INSTITUTO TECNOLÓGICO DE IZTAPALAPA

INGENIERÍA EN SISTEMAS COMPUTACIONALES

RESUMEN GRUPAL

LENGUAJES Y AUTÓMATAS I

PROFESOR:

M.C. Abiel Tomás Parra Hernández

ALUMNOS:

MENDOZA ZEPEDA VICTOR MANUEL 171080159

OSORIO GARCÍA KEVIN EMMANUEL 171080231

UMEGIDO CAMACHO ERICK JOSUE 171080157

VARGAS CAMACHO PAOLA 171080430

ISC-6AM

Los campos estrechamente relacionados en la informática teórica son el análisis de algoritmos y la teoría de la computabilidad, una distinción clave entre el análisis de algoritmos y la teoría de la complejidad computacional es que el primero se dedica a analizar la cantidad de recursos que necesita un algoritmo en particular para resolver un problema, mientras que el segundo hace una pregunta más general sobre todos los algoritmos posibles que podrían usarse para resolver el mismo problema, para resolver dicho problema, un cálculo resulta complejo si es difícil de realizar, en este contexto podemos definir la complejidad de cálculo como la cantidad de recursos necesarios para efectuar un cálculo.

Así, un cálculo difícil requerirá más recursos que uno de menor dificultad, los recursos comúnmente estudiados son el tiempo (número de pasos de ejecución de un algoritmo para resolver un problema) y el espacio (cantidad de memoria utilizada para resolver un problema, un algoritmo que resuelve un problema pero que tarda mucho en hacerlo, difícilmente será de utilidad, igualmente un algoritmo que necesite un gigabyte de memoria no será probablemente utilizado, a estos recursos se les puede añadir otros, tales como el número de procesadores necesarios para resolver el problema en paralelo, si un cálculo requiere más tiempo que otro decimos que es más complejo y lo llamamos complejidad temporal.

Por otro lado, si un cálculo requiere más espacio de almacenamiento que otro decimos que es más complejo, en este caso hablamos de una complejidad espacial, es claro que el tiempo que se requiere para efectuar un cálculo en un computador moderno es menor que el requerido para hacer el mismo cálculo con las máquinas de las décadas pasadas, y seguramente que el tiempo en una máquina de la próxima generación será mucho menor

La teoría de los lenguajes formales estudian entidades matemáticas abstractas denominadas lenguajes que en ningún momento debemos confundir o equiparar con las lenguas naturales, sin embargo, como veremos, los lenguajes formales pueden, en determinadas circunstancias, servirnos como modelos abstractos de determinadas propiedades de las lenguas naturales, de ahí la importancia de conocer los fundamentos de la teoría, un lenguaje formal es un conjunto, finito o infinito, de cadenas definidas sobre un alfabeto finito pero de infinitas combinaciones.

Hay dos puntos de vista desde los cuales podemos determinar si un lenguaje es interesante o no, por un lado, son interesantes aquellos lenguajes en los que se observa que se sigue alguna pauta regular en la construcción de las cadenas, desde este punto de vista, el lenguaje L1 es digno de estudio, pero no L2:

L1 = {abc, aabbccc, aaabbbccc, ...} L2 = {a, cab, bdac, ...}

Uno de los principales obstáculos a los que se enfrenta cualquiera que desee conocer la complejidad estructural de un lenguaje es el hecho de que esta difícilmente puede deducirse a partir de la simple observación de las cadenas que lo

componen, tampoco resultan particularmente informativos los constructores de conjuntos que podamos utilizar para definir los lenguajes, ya que estos quizá nos permitan hacernos una idea intuitiva de su complejidad, pero en ningún caso podremos establecerla de forma precisa; peor aún, con las herramientas de que disponemos en este momento, nos es totalmente imposible determinar si dos lenguajes L1 y L2 tienen la misma complejidad estructural o no.

Los lenguajes y autómatas están basados en el estudio de dispositivos de cálculo, las máquinas antes de que existiera la computadora como hoy la conocemos, existían máquinas que tenían las capacidades que hoy tienen las computadoras comunes (lo más básico, pero aun así facilitaban los trabajos a los usuarios con fines matemáticos).

