

INTRODUCCIÓN A LA TEORÍA DE AUTÓMATAS Y LENGUAJES FORMALES

Abelardo Gómez Andrade



Introducción a la teoría de autómatas y lenguajes formales

Se terminó de imprimir en agosto de 2014
en los talleres gráficos de Servicios Editoriales TRAUCO
Camino Real a Colima 285 Int. 56
Teléfono: (33) 32.71.33.33
Tlaquepaque, Jalisco.

ISBN : 607810460-8

C O N T E N I D O

PRÓLOGO	_____
COMENTARIOS PRELIMINARES	_____
CAPÍTULO 1: GRAMÁTICAS Y LENGUAJES FORMALES	_____
CONCEPTOS INTRODUCTORIOS	_____
CONCEPTOS IMPORTANTES	_____
NOMENCLATURA SIMPLIFICADA PARA CADENAS	_____
PREFIJOS, INFIJOS Y POSFIJOS (SUFIJOS)	_____
CONCATENACIÓN DE CADENAS	_____
OPERACIONES SOBRE LENGUAJES	_____
JERARQUÍA DE OPERADORES DE LENGUAJES	_____
DESCRIPCIÓN FORMAL DE LA GRAMÁTICA FORMAL	_____
GRAMÁTICA MAL ESTRUCTURADA	_____
DERIVACIÓN POR LA IZQUIERDA Y POR LA DERECHA	_____
GRAMÁTICAS CON LAMBDA O DECIDIBLES	_____
CARACTERIZACIÓN DE UNA GRAMÁTICA	_____
DISEÑO DE GRAMÁTICAS FORMALES	_____
GRAMÁTICAS EQUIVALENTES	_____
JERARQUÍA DE CHOMSKY	_____
GRAMÁTICA SIN RESTRICCIONES (TIPO 0)	_____
GRAMÁTICA SENSIBLE AL CONTEXTO (TIPO 1)	_____
GRAMÁTICA LIBRE DE CONTEXTO (TIPO 2)	_____
GRAMÁTICA REGULAR (TIPO 3)	_____
COMENTARIO FINAL SOBRE LOS TIPOS DE GRAMÁTICAS	_____
PROBLEMAS ADICIONALES	_____
ÁRBOLES DE DERIVACIÓN	_____
GRAMÁTICAS AMBIGUAS Y UNÍVOCAS	_____

- FORMA NORMAL DE CHOMSKY (CNF)**_____
- TRANSFORMACIÓN DE UNA GRAMÁTICA A LA CNF**_____
- CAPÍTULO 2: LAS GRAMÁTICAS FORMALES EN LA COMPUTACIÓN**_____
- FORMA NORMAL DE BACKUS - NAUR (BNF)**_____
- REGLAS DE LENGUAJES DE PROGRAMACIÓN PARA OPERADORES ASOCIATIVOS POR DERECHA E IZQUIERDA**_____
- APLICACIÓN DE LOS ANALIZADORES DE LENGUAJES**_____
- TEORÍA DE COMPILADORES**_____
- CASO DE ESTUDIO**_____
- CAPÍTULO 3: MÁQUINAS DE ESTADO FINITO**_____
- DEFINICIÓN FORMAL**_____
- DESCRIPCIÓN DE LA MÁQUINA DE ESTADO FINITO**_____
- EJEMPLOS DE APLICACIONES DE MÁQUINAS DE ESTADO FINITO**_____
- DIAGRAMA DE TRANSICIÓN DE LA MÁQUINA DE ESTADO FINITO**_____
- TABLA DE TRANSICIÓN DE LA MÁQUINA DE ESTADO FINITO**_____
- EJEMPLO: CONTADOR SÍNCRONO DE 3 BITS**_____
- CONCEPTOS ADICIONALES**_____
- CAPÍTULO 4: AUTÓMATAS DE ESTADO FINITO**_____
- CONCEPTOS GENERALES**_____
- DEFINICIÓN FORMAL DE AUTÓMATA DE ESTADO FINITO**_____
- ASPECTOS IMPORTANTES EN DISEÑO DE UN AUTÓMATA FINITO**_____
- AUTÓMATAS EQUIVALENTES**_____
- IMPLEMENTACIÓN DE UN AUTÓMATA FINITO**_____
- APLICACIONES DE LOS AUTÓMATAS DE ESTADO FINITO**_____
- SIMULACIÓN DE UN MODELO DEL MUNDO REAL
(MÁQUINA PARA VENTA DE REFRESCOS)**_____
- APLICACIONES EN CIRCUITOS DIGITALES SECUENCIALES
(RECONOCEDOR DE IMPARIDAD DE PULSOS)**_____
- APLICACIONES EN INTELIGENCIA ARTIFICIAL
(ESPACIOS DE ESTADOS)**_____
- APLICACIONES EN INTELIGENCIA ARTIFICIAL**_____

(REDES NEURONALES)_____

AUTÓMATA FINITO DETERMINISTA (AFD)_____

AUTÓMATA FINITO NO DETERMINISTA (AFND)_____

RELACIÓN ENTRE AUTÓMATAS Y GRAMÁTICAS_____

CONVERSIÓN DE UN AFND EN UN AFD_____

AUTÓMATAS PARA EXPRESIONES REGULARES_____

COMENTARIOS ADICIONALES_____

CAPÍTULO 5: AUTÓMATAS DE PILA_____

DESCRIPCIÓN GENERAL_____

DEFINICIÓN FORMAL DEL AUTÓMATA DE PILA_____

DISEÑO DE AUTÓMATAS DE PILA_____

REPRESENTACIÓN ALTERNA DE UN AUTÓMATA DE PILA_____

APÉNDICES_____

**1. EL PROBLEMA DE LA FALTA DE CREATIVIDAD EN EL ESTUDIANTE
DE MATEMÁTICAS**_____

**2. PROGRAMA VIGENTE DE LA ASIGNATURA CC209
(TEORÍA DE LA COMPUTACIÓN)**_____

3. PROBLEMAS PROPUESTOS PARA RESOLVER EN CLASE_____

PRÓLOGO

Un problema que se le presenta a un docente durante su práctica cotidiana es el hecho de tener que enfrentar los imprevistos que pueden surgir durante el transcurso de un ciclo regular y que le puede causar el incumplimiento para sus estudiantes de los objetivos previstos en el programa oficial de la asignatura.

En ocasiones los docentes asisten a sesiones de trabajo convocados por sus academias, acuden a cumplir con comisiones que les son asignadas, sufren de enfermedades imprevistas o de diversas eventualidades. Además existen sucesos inesperados como ocurrió en el año 2009 cuando la sociedad mexicana fue afectada por el embate de la epidemia de una enfermedad que se denominó AH1N1 y que ocasionó la suspensión de actividades durante algunas semanas en las instituciones educativas, además de otras organizaciones y empresas que resintieron los efectos.

Esos hechos llevan a un profesor a elaborar el presente texto oficial y elaborado a la medida para el curso de Teoría de la Computación que se imparte en el Centro Universitario de Ciencias Exactas e Ingenierías de la Universidad de Guadalajara. Cubre los temas que se estudian en las clases ordinarias de manera teórica y contiene varios problemas resueltos, para que el alumno con las bases adquiridas resuelva otros que aparecen propuestos en un apéndice al final del documento. No abarca los últimos dos módulos del programa vigente debido a que el estudiante hace una investigación final sobre los contenidos de ambos temas como parte de los productos de fin de ciclo escolar.

Un factor muy importante a considerar cuando se redacta este documento es el hecho de que la población estudiantil que cursa esta asignatura es muy heterogénea, pues hay alumnos que son demasiado competentes en el análisis lógico, mientras que otros tienen mayor dificultad para entender los conceptos relacionados con esa habilidad, siendo más frecuente este último caso. Por esa razón se ha decidido dar un enfoque básico a los asuntos tratados y se han explicado todos los conceptos con lujo de detalles.

De igual forma se determina que la manera como se desarrollan los temas será más orientada a los alumnos de las carreras de Tecnologías de la Información (Ingeniería en Computación y Licenciatura en Informática), pues representan la mayoría de quienes cursan la asignatura. Los alumnos de Licenciatura en Matemáticas son la minoría y eso hace que la forma como se imparten los conocimientos no sea tan formal y abstracta, por lo que se excluyen las demostraciones de ese tipo. No obstante, todos los alumnos observarán que este curso está totalmente encuadrado en el área del conocimiento de las Matemáticas Computacionales y que corresponde a las ciencias aplicadas, lo cual se demostrará con la elaboración de proyectos finales de implementación de programas basados en los modelos del curso.

Esperamos que el presente trabajo cumpla con el objetivo de apoyar al alumno de la asignatura de Teoría de la Computación y que sea amigable en su lectura y comprensión.

**Texto para el curso impartido en el CUCEI de la
Universidad de Guadalajara**

TEORÍA DE LA COMPUTACIÓN

ATENCIÓN: Se recomienda al alumno que lea detenidamente el programa de la materia incluido en el primer apéndice de este documento, ya que contiene información adicional demasiado importante, tal como los criterios de evaluación. Es mejor detenerse unos minutos y aclarar dudas en este momento, que tener problemas de interpretación al final del curso (¡¡¡y reprobarlo!!!).

COMENTARIOS PRELIMINARES

Aunque la asignatura de Matemáticas Discretas aparezca como único prerequisito de este curso, hay que advertir al alumno de que requiere tener un dominio sobre los temas que aprendió en algunos cursos anteriores o simultáneos, destacando sobre todo, en materias como Lógica y Conjuntos, Introducción a la Computación, Introducción a la Programación, Estructura de Datos, Lenguajes de Programación Comparados o Sistemas Digitales.

También es importante comentar que los conceptos que se habrán de adquirir en esta asignatura, le serán de utilidad al cursar materias, tanto de las que es prerequisito como Compiladores o Análisis y Diseño de Algoritmos, o bien de otras que no están incluidas en la Seriación.

Es necesario que el profesor de la asignatura les aclare a los alumnos que diversos conceptos se deben impartir bajo un criterio exclusivamente matemático, sin que se vea en ese momento cómo se aplicarán computacionalmente. Se debe pedir paciencia al estudiante, y aclararle que sin conocer el fundamento matemático, no podrá interpretar el significado de lo que se estará estudiando. El compromiso para el docente es hacer que el alumno lo pueda descubrir al avanzar en el ciclo escolar. *Los fundamentos matemáticos serán los cimientos del curso.*

Se recomienda hacer una sesión introductoria en la que se hable de los siguientes temas y se interactúe con los alumnos acerca de sus experiencias previas:

1. Relación de las Matemáticas con las Ciencias Computacionales.
2. Problemática de la enseñanza de la Matemática.
3. Los modelos matemáticos como una representación de la realidad.
4. Simulación y su importancia.
5. ¿Cuál es, en realidad, el cómputo avanzado?
6. ¿Es posible ser un profesional de la computación o de la informática sin dominar las matemáticas?

CAPÍTULO 1

GRAMÁTICAS Y LENGUAJES FORMALES

CONCEPTOS INTRODUCTORIOS

Considérense las siguientes definiciones iniciales como punto de partida para especificar los conceptos que habrán de estudiarse en este primer capítulo. Los términos son un poco abstractos y no debe preocuparse el lector si en este momento le parece que no es claro lo que se pretende expresar con ellos. Al avanzar en el curso se irán entendiendo mejor.

TEORÍA DE LOS LENGUAJES FORMALES: Estudio de los Lenguajes con una fundamentación matemática. Una rama importante de esta teoría se ocupa de la descripción finita de Lenguajes infinitos. Esta representación adopta la forma de un mecanismo abstracto para generar o reconocer cualquier cadena del Lenguaje (llamada Gramática). Esta rama se aplica a la sintaxis de los Lenguajes de Programación (en cuanto son distintos de su semántica, que requiere elementos de trabajo bastante diferentes). Así, el conjunto de todos los programas válidos de Java o de C, puede considerarse como un Lenguaje Formal sobre el Alfabeto de símbolos de cada uno de esos mismos Lenguajes de programación.

LENGUAJE FORMAL: Lenguaje con reglas explícitas y precisas para su sintaxis y semántica. Como ejemplo se pueden citar los Lenguajes de programación y también lógica como el cálculo de predicados. Así, los Lenguajes Formales se distinguen de los naturales tales como el castellano, cuyas reglas a medida que evolucionan con el uso dejan de ser una definición completa o precisa de la sintaxis del Lenguaje y mucho menos de su semántica.

GRAMÁTICA FORMAL: Es un esquema generativo para la representación finita de los Lenguajes, es decir, un solo modelo dinámico para generar palabras o cadenas de un lenguaje. Es la manera principal de especificar un Lenguaje Formal aunque sea infinito por medios finitos. Los Lenguajes pueden ser finitos o infinitos. El deseo de formalizar los Lenguajes naturales fue lo que llevó a Noam Chomsky a la iniciación de este tema en 1956.

Los conceptos anteriores son tomados de diccionarios de computación o de textos formales y expresan los conocimientos principales que se desarrollarán a lo largo de este capítulo. Sin embargo, se debe de iniciar a partir de definiciones más elementales, que posteriormente se irán complementando. Partiremos así de los conceptos iniciales básicos:

LENGUAJE FORMAL: Conjunto de PALABRAS, producidas en base a las **reglas** que conforman una Gramática Formal.

