Autómatas Finitos

Matemáticas Computacionales

Dr. Víctor de la Cueva

vcueva@itesm.mx

Teoría de autómatas

- Es el estudio de los dispositivos computacionales abstractos (máquinas) especiales llamados máquinas de estados finitos.
- Antes de que hubiera computadoras, en los 30's, A. Turing estudió una máquina abstracta que tenía todas las capacidades de las computadoras actuales, al menos en cuanto a lo que podían computar.
- En los 40s y 50s, los investigadores iniciaron el estudio de una máquinas muy simples llamados "Autómatas Finitos".
- Estos autómatas fueron propuestos originalmente para modelar el funcionamiento del cerebro pero resultaron ser de gran utilidad en un gran número de problemas.

Sistemas de estados finitos

- Un Autómata Finito (FA) es un modelo matemático de un sistema, con entradas y salidas discretas.
- El sistema puede estar en uno (cualquiera) de un número finito de configuraciones internas llamadas "estados".
- El estado del sistema sumariza la información concerniente a las entradas pasadas, que es necesaria para determinar el comportamiento del sistema para entradas subsecuentes.

Ejemplo: Elevador

- El mecanismo de control de un elevador es un buen ejemplo de un sistema de estados finitos.
- El mecanismo no recuerda todas las llamadas de servicio previas.
- Sólo recuerda:
  - El piso actual
  - La dirección de movimiento (arriba o abajo)
  - La colección de llamadas de servicio no atendidas

Uso de autómatas finitos

- Son muy útiles para modelar varios tipos de hardware y software.
  - Software para diseñar y verificar el comportamiento de circuitos digitales.
  - El “analizador de léxico” de un compilador típico, esto es, el componente que descompone el texto de entrada en unidades lógicas tales como: identificadores, palabras reservadas o signos de puntuación (tokens).
  - Software para escanear (analizar) grandes cantidades de texto, tales como colecciones de páginas Web, en busca de ocurrencias de palabras, frases u otros patrones.
  - Software para verificar sistemas de todo tipo que cuenten con un número finito de estados, tales como protocolos de comunicación o protocolos de intercambio seguro de información.

Más ejemplos

- CPU (circuito de switcheo)
- Una computadora. No es muy útil este modelo porque el número de estados del procesador central, la memoria principal y la memoria auxiliar en cualquier tiempo, son un gran número (pero finito).
- Cerebro. El número de neuronas es finito (probablemente 2^{35})

Estados

- Todos los ejemplos anteriores pueden ser vistos como "algo" que está en uno de un número finito de "estados", en cualquier tiempo (Sistemas de Estados Finitos).
- El propósito de un estado es "recordar" la porción relevante de la historia del sistema.
- El número de estados puede ser finito pero su historia es infinita. El sistema no puede recordarla toda y debe ser diseñado cuidadosamente para recordar lo que es importante y olvidar lo que no.
- La ventaja de tener un número finito de estados es que el sistema se puede implementar con una cantidad fija de recursos.

Definiciones básicas

- Un autómata finito (FA) consiste de un conjunto finito de estados y un conjunto de transiciones de un estado a otro que ocurren de acuerdo a símbolos de entrada seleccionados de un alfabeto Σ.
- Para cada símbolo de entrada hay exactamente una transición hacia afuera del estado (posiblemente de regreso al mismo estado).
- Un estado, usualmente denotado por q_0, es el estado inicial, en el cual el autómata inicia.
- Algunos estados son designados como estados finales o de aceptación.

Diagrama de Transición

- Un grafo dirigido, llamado un diagrama de transición, es asociado con un autómata finito como sigue:
  - Los vértices del grafo corresponden a los estados de un FA.
  - Si existe una transición de un estado q a un estado p en una entrada a, entonces, hay un arco etiquetado con una a, del estado q al estado p en el diagrama de transición.
  - El FA acepta un string x si la secuencia de transiciones correspondiente a los símbolos de x lleva al FA de un estado de inicio a un estado de aceptación.

Notación de una diagrama de transición

- Los estados son representados por círculos, con un nombre único.
- Hay arcos dirigidos entre los estados (flechas) que representan transiciones entre los estados.
  - Los arcos son etiquetados por "entradas", las cuales representan influencias externas al sistema.
- Uno de los estados es designado como el estado de inicio (*start state*) y representa el estado en el cual el sistema es colocado inicialmente.
  - Por convención, el estado inicial se indica con una flecha que apunta hacia dicho estado y que se etiqueta con la palabra "Start".
- Normalmente, es necesario indicar uno o más estados "finales" o de "aceptación", los cuales se representan con un doble círculo.
  - El entrar en uno de los estados finales, después de una secuencia de entradas, indica que la secuencia de entrada es "buena", en algún sentido.

