, P, S) donde

- N es un conjunto finito de símbolos no terminales

- T es un conjunto finito de símbolos terminales $N \cap T = \emptyset$

- P es un conjunto finito de producciones

Cada producción de P tiene la forma $\alpha \rightarrow \beta$, $\alpha = \phi A \rho$ y $\beta = \phi \omega \rho$ $\phi, \omega, \rho \in (N \cup T)^*$ y A es S ó $A \in N$ - S es el símbolo distinguido o axioma $S \notin (N \cup T)$ Restringiendo los formatos de producciones permitidas en una gramática, se pueden especificar cuatro tipos de gramáticas (tipo 0, 1, 2 y 3) y sus correspondientes clases de lenguajes.

Autómata es una máquina matemática M formada por 5 elementos M = (Σ, Q, s, F, δ) donde Σ es un alfabeto de entrada, Q es un conjunto finito de estados, s es el estado inicial, F es un conjunto de estados finales o de aceptación y δ (delta) es una relación de transición.

AUTÓMATAS FINITOS

Un autómata finito tiene un conjunto de estados y su “control” pasa de un estado a otro en respuesta a las “entradas” externas. Una de las diferencias fundamentales entre las clases de autómatas finitos es si dicho control es “determinista”, lo que quiere decir que el autómata no puede encontrarse en más de un estado a un mismo tiempo, o “no determinista”, lo que significa que sí puede estar en varios estados a la vez. Comprobaremos que añadir el no determinismo no nos permite definir ningún lenguaje que no pueda ser definido mediante un autómata finito determinista, aunque se obtiene una eficacia sustancial al describir una aplicación utilizando un autómata no determinista. En efecto, el no determinismo nos permite “programar” soluciones para los problemas utilizando un lenguaje de alto nivel. Entonces el autómata finito no determinista se “compila” mediante un algoritmo especifico, en un autómata determinista que puede “ejecutarse” en una computadora convencional.

Un autómata finito es un modelo matemático de una máquina que acepta cadenas de un lenguaje definido sobre un alfabeto A. Consiste en un conjunto finito de estados y un conjunto de transiciones entre esos estados, que dependen de los símbolos de la cadena de entrada. El autómata finito acepta una cadena x si la secuencia de transiciones correspondientes a los símbolos de x conduce desde el estado inicial a un estado final.

Si para todo estado del autómata existe como máximo una transición definida para cada símbolo del alfabeto, se dice que el autómata es determinístico (AFD). Si a partir

de algún estado y para el mismo símbolo de entrada, se definen dos o más transiciones se dice que el autómata es no determinístico (AFND).

Formalmente un autómata finito se define como una 5-upla

M = <E, A, δ, e0, F> donde

E: conjunto finito de estados

A: alfabeto o conjunto finito de símbolos de entrada

δ: función de transición de estados, que se define como

- δ: E x A → E si el autómata es determinístico

- δ: E x A → P(E) si el autómata es no determinístico (P(E) es el conjunto potencia de E, es decir el conjunto de todos los subconjuntos de E)

e0: estado inicial; e0 ∈ E

F: conjunto de estados finales o estados de aceptación; F ⊆ E

Generalmente se asocia con cada autómata un grafo dirigido, llamado diagrama de transición de estados. Cada nodo del grafo corresponde a un estado. El estado inicial se indica mediante una flecha que no tiene nodo origen. Los estados finales se representan con un círculo doble.

Si existe una transición del estado ei al estado ej para un símbolo de entrada a, existe entonces un arco rotulado a desde el nodo ei al nodo ej; es decir que δ(ei,a) = ej, se representa en el diagrama

Ejemplo 1:

Autómata finito determinístico que acepta el lenguaje

L1 = {an

cbm

/ n > 0 y m ≥ 0 }

M1D = < {e0, e1, e2}, {a, b, c}, δ1D, e0, {e2 }>

δ1D está definida por el siguiente diagrama de transición de estados

Autómata finito “determinista”

Un autómata finito determinista consta de:

1. Un conjunto finito de estados, a menudo designado como Q.

2. Un conjunto finito de símbolos de entrada, a menudo designado como Σ.

3. Una función de transición que toma como argumentos un estado y un símbolo de entrada y devuelve un estado. La función de transición se designa habitualmente como δ. En nuestra representación gráfica informal del autómata, δ se ha representa mediante arcos entre los estados y las etiquetas sobre los arcos. Si q es un estado y a es un símbolo de entrada, entonces δ (q,a) es el estado p tal que existe un arco etiquetado a que va desde q hasta p.

4. Un estado inicial, uno de los estados de Q.

5. Un conjunto de estados finales o de aceptación F. El conjunto F es un subconjunto de Q. A menudo haremos referencia a un autómata finito determinista mediante su acrónimo: AFD. La representación más sucinta de un AFD consiste en un listado de los cinco componentes anteriores. Normalmente, en las demostraciones, definiremos un AFD utilizando la notación de “quíntupla” siguiente: A = (Q,Σ,δ,q0,F) donde A es el nombre del AFD, Q es su conjunto de estados, Σ son los símbolos de entrada, δ es la función de transición, q0 es el estado inicial y F es el conjunto de estados finales.

AUTÓMATA FINITO “NO DETERMINISTA”

Un autómata finito “no determinista” (AFN) tiene la capacidad de estar en varios estados a la vez. Esta capacidad a menudo se expresa como la posibilidad de que el autómata “conjeture” algo acerca de su entrada. Por ejemplo, cuando el autómata se utiliza para buscar determinadas secuencias de caracteres (por ejemplo, palabras clave) dentro de una cadena de texto larga, resulta útil “conjeturar” que estamos al principio de una de estas cadenas y utilizar una secuencia de estados únicamente para comprobar la aparición de la cadena, carácter por carácter.

