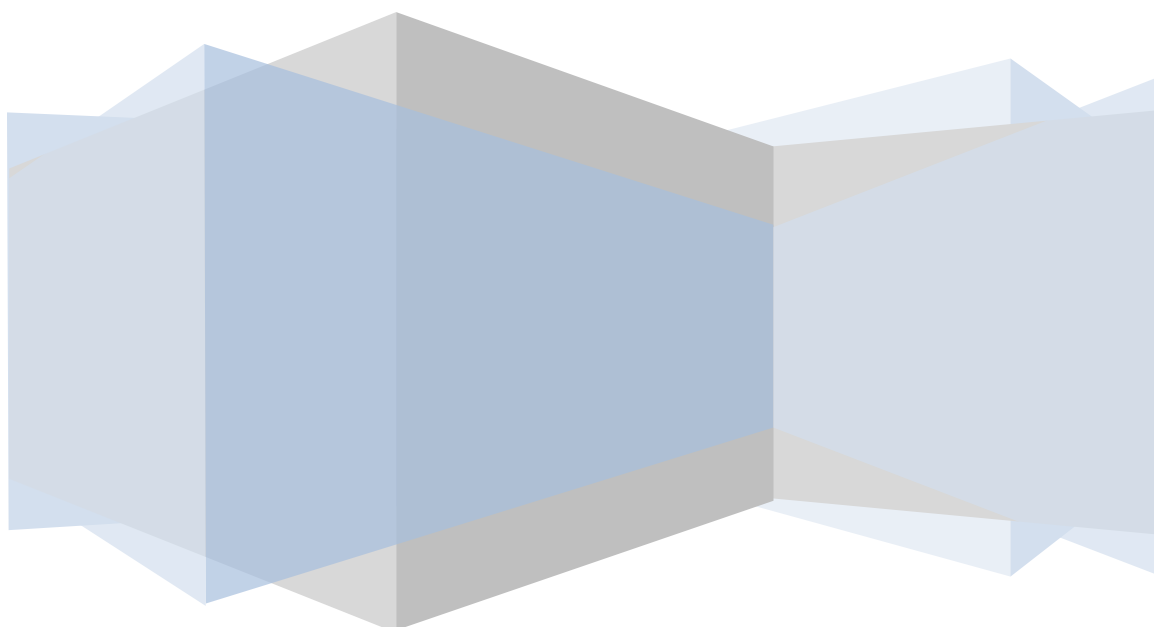




INSTITUTO TECNOLÓGICO DE TOLUCA

TEORÍA DE LA COMPUTACIÓN

BANY SABEL HERNÁNDEZ CARDONA



Agosto 2009

CONTENIDO	Pagina
1. INTRODUCCIÓN.....	1
1.1 AUTÓMATAS, COMPUTABILIDAD Y COMPLEJIDAD.....	3
1.2 NOCIONES MATEMÁTICAS.....	6
1.2.1 CONJUNTOS.....	6
1.2.2 FUNCIONES Y RELACIONES.....	7
1.2.3 CADENAS Y LENGUAJES.....	11
1.3 INDUCCIÓN MATEMÁTICA.....	13
EJERCICIOS PROPUESTOS.....	15
CONSULTAS REALIZADAS.....	17
2. LENGUAJES REGULARES.....	19
2.1 AUTÓMATAS FINITOS.....	22
2.1.1 AUTÓMATA FINITO DETERMINISTA (AFD).....	23
2.1.2 AUTÓMATAS FINITOS NO DETERMINISTA (AFND).....	25
2.2 EXPRESIONES REGULARES.....	26
2.3 LENGUAJES REGULARES	48
EJERCICIOS PROPUESTOS.....	51
CONSULTAS REALIZADAS.....	55
	57
3 LENGUAJES LIBRES DE CONTEXTO.....	
3.1 GRAMATICAS LIBRES DE CONTEXTO.....	60
3.2 ÁRBOLES DE DERIVACIÓN.....	61
3.3 FORMAS NORMALES DE CHOMNSKY.....	71
3.4 FORMAS NORMALES DE GREIBACH.....	73
3.5 ELIMINACIÓN DE FACTORES COMUNES IZQUIERDOS.....	75
3.6 ELIMINACIÓN DE RECURSIVIDAD IZQUIERDA.....	75
3.7 ELIMINACION DE AMBIGÜEDAD.....	76
3.8 AUTÓMATAS DE PUSH-DOWN.....	77
3.9 LENGUAJES NO REGULARES.....	89
EJERCICIOS PROPUESTOS.....	92
CONSULTAS REALIZADAS.....	94

CONTENIDO	Pagina
4.- MAQUINAS DE TURING.....	96
4.1 DEFINICION FORMAL DE UNA MÁQUINA DE TURING.....	98
4.2 CONSTRUCCIÓN MODULAR DE UNA MAQUINA DE TURING.....	109
4.3 LENGUAJES ACEPTADOS POR UNA MAQUINA DE TURING.....	113
4.4 VARIANTES DE UNA MAQUINA DE TURING.....	118
4.5 PROBLEMAS DE HILBERT.....	123
EJERCICIOS PROPUESTOS.....	128
CONSULTAS REALIZADAS.....	131
	133
5.- DECIBILIDAD.....	
5.1 LENGUAJES DECIDIBLES.....	134
5.2 EL PROBLEMA DE HALTING.....	137
5.3 DECIBILIDAD DE TEORÍAS LÓGICAS.....	139
EJERCICIOS PROPUESTOS.....	141
CONSULTAS REALIZADAS.....	142
	143
6.- REDUCIBILIDAD.....	
6.1 PROBLEMAS INSOLUBLES PARA LA TEORÍA DE LENGUAJES.....	145
6.2 UN PROBLEMA SIMPLE INSOLUBLE.....	147
6.3 FUNCIONES COMPUTABLES.....	150
6.4 REDUCIBILIDAD DE TURING.....	155
EJERCICIOS PROPUESTOS.....	156
CONSULTAS REALIZADAS.....	157

1. INTRODUCCIÓN

Competencias que deberán ser adquiridas

- Capacidad de investigación
- Autonomía en el aprendizaje
- Creatividad
- Capacidad de Análisis y síntesis
- Organización y Planificación
- Comunicación oral y escrita
- Utilización de las tecnologías de comunicación y de información (TIC'S)
- Trabajo en equipo

Objetivo de aprendizaje

Capacidad para aplicar conceptos y diseñar lenguajes formales y naturales.

Resultado del aprendizaje	Actividad Educativa	Evaluación	Tiempo
Comprensión y capacidad de utilizar en un contexto adecuado conceptos de Autómatas, complejidad y compatibilidad	• Búsqueda de conceptos por equipos en los que podrá leer, descubrir, cuestionar, preguntar indagar. • Discusión en clase para expresarse, comunicar ideas, etc. • Creación de ideas originales y obtenidas en consenso representadas en mapas mentales	• Informes por equipo y conclusiones grupales. • Técnicas de observación(registros, listas de control)	1 horas
Capacidad para aplicar la teoría de conjuntos , relaciones y funciones en el diseño de lenguajes	• Búsqueda de conceptos por equipos en los que podrá leer, descubrir, cuestionar, preguntar indagar sobre la teoría de conjuntos, relaciones y funciones. • Discusión en clase para expresarse, comunicar ideas, etc. • Búsqueda de conceptos en los que podrá leer, descubrir, cuestionar, preguntar indagar	• Informes por equipo y conclusiones grupales. • Generación de un lenguaje formal o natural usando conceptos y haciéndolos resaltar.	2 horas 2 horas

	sobre lenguajes • Discusión en clase para expresarse, comunicar ideas, etc. • Ejemplos de lenguajes • Aplicación de conceptos para diseñar lenguajes formales y naturales		1 hora
Capacidad para realizar demostraciones por medio de inducción Matemática	• Búsqueda de conceptos por equipos en los que podrá leer, descubrir, cuestionar, preguntar indagar. • Clase teórica sobre Inducción Matemática • Realizar ejercicios sobre inducción matemática	• Realizar ejercicios sobre inducción matemática • Prueba de respuesta con desarrollo.	2 horas

1. INTRODUCCIÓN

1.1 AUTÓMATAS, COMPUTABILIDAD Y COMPLEJIDAD

La teoría de la compatibilidad, también denominada teoría de la recursión, es una de las cuatro partes que constituyen la lógica matemática, siendo las otras tres, la teoría de conjuntos, la teoría de modelos y la teoría de la demostración, y se ocupa del estudio y clasificación de las relaciones y aplicaciones computables. Además, la teoría de la computabilidad, complejidad computacional, junto con la teoría de autómatas, lenguajes y máquinas, es el fundamento de la informática teórica y esta, a su vez, de la industria de los ordenadores.

La teoría de autómatas es una rama de las ciencias de la computación que estudia matemáticamente máquinas abstractas y problemas que éstas son capaces de resolver. La teoría de autómatas está estrechamente relacionada con la teoría del lenguaje formal ya que los autómatas son clasificados a menudo por la clase de lenguajes formales que son capaces de reconocer.

Un autómata es un modelo matemático para una máquina de estado finita, es una máquina que, dada una entrada de símbolos, "salta" a través de una serie de estados de acuerdo a una función de transición (que puede ser expresada como una tabla), esta función de transición dice al autómata a que estado cambiar dados un determinado estado y símbolo.

La entrada es leída símbolo por símbolo, hasta que es "consumida" completamente (piense en esta como una cinta con una palabra escrita en ella, que es leída por una cabeza lectora del autómata; la cabeza se mueve a lo largo de la cinta, leyendo un símbolo a la vez) una vez la entrada se ha agotado, el autómata se detiene.

Dependiendo del estado en el que el autómata para se dice que este ha aceptado o rechazado la entrada. Si este termina en el estado "acepta", el autómata acepta la palabra. Si lo hace en el estado "rechaza", el autómata rechazó la palabra, el conjunto de todas las palabras aceptadas por el autómata constituyen el lenguaje aceptado por el mismo.

Autómata del griego automatos, que significa espontáneo o con movimiento propio, puede referirse a:

- Autómata programable: Equipo electrónico programable en lenguaje no informático y diseñado para controlar, en tiempo real y en ambiente industrial, procesos secuenciales.
- Teoría de autómatas: Estudio matemático de máquinas abstractas (autómata finito, autómata con pila).
- Autómata (mecánico): Máquina que imita la figura y los movimientos de un ser animado.
- Robot: Máquina o ingenio electrónico programable, capaz de manipular objetos y realizar operaciones antes reservadas solo a las personas.

La Teoría de la computabilidad es la parte de la computación que estudia los problemas de decisión que pueden ser resueltos con un algoritmo o equivalentemente con una máquina de Turing. La teoría de la computabilidad se interesa a cuatro preguntas:

- ¿Qué problemas puede resolver una máquina de Turing?
- ¿Qué otros formalismos equivalen a las máquinas de Turing?
- ¿Qué problemas requieren máquinas más poderosas?
- ¿Qué problemas requieren máquinas menos poderosas?

El origen de los modelos abstractos de computación se encuadra en los años 30 (antes de que existieran los ordenadores modernos), para el trabajo de los lógicos Alonzo Church, Kurt Gödel, Stephen Kleene, Emil Leon Post, y Alan Turing. Estos trabajos iniciales han tenido una profunda influencia, tanto en el desarrollo teórico como en abundantes aspectos de la práctica de la computación; previendo incluso la existencia de ordenadores de propósito general, la posibilidad de interpretar programas, la dualidad entre *software* y *hardware*, y la representación de lenguajes por estructuras formales basados en reglas de producción.

El punto inicial de estos primeros trabajos fueron las cuestiones fundamentales que David Hilbert formuló en 1900, durante el transcurso de un congreso internacional.

Lo que Hilbert pretendía era crear un sistema matemático formal completo y consistente en el cual, todas las aseveraciones fueran planteadas con precisión. Su intención era encontrar un algoritmo que determinara la verdad o falsedad de cualquier proposición en el sistema formal. Al problema en cuestión se le denominó *Entscheidungsproblem*. En caso de que Hilbert hubiese cumplido su

objetivo, cualquier problema bien definido se resolvería simplemente al ejecutar dicho algoritmo.

Pero fueron otros los que mediante una serie de investigaciones mostraron que esto no era posible. En contra de esta idea K. Gödel sacó a la luz su conocido Primer Teorema de Incompletitud. Este viene a expresar que todo sistema de primer orden consistente que contenga los teoremas de la aritmética y cuyo conjunto de axiomas sea recursivo no es completo. Gödel construyó una fórmula que es satisfactoria pero que no puede ser probada en el sistema. Como consecuencia, no es posible encontrar el sistema formal deseado por Hilbert en el marco de la lógica de primer orden, a no ser que se tome un conjunto no recursivo de axiomas.

Una posterior versión, que resulta más general, del teorema de incompletitud de Gödel, indica que ningún sistema deductivo que contenga los teoremas de la aritmética, y con los axiomas recursivamente enumerables puede ser consistente y completo a la vez. Esto hace pensar, a nivel intuitivo, que no va a ser posible definir un sistema formal.

La teoría de la complejidad computacional es la rama de la teoría de la computación que estudia, de manera teórica, los recursos requeridos durante el cómputo de un algoritmo para resolver un problema. Los recursos comúnmente estudiados son el tiempo (mediante una aproximación al número y tipo de pasos de ejecución de un algoritmo para resolver un problema) y el espacio (mediante una aproximación a la cantidad de memoria utilizada para resolver un problema). Se pueden estudiar igualmente otros parámetros, tales como el número de procesadores necesarios para resolver el problema en paralelo. La teoría de la complejidad difiere de la teoría de la computabilidad en que esta se ocupa de la factibilidad de expresar problemas como algoritmos efectivos sin tomar en cuenta los recursos necesarios para ello.

Los problemas que tienen una solución con orden de complejidad lineal son los problemas que se resuelven en un tiempo que se relaciona linealmente con su tamaño.

Hoy en día las computadoras resuelven problemas mediante algoritmos que tienen como máximo una complejidad o costo computacional polinómico, es decir, la relación entre el tamaño del problema y su tiempo de ejecución es polinómica. Éstos son problemas agrupados en la clase P. Los problemas que no pueden ser resueltos por nuestras computadoras (las cuales son Máquinas Determinísticas), que en general poseen costes factorial o combinatorio pero

que podrían ser procesados por una máquina no-determinista, están agrupados en la clase NP. Estos problemas no tienen una solución práctica, es decir, una máquina determinística (como una computadora actual) no puede resolverlos en un tiempo razonable.

1.2 NOCIONES MATEMÁTICAS

1.2.1 CONJUNTOS

Un *conjunto* es simplemente cualquier colección definida de objetos llamados elementos o miembros del conjunto. Por ejemplo, el conjunto de todos los alumnos de Lenguajes y Autómatas, el conjunto de todos los números enteros.

Un *conjunto finito*, es un conjunto de elementos que se encuentra limitado y cuando no es demasiado extenso, se puede describir nombrando cada uno de sus elementos. Por ejemplo:

$$A = \{1,2,3,4\}$$

Cuando el conjunto finito es más extenso es conveniente expresarlo mediante una lista de propiedades que caractericen los elementos del conjunto. Por ejemplo:

$$B = \{x \mid x \text{ son los libros de lenguajes y autómatas}\}$$

Un *conjunto infinito* es un conjunto de elementos que no se pueden contar, que no tienen un límite y se expresan mediante una lista de propiedades. Por ejemplo:

$$C = \{y \mid y \text{ es un elemento par y positivo}\}$$

Un conjunto infinito, pasa a ser finito cuando se empiezan a generar reglas o condiciones para delimitarlo. Por ejemplo, a partir de:

$$C = \{x \mid x \text{ es un número cualquiera}\}$$

Que es un conjunto infinito, ya que no existe ninguna regla que lo limite, podemos delimitar así:

$$C = \{x \mid x \text{ es un número mayor de 50 y menor que 100}\}$$

El conjunto anterior, tiene una regla asignada que es un rango, por lo cual solo se podría enumerar los números entre 51 y 99. En este momento, el conjunto se hace finito.

En la tabla 1. Se muestran conjuntos de números especiales.

Tabla 1. Conjuntos especiales

$\emptyset$	Conjunto Vacío	
U	Conjunto Universal	
$\mathbb{P}$	Conjunto Números Enteros Positivos	1,2,3,...
$\mathbb{N}$	Conjunto Números Naturales	0,1,2,3,...
$\mathbb{Z}$	Conjunto Números Enteros	..., -3, -2, -1, 0, 1, 2, 3, ...
$\mathbb{Q}$	Conjunto Números Racionales (Razones de enteros, esto es quebrados)	p/q donde $p \in \mathbb{Z}$ y $q \in \mathbb{P}$
$\mathbb{I}$	Conjunto Números Irracionales	$\sqrt{2}, \pi, e, 0, \frac{3}{4}, \dots$

1.2.2 FUNCIONES Y RELACIONES

Relaciones

Sean A y B conjuntos. Una relación de A a B es cualquier subconjunto R del producto cartesiano $A \times B$. A se conoce como dominio y B como rango de R. Formalmente:

$$aRb = \{ \langle a, b \rangle \in R \mid a \in A \wedge b \in B \wedge R \subseteq A \times B \}$$

Ejemplos

$$P = \{ x \in \mathbb{P} \mid x \text{ es primo} \wedge x < 12 \} = \{ 1, 3, 5, 7, 11 \}$$

$$I = \{ x \in \mathbb{P} \mid x \text{ es impar} \wedge x < 10 \} = \{ 1, 3, 5, 7, 9 \}$$

$$P \times I = \{ \langle 1, 1 \rangle, \langle 1, 3 \rangle, \langle 1, 5 \rangle, \dots, \langle 11, 9 \rangle, \}$$

Sea:

$$R \subseteq P \times I$$

$$R = \{ \langle 1, 1 \rangle, \langle 3, 3 \rangle, \langle 5, 5 \rangle, \langle 7, 7 \rangle, \langle 11, 9 \rangle \}$$

En la tabla 2. Se muestran las propiedades de las relaciones.

Tabla 2. Propiedades para Relaciones Sea $R \in S \times S$

Reflexiva	Para toda $x \in S$, existe xRx	$E \in R$
Irreflexiva	Para toda $x \in S$, no existe xRx	$E \cap R = \emptyset$
Simétrica	Para toda xRy , existe yRx	$E \in R$
Asimétrica	Para cada xRy , no existe yRx	$E \cap R = \emptyset$
Antisimétrica	Para cada xRy , no existe yRx , pero si xRx	$E \in R$
Transitiva	Siempre que exista xRy , y yRz , existe xRz	$E \in R$

Funciones

Se llama función a una relación en la cual a cada elemento del conjunto de partida le corresponde sólo un elemento del conjunto de llegada.

Sean X, Y conjuntos.

Una función f de X a Y es una relación R de X a Y tal que para cada $f(x)$ existe un solo elemento $y \in Y$. Posee las siguientes propiedades.

- 1.- El dominio de f es X .
- 2.- Si $(x, y), (x, y') \in f$ entonces $y = y'$

Una función f de X a Y se denota también como $f: X \rightarrow Y$, para denotar que $(x, y) \in f$, se suele escribir $f(x) = y$

Una función f de X en Y es uno a uno o inyectiva si para $y \in Y$ existe a lo más un $x \in X$ tal que $f(x) = y$.

Si f es una función de X en Y y el rango de f es igual a Y , se dice que f es sobre Y o que la función es suprayectiva.

Una función que es uno a uno y sobre se llama función biyectiva.

1.2.3 CADENAS Y LENGUAJES.

Un *alfabeto* (Σ) es un conjunto de símbolos, es decir, un conjunto de elementos que son parte de un conjunto o lenguaje. Por ejemplo:

$\Sigma(a, b, c)$

$\Sigma(1, 2, 3, 4)$

$\Sigma(\#, \&, \$, \%)$

$\Sigma(\text{palabras reservadas, identificadores, números, cadenas, caracteres especiales})$

Una *cadena* (*string*) es una secuencia finita de símbolos o elementos yuxtapuestos que se forman con elementos del alfabeto.

La cadena *mínima* que se puede formar con un alfabeto, es la cadena vacía (λ), otros autores le nombran cadena nula. Si tenemos un alfabeto o vocabulario $\{a, b\}$, algunas cadenas son:

$\lambda, a, ab, aaa, aba, abb, \text{etc.}$

Cuando las cadenas no se limitan en tamaño o condiciones, el conjunto de cadenas que se pueden formar es infinito, pero cuando las cadenas se limitan en cuanto a características como tamaño, repetición de una letra, etc, el conjunto tiende a ser finito. Por ejemplo si $\Sigma(a, b)$

Las cadenas de longitud 3 terminadas en b son:

aab, abb, bab, bbb

Propiedades de cadenas

Una *cadena* (*string*) es una secuencia finita de símbolos yuxtapuestos. Por ejemplo a, b y c son símbolos, y abcb es una cadena. Es importante el orden, así las cadenas ab y ba son distintas.

La *longitud* de una cadena w, es el número de símbolos que componen la cadena.

La longitud de una cadena x se indica con $|x|$ y es el número de letras que la forman. Por ejemplo: Si x está formada por los elementos a, b, c, d, algunas cadenas posibles de formar son:

a, abcd, aabcdd, etc.

La longitud de $|a| = 1$ puesto que hay un solo elemento, mientras que la de $|abcd| = 4$ debido a que se encuentran 4 elementos.

Nota:

Si la cadena producida es el vacío $|\lambda| = 0$ ya que no existe ningún elemento.

Los *prefijos* de una cadena están formados por los primeros símbolos de éste; y los *sufijos*, por los últimos.

La *concatenación* de dos cadenas, es la cadena que se forma al escribir la primera cadena seguida de la segunda, sin que haya espacio entre ellas.

Ejemplos:

Si se tienen las cadenas $x = abc$, y $z = dea$; la concatenación de x con z es:

x concatenación $z = abcdea$;

La concatenación de la cadena x con la cadena vacía es:

$$\lambda x = x\lambda = x = abc$$

La *potencia* de una cadena se forma concatenando una cadena consigo mismo, tantas veces como indica el exponente. Si $x = a$, entonces:

$$x^0 = \lambda \quad x^1 = a \quad x^2 = aa$$

Nota: La potencia de una cadena **no** es similar a la potencia algebraica.

$A^0 = \{\lambda\}$; $A^1 = A$; $A^2 = AA$; Y en general $A^n = AA^{n-1}$ para $n > 0$.

Por ejemplo si $A = \{a, b\}$, entonces se tiene:

$$A^0 = \{\lambda\}$$

$$A^1 = \{a, b\}$$

$$A^2 = \{aa, ab, ba, bb\}$$

$$A^3 = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$$

Donde A está compuesto de 2 elementos, a y b.

En ocasiones se puede considerar a una cadena como un elemento, aún así la potencia se aplica de igual forma. Ejemplo:

Sea $X = \{abc\}$

La potencia de $X^0 = \lambda$ $X^1 = abc$ $X^2 = abcabc$

Sea $Y = \{cde, fgh\}$

$Y^0 = \lambda$ $Y^1 = cde, fgh$ $Y^2 = cdefgh, fghcde, cdecde, fghfgh$

Lenguajes

Lenguaje. Conjunto cualquiera de frases. Conjunto de símbolos que se concatenan para formar palabras válidas, las palabras se combinan para formar oraciones o frases a través de un conjunto de reglas gramaticales, en forma sistemática para comunicar pensamientos e ideas. Esta comunicación puede ser oral y escrita, existen lenguajes formales y lenguajes naturales, los lenguajes formales son los que definen los lenguajes de programación que son herramientas que nos permiten crear programas y software. Entre ellos tenemos Delphi, Visual Basic, Pascal, Java, etc., Entre los lenguajes naturales se encuentran los lenguajes español, inglés, francés, etc.

Representación finita del lenguaje

Para poder diseñar un lenguaje se necesita:

- Objetivo
- Alfabeto. (definido en símbolos y en palabras)
- Reglas.

Objetivo: aquí se define la meta, para lo que se diseña el lenguaje así como para quien está dirigido.

El alfabeto definido en símbolos: los signos que se usan para formar las palabras del lenguaje por ejemplo, el código ASCII.

El alfabeto definido en palabras: es la clasificación de palabras en conjuntos que forman el lenguaje por ejemplo un lenguaje de programación está formado por un conjunto de palabras reservadas, identificadores, operadores aritméticos, operadores relacionales, etc. Las *reglas:* se encargan de verificar

que las palabras, sentencias y programa fuente, sean válidas para el lenguaje. En caso de que no sean, se marcará un error.

Ejemplo:

Definir un lenguaje que represente expresiones aritméticas.

Objetivo: identificar expresiones aritméticas para los alumnos de teoría de la computación.

Alfabeto definido en símbolos: (,), 0,1,2,3,4,5,6,7,8,9,+,*/, -

Alfabeto definido en palabras: paréntesis_abierto, paréntesis_cerrado, números, signos.

Reglas para definir palabras: solo usar número enteros positivos, no exceder de una longitud de 3 números positivos.

Reglas para definir expresiones o sentencias: usar paréntesis balanceados, entre los números usar un operador aritmético, por ejemplo
(44*34+2+(4/3))

Genere un lenguaje que acepte todas las cadenas que sean válidas para aceptar identificadores en el lenguaje Pascal.

Objetivo: definir identificadores validos en lenguaje Pascal para los alumnos de teoría de la computación.

Alfabeto definido en símbolos: 0,1,2,3,4,5,6,7,8,9, a...z, A..Z, _.

Alfabeto definido en palabras: conjunto de letras (l), conjunto de números(n), guion bajo (_).

Reglas para definir identificadores:

- 1) Todos los identificadores empiezan con letra.
- 2) Identificadores terminan con letra y número
- 3) Puede existir un guión bajo (_) en medio de la cadena, pero no al principio ni al final
- 4) La longitud de la cadena no debe ser mayor de 10 caracteres

Ejemplos de identificadores válidos:

Contador_1, Contador, Contador2, etc.

Ejemplos de identificadores no válidos:

contador, 211mega, qqqqqqqqqqqq, contador, etc.

1.3 INDUCCIÓN MATEMÁTICA

El método deductivo, muy usado en matemáticas, obedece a que a partir de unos ciertos conjuntos de axiomas aceptados sin demostración y de reglas lógicas no contradictorias, se deducen otros enunciados llamados teoremas combinando los axiomas y respetando en cada etapa las reglas lógicas.

Otro método para demostrar resultados generales que dependen en algún sentido de los números naturales es conocido con el nombre de Inducción Matemática.

Una función proposicional $P(n)$ es verdadera para todos los valores de la variable n si se cumplen las siguientes condiciones:

Paso 1.- La función proposicional $P(n)$ es verdadera para $n = 1$, o bien, $P(1)$ es verdadera.

Paso 2.- Hipótesis de Inducción. Se supone que $P(h)$ es verdadera, donde h es un número natural cualesquiera.

Paso 3.- Tesis de Inducción. Se demuestra que $P(h + 1)$ es verdadera, o bien, $P(h)$ verdadera y $P(h + 1)$ verdadera:

La Inducción Matemática es actualmente una herramienta de uso práctico y teórico principalmente para matemáticos y personas que trabajan en Ciencias Computacionales.

Es conveniente usar inducción matemática para probar proposiciones sobre colecciones de números naturales que no contienen el 0. Por ejemplo, la proposición de la fórmula $1+2+\dots+(2n-1) = n^2$ no tiene sentido (no es verdadera para $n=0$). Sin embargo la fórmula se cumple para toda $n>0$.

Por ejemplo

$$1+2+\dots+(2n-1) = n^2$$

(1) Para $n=1$

$$n^2 = 1^2 = 1$$

(2) Para $n=n+1$

$$\begin{aligned} [1+2+\dots+(2n-1)] + (2(n+1)-1) &= (n+1)^2 \\ n^2 + (2n+2-1) &= n^2+2n+1 \\ n^2 + 2n+1 &= n^2+2n+1 \\ n^2 + 2n+1 &= n^2+2n+1 \end{aligned}$$

Por ejemplo $5+8+11+\dots+(3n+2)=\frac{1}{2}(3n^2+7n)$

(1) Para $n=1$

$$\frac{1}{2} [3(1)^2 + 7(1)] = \frac{1}{2}(3+7) = \frac{1}{2}(10)=5$$

(2) Para $n=n+1$

$$\begin{aligned} [5+8+11+\dots+(3n+2)]+(3(n+1)+2) &= \frac{1}{2}(3(n+1)^2+7(n+1)) \\ \frac{1}{2}(3n^2+7n) + (3(n+1)+2) &= \frac{1}{2}(3(n+1)^2+7(n+1)) \\ (\frac{3}{2}n^2+\frac{7}{2}n) + ((3n+3)+2) &= \frac{1}{2}(3(n^2+2n+1)+7n+7) \\ (\frac{3}{2}n^2+\frac{7}{2}n+3n+5) &= \frac{1}{2}(3n^2+6n+3+7n+7) \\ \frac{1}{2}(3n^2+13n+10) &= \frac{1}{2}(3n^2+13n+10) \end{aligned}$$

EJERCICIOS PROPUESTOS

1. Del alfabeto Σ (a, b) genere cadenas que comiencen en ab pero que no terminen en ab.

2. Del alfabeto $\Sigma(0, 1)$ genere cadenas que contengan 01

3. Generar Cadenas que contenga un número par de y's con el alfabeto {x, y}

4. Dadas las cadenas obtenga objetivo del lenguaje

a
aaaaaa
abababbbbb
aaaaabbb
aaab
bbbbbbb
ababaaaaaaaa

5. Diga si el siguiente conjunto es finito o infinito, justifique su respuesta.

$x = \{ x \mid x \text{ es un múltiplo de tres} \}$

6. Diga si el siguiente conjunto es finito o infinito, justifique su respuesta.

$A = \{1,2,3,4\}$

7. Del conjunto anterior obtenga A^4 .

8. Diseñe un lenguaje para receta de licuados de sabores.

9. Compare el lenguaje español con un lenguaje de programación.

10.Cuál es la longitud de la siguiente cadena:

aghs3cury34rv45v5yt

11. Genere cadenas que comiencen en 1 y terminen en cero.

12. Genere cadenas que acepten cualquier número (entero/ real)

13 Probar cada una de las proposiciones siguientes, usando inducción matemática

a) $n < 2^n$ para todo $n > 0$

b) $n^3 + 2n$ es divisible por 3.

c) $n^2 + 2$ es divisible por 2

d) $n(n + 1)(n + 2)$ es divisible por 3

e) $n^3 + 5n$ es divisible por 3.

f) $a^{2n} - b^{2n}$ es divisible por $a + b$ con $a, b \in \mathbb{Z}$ y $a + b \neq 0$

g) $x^n - y^n$ es divisible por $x + y$

h) $1 + 3 + 5 + 7 + \dots + (2n - 1) = n^2$

i) $2 + 7 + 12 + \dots + (5n - 3) = \frac{n(5n - 1)}{2}$

j) $\frac{1}{3} + \frac{1}{15} + \frac{1}{35} + \dots + \frac{1}{4n^2 - 1} = \frac{n}{2n + 1}$

k) $3^n > 2n + 1$

l) $2n < 2^n$

CONSULTAS REALIZADAS

Alfred V. Aho, Monica S. Lam, Sethi Ravi y Jeffrey D. Ullman
Compiladores principios, técnicas y herramientas.
Editorial Addison-Wesley Iberoamericana.
2008.

Beck Leland
Software de sistemas. Introducción a la programación de sistemas.
Editorial Addison-Wesley Iberoamericana.
1988.

Jonh E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman
Introducción a la teoría de autómatas, Lenguajes y Computación.
Editorial PEARSON Addison Wesley.
2002.

Dean Kelley
Teoría de autómatas y lenguajes formales.
Editorial Prentice Hall
2006

Johnsonbaugh, Richard
Matemáticas discretas
Editorial Iberoamercana
2002.

Jean Paúl Trembay, Ram Manohar
Matemáticas discretas con aplicación a las ciencias de la computación
Editorial CECSA.
1996.

http://es.wikipedia.org/wiki/Teor%C3%ADa_de_aut%C3%B3matas

<http://es.wikipedia.org/wiki/Aut%C3%B3mata>

http://es.wikipedia.org/wiki/Teor%C3%ADa_de_la_computabilidad

http://es.wikipedia.org/wiki/Complejidad_computacional

<http://www.edufuturo.com/educacion.php?c=4131>

<http://lideres.ong.googlepages.com/4tacontinducmat.ppt>

<http://webdia.cem.itesm.mx/ac/aesparza/compi/expreg.html>

<http://campus.gda.itesm.mx/~csainz/gram.htm>

http://docencia.udea.edu.co/SistemasDiscretos/contenido/capitulo_08.html

<http://mx.geocities.com/cygnusgraphix/Html/ProgJava.htm>

2. LENGUAJES REGULARES

Competencias que deberán ser adquiridas

- Capacidad de investigación
- Autonomía en el aprendizaje
- Creatividad
- Capacidad de Análisis y síntesis
- Organización y Planificación
- Comunicación oral y escrita
- Utilización de las tecnologías de comunicación y de información (TIC'S)
- Trabajo en equipo

Objetivo de aprendizaje

Capacidad para diseñar lenguajes regulares y no regulares representándolos por medio de autómatas y expresiones regulares.

Resultado del aprendizaje	Actividad Educativa	Evaluación	Tiempo
Comprensión y capacidad de representar un lenguaje por medio de un autómata finito	• Búsqueda de conceptos por equipos en los que podrá leer, descubrir, cuestionar, preguntar indagar. • Discusión en clase para expresarse, comunicar ideas, etc. • Creación de ideas originales y del maestro obtenidas en consenso • Ejemplos de lenguajes representados por autómatas finitos.	• Informes por equipo y conclusiones grupales. • Técnicas de observación(registros, listas de control)	2 horas
Comprensión y capacidad de representar un lenguaje por medio de un autómata finito determinista	• Búsqueda de conceptos por equipos en los que podrá leer, descubrir, cuestionar, preguntar indagar. • Discusión en clase para expresarse, comunicar ideas, etc. • Creación de ideas originales y del maestro obtenidas en consenso • Ejemplos de lenguajes representados por autómatas	• Informes por equipo y conclusiones grupales. • Técnicas de observación(registros, listas de control) • Diseño de un lenguaje de programación o natural	2 horas

		representado por un AFD, usando conceptos y haciéndolos resaltar.	
Comprensión y capacidad de representar un lenguaje por medio de un autómata finito no determinista	• Búsqueda de conceptos por equipos en los que podrá leer, descubrir, cuestionar, preguntar indagar. • Discusión en clase para expresarse, comunicar ideas, etc. • Creación de ideas originales y del maestro obtenidas en consenso • Ejemplos de lenguajes representados por autómatas no deterministas • Propuesta por equipo del lenguaje definido en la unidad 1, por un AFD o AFND	• Informes por equipo y conclusiones grupales. • Técnicas de observación(registros, listas de control) • Diseño de un lenguaje de programación o natural representado por un AFD, usando conceptos y haciéndolos resaltar.	2 horas
Capacidad de representar un lenguaje por medio de una expresión regular	• Búsqueda de conceptos por equipos en los que podrá leer, descubrir, cuestionar, preguntar indagar. • Discusión en clase para expresarse, comunicar ideas, etc. • Creación de ideas originales y del maestro obtenidas en consenso • Ejemplos de lenguajes representados por expresiones regulares • Propuesta por equipo de un lenguaje representado por una Expresión regular.	• Informes por equipo y conclusiones grupales. • Técnicas de observación(registros, listas de control) • Diseño de un lenguaje de programación o natural representado por un AFND, usando conceptos y haciéndolos resaltar.	2 horas 2 horas
Equivalencia de un AFND a AFD	• Búsqueda de diferentes metodologías para pasar de un AFND a AFD por equipos en los que podrá leer, descubrir, cuestionar, preguntar indagar. • Discusión en clase para expresarse, comunicar ideas,	• Informes por equipo y conclusiones grupales. • Técnicas de observación(registros, listas de	2 horas

	etc. Ejemplos de AFND y su equivalencia a AFD	control) Programación de un lenguaje de programación o natural usando el diseño de un AFND, AFD o una expresión regular.	2 horas
Capacidad para crear nuevos lenguajes a partir de otros por medio de concatenación, unión complemento, etc.	- Búsqueda de conceptos por equipos en los que podrá leer, descubrir, cuestionar, preguntar indagar. - Discusión en clase para expresarse, comunicar ideas, etc. - Creación de ideas originales y del maestro obtenidas en consenso - Ejemplos de lenguajes regulares	- Informes por equipo y conclusiones grupales. - Técnicas de observación(registros, listas de control) - Crear nuevos lenguajes a partir de otros	2 horas

2. LENGUAJES REGULARES

Un lenguaje Regular es aquel que puede ser representado por un autómata finito no determinista o un autómata finito determinista.