GRAMÁTICA FORMAL: Conjunto de REGLAS, empleadas para generar las palabras de un Lenguaje Formal.

Todo Lenguaje Formal estará vinculado siempre a una Gramática Formal, que especificará el mecanismo de producción de las palabras que lo conforman.

No existe un solo caso de un Lenguaje Formal que no sea producido por una Gramática Formal.

Se le llamará Lenguaje natural a todo aquel que empleamos los seres vivos para comunicarnos, destacando como ejemplo muy evidente el de cualquier idioma o dialecto. Sin embargo, se reconoce que este mecanismo no es útil para comunicarnos con una computadora, robot o Agente en general, por su característica de presentar muchas ambigüedades, que dichos equipos no pueden interpretar correctamente; sin embargo, sirven como modelo para la definición de los Lenguajes Formales.

Se tomará ahora una definición más explícita, que permitirá entender cómo se construye un Lenguaje Formal.

LENGUAJE FORMAL: Sea Σ un conjunto finito de elementos llamado Alfabeto Formal; un Lenguaje Formal L definido sobre Σ es un subconjunto de Σ^* (el conjunto de todos los arreglos o cadenas posibles de formar con los elementos de Σ), y determinado por las cadenas que cumplen con las reglas definidas por una Gramática Formal.

Se está mencionando un concepto ya conocido en términos naturales, que es el **ALFABETO**. En un enfoque FORMAL, se considera que debe ser el **conjunto finito de símbolos (no letras, ni caracteres necesariamente) que se emplean para formar las palabras**.

En términos naturales, un lingüista podría decir que sólo las letras pueden ser parte del Alfabeto del castellano, pero en términos formales también podrían tenerse otros símbolos; por ejemplo, como SR-71 es una palabra válida del castellano (es el nombre de un avión), entonces se considera que S, R, -, 7 y 1 son símbolos de nuestro Alfabeto, desde el punto de vista formal.

Ejemplos de posibles Alfabetos Formales válidos son:

$$\Sigma = \{ a, b, c, ch, d, e, f, g, h, \text{etc.} \} \quad \text{En idioma castellano.}$$

$$\Sigma = \{ .-, -..., -.-., ----, -.., ..-., --.,, ... \text{, etc.} \} \quad \text{En código Morse.}$$

$$\Sigma = \{ ., - \} \quad \text{En código Morse.}$$

$$\Sigma = \{ x \mid x \text{ es un carácter del código ASCII} \}$$

$$\Sigma = \{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 \}$$

$$\Sigma = \{ 0, 1 \} \quad \text{Para formar cadenas de bits.}$$

$$\Sigma = \{ +, -, *, /, \% \}$$

$$\Sigma = \{ a, e, i, o, u \}$$

Obsérvese que un símbolo de un Alfabeto podría estar formado por más de un carácter, como en el caso de la *ch*. Desde el punto de vista formal, dicha combinación de letras puede ser considerada perfectamente como un único símbolo, sobre todo tomando en cuenta que representa un solo fonema y no una combinación de éstos.

En términos computacionales, el Alfabeto del Lenguaje Formal integrado por los identificadores válidos en C estará formado por las letras, los dígitos y el guión bajo (_); el de los números enteros por todos los dígitos y los dos signos. El Alfabeto de los números flotantes se formaría con los dígitos, los dos signos y el punto decimal ¿Cuál sería el de los enteros hexadecimales sin signo, tanto en Lenguaje C como en ensamblador? ¿Y el de otros componentes léxicos comunes?

EJEMPLO:

Diseñar un Lenguaje Formal, a partir de ciertas reglas expresadas textualmente en castellano y con $\Sigma = \{x, y, z\}$. Las reglas se propondrán un poco más adelante, una vez que se definan todas las cadenas posibles de formar con los símbolos del Alfabeto. Se insiste en que el Lenguaje que se diseñará no sirve para algo útil, sino sólo para conocer un ejemplo concreto.

Es importante aclarar que por esta única ocasión se hará algo que es incorrecto, ya que en realidad para diseñar un Lenguaje Formal, se deberá emplear una Gramática Formal y no una natural, expresada en términos del castellano.

Para resolver este problema nos basaremos en la definición mencionada párrafos atrás, pero antes de evaluar Σ^* se hará una introducción a un nuevo concepto previo, que evitará divagar acerca de cuáles son todas las cadenas posibles de formar con los elementos del Alfabeto, puesto que son infinitas en su cantidad.

$\Sigma^n = \{s \mid s \text{ es toda cadena de longitud } n \text{ y que se pueda formar con los símbolos de } \Sigma\}$.

En este caso n debe tomar un valor específico de entre los números naturales, como 0, 1, 2, 3, 4, 5, 6, ... etc.

$\Sigma^0 = \{\lambda\}$ // Cadena vacía, de longitud 0. Puede ser representada también como ϵ ó Λ .

$\Sigma^1 = \Sigma = \{x, y, z\}$

$\Sigma^2 = \{xx, xy, xz, yx, yy, yz, zx, zy, zz\}$

$\Sigma^3 = \{xxx, xxy, xxz, xyx, xyy, xyz, xzx, xzy, xzz, yxx, yxy, yxz, \dots, zzx, zzy, zzz\}$

$\Sigma^4 = \{xxxx, xxxx, xxxz, xxyx, xxyy, xxyz, xxzx, xxzy, xxzz, \dots, zzzx, zyyy, zzzz\}$

... ... , etc.

Por lógica, se deduce la forma de relacionar los valores de las Σ^n para obtener Σ^* :

$\Sigma^* = \mathbf{U} \sum^n \mid n \in \mathbb{N}_0$, es decir, $\Sigma^* = \Sigma^0 \mathbf{U} \Sigma^1 \mathbf{U} \Sigma^2 \mathbf{U} \Sigma^3 \mathbf{U} \dots$

Σ^* se llama conjunto de todas las palabras posibles de formar sobre Σ .

Además, existe una variante muy parecida, pero que excluye a la cadena vacía λ :

$\Sigma^+ = \mathbf{U} \sum^n \mid n \in \mathbb{N}$, es decir, $\Sigma^+ = \Sigma^1 \mathbf{U} \Sigma^2 \mathbf{U} \Sigma^3 \mathbf{U} \Sigma^4 \mathbf{U} \dots$

Σ^+ se llama conjunto de todas las palabras posibles *no vacías* sobre Σ .

La forma de relacionar los dos conjuntos es: $\Sigma^* = \Sigma^+ \mathbf{U} \{\lambda\}$.

Conviene aclarar que $\mathbb{N}_0 = \{0, 1, 2, 3, 4, 5, \dots\}$, **mientras que** $\mathbb{N} = \{1, 2, 3, 4, 5, 6, \dots\}$. En este caso se consideran los dos criterios aceptados para los **números naturales**, según el autor consultado. Algunos consideran al cero como valor natural **por conveniencia**, pero otros no debido a que este número **históricamente** no fue descubierto junto con los demás valores aceptados en este conjunto. En Europa se conoció al cabo de varios años de iniciada nuestra era.

Determinar finalmente las palabras del Lenguaje Formal, en caso de que las reglas gramaticales (en forma natural o textual) fuesen, por ejemplo:

- **La cadena es de longitud menor o igual a tres.**
- **La cadena finaliza con x o con z.**
- **La cadena no inicia con y.**

Se debe recalcar que en este caso se está empleando una Gramática natural (no Formal) para no intentar asimilar tantos conceptos innovadores en forma simultánea. Posteriormente se analizará el concepto del Lenguaje Formal producido por una Gramática Formal, como debe ser. Habrá que recalcar que si no existen reglas gramaticales, no puede ser generado el Lenguaje.

El Lenguaje Formal resultante sería el siguiente:

$L = \{x, z, xx, xz, zx, zz, xxx, xxz, xyx, xyz, xzx, xzz, zxz, zyx, zyz, zzx, zzz\}$.

En la respuesta se observa que cada cadena definida cumple con la totalidad de las reglas señaladas. Éstas fueron determinadas en forma totalmente arbitraria, solamente para que el alumno conociera la estructura de un Lenguaje Formal, aunque sea completamente inútil para su aplicación.

Si hubieran sido otras las reglas gramaticales, el Lenguaje pudo haber sido, por ejemplo:

$L = \{\lambda, x, xy, yx, xz, zz, xyz, yyz, zxx, xxxx, xxxy, xyxz, zxyz, zzzyyz, \dots\}$

CONCEPTOS IMPORTANTES

Los elementos $a \in \Sigma$ se llaman **Símbolos del Alfabeto Σ** .

Los elementos $s \in \Sigma^n$ se llaman **Palabras, Cadenas o Arreglos sobre Σ** .

La longitud de una cadena se expresa como $| s |$ y consiste en la cantidad de símbolos que la forman (no la cantidad de caracteres).

La cadena de longitud cero se llama cadena vacía y se la puede representar de diversas formas a elegir, destacando como las más comunes λ , Λ ó ϵ . Es el elemento neutro en aritmética de cadenas y tiene mucha importancia en casos prácticos, aunque no recibe privilegios especiales y se la considera como a cualquier otra palabra, pudiendo pertenecer o no a un Lenguaje Formal.

Nótese que las cadenas de Σ^* se forman de la concatenación de ninguno, uno o muchos símbolos del Alfabeto (pudiendo repetirse los mismos). Para el caso de Σ^+ se tiene la concatenación de uno o muchos símbolos. Si el superíndice es n , entonces aparece esa misma cantidad de símbolos tomados del Alfabeto indicado en cada palabra del conjunto.

Los Lenguajes Formales pueden ser finitos o infinitos, según sea la cantidad de cadenas que contengan, aunque la gran mayoría son infinitos. El que se obtuvo en el ejemplo anterior resultó finito porque se estableció una longitud máxima para las palabras.

Los Lenguajes Formales se comportan siempre en la forma más parecida posible a los naturales, al tratar de ser un modelo que representa a estos últimos. Sin embargo serán muy diferentes en los casos en que convenga, por la aplicación que se les va a dar en términos de la programación y no de la comunicación humana.

NOMENCLATURA SIMPLIFICADA PARA CADENAS

Permite compactar la representación de las palabras con la indicación de un pequeño número como superíndice, que indica cuántas veces aparece o un símbolo o una secuencia de los mismos (indicada entre paréntesis). Por ejemplo:

$$\text{aaaaaaaaabb} = a^8b^2$$

$$\text{xxxxyzxxxx} = x^3y^2zx^2z^3,$$

$$\text{xyzxyzyyx} = xy(yz)^3y^2x = x(yz)^3y^3x,$$

$$a^0 = b^0 = \dots = z^0 = \lambda$$

El orden en el que aparecen los símbolos sí es importante. Por ejemplo $ab \neq ba$. Nótese que esta representación para cadenas para nada es compatible con los conceptos desde el enfoque algebraico, ya que por ejemplo $(ab)^3$ no representa a la palabra a^3b^3 sino a $ababab$.

En general, cualquier superíndice (como n , $+$, $*$) afecta al símbolo inmediato anterior únicamente, a menos que se encuentre una subcadena entre paréntesis, en cuyo caso afecta a toda ella como si fuera un solo símbolo.

PREFIJOS, INFIJOS Y POSFIJOS (SUFIJOS)

Considérese $S = uvw$ una cadena cualquiera. Entonces se tiene que para cualquier valor posible de u, v, w :

- a) u es llamado **prefijo**.
- b) v es llamado **infijo**.
- c) w es llamado **sufijo** o **posfijo**.

Aplicado a un ejemplo, sea la cadena $s = \text{COSMOS}$; entonces determinamos todos sus posibles prefijos, sufijos e infijos:

Prefijo: $\lambda, C, CO, COS, COSM, COSMO, COSMOS$

Sufijo: $\lambda, S, OS, MOS, SMOS, OSMOS, COSMOS$

Infijo: $\lambda, C,O,S,M,CO, OS, SM, MO, OS, COS, OSM, SMO, MOS, COSM, OSMO, SMOS, COSMO, OSMOS, COSMOS$.

Un prefijo es cualquier subcadena posible, incluso la vacía, que va al inicio de la cadena completa, mientras que el posfijo es similar, pero va al final. Observar como el infijo es la subdivisión más flexible, ya que puede ser, de hecho, cualquier subcadena posible, que esté ubicada en cualquier posición de la cadena completa.

En los Lenguajes naturales, un prefijo, infijo o posfijo deberá tener un significado por sí mismo; por ejemplo *inter* sí es prefijo, pero *internac* no lo es. En los Lenguajes Formales no se considera así y ahí no importa si tiene o no significado la subcadena.

CONCATENACIÓN DE CADENAS

Si $u = u_1 u_2 u_3 \dots u_m$ y $v = v_1 v_2 v_3 \dots v_n$ son cadenas de un Lenguaje Formal, entonces $u \bullet v = u v = u_1 u_2 \dots u_m v_1 v_2 \dots v_n$ y se la llama la concatenación de u con v .

Por ejemplo: Si $u = \text{dino}$ y $v = \text{saurio}$, entonces $uv = \text{dinosaurio}$.

Las cadenas a concatenar pertenecerán necesariamente al mismo conjunto Σ^* . El operador \bullet se puede omitir en la expresión.