Al igual que el AFD, un AFN tiene un conjunto finito de estados, un conjunto finito de símbolos de entrada, un estado inicial y un conjunto de estados de aceptación. También dispone de una función de transición, que denominaremos normalmente δ. La diferencia entre los AFD y los AFN se encuentra en el tipo de función δ. En los AFN, δ es una función que toma un estado y símbolos de entrada como argumentos (al igual que la función de transición del AFD), pero devuelve un conjunto de cero, uno o más estados (en lugar de devolver exactamente. Un AFN que acepta todas las cadenas que terminan en 01. Un estado, como lo hacen los AFD).

Los Autómatas Finitos No Deterministas son una herramienta de mucha ayuda cuando queremos diseñar un Autómata Determinista. Para cada Autómata No

Determinista existe un Autómata Determinista que lo representa y que acepta el mismo lenguaje.

Podemos definir un Autómata Finito No Determinista como:

A = {Q,I,F,Σ,δ}

dónde:

Q: Conjunto finito de estados.

I: Conjunto de estados iniciales donde I ∈ Q.

F: Conjunto de estados finales F ⊆ Q.

Σ: Alfabeto finito de entrada.

δ: Función de Transición Q × Σ → S donde S ⊆ Q.

Aparentemente es muy similar a un Autómata Finito Determinista, pero se perciben algunas diferencias. Puede existir más de un Estado inicial [8] y la función de transición δ ahora nos entrega un conjunto, tal vez vacío, de posibles estados. Precisamente esta es la diferencia entre un Autómata Determinista y uno No Determinista. Cuando todas las transiciones están determinadas en un Autómata, es decir para cada par de (estado, símbolo) existe uno y sólo un estado correspondiente, se tiene un Autómata Determinista. Si se tiene al menos una transición no definida o indeterminada entonces tenemos un Autómata No Determinista.

En la función de transición extendida también hay algunos cambios, se puede comenzar en cualquiera de los estados iniciales qI (donde qI ∈ I) y suma todas las posibles transiciones t que inicien en el mismo estado pero que lleven a estados diferentes a pesar que el símbolo leído a es el mismo. Definimos cualquier transición como t = δ(q,a) donde q ∈ Q es un estado y a ∈ Σ es un símbolo. Entonces podemos definir:

δ∗(qI,aw) = P n i=0 δ∗(ti,w)

El número máximo de transiciones para ese par (estado, símbolo) esta´ dado por n, es decir todas las flechas que salen de cualquier estado q y que están etiquetadas por el mismo símbolo a. Para comenzar a evaluar la cadena, donde w ∈ Σ∗, partimos de cualquier estado inicial qI, evaluamos el primer símbolo a y sumamos todas las transiciones posibles, nótese que ahora δ∗ nos entrega un conjunto de posibles estados, es decir δ∗ (qI,aw) → S donde S ⊆ Q se pueden dar los siguientes casos:

· Si se evalúa un solo símbolo δ∗ (q,a) = δ(q,a) el conjunto S que obtenemos tiene un solo elemento que es el estado siguiente.

· Si el símbolo a evaluar es el vacío δ∗ (qi,ε) = qi entonces no existe transición alguna para ese par (q,a), es decir ε se comporta como un valor neutro para el operador δ∗

· Si existen n transiciones para los mismos valores (δ∗(q,a) = q1)+(δ∗(q,a) =q2) + ... + (δ∗(q,a) = qn) entonces el conjunto S contiene los valoresq1,q2,...,qn.

Se dice que una cadena es aceptada si δ∗(qI,aw) = qf donde qf ∈ F.

Un ejemplo de un Autómata Finito No Determinista es la figura 3:

Figura 3: Diagrama de transición de un Autómata Finito No Determinista

Se puede observar que existe más de un estado inicial y las transiciones δ(q1,a2) y δ(q3,a1) no están determinadas, basta solo una de las condiciones anteriores para considerar al Autómata como No Determinista.

Su tabla de transición de la figura 3 es:

	a1	a2
↔q0	q1	q3
q1	q4	∅
←q2	q1	q0
q3	∅	q2
→q4	q2	q3

NFNA, DFA

Este video nos habla de la comparación qué se hizo entre DFA y NFA y se llegó a la conclusión que si le ponían más interacciones en DFA se construyó NFA. Este último siendo mucho más potente o más general que DFA. También se mostró qué DFA y NFA son equivalentes entre dos autómatas en este caso autómata A y autómata A-, tengan el mismo idioma aceptado en ambos autómatas, eso es equivalencia.

entonces mientras se pruebe la equivalencia veremos que una dirección es bastante obvia eso significa que DFA se puede tratar como NFA. DFA se puede tratar como NFA es un caso especial de NFA, para esto NFA será tratado cómo una transición Épsilon, esto lo que haga es construir una NFA equivalente a NFA sin transición Épsilon, Una vez que se realiza eso va a ser equivalente a la NFA de origen y se construirá una DFA equivalente. Eso significa que no habría ningún movimiento de un

estado de símbolo de entrada a varios estados siguientes, tiene que haber un próximo en mi movimiento. Prácticamente este video nos está hablando de él cómo funcionan los autómatas finitos determinísticos y autómatas finitos no determinísticos. esto haciendo una comparación de cómo es que funcionan dentro de un autómata ya que al final también si nosotros agregamos más datos de un DFA podemos construir un NFA, este no será tan funcional Pero podemos hacer que se compare haciéndose un DFA. Esto gracias a la comparación que se está haciendo hasta con el idioma porque es el idioma no coincide ambos estarán mal. (Conjuntos)