2.1 AUTÓMATAS FINITOS

Un Autómata finito es un modelo matemático de un sistema, con entradas y salidas discretas. El sistema puede estar en cualquiera de un número finito de configuraciones o estados. El estado del sistema resume la información concerniente a entradas anteriores y que es necesaria para determinar el comportamiento del sistema para entradas posteriores.

Entonces, un autómata finito consiste en un conjunto finito de estados y un conjunto de transiciones de estado a estado, que se dan sobre símbolos de entrada tomados de un alfabeto.

Para cada símbolo de entrada existe exactamente una transición a partir de cada estado (posiblemente de regreso al mismo estado). Un estado, por lo general se denota por q_0 , que es el estado inicial, en el que el autómata inicia.

Algunos estados deben estar designados como final o de aceptación.

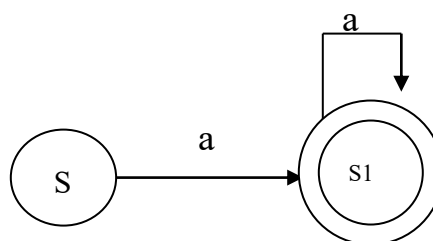
Para expresar en forma gráfica el comportamiento de un autómata se utiliza la siguiente simbología.

Símbolo	Nombre
$\rightarrow$	Flujo
$\bigcirc$	Estado
$\odot$	Estado de aceptación

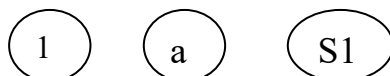
Ejemplo:

Cadenas que aceptan un número de a's uno a mas veces.

El autómata es:

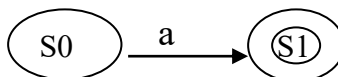
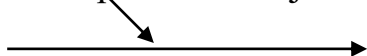


Por lo general dentro de los círculos se coloca un símbolo que identifique el estado del autómata, (no importando que sea de aceptación o no) puede ser un número, un símbolo, una letra pero lo más frecuente es la combinación de una letra con uno o más números por ejemplo S_0 , S_1 , S_{13} , etc.

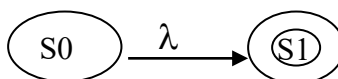


Los flujos que se representan por la flecha ($\rightarrow$) indican la transición de un estado a otro y por lo general en su parte superior llevan el carácter con el cual pasan de un estado a otro del autómata.

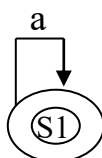
Parte superior del flujo



Dentro de los caracteres que pueden ir en la parte superior de un flujo se incluye el λ , denominándolo transición vacía.



Un flujo es un *lazo* cuando parte de un estado y llega a ese mismo. Es poco común que un lazo lleve un vacío, por lo general, se omite porque se considera que va implícito.



Dentro de los autómatas finitos existen 2 divisiones que son:

AFD: El autómata finito determinista

AFND: El autómata finito NO determinista

2.1.1 AUTÓMATA FINITO DETERMINISTA (AFD)

Un autómata finito determinista M es una 5-tupla o quinteta que se define como sigue:

$$M = \{Q, \Sigma, s, F, \delta\}$$

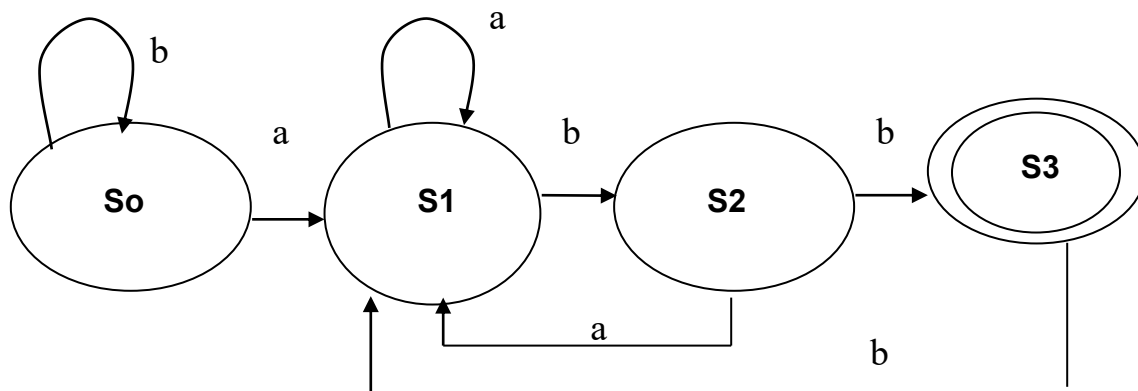
Donde:

- Q = Es una colección finita de estados.
- Σ = Es un conjunto finito de símbolos de entrada o el alfabeto de entrada al autómata.
- s = Es el estado inicial.
- F = Es una colección de estados finales o de aceptación.
- δ = Es una función $\delta: Q \times \Sigma \rightarrow Q$ que determina el único estado siguiente para el par (q_i, σ) correspondiente al estado actual y a la entrada.

Para distinguir un autómata finito determinista, existen dos características que puede hacer más fácil su identificación.

- Es aquel que en cualquier estado del autómata con diferente símbolo del alfabeto en la entrada se llega a estados diferentes del autómata por ejemplo: $\delta: (q_1, a) \rightarrow q_1$ y $\delta: (q_1, b) \rightarrow q_2$.
- También es aquel autómata que no contiene transiciones con el vacío, sin embargo puede aceptar cadenas vacías.

Autómata finito Determinista.



$Q = S0, S1, S2, S3$

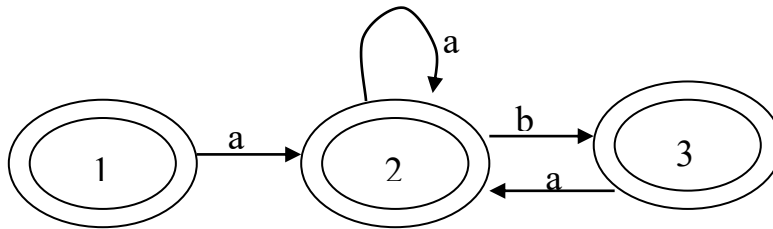
$\Sigma = a, b$

$S0 = S0$

$F = S3$

$\delta: (S0, b) \rightarrow S0$, $\delta: (S0, a) \rightarrow S1$, $\delta: (S1, a) \rightarrow S1$, $\delta: (S1, b) \rightarrow S2$,
 $\delta: (S2, a) \rightarrow S1$, $\delta: (S2, b) \rightarrow S3$, $\delta: (S3, b) \rightarrow S1$.

Cadenas válidas abb, aabb, babb, abbbabb
 Cadenas no validas: b, a, bbab, aaabbb



$Q = 1,2,3$

$\Sigma = a,b$

$S_0 = 1$

$F = 1,2,3$

$\delta: (1 a) \rightarrow 2, \delta: (2 a) \rightarrow 2, \delta: (2 b) \rightarrow 3, \delta: (3 a) \rightarrow 2.$

Cadenas válidas $\lambda, a, ab, aba, abaaaa, aaaaaaa$

Cadenas no validas: abbb, bbb, bab, c, ababbb

2.1.2 AUTÓMATAS FINITOS NO DETERMINISTA (AFND)

Autómata finito no determinista: Es aquella modificación al modelo del autómata finito determinista, para permitir más de una transición de un estado sobre el mismo símbolo de entrada que llegan a dos o más estados diferentes. En otras palabras es aquel que estando en un estado origen (so) con una misma entrada “a” llega a dos o más estados diferentes. Así también es aquel que tiene transiciones con el vacío. Que contenga transiciones con el vacío no implica que acepte cadenas vacías.

Ejemplo: $\delta: (q_1, a) \rightarrow q_1$ y $\delta: (q_1, a) \rightarrow q_2$.

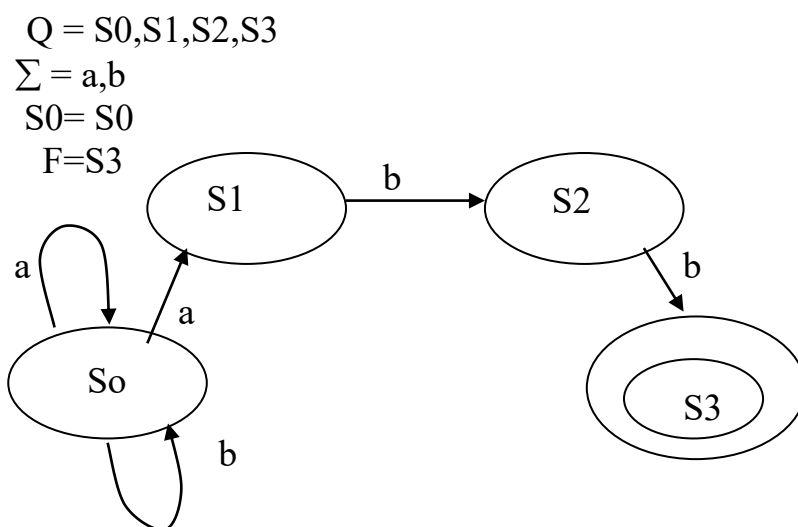
Un autómata finito no determinista M es una 5-tupla o quinteta que se define como sigue:

$$M = \{Q, \Sigma, s, F, \Delta\}$$

Donde:

- Q = Es una colección finita de estados.
- Σ = Es el alfabeto de entrada al autómata.
- s = Es el estado inicial.
- F = Es una colección de estados finales o de aceptación.
- Δ = Es una relación sobre $(Q \times \Sigma) \times Q$ y se llama relación de transición, esto indica que, para todo estado Q , se pueden tener cero o

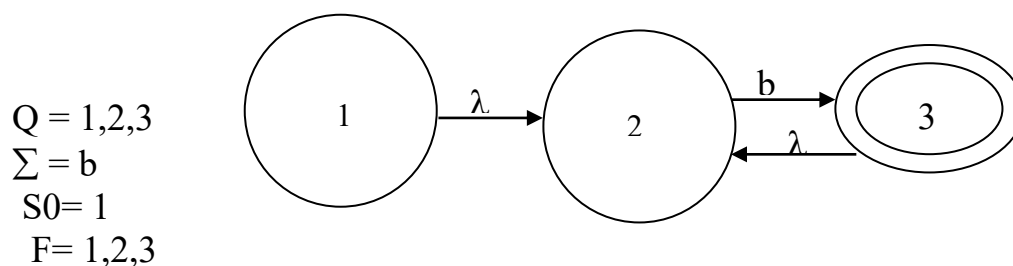
más alternativas a elegir como estado siguiente, todas para el mismo símbolo de entrada.



Cadenas válidas aabb, ababb, abb.

Cadenas no validas: aaaa, bbbbb, aaaabab.

NOTA: Como en el caso anterior, se llega a dos estados (S0, S1) con la misma entrada "a" por lo tanto es un autómata finito no determinista



Cadenas válidas b, bb, bbbb, bbb.

Cadenas no validas: b, a.

2.2 EXPRESIONES REGULARES

Una expresión regular es un patrón que describe un conjunto de cadenas válidas que se construyen basándose en un alfabeto y un conjunto de reglas.

La construcción de expresiones regulares depende de la asignación del significado especial para algunos caracteres. A éstas se les denomina cerraduras y entre ellas tenemos las siguientes:

- Cerradura Kleene o de estrella (*)
- Cerradura positiva (+)

Si A es un lenguaje sobre algún alfabeto (Σ), se define *Cerradura de Kleene* ó *cerradura de estrella (*)* de A como:

$$A^* = \bigcup_{n=0}^{\infty} A^n$$

Se dice que un cierre reflexivo A^* es la unión de sus potencias de A comenzando de la potencia cero es decir:

$$A^* = A^0 \cup A^1 \cup A^2 \cup A^3 \cup \dots \cup A^n$$

De tal manera que A se repetirá cero o más veces, por lo tanto en cualquier expresión regular que contenga el símbolo de * se interpretará de igual manera en cualquier caso. Ejemplo:

Sea a^*b^* una expresión regular que se leerá como que el elemento “a” vendrá cero o más veces seguido del elemento “b” cero o más veces. Las cadenas que se pueden generar son:

aaaaa, a, ab, b, bbbbbb, aaabb, λ , entre otras.

Si A es un lenguaje sobre algún alfabeto (Σ), se define *Cerradura positiva* de A como:

$$A^+ = \bigcup_{n=1}^{\infty} A^n$$

Se dice que un cierre transitivo A^+ , que es la unión de sus potencias de A, pero comenzando de la potencia uno es decir:

$$A^+ = A^1 \cup A^2 \cup A^3 \cup \dots \cup A^n$$

De tal manera que A se repetirá una o más veces, por lo tanto en cualquier expresión regular que contenga el símbolo de + se interpretará de igual manera en cualquier caso. Ejemplo:

Sea a^+b^+ una expresión regular que se leerá como que el elemento “a” vendrá una o más veces seguido del elemento “b” una o más veces. Las cadenas que se pueden generar son, aaaaab, ab, abbb, aabb, entre otras. Nótese que la

diferencia con la cerradura de Kleene es que aquí no se acepta la cadena vacía λ y que después de una “a” forzosamente viene una “b” y que antes de la primera “b” viene mínimo una “a”.

Por lo que podremos decir que el cierre reflexivo A^* se puede representar de la siguiente manera:

$$A^* = A^+ \cup A^0 = A^+ \cup \{ \lambda \}$$

Para describir un conjunto de reglas que se usan para construir una expresión regular se usan también los siguientes símbolos:

- $()$ que significan agrupación.
- $|$ que significan alternación.

La prioridad para la evaluación de las expresiones es: $()$, $|$, y * , $^+$ tienen la misma prioridad y se evalúan de izquierda a derecha.

Cadena válida: Un miembro de este conjunto de cadenas se dice que es válida cuando satisface la expresión regular. Ejemplo:

De la expresión regular: a^+b^* tenemos como cadenas válidas a:
abbb, aaaaa, aaabb, ab.

Cadena no válida: Es un miembro de un conjunto de cadenas que no satisface la expresión regular. Ejemplo:

De la expresión regular: a^+b^* tenemos como cadenas no válidas a:
bbbb, ababa, baaaa, baba, abbccc

Otra notación de las expresiones regulares son los exponentes, éstas pueden realizar a su vez las funciones de la cerradura de estrella y la cerradura positiva. Ejemplo:

A^i donde $i \geq 0$ es equivalente a la cerradura de estrella.

A^i donde $i \geq 1$ es equivalente a la cerradura positiva.

Cuando en una expresión los elementos tienen exponentes y con estos se definen condiciones, en donde un exponente dependa de otro, para formar cadenas, se dice que son **expresiones no regulares**. Ejemplos:

$a^i bc^i$ donde $i > 0$. Esta expresión tiene elementos con exponentes, cuando $i=1$ la cadena válida es abc , cuando $i=2$ la cadena válida es $aabcc$, por lo que elemento “c” depende del comportamiento que tenga “a”, ya que el mismo número de c’s debe ser igual al número de a’s.

Cadenas válidas:
aaaabccccc, aabcc, aaaccc

Cadenas no válidas:
aaabccccc, bbbb, aaac

La expresión $a^i b^j$, donde i distinto de j . El exponente “i” depende del comportamiento del exponente que tenga “j” y viceversa. Cuando $i=0$, j debe ser distinto de 0, cuando $j=0$, i debe ser distinto de 0.

Cadenas válidas:
bbb, aaaa, aab, abb, a, b

Cadenas no válidas:
ab, abab, baba, aabb, aaaacbbbb

Para poder expresar todo un conjunto de cadenas para un lenguaje, éste puede estar definido por reglas o por medio de expresiones a través de cierres reflexivos y transitivos. Por ejemplo:

Genere un lenguaje que acepte todas las cadenas que sean válidas para aceptar identificadores en el lenguaje Pascal.

Objetivo: definir identificadores validos en lenguaje Pascal para los alumnos de teoría de la computación.

Alfabeto definido en símbolos: 0,1,2,3,4,5,6,7,8,9, a...z, A..Z, _.

Alfabeto definido en palabras: conjunto de letras (l), conjunto de números(n), guion bajo (_).

Reglas para definir identificadores:

- 5) Todos los identificadores empiezan con letra.
- 6) Identificadores terminan con letra y número
- 7) Puede existir un guión bajo (_) en medio de la cadena, pero no al principio ni al final
- 8) La longitud de la cadena debe ser mayor de 1 caracteres

Ejemplos de identificadores válidos:

Contador_1, Contador, Contador2, etc.

Ejemplos de identificadores no válidos:

contador, 211mega, qqqqqqqqqqqq, contador, etc.

Por expresión regular es:

$$1(d|1|_1|_d)^*$$

Genere un lenguaje que acepte números reales

Objetivo: definir números reales para los alumnos de teoría de la computación.

Alfabeto definido en símbolos: ., 0,1,2,3,4,5,6,7,8,9.

Alfabeto definido en palabras: conjunto de números (d), punto(.).

Reglas para definir números reales:

- 1) Todos los números enteros con punto o sin punto.
- 2) Solo un punto en toda la cadena
- 3) Puede existir un (.) al principio o al final
- 4) La longitud de la cadena debe ser mayor de 1 caracteres

Ejemplos:

Cadenas válidas:

4,5.4, 4.02, .1402, 5., etc.

Cadenas no válidas:

5..2, 4.., .4.4., etc.

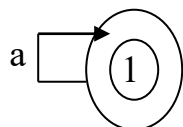
La expresión regular es: $d^+ | d^+ . d^+ | .d^+$

Los lenguajes aceptados por un autómata finito se describen mediante enunciados que definen las reglas del lenguaje o por expresiones simples llamadas expresiones regulares.

Las expresiones regulares utilizan cierta simbología (*, +, |, ()) cuyas propiedades de cada símbolo han sido definidas anteriormente.

Los símbolos de las expresiones regulares se interpretan en un AFD de la siguiente manera:

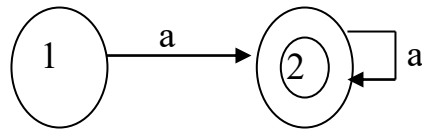
, Se define como cero o más elementos. Cuando se tiene en una expresión “a”, su representación en un autómata es:



Por lo general cuando existe un * se convierte en un lazo.

Cuando la expresión regular acepta la cadena vacía, el estado de inicio es de aceptación.

$+$, Se define como una o más veces. Cuando se tiene en una expresión “ a^+ ”, su representación en un autómata es:

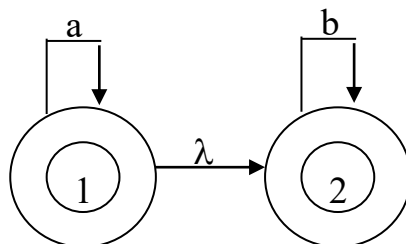


Por lo general cuando existe un $+$ se utilizan dos estados, el primero que garantice que al menos viene un elemento, y en el segundo para que realice el ciclo correspondiente.

Cuando la expresión no acepta la cadena vacía, el estado de inicio no es de aceptación.

Cuando en una expresión regular existen dos o más elementos y cada uno de ellos tiene un símbolo se pueden dar los siguientes casos:

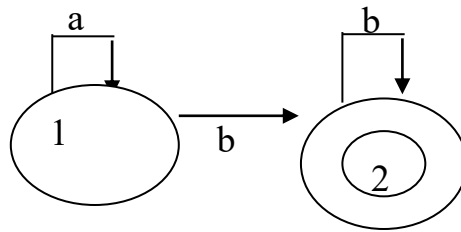
****** Cuando se tiene en una expresión “ $a^* b^*$ ”, su representación en un autómata es:



En realidad la expresión esta compuesta de dos partes por lo que se necesitan dos estados uno que realice la parte “ a^* ” y otro que realice la parte “ b^* ” y un flujo que conecte al primer estado con el segundo, el símbolo que contenga dicho flujo será “ λ ”. Cuando exista la unión de más de dos $*$ ($a^* b^* c^*$) aplique esta misma regla pero para la parte “ a ” cree dos flujos uno que vaya a la parte “ b^* ” con “ λ ” y otro a la parte “ c^* ” también con “ λ ”.

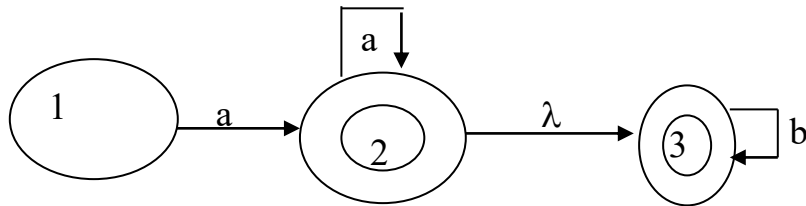
En otras palabras se necesita un estado por cada símbolo que contenga un $*$, cada uno de ellos contendrá un lazo con el símbolo, que conforme al orden que presente en la expresión regular, de estado a estado existirá un flujo que separe una parte de la otra y el símbolo que lleve dicho flujo será el que λ .

+ Cuando se tiene en una expresión “ $a^ b^+$ ”, su representación en un autómata es:



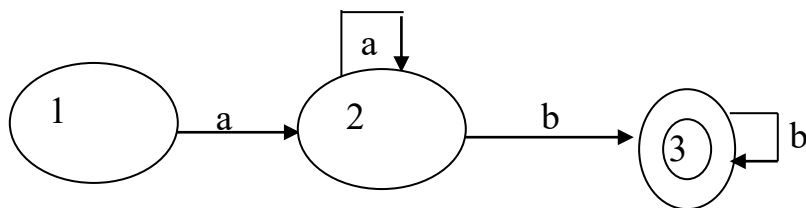
En realidad para la primera parte se utiliza la misma representación que a^* salvo que como la expresión no acepta cadenas vacías el estado de inicio no debe ser de aceptación, para la parte “ b^+ ” se aplica la misma filosofía del símbolo + partiendo del estado que contiene el lazo de “a”.

+* Cuando se tiene en una expresión “ $a^+ b^*$ ”, su representación en un autómata es:



En realidad para la primera parte se utiliza la misma representación que a^+ , para la parte “ b^* ” se aplica la misma filosofía del símbolo *, solo que debe existir un flujo el cual sale del estado que contiene el lazo de “a” hacia el estado que contenga el lazo de “b” con el símbolo de “ λ ”.

++ Cuando se tiene en una expresión “ $a^+ b^+$ ”, su representación en un autómata es:



En realidad para la primera parte se utiliza la misma representación que a^+ , al igual que para la segunda se aplica b^+ , como la expresión tiene que garantizar que después de una “a” venga una “b” el estado de aceptación se recorre de estado en estado hasta que garantice la cadena mínima.”ab”.

Conclusión:

- a) Cada vez que una parte “a” se une con una parte “b* ” se necesita crear un flujo con “λ”, que vaya de la parte “a” a la parte “b” y en esta parte debe existir el lazo de “b”.
- b) Cada vez que una parte “a” se une con una parte “b+ ” se necesita crear un flujo con “b”, que vaya de la parte “a” a la parte “b”, con “b” y en esta parte debe existir el lazo de “b”.

Cuando se diseñan autómatas a partir de expresiones regulares se pueden generar los dos tipos de autómatas finitos, pero lo ideal es siempre generar autómatas finitos deterministas (AFD), en caso de generar No Deterministas (AFND), es posible transformarlo a un AFD.

En la tabla 3. Se puntualizan las diferencias entre un autómata finito determinista y un autómata finito no determinista.

Tabla 3. Diferencias entre AFD y AFND

	Estando en un estado origen (S0) con una misma entrada “a” llega a dos o más estados diferentes.	Contienen transiciones con el vacío
AFND	✓	✓
AFD	X	X

NOTA: Que contengan transacciones vacías no implica que acepte cadenas vacías.

Equivalencia de AFND Y AFD

Cada AFD es un AFND, queda claro que la clase de lenguaje aceptado por los AFND incluye a los conjuntos regulares (los lenguajes aceptados por los AFD), sin embargo, resulta que éstos son los únicos conjuntos aceptados por los AFND. La demostración consiste en mostrar que los AFD pueden simular a los AFND; esto es, para cada AFND podemos construir un AFD equivalente (uno que acepta el mismo lenguaje). La forma en que un AFD simula a un AFND consiste en permitir que los estados del AFD correspondan a conjuntos de estados del AFND, el AFD que se ha construido mantiene, en su control finito, el rastro de todos los estados en los que el AFND puede estar después de leer la misma entrada que el AFD ha leído. La construcción formal se encuentra en un siguiente teorema:

Sea L un conjunto aceptado por un autómata finito no determinista, entonces existe un autómata finito determinista que acepta a L .

Un método que permite pasar de un AFND a AFD es la generación de la tabla de transiciones.

Tabla de transición corresponde al valor de $f(q_i, e_j)$, donde q_i es el elemento i -ésimo de Q , y e_j es el elemento j -ésimo de Σ . Tanto el estado inicial como los de aceptación por $\rightarrow$ y por $*$ respectivamente. Nota: los estados finales o de aceptación pueden ser indicados también rodeando el estado por un círculo.

Estados/ Σ	$(\Sigma): a_1 \dots a_n$
$\rightarrow q_0$	
.....	
q_i	
.....	
q_f	

Ejemplo: AFD

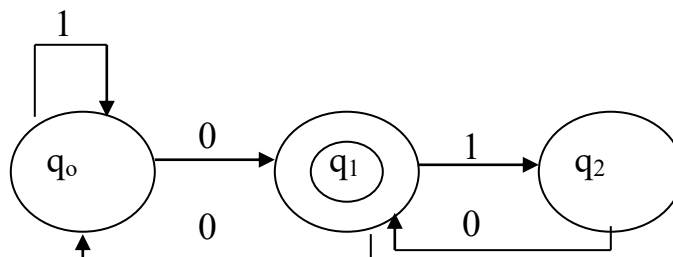


Tabla de Transición

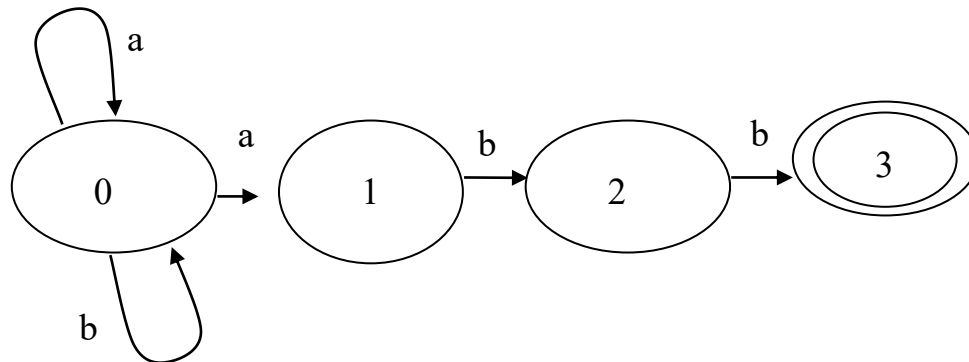
Estados/ Σ	0	1
$\rightarrow q_0$	q_1	q_0
q_1	q_0	q_2

q ₂	q ₁	-
----------------	----------------	---

Cadenas validas: 10, 1010, 11110, 11001110

Cadenas no validas: 1111, 101, 1100001

Ejemplo: *AFND*



Su expresión regular es: $(a|b)^* abb$.

Cadenas validas: aabb, aaaabbbbabb, bbbbaaabb.

Cadenas no validas: aaa, bbb, baa, aaabbbab.

Tabla de Transición

F	A	B
→ 0	0,1	0
1	-	2
2	-	3
3	-	-

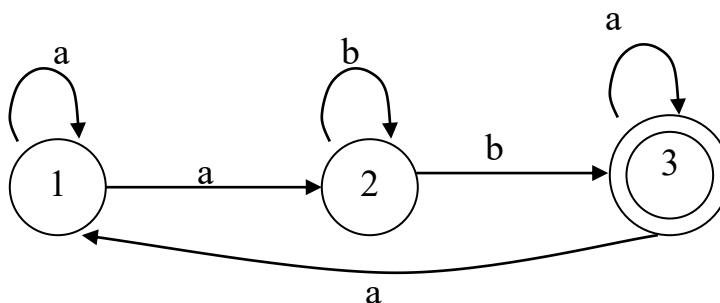


NOTA: Como en este caso, como en la entrada A se va del estado "0" a dos estados diferentes (0 y 1), se necesitan anotar los dos estados hacia los cuales llega el estado A (0 y 1)

Como ya se menciono existen dos casos de AFND y para cada uno se realizan diferentes pasos para convertirlos a AFD.

A continuación se explica como se puede convertirse un *AFND* a un *AFD* cuando no tiene transiciones vacías, y que mejor que con dos ejemplos que empieza en el siguiente autómata.

Ejemplo 1:



1.-Primero se pasan los estados a una tabla de transición.

Estados/ Σ	a	b
1	1,2	
2		2,3
3	3, 1	

2.-Generar los nuevos estados, observe su tabla y si en algún casillero o celda existen dos o más estados se va a generar un nuevo estado, cuyo nombre se dará conforme a la concatenación de estos números o elementos, si el estado resultante de la concatenación es igual a un estado existente, asígnele un nombre representativo a dicha concatenación, siempre y cuando dicho estado no haya sido generado por una concatenación anterior, y en el caso de que ya se haya generado, ya no es necesario realizar el nuevo estado. También son equivalentes las concatenaciones que cuenten con los mismos estados sin importar el orden en que estén.

Nota: Si alguno de los estados es de aceptación automáticamente el nuevo estado será de aceptación.

Estados/ Σ	a	b
1, 2		
(2,3)		
(3, 1)		

Como el estado 3 es de aceptación el estado 3,1 y 2,3 se hace de aceptación

3.- A los nuevos estados generados colóqueles en sus celdas las transiciones correspondientes de cada una de los estados que se concatenaron.

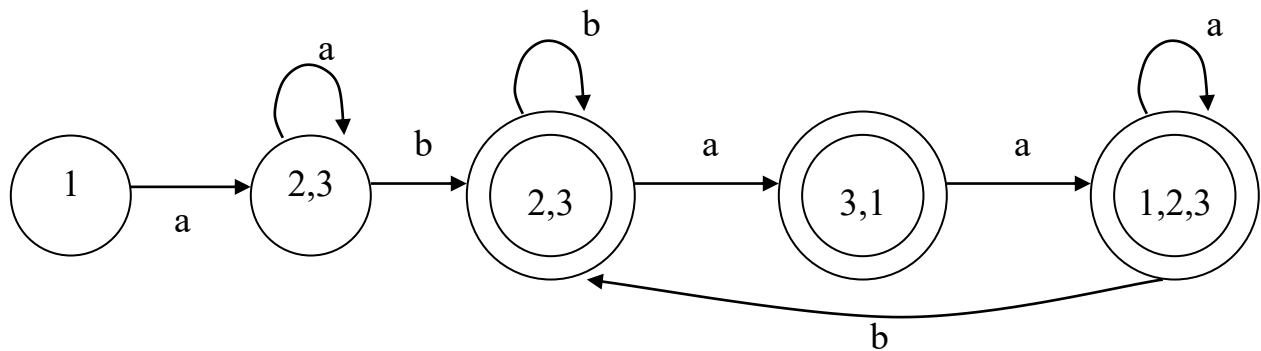
Estados/ Σ	a	b
1, 2	1,2	2,3
(2,3)	3,1	2,3
(3,1)	1,2,3	

4.-Si en los nuevos estados generados, existe también la unión de dos o más estados se realizan el paso 2 y así sucesivamente hasta que ya no se generen nuevas uniones.

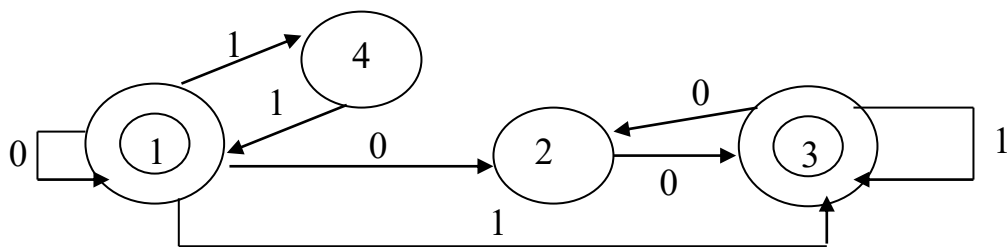
Estados/ Σ	a	b
1	1,2	
2		2,3

3	3,1	3,1
1,2	1,2	3,4
2,3	3,1	3,4
3,1	1,2,3	
1,2,3	1,2,3	2,3

El *autómata finito determinista* queda de la siguiente forma:



Ejemplo 2:



1.-Primero se pasan los estados a una tabla de transición.

Estados.	0	1
1	1,2	3,4

2	3	
3	2	3
4		1

2.-Generar los nuevos estados, observe su tabla y si en algún casillero o celda existen dos o más estados se va a generar un nuevo estado, cuyo nombre se dará conforme a la concatenación de estos números o elementos, si el estado resultante de la concatenación es igual a un estado existente, asígnele un nombre representativo a dicha concatenación, siempre y cuando dicho estado no haya sido generado por una concatenación anterior, y en el caso de que ya se haya generado, ya no es necesario realizar el nuevo estado. También son equivalentes las concatenaciones que cuenten con los mismos estados sin importar el orden en que estén.

Nota: Si alguno de los estados es de aceptación automáticamente el nuevo estado será de aceptación.

Estados.	0	1
12		
34		

Como el estado 1 es de aceptación el estado 12 se vuelve de aceptación

Como el estado 3 es de aceptación el estado 34 se vuelve de aceptación

3.- A los nuevos estados generados colóqueles en sus celdas las transiciones correspondientes de cada una de los estados que se concatenaron.

Estados	0	1
12	123	34
34	2	31

4.-Si en los nuevos estados generados, existe también la unión de dos o más estados se realiza el paso 2 y así sucesivamente hasta que ya no se generen nuevas uniones.