La **concatenación** de cadenas **no es una multiplicación**, aunque lo parezca en su notación; de hecho la operación es no conmutativa ($uv \neq vu$) mientras que esta segunda operación sí lo sería.

OPERACIONES SOBRE LENGUAJES

Puesto que los Lenguajes Formales son conjuntos, se les pueden aplicar todas las operaciones de los mismos, tales como unión, intersección, concatenación, resta o complemento, por ejemplo; sin embargo como operaciones binarias solamente son de utilidad la unión y la concatenación para casos prácticos, las cuales se definen respectivamente de la siguiente manera:

$$L \cup M = \{ s \mid s \in L \vee s \in M \} \quad // \text{ Como se aprendió en cursos previos.}$$

$$L \bullet M = L M = \{ 1m \mid 1 \in L, m \in M \} // \text{ Producto cartesiano de palabras.}$$

EJEMPLO:

Sean $L_1 = \{ \lambda, a, ac, bca \}$, $L_2 = \{ \lambda, b, cb, ab \}$, evaluar:

a) $L_1 \cup L_2 = \{ \lambda, a, ac, bca, b, cb, ab \}$

b) $L_1 \bullet L_2 = \{ \lambda, b, cb, ab, a, \cancel{ab}, acb, aab, ac, acb, accb, acab, bca, bcab, bcacb, bcaab \}$

Se muestra tachada la cadena ab, al surgir por segunda vez en el inciso b), para indicar que aparecería nuevamente, pero como ya se había obtenido previamente, se deberá omitir en su segunda aparición.

La unión se puede utilizar cuando las palabras de un Lenguaje tienen diferentes estructuras, mismas que no se pueden representar por una única **expresión formal**. Por cierto, se le llama de esta forma a las expresiones que se manejan en este curso, las cuales no contienen mayoritariamente operadores aritméticos o lógicos, sino superíndices, uniones o concatenaciones.

En el caso de la concatenación de Lenguajes, se aplica esa misma operación a cada una de las palabras del primer Lenguaje contra cada una de las del segundo, respetando el orden de las palabras concatenadas.

No confundir las dos operaciones anteriores ya que son muy diferentes entre sí.

También se puede aplicar la intersección, misma que se lleva a cabo de acuerdo a como se aprendió en el curso de Teoría de Conjuntos. En el mismo ejemplo anterior se tiene que la operación daría como resultado la cadena vacía.

Además, se tienen las operaciones unarias o unitarias. En este caso se consideran las variantes de L, en forma muy similar a las que se tienen con los Alfabetos Σ . Considérese primeramente la descripción del contenido de los L^n de la siguiente forma:

$L^n = \{ s \mid s \text{ es toda cadena que se pueda formar concatenando } n \text{ palabras del Lenguaje } L \text{ (pudiendo repetirse las cadenas utilizadas)} \}$

La fórmula que permite encontrar todas las cadenas que conforman al resultado es:

$$L^n = \begin{cases} \{ \lambda \} & \text{si } n = 0 \\ L \bullet L^{n-1} & \text{si } n > 0 \end{cases}$$

Ya que se trata
de una relación
de recurrencia.

EJEMPLO:

Dado $L = \{ a, ac, ba, cba \}$, evaluar L^0, L^1, L^2 y L^3 .

$$L^0 = \{\lambda\} \qquad \qquad \qquad // \text{ Por definición.}$$

$$L^1 = L \bullet L^0 = L = \{ a, ac, ba, cba \}$$

$$L^2 = L \bullet L^1 = \{ aa, aac, aba, acba, aca, acac, acba, accba, baa, baac, baba, bacba, cbaa, cbaac, cbaba, cbacba \}$$

$$L^3 = L \bullet L^2 = \{aaa, aaac, aaba, aacba, aaca, aacac, aacba, \dots\}$$

acaa, acaac, acaba, acacba, acaca, acacac, ...

baaa, baaac, baaba, baacba, baaca, baacac, ...

cbaaa, cbaaac, cbaaba, cbaacba, cbaaca, }

... y así sucesivamente.

Al considerar que estos cálculos no son complicados en lo más mínimo, sino solamente laboriosos, se han omitido algunas cadenas y se han dejado puntos suspensivos en su lugar. Los últimos indican que pueden ser evaluados más elementos de la serie de L^n , tales como L^4 , L^5 , L^6 , etcétera.

Mientras que todas las cadenas de un determinado Σ^n son de la misma longitud, las de un mismo L^n son generalmente de diferentes tamaños.

A partir de los valores de L^n , surgen las siguientes variantes de L:

$L^* = \{L^n \mid n \in \mathbb{N}_0\} = L^0 \cup L^1 \cup L^2 \cup L^3 \dots$; se llama **Cerradura de Kleene de L**.

$L^+ = \{ L^n \mid n \in \mathbb{N} \} = \{ L^1, L^2, L^3, L^4, \dots \}$; se llama **Cerradura Positiva de L** .

Nótese la similaridad operativa y conceptual que existe con los operadores **n**, **+**, ***** al aplicarse a Alfabetos o a los Lenguajes.

Si hacemos una comparación de L^* con L^+ , se observa que pueden presentarse dos casos:

$$\mathbf{L}^* = \mathbf{L}^+ \quad \text{si } \lambda \in L, \text{ ya que } \lambda \text{ está incluida inclusive en } L^1.$$

$\mathbf{L}^* = \mathbf{L}^+ \mathbf{U} \{ \lambda \}$ si $\lambda \notin L$, ya que λ nunca aparecerá en los L^n con $n > 0$.

EJEMPLO:

¿Qué características tienen las cadenas de $L = \{ 1, 10 \}^*$?

Nótese que el operador asterisco aparece afuera de las llaves; eso indica que tal operador afectará a las dos cadenas contenidas ahí. El Lenguaje deberá constituirse con concatenaciones de una, ninguna o muchas veces la combinación de 1 y/o 10. De manera comparativa se podría simplificar la comprensión del problema, haciendo la analogía con la construcción de palabras, por un bebé, uniendo bloques marcados con 1 y/o 10.

Al hacer tales combinaciones, se notará que las cadenas formadas, iniciarán con 1 necesariamente (excepto la cadena vacía). Además, nunca podrá haber 0

consecutivos en alguna palabra de este Lenguaje, ya que este se puede presentar solo después de un 1.

Por cierto, ese mismo Lenguaje puede ser expresado como $L = \{(1, 10)^*\}$, pero no como $L = \{1^*, (10)^*\}$ ya que son expresiones que contienen palabras muy diferentes. En este segundo Lenguaje, se tendrían cadenas formadas por 1 exclusivamente o por únicamente secuencias de 10, *más no por combinaciones de ambos elementos*.

JERARQUÍA DE OPERADORES DE LENGUAJES

Si una expresión formal compuesta, tal como $L_1^* \bullet L_2 \cup L_3 \bullet L_4^*$, contiene varias indicaciones de operaciones válidas con Lenguajes Formales, éstas se efectuarán jerarquizadas en el siguiente orden:

- 1) Paréntesis 2) Superíndices 3) Concatenación 4) Unión.

Así pues, para evaluar una expresión como $\{a, b\} \cup \{b\} \bullet \{a, c\}$ se debe concatenar primero y posteriormente hacer la unión, de manera que el resultado sería $\{a, b, ba, bc\}$.

Se considera la asociatividad por la izquierda cuando se presentan varias operaciones similares en una misma expresión.

DESCRIPCIÓN FORMAL DE LA GRAMÁTICA FORMAL

Hasta el momento se han empleado reglas gramaticales, pero expresadas de manera natural, lo cual ha sido necesario para no introducir tantos elementos nuevos simultáneamente. A continuación, se planteará la definición formal de la Gramática, para después ir analizándola con más detalle.

A la Gramática Formal también se la conoce como **Sistema de Estructuración de Frases** y consiste en el siguiente cuarteto ordenado o 4-tupla:

- 1) Un conjunto finito **N** de símbolos no terminales.
- 2) Un conjunto finito **T** de símbolos terminales, en donde $N \cap T = \emptyset$.
- 3) Un subconjunto finito **P** de $((N \cup T)^* - T^*) \rightarrow (N \cup T)^*$ llamado conjunto de composiciones o reglas de producción.
- 4) Un símbolo inicial **S** $\in N$, también conocido como símbolo distinguido.

Se expresa en forma sintética como el cuarteto ordenado **G = (N, T, P, S)**. *Es muy importante aclarar que esta nomenclatura, al igual que en algunos otros casos de los temas tratados en este curso, se pueden tener algunas diferencias según el autor de la referencia que se consulta, ya sea de manera impresa o en internet.*

El conjunto P es el más importante en la definición de una Gramática, ya que es el que contiene propiamente las reglas gramaticales para producir

las palabras; dichas reglas consisten en toda una serie de composiciones (que son posibles sustituciones de una subcadena por otra), disponibles para utilizarse en el orden que se desee, y las veces que se requieran, de forma que el símbolo inicial se convierta en una palabra completa conteniendo únicamente símbolos terminales.

Las composiciones no son sustituciones bidireccionales.

Ningún símbolo puede ser terminal y no terminal a la vez. El símbolo inicial deberá estar incluido también en el conjunto N y ser único.

Se acostumbra utilizar las primeras letras mayúsculas para designar los no terminales cuando tales símbolos no serán los terminales, pero es sólo costumbre y no una regla; lo mismo ocurre al tomar A como símbolo inicial, aunque como S es la primera letra de Start en ejemplos tomados de libros en inglés se usa ésta en diversos ejemplos para el símbolo distinguido. Los terminales son aquellos que van a formar las palabras del Lenguaje.

EJEMPLO:

Diseñar una Gramática Formal cualquiera (en este caso sin alguna especificación concreta) y determinar algunas cadenas del Lenguaje que genera, considerando a $G = (N, T, P, S)$.

$$N = \{ A, B, C, D \}$$

$$T = \{ 0, 1 \}$$

$$S = \{ A \}$$

$$P = \{ A \rightarrow 0DA, \quad A \rightarrow BD1, \quad A \rightarrow C0,$$

$$B \rightarrow 1C1, \quad B \rightarrow 11,$$

$$C \rightarrow 10A1, \quad C \rightarrow 010, \quad C1 \rightarrow A,$$

$$D \rightarrow C0, \quad D \rightarrow 1, \quad D1 \rightarrow 10 \}$$

Como ejemplo de cadenas $s \in L(G)$ podríamos producir las siguientes:

$$\underline{A} \Rightarrow 0\underline{DA} \Rightarrow 01\underline{A} \Rightarrow 01\underline{C}0 \Rightarrow 010100 \quad , \text{ que es lo mismo que } (01)^2 0^2.$$

$$\underline{A} \Rightarrow \underline{BD1} \Rightarrow \underline{B}11 \Rightarrow 1\underline{C}111 \Rightarrow 1010111 \quad , \text{ que es lo mismo que } (10)^2 1^3.$$

$$\underline{A} \Rightarrow \underline{BD1} \Rightarrow 1\underline{C}1D1 \Rightarrow 110\underline{A}11D1 \Rightarrow 110\underline{C}011D1 \Rightarrow 110010011\underline{D1} \Rightarrow 11001001110, \quad \text{que es lo mismo que } 1(10^2)^2 1^3 0.$$

La flecha doble (\Rightarrow) indica que se ha utilizado una regla; el empleo de las mismas permite ir transformando lo que inicialmente es un símbolo inicial en una cadena terminada, conteniendo únicamente símbolos terminales.

Al producir la cadena se recomienda subrayar lo que constituye el lado izquierdo de la composición empleada, y que se transformará en lo que indica el lado derecho en el siguiente paso. Por ejemplo, al producir la primera de las cadenas, aparece en el tercer paso subrayada la **A**; es tal símbolo el que se transforma en el siguiente paso en **C0**. Esto permite al momento de revisar, saber más fácilmente qué regla ha sido empleada.

EJEMPLO:

Comprobar que una Gramática con $P = \{ A \rightarrow 11 \mid 1001 \mid A0 \mid AA \}$ produce las cadenas que representan los primeros seis enteros positivos mayores que 0 y que sean divisibles entre 3, pero todos ellos en notación binaria. El símbolo inicial evidentemente es A .

$\underline{A} \Rightarrow 11$	// 3 en binario.
$\underline{A} \Rightarrow \underline{A}0 \Rightarrow 110$	// 6 en binario.
$\underline{A} \Rightarrow 1001$	// 9 en binario.
$\underline{A} \Rightarrow \underline{A}0 \Rightarrow \underline{A}00 \Rightarrow 1100$	// 12 en binario.
$\underline{A} \Rightarrow \underline{A}A \Rightarrow \underline{A}11 \Rightarrow 1111$	// 15 en binario.
$\underline{A} \Rightarrow \underline{A}0 \Rightarrow 10010$	// 18 en binario.

¿Produce esta misma Gramática los siguientes cinco múltiplos de 3 en binario?

REQUERIMIENTOS IMPORTANTES PARA DISEÑAR COMPOSICIONES VÁLIDAS.

Las reglas de producción deberán:

- a) Estar formadas exclusivamente con los elementos de $N \cup T$.
- b) Tener en su lado izquierdo al menos un símbolo no terminal.