Teorema de Nerode

Este vídeo nos habla del teorema de Nerode, primero te dé una pequeña introducción de la noción de los lenguajes formales y luego se busca la representación finita en ese contexto se introdujo la noción que se denomina expresión regular y los respectivos idiomas irregulares esto Gracias a la ayuda de las gramáticas lineales derechas que se llaman gramáticas regulares. Regresamos con el tema anterior DFA y NFA siendo estos dos equivalentes entonces aquí la noción de autómata finito puede ser DFA o NFA esta suposición lleva a capturar la información relacionada con los lenguajes habituales.

Proporciona una condición necesaria y suficiente para qué lenguaje sea regular. Y prácticamente nos está explicando qué va haber un lenguaje l y un par de cadenas de x, y, definimos una extensión que distinga hacer una cadena z tal que exactamente una de las dos cadenas xz, yz pertenece a l esto es una relación de equivalencia de cadenas y por lo tanto se divide el conjunto en todas las cadenas de clase de equivalencia.

Si L es un lenguaje regular, entonces, por definición, hay un DFA A que lo reconoce, con solo un número finito de estados. Si hay n estados, entonces particione el conjunto de todas las cadenas finitas en n subconjuntos, donde el subconjunto S i es el conjunto de cadenas que, cuando se dan como entrada al autómata A , hacen que termine en el estado i . Por cada dos cadenas X y Y que pertenecen al mismo subconjunto, y para cada opción de una tercera cadena z , autómata A alcanza el mismo estado en la entrada XZ , ya que llega en la entrada yz , y por lo tanto debe aceptar ya sea tanto de las entradas xz e yz o rechazar ambos. Por lo tanto, ninguna cadena z puede ser una extensión distintiva de x y y , así que deben estar relacionados por R L . Por lo tanto, s i es un subconjunto de una clase de equivalencia de R L . Combinando este hecho con el hecho de que cada miembro de una de estas clases de equivalencia pertenece a uno de los conjuntos S i , esto da una relación de muchos a uno de los estados de A las clases de equivalencia, lo que implica que el número de clases de equivalencia es finito y como máximo n.

7

MINIMIZACIÓN DE UN AFD

existe una forma de minimizar un AFD. Es decir, podemos tomar cualquier AFD y hallar un AFD equivalente que tenga el número mínimo de estados. En realidad, este AFD es único: dados cualesquiera dos AFD con un número mínimo de estados que sean equivalentes, siempre podemos encontrar una forma de renombrar los estados de manera que ambos AFD se conviertan en el mismo.

Si dos estados no pueden distinguirse mediante el algoritmo de llenado de tabla, entonces los estados son equivalentes. DEMOSTRACIÓN. Supongamos que tenemos el AFD A = (Q,Σ,δ,q0,F). Supongamos también que el teorema es falso; es decir, existe al menos un par de estados {p,q} tal que:

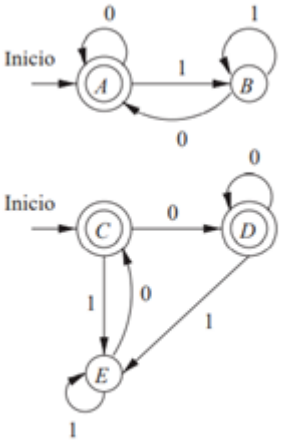
1. Los estados p y q son distinguibles, en el sentido de que existe una cadena w tal que δ(p,w) o δ(q,w) es de aceptación (uno solo de ellos).

2. El algoritmo de llenado de tabla no determina que p y q sean distinguibles.

El problema de minimizar autómatas se reduce al problema de encontrar dichas clases de equivalencia.

Ejemplo

Considere los dos AFD de la Figura 4.10. Cada AFD acepta la cadena vacía y todas las cadenas que terminan en 0; se trata del lenguaje representado por la expresión regular ε + (0+1)∗0. Podemos imaginar que la Figura 4.10 representa un único AFD con cinco estados, A hasta E. Si aplicamos el algoritmo de llenado de tabla a dicho autómata, el resultado es el mostrado en la Figura 4.11.



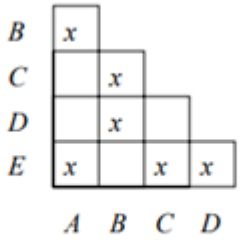


Figura 4.10. Dos AFD equivalentes. **Figura 4.11.** La tabla de estados distinguibles para la Figura 4.10.

Otra importante consecuencia de la comprobación de la equivalencia de estados es que podemos “minimizar” los AFD. Es decir, para cada AFD podemos encontrar otro AFD equivalente que tenga menos estados que cualquier AFD que acepte el mismo lenguaje. Además, excepto por la posibilidad de denominar a los estados con cualquier nombre que elijamos, este AFD con un número mínimo de estados es único para ese lenguaje. El algoritmo es el siguiente:

1. En primer lugar, eliminamos cualquier estado al que no se pueda llegar desde el estado inicial.

2. A continuación, se dividen los restantes estados en bloques, de modo que todos los estados de un mismo bloque sean equivalentes y que no haya ningún par de estados de bloques diferentes que sean equivalentes.