Ejemplo: Autómata on/off

- Quizá el más simple de los autómatas finitos no triviales es un switch on/off.
- El dispositivo recuerda si él está en el estado de on o en el estado de off.
- Permite al usuario presionar un botón cuyo efecto es diferente dependiendo el estado en el que se encuentre el switch.
  - Si está en off entonces, al presionar el botón cambia a on
  - Si está en on, al presionar el botón cambia a off

HMU 3

Ejemplo: Reconocimiento de una palabra

- Un uso muy común de los FA es el reconocimiento de palabras (analizador léxico de un compilador).
- Ej, reconocimiento de la palabra reservada “else”.

HMU 3

Ejemplo de strings

- Un FA que acepta todos los strings de 0's y 1's en el cual ambos, tanto el número de 0's como el de 1's, es par.
- Note que el FA usa sus estados para recordar solamente la paridad del número de 0's y el número de 1's, no el número actual de ellos, el cual podría requerir un número infinito de estados.

16AU

Representación Estructural

- Hay dos notaciones importantes que no son como-autómatas, pero que juegan un importante rol en el estudio de los autómatas y sus aplicaciones:
  - Gramáticas son modelos útiles cuando se diseña software que procesa datos con una estructura recursiva. El ejemplo más conocido es un "parser", el componente de un compilador que trata con las características anidadas recursivamente de un lenguaje de programación típico (e.g. expresiones).
  - Expresiones Regulares, que también denotan la estructura de los datos, especialmente strings de texto. Los patrones de strings descritos por las expresiones regulares son exactamente los mismos que pueden ser descritos por los autómatas finitos.

Autómatas y complejidad

- Los autómatas son esenciales para el estudio de los límites de la computación, en dos aspectos muy importantes:
  - ¿Qué puede hacer una computadora? Este estudio es llamado "decidibilidad" y los problemas que pueden ser resueltos por una computadora son llamados "decidibles".
  - ¿Qué puede hacer una computadora eficientemente? Este estudio es llamado "intratabilidad" y los problemas que pueden ser resueltos por una computadora usando no mas tiempo que el descrito por una función de crecimiento lento dependiente del tamaño de la entrada, son llamados "tratables". Normalmente se toma "crecimiento lento" como funciones polinomiales.

Autómatas finitos y lenguajes regulares

- En la sección anterior describimos lo que era un lenguaje regular.
- Estos lenguajes son exactamente los que pueden ser descritos usando autómatas finitos.
- Un autómata finitos tiene un conjunto de estados y un control de movimientos de un estado a otro en respuesta a entradas externas.

Control de movimientos

- Una de las distinciones cruciales entre las clases de autómatas finitos es si el control es:
  - Determinístico, que significa que el autómata no puede estar en más de un estado en cualquier tiempo.
  - No-determinístico, que significa que puede estar en muchos estados a la vez.
- Descubriremos que agregando no-determinismo no nos ayuda a definir ningún lenguaje que no pueda ser definido usando autómatas determinísticos, pero son más eficientes las describir la aplicación.
- El no-determinismo nos ayuda a "programar" la solución en lenguaje de "alto-nivel".
- Posteriormente, el Autómata No Determinístico es "compilado" para transformarlo en uno Determinístico, el cual es ejecutado.

Definición formal de FA

- Formalmente denotamos a un autómata finito por una 5-tupla (Q,Σ,δ,q_0,F), donde:
  - Q es el conjunto finito de estados
  - Σ es el alfabeto finito de entrada
  - $q_0 \in Q$ es el estado inicial
  - $F \subseteq Q$ es el conjunto de estados finales
  - δ es la función de transición mapeando $Q\text{X}\Sigma \rightarrow Q$, esto es, $\delta(q,a)$ es un estado para el estado actual q y un símbolo de entrada a

Extensión de δ

- Para la descripción formal del comportamiento de un FA sobre un string, debemos extender la función de transición δ para ser aplicado a un estado y un string más que a un estado un símbolo.
  - Definimos una función $\hat{\delta}$ de $Q \times \Sigma^* \rightarrow Q$
  - La intención es que $\hat{\delta}(q, w)$ es el estado de un FA en el que estará después de leer un string w, partiendo de q.
  - De otra forma, $\hat{\delta}(q, w)$ es el único estado p tal que hay un camino en el diagrama de transición de q a p, etiquetado con w.