Existen dos criterios distintos para relacionar al Alfabeto Σ con los elementos definidos en la Gramática Formal G:

$N \cup T \Rightarrow$ **Alfabeto de la Gramática** : Σ_G (Símbolos empleados en las reglas).

$T \Rightarrow$ **Alfabeto del Lenguaje** : Σ_L (Símbolos que forman las palabras).

EJEMPLO:

Explicar por qué razón en la definición del diseño de las composiciones se dice que éstas son de la forma $((N \cup T)^* - T^*) \rightarrow (N \cup T)^*$.

- El lado izquierdo de las composiciones deberá estar contenido en el conjunto $((N \cup T)^* - T^*)$.
- Además, el lado derecho de las mismas deberá ser una cadena de $(N \cup T)^*$.

Considerar los criterios anteriores en un ejemplo:

Suponer que se diseña una Gramática con $N = \{ X, Y \}$ y con $T = \{ a, b, c \}$. Determinar los valores posibles de las cadenas que pudieran tenerse en cada uno de los dos lados de una composición válida para esta Gramática.

$$N \cup T = \{ X, Y, a, b, c \}$$

$(N \cup T)^* = \lambda, X, Y, a, b, c, XX, XY, Xa, \dots, cc, XXX, \dots, ccc, \dots$ (posible lado derecho).

$$T^* = \lambda, a, b, c, aa, ab, ac, ba, \dots, cc, aaa, \dots, ccc, aaaa, \dots$$

$(N \cup T)^* - T^* = X, Y, XX, XY, Xa, Xb, Xc, Ya, \dots$ (posible lado izquierdo).

NOTA: $(N \cup T)^* - T^*$ contiene todas las cadenas formadas con los elementos de $N \cup T$, pero con al menos un elemento del conjunto N.

GRAMÁTICA MAL ESTRUCTURADA

En ocasiones, cada una de las reglas de producción para una Gramática Formal se diseña en forma correcta, pero con ciertos “errores de Lógica” que surgen al integrarse todas ellas en un conjunto P. Entonces se presentan deficiencias evidentes que resaltan al analizar la Gramática en su totalidad.

EJEMPLO:

Describir los errores que hacen de la siguiente una Gramática mal estructurada.

$$G = (N, T, P, S)$$

$$N = \{ A, B, C, D \} \quad T = \{ 0, 1 \} \quad S = \{ A \}$$

$$P = \{ A \rightarrow B1, A \rightarrow BC, A \rightarrow 0A, C \rightarrow CA, C \rightarrow 11A, DAC \rightarrow 01, D \rightarrow 11, D \rightarrow \lambda \}$$

Aunque la anterior parece ser una Gramática válida, de acuerdo a los criterios señalados en el tema anterior, se aprecian tres errores muy evidentes:

- No existe regla alguna que especifique lo que produce B.
- Ninguna regla produce D.
- Verificar que no haya reglas inútiles (nunca utilizadas), como es el caso de la que contiene la combinación DAC, que nunca se presentará al derivar.

Todo símbolo no terminal debe aparecer por lo menos una vez en cada lado de al menos una de las reglas de producción (y en composiciones distintas); la excepción es el símbolo inicial, que puede no aparecer al lado derecho.

Puede haber una infinidad de motivos por los que la Gramática esté mal estructurada. La mejor manera de detectar cuándo se presenta este problema, será con el conocimiento de la forma correcta de construirlas y empleando el sentido común y la lógica.

CONCEPTOS IMPORTANTES PARA REPASAR

Dada una Gramática G, puede construirse un Lenguaje L(G), utilizando composiciones para derivar en las cadenas que constituyen el Lenguaje.

$$L(G) = \{ s \mid s \in T^* \wedge S \Rightarrow \dots \Rightarrow s \}$$

Una Gramática genera en la mayoría de los casos un Lenguaje con un número infinito de cadenas, pero **todas ellas de la misma estructura gramatical**. Este principio es muy importante para justificar el siguiente tema, en el que se vinculará a una Gramática Formal con el Lenguaje Formal que produce.

DERIVACIÓN:

Es la acción de sustituir en una palabra que contiene símbolos no terminales una subcadena por su equivalente según las reglas de producción partiendo del símbolo inicial y hasta llegar a una cadena con terminales solamente; se expresa como $A \Rightarrow B$.

COMPOSICIÓN O REGLA DE PRODUCCIÓN:

Se expresa como $A \rightarrow B$, donde $A \in (N \cup T)^*$ - T^* y $B \in (N \cup T)^*$, y en ellas se especifica qué es lo que se puede sustituir al construir una cadena de un Lenguaje.

DERIVACIÓN POR LA IZQUIERDA Y POR LA DERECHA

Sea G una **Gramática Libre de Contexto** y s una cadena de $L(G)$; se la llama derivación por la **izquierda** si al derivar para obtener una cadena, **se sustituye siempre primeramente el símbolo no terminal que esté más a la izquierda en cada paso del proceso de derivar**. Similarmente, la derivación por la **derecha** ocurre cuando **el símbolo que se deriva antes que los demás es el que se ubica más hacia la derecha**. Esto se observa solo cuando aparecen varios símbolos no terminales durante el proceso por lo que no se emplearía en una Gramática Regular.

La utilidad de estos dos conceptos se conocerá cuando se estudie el caso de los analizadores de sintaxis en un compilador.

GRAMÁTICAS CON LAMBDA O DECIDIBLES

Son aquellas en las cuales $\lambda \in L(G)$. Debe existir una derivación que nos lleve a la cadena vacía λ partiendo del símbolo inicial.

Una Gramática sin reglas lambda (que son aquellas que constan de λ al lado derecho de la regla) no puede ser decidible. Una con reglas lambda, puede ser o no ser decidible, según la estructura de las composiciones.

Caso general	$P = \{ A \rightarrow \lambda, \dots \}$	Con Reglas λ	Sí decidible.
	$P = \{ A \rightarrow B, B \rightarrow \lambda, \dots \}$	Con Reglas λ	Sí decidible.
	$P = \{ A \rightarrow Bx, B \rightarrow \lambda, \dots \}$	Con Reglas λ	No decidible.

EJEMPLOS:

$P = \{ A \rightarrow xAz, A \rightarrow B, B \rightarrow By, B \rightarrow \lambda \}$ con $S = \{ A \}$. Esta es una Gramática decidible.

$P = \{ A \rightarrow 1A1 \mid 01A \mid 01 \mid 0 \}$ con $S = \{ A \}$. Esta es una Gramática no decidible.

$P = \{ A \rightarrow aA, A \rightarrow bA, A \rightarrow bB, B \rightarrow a \}$ con $S = \{ A \}$. Esta evidentemente es una Gramática no decidible, ya que al igual que la anterior ni siquiera contiene reglas λ .

CARACTERIZACIÓN DE UNA GRAMÁTICA

Consiste en expresar las características del Lenguaje producido por una Gramática preferentemente por medio de una expresión formal, o bien, por medio de enunciados en Lenguaje natural si lo anterior no es posible.

No existen métodos generales para caracterizar los Lenguajes pero **se debe tener en cuenta el orden en el que se aplicarían las composiciones y cuántas veces se usaría cada una de ellas**. La caracterización es una derivación general, que abarca todas las posibles derivaciones en particular.

En muchos casos es válido y hasta recomendable el obtener la caracterización, basados en la observación y sin tener que llevarla a cabo en una forma muy estructurada, paso a paso. El alumno obtendrá con la práctica y por medio de su razonamiento, una buena capacidad de análisis para caracterizar por la simple observación de la Gramática, lo cual es algo deseable pues en este curso se pueden mejorar esas habilidades, muy importantes para el futuro profesionista.

EJEMPLO:

Caracterizar la Gramática $G = (\{R, S\}, \{a, b\}, \{R \rightarrow bR, R \rightarrow aS, S \rightarrow bS, S \rightarrow b\}, \{R\})$

$\underline{R} \Rightarrow b\underline{R}$

$bb\underline{R}$

$bbb\underline{R}$

...

$b^* \underline{R}$

$b^* a \underline{S}$

$b^* ab \underline{S}$

$b^* abb \underline{S}$

...

$b^* ab^* S$

$b^* ab^* b = b^* ab^+$, o lo que es lo mismo, $b^m ab^n \mid m \in \mathbb{N}_0, n \in \mathbb{N}$.

$$L(G) = \{ b^* ab^+ \} = \{ b^m ab^n \mid m \in \mathbb{N}_0, n \in \mathbb{N} \}$$

Ambas formas de mostrar el resultado final son aceptadas, aunque se recomienda la primera representación por su simplicidad.

Nótese que las reglas recursivas se pueden aplicar una cantidad indefinida de veces o inclusive no aplicarse. Además, para estos casos, si son aplicadas se les debe dar prioridad.

La caracterización sirve para expresar las características que tienen todas las cadenas de un Lenguaje Formal, de donde le viene el nombre. A partir de ella se pueden deducir innumerables cadenas producidas por la Gramática, así como descubrir a simple vista si una cadena pertenece a $L(G)$.

Por ejemplo, las cadenas originadas por la Gramática anterior tienen una sola **a** y terminan en **b**, por lo que por observación simple descubrimos lo siguiente:

$$b^6ab^8 \in L(G) \quad ab^8 \in L(G) \quad b^5ab^2a \notin L(G) \quad b^6a \notin L(G)$$

No todas las Gramáticas se pueden caracterizar, sino sólo aquellas que tienen una estructura de sus composiciones bien elaborada y en orden. Por otra parte si se diera una que no pudiera caracterizarse, no sería útil para casos prácticos.

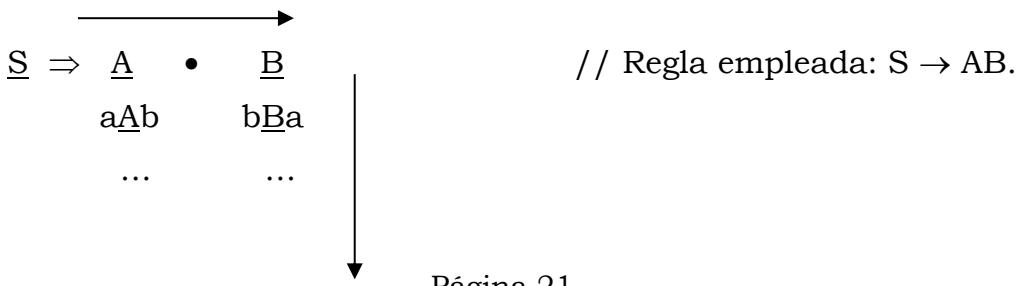
La caracterización de una Gramática Formal, junto con su diseño, constituyen los dos temas más importantes del presente capítulo, por lo que se recomienda su especial atención.

EJEMPLO:

Sean $N = \{ S, A, B \}$, $T = \{ a, b \}$, $S = \{ S \}$. Caracterizar el Lenguaje generado por la Gramática con $P = \{ S \rightarrow AB, A \rightarrow aAb \mid \lambda, B \rightarrow bBa \mid \lambda \}$.

En este caso, sabemos que la primera regla en ser aplicada ha de ser la de $S \rightarrow AB$, pero no sabemos si después debiera derivarse A ó B. El hecho de que A esté a la izquierda no significa que deba ser derivada antes que B.

Haremos un desarrollo algo formal, pero combinando la aplicación de reglas para que sean derivados de manera independiente los dos símbolos no terminales. Dichas derivaciones se harán en forma vertical y al ver el desarrollo, nos daremos cuenta de que si derivamos A y B en forma independiente, podremos al final, concatenar los dos resultados producidos para obtener la respuesta completa.



$$\begin{array}{ll}
 a^m \underline{A} b^m & b^n \underline{B} a^n \quad | \quad m, n \in \mathbb{N}_0 \quad // \text{ Reglas usadas: } A \rightarrow aAb \text{ y } B \rightarrow bBa. \\
 a^m b^m & b^n a^n \quad // \text{ Reglas empleadas: } A \rightarrow \lambda \text{ y } B \rightarrow \lambda.
 \end{array}$$

La primera regla se aplicó horizontalmente y posteriormente se derivó en forma vertical, y por separado, cada uno de los dos no terminales. **A** produjo **a^mb^m** y **B** produjo **bⁿaⁿ**, donde m, n son números naturales incluyendo al cero. Ahora se concatena lo que produjeron ambas y se tiene **a^m b^{m+n} aⁿ**, que es el resultado final, expresado entonces como $L = \{ a^m b^{m+n} a^n \mid m, n \in \mathbb{N}_0 \}$.

En este caso es importante el empleo de letras como superíndices y no asteriscos, ya que hay que especificar que la cantidad de las *a* y de las *b* tienen una relación bien definida: la cantidad de ambas es la misma (estando siempre todas las *b* consecutivas).

EJEMPLO:

Caracterizar la Gramática con reglas $P = \{ S \rightarrow AC, A \rightarrow aAb \mid ab, C \rightarrow cC \mid c \}$ y $S = \{ S \}$ y especificar si produce un Lenguaje finito o infinito.

De acuerdo a la estructura de la Gramática, se observa que el símbolo inicial produce solamente la concatenación AC. La A producirá una *a* y una *b* simultáneamente, quedando las *a* al inicio y las *b* enseguida; nos damos cuenta que se tendrá una cierta cantidad de *a* y luego la misma cantidad de *b*, por lo menos una de cada una. Luego, la C producirá una o muchas veces la *c*, de manera independiente al final de la cadena.