Sería lógico pensar que la misma técnica de partición de estados que sirve para minimizar los estados de un AFD podría aplicarse también para determinar un AFN equivalente con el mínimo número de estados a un AFN o un AFD dado. Aunque es posible, mediante un proceso de enumeración exhaustivo, determinar un AFN con los menos estados posibles que acepte un lenguaje regular dado, no podemos simplemente agrupar los estados del AFN dado.

AFN EQUIVALENTE

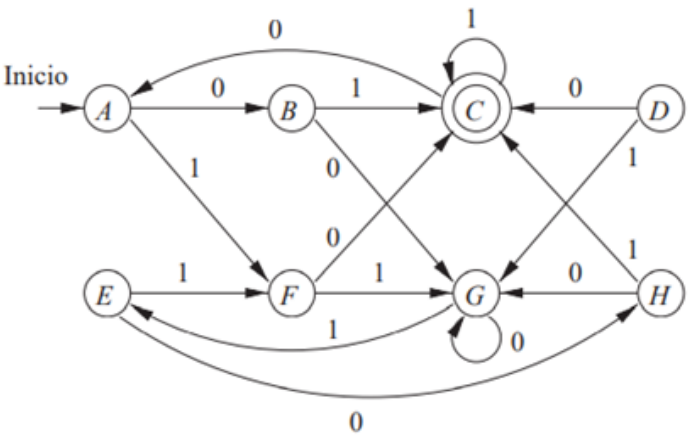


Figura 4.8. Un autómata con estados equivalentes.

EL MISMO AFN MINIMIZADO

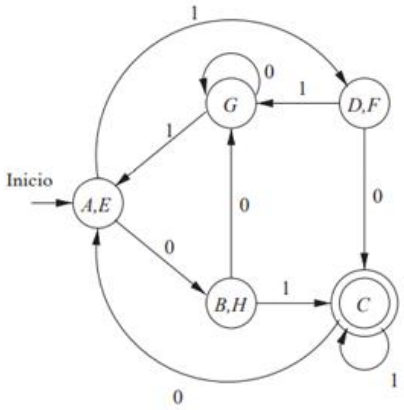


Figura 4.12. AFD con el número mínimo de estados equivalente al de la Figura 4.8.

CONVERSIÓN DE RE A FA

Para convertir el RE en FA, usaremos un método llamado método de subconjunto. Este método se utiliza para obtener FA a partir de la expresión regular dada. Este método se da a continuación:

Paso 1: Diseñe un diagrama de transición para una expresión regular dada, utilizando NFA con movimientos ε.

Paso 2: Convierta este NFA con ε en NFA sin ε.

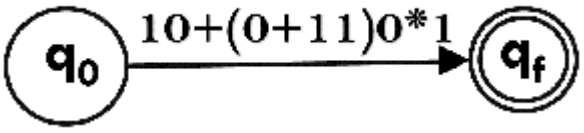
Paso 3: Convierta el NFA obtenido en DFA equivalente.

Ejemplo 1:

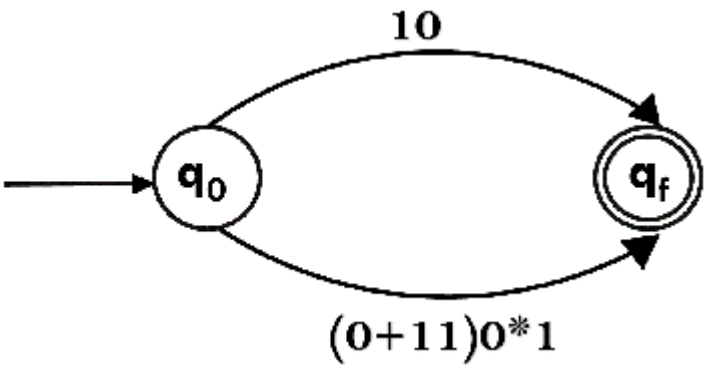
Diseñe un FA a partir de la expresión regular dada 10 + (0 + 11) 0 * 1.

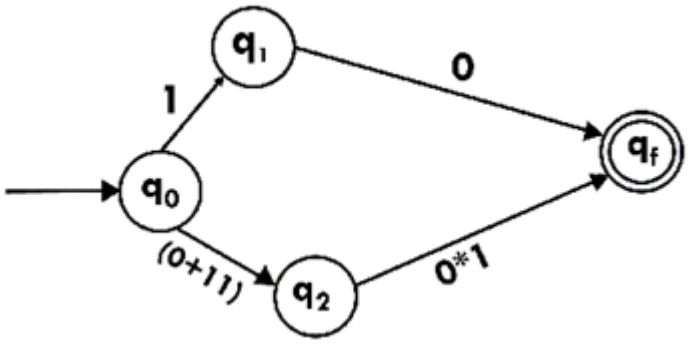
Solución: Primero construiremos el diagrama de transición para una expresión regular dada.