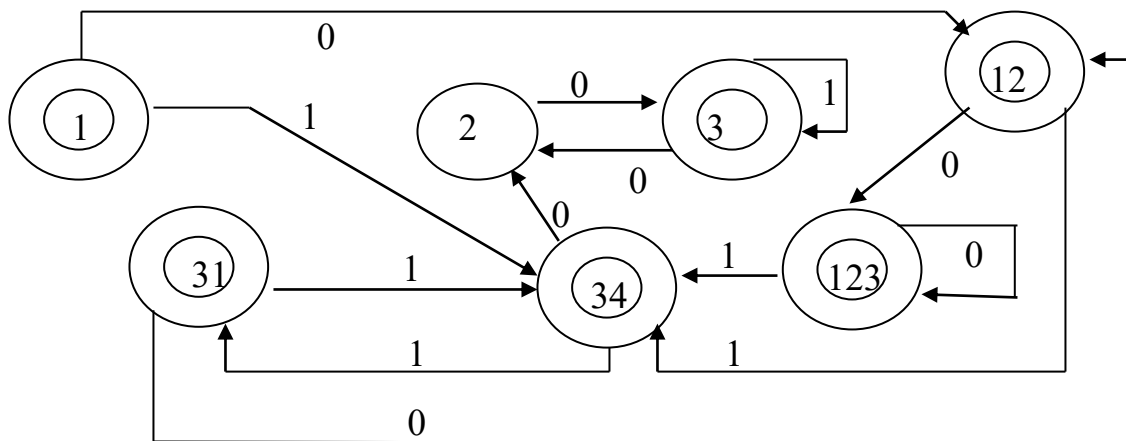
Estados	0	1
1	12	34
2	3	
3	2	3
4		1
12	12	34
34	2	31
123	123	34
31	12	34

5.-Si ya no se generó ningún estado nuevo se crea otro autómata. El cuál estará compuesto por los estados que se llamen en la tabla resultante, comenzando por el estado de inicio, no necesariamente se llamarán a todos los estados que componen la tabla.

Estados	0	1
1	12	34
12	123	34
34	2	31
123	123	34

31	12	34
2	3	
3	2	3

El *autómata finito determinista* queda de la siguiente forma:



Si existen dos o más estados los cuales tienen las mismas transiciones con el mismo símbolo y a su vez dichos estados son de aceptación o en su defecto ninguno es de aceptación, se dice que son *estados equivalentes*.

Si en la tabla de transición existen estados equivalentes, entonces se dice que el autómata es un *autómata finito determinista no reducido* (AFDNR.)

La tarea de realizar un *autómata determinista reducido* (AFDR) es sumamente importante para los lenguajes de programación ya que se economizan líneas y se quitan tareas repetitivas.

Para pasar un AFDNR a un AFDR se siguen los siguientes pasos los cuales se ejemplifican con el ejercicio siguiente:

Paso 1: Identifique los estados equivalentes.

Para este ejemplo: Los estados 1 y 31 son equivalentes; así mismo los estados 12 y 123.

Paso 2: Existen dos formas de realizar este paso a continuación se explica cada una de ellas.

2.1) De los estados en los cuales se presenta la equivalencia deje Solamente uno y elimine el resto, en la tabla donde llame Los estados eliminados sustitúyalos por el que dejó. Tome en Cuenta que el estado de inicio no desaparece, en todo caso Cambiará el símbolo de inicio.

Para el ejemplo: Como $1 = 31$ vamos a dejar el estado 1. Y como $12 = 123$ vamos a dejar el estado 12.

estados	0	1
1	12	34
12	12	34
34	2	1
123	12	34
31	12	34
2	3	
3	2	3

Paso 3: Una vez hecho el paso 2.1 observará que en la tabla existen estados que no son llamados, elimínelos de la tabla.

Para el ejemplo: Se elimina los estados 31 y 123 ya que no son llamados.

Estados	0	1
1	12	34

12	12	34
34	2	1
2	3	
3	2	3

2.2) A los estados en los cuales se presenta la equivalencia asígneles un nombre en común y en la tabla donde quiera que se encuentre a uno de estos sustitúyalo por su nuevo nombre.

Para el ejemplo: Como $1 = 31$ vamos a llamarlos 1. Y como $12 = 123$ los denominaremos 5.

Estados	0	1
1	5	34
5	5	34
34	2	1
5	5	34
1	5	34
2	3	
3	2	3

Paso 3: Una vez realizado el paso 2.2 observara que en la tabla existen dos o más renglones iguales deje solamente uno de ellos.

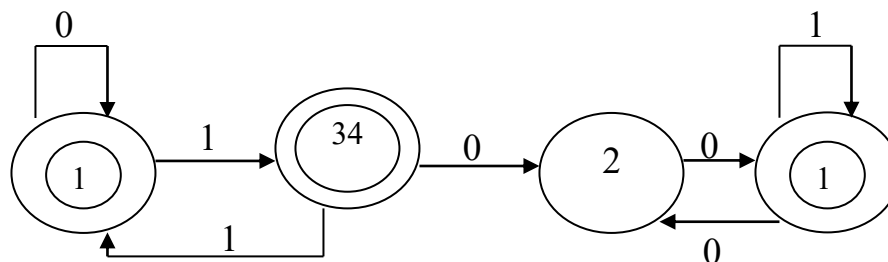
Estados	0	1
---------	---	---

1	5	34
12	12	34
34	2	1
2	3	
3	2	3

Paso 4: Revise que en su tabla no se generaron nuevos estados equivalentes en caso de que existan realice el paso dos para dichos estados.

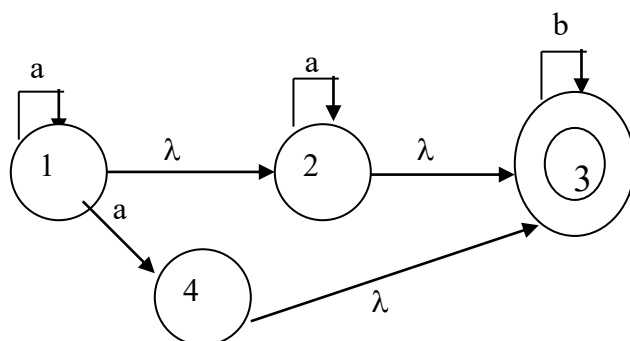
Estados.	0	1
1	1	34
34	2	1
2	3	
3	2	3

Paso 5: Cuando ya no existan estados equivalentes genera el nuevo autómata AFDR.



A continuación se explica como un AFND se pasa a AFD, *cuando tiene transiciones vacías.*

Considere el autómata para ejemplificar dicha explicación.



1.-Se genera la tabla de transición incluyendo una columna con el símbolo de vacío:

Para el ejemplo:

Estados.	a	b	$\square\square\square\lambda$
$\rightarrow 1$	14		2
2	2		3
3		3	
4			3

2.-Generación de nuevos estados de aceptación. Si en la intersección de un estado “a” con el vacío $\lambda\square$, cuenta con uno o más estados, y al menos uno de ellos es de aceptación, se crea un nuevo estado de aceptación que será el estado “a”. Si el estado “a” es de aceptación ya no es necesario ver su intersección con vacío.

Para el ejemplo los nuevos estados generados son 2 y 4.

Estados.	a	b	λ
$\rightarrow 1$	14		2

2	2		3
3		3	
4			3

Si se da el caso que algún estado “b” llame a un nuevo estado de aceptación “a’ ” y “b” no es de aceptación entonces se convierte también un nuevo estado de aceptación.

Como el 2 es de aceptación el estado1 se hace de aceptación:

Estados	a	b	λ
→ 1	14		2
2	2		3
3		3	
4			3

3.- En el casillero o celda donde se da la intersección de un estado “a” con el vacío, puede existir uno o más estados “b”, entonces todo lo que tenga “b” se sube al renglón del estado “a”.

Para ejemplo al estado 1 se le pega el estado 2, al 2 el 3 y al 4 el 3:

Estados.	a	b	λ
→ 1	142		2
2	2	3	3
3		3	

4			3
---	--	--	---

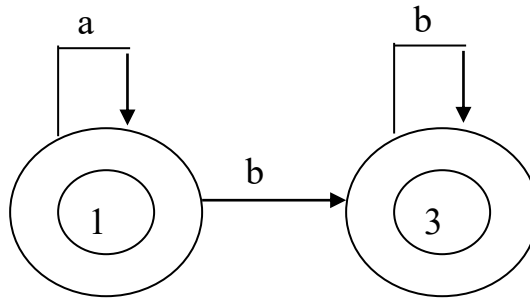
4.- Se vuelve a dar otra pasada hasta que ya no se pueda subir más estados.

Estados.	a	b	λ
→ 1	142	3	2
2	2	3	3
3		3	
4		3	3

5.- Cuando ya no suba más estados, elimine la columna de vacíos y manéjelo como un no determinista sin transiciones vacías.

Estados.	A	B
→ 1	142	3
2	2	3
3		3
4		3
142	142	3

Para el ejemplo el autómata final es:



NOTA:

Se considera una regla, para la reducción de estados, así como para pasar de un AFND a AFD, que el símbolo de inicio no desaparezca por ningún motivo, ya que genera una confusión sobre cuál es el estado de inicio.

2.3 LENGUAJES REGULARES

Un lenguaje formal es un conjunto de palabras (cadenas de caracteres) de longitud finita en los casos más simples o expresiones válidas (formuladas por palabras) formadas a partir de un alfabeto (conjunto de caracteres) finito. El nombre lenguaje se justifica porque las estructuras con las que se forman tienen reglas de buena formación (gramática) e interpretación semántica (significado) en una forma muy similar a los lenguajes hablados o naturales.

Ejemplos de lenguajes formales

- El conjunto de todas las palabras sobre $\{a, b\}$.
- El conjunto $\{a^n : n \text{ es un número primo}\}$.
- El conjunto de todos los programas sintácticamente válidos en un determinado lenguaje de programación.
- El conjunto de sentencias bien formadas en lógica de predicados.

Especificación de lenguajes formales

Los lenguajes formales se pueden especificar de una amplia variedad de formas, como por ejemplo:

- Cadenas producidas por una gramática formal (Jerarquía de Chomsky).
- Cadenas producidas por una expresión regular.
- Cadenas aceptadas por un autómata.

Se pueden utilizar varias operaciones para producir nuevos lenguajes a partir de otros dados. Supóngase que L_1 y L_2 son lenguajes sobre un alfabeto común.

Entonces:

- la *concatenación* L_1L_2 consiste de todas aquellas palabras de la forma vw donde v es una palabra de L_1 y w es una palabra de L_2
- la *intersección* $L_1 \& L_2$ consiste en todas aquellas palabras que están contenidas tanto en L_1 como en L_2
- la *unión* $L_1 | L_2$ consiste en todas aquellas palabras que están contenidas ya sea en L_1 o en L_2
- el *complemento* $\sim L_1$ consiste en todas aquellas palabras producibles sobre el alfabeto de L_1 que no están ya contenidas en L_1
- el *cociente* L_1 / L_2 consiste de todas aquellas palabras v para las cuales existe una palabra w en L_2 tales que vw se encuentra en L_1
- la *estrella* L_1^* consiste de todas aquellas palabras que pueden ser escritas de la forma $W_1W_2...W_n$ donde todo W_i se encuentra en L_1 y $n \geq 0$. (Nótese que esta definición incluye a ϵ en cualquier L^*)
- la *intercalación* $L_1^*L_2$ consiste de todas aquellas palabras que pueden ser escritas de la forma $v_1w_1v_2w_2...v_nw_n$; son palabras tales que la concatenación $v_1...v_n$ está en L_1 , y la concatenación $w_1...w_n$ está en L_2

Un lenguaje regular sobre un alfabeto Σ dado se define recursivamente como:

- El lenguaje vacío $\emptyset$ es un lenguaje regular
- El lenguaje cadena vacía $\{\epsilon\}$ es un lenguaje regular
- Para todo símbolo $a \in \Sigma$ $\{a\}$ es un lenguaje regular
- Si A y B son lenguajes regulares entonces $A \cup B$ (unión), $A \bullet B$ (concatenación) y A^* (clausura o estrella de Kleene) son lenguajes regulares
- Si A es un lenguaje regular entonces (A) es el mismo lenguaje regular
- No existen más lenguajes regulares sobre Σ

Todo lenguaje formal finito constituye un lenguaje regular. Otros ejemplos típicos son todas las cadenas sobre el alfabeto $\{a, b\}$ que contienen un número par de a 's o el lenguaje que consiste en varias a 's seguidas de varias b 's.

Todas las expresiones regulares se pueden representar mediante autómatas finitos y todos los autómatas finitos representan *lenguajes regulares*.

Todo lo que es verdad para lenguajes regulares también es verdad para lenguajes aceptados por un autómata finito y viceversa. Así, por ejemplo la colección de lenguajes regulares es cerrada con respecto a la concatenación,

unión y cerradura de estrella por que los lenguajes aceptados por un autómata finito también lo son.

Existen pocos lenguajes que sean regulares y, en el caso de un lenguaje infinito la búsqueda exhaustiva de una expresión regular o de un AF puede resultar inútil.

EJERCICIOS PROPUESTOS

1) Genere autómatas de las siguientes expresiones:

- 1.1) $(a|b)^*abb$
- 1.2) $a^*(ba^+)bb^+$
- 1.3) $(r(sr^+)^+)$
- 1.4) $a^*b^*c^*$
- 1.5) b^+c^*

2) Obtenga su expresión regular, y genere AFD.

2.1)

$So \rightarrow aSo | a$

2.2)

$Vo \rightarrow xVo$

$Vo \rightarrow yV_1$

$V_1 \rightarrow yV_1$

$V_1 \rightarrow z$

$T \{x, y, z\}$

$NT \{Vo, V_1\}$

2.3)

$Vo \rightarrow xVo$

$Vo \rightarrow yVo$

$Vo \rightarrow z$

$T \{x, y, z\}$

$NT \{Vo\}$

2.4)

$Vo \rightarrow a + V_1$

$V_1 \rightarrow a + V_2$

$V_2 \rightarrow a + V_2$

$V_2 \rightarrow a$

$T \{a, +\}$

$NT \{Vo, V_1, V_2\}$

2.5)

$Vo \rightarrow aV_1$

$V_1 \rightarrow bVo$

$V_1 \rightarrow a$

$T \{a, b\}$

$NT \{Vo, V_1\}$

2.6)

$V_0 \rightarrow V_0 V_1$	
$V_0, V_1 \rightarrow V_2 V_0$	
$V_2 V_0 \rightarrow xy$	$T \{x, y, z\}$
$V_2 \rightarrow x$	$NT \{V_0, V_1, V_2\}$
$V_1 \rightarrow z$	

2.7)

$S_0 \rightarrow b S_0 \mid aA \mid b$	$T \{a, b\}$
$A \rightarrow aS_0 \mid bA \mid a$	$NT \{S_0, A\}$

2.8)

$S_0 \rightarrow A$	
$S_0 \rightarrow AAB$	
$Aa \rightarrow Aba$	
$A \rightarrow aa$	$T \{a, b\}$
$Bb \rightarrow Abb$	$NT \{A, B\}$
$AB \rightarrow ABB$	
$B \rightarrow b$	

2.9)

$S_0 \rightarrow BAB \mid ABA$	
$A \rightarrow AB$	
$B \rightarrow BA$	$T \{a, b\}$
$A \rightarrow aA \mid ab$	$NT \{S_0, A, B\}$
$B \rightarrow b$	

2.10)

$S \rightarrow bS \mid aA \mid a$	
$A \rightarrow aS \mid bB$	$T \{a, b\}$
$B \rightarrow bA \mid aS \mid b$	$NT \{S, A, B\}$

2.11)

$E \rightarrow bE \mid aO \mid b$	$T \{b, a\}$
$O \rightarrow aE \mid bO \mid a$	$NT \{E, O\}$

3) Genere autómatas finitos deterministas reducidos de las siguientes expresiones:

3.1) $(ba|b)^* (bb|a)^* c^*$

3.2) $((ab|aab)^* a^*)^*$

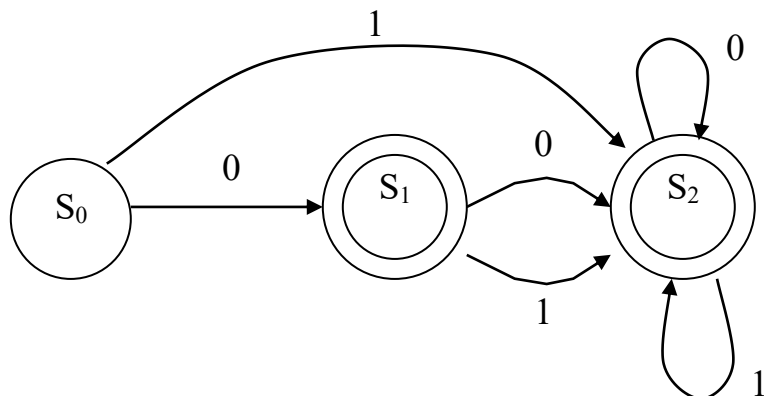
3.3) $b^* (ab+)^* a b^* | b^*$

3.4) $(a|b)^* bba$

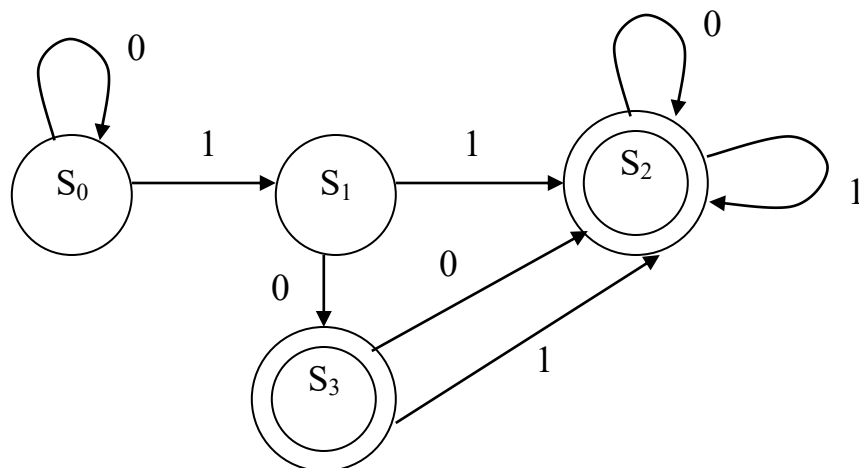
3.5) Todas las cadenas que terminen en ab

4) Dar la expresión regular de los siguientes autómatas.

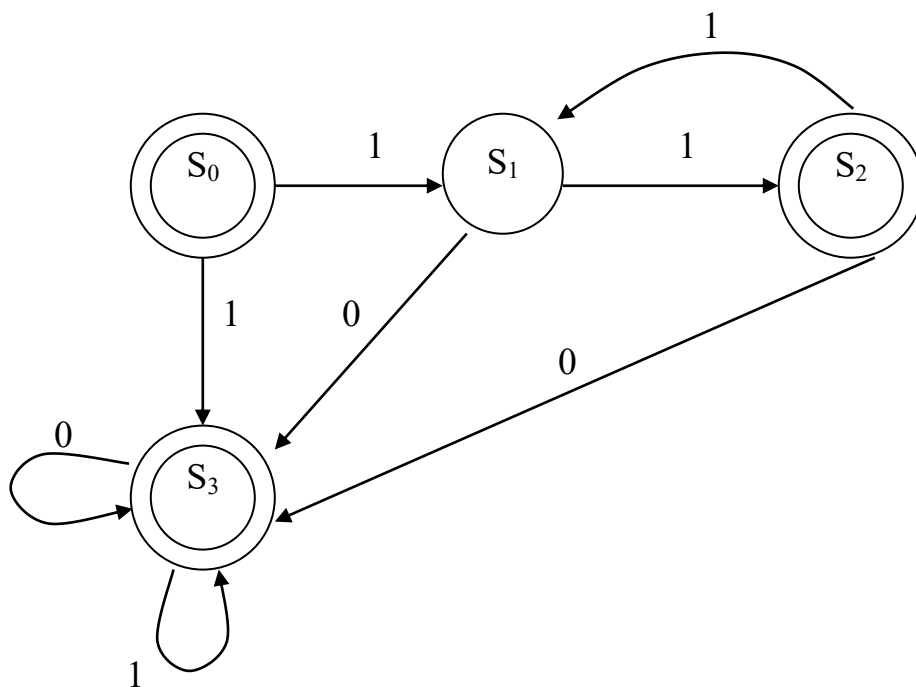
4.1)



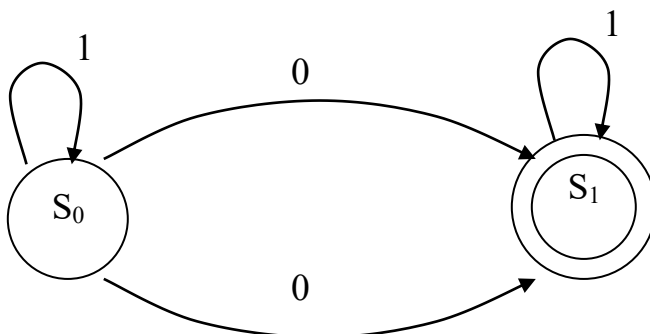
4.2)



4.3)



4.4)



5) Con el alfabeto $\{0, 1\}$ construye un autómatas que:

- 5.1) cadenas que empiecen con cero.
- 5.2) Cadenas que terminen en 101
- 5.3) cadenas que terminen en 00
- 5.4) Cadenas que tengan como subcadena 0011
- 5.5) Cadenas que empiecen 10 ó 001, y terminen en 010 ó 101.
- 5.6) Cadenas que tengan como subcadena 11011 ó 0110.
- 5.7) $(01(11)^*)^*$
- 5.8) $[((001|11)^*(00)) \mid (010)^+ (1|0)]^+$
- 5.9) $((001|11)^*(00)^*)^*$

CONSULTAS REALIZADAS

Alfred V. Aho, Monica S. Lam, Sethi Ravi y Jeffrey D. Ullman
Compiladores principios, técnicas y herramientas.
Editorial Addison-Wesley Iberoamericana.
2008.

Beck Leland
Software de sistemas. Introducción a la programación de sistemas.
Editorial Addison-Wesley Iberoamericana.
1988.

Jonh E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman
Introducción a la teoría de autómatas, Lenguajes y Computación.
Editorial PEARSON Addison Wesley.
2002.

Dean Kelley
Teoría de autómatas y lenguajes formales.
Editorial Prentice Hall
2006

Johnsonbaugh, Richard
Matemáticas discretas
Editorial Iberoamercana
2002.

Jean Paúl Trembay, Ram Manohar
Matemáticas discretas con aplicación a las ciencias de la computación
Editorial CECSA.
1996.

<http://www.wikiteka.com/apuntes/teou2/>

http://es.wikipedia.org/wiki/Lenguaje_regular

<http://lanzadobits.wordpress.com/>

http://wapedia.mobi/es/Lenguaje_formal

<http://webdia.cem.itesm.mx/ac/aesparza/compi/expreg.html>

<http://campus.gda.itesm.mx/~csainz/gram.htm>

<http://campus.gda.itesm.mx/~csainz/gramLR.htm>

<http://www.ls1.us.es/docencia/asignaturas/lfail.html>

<http://www.eluvigo.es/~cabrero/galena.dcg.html>

<http://www.guegue.com.ni/free/painrob/g1.html>

<http://www.hum.uva.ni/~teun/beliar.spa>

<http://148.247.2.21/TmTeAut.html>

<http://www.inf.udec.cl/~lenform/index.html>

<http://www.inf.udec.cl/~lenform/21.html>

3. LENGUAJES LIBRES DE CONTEXTO

Competencias que deberán ser adquiridas

- Capacidad de investigación
- Autonomía en el aprendizaje
- Creatividad
- Capacidad de Análisis y síntesis
- Organización y planificación
- Comunicación oral y escrita
- Utilización de las tecnologías de comunicación y de información (TIC'S)
- Trabajo en equipo

Objetivo de aprendizaje

Capacidad para diseñar un lenguaje a través de una gramática eliminando ambigüedad, recursión y factorización así como pasar a forma normal de Chomsky o de Greibach según sea el caso.

Resultado del aprendizaje	Actividad Educativa	Evaluación	Tiempo
Comprensión y capacidad de diseñar gramáticas libres de contexto.	• Búsqueda de conceptos por equipos en los que podrá leer, descubrir, cuestionar, preguntar indagar, sobre gramáticas libres de contexto. • Discusión en clase para expresarse, comunicar ideas, etc. • Creación de ideas originales obtenidas en consenso • Ejemplos de lenguajes representados por gramáticas independientes de contexto.	• Informes por equipo y conclusiones grupales. • Técnicas de observación(registros, listas de control) • Diseño de un lenguaje formal o natural representado por una GIC.	2 horas
Capacidad para transformar una gramática a la forma normal de Chomsky.	• Búsqueda de conceptos por equipos en los que podrá leer, descubrir, cuestionar, preguntar indagar sobre transformar una gramática a la forma normal de Chomsky. • Discusión en clase para expresarse, comunicar ideas, etc. • Clase teórica sobre la eliminación de vacíos	• Informes por equipo y conclusiones grupales. • Generación de un lenguaje formal o natural, representado en forma normal de chomsky,	2 horas

	Ejemplos de lenguajes representados por gramáticas transformado a forma normal de chomsky		
Capacidad para transformar una gramática a la forma normal de Greibach.	- Búsqueda de conceptos por equipos en los que podrá leer, descubrir, cuestionar, preguntar indagar sobre forma normal de Greibach. - Discusión en clase para expresarse, comunicar ideas, etc. - Ejemplos de lenguajes representados por gramáticas transformadas a forma normal de Greibach.	- Informes por equipo y conclusiones grupales. - Generación de un lenguaje formal o natural, representado en forma normal de Greibach.	2 horas
Capacidad para eliminar recursión, factorización y ambigüedad de una gramática	- Búsqueda de conceptos por equipos en los que podrá leer, descubrir, cuestionar, preguntar indagar sobre recursión, factorización y ambigüedad - Discusión en clase para expresarse, comunicar ideas, etc. - Ejemplos de lenguajes representados por gramáticas eliminando recursión, factorización y ambigüedad	- Informes por equipo y conclusiones grupales. - Generación de un lenguaje formal o natural, por una gramática, eliminando recursión, factorización y ambigüedad	2 horas
Diseñar un lenguaje por medio de un autómata de pila	- Búsqueda de conceptos por equipos en los que podrá leer, descubrir, cuestionar, preguntar indagar sobre autómatas de pila - Discusión en clase para expresarse, comunicar ideas, etc. - Ejemplos de lenguajes representados por autómatas de pila.	- Informes por equipo y conclusiones grupales. - Diseño de un lenguaje representado por autómata de pila y programarlo	
Capacidad para distinguir entre un lenguaje regular y un lenguaje no regular	- Búsqueda de conceptos por equipos en los que podrá leer, descubrir, cuestionar, preguntar indagar. - Discusión en clase para expresarse, comunicar ideas,	- Informes por equipo y conclusiones grupales. - Técnicas de observación(regis	2 horas

	etc. • Ejemplos de lenguajes regulares y no regulares	tros, listas de control) • Demostrar que un lenguaje es no regular	
--	--	---	--

3 LENGUAJES LIBRES DE CONTEXTO

3.1 GRAMATICAS LIBRES DE CONTEXTO

Gramática: Conjunto de reglas que denotan un lenguaje, por lo que las gramáticas son utilizadas para generar cadenas formadas correctamente sobre un alfabeto preestablecido. El símbolo de inicio o axioma, permite iniciar el proceso de generación de cadenas aceptables.

Definición: Una gramática es un sistema para definir un lenguaje, como también una herramienta para imponer a las reglas del lenguaje una estructura útil.

Estructura de las gramáticas

Con base a un alfabeto, se forman diferentes gramáticas, formadas por un conjunto de producciones (no terminales), un conjunto de elementos terminales, y un símbolo de inicio.

Una gramática G es una 4-tupla y se define como sigue:

$$G = \{T, NT, S_0, \rightarrow\}$$

S_0 (Símbolo de inicio), donde se deriva la primera producción de la gramática, ésta debe iniciar con una letra mayúscula y un subíndice que indique que es la primera producción y dentro de un árbol se indica como el nodo principal, por ejemplo:

$$V_0, S_0, R_0, \dots, Z_0 \text{ ó } V_1, S_1, \dots, Z_1$$

NT (No terminales) como su nombre lo indica, no terminan con la producción de la gramática, sino que estos derivan elementos NT y elementos terminales (T), éstos elementos deben ser representados con letras mayúsculas y dentro de un árbol se indica como el nodo padre. Las letras mayúsculas pueden ser:

$$A, B, C, \dots, Z. \text{ (siempre con mayúsculas).}$$

T (Terminales o alfabeto) como su nombre lo dice termina con la producción de la gramática, estos elementos deben ser representados por letras minúsculas y dentro de un árbol son representados por las hojas, por ejemplo:

$$a, b, c, \dots z. \text{ (siempre con minúsculas), hojas.}$$

→ (producciones) conjunto de producciones, las cuales especifican como se pueden combinar los terminales y los no terminales para formar cadenas también se representa con el símbolo $::=$.

Nota:

Algunos autores consideran como parte de la gramática al vacío y se denota y define así:

Vacío (λ), se le da éste nombre para producir dentro de una gramática nada o bien para terminar la producción de una gramática (cuando no se produce). En éste texto se considera como una producción más de la gramática.

Un ejemplo de una gramática es el siguiente:

Los elementos de la Gramática son:

$$\begin{aligned}T &= \{a, b\} \\ NT &= \{S_0, S_1\} \\ S_0 &= \{S_0\}\end{aligned}$$

El conjunto de producciones → es :

$$\begin{aligned}S_0 &\rightarrow aS_0 \\ S_0 &\rightarrow bS_1 \\ S_1 &\rightarrow bS_1 \\ S_1 &\rightarrow a\end{aligned}$$

Se usa un árbol de derivación para definir el lenguaje de la gramática, un árbol que está compuesto por raíz, ramas y hojas a esto se les conoce como árboles de análisis sintáctico.

3.2 ÁRBOLES DE DERIVACIÓN

Un árbol de derivación o de análisis sintáctico indica gráficamente la representación de la gramática y el símbolo inicial de una gramática inicia la derivación de una cadena del lenguaje.

Propiedades de los árboles de análisis sintáctico:

1. La raíz es etiquetada con el símbolo inicial
2. Cada hoja es etiquetada con un elemento terminal de la gramática o un elemento vacío (λ).

3. Cada nodo (rama) interior es etiquetado con un no terminal
4. Si A es el no terminal que etiqueta a algún nodo interior y $X_1, X_2, \dots, X_n$ son las etiquetas de los hijos de ese nodo, de izquierda a derecha, entonces $A \rightarrow X_1, X_2, \dots, X_n$ es una producción. Aquí, $X_1, X_2, \dots, X_n$ representa un símbolo que es un terminal o un no terminal. Como caso especial, si $A \rightarrow \epsilon$, entonces un nodo etiquetado con A tiene solo un hijo etiquetado con λ .

Las hojas de un árbol de análisis sintáctico, leídas de izquierda a derecha, forman la producción del árbol, que es la cadena generada o derivada del no terminal de la raíz del árbol de análisis sintáctico.

Cualquier árbol imparte un orden natural, de izquierda a derecha a sus hojas.

Tomando en cuenta el ejemplo de la gramática anterior queda de la siguiente manera generando el árbol de análisis sintáctico.

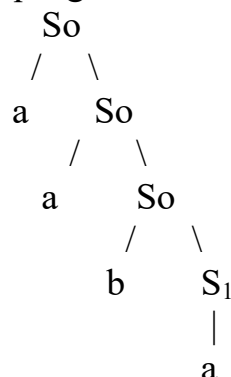
Los elementos de la Gramática son:

$$\begin{aligned} T &= \{a, b\} \\ NT &= \{S_0, S_1\} \\ S_0 &= \{S_0\} \end{aligned}$$

El conjunto de producciones es:

$$\begin{aligned} S_0 &\rightarrow aS_0 \\ S_0 &\rightarrow bS_1 \\ S_1 &\rightarrow bS_1 \\ S_1 &\rightarrow a \end{aligned}$$

El árbol de derivación que genera el conjunto de producciones es:



La cadena válida es: aaba

La expresión regular correspondiente es: a^+b^+a

Otro ejemplo es el siguiente:

Los elementos de la Gramática son:

$T = \{a, b\}$

$NT = \{S_0, S_1\}$

$S_0 = \{S_0\}$

El conjunto de producciones es:

$S_0 \rightarrow S_0S_1$

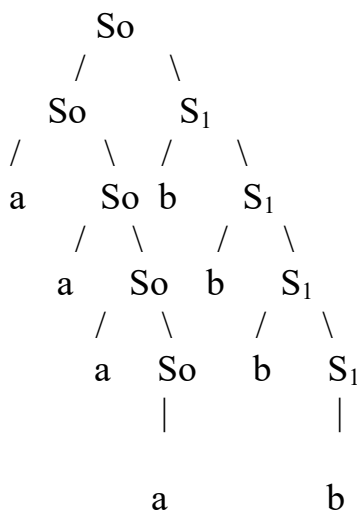
$S_0 \rightarrow aS_0$

$S_0 \rightarrow a$

$S_1 \rightarrow bS_1$

$S_1 \rightarrow b$

El árbol de derivación correspondiente es:



La cadena válida es : aaaabbbb, a,ab,abb,abbbb,aaaaa,aaaaaab

La expresión regular sería: a^+b^+

En las gramáticas para economizar líneas se suele utilizar el símbolo “|” que se lee como “ó “. Ejemplo:

$T = \{a, b\}$
 $NT = \{S_0\}$
 $S_0 = \{S_0\}$

El conjunto de producciones es:

$S_0 \rightarrow aS_0$
 $S_0 \rightarrow a$
 $S_0 \rightarrow bS_0$

Se puede simplificar como:

$S_0 \rightarrow aS_0 \mid bS_0 \mid a$

La gramática se leerá como S_0 que deriva a S_0 o bS_0 o a y la gramática seguirá produciendo lo mismo.

Cada una de las producciones de las gramáticas libres de contexto, la forman dos partes, la izquierda y la derecha, separadas por el símbolo $\rightarrow$ (producción). El *lado izquierdo* de la producción está formada por un elemento no terminal, seguido de una flecha ($\rightarrow$), y una secuencia de componentes terminales y no terminales o ambos, incluyendo la producción vacía forman el *lado derecho* de la producción.

Clasificación de las gramáticas (Chomsky)

Lenguajes recursivamente enumerables (tipo 0)

Gramáticas de tipo 0 (sin restricciones), que incluye a todas las gramáticas formales. Estas gramáticas generan todos los lenguajes capaces de ser reconocidos por una máquina de Turing. Los lenguajes son conocidos como lenguajes recursivamente enumerables. Nótese que esta categoría es diferente de la de los lenguajes recursivos, cuya decisión puede ser realizada por una máquina de Turing para que se detenga.