Resulta claro, aún sin desarrollar paso a paso la caracterización, que el resultado se expresa como $L = \{ a^m b^m c^n \mid m, n \in \mathbb{N} \}$.

En este caso es necesario emplear (*m*, *n*) y no (*, +) en el resultado, porque solo con los primeros se pueden especificar cantidades iguales de ciertos símbolos en el Lenguaje. En los casos en los que son indistintas las cantidades en las que aparecen los símbolos, se pueden usar tanto letras como * y + aunque se recomiendan estos últimos.

El Lenguaje obtenido se considera infinito debido a que contiene una cantidad infinita de cadenas.

EJEMPLO:

Dada la Gramática Formal con las composiciones $P = \{ A \rightarrow xA, A \rightarrow xBz, B \rightarrow yBw, B \rightarrow zz \}$ y con $S = \{ A \}$, determinar cuál de las siguientes respuestas corresponde a la caracterización de la misma.

- | | |
|--------------------------------|-------------------------------------------------------------------------------|
| a) $L = \{ x^*y^*z^2w^*z \}$. | e) $L = \{ x^my^nz^2w^{n+2} \mid m \in \mathbb{N}_0, n \in \mathbb{N}_0 \}$. |
| b) $L = \{ x^*y^+z^2w^*z \}$. | f) $L = \{ x^my^nz^2w^{n+2} \mid m \in \mathbb{N}_0, n \in \mathbb{N} \}$. |
| c) $L = \{ x^+y^+z^2w^+z \}$. | g) $L = \{ x^my^nz^2w^{n+2} \mid m \in \mathbb{N}, n \in \mathbb{N}_0 \}$. |

$$d) L = \{ x^+y^*z^2w^+z \}.$$

$$h) L = \{ x^m y^n z^2 w^n z \mid m \in \mathbb{N}, n \in \mathbb{N} \}.$$

Se pudieran inclusive agregar diversos incisos adicionales a los anteriormente propuestos.

Primeramente se observa que en base a la primera regla de producción que es recursiva se podrían generar **x** en una cantidad de una, muchas o ninguna; al aplicar después la segunda composición se deberá generar forzosamente una **x** al inicio y al final de la cadena una **z**. Al derivar la B se deberá de la misma manera aplicar primero la regla recursiva si se llega a usar, con lo que se producen la misma cantidad de **y** que de **w**, en la parte media, pero recordando que justo al centro se mantiene la mencionada B. Por último se producen las dos **z** finales entre los símbolos producidos por la regla anterior.

La clave para descartar definitivamente las cuatro primeras opciones es que en ellas solo aparecen comodines + y *, debiendo empatar la cantidad de **y** con la de **w** que se generan por la tercera composición y esto no se puede lograr con estos símbolos. Como debe haber al menos una **x** al inicio y puede haber o no **y** y **w** entonces se determina que la única respuesta correcta sería la que se refiere en el inciso g.

DISEÑO DE GRAMÁTICAS FORMALES

TAMBIÉN SE PUEDE PRESENTAR EL CASO INVERSO: QUE SE PRESENTE COMO DATO UN LENGUAJE Y EL PROBLEMA CONSISTA EN DISEÑAR LA GRAMÁTICA FORMAL QUE LO PRODUCE.

De acuerdo a los conocimientos previos sobre caracterización de Gramáticas se podrán resolver problemas de diseño, por lo que en este tema se procederá directamente a la resolución de algunos casos demostrativos.

EJEMPLO:

Diseñar una Gramática que genere el Lenguaje $L = \{ (a,b)^* b^2 \}$.

La Gramática sería la que contiene las reglas $P = \{ A \rightarrow aA, A \rightarrow bA, A \rightarrow bB, B \rightarrow b \}$ y con $S = \{ A \}$. Las dos primeras reglas se emplean para generar de manera indistinta una combinación de **a** y **b** de cualquier manera posible; con la tercera composición se genera la primera de las **b** finales y la última **b** con la regla final.

EJEMPLO:

Diseñar una Gramática que genere el Lenguaje $L = \{ x^m y^n z^m \mid m, n \in \mathbb{N}_0 \}$.

Definamos un símbolo inicial; se propone para ese fin la letra **A**, lo cual constituye una costumbre que asumimos en este curso (más no es una regla). Observamos que la cantidad de (**x**, **z**) es similar, lo cual nos hace darnos cuenta de que se deben producir por la misma composición. Las **y** se obtienen de manera separada, ya que su cantidad es independiente que la de (**x**, **z**). Es importante

aclarar que el hecho de que dos símbolos tengan diferentes superíndices significa que la cantidad de símbolos a los que afectan puede ser la misma o diferente.

Si se producen primero las **y**, no se podrán producir después las (**x**, **z**). Por lo tanto debemos tratar de producir primero éstas últimas, y no en forma indistinta.

Se propone iniciar con la regla $A \rightarrow xAz$, regla que produce simultáneamente a **x** y **z** en el orden adecuado. La recursividad permite la asignación de una, ninguna o muchas veces la aparición de tales símbolos. La siguiente regla es $A \rightarrow B$, para evitar que se vuelvan a producir (**x**, **z**) además de evitar que se revuelvan con las **y**.

Luego se agrega la regla recursiva $B \rightarrow By$, o bien la regla $B \rightarrow yB$. Con ello, aparecerán las **y** que sean necesarias. No importa si se produce By o yB , ya que no se pueden mezclar las **y** con los otros dos símbolos. Finalmente $B \rightarrow \lambda$ para eliminar el no terminal y concluir la cadena que se vaya a producir.

En síntesis, el resultado obtenido es $P = \{ A \rightarrow xAz, A \rightarrow B, B \rightarrow By, B \rightarrow \lambda \}$ con $S = \{ A \}$.

EJEMPLO:

Diseñar una Gramática que genere todas las cadenas de bits que contengan una cantidad par de 0 y una cantidad par de 1.

Es muy importante que se considere de inicio cuál debe ser la cadena mínima que cumple con las características referidas y fácilmente se observa que es la cadena vacía λ . La única forma de obtenerla en este caso es cuando se hace a partir del símbolo inicial que podría ser A , por lo que una regla obligatoria será $A \rightarrow \lambda$.

Por otra parte para generar las demás cantidades pares de 0 y de 1 se debe considerar que si a un número par se le suma dos se obtiene la siguiente cantidad par por lo que entonces se agregarían $A \rightarrow 00A$ y $A \rightarrow 11A$ como composiciones de la Gramática; se emplearían reglas por separado para que las cantidades sean independientes, y cada vez que se aplica cada una de ellas se agregan dos ceros o dos unos para tener cualquier cantidad par posible. La regla mencionada en el párrafo anterior serviría también para terminar con la presencia de A en las derivaciones.

El problema que surge en este momento es que a pesar de que las cadenas tendrían ambos bits en cantidades pares, al no saber cuántos habría ni en qué orden aparecerían no se podrían formar todas las palabras posibles. Se podrían agregar reglas como $A \rightarrow A00$, $A \rightarrow 0A0$, $A \rightarrow A11$ y $A \rightarrow 1A1$ pero cadenas tan simples como 0101 no se podrían producir. Se requeriría una regla como $A \rightarrow 01A$ o $A \rightarrow 10A$ pero al emplear estas últimas ya no se tienen las cantidades pares. Imaginemos cuántas combinaciones se podrían hacer con 48 ceros y 16 unos en posiciones cualesquiera, por citar un caso, y las diversas reglas que se requerirían.

Una alternativa para solucionar este problema sería si solamente se toman las reglas $A \rightarrow OOA$, $A \rightarrow IIA$, $A \rightarrow \lambda$ y además de las anteriores se consideran las

siguientes: $OI \rightarrow IO$, $IO \rightarrow OI$, $O \rightarrow O$, $I \rightarrow I$. La intención es que con las dos primeras composiciones se produzcan pares de O y de I en las cantidades que se requiera, terminado con la presencia de A por medio de la tercera regla. Posteriormente y siendo letras la O y la I pueden estar en el lado izquierdo de las siguientes reglas en las que se pueden intercambiar ambas y de esta manera una cadena por ejemplo OOIIOOOOIIOOII se pudiera transformar en OOIOOOIIIOIOOIOI únicamente por intercambios de lugar entre dos bits consecutivos, algo parecido a la forma como se ordenan elementos de una lista con el algoritmo de la burbuja que se estudia en los cursos de Estructura de Datos. Finalmente las O se transforman en 0 y las I en 1.

Aunque esta no sería la única solución posible se ha considerado que es adecuada para cumplir con la especificación deseada.

Obsérvese que es importante que se produzcan las cadenas especificadas, desde la más mínima, que en los dos casos anteriormente resueltos sería λ . Las Matemáticas son Ciencias Exactas, por lo que debe tenerse especial cuidado en la exactitud de un resultado obtenido.

MUY IMPORTANTE: CUANDO SE DISEÑA UNA GRAMÁTICA, ÉSTA DEBERÁ PRODUCIR TODAS LAS CADENAS QUE SE ESPECIFICAN, PERO NI UNA SOLA PALABRA MÁS. NI UNA MÁS, NI UNA MENOS.

GRAMÁTICAS EQUIVALENTES

Dos Gramáticas G_1 y G_2 se llaman equivalentes, en símbolos $G_1 \sim G_2$, si $L(G_1) = L(G_2)$, es decir, si producen exactamente el mismo Lenguaje a pesar de ser diferentes. Si dos Gramáticas son equivalentes, entonces tendrán los mismos símbolos terminales, y probablemente el mismo símbolo inicial. Sin embargo, las composiciones serían diferentes y, muy posiblemente, también el conjunto de los símbolos no terminales.

Por ejemplo, son equivalentes las tres siguientes Gramáticas:

$$P = \{ A \rightarrow xA, A \rightarrow Ay, A \rightarrow xy \}$$

$$P = \{ A \rightarrow xA, A \rightarrow xB, B \rightarrow By, B \rightarrow y \}$$

$$P = \{ A \rightarrow BC, B \rightarrow xB, B \rightarrow x, C \rightarrow Cy, C \rightarrow y \}$$

Ya que todas ellas producen al mismo Lenguaje $L(G) = \{ x^+y^+ \}$.

También son equivalentes las Gramáticas con $P = \{ A \rightarrow xA, A \rightarrow Ay, A \rightarrow \lambda \}$ y $P = \{ A \rightarrow xA, A \rightarrow yB, A \rightarrow y, A \rightarrow \lambda, B \rightarrow yB, B \rightarrow y \}$, siendo A el símbolo inicial, ya que ambas generan el Lenguaje $L = \{ x^*y^* \}$, aunque se ve fácilmente que la mejor opción por su simplicidad es la primera, pues es la que tiene menos reglas y símbolos no terminales.

Nótese cómo, si bien una Gramática debe producir siempre el mismo Lenguaje, un Lenguaje puede ser generado por distintas Gramáticas. Esta afirmación concuerda con las características de una función en términos matemáticos, recordando que $L = f(G)$, dependencia que expresamos también como $L(G)$.

Considerar este criterio cuando en la tarea se resuelve un problema que consista en el diseño de una Gramática, ya que el alumno podría obtener una respuesta diferente a la que obtiene el profesor en clase y sin embargo podría ser correcta también.

JERARQUÍA DE CHOMSKY

Es una clasificación en cuatro clases o subtipos de Gramáticas Formales cuya definición, realizada por *Noam Chomsky* en 1956, marcó el comienzo de esta teoría. Estas clases de Gramáticas se denominan como tipo 0, 1, 2 y 3, siendo cada una de ellas un subconjunto de la anterior. Todas ellas pueden relacionarse con una clase de Gramática o por una clase de Autómata o reconocedor, como se indica en la tabla que se muestra a continuación.

GRAMÁTICA	TIPO	 LENGUAJE	AUTÓMATA
Sin restricciones o no restringida	0	Recursivamente Ennumerable	Máquina de Turing
Sensible al Contexto	1	Sensible al Contexto	Autómata Linealmente Acotado
Libre o independiente de Contexto	2	Libre o independiente de Contexto	Autómata de Pila
Regular	3	Regular	Autómata Finito

En la tabla anterior se especifica y relaciona, por cada renglón, un tipo de Gramática (por nombre o por número) con el tipo de Lenguaje que produce, además de mencionar el analizador o Autómata que se emplearía para reconocer las cadenas que pertenecen a ese Lenguaje.

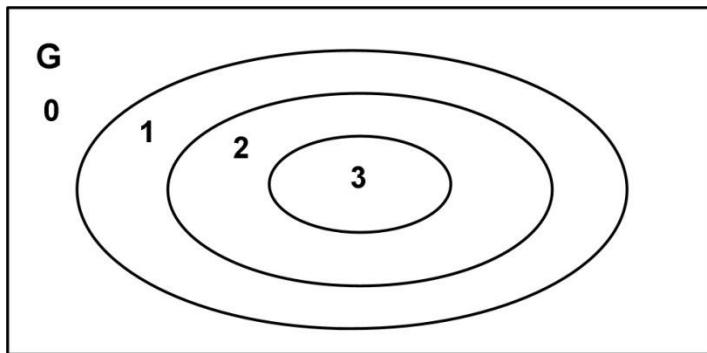
La Jerarquía de Chomsky constituye el criterio oficial para clasificar las Gramáticas Formales e inclusive a los Lenguajes que producen. Es importante conocerla, ya que cada uno de los cuatro tipos tiene características muy importantes en aspectos computacionales.