Paso 1:

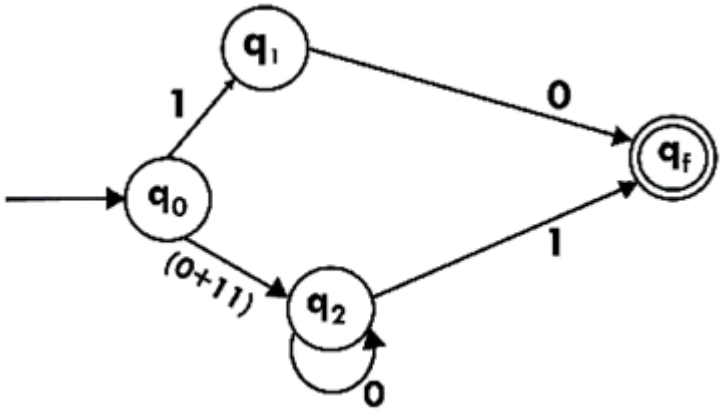


Paso 2:

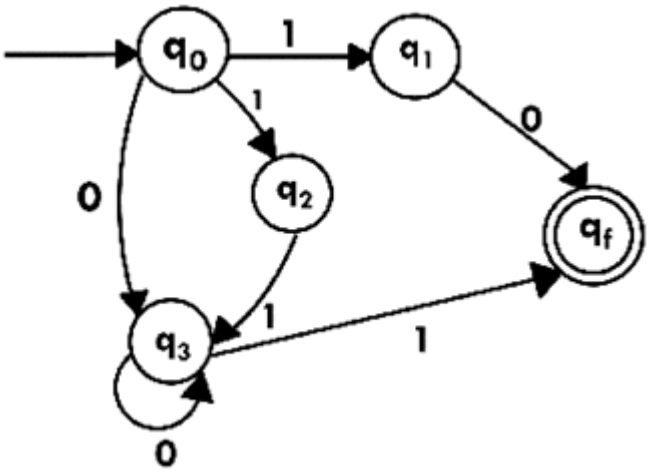




Paso 3:



Paso 4:



Paso 5:

Ahora tenemos NFA sin ε. Ahora lo convertiremos en DFA requerido para eso, primero escribiremos una tabla de transición para este NFA.

Estado	0	1
→ q0	q3	{q1, q2}
q1	qf	ϕ
q2	ϕ	q3
q3	q3	qf
* qf	ϕ	ϕ

El DFA eq

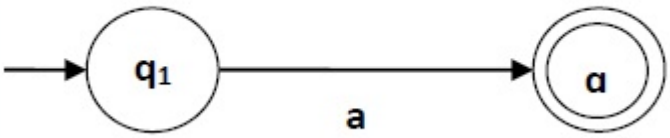
Estado	0	1
→ [q0]	[q3]	[q1, q2]
[q1]	[qf]	ϕ
[q2]	ϕ	[q3]
[q3]	[q3]	[qf]
[q1, q2]	[qf]	[qf]
* [qf]	ϕ	ϕ

CONVERSIÓN DE FA A RE

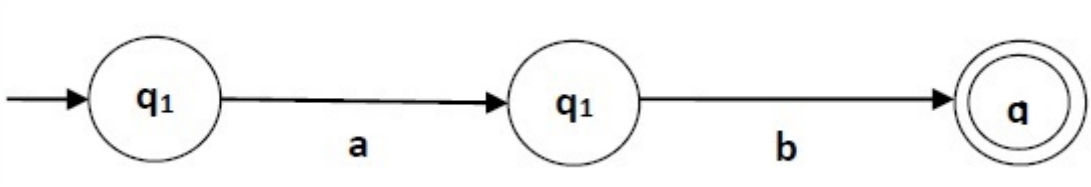
Podemos usar la construcción de Thompson para encontrar un autómata finito a partir de una expresión regular. Reduciremos la expresión regular a expresiones regulares más pequeñas y las convertiremos a NFA y finalmente a DFA.

Algunas expresiones RA básicas son las siguientes:

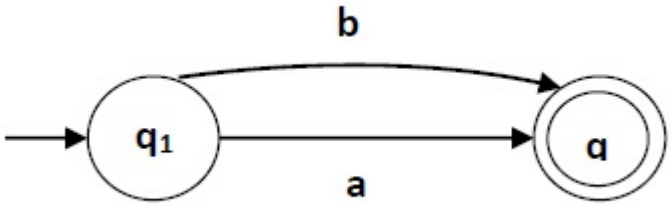
Caso 1 - Para una expresión regular 'a', podemos construir el siguiente FA -



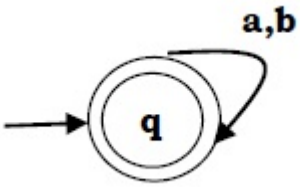
Caso 2 - Para una expresión regular 'ab', podemos construir el siguiente FA -



Caso 3 - Para una expresión regular (a + b), podemos construir el siguiente FA -



Caso 4 - Para una expresión regular (a + b) *, podemos construir el siguiente FA -



Método

Paso 1 Construya un NFA con movimientos nulos a partir de la expresión regular dada.

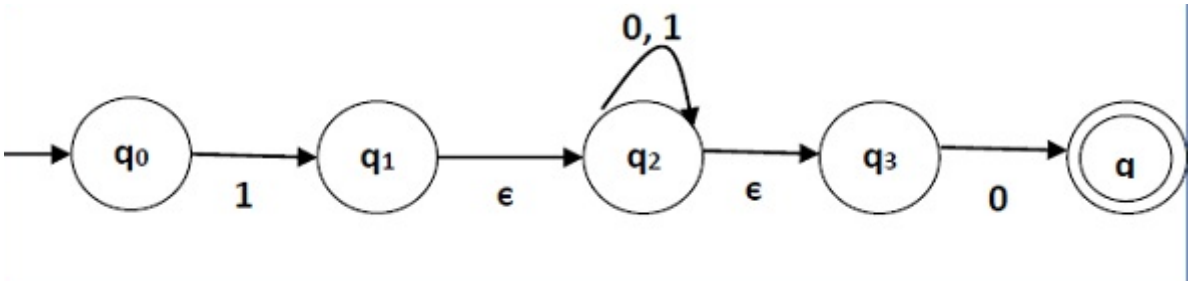
Paso 2 Elimine la transición nula de la NFA y conviértela en su DFA equivalente.