Ejemplos de Gramáticas de Tipo 0 son:

Los elementos Terminales son: $\{a, b, c\}$

Los elementos No Terminales son: $\{S_0, S_1, S_2, S_3\}$

El símbolo de inicio es: $\{S_0\}$

$S_0 \rightarrow aS_1$

$S_1 \rightarrow bS_2$

$$\begin{aligned} S_2 &\rightarrow bS_2 \\ S_2 &\rightarrow cS_3 \\ S_3 &\rightarrow \lambda \end{aligned}$$

Los Elementos Terminales son: $\{a, b\}$
 Los Elementos No Terminales son: $\{S_0, S_1\}$
 El símbolo de inicio es: $\{S_0\}$

$$\begin{aligned} S_0 &\rightarrow aS_0S_1 \\ S_0S_1 &\rightarrow aS_0 \mid \lambda \\ aS_1 &\rightarrow b \end{aligned}$$

Los elementos Terminales son: $\{a\}$
 Los elementos No Terminales son: $\{S_0\}$
 El símbolo de inicio es: $\{S_0\}$

$$S_0 \rightarrow aS_0 \mid a$$

Lenguajes sensibles al contexto (tipo 1)

Gramáticas de tipo 1 (gramáticas sensibles al contexto) generan los lenguajes sensibles al contexto. Estas gramáticas tienen reglas de la forma $\alpha A \beta \rightarrow \alpha \gamma \beta$ con A un no terminal y α , β y γ cadenas de terminales y no terminales. Las cadenas α y β pueden ser vacías, pero γ no puede serlo. La regla $S \rightarrow \epsilon$ está permitida si S no aparece en la parte derecha de ninguna regla.

Ejemplos de gramáticas Tipo 1 son:

Los Elementos Terminales son: $\{a, b\}$
 Los Elementos No Terminales son: $\{S_0, S_1\}$
 El símbolo de inicio es: $\{S_0\}$

$$\begin{aligned} S_0 &\rightarrow S_0S_1 \\ S_0S_1 &\rightarrow aS_0b \\ ab &\rightarrow abS_0 \mid a \end{aligned}$$

Los Elementos Terminales son: $\{a, b\}$
 Los Elementos No Terminales son: $\{S_0, S_1, S_2, S_3, S_4, S_5\}$
 El símbolo de inicio es: $\{S_0\}$

$$\begin{aligned} S_0 &\rightarrow S_4S_5 \\ S_4S_5 &\rightarrow S_0 \end{aligned}$$

$$S_0 \rightarrow bS_0 \mid bS_1$$
$$S_1 \rightarrow aS_2 \mid aS_1$$
$$S_2 \rightarrow bS_1$$
$$S_3 \rightarrow b$$

Los Elementos Terminales son: $\{a, b, c\}$

Los Elementos No Terminales son: $\{S_0, S_1, S_2\}$

El símbolo de inicio es: $\{S_0\}$

$$S_0 \rightarrow c$$
$$S_0 \rightarrow S_1cS_0 \mid S_1S_0$$
$$S_1 \rightarrow aS_2$$
$$S_2 \rightarrow bS_1 \mid b$$

Lenguajes independientes del contexto (tipo 2)

Gramáticas de tipo 2 (gramáticas libres de contexto GIC) generan los lenguajes independientes del contexto. Las reglas son de la forma $A \rightarrow \gamma$ con A un no terminal y γ una cadena de terminales y no terminales. Estos lenguajes son aquellos que pueden ser reconocidos por un autómata con pila.

Ejemplos de gramáticas Tipo 2 son:

Los Elementos Terminales son: $\{a, b\}$

Los Elementos No Terminales son: $\{S_0, S_1, S_2\}$

El símbolo de inicio es: $\{S_0\}$

$$S_0 \rightarrow bS_1$$
$$S_1 \rightarrow bS_1 \mid S_2$$
$$S_2 \rightarrow abS_2 \mid ab$$

Los elementos Terminales son: $\{a, b\}$

Los elementos No terminales son: $\{S_0, S_1, S_2\}$

El símbolo de inicio es: $\{S_0\}$

$$S_0 \rightarrow aS_0 \mid bS_0$$
$$S_0 \rightarrow aS_1$$
$$S_1 \rightarrow bS_1 \mid bS_0 \mid bS_2$$
$$S_2 \rightarrow bb$$

Los elementos Terminales son: $\{a, b\}$

Los elementos No terminales son: $\{S_0, S_1\}$

El símbolo de inicio es: $\{S_0\}$

$$S_0 \rightarrow aS_0 a$$

$$S_0 \rightarrow aS_1 a$$

$$S_1 \rightarrow bS_1 a$$

$$S_1 \rightarrow ba$$

Los elementos Terminales son: $\{a, b\}$

Los elementos No terminales son: $\{S_0, S_1\}$

El símbolo de inicio es: $\{S_0\}$

$$S_0 \rightarrow aS_0 b$$

$$S_1 \rightarrow ab$$

Lenguajes regulares (tipo3)

Gramáticas de tipo 3 (gramáticas regulares) generan los lenguajes regulares. Estas gramáticas se restringen a aquellas reglas que tienen en la parte izquierda un no terminal, y en la parte derecha un solo terminal, posiblemente seguido de un no terminal. La regla $S \rightarrow \epsilon$ también está permitida si S no aparece en la parte derecha de ninguna regla. Estos lenguajes son aquellos que pueden ser aceptados por un autómata finito. También esta familia de lenguajes puede ser obtenida por medio de expresiones regulares.

Ejemplos de gramáticas Tipo 3 son:

Los Elementos Terminales son: $\{a, b\}$

Los Elementos No Terminales son: $\{S_0, S_1\}$

El símbolo de inicio es: $\{S_0\}$

$$S_0 \rightarrow aS_0$$

$$S_1 \rightarrow aS_1$$

$$S_1 \rightarrow bS_1$$

$$S_1 \rightarrow a$$

Los Elementos Terminales son: $\{a, b\}$

Los Elementos No Terminales son: $\{S_0, S_1, S_2, S_3\}$

El símbolo de inicio es: $\{S_0\}$

$$S_0 \rightarrow bS_0 \mid bS_1 \mid bS_2 \mid cS_1 \mid aS_3 \mid a$$

$$S_1 \rightarrow cS_1 \mid cS_2 \mid aS_3 \mid a$$

$$S_2 \rightarrow aS_3$$

$$S_3 \rightarrow aS_3 \mid a$$

Los Elementos Terminales son: $\{ x, y, z \}$

Los Elementos No Terminales son: $\{ V_0, V_1, V_2 \}$

El símbolo de inicio es: $\{ V_0 \}$

$$V_0 \rightarrow x V_0$$
$$V_0 \rightarrow x V_1$$
$$V_1 \rightarrow y V_2 / y$$
$$V_2 \rightarrow z V_2 / z$$

Los Elementos Terminales son: $\{ a, b \}$

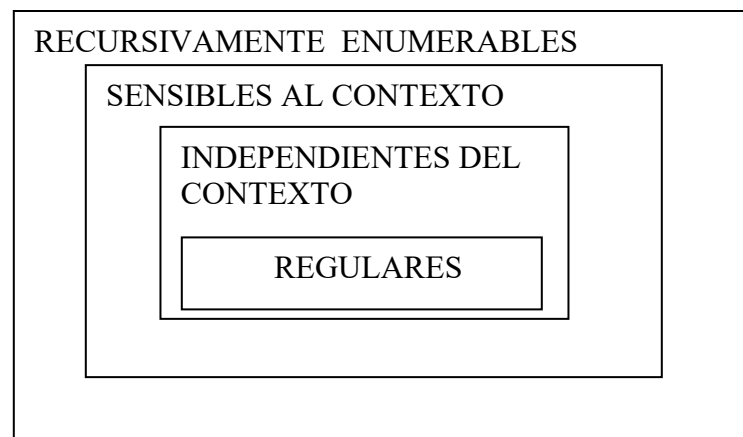
Los Elementos No Terminales son: $\{ S_0 \}$

El símbolo de inicio es: $\{ S_0 \}$

$$S_0 \rightarrow a S_0 \mid b S_0 \mid a \mid b$$

Todos los lenguajes regulares, independientes de contexto, dependientes de contexto y recursivos son recursivamente enumerables.

La siguiente figura muestra la jerarquía de las gramáticas según Chomsky



Una característica que presentan los tipos de gramáticas es que la(s) mayor(es) soporta a la(s) menor(es) pero la(s) menor(es) no soporta a la(s) mayor(es.) Es decir una gramática de Tipo 3 es a su vez de Tipo 2,1,0. Una de Tipo 2 es de Tipo 1,0. Pero no puede ser de Tipo 3. Una de Tipo 1 es de Tipo 0 pero no es de Tipo 2,3. Una de Tipo 0 no es de Tipo 1,2,3. Esto se puede observar más claramente en la figura anterior.

Ejemplos:

Gramática sólo Tipo 0

$T = a, b$

$NT = S_0, S_1, S_2$

$S_0 = S_0$

$S_0 \rightarrow b S_0 \mid a S_1$

$S_1 \rightarrow b S_1 \mid a S_2$

$S_2 \rightarrow b S_2 \mid \lambda$

No es Tipo 1,2,3

Gramática sólo Tipo 1,0

$T = a, b$

$NT = S_0, S_1, S_2$

$S_0 = S_0$

$S_0 \rightarrow a S_0 b$

$S_0 b \rightarrow bw$

$abw \rightarrow c$

No es Tipo 2,3

Gramática sólo Tipo 2,1,0

$T = a, b$

$NT = S_0, S_1, S_2$

$S_0 = S_0$

$S_0 \rightarrow a S_0$

$S_1 \rightarrow b$

$S_1 \rightarrow S_0$

$S_1 \rightarrow a$

No es Tipo 3.

Gramática sólo Tipo 3,2,1,0

$T = a, b$

$NT = S_0, S_1, S_2$

$S_0 = S_0$

$S_0 \rightarrow a S_0$

$S_0 \rightarrow b S_1$

$S_1 \rightarrow a S_1$

$S_1 \rightarrow b S_1$

$S_2 \rightarrow b S_2$

$S_1 \rightarrow b$

$S_2 \rightarrow a$

$S_2 \rightarrow b$

Pero en general a las gramáticas se les denomina por el nivel más alto que tengan es decir a una gramática de Tipo 3 no se le llama gramática de tipo 3,2,1,0 simplemente se dice del Tipo 3.

Como ya se mencionó una gramática menor no soporta a una mayor, pero sin embargo se pueden realizar modificaciones a la gramática de tal forma que presente las características de otro nivel ya sea superior o inferior siempre y cuando la gramática siga aceptando las mismas cadenas que la gramática original.

De manera general se recomienda que para pasar de un tipo a otro, ya sea de mayor a menor, o menor a mayor, generar árboles sintácticos, para obtener un conjunto representativo de cadenas válidas, a partir de estas, si es necesario

generar su expresión regular, para así proponer el nuevo conjunto de producciones de la gramática.

Ejemplos

Pasar Gramática de Tipo 2 a Tipo 3

Los Elementos Terminales son: a, b, c

Los Elementos No Terminales son: V_0 , V_1 , V_2 .

El símbolo de inicio es: V_0

$V_0 \rightarrow aaV_0$	}	Gramática Tipo 2
$V_0 \rightarrow bV_1$		
$V_1 \rightarrow cV_2b$		
$V_1 \rightarrow cb$		
$V_2 \rightarrow bbV_2$		
$V_2 \rightarrow bb$		

La expresión regular de éste conjunto de producciones es: $(aa)^*bc(bb)^*b$, por lo que a partir de esta expresión se propone el siguiente conjunto de producciones.

La gramática de Tipo 3 es:

Los Elementos Terminales son: a, b

Los Elementos No Terminales son: V_0 , V_1 , V_2 , V_3 , V_4

El símbolo de inicio es: V_0

$$\begin{aligned} V_0 &\rightarrow aV_1 \mid bV_2 \\ V_1 &\rightarrow aV_0 \\ V_2 &\rightarrow cV_3 \\ V_3 &\rightarrow bV_4 \mid b \\ V_4 &\rightarrow bV_3 \end{aligned}$$

Otra tarea es partir de una expresión regular para construir su gramática y viceversa.

Por ejemplo; partiendo de la expresión regular genere una gramática de tipo 1 sin que sea de Tipo 2 de la siguiente expresión:

$(a / b)^+(ab)^*$

$NT = S_0, S_1, S_2$

$T = a, b$

$S_0 = S_0$

$S_0 \rightarrow aS_0 / bS_1$

$aS_0 \rightarrow S_1$

$bS_1 \rightarrow S_2$

$S_2 \rightarrow aS_2 / bS_2$

$S_2 \rightarrow aS_2 / bS_2 / a / b$

En el siguiente ejemplo se genera una expresión regular partiendo de una gramática.

$NT = S_0$

$T = a$

$S_0 = S_0$

$S_0 \rightarrow a / aS_0$

Expresión regular a^+

3.3 FORMAS NORMALES DE CHOMNSKY

La Forma normal de Chomsky, es un tipo estándar para representar gramáticas. Si a una gramática le aplicamos determinados algoritmos que den lugar a que todas sus REGLAS sean de la forma:

- $S \rightarrow AB$
- $S \rightarrow a$

Donde **S** es el símbolo inicial, **A** y **B** son símbolos no terminales y **a** es un símbolo terminal. La forma normal de Chomsky, no genera la cadena vacía (ϵ). Entonces tenemos dicha gramática representada en forma normal de Chomsky. Esto es útil para poder utilizar la gramática en diversos algoritmos.

Los Elementos Terminales son: $\{ x, y, z \}$

Los Elementos No Terminales son: $\{ V_0, V_1, V_2 \}$

El símbolo de inicio es: $\{ V_0 \}$

$V_0 \rightarrow xV_0$

$o \rightarrow xV_1$

$V_1 \rightarrow yV_2 / y$

$V_2 \rightarrow zV_2 / z$

Forma normal de Chomsky

Los Elementos Terminales son: $\{x, y, z\}$

Los Elementos No Terminales son: $\{V_0, V_1, V_2, X, Y, Z\}$

El símbolo de inicio es: $\{V_0\}$

$$V_0 \rightarrow XV_0$$

$$V_0 \rightarrow XV_1$$

$$V_1 \rightarrow YV_2 / Y$$

$$V_2 \rightarrow ZV_2 / Z$$

$$X \rightarrow x$$

$$Y \rightarrow y$$

$$Z \rightarrow z$$

Los Elementos Terminales son: $\{a, b\}$

Los Elementos No Terminales son: $\{S_0, S_1\}$

El símbolo de inicio es: $\{S_0\}$

$$S_0 \rightarrow aS_1$$

$$S_1 \rightarrow bS_1 \mid bS_2 \mid \lambda$$

$$S_2 \rightarrow c$$

Eliminando la producción vacía.

Los Elementos Terminales son: $\{a, b, c\}$

Los Elementos No Terminales son: $\{S_0, S_1\}$

El símbolo de inicio es: $\{S_0\}$

$$S_0 \rightarrow aS_1 \mid a$$

$$S_1 \rightarrow bS_1 \mid bc$$

Forma normal de Chomsky

Los Elementos Terminales son: $\{a, b, c\}$

Los Elementos No Terminales son: $\{S_0, S_1, A, B, C\}$

El símbolo de inicio es: $\{S_0\}$

$$S_0 \rightarrow AS_1 \mid a$$

$$S_1 \rightarrow BS_1 \mid BC$$

$$A \rightarrow a$$

$$B \rightarrow b$$

$$C \rightarrow c$$

3.4 FORMAS NORMALES DE GREIBACH

Se dice que una gramática independiente del contexto (GIC) está en Forma normal de Greibach (FNG), si todas y cada una de sus reglas de producción tienen un consecuente que empieza por un carácter del alfabeto, también llamado símbolo terminal. Formalmente, cualquiera de las reglas tendrá la estructura:

- $A \rightarrow aw$

Donde "A" es el antecedente de la regla, que en el caso de las GIC debe ser necesariamente un solo símbolo auxiliar. Por su parte, "a" es el mencionado comienzo del consecuente y, por tanto, un símbolo terminal. Finalmente, "w" representa una concatenación genérica de elementos gramaticales, esto es, una sucesión de auxiliares y terminales entremezclados, inclusive, pudiera ser la palabra vacía; en este caso particular, se tendría una regla llamada "terminal":

- $A \rightarrow a$

Existe un teorema que prueba que cualquier GIC, cuyo lenguaje no contiene a la palabra vacía, si no lo está ya, se puede transformar en otra equivalente que sí esté en FNG. Para su demostración, normalmente, se procede por construcción, es decir, se plantea directamente un algoritmo capaz de obtener la FNG a partir de una GIC dada.

Los Elementos Terminales son: a, b, c, d, e

Los Elementos No Terminales son: V_0, V_1, V_2 .

El símbolo de inicio es: V_0

$V_0 \rightarrow aaV_0$

$V_0 \rightarrow bV_1$

$V_1 \rightarrow cV_2e$

$V_1 \rightarrow c$

$V_2 \rightarrow d$

Forma Normal de Greibach

Los Elementos Terminales son: a, b, c, d, e

Los Elementos No Terminales son: $V_0, V_1, V_2, V_3, V_4, E$

El símbolo de inicio es: V_0

$V_0 \rightarrow aV_3$

$V_0 \rightarrow bV_1$

$$\begin{aligned} V_1 &\rightarrow cV_4 \\ V_1 &\rightarrow c \\ V_2 &\rightarrow d \\ V_3 &\rightarrow aV_0 \\ V_4 &\rightarrow dE \\ E &\rightarrow e \end{aligned}$$

Los Elementos Terminales son: a, b, c, d, e
 Los Elementos No Terminales son: V_0, V_1, V_2 .
 El símbolo de inicio es: V_0

$$\begin{aligned} V_0 &\rightarrow aaV_0 \\ V_0 &\rightarrow bV_1 \\ V_1 &\rightarrow cV_2e \\ V_1 &\rightarrow \lambda \\ V_2 &\rightarrow d|\lambda \end{aligned}$$

Eliminando producción vacía
 Los Elementos Terminales son: a, b, c, d, e
 Los Elementos No Terminales son: V_0, V_1
 El símbolo de inicio es: V_0

$$\begin{aligned} V_0 &\rightarrow aaV_0 \\ V_0 &\rightarrow bV_1|b \\ V_1 &\rightarrow ce|cde \end{aligned}$$

Forma Normal de Greibach
 Los Elementos Terminales son: a, b, c, d, e
 Los Elementos No Terminales son: V_0, V_1, V_2, V_3, E
 El símbolo de inicio es: V_0

$$\begin{aligned} V_0 &\rightarrow aV_3 \\ V_0 &\rightarrow bV_1|b \\ V_1 &\rightarrow cE \\ V_2 &\rightarrow dE \\ V_3 &\rightarrow aV_0 \\ E &\rightarrow e \end{aligned}$$

3.5 ELIMINACIÓN DE FACTORES COMUNES IZQUIERDOS

Existen gramáticas que tiene producciones de la forma $A \rightarrow \alpha \beta_1 \mid \alpha \beta_2$ como por ejemplo:

$S \rightarrow i E t S e S \mid i E t S$

Donde α es el término común en las producciones de A.

Sin embargo para poder llevar a cabo el análisis sintáctico de las mismas producciones, se debe eliminar los términos comunes izquierdos llevando a cabo el proceso de factorización siguiente:

Las producciones $A \rightarrow \alpha \beta_1 \mid \alpha \beta_2$ se transforman en las siguientes

$A \rightarrow \alpha A'$

$A' \rightarrow \beta_1 \mid \beta_2$

Los Elementos Terminales son: $\{x, y, z\}$

Los Elementos No Terminales son: $\{V_0, V_1, V_2\}$

El símbolo de inicio es: $\{V_0\}$

$V_0 \rightarrow xV_0 \mid xV_1$

$V_1 \rightarrow yV_2 \mid y$

$V_2 \rightarrow zV_2 \mid z$

Eliminando factores comunes

Los Elementos Terminales son: $\{x, y, z\}$

Los Elementos No Terminales son: $\{V_0, V_1, V_2\}$

El símbolo de inicio es: $\{V_0\}$

$V_0 \rightarrow xV_0'$

$V_0' \rightarrow V_0 \mid V_1$

$V_1 \rightarrow yV_1'$

$V_1' \rightarrow V_2 \mid \lambda$

$V_2 \rightarrow zV_2'$

$V_2' \rightarrow V_2 \mid \lambda$

3.6 ELIMINACIÓN DE RECURSIVIDAD IZQUIERDA

Existen gramáticas que tienen producciones de la forma $A \rightarrow A \alpha \mid \beta$ como por ejemplo:

Donde S es un término en el que se produce la recursión.

Sin embargo para poder llevar a cabo el análisis sintáctico de las mismas producciones, se debe eliminar la recursión, llevando a cabo el proceso siguiente:

Las producciones $A \rightarrow A \alpha \mid \beta$ se transforman en las siguientes:

$$A \rightarrow \beta A'$$

$$A' \rightarrow \alpha A'$$

Los Elementos Terminales son: {a,b}

Los Elementos No Terminales son: { S }

El símbolo de inicio es: {S}

$$S \rightarrow S a | b$$

Eliminando recursividad izquierda

Los Elementos Terminales son: { a,b}

Los Elementos No Terminales son: { S }

El símbolo de inicio es: {S}

$$S \rightarrow b S'$$

$$S' \rightarrow a S' | \lambda$$

Los Elementos Terminales son: {a,b,c,d}

Los Elementos No Terminales son: { S }

El símbolo de inicio es: {S}

$$S \rightarrow S a | S b | S c | d$$

Eliminando recursividad izquierda

Los Elementos Terminales son: { a,b,c,d}

Los Elementos No Terminales son: { S }

El símbolo de inicio es: {S}

$$S \rightarrow d S'$$

$$S' \rightarrow a S' | b S' | c S' | \lambda$$

3.7 ELIMINACION DE AMBIGÜEDAD

Una gramática se dice que es ambigua si hay dos o más árboles de derivación distintos para una misma cadena. Una gramática en la cual, para toda cadena w, todas las derivaciones de w tienen el mismo árbol de derivación, es no ambigua.

Problemas con la Ambigüedad

Cuando el significado de un lenguaje depende de su estructura, la ambigüedad puede significar un problema.

Ejemplo: En lenguaje natural, la frase: “*Juan vio a un hombre con un telescopio*”

Es una frase ambigua ya que podría interpretarse de dos formas.

Ejemplo: En un lenguaje de programación, la siguiente gramática podría ser para asignación de expresiones:

$$A \rightarrow I:=E$$
$$I \rightarrow a \mid b \mid c$$
$$E \rightarrow E+E \mid E * E \mid (E) \mid I$$

Esta gramática es ambigua. Para la cadena $a:=b+c*a$ es posible construir dos árboles de derivación diferentes, los cuales no dejan claro si su evaluación es $b+(c*a)$ o $(b+c)*a$.

Tipos de ambigüedad

Dentro del estudio de gramáticas existen dos tipos fundamentales de ambigüedad, los cuales son:

Ambigüedad Inherente: Las gramáticas que presentan este tipo de ambigüedad no pueden utilizarse para lenguajes de programación, ya que por más transformaciones que se realicen sobre ellas, NUNCA se podrá eliminar completamente la ambigüedad que presentan.

Ambigüedad Transitoria: Este tipo de ambigüedad puede llegar a ser eliminada realizando una serie de transformaciones sobre la gramática original. Una vez que se logra lo anterior, la gramática queda lista para ser reconocida por la mayor parte de los analizadores sintácticos. (Se le considera "ambigüedad" porque existen métodos para realizar análisis sintáctico que no aceptan gramáticas con estas características).

La ambigüedad de las gramáticas se elimina sustituyendo producciones y eliminando símbolos inútiles, quitando factorización, recursión y a veces dando prioridad en algunos símbolos.

3.8 AUTÓMATAS DE PUSH-DOWN

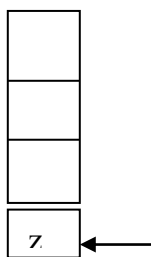
Un *autómata de pila* es esencialmente un autómata finito que posee control sobre una cinta de entrada y una lista en forma de pila, es decir, sólo se pueden acceder los símbolos que se encuentran en la cima o al frente de la lista.

Tienen la filosofía del primero que entra es el último que sale, o bien el último en llegar es el primero en salir.

Por su filosofía no resuelve todos los casos de expresiones pero sin embargo si es capaz de resolver todas las expresiones regulares.

Los autómatas de pila generalmente se utilizan en los lenguajes de programación para evaluar expresiones postfijas y/o sufijas.

Gráficamente se representa así:



pila

El autómata de pila está compuesto por una 7-tupla:

$$M = (\Sigma, T, S_0, F, S, z, \Delta)$$

Σ Alfabeto de la entrada.

T alfabeto de la pila.

S_0 estado de inicio.

F conjunto de transiciones.

S estado de aceptación.

z Símbolo de inicio de la pila

Δ Función de transición

Una función de transición está dada por: $(q_0, a, z) = (q_1, az)$

Donde “ q_0 ” es el estado origen de la pila.

“ a ” Es la entrada o elemento de la cadena.

“ z ” es el fondo de la pila.

“ q_1 ” es el estado destino de la pila.

“ az ” es como queda la pila.

Nota:

Recuerde que “ q_1 ” y “ q_0 ” pueden variar.

Una de las tareas importantes es construir el autómata de pila, para ello, el diseñador de lenguajes puede partir de varios puntos, como son un conjunto de

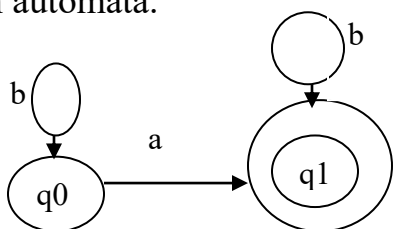
enunciados que den la característica del comportamiento del lenguaje, una expresión regular o un autómata finito.

A continuación se muestra como se puede construir un autómata de pila partiendo de cualquier caso, al mismo tiempo se muestra los movimientos de la pila.

El problema se define por el siguiente enunciado, genere un autómata de pila que acepte todas las cadenas que contengan únicamente una letra a.

Construya una expresión regular. b^*ab^*

Construya un autómata.



Con base al autómata construido, verifique que efectivamente sea un autómata finito determinista reducido (AFDR) en caso de que no lo sea aplique alguna técnica para pasarlo a un AFDR.

Pasos para realizar un autómata de pila.

1) Proponga una serie de cadenas.

Cadenas válidas. a, bbabb, abb, bba

La cadena mínima. a

Una cadena máxima. bbbabbb

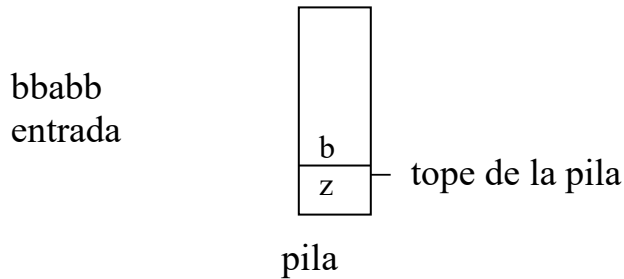
2) Comience la construcción de la pila con la cadena más larga, que es la cadena que va a ser aceptada por la pila:

bbabbb

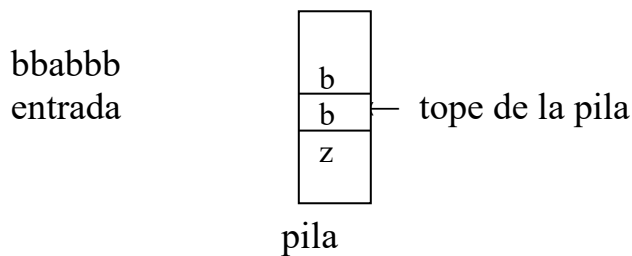
El conjunto de transiciones se construye como sigue:

Estando en q_0 con una “b” en la entrada (cadena) y con una z en el fondo de la pila se inserta a la pila “bz” y no cambia de estado (q_0). Esto en el caso que la cadena comience con “b”

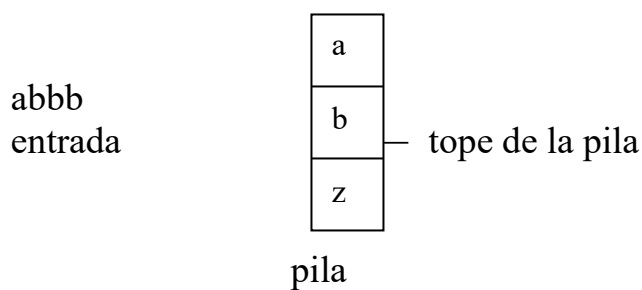
Gráficamente. $(q_0, b, z) = (q_0, bz)$



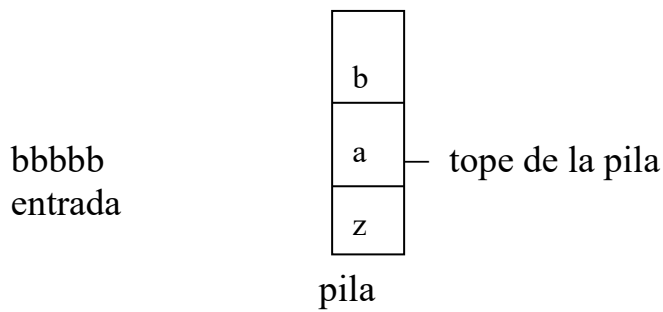
Estando en q_0 con una “b” en la entrada (cadena) y con una “b” en el fondo de la pila se inserta a la pila “bb”. Esto en el caso que la cadena continúe con “b” realiza un ciclo de b’s al principio.
Gráficamente. $(q_0, b, b) = (q_0, bb)$



Estando en q_0 con una “a” en la entrada (cadena) y con una “b” en el fondo de la pila se inserta a la pila “ab”. Esto en el caso que la cadena comience con una o más b’s. Y asegura que al menos venga una letra a
Gráficamente. $(q_0, a, b) = (q_1, ab)$

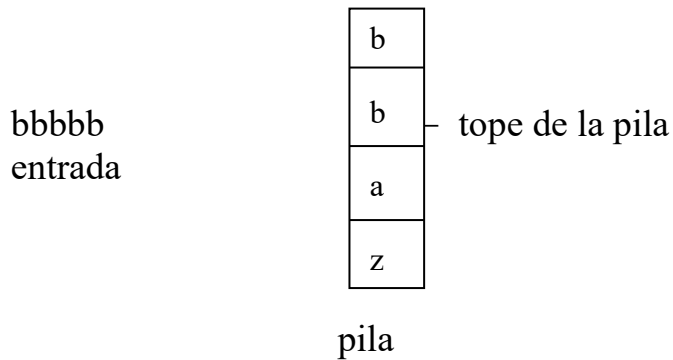


Estando en q_1 con una “b” en la entrada (cadena) y con una “a” en el fondo de la pila se inserta a la pila “ba”. Esto en el caso que la cadena continúe con “b”
Gráficamente. $(q_1, b, a) = (q_1, ba)$



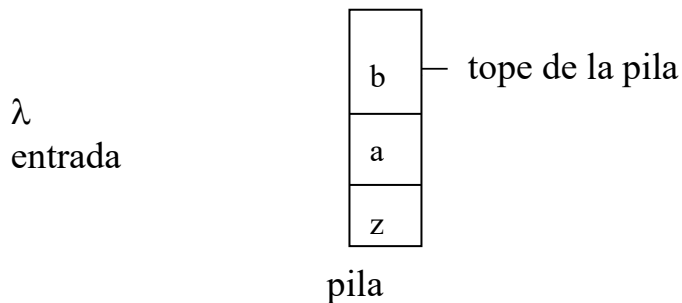
Estando en q_1 con una “b” en la entrada (cadena) y con una “b” en el fondo de la pila se inserta a la pila “bb”. Esto en el caso que la cadena continúe con “b” realiza un ciclo de b’s al final.

Gráficamente. $(q_1, b, b) = (q_1, bb)$



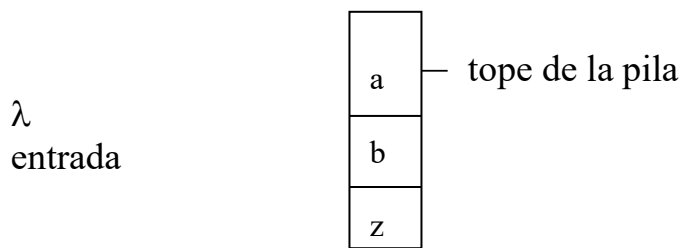
Estando en q_1 con una “λ” en la entrada (cadena) y con una “b” en el fondo de la pila se inserta a la pila “b”. Esto en el caso que la cadena termine con “b”. Y q_2 es un estado de aceptación.

Gráficamente. $(q_1, \lambda, b) = (q_2, b)$



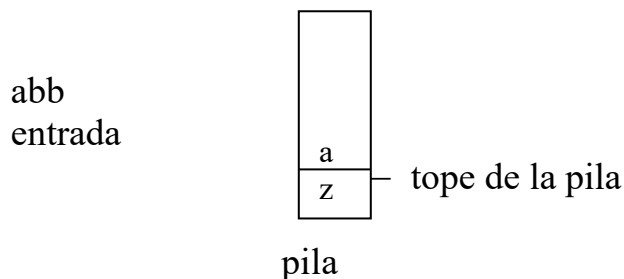
Estando en q_1 con una “λ” en la entrada (cadena) y con una “a” en el fondo de la pila se inserta a la pila “a”. Esto en el caso que la cadena terminé con “a” y q_2 es un estado de aceptación.

Gráficamente. $(q_1, \lambda, a) = (q_2, a)$



3) Continúe con las cadenas cortas donde las cadenas empiecen con “a”,
Estando en q_0 con una “a” en la entrada (cadena) y con una z en el fondo de la pila se, inserta a la pila “az” y cambia de estado (q_1). Esto en el caso que la cadena comience con “a”.

Gráficamente. $(q_0, a, z) = (q_1, az)$



El conjunto de producciones para la expresión regular b^*ab^* es:

$(q_0, a, z) = (q_1, az)$
 $(q_0, b, z) = (q_0, bz)$
 $(q_0, a, b) = (q_1, ab)$
 $(q_0, b, b) = (q_0, bb)$
 $(q_1, b, a) = (q_1, ba)$
 $(q_1, b, b) = (q_1, bb)$
 $(q_1, \lambda, b) = (q_2, b)$
 $(q_1, \lambda, a) = (q_2, a)$

Verifique que la pila realiza las mismas cadenas propuestas en el Paso 1

Proponga cadenas no válidas y verifique que la pila no las realiza, en caso de que alguna cadena no válida sea aceptada verifique que realmente no se puede hacer y verifique su expresión regular o autómata o pila y donde detecte el error repita los pasos que le falten o en su defecto replantee la pila.

NOTA:

- Generalmente se utiliza como fondo de pila la letra z, pero en casos que se utilice como parte del alfabeto de entrada sustitúyala, por algún símbolo que no forme parte del alfabeto de entrada.
- En las expresiones regulares generalmente las pilas se llenan y casi no se vacían.
- Por lo general solo existe un solo estado de aceptación, y para que este se identifique más fácilmente, se marcara con dos líneas de debajo de este símbolo.
- Cuando se parte de un autómata generalmente el estado de aceptación es $s+1$ donde s es el número total de estados.
- Para indicar que una cadena ha llegado a su fin se dice que enfrente se encuentra un vacío " λ " que no se coloca en la cadena.