Formalmente

1. $\hat{\delta}(q, \epsilon) = q$, y
2. Para todos los strings w y símbolo de entrada a,

$$\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$$

- (1) sostiene que sin la lectura de un símbolo de entrada el FA no puede cambiar de estado
- (2) nos dice cómo encontrar el estado después de leer un string no vacío wa. Esto es, encuentra el estado, $p = \hat{\delta}(q, w)$, después de leer w. Entonces computa el estado $\delta(p, a)$.
- Por convención, $\hat{\delta} = \delta$, serán lo mismo.

Aceptación de un string

- Un string x se dice aceptado por un autómata finito M=(Q,Σ,δ,q_0,F) si $\delta(q_0, x) = p$ para alguna p en F.
- El lenguaje aceptado por M, designado como L(M), es el conjunto $\{x|\delta(q_0, x)\ está\ en\ F\}$.
- Un lenguaje es un conjunto regular (o sólo regular) si es un conjunto aceptado por algún autómata finito (el término regular viene de las expresiones regulares).
- Cuando se habla de algún lenguaje aceptado por un autómata finito M, nos referimos al conjunto específico L(M), no a todos los strings que sean aceptados por M.

Ej18AU

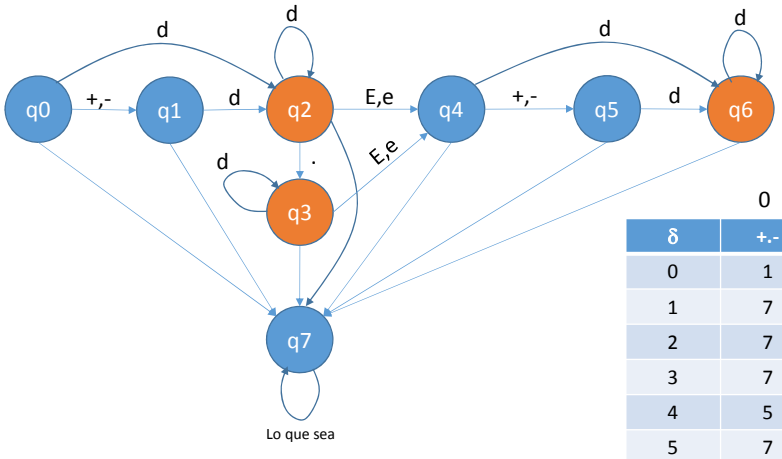
Un programa para aceptar o rechazar

- En todos los autómatas, el proceso para decidir si se acepta o no un string dado es exactamente el mismo y se basa en la función de transición, representada por medio de una tabla de transición.
- Si la información se tiene en un archivo que contenga toda la definición de las 5 partes de un autómata, se puede hacer fácilmente un programa que lo lea, pida un string y conteste si se acepta o no.

Ejemplo

- Hacer un autómata que reconozca si un string dado es un número o no.
- Un número puede ser:
  - Entero. 3569
  - Real: -12.45
  - Real con notación exponencial: 15.42E-12
- Pueden ser con signo (+ o -) o no
- Si el string contiene cualquier caracter que no cumpla con esas reglas, se detectará como rechazado.

Autómata y tabla



	0	1	2	3	4
δ	+.-	.	d	E,e	otro
0	1	7	2	7	7
1	7	7	2	7	7
2	7	3	2	4	7
3	7	7	3	4	7
4	5	7	6	7	7
5	7	7	6	7	7
6	7	7	6	7	7
7	7	7	7	7	7

Ver ejemplo de archivo y código

Autómata finito no determinístico (NFA)

Autómata Finito No Determinístico (NFA)

- En realidad, cualquier conjunto que sea aceptado por un NFA para a ser aceptado por un FA (o DFA).
- Sin embargo, el NFA es un concepto útil en la prueba de teoremas y el concepto de no-determinístico juega un papel fundamental tanto en la teoría de lenguajes como en la teoría de la computación.
- Para crear el nuevo modelo NFA considere modificar el modelo de FA para permitir 0, 1 o más transiciones de un estado con el mismo símbolo de entrada.
- También se pueden formar diagramas de transición para un NFA.