Problema

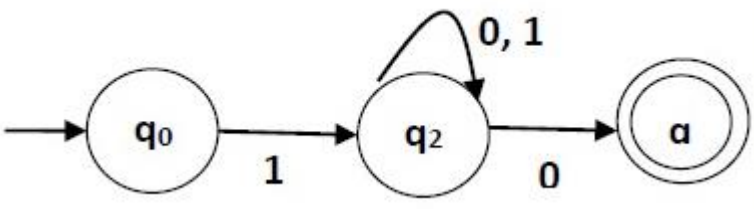
Convierta el siguiente RA en su DFA equivalente - 1 (0 + 1) * 0

Solución

Concatenamos tres expresiones "1", "(0 + 1) *" y "0"



Ahora eliminaremos las transiciones ε . Después de eliminar las transiciones ε del NDFA, obtenemos lo siguiente:



AUTÓMATAS FINITOS Y GRAMÁTICA REGULAR

Para cada alfabeto {L(G): G es una gramática regular de } = {L(M):M es un autómata finito de }.

DEMOSTRACIÓN

1. Si G es una gramática regular de , podemos convertir en una gramática regular G' que genera el mismo lenguaje pero que no contiene reglas de reescritura cuyo lado derecho consiste en un solo terminal. Entonces podemos definir M como un autómata finito no determinista (S, , p, i, F), donde S es la colección de no terminales de G', i es el símbolo inicial de G', F es la colección de no terminales de G' que aparecen en el lado izquierdo de alguna regla c , y p consiste en la tripleta (P, x, Q) para el cual G' contiene una regla de reescritura de la forma PxQ. A la inversa, si M es el autómata finito no determinista (S, ,p ,i ,F), se define G' como la gramática regular de para la cual los no terminales son los estados de S, el símbolo inicial es i, y las reglas de reescritura son de la forma PxQ si (P, x, Q) esta en p y Q c si Q esta en F.

En ambos casos, L(M)=L(G') ya que la derivación de una cadena de la gramática G' corresponde directamente a una ruta en el diagrama de transiciones de M que conduce del estado inicial a un estado de aceptación y viceversa.

EJEMPLO

Sea la gramática regular para el alfabeto S = { x, y } tal que, si x=5 e y=10, representa el lenguaje de todas las cadenas que suman 20:

S ® xX
S ® yY
X ® xY
X ® yZ
Y ® xZ
Y ® y
Z ® x

- Primero tenemos que convertirla en una gramática G' que genera el mismo lenguaje pero que no contiene reglas de reescritura cuyo lado derecho sea solo un terminal. Para eliminar estas reglas se sustituye por dos reglas de reescritura, una cuya parte derecha será siempre un terminal seguido de un no terminal, el cual no existía en la gramática original y la otra cuya parte izquierda es ese no terminal nuevo y la derecha es la cadena vacía.

N® x se sustituye por N ® xX

X ®l

La gramática que nos ocupa tiene las dos últimas reglas de reescritura con un solo terminal y por lo tanto tenemos que convertirla. Para ello vamos a insertar el no terminal Apara la regla vacía; la gramática G' quedará entonces:

S ® xX

S ® yY

X ® xY

X ® yZ

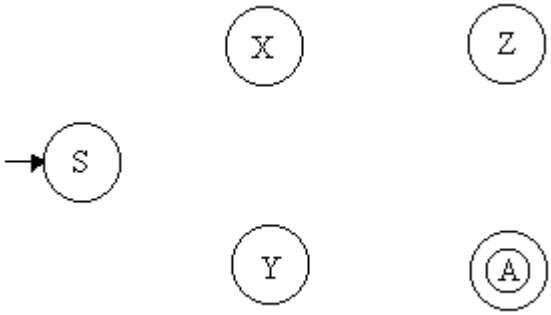
Y ® xZ

Y ® yA

Z ® xA

A ® l

- S es la colección de no terminales de G'. En nuestro caso particular, los estados serán: S = { S, X, Y, Z, A }. (no confundir la S del conjunto -no terminales- con la S del elemento -símbolo de inicio de la gramática-, el cual también conviene cambiar, pero que no se ha hecho por claridad).
- i es el símbolo inicial de G'. Es decir, el estado S
- F es la colección de no terminales de G' que aparecen en el lado izquierdo de alguna regla l. En nuestro caso va a ser sólo el estado A.