Si la expresión obtenida fue no regular, es necesario recurrir a una pila.
Se recomienda que de las cadenas propuestas se vea el comportamiento.

Ejemplo la expresión no regular $a^n b^{2n}$ donde $n \geq 0$

Paso 1

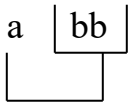
Proponga una serie de cadenas.

Cadenas válidas. Abb , $aabbbb$, λ , $aaabbbbb$

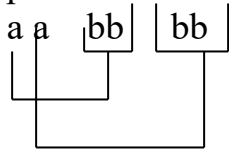
La cadena mínima. λ

Una cadena máxima. $aaabbbbb$

Obsérvese que por una a vienen dos b 's



por una " a " dos " b 's"

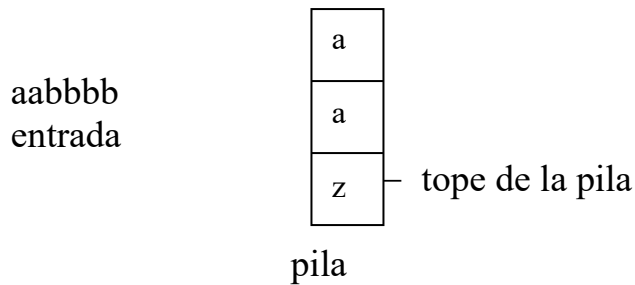


Con base a lo anterior se puede decir que cada vez que venga una " a " se necesitan meter a la pila dos símbolos, para que cuando tenga que sacar las " b 's", por una b con la entrada se saca una de la pila.

Una vez que ha verificado el movimiento de la pila constrúyala.

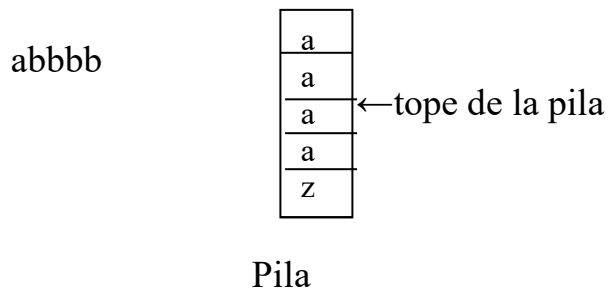
Estando en q_0 con una “a” en la entrada (cadena) y con una “z” en el fondo de la pila se inserta a la pila “aaz”. Esto en el caso que se tenga elementos.

Gráficamente. $(q_0, a, Z) = (q_0, aaZ)$



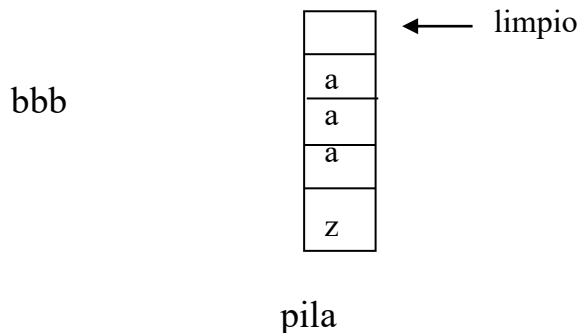
Estando en q_0 con una “a” en la entrada (cadena) y con una “a” en el fondo de la pila se inserta a la pila “aaa”. Esto en el caso que se tenga más de una a para realizar un ciclo.

Gráficamente. $(q_0, a, a) = (q_0, aaa)$



Estando en q_0 con una “b” en la entrada (cadena) y con una “a” en el fondo de la pila se limpia la pila. Y cambia de estado.

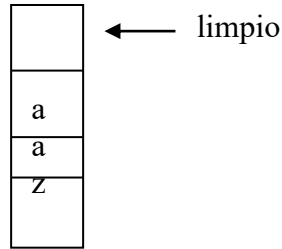
Gráficamente. $(q_0, b, a) = (q_1, \lambda)$



Estando en q_1 con una “b” en la entrada (cadena) y con una “a” en el fondo de la pila se limpia la pila. Esto en el caso que se tenga más de una “b” para realizar un ciclo.

Gráficamente. $(q_1, b, a) = (q_1, \lambda)$

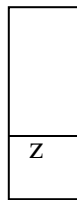
bb



pila

Estando en q_1 con un “ λ ” en la entrada (cadena) y con una “ z ” en el fondo de la pila se mete a la pila “ z ”. Esto en el caso que se tenga más de una “ a ” para realizar un ciclo.

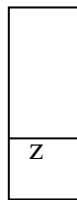
Gráficamente. $(q_1, \lambda, z) = (q_2, z)$



pila

Se realiza la parte donde $n = 0$ es decir se acepta la cadena vacía.

Gráficamente. $(q_0, \lambda, z) = (q_2, z)$



pila

Por lo tanto el conjunto de producciones para la pila es el siguiente:

$(q_0, a, Z) = (q_0, aaZ)$

$(q_0, a, a) = (q_0, aaa)$

$(q_0, a, a) = (q_0, aaa)$

$(q_0, b, a) = (q_1, \lambda)$

$(q_1, b, a) = (q_1, \lambda)$

$(q_1, \lambda, z) = (q_2, z)$

$$(q_0, \lambda, z) = (q_2, z)$$

NOTA:

- Generalmente en las expresiones no regulares se inserten y se sacan elementos de la pila hasta que queda z en la pila.
- La z que es el fondo de la pila debe de permanecer es decir raramente se elimina de la pila.
- $(q_0, b, a) = (q_1, \lambda)$ significa que limpia el elemento.
- $(q_0, a, a) = (q_0, aaa)$ el programador se puede valer de algún artificio de programación para verificar que se realice correctamente la pila.
- $(q_0, a, a) = (q_0, aaa)$ se dice que en una pila se remarca el fondo y se escribe en las casillas siguientes en este caso se remarca la letra “a” y se meten dos letras a’s más.
- $(q_0, \lambda, z) = (q_2, z)$ en algunos casos solo se remarca el fondo.

Para todas las expresiones regulares, se puede construir un autómata finito, por lo que es recomendable partir de un autómata finito determinista reducido, en caso contrario genera pilas no deterministas.

Lenguajes aceptados por un autómata de pila

El autómata de pila que es capaz de reconocer todo lenguaje independiente del contexto.

El problema que se plantea con los autómatas finitos es que sólo tienen capacidad para “memoria” finita. Lenguajes independientes del contexto tan sencillos como $\{a^n b^n \mid n \geq 0\}$ necesitan guardar gran cantidad de información; se debe verificar no solamente que todas las a ’s preceden a las b ’s, sino que además se tienen que contar las a ’s. Puesto que el número de a ’s es arbitrario, necesitamos establecer una limitación sobre el número de a ’s que se pueden contar.

El lenguaje independiente del contexto $\{wcw^1 \mid w \in \{a,b,c\}^*\}$ necesita algo más que una capacidad sin límites para contar los símbolos. Además se deben guardar los símbolos de la cadena w para compararlos con los símbolos de la cadena w^1 .

En un lenguaje de programación las expresiones aritméticas, que pasan a expresiones prefijas, infijas, posfijas para verificar que son correctas se realiza a través de autómatas de pila.

Autómata de pila determinista (APD)

Se dice que un autómata de pila es *determinista*, si existe que para cada producción (q_0, α, β) , existe una y solamente una producción “ α ” en la entrada, con “ β ” en el fondo de la pila.

Se dice que un autómata de pila es *no determinista (APND)*, si existe que para cada producción (q_0, α, β) , existen dos o más producciones “ α ” en la entrada con “ β ” en el fondo de la pila. Necesariamente deben tener el mismo estado origen q_0 , donde q_0 es un elemento cualquiera del conjunto de estados para el autómata de pila. En otras palabras un autómata de pila es no determinista si existen en él, dos o más producciones que con el mismo estado origen “ q_0 ”, con la misma entrada en “ α ” en la cadena y el mismo fondo “ β ” en la pila, se llega a dos estados “ q_1 ” diferentes, o en su defecto al mismo estado pero el fondo de la pila realiza acciones distintas.

Ejemplos de producciones no deterministas para el autómata de pila:

a)

$(q_0, a, z) = (q_1, az)$

$(q_0, a, z) = (q_1, bz)$

b)

$(q_0, a, z) = (q_1, az)$

$(q_0, a, z) = (q_2, az)$

c)

$(q_0, a, z) = (q_1, az)$

$(q_0, a, z) = (q_2, bz)$

d)

$(q_0, a, z) = (q_0, az)$

$(q_0, a, z) = (q_0, bz)$

Equivalencia entre autómata de pila no determinista y determinista

Los autómatas de pila utilizan la misma filosofía de equivalencia que los AF, por lo que si se llegase a dar un APND, se tendrá que realizar un análisis para verificar que la pila cuenta con un AF, si cuenta con él, se recomienda que se trabaje con el AF para pasarlo de no determinista a determinista, posteriormente se construya el AP, que seguramente será determinista.

Cuando no se puede generar un AF, es necesario que observe el comportamiento de la pila, y replantéela.

Los AP se utilizan al igual que los AF, para verificar que las cadenas son válidas, o expresan un conjunto de sentencias, instrucciones, o funciones de un lenguaje específico, sin duda otra parte interesante de los autómatas de pila es verificar si las cadenas son o no válidas.

Una forma fácil de verificar si son o no son válidas las cadenas es la siguiente que se explica con un ejemplo que parte una vez construido el APD.

$$Q = \{q_0, q_1, q_2, q_3, q_4\}$$

$$F = \{q_4\}$$

$$S_0 = \{q_0\}$$

$$(q_0, a, z) = (q_1, az)$$

$$(q_1, a, a) = (q_1, aa)$$

$$(q_1, b, a) = (q_2, ba)$$

$$(q_2, b, b) = (q_2, bb)$$

$$(q_2, c, b) = (q_3, cb)$$

$$(q_3, c, c) = (q_3, cc)$$

$$(q_3, \lambda, c) = (q_4, c)$$

Estado origen	Cadena	Pila
q0	aaaabbc	Z
q1	aaabbc	az
q1	aabbc	aaaz
q1	abbc	aaaaz
q1	bbc	baaaaaz
q2	bc	bbaaaaaz
q2	c	cbbaaaaaz
q3	λ	Cadena válida.
q4	Estado de aceptación.	

En el estado origen se marca en el renglón siguiente donde la pila lo llama, la cadena se lee de izquierda a derecha, a la pila se agrega el elemento por la izquierda. Conforme al ejemplo se dice que estando en “q0” con “a” en la entrada y “z” en la pila se va al estado “q1” y se inserta “az” en la pila.

Los autómatas de pila solo resuelven algunas expresiones regulares, pero como WW , $x^n y^n z^n$, no se pueden controlar con AP y es por eso que nacen las máquinas de Turing, que serán objeto de estudio más adelante.

3.8 LENGUAJES NO REGULARES

Las expresiones no regulares no se pueden representar por medio de autómatas finitos por lo tanto no generan lenguajes regulares, el lenguaje que generan las expresiones no regulares se le denomina *lenguaje no regular*. Este tipo de lenguajes trabajan con autómatas de pila.

No todos los lenguajes son regulares, para demostrar que ciertos lenguajes no son regulares se puede usar el “lema de bombeo”

Lema de bombeo para los lenguajes regulares.

El lema del bombeo es útil para demostrar que un lenguaje dado no es regular. Debemos recalcar que no es adecuado usar el lema del bombeo para descubrir la regularidad de un lenguaje puesto que las condiciones del lema de bombeo no son necesarias para que un lenguaje sea regular no son suficientes.

Si L es un lenguaje regular entonces existe un número $n \in \mathbb{N}$, llamado constante de bombeo para L , tal que para cualquier cadena w de L con $|w| \geq n$ existen cadenas u, v, x tales

que:

1. $w = uvx$
2. $v \neq \epsilon$
3. $|uv| \leq n$
4. $\forall i \geq 0, uv^i x \in L$.

Para probar que L no es regular se procede por contradicción usando el lema de bombeo como sigue:

- Si L fuera regular existiría una constante de bombeo n .
- Cualquier palabra w se descompone como $w = uvx$, $v \neq \epsilon$, $|uv| \leq n$.
- Se llega a una contradicción como sigue:

Por el lema del bombeo $uv^ix \in L$ para toda $i \geq 0$.
 Por la definición de L , existe alguna i tal que $uv^ix \in L$.

Debemos observar que encontrar la i adecuada depende del problema particular y no hay un método general, pero generalmente basta con valores pequeños de i .

Ejemplo

$L = \{a^i b^i \mid i \in \mathbb{N}\}$ no es regular

- Supóngase que L es regular y sea n una constante de bombeo.
- Considérese la palabra $w = a^n b^n$ y su descomposición $w = uvx$ con $v \neq \varepsilon$ y $|uv| \leq n$.
 Entonces u, v constan de puras a 's, digamos $u = a^k$, $v = a^l$, $k \geq 0, l \geq 1$.
- De manera que $w = a^{n-k-l} b^n$.
- Si hacemos $i = 2$ tenemos que $uv^2x \in L$ por el lema de bombeo, pero $uv^2x = a^k a^l a^l a^{n-k-l} b^n = a^{n+l} b^n \in L$

Podemos describir el lema de bombeo como un juego para dos personas:

- Juego para un lenguaje L .
- Jugador 1 selecciona n .
- Jugador 2 selecciona $w \in L$ con $|w| \geq n$
- Jugador 1 selecciona partición $w = xyz$ con $y \neq \varepsilon$ y $|xy| \leq n$.
- Jugador 2 selecciona k .
- si xy^kz no pertenece a L , gana jugador 2, sino gana jugador 1.
- si jugador 2 puede ganar siempre, entonces L no es regular.

El lema de bombeo es el jugador 1, ¿quién es el jugador 2?

Ejemplo

$a^n b^{2n}$ es un lenguaje no regular

$n=2$

$w = aabbbb$

$|w| \geq 2$

$x = a$ $y = ab$ $z = bbb$

$k=2$

$w = aababbbb \in L$

Ejemplo

$a^n b^n$ es un lenguaje no regular

$n=2$

$w=aabb$

$|w| \geq 2$

$x=aa \quad y=b \quad z=b$

$k=2$

$w=aabbb \in L$

EJERCICIOS PROPUESTOS

Genere Gramáticas de los Tipos 0,1, 2, 3 y sus Expresiones Regulares.

1. $V_0 \rightarrow xV_0$
 $V_0 \rightarrow yV_1$ $T \quad \{x, y, z\}$
 $V_1 \rightarrow yV_1$ $NT \quad \{V_0, V_1\}$
 $V_1 \rightarrow z$ $S_0 \quad \{V_0\}$
2. $V_0 \rightarrow xV_0$
 $V_0 \rightarrow yV_0$ $T \quad \{x, y, z\}$
 $V_0 \rightarrow z$ $NT \quad \{V_0\}$
 $S_0 \quad \{V_0\}$
3. $V_0 \rightarrow a + V_1$
 $V_1 \rightarrow a + V_2$ $T \quad \{a, +\}$
 $V_2 \rightarrow a + V_2$ $NT \quad \{V_0, V_1, V_2\}$
 $V_2 \rightarrow a$ $S_0 \quad \{V_0\}$

De la siguiente expresión no regular $L = \{x^r y^s z^t : s = r + t, r \geq 1, t \geq 0\}$

Realice lo siguiente:

- 4) Genere gramática de tipo 0 que no sea de tipo 2
- 5) Gramática de tipo 3
- 6) Gramática de tipo 2 que no sea de tipo 3
- 7) Gramática de tipo 2 que no sea de tipo 1
- 8) Genere gramática que acepte cadenas que acepten m símbolos a, donde m es múltiplo de 3. Para el alfabeto $\{a, b\}$
- 9) Genere Expresión de la siguiente gramática. Para el alfabeto x,y
 $S \rightarrow SS/MN \quad N \rightarrow SP/y \quad M \rightarrow x \quad P \rightarrow y$
- 10 Genere forma normal de Greibach de la siguiente gramática
a) $S_1 \rightarrow S_2 S_3 \quad S_2 \rightarrow S_3 S_1/b \quad S_3 \rightarrow S_1 S_2/a$
b) $S \rightarrow BAB/ABA \quad A \rightarrow AB/aA/ab \quad B \rightarrow BA/b$
- 11 Genere forma normal de Chomsky de la siguiente gramática
 $S \rightarrow bA/aB \quad A \rightarrow bAA/aS/a \quad B \rightarrow aBB/bS/b \quad T\{a, b\}$
 $S \rightarrow AC \quad A \rightarrow aAb/\& \quad C \rightarrow bCc/\&$

12 Elimine recursión, factorización y ambigüedad de la siguiente gramática.
 $S \rightarrow CBA/D \quad C \rightarrow eA/F/C \quad A \rightarrow bbC \quad D \rightarrow E/SABC \quad B \rightarrow Sc/ddd \quad E \rightarrow gh$

13. Utiliza el lema de bombeo para demostrar que los siguientes lenguajes no son regulares.

- a. $L = \{ a_i b_{2i} \mid i \geq 1 \}$
- b. $L = \{ a_i b_j c_{2i} \mid i, j \geq 1 \}$
- c. $L = \{ a_i b_j c_j d_i \mid i \geq 1 \wedge j \geq 1 \}$

14. ¿Cuáles de los siguientes lenguajes son regulares? Si crees que un lenguaje no es regular, entonces utiliza el lema de bombeo para probarlo. Si crees que el lenguaje es regular, entonces define una expresión regular que denote a dicho lenguaje. El alfabeto sobre el que están definidos todos los lenguajes es $\Sigma = \{0,1\}$

- a. $L = \{0_{2i} \mid i \geq 0\}$
- b. $L = \{1_i 0_j 1_i \mid i \geq 1 \wedge j \geq 1\}$
- c. $L = \{0_i 1_j 0_{i+j} \mid i \geq 1 \wedge j \geq 1\}$
- d. El conjunto de todas las cadenas que no tienen tres ceros consecutivos.
- e. El conjunto de todas las cadenas que tiene igual número de ceros que de unos.
- f. El conjunto de todas las cadenas que se escriben igual de izquierda a derecha que de derecha a izquierda

CONSULTAS REALIZADAS

Alfred V. Aho, Monica S. Lam, Sethi Ravi y Jeffrey D. Ullman
Compiladores principios, técnicas y herramientas.
Editorial Addison-Wesley Iberoamericana.
2008.

Beck Leland
Software de sistemas. Introducción a la programación de sistemas.
Editorial Addison-Wesley Iberoamericana.
1988.

Jonh E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman
Introducción a la teoría de autómatas, Lenguajes y Computación.
Editorial PEARSON Addison Wesley.
2002.

Dean Kelley
Teoría de autómatas y lenguajes formales.
Editorial Prentice Hall
2006

Johnsonbaugh, Richard
Matemáticas discretas
Editorial Iberoamercana
2002.

Jean Paúl Trembay, Ram Manohar
Matemáticas discretas con aplicación a las ciencias de la computación
Editorial CECSA.
1996.

<http://webdia.cem.itesm.mx/ac/aesparza/compi/expreg.html>

<http://campus.gda.itesm.mx/~csainz/gram.htm>

<http://campus.gda.itesm.mx/~csainz/gramLR.htm>

<http://www.ls1.us.es/docencia/asignaturas/lfail.html>

<http://www.eluvigo.es/~cabrero/galena.dcg.html>

<http://mx.geocities.com/mvd09/SINTESIS.doc>

<http://es.wikipedia.org/wiki/>

http://mx.geocities.com/bellota_epg/TC_Unidad_EveliaPG.ppt

<http://www.scribd.com/doc/16233072/2-Lenguajes-regulares>

<http://www.guegue.com.ni/free/painrob/g1.html>

<http://www.hum.uva.ni/~teun/beliar.spa>

<http://148.247.2.21/TmTeAut.html>

<http://www.inf.udec.cl/~lenform/index.html>

<http://www.inf.udec.cl/~lenform/21.html>

4. MÁQUINAS DE TURING

Competencias que deberán ser adquiridas

- Capacidad de investigación
- Autonomía en el aprendizaje
- Creatividad
- Capacidad de Análisis y síntesis
- Organización y planificación
- Comunicación oral y escrita
- Utilización de las tecnologías de comunicación y de información (TIC'S)
- Trabajo en equipo

Objetivos del aprendizaje

Capacidad para diseñar un lenguaje a través de una Máquina de Turing.

Resultado del aprendizaje	Actividad Educativa	Evaluación	Tiempo
Comprensión y capacidad de diseñar una máquinas de Turing	• Búsqueda de conceptos por equipos en los que podrá leer, descubrir, cuestionar, preguntar indagar, sobre el diseño de una maquina de Turing. • Discusión en clase para expresarse, comunicar ideas, etc. • Creación de ideas originales obtenidas en consenso • Ejemplos de lenguajes aceptados por una máquina de Turing.	• Informes por equipo y conclusiones grupales. • Técnicas de observación(registros, listas de control) • Diseño de un lenguaje formal o natural representado por una MT.	6 horas
Comprensión y capacidad de diseñar una máquinas de Turing en forma modular	• Búsqueda de conceptos por equipos en los que podrá leer, descubrir, cuestionar, preguntar indagar, sobre el diseño de una maquina de Turing por módulos. • Discusión en clase para expresarse, comunicar ideas, etc. • Creación de ideas originales obtenidas en consenso • Ejemplos de lenguajes aceptados por una máquina de Turing por módulos.	• Informes por equipo y conclusiones grupales. • Técnicas de observación(registros, listas de control) • Diseño de un lenguaje formal o natural representado por una MT por módulos.	6 horas

Comprensión y capacidad de diseñar una máquinas de Turing en los diferentes lenguajes aceptados por una MT	• Búsqueda de conceptos por equipos en los que podrá leer, descubrir, cuestionar, preguntar indagar, sobre los diferentes lenguajes aceptados por una MT. • Discusión en clase para expresarse, comunicar ideas, etc. • Creación de ideas originales obtenidas en consenso • Ejemplos de lenguajes aceptados por diferentes lenguajes en una máquina de Turing.	• Informes por equipo y conclusiones grupales. • Técnicas de observación(registros, listas de control) • Diseño de un lenguaje formal o natural representado por los diferentes lenguajes aceptados por una MT.	6 horas
Comprensión y capacidad de diseñar una máquina de Turing con una de las variantes de MT	• Búsqueda de conceptos por equipos en los que podrá leer, descubrir, cuestionar, preguntar indagar, sobre los diferentes variantes de una MT. • Discusión en clase para expresarse, comunicar ideas, etc. • Creación de ideas originales obtenidas en consenso • Ejemplos de lenguajes aceptados por diferentes lenguajes en una máquina de Turing.	• Informes por equipo y conclusiones grupales. • Técnicas de observación(registros, listas de control) • Diseño de un lenguaje formal o natural representado por una de las variantes de una MT.	6 horas

4 MÁQUINAS DE TURING

4.1 DEFINICIÓN DE MÁQUINA DE TURING (MT)

Alan Turing inventó la máquina que lleva su nombre (Máquina de Turing) en un intento por determinar si toda la matemática podía ser reducida a algún tipo simple de computación. Su objetivo fue desarrollar la máquina más simple posible, capaz de realizar computación. La máquina propuesta por Turing es un dispositivo relativamente simple, pero capaz de realizar cualquier operación que no sean resueltas con gramáticas, AF, AP. Turing abrigó la ilusión de que su máquina tenía una capacidad tal que, potencialmente, podría ser capaz de realizar cualquier cosa que pueda hacer el cerebro humano, incluyendo la capacidad de poseer conciencia de sí mismo.

Una Máquina de Turing, es una colección de celdas de almacenamiento que se extiende infinitamente en ambas direcciones, esencialmente es una cinta infinita donde cada celda es capaz de almacenar un símbolo.

Es una 7- tupla, M

$$M = (Q, \Sigma, T, s, \emptyset, F, \delta)$$

Q = Es un conjunto finito de estados.

Σ = Representa el alfabeto de entrada.

T = Representa el alfabeto de la cinta.

s = Representa estado de inicio.

$\emptyset$ = Representa símbolo en blanco.

F = Representa el conjunto de estados finales o aceptación.

δ = Función de transición

Modelo matemático simple de una computadora, independiente de su simplicidad. La cual tiene la capacidad de modelar la capacidad de cálculo de una computadora de propósito general.

Es una representación abstracta del ambiente de computación y consiste en buscar la cabeza para leer o escribir.

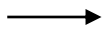
Es un modelo por el cual se puede validar expresiones de acuerdo a un patrón establecido.

La Máquina de Turing es un modelo matemático simple de una computadora. A pesar de su simplicidad, modela la capacidad de computación (cálculo) de una computadora de propósito general.

Funcionamiento de la máquina de turing

Para poder entender el funcionamiento de la máquina de Turing, se deben conocer las partes que constituyen una máquina.

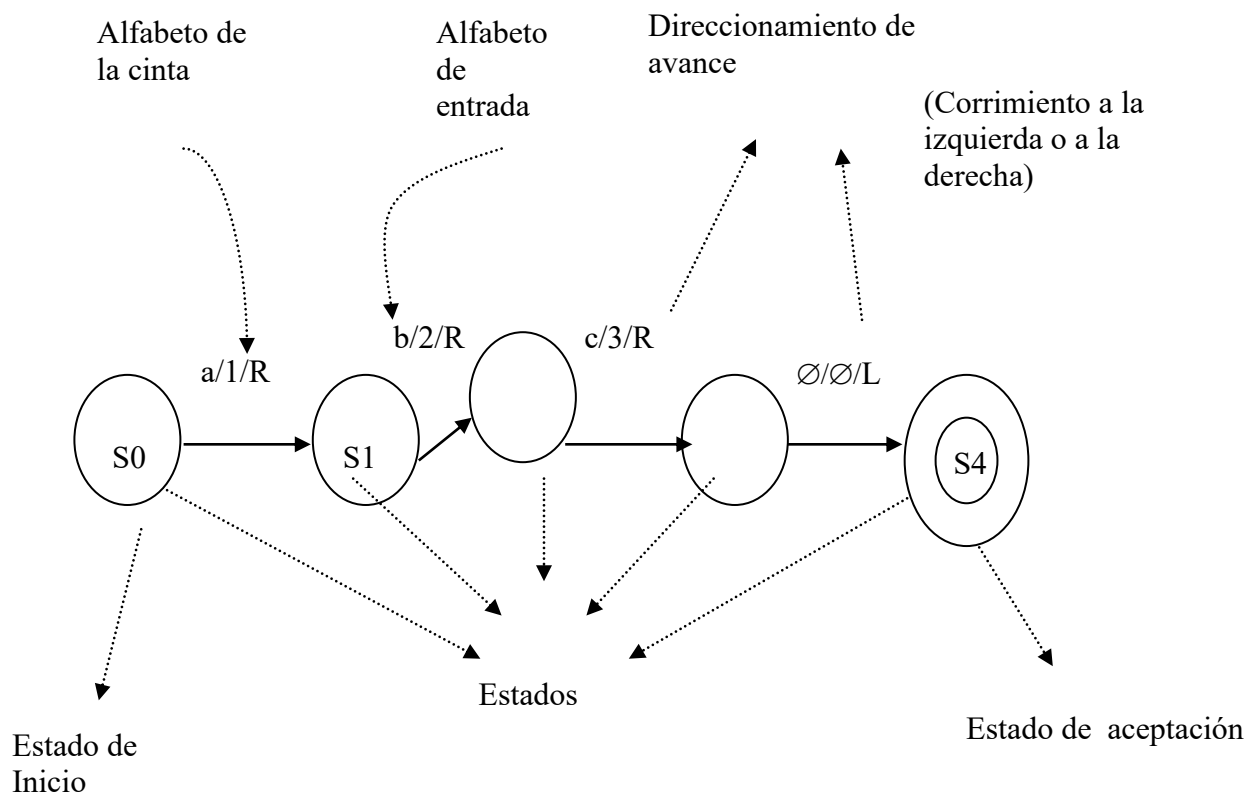
$$T = \{Q, \Sigma, T, s, \emptyset, F, \delta\}$$

Partes de la máquina de Turing	Ejemplo
Q = conjunto finito de estados	$s_0, s_1, \dots, s_n,$
Σ = alfabeto de entrada	$a, b,$ 
T = alfabeto de la cinta	$1, 2, 3$
S = estado de inicio	S_0
$\emptyset$ = espacio en blanco Representa el final de una cadena	$\emptyset$
F = estado de aceptación	
Direccionamiento	R (corrimiento a la derecha) L (corrimiento a la izquierda)
Flujo	
δ Función de transición	$\delta (q_0, 1) = (q_1, 2, R)$

La máquina Turing consiste de una cinta de longitud infinita dividida en celdas. En cada celda es posible escribir un número finito de símbolos mediante un cabezal de lectura y escritura. En cada paso del proceso, la cinta puede ser desplazada una celda hacia la izquierda o hacia la derecha con respecto al Cabezal lectura-escritura.

La máquina se encuentra siempre en uno de un conjunto finito de posibles estados, los que pueden ser representados por cualquier símbolo (no necesariamente números).

En la siguiente figura se muestran las partes de las máquinas de Turing:



Donde:

$Q = \{ S_0, S_1, S_2, S_3, S_4 \}$

$T = \{ 1, 2, 3 \}$

$\Sigma = \{ a, b, c \}$

$S = S_0$

$F = S_4$

Un concepto del funcionamiento se puede resumir en 4 puntos:

- 1.-Lectura del símbolo situado bajo del dispositivo de lectura/ escritura.
- 2.-Consulta la entrada de la máquina determinada por el estado de la máquina y el símbolo de entrada.
- 3.- Escritura del símbolo de entrada de la máquina.
- 4.-Se desplaza la cinta en la dirección dada (R/L) y pasa al estado siguiente.

Las máquinas de Turing, pueden ser representadas por un conjunto de funciones de transición.

A continuación se muestra un ejemplo de un conjunto de transiciones.

$Q = \{q_0, q_1, q_2, q_3\}$

$S_0 = \{q_0\}$

$F = \{q_3\}$

$\Sigma = \{a, b\}$

$T = \{a, b, \emptyset\}$

$(q_0, \text{b}) = (q_3, \text{b}, R)$

$(q_0, a) = (q_1, a, R)$

$(q_1, a) = (q_1, a, R)$

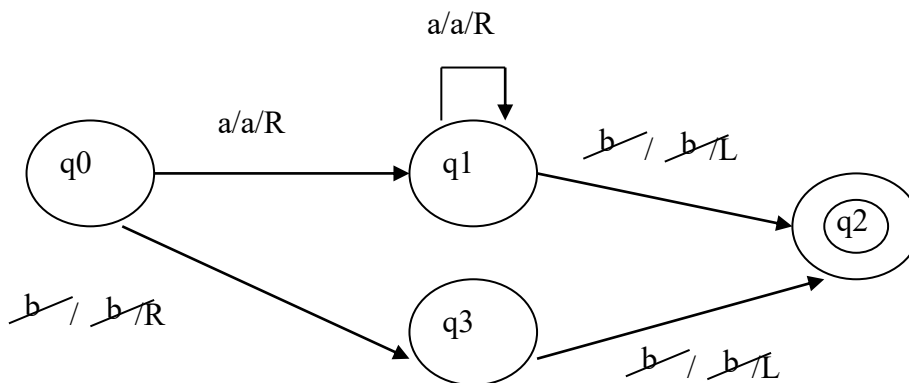
$(q_1, \text{b}) = (q_2, \text{b}, L)$

$(q_3, \text{b}) = (q_2, \text{b}, L)$

La función de transición $(q_0, \text{b}) = (q_3, \text{b}, R)$, se lee como sigue:

Situándose bajo el dispositivo de lectura en el estado q_0 , consulta en la entrada b , para pasar al estado q_3 y sobrescribir un b y se desplaza la cinta a la derecha.

Por medio de un grafo se representa como sigue:

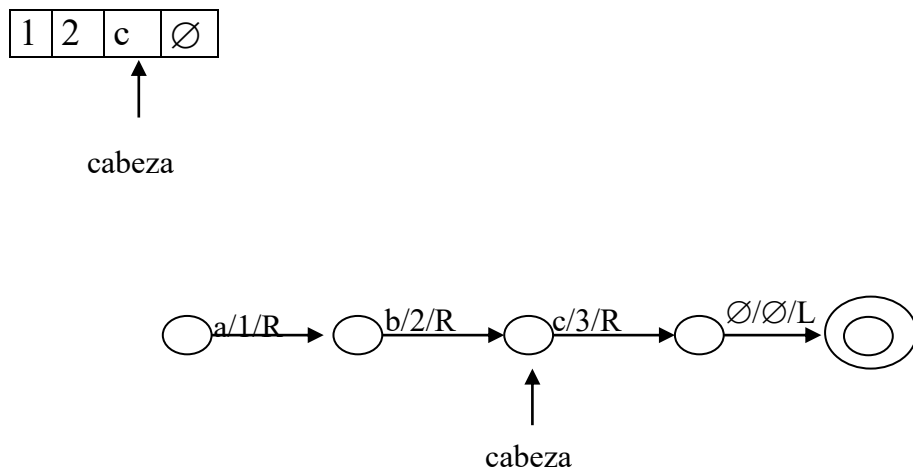
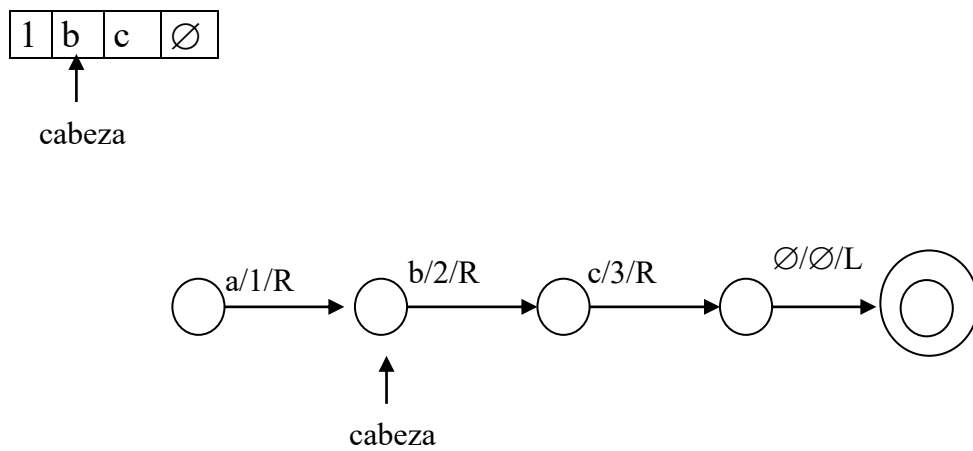
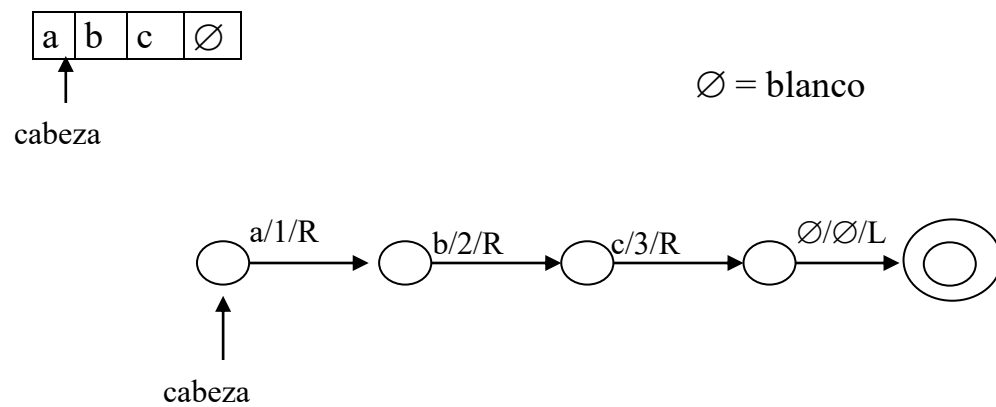


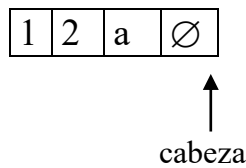
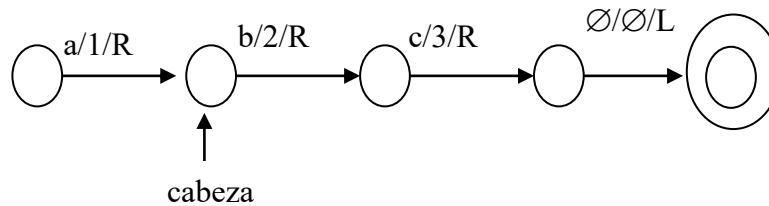
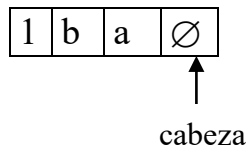
La expresión regular aceptada por la máquina de Turing es a^* .