Existen, sin embargo, algunas propuestas poco usuales para hacer una clasificación extendida que a diferencia de la actual tiene una mayor cantidad de tipos. Para fines prácticos, no se considerarán esos criterios en este curso.

De la tabla anterior, se descubre cuál es el Autómata idóneo para aceptar o rechazar las cadenas para un determinado tipo de Lenguaje, sabiendo el tipo de Gramática empleada.

Emplearemos esta tabla para hacer demostraciones acerca de la pertenencia a un determinado tipo de Gramática o Lenguaje, por medio de condiciones suficientes de cumplir. Por ejemplo, se demuestra satisfactoriamente que un Lenguaje es Libre de Contexto si existe una Gramática de ese tipo que lo produzca, o bien si es aceptado por un Autómata de Pila.

El nombre de Jerarquía se debe a que los cuatro tipos no son disjuntos, sino que se jerarquizan en subconjuntos, como se observa en el siguiente Diagrama de Venn:



A una Gramática se le debe considerar de acuerdo al tipo más selectivo que logre alcanzar, resultando ser obvio que también cumple con los requisitos de las Gramáticas de los tipos precedentes. Por ejemplo, si una Gramática es de tipo 2 o Libre de Contexto, se le considera y denomina de esta manera y no como Sensible al Contexto o Sin Restricciones, a pesar de que cumple con las condiciones para ambos tipos.

Es importante aclarar que como consecuencia de la Jerarquía pueden darse casos en que se puedan emplear analizadores o Autómatas que sin embargo no son los idóneos. Por ejemplo, un Autómata de Pila podría emplearse para reconocer un Lenguaje Regular (desaprovechando la pila), pero uno Finito no podrá reconocer un Lenguaje Libre de Contexto. ¿Por qué?

GRAMÁTICA SIN RESTRICCIONES (TIPO 0)

No existen restricciones en las reglas de producción, excepto que cumplan con las características para tener sus composiciones elaboradas en forma correcta. De hecho, cualquier Gramática, debe ser por lo menos de tipo 0 y es que todas deben quedar clasificadas dentro de alguno de los cuatro tipos.

EJEMPLO:

Diseñar una Gramática de Chomsky, en este caso sin especificaciones concretas.

$$G = (N, T, P, S)$$

$$N = \{A, B, C, D\}$$

$$T = \{0, 1, 2\}$$

$$S = \{A\}$$

$$P = \{A \rightarrow B02 \mid D01 \mid 0, B \rightarrow CA, B2 \rightarrow A, 1B \rightarrow 1A, AB \rightarrow BA \mid D, C \rightarrow 0A \mid 1B2A \mid 1, D \rightarrow A0 \mid B02 \mid 2 \mid \lambda\}$$

Al reconocer las siguientes clases de Gramática se verá que la anterior tiene tres reglas que no cumplen con las características para ser clasificada según otra categoría y que son $B2 \rightarrow A$, $AB \rightarrow BA$ y $D \rightarrow \lambda$.

Nótese que aunque parezca extraño, son consideradas como válidas las reglas aunque contengan terminales en su lado izquierdo siempre y cuando sean acompañados de un no terminal o las que llevan menos símbolos al lado derecho, incluido el caso de las reglas λ .

GRAMÁTICA SENSIBLE AL CONTEXTO (TIPO 1)

Toda composición es de la forma $\alpha A \beta \rightarrow \alpha \delta \beta$ donde: $\alpha, \beta \in (N \cup T)^*$; $A \in N$; $\delta \in (N \cup T)^+$. Se interpreta como que el prefijo derecho sea igual que el izquierdo, que el sufijo derecho sea igual que el izquierdo, que el infijo izquierdo sea un solo símbolo no terminal, y que el infijo derecho sea cualquier cadena con excepción de λ .

Hay algunos autores que aceptan también para este tipo de Gramática las reglas de formato $A \rightarrow \lambda$, donde un no terminal cualquiera produce la cadena vacía; otros las aceptan solo si A es el símbolo inicial; otros más no las aceptan. Para este curso la regla de producción del tipo mencionado NO se considerará válida para este tipo de Gramática.

Se deben seleccionar esos cuatro elementos en forma tal que de alguna forma posible cumplan el requisito correspondiente. Se recomienda que se asegure primero el adecuado infijo izquierdo (A), ya que es el más difícil de obtener.

EJEMPLO:

Diseñar una Gramática Sensible al Contexto, en este caso sin especificaciones concretas.

$$G = (N, T, P, S)$$

$$N = \{ S, A, B, C, D, E \}$$

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

$$S = \{ S \}$$

$$P = \{ S \rightarrow aAB \mid aB, A \rightarrow aAC \mid aC, B \rightarrow Dc, D \rightarrow b, CD \rightarrow CE, CE \rightarrow DE, DE \rightarrow Dc, Cc \rightarrow Dcc \}.$$

Obtener una cadena $s \in L(G)$ y demostrar que en muchos casos no es muy recomendable trabajar con este tipo de Gramáticas debido a la indecibilidad.

$$S \Rightarrow a\underline{AB} \Rightarrow aaAC\underline{B} \Rightarrow aaACD\underline{c} \Rightarrow aaACbc \Rightarrow aaaCCbc \Rightarrow ?$$

Al intentar producir una cadena de $L(G)$ se llegó a un paso de la derivación en donde ya no se podía continuar, por no existir una regla adecuada y que se pudiera aplicar en ese momento. Podría pensarse que la Gramática no es válida pero en realidad sí lo es, aunque la misma está mal estructurada.

De hecho, el símbolo C sí aparece al lado izquierdo, no sólo en una regla, sino en tres, aunque siempre acompañada de otro símbolo.

A pesar de que en el caso anterior no se obtuvo alguna cadena, éstas sí se pueden obtener con otras derivaciones posibles, inclusive es producido el Lenguaje $L(G) = \{ a^m b^n c^m \mid m, n \in \mathbb{N} \}$.

Dado que puede existir indecibilidad en esta Gramática (cuando se presenta el caso de una derivación que no se pueda concluir en una cadena válida); es lógico suponer que también existirá y con mayor razón en la de tipo No Restringida, ya que no hay en ellas ni la más mínima restricción, además de ser el superconjunto de la tipo 1. Ambas clases de Gramáticas son muy poco usuales en aplicaciones computacionales, por lo que poco se hablará de ellas en el resto de este curso.

Si $N = \{ A, B, C, D \}$, $T = \{ a, b, c \}$ y $S = \{ A \}$ se podrían considerar como posibles reglas para una Gramática de este tipo algunas como las siguientes:

$$\underline{aC} \underline{D} b \rightarrow \underline{aC} \underline{B} a b, \quad \underline{B} \underline{C} \rightarrow \underline{B} \underline{A}, \quad \underline{a} \underline{D} \rightarrow \underline{a} \underline{B}, \quad \underline{\underline{A}} \underline{c} \rightarrow \underline{\underline{c}} \underline{b} \underline{c} \quad \underline{\underline{B}} \rightarrow \underline{\underline{c}}$$

Dado que se cumple que $|\alpha \mathbf{A} \beta| \leq |\alpha \delta \beta|$, una característica de las derivaciones en una Gramática Sensible al Contexto, es que la longitud de la cadena nunca se decremente al derivar. **En todas las composiciones debe haber mayor o igual cantidad de símbolos en el lado derecho que en el izquierdo. POR ESTA RAZÓN NO SE DEBERÍAN ACEPTAR LAS REGLAS LAMBDA COMO PARTE DE ESTAS GRAMÁTICAS, PUES EN ELLAS HAY CERO SÍMBOLOS A LA DERECHA Y NO SE CUMPLE CON EL REQUISITO MENCIONADO.**

El nombre de sensible (o dependiente) al contexto le resulta de que, para cambiar A por δ , que es en sí lo único que cambia cuando se deriva, se depende necesariamente de la presencia de α y β , flanqueando a la A .

GRAMÁTICA LIBRE DE CONTEXTO (TIPO 2)

Toda derivación es de la forma $\mathbf{A} \rightarrow \delta$ donde $\mathbf{A} \in N$ y $\delta \in (N \cup T)^*$. El lado izquierdo de las composiciones es un único símbolo no terminal y el derecho podrá ser cualquier combinación de terminales y no terminales, excepto λ .

Hay algunos autores que aceptan también para este tipo de Gramática las reglas de formato $A \rightarrow \lambda$, donde un no terminal cualquiera produce la cadena vacía; otros las aceptan solo si A es el símbolo inicial; otros más no las aceptan. Para este curso la regla de producción del tipo mencionado Sí se considerará válida para este tipo de Gramática.

EJEMPLO:

Diseñar una Gramática Libre de Contexto, en este caso sin especificaciones concretas.

$$G = (N, T, P, S).$$

$$N = \{ E, F, G, H \} \quad T = \{ q, r, s \} \quad S = \{ E \}$$

$$P = \{ \begin{array}{l} E \rightarrow FGs \mid rE \mid H \mid rs, \\ F \rightarrow Hq \mid r, \\ G \rightarrow Ers \mid Fq \mid sE \mid s, \\ H \rightarrow GEs \mid sG \mid q \end{array} \}$$

En una Gramática de este tipo diseñada correctamente nunca se presenta un caso de indecibilidad. En este caso, para derivar un no terminal, no se requiere de la presencia de símbolo alguno a un lado, es decir, no depende del contexto.

Este tipo es muy importante en las aplicaciones computacionales, ya que es el que se emplea para definir la sintaxis de los Lenguajes de programación como C, Java o Prolog.

GRAMÁTICA REGULAR (TIPO 3)

Si toda composición es de una de las dos formas válidas, que son $A \rightarrow a$, o bien $A \rightarrow aB$; donde $A, B \in N$ y $a \in T$. El lado izquierdo está formado por un símbolo no terminal y el derecho por un terminal, o un terminal seguido de no terminal.

Hay algunos autores que aceptan también para este tipo de Gramática las reglas de formato $A \rightarrow \lambda$, donde un no terminal cualquiera produce la cadena vacía; otros las aceptan solo si A es el símbolo inicial; otros más no las aceptan. Para este curso la regla de producción del tipo mencionado Sí se considerará válida para este tipo de Gramática.

Siempre debe aparecer el símbolo terminal antes que el no terminal en la parte derecha de las reglas. Por ejemplo, no es válido que una regla sea $A \rightarrow Ax$.

Al derivar para producir una cadena, aparece un solo símbolo no terminal en cada paso, quedando siempre hasta el final de la cadena que se está produciendo (hasta la derecha); la cadena se va formando símbolo a símbolo terminal de izquierda a derecha. Obsérvese la regularidad que manifiesta y que le significó el nombre.

EJEMPLO:

Diseñar una Gramática Regular, en este caso sin especificaciones concretas.

$G = (N, T, P, S)$

$N = \{ A, B, C, D, E \}$

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

$S = \{ A \}$

$P = \{ A \rightarrow xA \mid yC \mid x, \quad B \rightarrow xC \mid xD \mid zE \mid x,$

$C \rightarrow yA \mid xD \mid yE \mid y, \quad D \rightarrow xA \mid xB,$

$E \rightarrow yB \mid yE \mid z \}$

Una cadena producida por esta Gramática podría ser:

$A \Rightarrow xA \Rightarrow xyC \Rightarrow xyyE \Rightarrow xyyyE \Rightarrow xyyyyE \Rightarrow xyyyyyB \Rightarrow xyyyyyx.$

Observar que una cadena se construye símbolo por símbolo y de izquierda a derecha en una forma muy sencilla, como ya se había dicho. El no terminal siempre aparece como sufijo.

No es posible diseñar un Lenguaje Regular en el que se igualen o se relacionen de alguna manera las cantidades de símbolos implicados en la expresión; por lo anteriormente expuesto no serían Lenguajes Regulares, ni $L = \{ x^n y^n \mid n \in \mathbb{N} \}$, ni $L = \{ x^n y^{2n-1} \mid n \in \mathbb{N} \}$, ni $L = \{ x^m y^n \mid m > n ; m, n \in \mathbb{N} \}$, por ejemplo.

Se llama Expresión Regular a aquella que representa un Lenguaje Regular. En programación tienen una gran importancia en el desarrollo de editores de texto y aplicaciones para buscar y manipular cadenas, entre otras.

Este tipo de Gramática Formal es el que se emplea para definir la estructura de los componentes léxicos (tokens) en los Lenguajes de programación como C, Java o Prolog.

EJEMPLO:

Diseñar una Gramática Regular con $T = \{ a, b \}$, que genere todos los arreglos que contengan el infijo ba. El Lenguaje se representaría por medio de la expresión regular $L = \{ (a, b)^* b a (a, b)^* \}$.