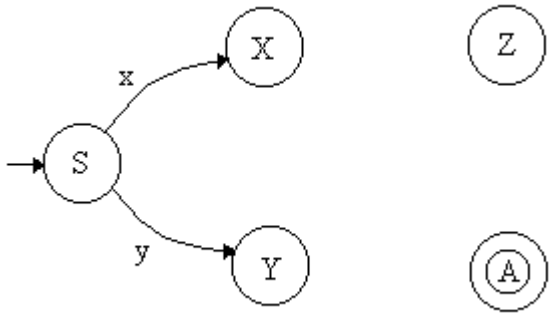


<="" p="">

- r consiste en la tripleta (P, x, Q) (de aquí que el autómata sea no determinista)para el cual G' contiene una regla de reescritura de la forma P® xQ

  + S ® xX : transición del estado S al X con etiqueta x

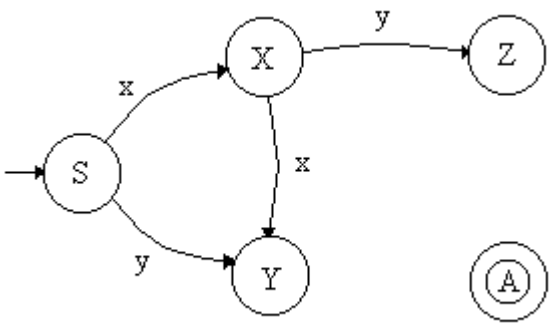
  + S ® yY: Transición del estado S al Y con etiqueta y



<="" p="">

  + X ® xY : Transición del estado X al Y con etiqueta x

  + X ® yZ: Transición del estado X al Z con etiqueta y



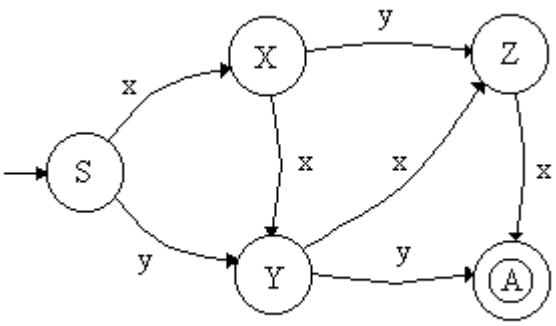
<="" p="">

+ Y ® xZ: Transición del estado Y al Z con etiqueta x

+ Y ® yA: Transición del estado Y al A con etiqueta y

+ Z ® xA: Transición del estado Z al A con etiqueta x

+ A ® l: No hay transición definida



<="" p="">

M,NB I U joRV

ÁLGEBRA DE LAS EXPRESIONES REGULARES.

La conmutatividad es la propiedad de un operador que establece que se puede cambiar el orden de sus operandos y obtener el mismo resultado. Anteriormente hemos dado un ejemplo para la aritmética: x + y = y + x. La asociatividad es la propiedad de un operador que nos permite reagrupar los operandos cuando el operador se aplica dos veces. Por ejemplo, la ley asociativa de la multiplicación es (x×y)×z = x×(y×z). Las tres leyes de este tipo que se cumplen para la expresiones regulares son:

- L+M = M +L. Esta ley, la ley conmutativa de la unión, establece que podemos efectuar la unión de dos lenguajes en cualquier orden.
- (L+M)+N = L+ (M +N). Esta ley, la ley asociativa para la unión, establece que podemos efectuar la unión de tres lenguajes bien calculando primero la unión de los dos primeros, o bien la unión de los dos últimos. Observe que, junto con la ley conmutativa de la unión, podemos concluir que es posible obtener la

unión de cualquier colección de lenguajes en cualquier orden y agrupamiento, y el resultado siempre será el mismo. Intuitivamente, una cadena pertenece a L1 ∪ L2 ∪ ⋯∪ Lk si y sólo si pertenece a uno o más de los Li.

- (LM)N = L(MN). Esta ley, la ley asociativa para la concatenación, establece que podemos concatenar tres lenguajes concatenando primero los dos primeros o bien los dos últimos. Falta en esta lista la "ley" que establece que LM = ML, es decir, que la concatenación es conmutativa. Sin embargo, esta ley es falsa.

ELEMENTO IDENTIDAD Y ELEMENTO NULO

El elemento identidad de un operador es un valor tal que cuando el operador se aplica al propio elemento identidad y a algún otro valor, el resultado es ese otro valor. Por ejemplo, 0 es el elemento identidad para la suma, ya que 0+x = x+0 = x, y 1 es el elemento identidad de la multiplicación, puesto que 1×x = x×1 = x. El elemento nulo de un operador es un valor tal que cuando el operador se aplica al propio elemento nulo y a algún otro valor, el resultado es el elemento nulo. Por ejemplo, 0 es el elemento nulo de la multiplicación, ya que 0 × x = x × 0 = 0. La suma no tiene elemento nulo. Existen tres leyes para las expresiones regulares que implican estos conceptos y que enumeramos a continuación.

- /0+L = L+ /0 = L. Esta ley establece que /0 es el elemento identidad para la unión.
- εL = Lε = L. Esta ley establece que ε es el elemento identidad para la concatenación.
- /0L = L/0 = 0. Esta ley establece que / / 0 es el elemento nulo de la concatenación.

Estas propiedades son importantes herramientas en las tareas de simplificación. Por ejemplo, si tenemos una unión de varias expresiones, algunas de las cuales están simplificadas, o han sido simplificadas, a /0, entonces los 0 pueden eliminarse de la unión.

LEYES DISTRIBUTIVAS

Una ley distributiva implica a dos operadores y establece que un operador puede aplicarse por separado a cada argumento del otro operador. El ejemplo más común en aritmética es la ley distributiva de la multiplicación respecto de la suma, es decir, x×(y+z) = x×y+x×z. Puesto que la multiplicación es conmutativa, no importa que la multiplicación esté a la izquierda o a la derecha de la suma. Sin embargo, existe una ley análoga para las expresiones regulares, que tenemos que establecer de dos formas, ya que la concatenación no es conmutativa. Estas leyes son:

- L(M + N) = LM + LN. Ésta es la ley distributiva por la izquierda de la concatenación respecto de la unión.

- (M +N)L = ML+NL. Ésta es la ley distributiva por la derecha de la concatenación respecto de la unión.