20HU

Secuencia aceptada por un NFA

- Una secuencia de entrada $a_1a_2…a_n$ es aceptada por un NFA si existe una secuencia de transiciones, correspondientes a la secuencia de entrada, que lleva al NFA de un estado inicial a un estado final.
- Ejemplo: Un NFA que acepte strings que tengan dos 0's o dos 1's consecutivos. 20HU
- En un DFA, para un string w y un estado q dados, existe una única transición para cada símbolo. Para determinar si un string es aceptado por un DFA basta con checar si existe este único camino.
- Para un NFA hay muchos caminos etiquetados con w y TODOS deben ser checados para verificar si al menos 1 termina en un estado final. 20HU

Formalmente

- Denotamos un NFA por medio de una 5-tupla $(Q, \Sigma, \delta, q_0, F)$, donde todos los elementos tienen el mismo significado que un FA excepto δ que ahora mapea $Q \times \Sigma \rightarrow 2^Q$, donde 2^Q es el conjunto potencia de Q (el conjunto de todos los posible subconjuntos de Q).
- La intención es que $\delta(q,a)$ es el conjunto de todos los estados p tales que hay una transición de q a p etiquetada con a. 21HU
- La función δ puede ser extendida a $\hat{\delta}$ que mapea de $Q \times \Sigma^* \rightarrow 2^Q$ y se refiere a secuencias de entradas:
  1. $\hat{\delta}(q, \epsilon) = \{q\}$
  2. $\hat{\delta}(q, wa) = \{p | \text{ para algún estado } r \text{ en } \hat{\delta}(q, w), p \text{ está en } \delta(r, a)\}$

Extensión

- Note que $\hat{\delta}(q,a) = \delta(q,a)$ para un símbolo de entrada a.
- De esta forma, nuevamente usaremos el símbolo δ en lugar de $\hat{\delta}$.
- También es útil extender δ a argumentos en 2^Q X Σ^*:

3. $\delta(P,w) = \bigcup_{q\in P} \delta(q,w)$

para cada conjunto de estados P $\subseteq$ Q

L(M) ,donde M es el NFA (Q, Σ, δ, q0, F), es

$$\{w|\delta(q_0,w)\ contiene\ un\ estado\ en\ F\}$$

21HU

Equivalencia de DFAs y NFAs

- **Teorema**: Sea L un conjunto aceptado por un NFA. Entonces, existe un DFA que acepta a L.
- Demostración (construcción): La prueba dependerá de mostrar que los DFAs pueden simular NFAs; esto es, para cada NFA podemos construir un DFA equivalente (uno que acepte el mismo lenguaje)
  - La forma en la que un DFA simula un NFA es permitiendo que los estados de un DFA correspondan a conjuntos de estados del NFA.
  - El DFA que se construye así, realiza un seguimiento en su control finito de todos los estados en los que el NFA podría estar después de leer la misma entrada que el DFA ha leído.

22HU

Creación de un DFA a partir de un NFA

- La demostración anterior proporciona un método para crear un DFA que acepte el mismo lenguaje que un NFA (un DFA equivalente). 23HU
- En la práctica, es común que muchos de los estados de un DFA equivalente no puedan ser alcanzados a partir de $[q_0]$, por lo que no deben estar en la tabla final.
- Se recomienda iniciar con $[q_0]$ e ir agregando estados a la tabla, dependiendo de cuáles sí se pueden alcanzar.

NFA con movimientos-ϵ

- Se puede extender el modelo de NFA para que incluya transiciones con la entrada nula ϵ.
- Ejemplo: Un NFA que acepta el lenguaje consistente en cualquier número de 0's (incluyendo 0), seguido de cualquier número de 1's y seguido de cualquier número de 2's. 24HU
- Desde luego, se pueden incluir aristas etiquetadas con ϵ en el camino de aceptación aún y cuando las ϵ's no aparezcan explícitamente en el string w. 24HU

Formalmente

- Un NFA con ϵ-moves es una quíntupla (5-tupla) (Q,Σ,δ,q_0,F) con todos los componentes definidos igual que para un NFA, pero con δ, la función de transición, que mapea $Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$.
- La intención es que $\delta(q,a)$ consista de todos los estados p tales que hay una transición de p a q, etiquetada con a, donde a es un símbolo de Σ o ϵ.
- Desde luego que se puede crear una tabla de transición para definir δ, donde ahora se agrega ϵ como una posible entrada.

25HU

$\hat{\delta}$

- Como siempre, extendemos δ a una función $\hat{\delta}$ que mapea $Q\times\Sigma^* \rightarrow 2^Q$.
- $\hat{\delta}(q,w)$ debe dar todos los estados p, tales que se pueda ir de q a p con un camino etiquetado w, quizá incluyendo aristas etiquetadas con ϵ.
- Obviamente, en la construcción del $\hat{\delta}$ es importante encontrar el conjunto de estados alcanzables de un estado q dado, usando transiciones ϵ.
- Usamos $\epsilon - CLOSURE$ para denotar el conjunto de todos los vértices p tales que hay un camino de q a p etiquetado ϵ.