Desde un principio de la existencia de las máquinas inteligentes siempre han existido problemas entre creadores, investigadores, usuarios por diversas razones. Siempre se han tenido cuestiones como "¿Son las computadoras todopoderosas?", por estas razones empezaron a crear diferentes tipos de computadoras ya que cada creador quería que su sistema fuera magnífico y el mejor entre ellas, todas tenían y tienen sus ventajas y desventajas

Se hicieron estudios de quien llevaba estos trabajos y se dieron cuenta que eran solo hombres, a lo largo del tiempo ya que las máquinas fueron evolucionando tanto hardware y software, en este tipo de proyectos sí aceptaron las mujeres pero había muy poca colaboración, de ellas hacia las empresas ya que había machismo y no dejaban que las mujeres sacarán a flote sus capacidades y conocimientos.

Para el mejoramiento de ello, hicieron las fases de los compiladores como: el código en lenguaje estructurado, los compiladores, los códigos de lenguaje máquina, los códigos en lenguaje estructurado para entender mucho mejor, las interpretaciones y las instrucciones en lenguaje máquina para pasar a posteriores instrucciones.
Los autómatas finitos tiene muchos modelos útiles tanto para hardware y software, se constituye de identificadores, palabras claves y algunos signos de puntuación, un ejemplo básico de un autómata finito sería el de encendido y apagado.
existen dos tipos de Autómatas clásicos:

- autómata finito determinista (AFD): Con cualquier símbolo del alfabeto, existe siempre no más de una transición posible desde ese estado y con ese símbolo.

- autómata finito no determinista (AFND): Este posee al menos un estado q ∈ Q, tal que para un símbolo a ∈ Σ del alfabeto, existe más de una transición δ(q,a) posible.

Las expresiones regulares, estos le dan un peso algebraico a un autómata. También se utiliza la menera de forma declarativa de expresar cadenas que queremos utilizar

También desde un principio y hasta hoy, es muy valioso para los intereses políticos, financieros y comerciales, ya que esta carrera tiene altas ramas de trabajo desde tiempo atrás y que en este tiempo presente y futuro los ingenieros en sistemas computacionales serán de gran de utilidad para todo el mundo.

Lenguajes regulares resultantes de operaciones

Adicionalmente un lenguaje es **regular** si resultó de una de estas operaciones:

1. *Unión*

   Si L_1 y L_2 son regulares, entonces $L_1 \cup L_2$ es regular también.

2. *Concatenación*

   Si L_1 y L_2 son regulares, entonces $L_1 L_2$ es regular también.

3. *Cerradura estrella*

   Si L es regular, entonces L^* es regular también.

4. *Priorización por paréntesis*

   Si L es regular, entonces (L) es regular también.

Concatenación de dos lenguajes

La concatenación de dos lenguajes da como resultado un nuevo lenguaje de todas las cadenas posibles de concatenas cadedas del primer lenguaje con el segundo lenguaje.

$L_1 L_2 = \{w_1 w_2 | w_1 \in L_1, w_2 \in L_2\}$

Si suponemos $L_1 = \{aa\}$, $L_2 = \{a, b\}$, $L_3 = \{\varepsilon\}$ y $L_4 = \{a, aa, aaa, aaaa, ...\}$

1. $L_1 L_2 = \{aaa, aab\}$
2. $L_2 L_1 = \{aaa, baa\}$
3. $L_2 L_2 = \{aa, ab, ba, bb\}$
4. $L_2 L_3 = \{a, b\}$
5. $L_1 L_4 = \{aaa, aaaa, aaaaa, aaaaaa, \ldots\}$

Teoría de la computación :Habla sobre que se encarga de determinar qué problemas pueden ser resueltos computacionalmente, esta teoría considera los modelos de cómputo, como los autómatas finitos, las máquinas de Turing y los autómatas de pila. También incluye los lenguajes formales que son muy importantes en la teoría ya que estas permiten expresar propiedades de los programas

Modelos de la computación: Autómata finito: Este es un modelo matemático de una máquina que acepta cadena de un lenguaje definido sobre un alfabeto, este acepta una cadena x si la secuencia de transiciones correspondientes a los símbolos de x conduce desde un estado inicial a uno final

Autómata de pila: Son parecidos a uno finito, estos cuentan con una cinta de entrada y un mecanismo de control que puede encontrarse en un uno de entre un número finito de estados

Máquina de Turing: La maquina de Turing es una de los teoremas más importantes sobre las máquinas y esta puede simular el comportamiento de una computadora, por eso si un problema no puede ser resuelto por una máquina de estas entonces no puede ser resuelto por una computadora

Lenguajes formales: Este es un conjunto finito o infinito de cadenas de símbolos primitivos, dichas cadenas están formadas gracias a un alfabeto y a una gramática que están establecidos por reglas.