En las siguientes figuras se muestra el funcionamiento de máquinas de Turing, para una cadena válida para la expresión “abc”.

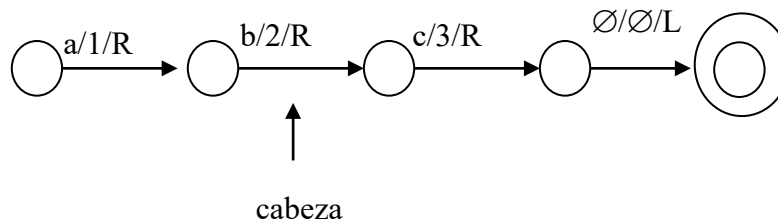
Funcionamiento

Cinta





Cadena no válida
porque no existe una
letra "a" después de la
letra "b"

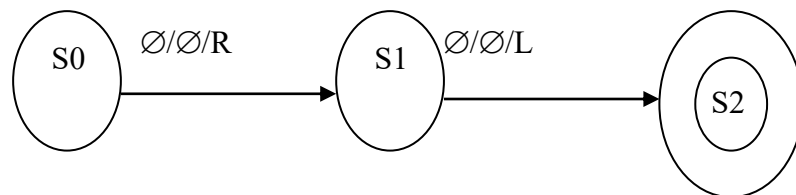


Nota:

Para que una cadena sea válida, necesita llegar a un estado de aceptación.

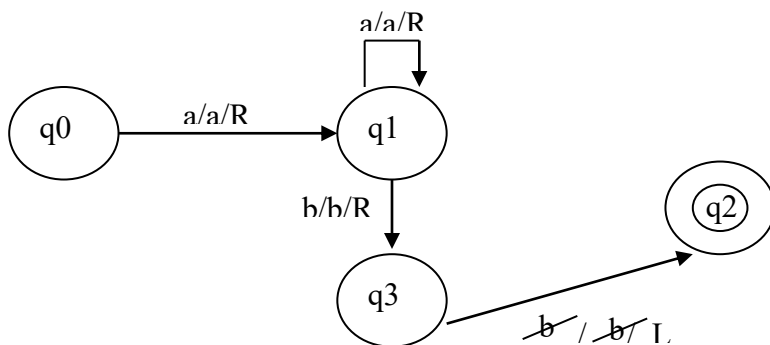
Una regla fundamental de las máquinas de Turing, es que el estado de inicio, no puede ser de aceptación, debido a que como se ha mencionado la Máquina lee sobre una cinta, por lo que necesita tener un movimiento, para reconocer la cadena vacía. Si se desea que la MT acepte la cadena vacía se necesitan realizar dos movimientos que se resumen en la siguiente figura.

$\emptyset$ = blanco

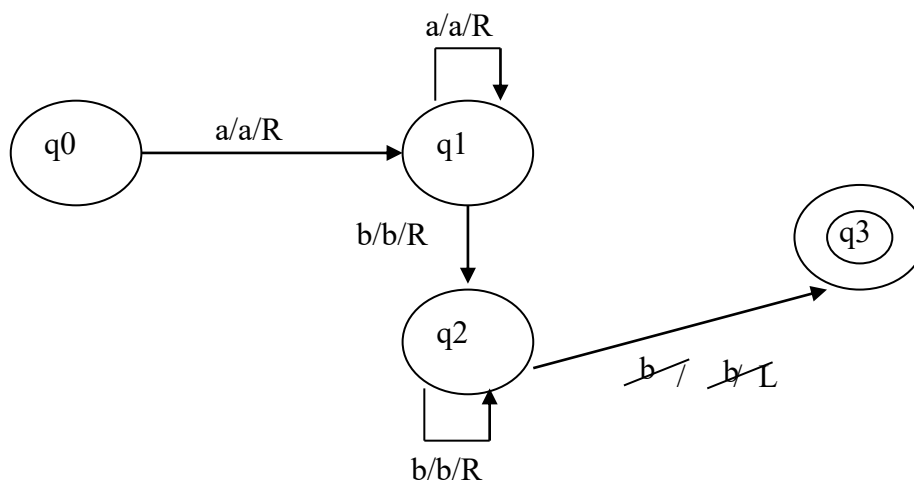


Ejemplos de máquinas de turing

Máquina de Turing que acepta una o más a's seguida de una b

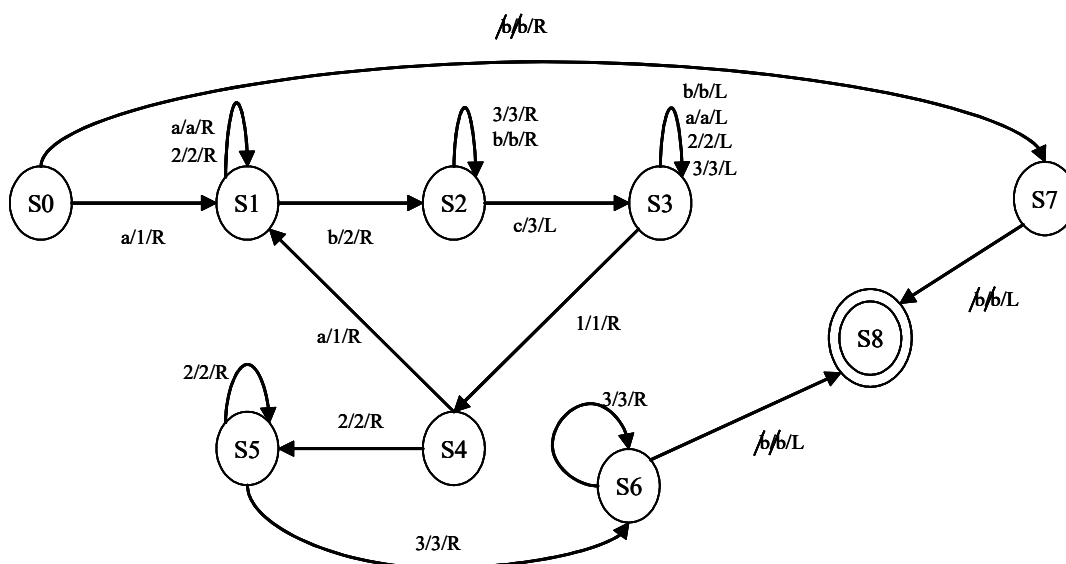


Máquina que acepta una o más a's seguida(s) por una o más b's



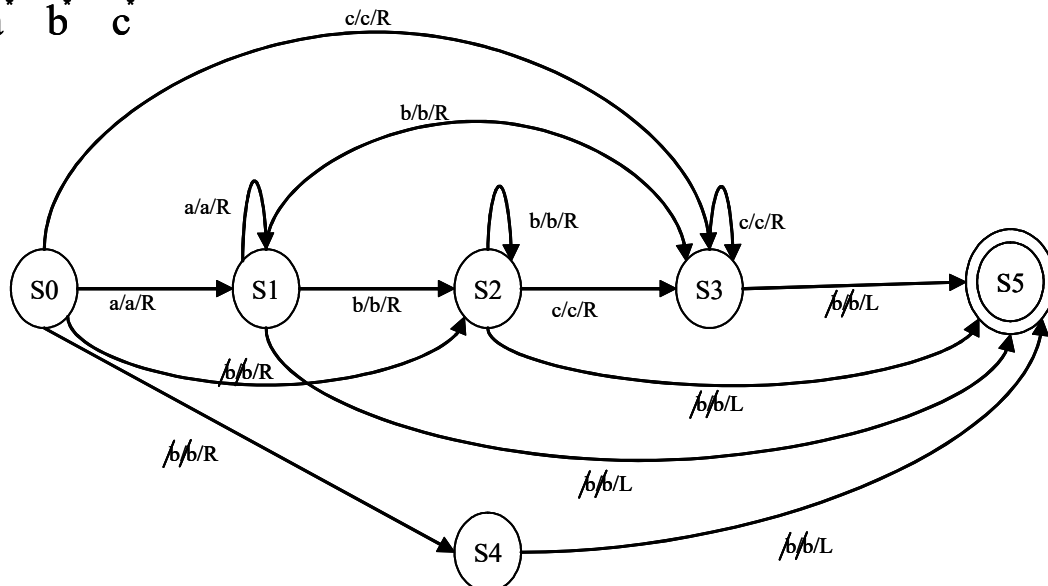
Máquina que acepta la expresión no regular:

$a^n b^n c^n$



Máquina que acepte abc cero o más veces

$a^* b^* c^*$



Maquinas de Turing, a partir de expresiones regulares, gramáticas, AF, AP.

Máquina de Turing a partir de la siguiente expresión regular $(a^*abb^*baa^*)^+$

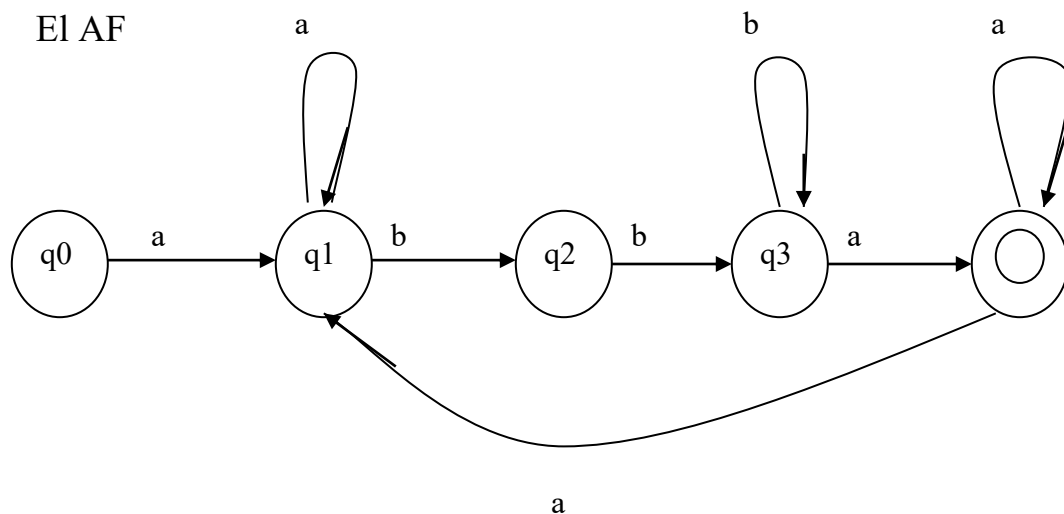
El conjunto de producciones de la expresión regular es:

$S0 \rightarrow aS0 | abS1$

$S1 \rightarrow bS1 | baS2 | ba$

$S2 \rightarrow aS2 | a | S0$

El AF

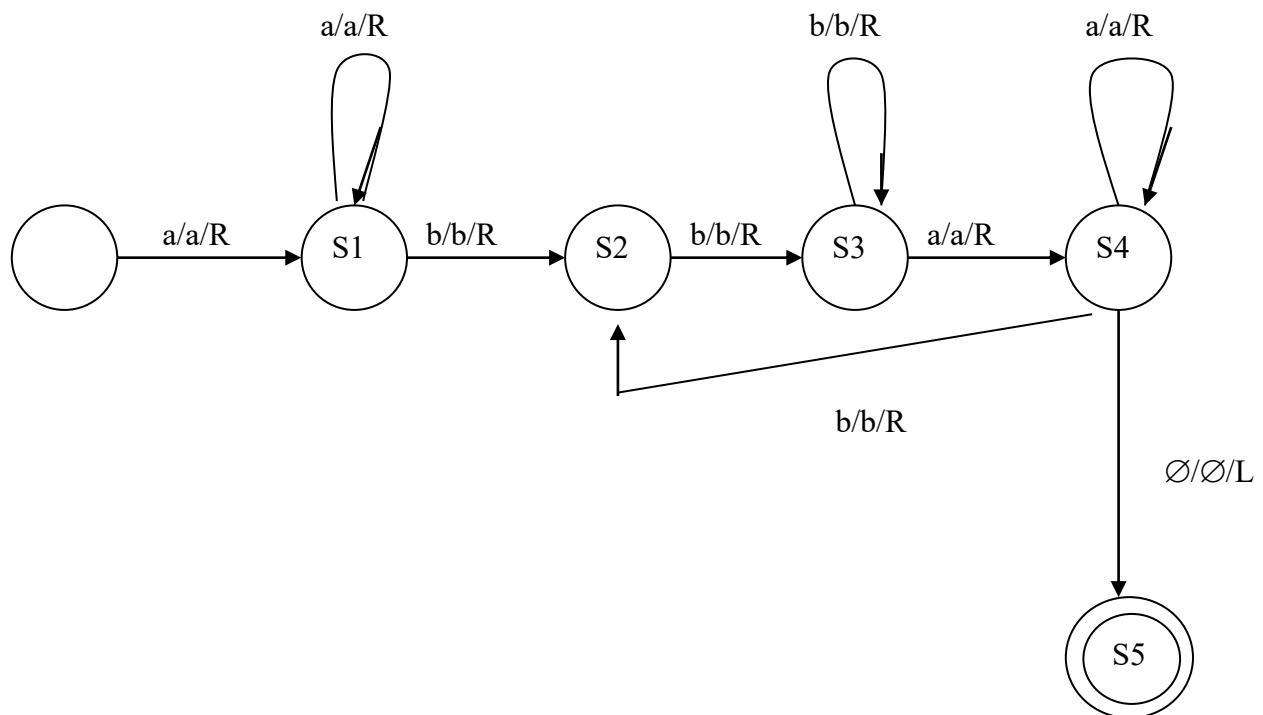


El AP

$(q_0, a, z) = (q_1, az)$
 $(q_1, a, a) = (q_1, aa)$
 $(q_1, a, b) = (q_2, ab)$
 $(q_2, b, b) = (q_3, bb)$
 $(q_3, b, b) = (q_3, bb)$
 $(q_3, b, a) = (q_4, ba)$
 $(q_4, a, a) = (q_4, aa)$
 $(q_4, a, a) = (q_1, aa)$
 $(q_4, a, \lambda) = (q_4, a)$

Donde q_4 es estado de aceptación

Máquina de Turing



Máquina de Turing a partir de la siguiente expresión no regular:

$a^n bc d^m$, donde: $n < m$:

El conjunto de producciones de la expresión regular es:

$S_0 \rightarrow aS_0d$

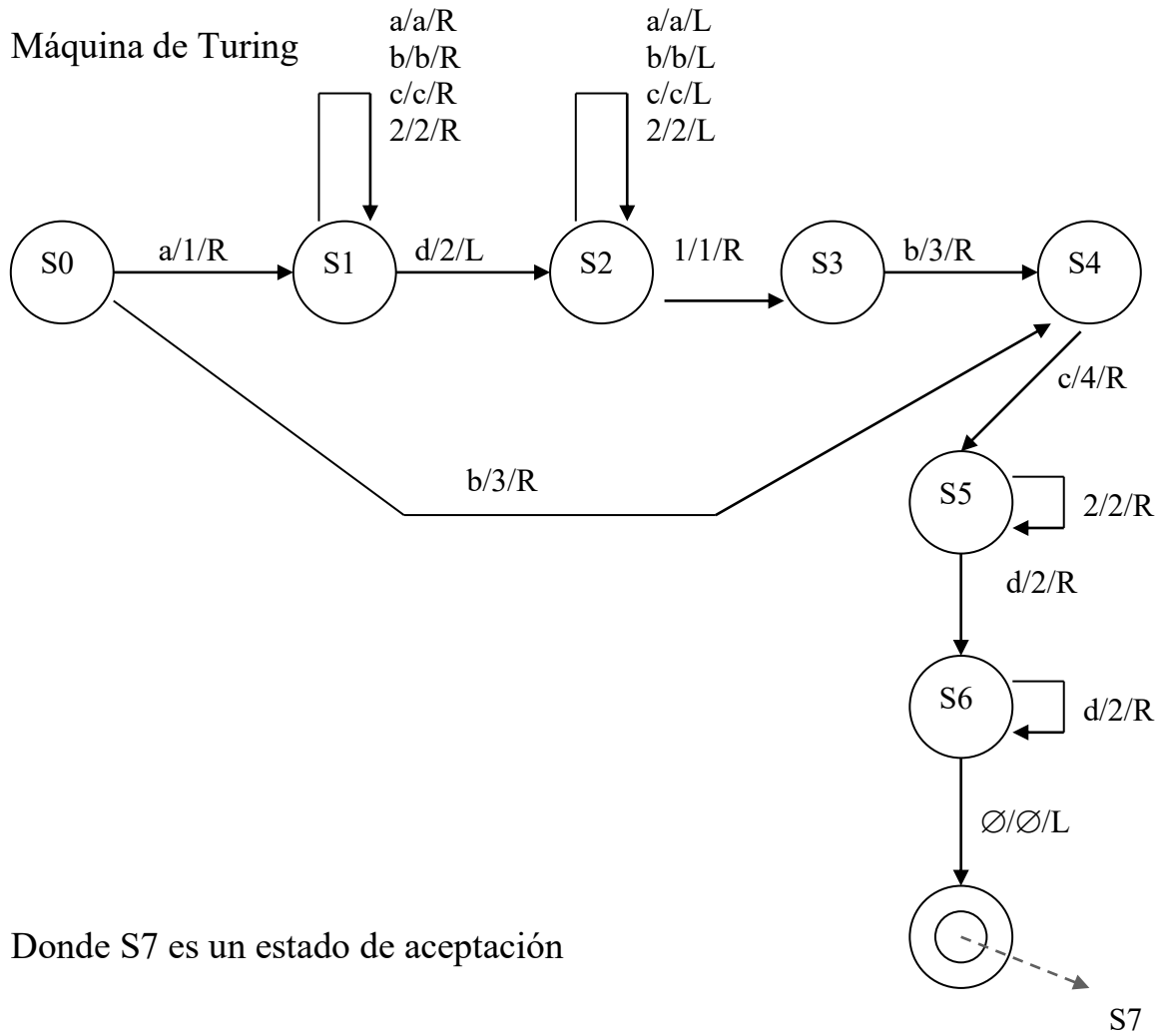
$S_0 \rightarrow bS_1$

$S_1 \rightarrow cS_2$

$S_2 \rightarrow dS_2|d$

El AP no es posible generarlo.

$(S0, a, z) = (S0, az)$
 $(S0, b, z) = (S0, bz)$
 $(S0, b, a) = (S1, ba)$
 $(S1, c, b) = (S2, \lambda)$
 $(S2, d, a) = (S3, da)$
 $(S2, d, z) = (\underline{S4}, z)$
 $(S3, d, d) = (S3, \lambda)$
 $(S3, d, a) = (S3, \lambda)$
 $(S3, \lambda) = (\underline{S4}, z)$



Donde $S7$ es un estado de aceptación

4.2 CONSTRUCCIÓN MODULAR DE UNA MAQUINA DE TURING

La creación modular de máquinas de Turing, permite desarrollar máquinas complejas a partir de bloques de elementales partiendo de máquinas más pequeñas, mediante diagramas de transiciones.

La construcción de máquinas de Turing se lleva a cabo mediante los diagramas de transición y combinarlos de manera parecida a lo que se realiza en la formación de la unión y concatenación de los autómatas finitos.

Pasos para la construcción de una máquina de Turing

- 1.-Elimine las características de inicio de los estados iniciales de las máquinas, excepto la de aquel donde iniciara la máquina compuesta.
- 2.-Elimine las características de detención de los estados de parada de todas la máquinas e introduzca un nuevo estado de parada que no se encuentre en ninguno de los diagramas que se combinan.
- 3.-Para cada uno de los antiguos estados de parada p y cada x en T , dibuje un arco de la siguiente forma:
 - a. Si la máquina compuesta debe detenerse al llegar a p con el símbolo actual x , dibuje un arco con etiqueta x/x de p al nuevo estado de parada.
 - b. Si al llegar al estado p con el símbolo actual x , la máquina compuesta debe transferir el control a la máquina $M=(Q,\Sigma,T,\delta,s,f)$, dibuje entonces un arco con etiqueta x/z de p al estado de M , donde $\delta(t,x)=(q,z,R)$.

Se puede combinar dos máquinas de Turing permitiendo que compartan la misma cinta y, que cuando una termine su ejecución, la otra empiece. El contenido de la cinta cuando comienza la ejecución de la segunda máquina de Turing, está formado por todo lo que dejó la primera máquina de Turing, y la cabeza de lectura/escritura de la segunda se situará, al comienzo de la ejecución, sobre la celda de la cinta sobre la que terminó la primera.

$$\rightarrow M_1 \rightarrow M_2$$

La máquina compuesta simula las acciones de M_1 hasta el punto donde normalmente se detendría y luego simula las acciones de M_2 .

Ejemplo:

Sea M_1 dada por:

$$Q_1 = \{q_1, q_2, q_3, q_4\}$$

$$\Sigma = a, b$$

$$T = \{a, \text{b}\}$$

$$S1 = \{q1\}$$

$$F1 = \{q4\}$$

y δ esta dada por:

$$(q1, a) = (q2, a, R)$$

$$(q2, a) = (q2, a, R)$$

$$(q2, b) = (q3, b, R)$$

$$(q3, \text{b}) = (q4, \text{b}, R)$$

Sea M2 dada por:

$$Q2 = \{p1, p2, p3, p4\}$$

$$\Sigma = a, b$$

$$T = \{a, \text{b}\}$$

$$S2 = \{p1\}$$

$$F2 = \{p4\}$$

y δ esta dada por:

$$(p1, b) = (p2, b, R)$$

$$(p2, b) = (p2, b, R)$$

$$(p2, a) = (p3, a, R)$$

$$(p3, \text{b}) = (p4, \text{b}, L)$$

$$M1 = (Q1, \Sigma, T, s1, F1, \delta1) \quad y \quad M1 = (Q2, \Sigma, T, s2, F2, \delta2)$$

Se supone que $Q1 \cap Q2 = \emptyset$. La composición de las máquinas de Turing M1 y M2 es la máquina de Turing $M = (Q, \Sigma, T, s, F, \delta)$ que se denota $M1M2$.

Tendrá las transiciones dadas mediante:

$$(q1, a) = (q2, a, R)$$

$$(q2, a) = (q2, a, R)$$

$$(q2, b) = (q3, b, R)$$

$$(q3, \text{b}) = (p1, \text{b}, R) \quad s=q1 \quad y \quad F= p4$$

$$(p1, b) = (p2, b, R)$$

$$(p2, b) = (p2, b, R)$$

$$(p2, a) = (p3, a, R)$$

$$(p3, \text{b}) = (p4, \text{b}, L)$$

M1 busca el primer blanco que haya a la derecha de la posición actual de la cabeza. Consideremos una cinta:

a a a b**␣**bbba**␣**



Posición actual de la cabeza

La máquina de Turing compuesta terminaría:

a a a b**␣**bbba**␣**



Posición final

Se puede combinar dos máquinas de Turing permitiendo que compartan la misma cinta y que cuando una termine su ejecución la otra empiece solo si la celda actual contiene una x de lo contrario, se detiene al concluir las acciones de la primera máquina.

$$\rightarrow M_1 \xrightarrow{x} M_2$$

Ejemplo:

Sea M1 dada por:

$$Q1 = \{q1, q2, q3, q4\}$$

$$\Sigma = a, b$$

$$T = \{a, \text{␣}\}$$

$$S1 = \{q1\}$$

$$F1 = \{q4\}$$

y δ esta dada por:

$$(q1, a) = (q2, a, R)$$

$$(q2, a) = (q2, a, R)$$

$$(q2, b) = (q3, b, R)$$

$$(q3, \text{␣}) = (q4, \text{␣}, R)$$

Sea M2 dada por:

$$Q2 = \{p1, p2, p3, p4\}$$

$$\Sigma = a, b$$

$$T = \{a, \text{b}\}$$

$$S2 = \{p1\}$$

$$F2 = \{p4\}$$

y δ esta dada por:

$$(p1, b) = (p2, b, R)$$

$$(p2, b) = (p2, b, R)$$

$$(p2, a) = (p3, a, R)$$

$$(p3, \text{b}) = (p4, \text{b}, L)$$

$$M1 = (Q1, \Sigma, T, s1, F1, \delta1) \quad \text{y} \quad M1 = (Q2, \Sigma, T, s2, F2, \delta2)$$

Se supone que $Q1 \cap Q2 = \emptyset$. La composición de las máquinas de Turing M1 y M2 es la máquina de Turing $M = (Q, \Sigma, T, s, F, \delta)$ que se denota $M1M2$.

Tendrá las transiciones dadas mediante:

$$(q1, a) = (q2, a, R)$$

$$(q2, a) = (q2, a, R)$$

$$(q2, b) = (q3, b, R)$$

$$(q3, \text{b}) = (p8, \text{b}, R)$$

$$(p8, x) = (p1, x, R)$$

$$(p8, \text{b}) = (p8, \text{b}, L) \quad s = q1 \text{ y } F = p8, p4$$

$$(p1, b) = (p2, b, R)$$

$$(p2, b) = (p2, b, R)$$

$$(p2, a) = (p3, a, R)$$

$$(p3, \text{b}) = (p4, \text{b}, L)$$

M1 busca el primer blanco que haya a la derecha de la posición actual de la cabeza si viene una x continua con la máquina M2. Consideremos una cinta:

a a a b b b b a b



Posición actual de la cabeza

La máquina de Turing compuesta terminaría:

a a a b **x** b b a **␣**
 ↑
 Posición final

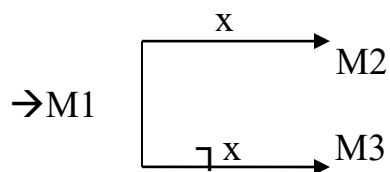
M1 busca el primer blanco que haya a la derecha de la posición actual de la cabeza si no viene una x termina sin procesar la máquina M2. Consideremos una cinta:

a a a b **␣** b b b a **␣**
 ↑
 Posición actual de la cabeza

La máquina de Turing compuesta terminaría:

a a a b **x** b b a **␣**
 ↑
 Posición final

Se puede combinar tres máquinas de Turing permitiendo que compartan la misma cinta y cuando M1 termine su ejecución empiece M2 si solo si la celda actual contiene una x de lo contrario se ejecuta M3



Una subrutina de una máquina de Turing consiste en un conjunto de estados que realiza un proceso útil. Este conjunto de estados incluye un estado inicial y otro estado en el que, temporalmente no se realizan movimientos y que sirve como estado de retorno para poder pasar el control a cualquier otro conjunto de estados que llame a la subrutina

4.3 LENGUAJES ACEPTADOS POR UNA MÁQUINA TURING

Cada lenguaje regular es a su vez libre del contexto, asimismo un lenguaje libre del contexto es también dependiente del contexto, éste es recursivo y a su

vez, recursivamente enumerable. Las inclusiones son, sin embargo, propias, es decir, existen en cada nivel lenguajes que no están en niveles anteriores.

En la tabla 4. Se relacionan los tipos de lenguajes, lenguaje, autómata y reglas de producción que son aceptados por una máquina de Turing.

Tabla 4. Lenguajes aceptados por una máquina de Turing.

Tipo	Lenguaje	Autómata	Reglas de producción de gramáticas
0	recursivamente enumerable (LRE)	Máquina de Turing (MT)	Sin restricciones
1	dependiente del contexto (LSC)	Autómata linealmente acotado (ALA)	$\alpha A \beta \rightarrow \alpha \gamma \beta$
2	independiente del contexto (LLC)	Autómata de pila(AP)	$A \rightarrow \gamma$
3	regular (RL)	Autómata finito(AF)	$A \rightarrow aB$ $A \rightarrow a$

Lenguajes Recursivamente Enumerables (de tipo 0, parcialmente decidible, Turing computable).

Las gramáticas que generan estos lenguajes pueden tener reglas compresoras.

Las reglas de producción son de la siguiente forma:

$$P = \{(u \rightarrow v) | u = xAy; u \in \Sigma^+; v, x, y \in \Sigma^*; A \in N\}$$

Lenguajes Dependientes del Contexto (sensibles al contexto, de tipo 1)

No existen reglas, salvo, opcionalmente, la que deriva el axioma a la palabra vacía. Existen reglas en las que un símbolo no terminal puede derivar a formas sentenciales distintas, según los símbolos que aparezcan a su alrededor.

Las reglas de producción son de la siguiente forma:

$$P = \{(S \rightarrow \lambda) \vee (xAy \rightarrow xvy) \mid v \in \Sigma^+; x, y \in \Sigma^*; A \in N\}$$

Lenguajes Independientes del Contexto (Libres de contexto, de tipo 2)

La mayoría de los lenguajes de programación entran en ésta categoría.

Las reglas de producción son de la siguiente manera:

$$P = \{(S \rightarrow \lambda) \vee (A \rightarrow v) \mid v \in \Sigma^+; A \in N\}$$

Lenguajes Regulares (de tipo 3)

Son los lenguajes más simples dentro la Jerarquía de Chomsky. Se suelen expresar mediante expresiones regulares.

Existen 2 tipos: lineales por la derecha y lineales por la izquierda. Las reglas de producción son de la siguiente forma:

Lineales por la

derecha: $P = \{(S \rightarrow \lambda) \vee (A \rightarrow aB) \vee (A \rightarrow a) \mid a \in T; A, B \in N\}$

Lineales por la

izquierda: $P = \{(S \rightarrow \lambda) \vee (A \rightarrow Ba) \vee (A \rightarrow a) \mid a \in T; A, B \in N\}$

Lenguaje recursivamente enumerable y recursivo

Un lenguaje L sobre un alfabeto Σ se dice que es *recursivamente enumerable* si es aceptado por alguna máquina de Turing. Es decir, L es recursivamente enumerable si para alguna máquina de Turing M tenemos que:

$$L(M) = \{w \in \Sigma^* \mid qw \vdash^* upv \text{ para } p \in F \text{ y } u, v \in \Sigma^*\}$$

(Donde q es el estado inicial de M y F es el conjunto de estados finales de M).

Un lenguaje L es *recursivo* si L es recursivamente enumerable y hay alguna máquina de Turing que para sobre *todas* las entradas que acepta L .

Lenguajes regulares, independientes del contexto, recursivos y recursivamente enumerables.

Sea $M = (Q, \Sigma, s, F, \delta)$ un autómata finito determinista. Se puede construir una máquina de Turing $M' = (Q', \Sigma', \Gamma, s', ?, F', \delta')$ para la cual $L(M) = L(M')$ por medio de:

$$Q' = Q \cup \{q'\}, \text{ donde } q' \text{ es un nuevo estado que no está en } Q.$$

$$\Sigma' = \Sigma$$

$$\Gamma = \Sigma \cup \{?\}$$

$$F' = \{q'\}$$

$$\delta'(q, \sigma) = (\delta(q, \sigma), \sigma, R), \text{ para } q \in Q \text{ y } \sigma \in \Sigma'$$

$$\delta'(q, ?) = (q', ?, S), \text{ para todo } q \in F, \text{ donde } S \text{ es la directiva de cinta}$$

“no moverse”

Teorema. Si L es un lenguaje regular, entonces L es también un lenguaje recursivo.

Teorema. Si L es un lenguaje independiente del contexto, entonces L es también un lenguaje recursivo.

Teorema. Si L_1 y L_2 son lenguajes recursivos, entonces $L_1 \cap L_2$ también lo es.

Lema. Si L es un lenguaje recursivo, entonces $\Sigma^* - L$ es un lenguaje recursivo.

Teorema. Hay un lenguaje recursivamente enumerable L para el cual $\Sigma^* - L$ no es recursivamente enumerable.

Teorema. Si L_1 y L_2 son lenguajes recursivamente enumerables, entonces $L_1 \cup L_2$ es también recursivamente enumerable.

Teorema. Si L es un lenguaje recursivamente enumerable para el cual $\Sigma^* - L$ también es recursivamente enumerable, entonces L es un lenguaje recursivo.

Teorema. Un lenguaje L es recursivamente enumerable si y solo si L es enumerado por alguna máquina de Turing.

Definición. Una *gramática no restringida* (que también se conoce como una gramática estructurada por frase) es una 4-tupla $G = (N, \Sigma, S, P)$, donde

N es un alfabeto de símbolos no terminales

Σ es un alfabeto de símbolos terminales con $N \cap \Sigma = \emptyset$

$S \in N$ es el símbolo inicial

P es un conjunto finito de producciones de la forma $\alpha \rightarrow \beta$, donde $\alpha \in (N \times \Sigma)^+$ y $\beta \in (N \times \Sigma)^*$ (es decir, $P \subseteq (N \times \Sigma)^+ \times (N \times \Sigma)^*$ y es un conjunto finito).

Teorema. Si G es una gramática no restringida, entonces $L(G)$ es un lenguaje recursivamente enumerable.

Teorema. Si $L = L(M)$ es un lenguaje recursivamente enumerable y G se construye de la forma descrita, entonces $L = L(G)$.

Teorema. Un lenguaje L es recursivamente enumerable si y solo si $L = L(G)$ para alguna gramática G , no restringida.

Definición. Una gramática $G = (N, \Sigma, S, P)$ es una *gramática sensible al contexto* si todas las producciones son de la forma $\alpha \rightarrow \beta$, donde $\alpha, \beta \in (N \times \Sigma)^+$ y $|\alpha| \leq |\beta|$.

Lema. El conjunto de los lenguajes sensibles al contexto contiene al conjunto de los lenguajes independientes del contexto.

Lema. Sea $G = (N, \Sigma, S, P)$ una gramática sensible al contexto. Entonces, existe una gramática $G_1 = (N_1, \Sigma, S_1, P_1)$ sensible al contexto, para la cual $L(G) = L(G_1)$ y S_1 nunca aparece en el lado derecho de una producción de P_1 .

Lema. Sea $G = (N, \Sigma, S, P)$ una gramática sensible al contexto. Entonces, existe una máquina de Turing T , que para sobre toda entrada y acepta $L(G)$.

Teorema. Si L es un lenguaje sensible al contexto, entonces L es recursivo.

Lema. Hay lenguajes recursivos que no son lenguajes sensibles al contexto.