ϵ-CLOSURE(q)

- Para extender δ a $\hat{\delta}$ es necesario calcular el conjunto de estados alcanzables a partir de un estado q usando una transición ϵ.
- Se utiliza *ϵ-CLOSURE(q)* para denotar el conjunto de todos los vértices p tales que hay un camino de q a p etiquetado con ϵ. 25HU
- También se puede calcular ϵ-CLOSURE(P), donde P es un conjunto de estados, como:

$$\epsilon - CLOSURE(P) = \bigcup_{q \in P} \epsilon - CLOSURE(q)$$

Definición formal de ϵ-CLOSURE

- Se hace en forma recursiva.
- La ϵ-CLOSURE de un estado q_i, denotada por ϵ-CLOSURE(q_i), es definida recurrentemente por:
  - Base: $q_i \in \epsilon$-CLOSURE(q_i)
  - Recursivo: Sea q_j un elemento de ϵ-CLOSURE(q_i).
    Si $q_k \in \delta(q_j, \in)$, entonces $q_k \in \epsilon$-CLOSURE(q_i).
  - Cerradura: …

Definición de $\hat{\delta}$

- Se puede entonces definir $\hat{\delta}$ como sigue:
  1. $\hat{\delta}(q, \epsilon) = \epsilon - CLOSURE(q)$
  2. Para $w \in \Sigma^*$ y $a \in \Sigma$, $\hat{\delta}(q, wa) = \epsilon - CLOSURE(P)$, donde
     $P = \{p \mid \text{para alguna } r \in \hat{\delta}(q,w),\ p \in \delta(r,a)\}$
- Es conveniente extender δ y $\hat{\delta}$ a conjuntos de estados R como:
  3. $\delta(R, a) = \bigcup_{q \in R} \delta(q, a)$, y
  4. $\hat{\delta}(R, w) = \bigcup_{q \in R} \hat{\delta}(q, w)$
- En este caso $\delta(q,a)$ no necesariamente es igual a $\hat{\delta}(q,a)$ ya que incluye paths etiquetados con a (incluyendo paths etiquetados por ϵ). Ni $\hat{\delta}(q, \epsilon)$ es necesariamente igual a $\delta(q,\epsilon)$.

Función de transición de entrada t

- La función de transición de entrada t de un NFA-$\in$ M es una función de $Q \times \Sigma \rightarrow 2^Q$, definida por:

$$t(q_i, a) = \bigcup_{q_j \in \epsilon - CLOSURE(q_i)} \epsilon - CLOSURE(\delta(q_j, a))$$

Donde δ es la función de transición de M.

Tres partes para t

- $t(q_i, a)$ puede ser separada en 3 partes:
  1. Obtener el conjunto de estados que pueden ser alcanzados de q_i sin procesar un solo símbolo, es decir, con $\in$-*moves*, los cuales NO están en t.
  2. Obtener el conjunto de estados que pueden ser alcanzados al procesar un símbolo *a* de todos los estados del conjunto, los cuales están en t.
  3. Obtener el conjunto de estados a los que se puede llegar con los $\in$-arcos a partir del conjunto obtenido en 2, los cuales, también están en t.

171Su

NOTA: Para un NFA (sin $\in$-moves) la t es la misma que su δ. La función t es usada para construir un DFA equivalente.

Equivalencia entre NFA y NFA-ϵ

- Igual que el no-determinismo, la habilidad para hacer transiciones con ϵ no permite a un NFA aceptar conjuntos no regulares.
- Esto se mostrará simulando un NFA con ϵ-*moves* por medio de un NFA sin tales transiciones.
- **Teorema**. Si L es aceptado por un NFA con ϵ-*moves*, entonces L es aceptado por un NFA sin ϵ-*moves*.
- Demostración: Es por un procedimiento de construcción…

26HU

Construcción de un DFA a partir de un NFA-∈

input: an NFA-λ M = (Q, Σ, δ, q_0, F)
input transition function t of M

1. initialize Q′ to λ-*closure*(q_0)
2. **repeat**
   2.1. **if** there is a node X ∈ Q′ and a symbol $a \in \Sigma$ with no arc leaving X labeled a, **then**
   2.1.1. let $Y = \bigcup_{q_i \in X} t(q_i, a)$
   2.1.2. **if** $Y \notin Q'$, **then** set $Q' := Q' \cup \{Y\}$
   2.1.3. add an arc from X to Y labeled a
   else done := *true*
   until done
3. the set of accepting states of DM is $F' = \{X \in Q' \mid X \text{ contains an element } q_i \in F\}$