Vamos a demostrar la ley distributiva por la izquierda; la otra se demuestra de manera similar. La demostración sólo hará referencia a lenguajes y no depende de que estos sean regulares.

LEY DE IDEMPOTENCIA

Se dice que un operador es idempotente si el resultado de aplicarlo a dos valores iguales es dicho valor. Los operadores aritméticos habituales no son idempotentes; en general, x + x = x y x × x = x (aunque existen algunos valores de x para lo que se cumple la igualdad, como por ejemplo, 0+0 = 0). Sin embargo, la unión y la intersección son ejemplos comunes de operadores idempotentes. Por tanto, para expresiones regulares, podemos establecer la siguiente ley:

- L + L = L. Ésta es la ley de idempotencia para la unión, que establece que si tomamos la unión de dos expresiones idénticas, podemos reemplazarla por una copia de la de la expresión.

LEYES RELATIVAS A LAS CLAUSURAS

Existe una serie de leyes relacionadas con los operadores de clausura y sus variantes de estilo UNIX + y ?. Vamos a enumerarlas a continuación junto con una breve explicación acerca de por qué son verdaderas. $(L*)* = L*$.

Esta ley dice que clausurar una expresión que ya está clausurada no modifica el lenguaje. El lenguaje de $(L*)*$ está formado por todas las cadenas creadas mediante la concatenación de cadenas pertenecientes al lenguaje $L*$. Pero dichas cadenas están formadas a su vez por cadenas de L. Por tanto, la cadena perteneciente a $(L*)*$ también es una concatenación de cadenas de L y, por tanto, pertenece al lenguaje de $L*$.

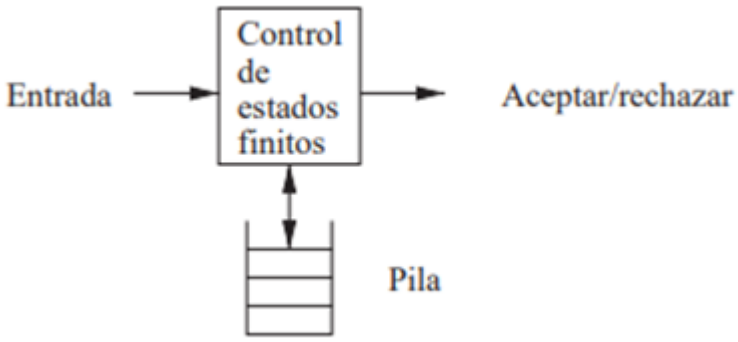
DESCUBRIMIENTO DE PROPIEDADES DE LAS EXPRESIONES REGULARES

Cada una de las leyes anteriores se han demostrado formal o informalmente. Sin embargo, puede proponerse una variedad infinita de propiedades relativas a las expresiones regulares. ¿Existe una metodología de carácter general que facilite la demostración de las propiedades correctas? Basta con reducir la veracidad de una ley a una cuestión de la igualdad de dos lenguajes específicos. Es interesante observar que esta técnica está estrechamente relacionada con los operadores de las

expresiones regulares y que no se puede extender a expresiones que implican otros operadores, como por ejemplo el de intersección.

DEFINICIÓN DE AUTÓMATA A PILA

En esta sección vamos a presentar primero de manera informal el autómata a pila, y después veremos una construcción formal. Fundamentalmente, el autómata a pila es un autómata finito no determinista con transiciones-ε y una capacidad adicional: una pila en la que se puede almacenar una cadena de “símbolos de pila”. La presencia de una pila significa que, a diferencia del autómata finito, el autómata a pila puede “recordar” una cantidad infinita de información.



NOTACIÓN GRÁFICA PARA LOS AUTÓMATAS A PILA

En ocasiones, un diagrama, que generaliza el diagrama de transiciones de un autómata finito, mostrará más claramente aspectos del comportamiento de un determinado autómata a pila. Por tanto, vamos a ver y a utilizar un diagrama de transiciones de un autómata a pila en el que:

- Los nodos se corresponden con los estados del autómata a pila.
- Una flecha etiquetada como Inicio indica el estado inicial y los estados con un círculo doble se corresponden con los estados de aceptación, al igual que en los autómatas finitos.
- Los arcos corresponden a las transiciones del autómata a pila de la forma siguiente: un arco etiquetado con a,X/α del estado q al estado p quiere decir que δ(q,a,X) contiene el par (p,α), quizá entre otros pares. Es decir,la etiqueta del arco nos indica qué entrada se utiliza y también proporciona los elementos situados en la cima de la pila nuevo y antiguo.

DESCRIPCIONES INSTANTÁNEAS DE UN AUTÓMATA A PILA

Hasta el momento, sólo disponemos de una noción informal de cómo “calcula” un autómata a pila. Intuitivamente, el autómata a pila pasa de una configuración a otra, en respuesta a los símbolos de entrada (o, en ocasiones, a ε), pero a diferencia del autómata finito, donde el estado es lo único que necesitamos conocer acerca del mismo, la configuración del autómata a pila incluye tanto el estado como el contenido de la pila. Siendo arbitrariamente larga, la pila es a menudo la parte más importante de la configuración total del autómata a pila en cualquier instante. También resulta útil representar como parte de la configuración la parte de la entrada que resta por analizar. Por tanto, representaremos la configuración de un autómata a pila mediante (q,w, γ), donde:

- q es el estado.
- w es lo que queda de la entrada.
- γ es el contenido de la pila.