Tipos de Lenguajes

Lenguajes declarativos: Es fundamentalmente lenguajes de órdenes, dominados por Sentencias que expresan “lo que hay que hacer” en vez de “cómo hacerlo”.

Lenguaje de alto nivel: Son los más utilizados como lenguajes de programación permiten que los algoritmos se expresen en un nivel y estilo de escritura fácilmente legible y comprensible por otros programadores.

Lenguaje ensamblador: Es el programa en que se realiza la tracción de un programa escrito en un programa escrito en ensamblador y lo pasa a lenguaje máquina. Directa o no directa de la traducción en que las instrucciones no son más que instrucciones que ejecuta la computadora.

Lenguaje maquina: Es como la maquina interpreta lo que nosotros queremos hacer es una lectura de 0 y 1 es decir binario

Clasificación: Chomsky clasifica jerárquicamente las gramáticas formales que generan lenguajes regulares en estos tipos:
Tipo 3: Gramáticas regulares que generan lenguajes regulares
Tipo 2: Gramáticas incontextuales que generan lenguajes incontextuales
Tipo 1: Gramáticas contextuales que generan lenguajes contextuales
Tipo 0: Gramáticas libres que generan lenguajes sin ningún tipo de restricción

Lenguajes y representación finita: Una representación finita es un grupo de cadenas de un alfabeto, y usualmente se aplican reglas a estas cadenas, como por ejemplo el lenguaje español que consiste de todas las cadenas de palabras que conocemos en forma de oraciones sumadas de las reglas gramaticales

Gramáticas ambigua: La ambigüedad es un problema muy importante en informática.
No hay métodos automatizados para obtener gramáticas no ambiguas, una gramática es ambigua si existe alguna cadena de terminales que pueda obtenerse mediante árboles de derivación distintos (dos árboles distintos dan la misma cadena)

Teoría de lenguajes formales: La Teoría de los lenguajes formales estudia los lenguajes prestando atención únicamente a sus propiedades estructurales, definiendo clases de complejidad estructural y estableciendo relaciones entre las diferentes clases.

Teoría de la complejidad computacional: La Teoría de la complejidad computacional estudia los lenguajes prestando atención a los recursos que utilizará un dispositivo mecánico para completar un procedimiento de decisión, definiendo así diferentes clases de complejidad computacional y las relaciones que existen entre ellas.
Uno de los roles de la teoría de la complejidad computacional es determinar los límites prácticos de lo que las computadoras pueden y no pueden hacer

Cadenas
Una cadena de un conjunto X es una secuencia finita de elementos de X. Las cadenas son objetos fundamentales usados en la definición de lenguajes. Cadena de caracteres:que también se denomina en ocasiones palabra, es una secuencia finita de símbolos seleccionados de algún alfabeto.Cadena vacía: Es aquella cadena que representa cero apariciones de símbolos. Esta cadena, esta designada por ε , | ε |= 0, es una cadena que puede construirse en cualquier alfabeto.

Longitud de una cadena.
Es el número de posiciones ocupadas por símbolos dentro de la cadena, por ejemplo 01101 tiene una longitud de cinco, es habitual decir que la longitud de una cadena es igual al número de símbolos que contiene.

Traductores: El traductor necesita menos memoria que el compilador.Permite una mayor interactividad con el código en tiempo de desarrollo.

Alfabetos: Es un conjunto finito A. Sus elementos se llamarán símbolos o letras.

Subcadenas: Decimos que una cadena z es subcadena de otra cadena ω si existen cadenas x, y $\epsilon \sum^*$ tal que $\omega = x \cdot z \cdot y$ Prefijo(ω) = $\{x \epsilon \sum^* \mid \exists z \epsilon \sum^* : \omega = x \cdot z\}$
Un prefijo de una cadena x es una subcadena inicial de x, es decir una cadena y es un prefijo si existe una cadena z tal que x = yz y de manera similar z es un sufijo de x.