27HU
172Su

Tomado de [1] pp. 172

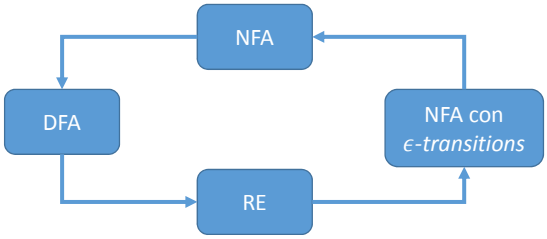
Lenguaje aceptado

- Se define L(M), el lenguaje aceptado por M = (Q, Σ, δ, q_0, F) como $\{w \mid \hat{\delta}(q_0, w) \text{ contiene un estado en } F\}$

26HU

Equivalencia de DFA y RE

- El plan será mostrar por inducción sobre el tamaño (número de operadores en) de una expresión regular que hay un NFA con *ϵ-transitions* denotando el mismo lenguaje.
- Esta demostración, junto con los teoremas de equivalencia anteriores:

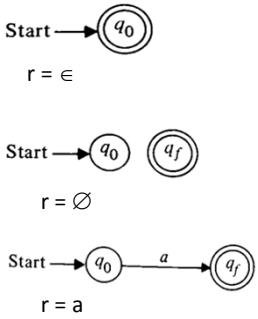


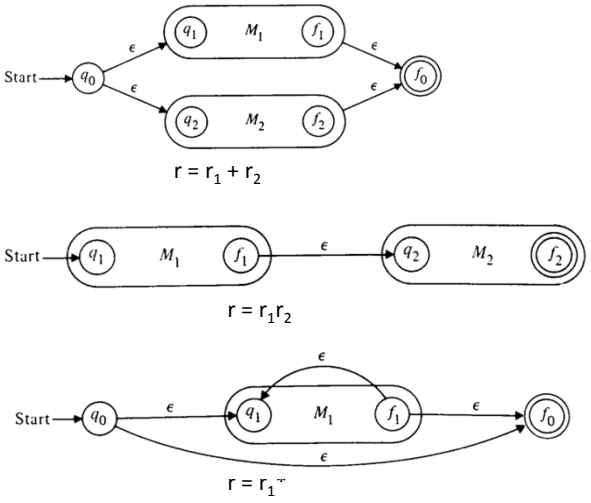
RE a DFA

- **Teorema**. Sea r una expresión regular. Entonces, existe un NFA con *ϵ-transitions* que acepte L(r).
- Demostración:…

30HU

Construcción de un DFA a partir de una RE





32HU

DFA a RE

- **Teorema**: Si L es aceptado por un DFA, entonces, L es denotado por una expresión regular.
- Demostración:…

33HU

Construcción de una RE a partir de un DFA

- La equivalencia de procesos de un DFA y sus caminos nos proporciona un método heurístico para determinar el lenguaje que es aceptado por un DFA.
- Los strings aceptados en un estado q_j (estado final) son precisamente aquellos deletreados por el camino de q_0 a q_i.
- Podemos entonces separar la determinación de esos caminos en dos partes:
  - Encontrar las expresiones regulares $u_1, \ldots, u_n$ para los strings en todos los caminos de q_0 que alcanzan qi por primera vez.
  - Encontrar las expresiones regulares $v_1, \ldots, v_m$ para todos los caminos que dejan q_i y regresan a q_i.
  - Los strings aceptados por q_i son $(u_1 \cup \ldots \cup u_n)(v_1 \cup \ldots \cup v_m)^*$.
- Los strings aceptados por el DFA son la unión de los aceptados por cada estado final q_i.

152Su

Tarea 3

Ejercicios de las páginas 46-50 de [2]:

2.1

2.2

2.4

2.5

2.6

2.8

2.9

2.12

Proyecto 3

Referencias

1. T.A. Sudkamp. Languages and Machines: An Introduction to the Theory of Computer Science. Pearson, 3rd Edition (2005).
2. J.E. Hopcroft, R. Motwani, J.D. Ullman. Introduction to Automata Theory, Languages, and Computation. Pearson, 3rd Edition (2006).