Sea A el símbolo inicial. De acuerdo al formato que se mencionó anteriormente, se deben producir los terminales símbolo a símbolo. Por tanto, debemos pensar primero en producir la parte inicial de las cadenas de este Lenguaje, que consiste en una mezcla de una, ninguna o muchas (a, b) . Las primeras reglas son $A \rightarrow aA$ y $A \rightarrow bA$. Al emplearlas en forma indistinta se forma el prefijo $(a, b)^*$. Para continuar, se produce la b , pero la a final todavía no; la composición será $A \rightarrow bB$, con la cual se obliga a cambiar de no terminal (de A a B) para que la siguiente regla a emplear sea forzosamente la de $B \rightarrow aC$ y con ello se produzca la a deseada pero con la posibilidad de que la C genere el posfijo $(a, b)^*$. Por supuesto habría que agregar las infaltables reglas $C \rightarrow aC$, $C \rightarrow bC$ además de $C \rightarrow a$ y $C \rightarrow b$ para dar fin a la generación de símbolos y el posfijo requerido.

Es importante agregar antes una composición $B \rightarrow a$ por si ya no hubiera más símbolos al final

Entonces la respuesta es $P = \{ A \rightarrow aA, A \rightarrow bA, A \rightarrow bB, B \rightarrow aC, B \rightarrow a, C \rightarrow aC, C \rightarrow bC, C \rightarrow a, C \rightarrow b \}$ con $S = \{ A \}$.

TEOREMAS:

- Si L_1 y L_2 son Lenguajes regulares, entonces $L_1 \cup L_2, L_1 \bullet L_2, L_1^*, L_2^*$, también son Lenguajes Regulares.
- Si L es un Lenguaje Regular, el Lenguaje $L^R = \{ x_n \dots x_2 x_1 \mid x_1 x_2 \dots x_n \in L \}$ también es Regular.

COMENTARIO FINAL SOBRE LOS TIPOS DE GRAMÁTICAS

Al consultar la bibliografía alterna (o las páginas de internet sobre este tema) el alumno se puede dar cuenta de que pueden existir algunas diferencias en los criterios para clasificarlas en el tipo que corresponde. Por ejemplo, **VARIOS AUTORES CONSIDERAN QUE LAS REGLAS LAMBDA SON VÁLIDAS EN LAS GRAMÁTICAS DE TIPO 2 O DE TIPO 3 BAJO CIERTAS CONDICIONES, AUNQUE ESA DECISIÓN PUEDE AFECTAR A LA JERARQUÍA DE CHOMSKY MIENTRAS QUE OTROS NO LAS CONSIDERAN COMO PERMITIDAS**; otros reconocen a las Gramáticas Regulares por la derecha o por la izquierda, donde las primeras son las que se están estudiando en este curso y las segundas son aquella en las que las reglas son todas del tipo $A \rightarrow a$ o $A \rightarrow Ba$; algunos opinan que en la Gramática Sensible al Contexto basta que las reglas tengan menos símbolos a la izquierda que a la derecha para cumplir con este tipo. *Para desarrollar estas notas, ha sido considerada la definición que se ha considerado es la más conveniente para los objetivos del curso.*

Además, algunos autores consideran como **Gramática Lineal** a la que cada producción contiene a lo sumo, un no terminal en el lado derecho de las reglas. Esta es lineal derecha si un no terminal puede únicamente aparecer como su símbolo extremo derecho, es decir si cada producción tiene una de las formas $A \rightarrow \omega$ o $A \rightarrow \omega B$, donde A, B son sendos no terminales y ω es una cadena constituida por uno o muchos terminales. Una Gramática lineal izquierda puede definirse como la que contiene reglas $A \rightarrow \omega$ o $A \rightarrow B\omega$.

Ejemplo de Gramática lineal derecha:

$$P = \{ A \rightarrow aA \mid abA \mid abB \mid abc, B \rightarrow cA \mid bcB \mid ab \}$$

En sí esta es una Gramática Libre de Contexto. Téngase en cuenta que este tipo de Gramática no forma parte de la Jerarquía de Chomsky, y que su empleo es más bien debido a que en este caso se puede identificar fácilmente cuales son las composiciones que llevan a alguna cadena producida específica.

Sin embargo, si se definen las secuencias de caracteres consecutivos presentes en las composiciones, como a, c, ab, bc y abc para que cada una de ellas sea un único terminal, entonces sí sería una Gramática Regular.

PROBLEMAS ADICIONALES

EJEMPLO:

Determinar en cada inciso el tipo de Gramática que se tiene, expresar si es o no decidable y caracterizarla.

a) $G = (\{ A, B \}, \{ 0, 1 \}, \{ A \rightarrow 0B, B \rightarrow 10B \mid \lambda \}, \{ A \})$.

La Gramática es Tipo 2 o Libre de Contexto. Se observa que en todas las reglas de producción se tiene que un único símbolo no terminal produce algo y como en este curso no importa que se genere una cadena vacía, ese será motivo suficiente para clasificarla así.

No es decidable, ya que la cadena vacía no se puede producir con estas producciones.

La caracterización es $L(G) = \{ 0 (10)^* \}$.

b) $G = (\{ A \}, \{ 0, 1 \}, \{ A \rightarrow A01 \mid 0 \}, \{ A \})$.

La Gramática es de Tipo 2 o Libre de Contexto. El lado izquierdo de las reglas consiste en un único no terminal, mientras que el derecho nunca es la cadena vacía.

No es decidable, ya que la cadena vacía no se puede producir con estas reglas.

La caracterización es $L(G) = \{ 0 (01)^* \}$.

c) $G = (\{ I, R, S \}, \{ x, y, z \}, \{ I \rightarrow RIS \mid yI \mid \lambda, R \rightarrow x, S \rightarrow z \}, \{ I \})$.

La Gramática es Tipo 2 o Libre de Contexto. Se debe a las mismas razones que la analizada en el inciso a.

Sí es decidable, ya que la cadena vacía sí se puede producir con estas reglas, es decir que $\lambda \in L(G)$.

Para caracterizar, hay que observar que las reglas $I \rightarrow RIS$ y $I \rightarrow yI$ pueden ser empleadas de manera alternada y no tiene prioridad en realidad ninguna de las dos. Al analizar, nos damos cuenta que las cadenas que se producen, *contienen la misma cantidad de x que de z, y que todas las z se colocan al final, mientras que las x y las y se pueden mezclar al inicio de la palabra*. Por supuesto, se debe tomar en cuenta la palabra λ , que también es producida. ¿Cómo se puede representar tal Lenguaje con una expresión formal?

EJEMPLO:

Dada la Gramática con $P = \{S \rightarrow A \mid AAB, Aa \rightarrow ABa, A \rightarrow aa, AB \rightarrow ABB, Bb \rightarrow ABb, B \rightarrow b\}$

a) ¿De qué tipo es, según la Jerarquía de Chomsky?

Es una Gramática Sensible al Contexto o de tipo 1. En los casos de las composiciones donde el lado izquierdo lo compone un único no terminal, no hay problema para aceptarlas, ya que son válidas, inclusive en el caso de Gramáticas Libres de Contexto. Para el caso de las reglas $\underline{A} \underline{a} \rightarrow \underline{AB} \underline{a}$, $\underline{A} \underline{B} \rightarrow \underline{A} \underline{BB}$ y $\underline{B} \underline{b} \rightarrow \underline{Ab} \underline{b}$ se puede verificar que sí se cumple con el formato requerido de $\alpha\delta\beta$ de acuerdo a como se muestran las partes separadas en el subrayado. En caso de que no quede muy clara la justificación de la respuesta, preguntar al profesor.

b) Demostrar que $a^2b^2a^2b$ es una cadena de L(G).

Se demuestra por medio de la siguiente derivación:

$\underline{S} \Rightarrow A\underline{AB} \Rightarrow A\underline{aa}B \Rightarrow A\underline{Baa}B \Rightarrow A\underline{BBaa}B \Rightarrow aa\underline{BBaa}B \Rightarrow aab\underline{Baa}B \Rightarrow aabbaa\underline{B} \Rightarrow aabbaab.$

c) ¿Es decidible? ¿Por qué?

No es una Gramática decidible porque no se puede producir la cadena vacía con las reglas del conjunto P. Inclusive, ni siquiera existen reglas lambda en la Gramática.

d) Determinar, si es posible, una cadena de L(G) tal que $|s| = 6$ por derivación a la derecha.

No es posible hacer esa derivación, ya que el enunciado que describe cómo se hace una derivación por derecha (o por izquierda) menciona que la Gramática deberá ser Libre de Contexto, y ésta no lo es, como se mencionó en a). Por otra parte, sí se pueden hacer derivaciones de cadenas de L(G) con longitud 6.

EJEMPLO:

Dada la Gramática con $P = \{ S \rightarrow bS \mid aA \mid b, A \rightarrow aS \mid bA \mid a \}$ con símbolo inicial S.

a) ¿Qué cantidad de a y de b pueden tener las cadenas de L(G)?

La cantidad de letras **b** que se obtienen es un número CUALQUIERA desde 0 en adelante. La cantidad de letras **a** que se obtienen es un número PAR desde 0 en adelante.

Obsérvese que es imposible que esta Gramática produzca cadenas con una cantidad impar de letras **a**.

b) ¿Es una Gramática decidible? ¿Por qué?

No es una Gramática decidible, porque la cadena vacía no es parte del Lenguaje L(G).

c) ¿De qué tipo es, según la Jerarquía de Chomsky?

Es una Gramática Regular o de Tipo 3.

d) Encontrar, si es posible, una cadena de longitud 6 por derivación a la derecha y otra por la izquierda.

$S \Rightarrow b\underline{S} \Rightarrow bb\underline{S} \Rightarrow bba\underline{A} \Rightarrow bbab\underline{A} \Rightarrow bbabb\underline{A} \Rightarrow bbbabba$.

La derivación anterior nos da una cadena de longitud 6. En este caso se tiene que es tanto por la derecha como por la izquierda, ya que se derivó primero el símbolo no terminal que está más hacia la derecha o la izquierda, según como se quiera ver. ¡Siempre aparece un único símbolo no terminal a la vez, cada vez que se deriva!

En toda Gramática Regular, como nunca aparece más de un símbolo no terminal a la vez cuando se está derivando, siempre es la misma la derivación por la izquierda que por la derecha, por lo que no es usual el aplicar este concepto en esta clase de Gramáticas.

EJEMPLO:

Demostrar que $L(G) = \{ a^m b^m c^n \mid m, n \in \mathbb{N} \}$ es un Lenguaje Libre de Contexto.

Esta demostración se podría hacer de dos maneras principalmente, siendo una de ellas por medio de un procedimiento muy formal y que cumple con las características expuestas en los cursos curriculares de Matemáticas, llamado **Teorema o Lema del Bombeo**. Aunque esta sería la manera más correcta de hacer la demostración, se considera que debido a que se requiere de una cantidad extensa de tiempo para hacer la exposición del tema, se buscaría otra opción más sencilla y rápida pero siendo también válida y aceptada.

La forma más simple y basada en condiciones suficientes y necesarias sería en base a una tabla que se ha expuesto anteriormente en la que se hace la clasificación de Gramáticas según la Jerarquía de Chomsky y en la cual también por consecuencia se clasifican los Lenguajes Formales y se define cuál es el Autómata o reconocedor para cada tipo. Existe un renglón en el que se observa que las Gramáticas Libres de Contexto, también llamadas de tipo 2, producen Lenguajes de esa misma clase.

Como la Gramática que genera el Lenguaje sería $P = \{ A \rightarrow BC, B \rightarrow aBb \mid ab, C \rightarrow cC \mid c \}$, la cual es Libre de Contexto, se considera que el Lenguaje es de ese mismo tipo.

Sin embargo podría suceder que exista otra variante de la Gramática que produjera ese mismo Lenguaje pero siendo Regular, en cuyo caso aquella ya no sería llamada Libre de Contexto.

En razón de que no es posible que se pueda construir dicha variante, debido a que en las Gramáticas Regulares se forman las cadenas de izquierda a derecha, símbolo por símbolo. Nuestro intento por diseñar esa variante podría ser con $P = \{ A \rightarrow aA \mid aB, B \rightarrow bB \mid bC, C \rightarrow cC \mid c \}$. Cuando se han producido las **a**, y se procediera a generar las **b**, ¿cómo se podría saber cuántas de las primeras se

generaron? En este caso el Lenguaje que se genera es $L(G) = \{ a^m b^n c^n \mid m, n, \tilde{n} \in \mathbb{N} \}$ el cual sí es Regular.

Por lo anteriormente expuesto queda plenamente demostrado que el Lenguaje expuesto es Libre de Contexto.

ÁRBOLES DE DERIVACIÓN

CONCEPTO: Es una forma de indicar gráficamente cómo una **Gramática Independiente de Contexto** deriva una palabra particular. En este árbol matemático, los nodos de las hojas del árbol son símbolos terminales y los nodos interiores son los no terminales. En este caso ocurre que secuencias de derivación diferentes corresponden al mismo árbol de derivación, ya que en esta representación no se conoce el orden en que se aplicó cada composición.

Es útil en muchos casos presentar las derivaciones como árboles. Estos diagramas conocidos también como *árboles de análisis gramatical o de análisis sintáctico* suponen una estructura sobre las palabras de un Lenguaje y son de utilidad en aplicaciones tales como la compilación de los Lenguajes de programación.