Los prefijos propios son aquellas cadenas que son prefijos de una palabra pero no iguales a la misma, un prefijo y de x es un prefijo propio de
x si y≠ϵ y y≠x Sufijo(ω) = {x ϵ ∑* | ∃ z ϵ ∑* : ω = z · x}]

Representación finita

Un autómata finito determinista consta de:

Un conjunto finito de estados, a menudo designado como Q.
Un conjunto finito de símbolos de entrada, a menudo designado como Σ.
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.
Un estado inicial, uno de los estados de Q.
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 represen- tación 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.

Gramatica

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.

Mediante la aplicación sucesiva de reglas de G se puede transformar una frase [zeta]1 del vocabulario en otra [zeta]n. Escribiremos esto como [zeta]1[zeta]n; [zeta]1,[zeta]n[propersubset]W*; y si D([zeta]n)=([zeta]1,[zeta]2,[zeta]3,...,[zeta]n), [zeta]i[propersubset]W*, es la secuencia de frases que han llevado de [zeta]1 a [zeta]n, diremos que D es una derivación de [zeta]n desde [zeta]1. Se define entonces el lenguaje generado por G como el conjunto de todas las cadenas del alfabeto que se pueden derivar del axioma de G: L(G)={ a : a[propersubset]V*, Sa }.

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.

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:

Tipo 0: Gramáticas sin restricciones en las reglas.

Tipo 1: Gramáticas sensibles al contexto, con reglas de la forma [zeta]1A[zeta]2->[zeta]1[beta][zeta]2. Es decir, el no terminal A se sustituye por la frase [beta] del vocabulario en el contexto [zeta]1[zeta]2 ([zeta]1,[zeta]2[propersubset]W*, [beta][propersubset]W+).

Tipo 2: Gramáticas de independientes del contexto, en las que A->[beta] independientemente del contexto.

Tipo 3: Gramáticas regulares, cuyas reglas son de la forma A->aB o A->a; a[propersubset]V; A,B[propersubset]N.

La gramáticas de tipo 0 y 1 son las que proporcionan el mayor poder descriptivo, aunque son las gramáticas del tipo 2 y 3 las más utilizadas en aplicaciones prácticas como el reconocimiento de formas, principalmente debido a su mucho menor complejidad. Todas ellas son las llamadas gramáticas formales.

Arboles de Derivacion

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.

La representación de grupos es una parte de las matemáticas que examina cómo actúan los grupos sobre estructuras dadas.

Aquí la atención se centra en particular en las operaciones de grupos en espacios vectoriales . No obstante, también se consideran los grupos que actúan sobre otros grupos o sobre sets . Para obtener más detalles, consulte la sección sobre representaciones de permutación .

Con la excepción de algunas excepciones marcadas. También nos restringimos a espacios vectoriales sobre campos de característica cero. Debido a que la teoría de campos algebraicamente cerrados de característica cero es completa, una teoría válida para un campo algebraicamente cerrado especial de característica cero

también es válida para cualquier otro campo algebraicamente cerrado de característica cero.

La propiedad de un conjunto finito está bajo ciertas condiciones:

- La unión de dos (o una cantidad finita cualquiera) de conjuntos finitos es finita.
- La intersección de un conjunto finito con uno o más conjuntos arbitrarios es finita.
- Todo subconjunto de un conjunto finito es finito a su vez.
- El conjunto potencia de un conjunto finito con *N* elementos es finito, y posee 2^n elementos.

Las representaciones finitas se describen con facilidad mediante expresiones simples llamadas expresiones regulares.

Existen distintas representaciones como:

- ❏ Representaciones lineales
- ❏ Representación de permutación
- ❏ Representación trivial
- ❏ Representación regular izquierda y derecha
- ❏ Representación derecha-regular
- ❏ Representaciones, módulos y álgebra de convolución

Hemos visto que muchos lenguajes no son regulares. Por lo que necesitamos una clase más grande de lenguajes.

P es una variable o s´ımbolo no terminal o categoría sintáctica.
P es en esta gramática también el símbolo inicial.
Se tiene dos Derivaciones:

- ❏ Derivación e Inferencia:
  Son equivalentes, osea que si podemos inferir que una cadena de símbolos terminales w está en el lenguaje de una variable A.
- ❏ Derivaciones más a la izquierda y más a la derecha:
  Para restringir el número de opciones para derivar una cadena.
  • Derivación más a la izquierda: Siempre reemplaza la variable más a la izquierda por uno de los cuerpos de sus producciones.
  • Derivación más a la derecha: Siempre reemplaza la variable más a la derecha por uno de los cuerpos de sus producciones.