Teorema. Los lenguajes sensibles al contexto contienen, propiamente, a los lenguajes independientes del contexto. A su vez, los lenguajes recursivos contienen, propiamente, a los lenguajes sensibles al contexto.

Teorema. Sobre un alfabeto dado, el conjunto de los lenguajes recursivamente enumerable contienen propiamente a los lenguajes recursivos que contienen propiamente al conjunto de los lenguajes sensibles al contexto que contienen propiamente al conjunto de los lenguajes independientes del contexto. Que a su vez, contienen propiamente los lenguajes regulares.

$$L_{\text{regular}} \subset L_{\text{i.c.}} \subset L_{\text{s.c.}} \subset L_{\text{recursivo}} \subset L_{\text{r.e}}$$

4.4 VARIANTES DE UNA MAQUINA DE TURING

Hay otras definiciones de las máquinas de Turing que son equivalentes. Algunos de esos modelos alternativos son mucho más complicados aunque todos tienen la misma potencia computacional (o de cálculo). Muchas de ellas dotan de mayor flexibilidad al diseño de una máquina de Turing que resuelva un problema en particular.

Máquina de Turing con Directiva de Permanecer

Recuérdese que la máquina de Turing sencilla sitúa la cabeza de lectura/escritura sobre el primer B que haya a la izquierda de la posición actual. Para hacerlo, busca fuera de la celda actual y retrocede. Esto es debido a la definición original que requiere que por cada transición se mueva la cabeza de la cinta.

La función de transición estaba definida como: $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, L\}$ y puede ser modificada como: $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, L, S\}$ donde S significa “permanecer”, es decir no mover la cabeza de lectura/escritura.

Por tanto $\delta(q, \sigma)=(p, \sigma', S)$ significa que se pasa del estado q al p, se escribe σ' en la celda actual y la cabeza se queda sobre la celda actual.

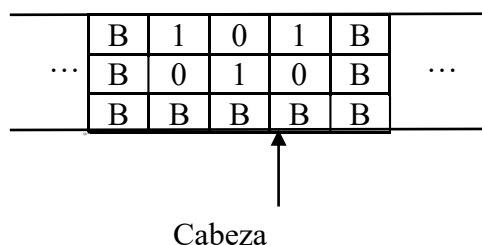
Máquina de Turing Multipista

Es aquella mediante la cual cada celda de la cinta se divide en subceldas. Cada subcelda es capaz de contener símbolos de la cinta. La cinta tiene cada celda subdividida en tres subceldas. Se dice que esta cinta tiene múltiples pistas. Puesto que cada celda de esta máquina de Turing contiene múltiples caracteres, el contenido de las celdas de la cinta puede ser representado mediante n-tuplas ordenadas. Los movimientos que realice esta máquina dependerán de su estado actual y de la n-tupla que represente el contenido de la celda actual.

	B	b	B	
...	a	a	b	...
	a	a	B	

Una máquina de Turing multipista no tiene más potencia que la máquina de Turing original. Sin embargo, hace que sea más fácil la construcción de máquinas de Turing que resuelvan ciertos problemas.

Ejemplo: Para una máquina de Turing que sume dos números binarios. Primero se construye una máquina de Turing de tres pistas. La entrada serán dos números binarios que ocupen las dos pistas superiores de la cinta. Suponiendo que sus dígitos se alinean por la derecha, que sus representaciones binarias son de la misma longitud (lo que se puede conseguir rellenándolas con tantos ceros como sea necesario) y que la cabeza de lectura/escritura se sitúa sobre la celda del extremo izquierdo de la cadena. Por tanto, si tuvieran que sumar 101 y 10, la cinta debería contener.



La máquina de Turing realizará la suma en la tercera pista. Por tanto, el alfabeto de cinta estará formado por las ternas:

(B, B, B)(1, 1, B)
 (1, 1, 0)(1, 1, 1)
 (0, 0, B)(0, 0, 0)
 (0, 0, 1)(B, B, 1)
 (0, 1, B)(0, 1, 0)
 (0, 1, 1)(1, 0, B)
 (1, 0, 0)(1, 0, 1)

Esta máquina de Turing buscará primero hacia la derecha el extremo derecho de los números que van a ser sumados. Entonces sumará pares de dígitos, desde la derecha hacia la izquierda, llevando la cuenta de los resultados que se obtengan y sumando a quienes corresponda.

Máquina de Turing de Cinta infinita en una Dirección

Máquina de Turing que usa una cinta que se extiende infinitamente en una única dirección. Generalmente, se tiene una cinta que se extiende infinitamente hacia la derecha. No está permitido realizar ningún movimiento hacia la izquierda a partir de la celda del extremo izquierdo. Desde luego,

cualquier máquina de Turing de esta forma puede ser simulada por una de las que responden a la definición original. Para cada computación, simplemente se marca una de las celdas de la cinta infinita por los dos lados, como la celda que se encuentra en el límite izquierdo.

Máquina de Turing en Dos Direcciones

Una máquina de Turing con una cinta infinita en un sentido puede simular una máquina de Turing con la cinta infinita en los dos sentidos pero con dos pistas. Sea M una máquina de Turing con una cinta infinita en los dos sentidos. La máquina de Turing M' , que tiene una cinta infinita en un sentido, puede simular a M si tiene una cinta con dos pistas. La cinta superior contiene la información correspondiente a la parte derecha de la cinta M , a partir de un punto de referencia dado. La pista inferior contiene la parte izquierda de la cinta M (en orden inverso).

Máquina de Turing Multicinta

La máquina de Turing multicinta tiene varias cintas, cada una de las cuales tiene su propia cabeza de lectura/escritura. Las cabezas de lectura/escritura se controlan independientemente (es decir, al mismo tiempo, no tienen que moverse en la misma dirección, ni realizar el mismo número de movimientos, ni incluso, hacer nada a la vez).

Cambia de estado dependiendo del estado actual y del contenido de las celdas de todas las cintas, que están analizando actualmente las cabezas de lectura/escritura.

Escriben un nuevo símbolo en cada una de las celdas barridas por sus cabezas de lectura/escritura.

Mueve cada una de sus cabezas hacia la izquierda o hacia la derecha (de forma independiente al resto de las cabezas).

Por tanto, la función de transición para una máquina de Turing con n cintas, es de la forma $\delta: Q \times \Gamma^n \rightarrow Q \times \Gamma^n \times \{R, L\}^n$ donde una transición de la forma $\delta(q, (\sigma_1, \sigma_2, \dots, \sigma_n)) = (p, (\tau_1, \tau_2, \dots, \tau_n), (X_1, X_2, \dots, X_n))$ significa que cambia del estado q a p , reemplaza σ_i por τ_i en la cinta i y mueve la cabeza de la cinta i en la dirección X_i .

Máquina de Turing Multidimensional.

La máquina de Turing multidimensional es aquella que permite que la cinta tenga muchas dimensiones. Por ejemplo, una cinta de dos dimensiones que se extienda hacia abajo y hacia arriba, al igual que hacia la derecha y hacia la izquierda. Dependiendo del estado actual de la máquina de Turing y del símbolo analizado, cambia de estado, escribe un símbolo en la celda actual y

se mueve a la izquierda, al derecha, hacia arriba o hacia abajo. Por tanto, la función de transición para esta máquina de Turing será de la forma:

$$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, L, U, D\}$$

Una máquina de Turing multidimensional simula una máquina de Turing estándar. Simplemente realizando todas sus computaciones en una única dimensión. Una máquina de Turing estándar también puede simular una máquina de Turing multidimensional y, por tanto, la complejidad y la flexibilidad adicional que se debe a la múltiple dimensión, no es una capacidad real. Para simular una máquina de Turing de dos dimensiones mediante una máquina de Turing estándar, primero se asociara una dirección a todas las celdas de la cinta. Una forma de hacerlo es fijar, de forma arbitraria, un lugar en la cinta a partir del cual se asignarán las coordenadas a las celdas de la misma forma que se realiza en un plano de coordenadas.

Entonces, se usara una cinta de dos pistas para simular la máquina de Turing. Una pista se encargará de almacenar el contenido de las celdas y la otra las coordenadas, utilizando un símbolo (*) para separar los valores de las coordenadas. Para simular un movimiento de una máquina de Turing de dos dimensiones, esta máquina calcula la dirección de la celda a la que se moverá la máquina de Turing dos dimensiones.

Máquina de Turing determinista.

Una máquina de Turing determinista utiliza la misma filosofía que AF y AP por lo cual se puede decir que una MT es determinista si existe una y solo una producción del tipo (q_0, α) . Es decir existe solo un camino (flujo) estando en q_0 , con α .

Máquina de Turing no determinista.

Una MTND es un dispositivo con un control finito y una sola cinta infinita de una dirección. Para un estado dado y un símbolo de cinta barrido por la cabeza, la máquina tiene un número finito de alternativas para el siguiente movimiento, cada alternativa consiste en un nuevo estado, un símbolo en la cinta para imprimir y una dirección del movimiento de la cabeza, una máquina de Turing no determinista no se le está permitido hacer un movimiento en el cual el siguiente estado sea seleccionado de una alternativa, y el símbolo impreso y/ o la dirección del movimiento de la cabeza sean tomados de las otras alternativas.

La MTND acepta su entrada, si cualquier sucesión de alternativas de movimiento lleva a un estado de aceptación, si L es aceptado por una MTND M1, entonces L es aceptado por alguna MTD M2.

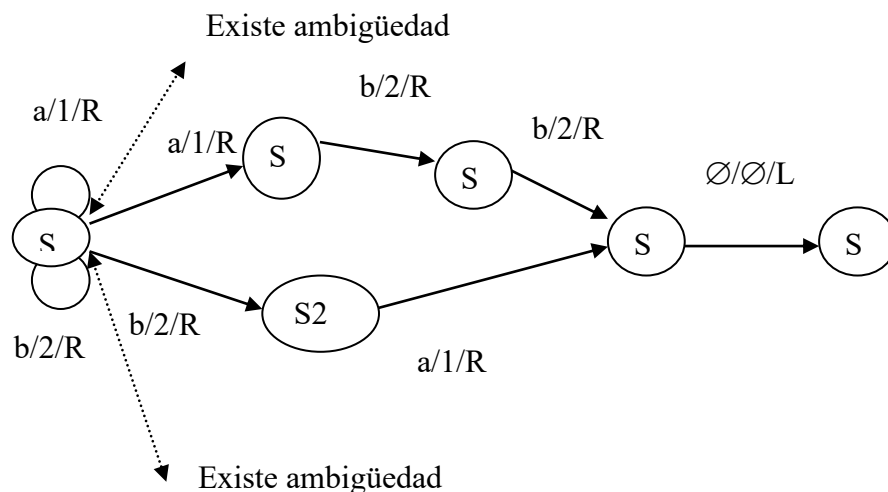
Para cualesquier estado y símbolo de cinta de M1, existe un número finito de alternativas para el movimiento siguiente. Tales alternativas pueden numerarse 1,2...r, sea r el número máximo de alternativas para cualquier par de símbolos de estados de cinta. Entonces cualquier secuencia finita de alternativas puede ser representada por una sucesión finita de alternativas puede ser representado por una sucesión de dígitos del 1 hasta r. No todas estas secuencias pueden representar alternativas de movimientos ya que, en algunas situaciones, pueden ser menores en número que r.

Donde M2 tendrán tres cintas, de las cuales una contendrá a la entrada, la segunda generará sucesiones de los dígitos de 1 al r, de una forma sistemática específicamente, las secuencias serán generadas con las más cortas al principio. Las sucesiones de igual longitud son generadas en orden numérico. Para cada sucesión generado en la cinta 2 M2, copia la entrada en la cinta 3 y entonces simula a M1 en la cinta 3, utilizando la secuencia de la cinta 2 para dictar los movimientos de M1. Si M1 accesa un estado de aceptación, M2 también acepta, si existe una sucesión de alternativas que llevan a la aceptación, está en algún momento, puede ser generada en la cinta 2.

Cuando sea simulado, M2 aceptara pero si ninguna sucesión de alternativas de movimientos de M1 conduce la aceptación de M2.

A continuación se presenta una máquina de Turing en sus dos aspectos (determinista y no determinista) con la siguiente expresión.

Expresión: $(a/b)^*abb/ba$



$Q = \{ S_0, S_1, S_2, S_3, S_4, S_5, S_6 \}$

$T = \{ 1, 2, \}$

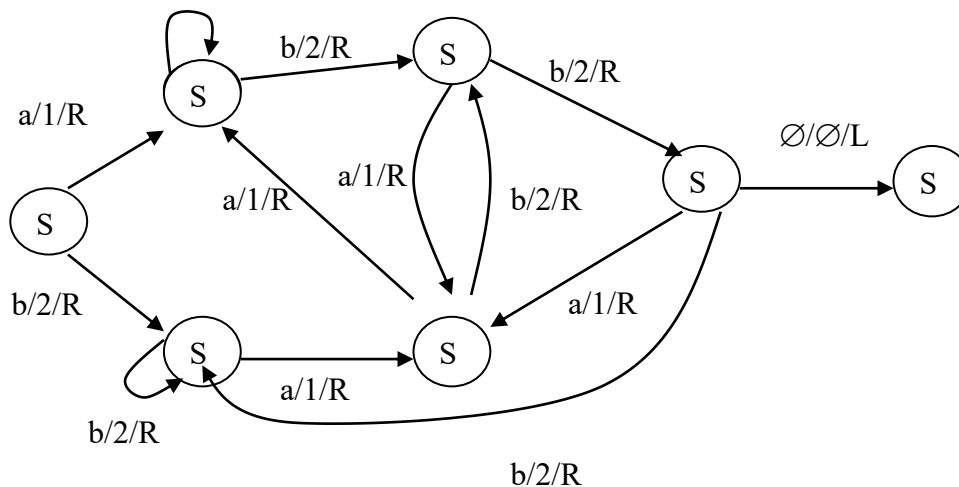
$\Sigma = \{ a, b, \}$

$s = S_0$

$b = \emptyset$

$F = S_5$

Expresión: $(a/b)^*abb/ba$



$Q = \{ S_0, S_1, S_2, S_3, S_4, S_5, S_6 \}$

$T = \{ 1, 2, \}$

$\Sigma = \{ a, b, \}$

$s = S_0$

$b = \emptyset$

$F = S_6$

4.5 PROBLEMAS DE HILBERT

Los problemas de Hilbert son una lista de 23 problemas matemáticos compilados por el matemático alemán David Hilbert para la conferencia en París del Congreso Internacional de Matemáticos de 1900. Los problemas estaban todos por resolver en aquel momento, y varios resultaron ser muy influyentes en la matemática del siglo XX. Hilbert presentó diez de los problemas (1, 2, 6, 7, 8, 13, 16, 19, 21 y 22) en la conferencia, en un acto el 8 de agosto en La Sorbona.

Aunque se han producido intentos de repetir el éxito de la lista de Hilbert, ningún otro conjunto tan variado de problemas o conjeturas ha tenido un

efecto comparable en el desarrollo del tema y obtenido una fracción importante de su celebridad.

Los veintitrés problemas de Hilbert son:

Problema	Explicación breve	Estado
1 ^{er}	La hipótesis del continuo (esto es, no existe conjunto cuyo tamaño esté estrictamente entre el de los enteros y el de los números reales)	Se ha probado la imposibilidad de probarlo como cierto o falso mediante los axiomas de Zermelo-Fraenkel. No hay consenso al respecto de considerar esto como solución al problema. ¹
2 ^o	Probar que los axiomas de la aritmética son consistentes (esto es, que la aritmética es un sistema formal que no supone una contradicción).	Parcialmente resuelto: hay quienes sostienen que se ha demostrado imposible de establecer en un sistema consistente, finitista y axiomático Sin embargo, Gentzen probó en 1936 que la consistencia de la aritmética se deriva del buen fundamento del ordinal ε_0 , un hecho sujeto a la intuición combinatoria.
3 ^{er}	Dados dos poliedros de igual volumen, ¿es siempre posible cortar el primero en una cantidad finita de piezas poliédricas que puedan ser ensambladas de modo que quede armado el segundo?	Resuelto. Resultado: no, probado usando invariantes de Dehn
4 ^o	Construir todas las métricas cuyas rectas sean geodésicas.	Demasiado vago para decidir si se ha resuelto o no.

5º	¿Son los grupos continuos grupos diferenciales de forma automática?	Resuelto por Andrew Gleason (1952)
6º	Axiomatizar toda la física	• La mecánica clásica: Hamel (1903). • La termodinámica: Caratheodory (1909). • La relatividad especial: Robb (1914) y Caratheodory (1924) independientemente. • La teoría de probabilidades: Kolgomorov (1930). • La teoría cuántica de campos: Wightman a finales de los años cincuenta del siglo XX.
7º	¿Es a^b trascendental, siendo $a \neq 0,1$ algebraico y b irracional algebraico?	Resuelto. Resultado: sí, ilustrado por el teorema de Gelfond o el teorema de Gelfond-Schneider
8º	La hipótesis de Riemann (la parte real de cualquier cero no trivial de la función zeta de Riemann es $\frac{1}{2}$) y la conjetura de Goldbach (cada número par mayor que 2 se puede escribir como la suma de dos números primos).	Abierto.
	Encontrar la ley más general del teorema de reciprocidad en cualquier cuerpo numérico algebraico	Parcialmente resuelto
10º	Encontrar un algoritmo que determine si una ecuación diofántica polinómica dada	Resuelto. Resultado: no, el teorema de Matiyasevich (1970) implica que no existe

	con coeficientes enteros tiene solución entera.	tal algoritmo.
11º	Resolver las formas cuadráticas con coeficientes numéricos algebraicos.	Parcialmente resuelto: • Sobre los números racionales: Hasse (1923-1924). • Sobre los números enteros: Siegel en los años treinta
12º	Extender el teorema de Kronecker sobre extensiones abelianas de los números racionales a cualquier cuerpo numérico de base.	Abierto
13º	Resolver todas las ecuaciones de 7º grado usando funciones de dos parámetros.	Resuelto negativamente por Vladímir Arnold y Andréi Kolmogórov en 1957.
14º	Probar la finitud de ciertos sistemas completos de funciones.	Resuelto. Resultado: no, en general, debido a un contraejemplo, Nagata (1962).
15º	Fundamento riguroso del cálculo enumerativo de Schubert.	Parcialmente resuelto, Van der Waerden a finales de los años treinta.
16º	Topología de las curvas y superficies algebraicas.	Abierto
17º	Expresión de una función definida racional como cociente de sumas de cuadrados	Resuelto. Resultado: se estableció un límite superior para el número de términos cuadrados necesarios, Pfister (1967). La solución negativa en general se debe a Du Bois (1967).
18º	¿Existe un poliedro irregular y que construya otros poliedros? ¿Cual es el apilamiento compacto más denso?	Resuelto

19º	¿Son siempre analíticas las soluciones de los Lagrangianos?	Resuelto por Bernstein (1904). Resultado: sí
20º	¿Tienen solución todos los problemas variacionales con ciertas condiciones de contorno?	Resuelto. Ha supuesto un área importante de investigación durante el siglo XX, culminando con las soluciones al caso no lineal.
21º	Probar la existencia de ecuaciones lineales diferenciales que tengan un grupo monodrómico prescrito	Resuelto. Resultado: sí o no, dependiendo de una formulación más exacta del problema. Según Gray resuelto de forma negativa por Anosov y Bolibruch (1994).
22º	Uniformización de las relaciones analíticas por medio de funciones automórficas	Resuelto por Koebe (1907) y Poincaré independientemente (1907).
23º	Extensión de los métodos del cálculo de variaciones	Resuelto

EJERCICIOS PROPUESTOS

Realice MTD.

Todas las cadenas que empiezan en “xyy” y no terminan en “yy”

$E \rightarrow E * E \mid (E) \mid E \mid a$ Donde el $*$ es el operador de la multiplicación.
(El ejemplo anterior es una gramática con alternaciones)

Expresión no regular $a^i b a^j b$ donde $i < j$

$$\begin{aligned} S &\rightarrow zPzQz \\ P &\rightarrow xPx \mid Q \\ Q &\rightarrow yPy \mid \lambda \quad T \{ x, y, Z \} \end{aligned}$$

$L = \{ ww \mid w \text{ se encuentra } (0|1)^* \}$

Diseñe una MT modular que acepte los siguientes lenguajes:

$L = \{ w \mid w \in \{0,1\}^n, 0^n 1^n n \geq 1 \}$

$L = \{ w \mid w \in \{ (,) \}^n, w \text{ Se abre un número de paréntesis y se cierra el mismo número de paréntesis} \}$

$L = \{ w \mid w \in \{a,b\}^n, a^i b^j \text{ con } i \leq j \}$

$L = \{ w \mid w \in \{a,b\}^n, a^i c^j b^i \text{ con } i, j \geq 0 \}$

$L = \{ w \mid w \in \{a,b\}^n, a^i b^j c^k \text{ con } i+k = j \}$

$L = \{ w \mid w \in \{a,b\}^n, w \text{ tiene } |a|=|b| \}$

$L = \{ w \mid w \in \{a,b\}^n, w \text{ tiene } 2|a|=|b| \}$

$L = \{ w \mid w \in \{a,b\}^n, a^n b^m \text{ con } n \leq m \leq 2n \}$

$L = \{ w \mid w \in \{a,b\}^n, \text{ tal que } wcw^{-1} \}$

$L = \{ w \mid w \in \{a,b\}^n, \text{ tal que } ww^{-1} \}$

De los siguientes AP construya MT

1.

$$\begin{array}{ll} (q_1, (, z) = (q_2, (z) & (q_2, (, () = (q_2, (() \\ (q_2,), () = (q_3,)() & (q_3,),)) = (q_3,))) \\ (q_3, (,)) = (q_2, ()) & (q_3 \lambda,)) = (q_4,)) \end{array}$$

2.

$$\begin{array}{ll} (q_0, x, \text{tope}) = (q_1, x \text{ tope}) & (q_1, x, x) = (q_1, x x) \\ (q_1, y, x) = (q_2, y x) & (q_2, y, y) = (q_2, y y) \\ (q_0, w, \text{tope}) = (q_3, w \text{ tope}) & (q_3, w, w) = (q_3, w w) \\ (q_3, z, w) = (q_4, z w) & (q_4, z, z) = (q_4, z z) \\ (q_3, x, w) = (q_1, x w) & (q_1, w, x) = (q_3, w x) \\ (q_4, y, z) = (q_2, y z) & (q_2, z, y) = (q_4, z y) \\ (q_2, \lambda, y) = (q_5, y) & (q_4, \lambda, z) = (q_5, z) \end{array}$$

3.

$$\begin{array}{ll} (q_1, \lambda, z) = (q_4, z) & (q_1, 1, z) = (q_1, 1z) \\ (q_1, 1, 1) = (q_1, 1 1) & (q_0, 0, 1) = (q_2, 01) \\ (q_1, 0, z) = (q_2, 0 z) & (q_2, 1, 0) = (q_2, 1 0) \\ (q_2, 1, 1) = (q_2, 1 1) & (q_2, 0, 0) = (q_3, 0 0) \\ (q_2, 0, 1) = (q_3, 0 1) & (q_3, 1, 0) = (q_3, 1 0) \\ (q_3, 1, 1) = (q_3, 1 1) & (q_3, 0, 1) = (q_2, 0 1) \\ (q_3, 0, 0) = (q_2, 0 0) & (q_3, \lambda, 0) = (q_4, 0) \\ (q_3, \lambda, 1) = (q_4, 1) & (q_4, \lambda, Z) = (q_5, 1) \end{array}$$

4.

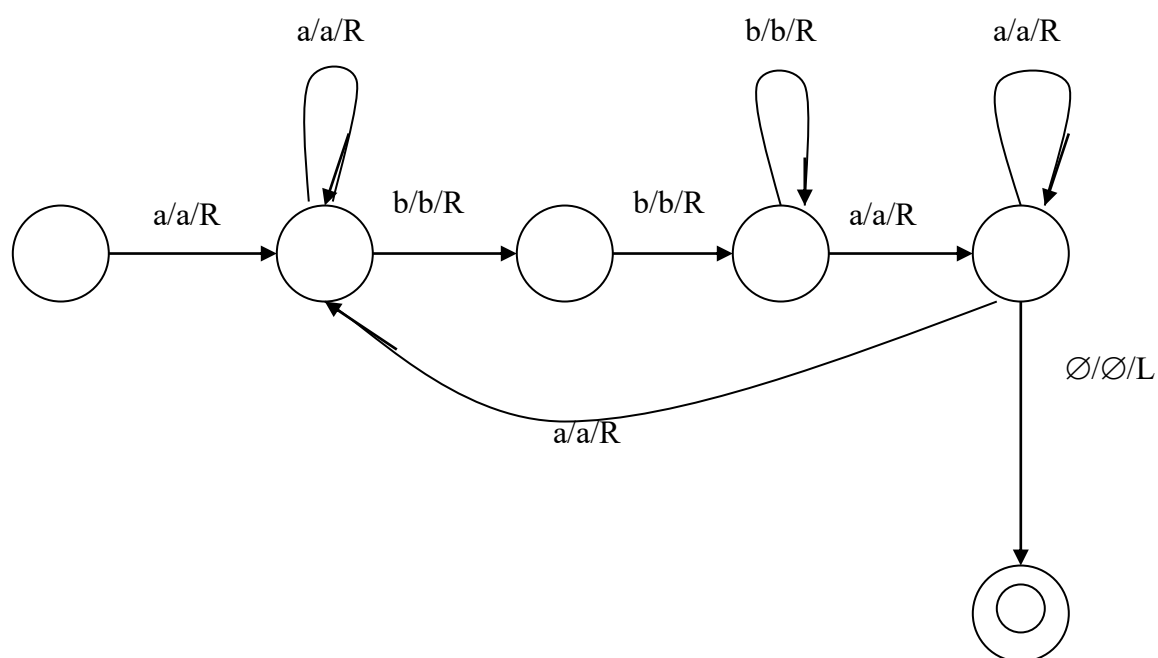
$$\begin{array}{ll} (q_0, a, z) = (q_6, a z) & (q_0, (, z) = (q_0, (z) \\ (q_0, (, () = (q_0, (() & (q_0, a, () = (q_1, a () \\ (q_1, ,, a) = (q_2, ,, a) & (q_2, a, ,) = (q_1, a ,) \\ (q_2, (, ,) = (q_0, (,) & (q_1,), a) = (q_4,) a) \\ (q_4,),)) = (q_4,))) & (q_4, \lambda, ,) = (q_5,)) \\ (q_6, \lambda, , a) = (q_5, a) & (q_4, \lambda,) = (q_2,)) \end{array}$$

5.

$$\begin{array}{ll} (q_0, a, z) = (q_1, a z) & (q_1, a, a) = (q_1, a a) \\ (q_1, b, a) = (q_2, b a) & (q_2, b, b) = (q_2, b b) \\ (q_2, c, b) = (q_2, \lambda) & (q_2, \lambda, z) = (q_3, z) \end{array}$$

$$\begin{array}{ll}
 (q_2, c, a) = (q_2, \lambda) & (q_2, c, b) = (q_3, \lambda) \\
 (q_3, c, b) = (q_3, \lambda) & (q_3, c, a) = (q_4, \lambda) \\
 (q_4, c, a) = (q_4, \lambda) & (q_4, \lambda, z) = (q_5, z)
 \end{array}$$

Obtenga la expresión regular de la siguiente MT.



CONSULTAS REALIZADAS

Alfred V. Aho, Monica S. Lam, Sethi Ravi y Jeffrey D. Ullman
Compiladores principios, técnicas y herramientas.
Editorial Addison-Wesley Iberoamericana.
2008.

Beck Leland
Software de sistemas. Introducción a la programación de sistemas.
Editorial Addison-Wesley Iberoamericana.
1988.

Jonh E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman
Introducción a la teoría de autómatas, Lenguajes y Computación.
Editorial PEARSON Addison Wesley.
2002.

Dean Kelley
Teoría de autómatas y lenguajes formales.
Editorial Prentice Hall
2006

Johnsonbaugh, Richard
Matemáticas discretas
Editorial Iberoamercana
2002.

Jean Paúl Trembay, Ram Manohar
Matemáticas discretas con aplicación a las ciencias de la computación
Editorial CECSA.
1996.

http://www.unimar.edu.ve/gonzalezalexis/descarga_historia_logica.htm

http://es.wikipedia.org/wiki/Jerarqu%C3%ADa_de_Chomsky

<http://emeny81.tripod.com/derechos.htm>

<http://www.monografias.com/trabajos16/automatas-y-gramaticas/automatas-y-gramaticas.shtml>

http://es.wikipedia.org/wiki/Problemas_de_Hilbert

<http://salonmaganize.com/artificial/75>

[http:// www-csli.stanford.edu/hp/Turing1.html](http://www-csli.stanford.edu/hp/Turing1.html)

[http:// www-csli.stanford.edu/hp/Turing2.html](http://www-csli.stanford.edu/hp/Turing2.html)

[http:// www-holtsotf.com/Turing1/essay.html](http://www-holtsotf.com/Turing1/essay.html)

<http://plato-csli.stanford.edu/hp/Turing1.html>

[http:// www.pc actual.com](http://www.pc-actual.com)

5. DECIBILIDAD

Competencias que deberán ser adquiridas

- Capacidad de investigación
- Autonomía en el aprendizaje
- Creatividad
- Capacidad de Análisis y síntesis
- Organización y Planificación
- Comunicación oral y escrita
- Utilización de las tecnologías de comunicación y de información (TIC'S)
- Trabajo en equipo

Objetivo de aprendizaje

Capacidad para distinguir problemas decidibles e indecidibles.

Resultado del aprendizaje	Actividad Educativa	Evaluación	Tiempo
Comprensión y capacidad de utilizar en un contexto adecuado conceptos de decibilidad, lenguajes recursivamente e numerables y recursivos	• Búsqueda de conceptos por equipos en los que podrá leer, descubrir, cuestionar, preguntar indagar. • Discusión en clase para expresarse, comunicar ideas, etc. • Creación de ideas originales obtenidas en consenso	• Informes por equipo y conclusiones grupales. • Técnicas de observación(registros, listas de control)	1 horas
Capacidad para demostrar que el problema de parada en un problema indecidible	• Búsqueda de conceptos por equipos en los que podrá leer, descubrir, cuestionar, preguntar indagar sobre el problema de parada • Discusión en clase para expresarse, comunicar ideas, etc.	• Informes por equipo y conclusiones grupales. • Técnicas de observación(registros, listas de control)	1 hora

5. DECIBILIDAD

5.1 LENGUAJES DECIDIBLES

Un problema de decisión es aquel formulado por una pregunta (referida a alguna propiedad) que requiere una respuesta de tipo “si/no”.

Un problema de decisión es:

- **Soluble** si existe un algoritmo total para determinar si la propiedad es verdadera (Existe una MT que siempre para al resolver el problema).
- **Parcialmente soluble** si existe un algoritmo parcial para determinar si la propiedad es verdadera (existe una MT que resuelve el problema, pero puede no parar).
- **Insoluble** si no existe un procedimiento efectivo para determinar si la propiedad es verdadera (no existe una MT),

Ejemplos de problemas de decisión:

¿Existe un algoritmo para decidir si un número natural cualquiera es par?

Si es soluble.

¿Existe un algoritmo para decidir si dos autómatas finitos cualesquiera son equivalentes?

Si es un problema soluble.

¿Es un número entero dado primo? Una instancia de este problema sería: ¿Es 17 primo?

Si es un problema soluble

¿Existe un algoritmo para determinar si una gramática es ambigua o no?

Es insoluble.

Muchos problemas insolubles son paradójicos en su naturaleza. Ejemplo: la paradoja de Russell.

Un peluquero afeita a todas las personas que no se afectan a sí mismas. El peluquero: ¿se afeita así mismo? (insoluble).

En teoría de la computación, un problema es un conjunto de frases de longitud finita que tienen asociadas frases resultantes también de longitud finita. Un problema de decisión es un problema en donde las respuestas posibles son SI o NO.

Un problema de decisión también se puede formalizar como el problema de decidir si una cierta frase pertenece a un conjunto dado de frases, también

llamado lenguaje formal. El conjunto contiene exactamente las frases para las cuales la respuesta a la pregunta es positiva.

Si existe un algoritmo que pueda decidir para cada posible frase de entrada si esa frase pertenece al lenguaje, entonces se dice que el problema es decidible, de otra forma se dice que es un problema indecidible. Cuando existe un algoritmo que puede responder positivamente cuando la frase está en el lenguaje, pero que corre indefinidamente cuando la frase no pertenece al lenguaje se dice que el problema es parcialmente decidible. En Teoría de la computabilidad, se estudia cuales lenguajes son decidibles con diferentes tipos de máquinas. En teoría de la complejidad computacional, estudia cuantos recursos necesita un algoritmo decidible para ejecutar (recursos de tiempo, espacio, número de procesadores, tipo de máquina, etc.).

Esos son algunos ejemplos de problemas de decisión expresados como lenguajes:

- Las frases sobre el alfabeto $\{a, b\}$ que contienen alternadas las letras a y b.
- Las frases sobre el alfabeto $\{a, b, c\}$ que contienen igual número de letras a y b.
- Las frases que describen un grafo con aristas etiquetadas con números naturales que indican su longitud, dos vértices del grafo y un camino en el grafo que es el camino más corto entre esos dos vértices.
- Las frases que describen una máquina de Turing y una cinta de entrada para esta máquina tal que la máquina se para en un tiempo finito al procesar esa entrada.

Lenguaje decidible: es aquel lenguaje L para el cual existe una máquina de turing que puede aceptar cualquier cadena w que pertenece a L y rechazar cualquier cadena W que no pertenece a L .

Lenguaje aceptable: es aquel lenguaje L para el cual no existe ninguna máquina de Turing que puede aceptar cualquier cadena W que pertenece a L y rechazar cualquier cadena W que no pertenece a L .

Lenguaje recursivamente enumerable: Lenguajes que son aceptados por una máquina de Turing que siempre para al aceptar la cadena o rechazar la cadena por no estar en un estado de aceptación o iterar indefinidamente, también se les conoce como: lenguajes estructurado por frases, tipo 0, parcialmente decidible o Turing computable.

Lenguajes recursivos: lenguajes que son aceptados por una máquina de Turing que siempre para sobre alguna entrada, se les conoce también como lenguajes decidible o Turing decidible.

Una máquina de Turing puede parar con 3 formas posibles:

- 1.- M termina en un estado final $\rightarrow w \in L(M)$
- 2.- M termina en un estado no final $\rightarrow w \notin L(M)$
- 3.- M no termina $w \notin L(M)$

L es un lenguaje RECURSIVO si existe un MT que para ante cualquier entrada tal que:

Si $w \in L \rightarrow M$ termina en un estado final

Si $w \notin L \rightarrow M$ termina en un estado no final

L es un lenguaje RECURSIVAMENTE ENUMERABLE si existe un MT (M) tal que:

Si $w \in L \rightarrow M$ termina en un estado final

Si $w \notin L \rightarrow M$ termina en un estado no final o M no termina.

Los problemas se pueden clasificar desde el punto de vista de la teoría de computabilidad en resolubles y no resolubles. Los problemas resolubles son objeto de estudio de la teoría de complejidad computacional.