Derivation Trees: Árbol de derivación
En un árbol de derivación, la raíz es la variable de inicio, todos los nodos internos están etiquetados con variables, mientras que todas las hojas están etiquetadas con

terminales. Los hijos de un nodo interno están etiquetados de izquierda a derecha con el lado derecho de la producción utilizada
Ejemplo:

Recall the CFG for equal 0's and 1's:

$$S \to 0S1S \mid 1S0S \mid \varepsilon$$

The derivation for 011100 given earlier was left-most:

$$S \Longrightarrow 0S1S \Longrightarrow 01S \Longrightarrow 011S0S \Longrightarrow 0111S0S0S$$
$$\Longrightarrow 01110S0S \Longrightarrow 011100S \Longrightarrow 011100$$

Este tiene 3 nodos:

- Nodos raíz: Es una variable no terminal en la producción de la gramática que existe en el lado izquierdo de las reglas de producción o la raíz del árbol está representada por el símbolo de inicio de las reglas de producción de gramática libre de contexto (CFG).

- Nodos intermedios: todas las variables aceptan el nodo raíz considerado como nodos intermedios. Excepto que el símbolo de inicio de no terminales en las reglas de producción se convierte en parte del nodo intermedio.
- Nodos de hojas: Nodos que no tienen hijos considerados hojas. Cada nodo hoja está representado por una terminal.

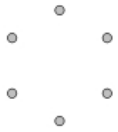
Grafo regular
En Teoría de grafos, un Grafo regular es un grafo donde cada vértice tiene el mismo grado o valencia. Un Grafo regular con vértices de grado k es llamado Grafo k-regular o Grafo regular de grado k.

Los Grafos regulares de grado hasta 2 son fáciles de clasificar: Un grafo 0-regular consiste en un grafo con vértices desconectados, un grafo 1-regular consiste en un grafo con aristas desconectadas, y un grafo 2-regular consiste en un ciclo.

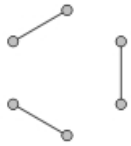
Un grafo 3-regular es conocido como un grafo cubo.

Un grafo completo Kn es (n-1)-regular

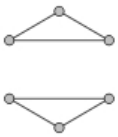
Grafo 0-regular



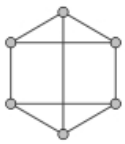
Grafo 1-regular



Grafo 2-regular



Grafo 3-regular



Autómatas Finitos

Este tipo de autómatas admite su definición de dos maneras bien diferentes:: Como autómatas traductores o reconocedores. La definición como autómatas traductores continua a la definición de las máquinas secuenciales, y se los podría definir como una subclase de estas, ya que los autómatas finitos tendrían como limitante no poder iniciar desde cualquier estado como lo hacen en las máquinas secuenciales.

La forma que adoptaremos para la definición de los autómatas finitos deterministas es como autómatas reconocedores, ya que se ajusta con los contenidos de la informática teórica y utilización que se les da dentro del diseño de los analizadores léxicos.

Estos autómatas solo se limitarán a aceptar o no una determinada cadena recibida en la entrada, por lo tanto podemos decir que la salida de los mismos solo tendrá dos valores posibles aceptar o no aceptar a la palabra de entrada.

Al igual que en las máquinas secuenciales, estos autómatas transitarán entre un conjunto finito de estados posibles, a medida que reciban sucesivamente los caracteres de entrada, en un instante determinado de tiempo el autómata solo podrá estar en uno y solo uno de los estados posibles.

Una característica importante de este tipo de autómatas es el determinismo, lo cuál significa que estando en un estado y recibiendo una entrada del exterior el autómata tendrá la posibilidad de transitar a uno y solo un estado del conjunto de estados posibles.

Con respecto al conjunto de estados ahora se pueden clasificar en tres tipos: Estado Inicial, que es donde comenzará la ejecución de la máquina; Estados finales o de aceptación que

será un subconjunto del conjunto de estados por los que transitará la máquina, y si cuando se hayan terminado de procesar todos los símbolos de entrada y no reste ningún símbolo por leer, la máquina quede posicionada en uno de estos estados de aceptación, se concluirá que la cadena procesada será aceptada por el autómata. y Estados Intermedios, que tienen comportamiento idéntico a los definidos en las máquinas secuenciales.

Definición:

Los autómatas finitos deterministas quedarán formalmente definida mediante una quíntupla como sigue:

$AFD = (\sum , Q, q_0, F, f)$

donde:

$\sum$	Alfabeto de símbolos de entrada.
Q	Conjunto finito de estados
q_0	$q_0 \varepsilon Q$ – estado inicial previsto
F	$F \subseteq Q$ - es el conjunto de estado finales de aceptación.
f	Función de transición de estados definida como $f: Q \times \sum \longrightarrow Q$

Interpretación de funcionamiento:

Este autómata recibirá una cadena en la cinta de entrada e ira procesando de uno a la vez los símbolos de entrada. Comenzará su funcionamiento posicionada en el estado inicial, y desde este estado comenzará su ejecución.

En cada intervalo de tiempo discreto realizará dos acciones las cuales serán consideradas como acciones indivisibles en un determinado intervalo de tiempo.

Las acciones que realiza son:

- Leer el próximo símbolo, desde la cinta de entrada.
- Transitar, cambiar de estado

Extensión a palabras.

El autómata finito determinista realizará transiciones de estados a través de la función f sólo cuando reciba un símbolo de entrada. Esto puede generalizarse a una palabra completa, o cuando reciba la palabra vacía, en este caso se denominará una función de transición f´ como la función.

$f' : Q \times \sum^* \longrightarrow Q$

Donde:

$f' (q, ax) = f'(f(q,a),x)$

$f' (q, \lambda) = q$ con $a \varepsilon \sum ; x \varepsilon \sum^* ; q \varepsilon Q$

Aceptación de Palabras.

Una palabra será aceptada por un AFD si estando formada por símbolos pertenecientes al alfabeto de entrada, al finalizar de procesar la misma, el autómata queda posicionado en una de los estados perteneciente al conjunto de estados finales de aceptación.

Dada:

$x \varepsilon W(\Sigma)$

$$f(q_0, x) = q_n \quad \text{Si } q_n \varepsilon F$$

Lenguaje reconocido por un AFD.

Es el conjunto de todas las palabras reconocidas por el Autómata Finito Determinista.

$$L_{AFD} = \{ x / x \varepsilon \Sigma^* \text{ y } f(q_0, x) = q_n \text{ con } q_n \varepsilon F \}$$

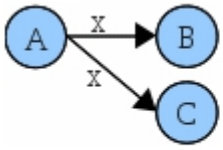
Autómata finito no determinista.

Autómata finito no determinista. Es el autómata finito que tiene transiciones vacías o que por cada símbolo desde un estado de origen se llega a más de un estado destino.

Los AFND son definiciones no tan deseables dentro de los lenguajes regulares porque dificultan su implementación tanto mecánica como informática; aunque en la mayoría de las transformaciones a lo interno de los LR (expresiones regulares a AF, gramáticas regulares a AF) conducen a AFND.

Definición.

Sea un autómata finito definido por la 5-tupla A=<Q, T, g, F, q0>, donde Q es el conjunto de estados, T el alfabeto de símbolos terminales, la relación de transiciones Definición relación transiciones af.gif ó Definición relación transiciones afd ext.gif (léase: del estado qi mediante el terminal x se va a qj), F son los estados finales o de llegada dentro de Q, q0 es el estado inicial o de partida; se dice que A es un autómata finito no determinista (AFND) si y sólo si existen en g al menos una de las siguientes transiciones:



ó $<q_i, x, q_j>$ y $<q_i, x, q_k>$ son transiciones de g y $q_j \neq q_k$.

$<q_i, \xi, q_j>$.

La definición formal de AFND se basa en la consideración de que a menudo según los algoritmos de transformación de expresiones y gramáticas regulares a AF terminan obteniéndose autómatas con transiciones múltiples para un mismo símbolo o transiciones vacías. Independientemente que sean indeseables, sobre todo para la implementación

material, fundamentalmente mecánica, de los autómatas finitos, son imprescindibles durante la modelación de analizadores lexicográficos de los elementos gramaticales de los lenguajes de programación, llamados tókens, como literales numéricos, identificadores, cadenas de texto, operadores, etc.