En el contexto de complejidad computacional, el interés está centrado en establecer una medida de la cantidad de recursos computacionales (en términos de tiempo y/o espacio) necesarios para resolver un determinado problema o equivalentemente reconocer un lenguaje.

Los problemas resolubles se subdividen en tratables e intratables:

Los problemas tratables son: Aquellos para los cuales existe un algoritmo eficiente que los resuelve.

Los intratables son: Aquellos para los cuales no se conoce (o tal vez no exista) un algoritmo eficiente que los resuelva

Hoy en día las máquinas resuelven problemas mediante algoritmos que tienen como máximo una complejidad o costo computacional polinómico, es decir, la relación entre el tamaño del problema y su tiempo de ejecución es polinómica.

Éstos son problemas agrupados en el conjunto P. Los problemas con costo factorial o combinatorio están agrupados en NP. Estos problemas no tienen una solución práctica, es decir, una máquina no puede resolverlos en un tiempo razonable.

5.2 EL PROBLEMA DE PARADA (HALTING)

El problema de parada para máquinas de Turing es un ejemplo de problema irresoluble más conocido. Fue además el primer problema que se demostró formalmente que no tenía solución. El concepto de problema indecidible o irresoluble se aplica a problemas de decisión, es decir, problemas a los que podemos decir si tienen solución o no. Dentro de estos problemas, existe un conjunto al que no le podemos asignar una respuesta, ni afirmativa ni negativa: no existe un algoritmo que nos permita determinar si el problema tiene solución. Una de las razones por la que es importante conocer que el problema de la parada no tiene solución, es que nos permite decidir si otros problemas son resolubles o no.

Definición. Sea M una máquina de Turing arbitraria con un alfabeto de entrada Σ . ¿Puede decidirse si la máquina M se detendrá con la entrada w ?

Solución. La respuesta a esta pregunta es negativa. No se puede determinar si una máquina de Turing se detiene con una entrada arbitraria.

Demostración. Para demostrarlo, supongamos que el problema de la parada tiene solución, es decir, supondremos que existe una máquina de Turing que es capaz de determinar si otra máquina de Turing para con una entrada determinada.

Consideremos una máquina de Turing P , que recibe como entrada una máquina de Turing M y una cadena w codificada en la cinta y una continuación de la otra (Mw), y que se encarga de ejecutar M sobre la cadena w . La máquina P parará y aceptará la entrada si M para con w , y parará y rechazará la entrada si M no para con w .

Modificamos la máquina P , creando una máquina P' equivalente. Esta máquina no parará si M para con w , y parará si M no para con w .

Ahora crearemos una máquina D , cuya función es la siguiente. Recibe una máquina M , la pasa por una máquina que se encarga de copiar la máquina M a continuación. Por lo tanto, a la salida de la máquina copia, la cinta contendrá MM (la codificación de la máquina repetida). A continuación, D coge este resultado y lo pasa a través de P' . Con esto intentamos decidir si la máquina M para con la entrada M . Es decir, si M para con la entrada M , entonces D no para, y si M no para con la entrada M , entonces D para.

Por último, tomaremos una máquina D (denominaremos SD), y le aplicaremos como entrada una máquina D . SD aplica como entrada a la máquina que recibe, la misma máquina. Por lo tanto, esta máquina en principio parará si D no para con entrada D , y no parará si D para con entrada D . Pero si SD no para y si D para con entrada D , sabiendo que $D=SD$, llegamos a una contradicción, por que aplicar D a SD debería dar como resultado lo mismo que aplicar D sobre D . Del mismo modo para el otro caso. Por lo tanto, el problema de la parada no tiene solución.

Otro problema que suele utilizarse como paradigma de los problemas de este tipo es el conocido como el “juego del castor laborioso” (busy beaver), bautizado así por Tibor Rado a principios de los años 60. Rado se preguntaba cuántos unos podrían hacérsele imprimir a una Máquina de Turing antes de que se detuviera. Específicamente, si una Máquina de Turing es capaz de adoptar n estados y comienza a trabajar sobre una cinta totalmente ocupada por ceros, ¿cuál es el número máximo de unos que puede imprimir sobre la cinta antes de detenerse? La respuesta se conoce para los casos en los que $n = 1$, $n = 2$; $n = 3$, y $n = 4$, pero no para $n = 5$, ni tampoco para valores superiores de n . En 1983 se celebró en Dortmund, Alemania, un concurso para ver quién lograba diseñar el más afanoso “castor” de cinco estados. A lo largo del año que precedió al concurso se compusieron programas destinados a generar Máquinas de Turing “de competición”, y se puso a punto el soporte físico necesario. En el curso de esos trabajos se descubrieron buen número de “castores” de extravagantes conductas.

5.3 DECIBILIDAD DE TEORIAS LOGICAS

Una teoría lógica (TL) se define a partir de un conjunto de enunciados dados llamados axiomas, unas reglas de inferencia y un esquema de derivación. A partir de los axiomas y aplicando las reglas de inferencia y el esquema de derivación se infieren los teoremas de la teoría. El conjunto de teoremas de la teoría forman un lenguaje formal.

Si es posible definir una máquina de Turing tal que reconozca el lenguaje de los teoremas, este lenguaje es decidible y la teoría también lo es en consecuencia. Dicho en otras palabras, si el conjunto de teoremas visto como un lenguaje es reconocido por una máquina de Turing, entonces la TL es decidible. Y viceversa.

Puede hablarse entonces de manera indistinta de teorías lógicas o de lenguajes decidibles, como aquellos para los que existe una máquina de Turing capaz de reconocerlos. Luego, la correspondencia entre la sintaxis de una teoría lógica (lenguaje formal) y el reconocimiento simbólico del mismo por parte de un autómata queda establecida.

Desde el punto de vista semántico, las interpretaciones de las cadenas del lenguaje se realizan ya sea por el intérprete ó bien por el compilador del lenguaje de programación en el cual se dan las instrucciones. Las cadenas que resultan en instrucciones realizadas por la computadora pueden considerarse interpretadas como verdaderas y por tanto tienen, al menos, un modelo de la Teoría Lógica formada por tales cadenas.

En particular, los axiomas se consideran teoremas de la teoría, los cuales se derivan aplicando cero veces las reglas de inferencia.

Se sabe que existen algoritmos para tejer un suéter, construir un modelo de aeroplano, preparar un pastel y ejecutar una sonata de Beethoven. Es conocido que las computadoras pueden controlar señales de tráfico, líneas de producción y plantas químicas. Pueden llevar las revisiones de vuelos en líneas aéreas, controlar robots y producir nominas. Existen algoritmos para preparar una taza de café, determinar cuál es el mayor de un conjunto de números, descubrir si un número es primo o no o no primo e imprimir el máximo común divisor de 2 números. Desde la época escolar se recuerda que existen algoritmos para sumar, resta, multiplicar y dividir números enteros y para calcular raíces cuadradas. Sin duda existen algoritmos para calcular

logaritmos, determinar la frecuencia de las palabras en un fragmento dado de texto y controlar un submarino nuclear. Existen trabajos que las computadoras no pueden realizar. Hay muchas cosas que una computadora no puede hacer. En realidad, el número de cosas que pueden calcularse o computarse es infinitesimal en comparación con el número de cosas que a uno le gustaría calcular, las computadoras no pueden hacer muchas cosas.

EJERCICIOS PROPUESTOS

- 1.- Implemente y compruebe con un programa que decida si un número natural cualquiera es par.
- 2.- Diseñe un algoritmo para decidir si dos autómatas finitos cualesquiera son equivalentes.
3. Implemente y compruebe que un número entero dado es primo, por ejemplo ¿Es 17 primo?
4. ¿Es posible implementar un algoritmo para determinar si una gramática es ambigua o no?
5. Explique con diferentes ejemplos, la paradoja de Russell.
- 6.- Diseñe un lenguaje que acepte identificadores que estén compuestos de 5 caracteres como máximo.
- 7.- Defina con un ejemplo el problema de parada
- 8.- En qué consiste la decibilidad de las teorías lógicas
- 9.- Diseñe un lenguaje que acepte teoremas lógicos
- 10.- Diseñe un lenguaje que acepte recetas para hacer pasteles de chocolate
- 11.- Investigue en qué consiste la cinta de Markov

CONSULTAS REALIZADAS

Dean Kelley

Teoría de autómatas y lenguajes formales.

Editorial Prentice may

1995

Ravi Sethi

Lenguajes de Programación Conceptos y Constructores.

Editorial: Addison – Wesley Iberoamericana

1989

John E. Hapcroft

Jeffrey D. Ullman

Introducción a la teoría de autómatas, lenguajes y computación.

Cecsa

Md. Davis y E.J Weyuker

Computability complexity and languages

Academia press

http://eduditec.tecvallarta.edu.mx/tutoriales/index.php?view=article&catid=86%3Ateoria&id=265%3Aunidad6&option=com_content&Itemid=54

<http://www.cs.uns.edu.ar/~se/fcc/.../15-TesisTuringChurch-2009-bn.pdf>

http://www.wiphala.net/courses/...I/.../class_31_complexity_analysis.ppt

http://es.wikipedia.org/wiki/Problema_de_decisi%C3%B3n

<http://www.mitecnologico.com/Main/LenguajesDecidibles>

http://www.diclib.com/cgi-bin/d1.cgi?l=esbase=es_wiki_10page=showidid=31453

<http://guillermo.wordpress.com/2008/02/04/algoritmos-y-maquinas-de-turing/>

<http://es.geocities.com/teoriadelacomputacion2385/unidad5.html>

6. REDUCIBILIDAD

Competencias que deberán ser adquiridas

- Capacidad de investigación
- Autonomía en el aprendizaje
- Creatividad
- Capacidad de Análisis y síntesis
- Organización y Planificación
- Comunicación oral y escrita
- Utilización de las tecnologías de comunicación y de información (TIC'S)
- Trabajo en equipo

Objetivo de aprendizaje

Capacidad para demostrar que un problema es insoluble.

Resultado del aprendizaje	Actividad Educativa	Evaluación	Tiempo
Capacidad de distinguir entre problemas solubles e insolubles	• Búsqueda de conceptos por equipos en los que podrá leer, descubrir, cuestionar, preguntar indagar. • Discusión en clase para expresarse, comunicar ideas, etc. • Creación de ideas originales obtenidas en consenso.	• Informes por equipo y conclusiones grupales. • Técnicas de observación(registros, listas de control)	1 horas
Aplicación de un problema simple insoluble.	• Búsqueda de problemas insolubles por equipos en los que podrá leer, descubrir, cuestionar, preguntar indagar. • Exposición en clase para expresarse, comunicar ideas, etc.	• Informes por equipo de problemas insolubles y conclusiones grupales.	1 hora
Funciones computables	• Búsqueda de conceptos por equipos en los que podrá leer, descubrir, cuestionar, preguntar indagar. • Realizar ejercicios sobre funciones computables	• Informes por equipo de funciones computables y conclusiones grupales. • Ejercicios sobre funciones computables.	2 horas

		• Prueba de respuesta con desarrollo.	
Reducibilidad de Turing	• Búsqueda de conceptos por equipos en los que podrá leer, descubrir, cuestionar, preguntar indagar. • Realizar ejercicios sobre reducibilidad de Turing	• Informes por equipo de funciones computables y conclusiones grupales. • Ejercicios sobre funciones computables. • Prueba de respuesta con desarrollo.	2 horas

6. REDUCIBILIDAD

6.1 PROBLEMAS INSOLUBLES PARA LA TEORÍA DE LENGUAJES

Se ha visto que los algoritmos son un concepto fundamental de la ciencia de la computación y se ha desarrollado cierto conocimiento interno sobre la forma en que pueden construirse. Se sabe que existen algoritmos para tejer un suéter, construir un modelo de aeroplano, preparar un pastel y ejecutar una sonata de Beethoven. Es conocido que las computadoras pueden controlar señales de tráfico, líneas de producción y plantas químicas. Pueden llevar las revisiones de vuelos en líneas aéreas, controlar robots y producir nominas. Existen algoritmos para preparar una taza de café, determinar cuál es el mayor de un conjunto de números, descubrir si un número es primo o no primo e imprimir el máximo común divisor de 2 números.

Desde la época escolar se recuerda que existen algoritmos para sumar, resta, multiplicar y dividir números enteros y para calcular raíces cuadradas. Sin duda existen algoritmos para calcular logaritmos, determinar la frecuencia de las palabras en un fragmento dado de texto y controlar un submarino nuclear. Existen trabajos que las computadoras no pueden realizar. Hay muchas cosas que una computadora no puede hacer. En realidad, el número de cosas que pueden calcularse o computarse es infinitesimal en comparación con el número de cosas que a uno le gustaría calcular, las computadoras no pueden hacer muchas cosas. Algunos ejemplos que no se pueden resolver por una computadora son:

Los que ya conocemos

- El problema de parada: Dado un programa p y una entrada x , ¿ p con entrada x para?
- Dados dos programas, ¿calculan lo mismo?
- Los que nos da el teorema de Rice:
 - Dado un programa p , ¿ p calcula una función inyectiva?
 - Dado un programa p , ¿ p para con alguna entrada?
 - Dadas 5 imágenes. ¿Cuál es la más llamativa?

Algunos parecidos

El “busy beaver”:

- Dado n natural, t natural ¿Es t el tiempo máximo que tardan los programas de longitud n con entrada 0?

Es decir,

$\forall p$ con $|p|=n$, $p(0)\uparrow$ ó $(p(0)\downarrow$ y tarda tiempo $\leq t$)

y existe un q con $|q|=n$ y $q(0)$ tarda tiempo t

Y por supuesto.

El anti-virus:

Dado p un programa y x una entrada ¿Ejecutar p con entrada x “infectará” a esta computadora y a otras?

Infectar= causar destrucción de información y/o que algunas aplicaciones no funcionen como deben y/o colapsar la computadora.

Problemas indecidibles famosos

El problema de correspondencia de Post

Dadas dos listas de palabras $x_1, x_2, \dots, x_k$ y $y_1, y_2, \dots, y_k$

¿Existen $a_1, a_2, \dots, a_n$ para los cuales $x_{a_1} x_{a_2} \dots x_{a_n} = y_{a_1} y_{a_2} \dots y_{a_n}$?

El problema de Post, si se tiene que:

$u_1 = aba$	$u_2 = bbb$	$u_3 = aab$	$u_4 = bb$
$v_1 = a$	$v_2 = aaa$	$v_3 = abab$	$v_4 = babba$

Ejemplo 1:

Datos

aba	bbb	aab	bb
a	aaa	abab	babba

El problema consiste en que la primera cadena sea igual a la segunda. Para cadena valida:

aba	bb	aab	aba
a	babba	abab	a

Ejemplo 2:

Datos

abb	bb	abb
a	aaa	abab

El problema de Post no se cumple ya que no es posible generar la cadena

El problema de Post

Ejemplo:

Datos

I	IPP	IS	M
PPI	I	I	M

Si se cumple la correspondencia de post para esta cadena:

M	IS	S	IS	S	IPP	I
M	I	SS	I	SS	I	PPI

El décimo problema de Hilbert

- Dada una ecuación con coeficientes enteros, ¿existe una solución entera?

- Ejemplos: x

$$2+y^2=13$$

$$2x-11=0$$

6.2 UN PROBLEMA SIMPLE INSOLUBLE.

Sea el problema Paccept el siguiente problema:

Paccept: ¿Existe un procedimiento efectivo capaz de determinar si una máquina de Turing acepta una determinada cadena α ? Otra manera de formular el problema sería:

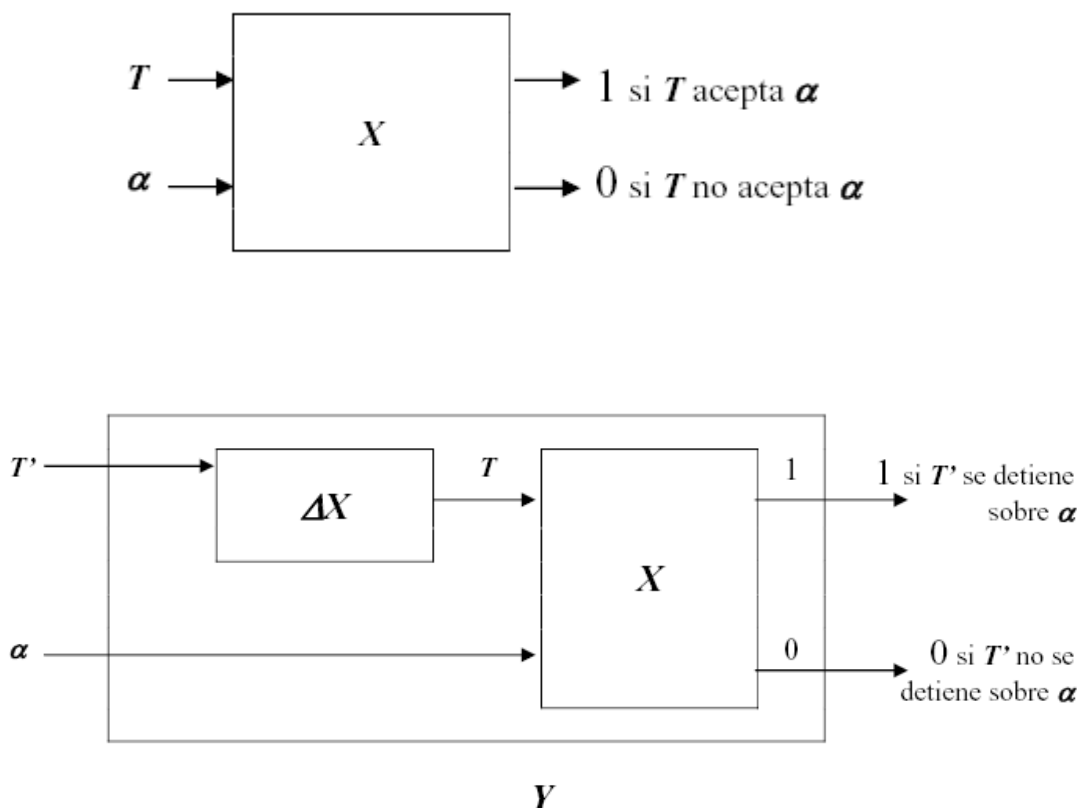
Paccept: ¿Existe un procedimiento efectivo capaz de determinar si una determinada cadena, α , pertenece al lenguaje que reconoce la máquina de Turing T?

Para demostrar que **Paccept** no es soluble, comenzaremos suponiendo que lo es, llegando de esta manera a un absurdo. Este absurdo tendrá lugar debido a que la solubilidad de **Paccept** implica la solubilidad de **Pdet**. Expresado de otra manera, demostramos que **Pdet** se reduce a Paccept.

Demostración:

Supongamos que **Paccept** es un problema de decisión soluble. Al ser un problema soluble, entonces existe un procedimiento efectivo (o máquina universal de Turing), **X**, que resuelve **Paccept**. **X** toma como datos de entrada la descripción de una máquina de Turing **T** y una cadena α y determina en un

tiempo finito si T acepta o no a la cadena α . Es decir X recibe como entrada al par (T, α) y retorna un 1 si T acepta α , mientras que devuelve la salida 0 si T no acepta α .



Combinando las máquinas X y ΔX , tenemos la máquina universal Y que tiene el siguiente comportamiento:

1. Y recibe como entrada el par (T', α) .
2. La máquina universal Y utiliza ΔX , la cual a partir de T' construye T .
3. El par (T, α) es ingresado a la máquina universal X .
4. Si la respuesta de X es 1 entonces T acepta α pero entonces T' se detiene sobre α . Dado que T se comporta como T' solo que siempre que se detiene acepta, podemos afirmar que T' , se detiene sobre α y que por lo tanto Y emite un 1 .
5. Si la respuesta de X es 0 entonces T no acepta α pero entonces T' no se detiene sobre α . Dado que T tiene a todos sus estados como aceptados, significa que para no aceptar la cadena, la única posibilidad es que T no se haya detenido. Como T se comporta como T' , podemos afirmar que esta tampoco se detiene y por lo tanto Y emite un 0 , ya que T' no se detiene sobre α .

Conclusión:

- La máquina universal Y retorna 1 (T' se detiene sobre α) si X retorna 1 (T acepta α).
- La máquina universal Y retorna 0 (T' no se detiene sobre α) si X retorna 0 (T no acepta α).

Hemos mostrado como construir la máquina universal Y que resuelve el problema de la detención a partir de la máquina universal X que resuelve un problema que sabemos por hipótesis que el problema de la detención es un problema insoluble, por lo tanto la solución encontrada mediante la máquina universal Y no puede existir. Lo que implica que alguna de sus componentes no puede existir, es decir o bien X , o bien ΔX no existe. Como por construcción ΔX existe, luego X no.

Podemos entonces concluir que **Pacept** es un problema no-soluble.

LA CUADRATURA DEL CÍRCULO: UN PROBLEMA INSOLUBLE PERO DIVERTIDO.

Aunque hace tiempo que se sabe que la cuadratura con regla y compás es imposible (de hecho se ha convertido en el paradigma de problema insoluble) el buscar soluciones aproximadas resulta ser un interesante desafío de geometría recreativa.

Parece que en otro tiempo hubo personas que soñando con la indudable fama que les proporcionaría resolver este problema se ofuscaron peligrosamente en él. No se pretende aquí resucitar tan peligrosa enfermedad. Se trata solo de un juego que podría tener una cierta utilidad pedagógica. Y que, al menos hasta donde yo he explorado, requiere solo unos conocimientos más bien elementales de geometría. (Poco más que el teorema de Thales y, por supuesto el de Pitágoras)

Si se trata de explicar en qué consiste realmente el problema, resulta altamente instructivo proponer una construcción aproximada de la cuadratura del círculo de modo que se pueda experimentar con lápiz y papel en qué consiste tal problema.

Una construcción geométrica aproximada de la cuadratura del círculo con regla y compás con tales fines "pedagógicos" o simplemente recreativos debería cumplir los siguientes requisitos.

- 1) la **aproximación** de π debería ser la **mejor** posible
- 2) el **número de pasos** debería ser el **mínimo** posible
- 3) la construcción debería poder hacerse siguiendo la lógica de cualquier

problema: partir del dato ... para llegar a la solución, en este caso partir del **radio** del círculo (el **dato**) para llegar al lado del **cuadrado** (la **solución**).

Por ejemplo (ver Fig. 1) partiendo del radio OF (dato) de la circunferencia a cuadrar se haya su mitad (punto A) y luego la mitad de esta, es decir la cuarta parte del radio, de modo que se obtenga un segmento igual a $5/4$ del radio (segmento OB) y tomando como radio este segmento se traza una circunferencia con el mismo centro (O) de la circunferencia de partida: los puntos de corte de esta circunferencia con los ejes de coordenadas (C, D E y F) nos dan los cuatro vértices del cuadrado solución.

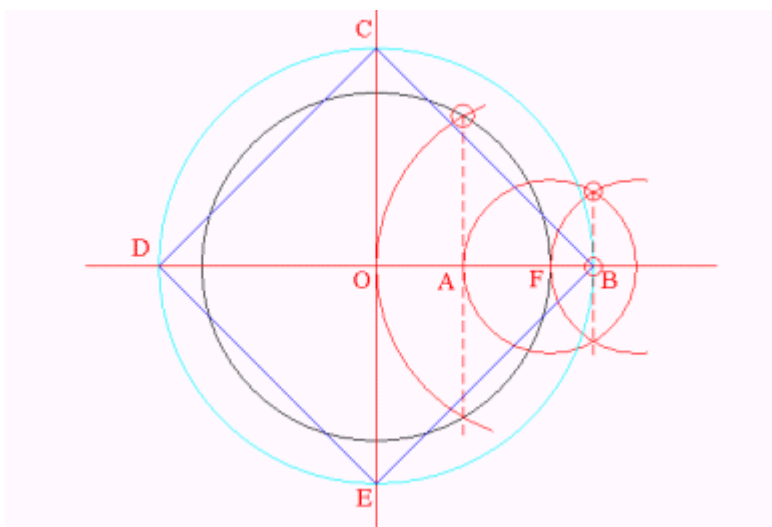


Fig. 1

6.3 FUNCIONES COMPUTABLES

El término función fue usado por primera vez en 1637 por el matemático francés René Descartes para designar una potencia x^n de la variable x . Aunque, el hecho de precisar el concepto de función se le atribuye al matemático suizo Leonhard Euler (1707-1783). Leonhard Euler realizó un estudio de todas las funciones elementales, incluyendo derivadas e integrales.

El matemático alemán Gottfried Wilhelm Leibniz utilizó el término función para referirse a varios aspectos de una curva, como es por ejemplo su pendiente.

En 1829, se define el uso más generalizado de las funciones por el matemático alemán, Lejeune-Dirichlet (1805-1859), quien escribió: "Una variable es un símbolo que representa un número dentro de un conjunto de ello. Dos variables X y Y están asociadas de tal forma que al asignar un valor a X entonces, por alguna regla o correspondencia, se asigna automáticamente un

valor a Y, se dice que Y es una **función** (unívoca) de X. La variable X, a la que se asignan libremente valores, se llama variable independiente, mientras que la variable Y, cuyos valores dependen de la X, se llama variables dependientes. Los valores permitidos de X constituyen el dominio de definición de la función y los valores que toma Y constituye su recorrido".

Definición de función

Una **función f de A en B** es una relación que le hace corresponder a cada elemento $x \in A$ uno y solo un elemento $y \in B$, llamado **imagen de x por f**, que se escribe $y = f(x)$. Esto se representa de la siguiente manera:

$f: A \rightarrow B$ Para que una relación de un conjunto A en otro B sea función, se deben cumplir dos condiciones:

La primera es que todo elemento del conjunto de partida A debe tener imagen.

Y la segunda, que la imagen de cada elemento $x \in A$ debe ser única, ningún elemento del dominio puede tener más de una imagen.

El conjunto formado por todos los elementos de B que son imagen de algún elemento del dominio se denomina **conjunto imagen o recorrido de f**.

Función recursiva

Las funciones recursivas son aquellas que se llaman a sí mismas, también se pueden llamar funciones resursivas-μ.

En la Teoría de la Computabilidad se demuestra que las **funciones recursivas** son las funciones que pueden ser calculadas con las máquinas de Turing.

Se puede construir la definición de función recursiva basándose en las funciones primitivas recursivas.

Ejemplos

Algunos ejemplos típicos de funciones recursivas pueden ser el cálculo del factorial de un número, el método de ordenación quick-sort, etc. Vamos a distinguir un **método iterativo** frente a un **método recursivo** con el ejemplo del cálculo del **factorial** de un número n.

FACTORIAL DE N: $n! = n \cdot (n-1) \cdot (n-2) \dots 2 \cdot 1$

Método **iterativo** para calcular el factorial de un número n dado:

```
int factorial(int n){
```

```

int resultado=1;
for(int i=1; i<=n; i++){
    resultado = resultado * i;
}
return resultado;
}

```

Método **recursivo** que realiza lo mismo que el código anterior pero de forma recursiva:

```

int factorial(int n){
    if(n==0){
        return 1;
    }
    return (n*factorial(n-1));
}

```

Vamos a ver cómo se construyen las **funciones recursivas primitivas**, definidas en $f:N^n \rightarrow N$.

Para definir las funciones recursivas primitivas, se va a utilizar la definición de función recursiva. Se utiliza un conjunto de funciones que por definición son recursivas, este conjunto inicial de funciones se denomina **base de la recursión**. Al aplicar un conjunto de reglas sobre las funciones recursivas primitivas, se obtienen nuevas funciones recursivas.

La **base de la recursión** está formada por las siguientes funciones:

- Función "**Nula**" o "**Cero**"
- Función "**Sucesor**"
- **Proyección i-ésima de orden n**

Las reglas que se aplican sobre estas funciones son las siguientes:

- **Composición**
- **Recursión primitiva**

A continuación, se van a definir estas funciones que forman la base de recursión, así como las reglas.

Base de recursión

Como se ha dicho anteriormente, hay tres funciones que forman la base de recursión. A continuación se explica cada una de ellas.

Función "Nula" o "Cero" $N(x) = 0$

La función que hace corresponder a cualquier número natural el número 0 es una función recursiva primitiva.

Función sucesor $S(x) = x'$

Esta función hace corresponder a cada número natural su sucesor, que denotamos por x' . Produce el siguiente número entero según los axiomas de Peano. Es una función recursiva primitiva.

Función proyección $I_i^n(x_1, x_2, \dots, x_n) = x_i$

Esta función es una función recursiva primitiva. A cada n-tupla se le hace corresponder su i-ésima componente, es decir, produce como resultado el argumento de la posición i.

Reglas

El conjunto de las funciones recursivas primitivas se define según las bases de recursión anteriores y las siguientes reglas:

Regla de composición

Dadas las funciones $g_i(x_1, x_2, \dots, x_n), (i = 1, 2, \dots, m)$ y la función f que depende de las m funciones anteriores, es decir $f = (g_1, g_2, \dots, g_m)$, entonces decimos que $h(x_1, x_2, \dots, x_n)$ es la función compuesta de las g_i mediante f si está definida de la siguiente forma:

$$h(x_1, x_2, \dots, x_n) = f(g_1(x_1, x_2, \dots, x_n), g_2(x_1, x_2, \dots, x_n), \dots, g_m(x_1, x_2, \dots, x_n))$$

Regla de recursión primitiva

Vamos a diferenciar las reglas de recursión primitivas con parámetros y sin parámetros.

Regla de recursión primitiva con parámetros

Dadas dos funciones recursivas conocidas: $f(x_1, x_2, \dots, x_n)$, $g(y, x_1, x_2, \dots, x_n, z)$, que dependen de los parámetros $x_1, x_2, \dots, x_n$, se define la función

$h(y, x_1, x_2, \dots, x_n)$, mediante el siguiente esquema de recursión:

$$h(0, x_1, x_2, \dots, x_n) = f(x_1, x_2, \dots, x_n)$$

$$h(y', x_1, x_2, \dots, x_n) = g(y, x_1, x_2, \dots, x_n, h(y, x_1, x_2, \dots, x_n))$$

Regla de recursión primitiva sin parámetros

Es un caso particular del anterior en el que no aparecerán los parámetros y el esquema de recursión sería:

$$h(0) = k$$

$$h(y') = g(y, h(y))$$

donde $g(y, z)$ es una función recursiva conocida y k es una constante (un número natural dado).

Las funciones definidas de esta manera son construidas a partir de un conjunto base y mediante procedimientos de cálculo sencillos y pasos sucesivos; por lo tanto, se pueden calcular efectivamente. Esta clase de funciones es la misma que la que se define utilizando las Máquinas de Turing.

Ejemplos

FUNCIÓN SUMA:

$$+ : \mathbb{N}^2 \rightarrow \mathbb{N}$$

$$+ (x, y) = x + y$$

FUNCIÓN PRODUCTO:

$$* : \mathbb{N}^2 \rightarrow \mathbb{N}$$

$$* (x, y) = x * y$$

FUNCIÓN EXPONENCIAL:

$$\exp : \mathbb{N}^2 \rightarrow \mathbb{N}$$

$$\exp(x, y) = 1 \text{ si } x = 0 \wedge y = 0$$

$$\exp(x, y) = x^y \text{ si e.c.o.c}$$

FUNCIÓN FACTORIAL:

$$fact:N^2 \rightarrow N$$

$$fact(x) = 1 \text{ si } x = 0$$

$$fact(x) = \prod j \text{ para } 1 \leq j \leq x \text{ si e.c.o.c}$$

6.4 REDUCIBILIDAD DE TURING

Se dice que un problema L1 se reduce en tiempo polinomial determinístico a otro problema L2, si asumiendo que existe un algoritmo A2 en P que resuelve L2 es posible construir un algoritmo A1 en P que resuelva L1.

Escribiremos L1 W L2 para significar que L1 se reduce a L2. Intuitiva: Una problema P1 se reduce polinomialmente a otro problema P2, si existe un algoritmo que transforme una instancia del problema P1 en una instancia del problema P2 en tiempo polinomial determinístico.

Ejemplo

Ordenar se reduce a encontrar el menor

Sabemos que existe Menor(i; j), que devuelve el elemento menor del segmento del arreglo A[i, j].

Ejemplo

PROCEDIMIENTO Ordena(A; n)

COMIENZA

PARA i =1 A n HAZ

j = Menor(i; n)

intercambia(i; j)

FINPARA

TERMINA

EJERCICIOS PROPUESTOS

1. Mostrar que cada una de las siguientes funciones es recursiva primitiva:

(a) $f(n,m) = n + m$

(b) $f(n,m) = n * m$

(c) $f(n) = n!$

(d) $f(n,m) = n^m$

(e) $a - b = a - b \quad \text{Si } a \geq b \quad \text{o} \quad 0 \quad \text{Si } a < b$

(f) $\max(n,m) = n \text{ si } n \geq m; \max(n,m) = m \text{ si } m < n$

2. Probar que cada una de las siguientes funciones son recursivas primitivas:

(a) $f(x) = (3 * x^2 + 5 * x) - 2$

(b) $f(x) = 10^x$

3. Diseñar una función recursiva primitiva que de resultado 1 si su argumento es un número par y, en caso contrario, devuelva 0. Usando esta última función, demostrar que la función siguiente es una función recursiva primitiva.

$$f(x) = x^2 \text{ si } x \text{ es par} \quad \text{o} \quad x + 1 \text{ si } x \text{ es impar}$$

4. Demostrar que el siguiente predicado es recursivo primitivo:

$$(2x + y)^{5z} < 10$$

5. Demostrar que las siguientes funciones son recursivas primitivas:

(a) $f(x) = 2 + (x \times 3)^5$

(b) $f(x) = 2 + (1 \times 3)^1 + (2 \times 3)^2 + \dots + (x \times 3)^x$

(c) $f(x) = 1^3 + 2^3 + 3^3 + \dots + x^3 \text{ si } x \geq 1 \text{ o } 0 \text{ en cualquier otro caso}$

6. Mostrar que la construcción if-then-else de los lenguajes de programación estructurados es definible como función recursiva primitiva.

FUENTES CONSULTADAS

Dean Kelley

Teoría de autómatas y lenguajes formales.

Editorial Prentice may

1995

Ravi Sethi

Lenguajes de Programación Conceptos y Constructores.

Editorial: Addison – Wesley Iberoamericana

1989

John E. Hapcroft

Jeffrey D. Ullman

Introducción a la teoría de autómatas, lenguajes y computación.

Cecsa

Md. Davis y E.J Weyuker

Computability complexity and languages

Academia press

<http://entucaramcfly.blogspot.com/2009/06/teoria-de-la-computacion-segunda.html>

http://campusvirtual.unex.es/calade/epistemowikia/index.php?title=Funciones_computables

http://reduc-turing.blogspot.com/2009_06_19_archive.html

<http://webdiis.unizar.es/~elvira/mac/ProblemasIndecidibles.pdf>

<http://www.cs.uns.edu.ar/~se/fcc/downloads/Clases%20Practicas/ProblemasInsolubles2009.pdf>

<http://webs.adam.es/rlllorens/picquad/picarta01.htm>

<http://www.mitecnologico.com/Main/DefinicionFuncion>

<http://www.monografias.com/trabajos7/mafu/mafu.shtml>

<http://hp.fciencias.unam.mx/~fhq/alfnotas.pdf>

<http://lazarroabel.tripod.com/>

http://drakox.iespana.es/ilp/MPU6_ILP.ppt