El símbolo inicial se define como el nodo raíz del árbol. Si se va a emplear la regla $A \rightarrow x_1 x_2 \dots x_n$, al ver que en ese momento A es un vértice terminal, entonces al mencionado nodo se le asignan sus hijos, que estarán etiquetados como x_1, x_2, \dots, x_n siendo cada uno de ellos colocado en una rama por separado. Se debe derivar hasta que todas las hojas sean terminales.

PREGUNTA IMPORTANTE: ¿Por qué no se pueden usar los árboles de derivación en Gramáticas que no sean Libres de Contexto?

Pistas: ¿Cómo se diseñaría la parte del árbol que corresponde a la aplicación de una regla tal como $AB \rightarrow A1$? ¿Sería realmente un árbol el que se obtendría? ¿Se pueden tener a dos nodos padre al ramificar, en este caso, A y B? ¿Y si se tiene una regla lambda como $C \rightarrow \lambda$, cuál o cuáles serían los hijos?

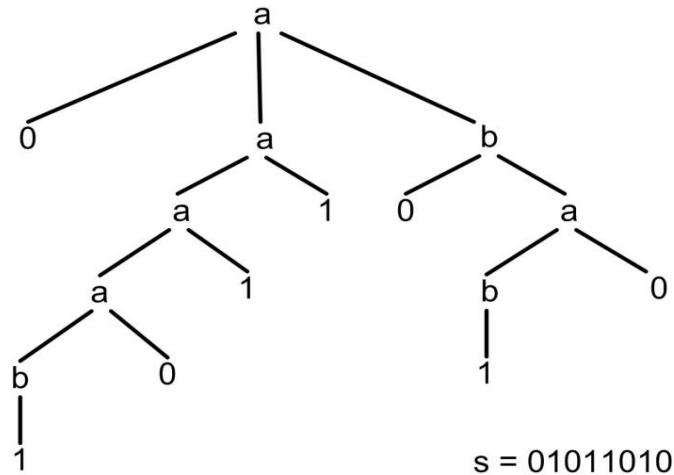
EJEMPLO:

Dada la Gramática $G = (\{a, b\}, \{0, 1\}, \{a \rightarrow 0ab \mid a1 \mid b0, b \rightarrow 0a \mid 1\}, \{a\})$, determinar alguna cadena $s \mid s \in L(G)$.

Originalmente, se obtenían las cadenas con la derivación en la forma tradicional, de la siguiente manera:

$$\begin{array}{l} a \Rightarrow 0\underline{a}b \Rightarrow 0a1\underline{b} \Rightarrow 0\underline{a}10a \Rightarrow 0\underline{a}110a \Rightarrow 0b0110\underline{a} \Rightarrow 0b0110b0 \Rightarrow 010110\underline{b}0 \Rightarrow \\ 01011010 \qquad \qquad \qquad s = (01)^2(10)^2 \qquad \qquad \qquad |\underline{s}| = 8 \end{array}$$

El árbol de derivación que se hubiera diseñado con el mismo fin, es el siguiente:



El árbol se lee primero en profundidad en forma similar a como se hace con los que se utilizan en las estructuras de datos.

Al analizar el árbol notaremos que todo nodo interior constituye el lado izquierdo de una composición utilizada en ese momento, y la concatenación de los nodos hijos representarían el lado derecho de la misma.

Obsérvese cómo en la producción original por derivaciones sucesivas no hay dudas en cuanto al orden de aplicación de las composiciones, pero sí cuando se obtiene el árbol y deseamos saber cómo se realizaron las derivaciones que lo produjeron. Como en muchas ocasiones no importa ese orden, esta limitación no representa problema alguno.

Ni la derivación por la derecha ni por la izquierda se pueden expresar por árboles de derivación, porque en estos no se conoce el orden en que se han utilizado las reglas.

GRAMÁTICAS AMBIGUAS Y UNÍVOCAS

Una Gramática puede tener **más de un árbol de derivación que genere una cadena determinada**, en cuyo caso se la llamará **ambigua**. Para demostrar que una Gramática es ambigua lo único que se requiere es encontrar una cadena que tenga más de un árbol de derivación que la produzca. En contraposición, la **únivoca** es una Gramática Libre de Contexto que tiene asociado **un solo árbol de derivación para toda cadena** de $L(G)$.

Un Lenguaje **L** se llama **AMBIGUO** si existe una Gramática ambigua que lo genere y si no es así, en contraposición se lo llama **UNÍVOCO**.

Teóricamente, para afirmar que una Gramática es unívoca, debería demostrarse que todas y cada una de las cadenas que produce, son generadas con un único posible árbol de derivación. Como esto no es posible, debe analizarse la estructura general de las composiciones. No basta con que una

sola cadena sea producida por un único árbol para afirmar que no existe ambigüedad.

EJEMPLO:

Dada la Gramática con $P = \{ S \rightarrow A \mid B, A \rightarrow a, B \rightarrow a \}$ con símbolo inicial S , determinar si es ambigua o unívoca.

Es ambigua, ya que se pueden producir dos árboles diferentes para producir la cadena a . Sólo cambia el nodo intermedio del árbol de derivación, que puede ser A ó B , razón que es suficiente.

Se puede demostrar que el idioma español es un Lenguaje ambiguo, a partir de una cadena válida en este Lenguaje, tal como "nada en el agua".

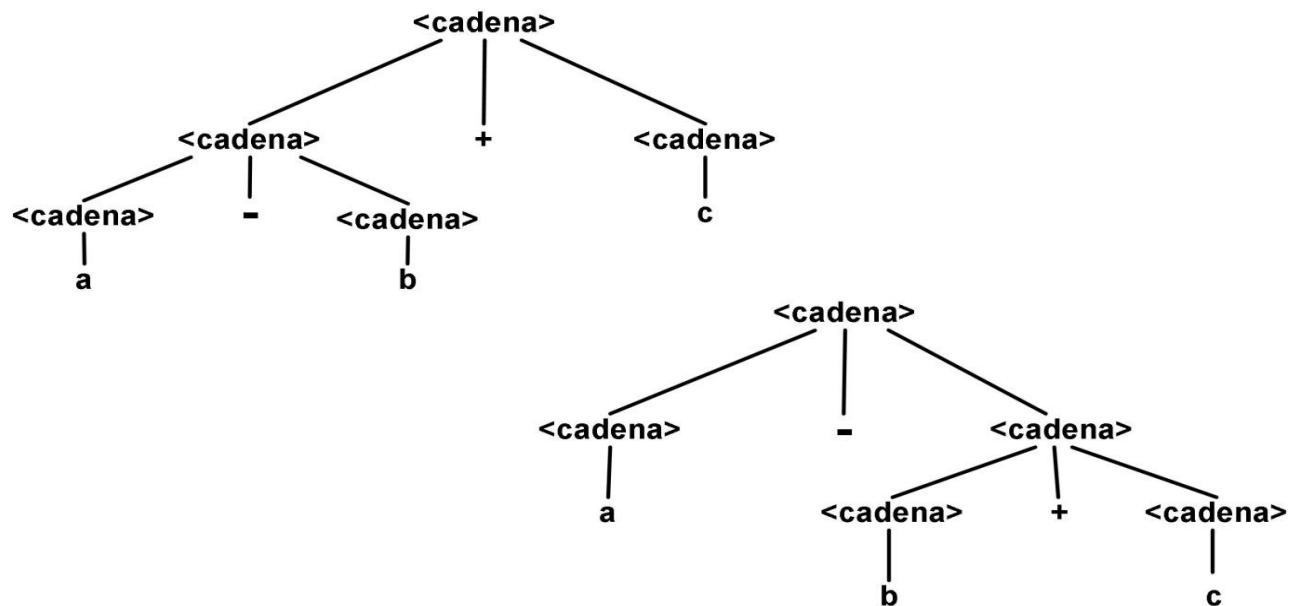
Para lo anterior, bastará con demostrar que para la mencionada cadena, o cualquier otra que sea una frase válida producida en ese idioma, existen dos diferentes árboles que la producen. Es suficiente con que un nodo cambie, para que un árbol sea diferente a otro.

De hecho, la clave está en que la palabra **nada**, pueda interpretarse como un adverbio de cantidad (ausencia de algo) o como un verbo (de la acción de nadar).

A propósito, obsérvese que las palabras del Lenguaje Formal, ya no son cadenas, como abbbcbbac, sino frases de un Lenguaje Natural y lo anterior con la misma teoría empleada en el capítulo anterior.

EJEMPLO:

Considérese la Gramática G con reglas $P = \{ \langle \text{cadena} \rangle ::= \langle \text{cadena} \rangle + \langle \text{cadena} \rangle \mid \langle \text{cadena} \rangle - \langle \text{cadena} \rangle \mid a \mid b \mid c \}$. Determinar si G es ambigua o unívoca.



Como la cadena $a - b + c$ se puede producir por cualquiera de los dos árboles, la Gramática G es ambigua. Para demostrarlo basta con una sola cadena de $L(G)$.

Sin embargo, la Gramática con las siguientes composiciones es unívoca:

$P = \{ \langle \text{cadena} \rangle ::= \langle \text{cadena} \rangle + \langle \text{id} \rangle \mid \langle \text{cadena} \rangle - \langle \text{id} \rangle \mid \langle \text{id} \rangle, \langle \text{id} \rangle ::= a \mid b \mid c \mid d \}$.

Queda para el alumno la demostración mediante el análisis de la Gramática considerando las cadenas que produce, tales como $a - b + c$.

FORMA NORMAL DE CHOMSKY (CNF)

CONCEPTO: Se presenta cuando el lado derecho en todas las reglas de producción de una Gramática Libre de Contexto consiste en un único símbolo terminal o exactamente dos no terminales.

Obsérvese que, para una Gramática en Forma Normal de Chomsky, el árbol de derivación para cualquier derivación está bastante bien construido ya que, excepto en las hojas, el árbol es binario.

EJEMPLO:

Mostrar dos Gramáticas equivalentes, una de ellas en CNF.

$$P_1 = \{ S \rightarrow xSy, S \rightarrow xy \}$$

$$P_2 = \{ S \rightarrow XM, S \rightarrow XY, M \rightarrow SY, X \rightarrow x, Y \rightarrow y \}$$

Se ve claramente que $L(G_1) = \{ x^+ y^+ \} = L(G_2) \therefore G_1 \sim G_2$.

Observar que G_2 está en Forma Normal de Chomsky.

TEOREMA

Si G es una Gramática Libre de Contexto, entonces existe otra Gramática Libre de Contexto G' expresada en CNF tal que G ~ G'.

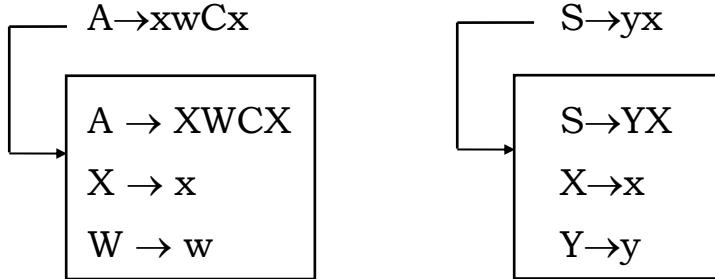
Se interpreta como que cualquier Gramática Libre de Contexto se puede transformar en otra equivalente y expresada en Forma Normal de Chomsky.

TRANSFORMACIÓN DE UNA GRAMÁTICA A LA CNF

PASO 1) Se transforman las composiciones que tienen al lado derecho combinaciones de terminales y no terminales, o bien donde haya más de un terminal. Para cada terminal a incluida en las producciones, se define un no terminal específico A , y la regla $A \rightarrow a$ con lo que se sustituye cada ocurrencia de a por A .

Después de aplicar este paso, todos los símbolos terminales se producen exclusivamente por una regla del tipo $A \rightarrow a$.

EJEMPLOS:



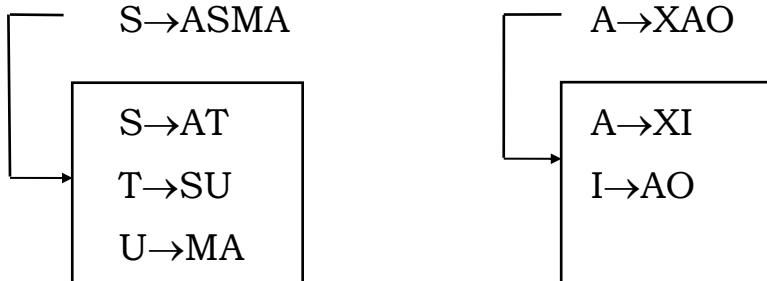
PASO 2) Se transforman las reglas en las que en el lado derecho hay más de dos no terminales. Se hace de la siguiente manera. La regla $N \rightarrow N_1 N_2 N_3 \dots N_t$ se sustituye por el encadenamiento de $t-1$ reglas:

$$N \rightarrow N_1 R_1 \quad R_1 \rightarrow N_2 R_2 \quad R_2 \rightarrow N_3 R_3 \dots \quad R_{t-2} \rightarrow N_{t-1} N_t$$

donde $R_1, R_2, R_3, \dots, R_{t-2}$ son símbolos no terminales de nueva aparición.

Después de aplicar este paso, todas las reglas en las que se producen no terminales serán del tipo $A \rightarrow BC$ o $A \rightarrow B$.

EJEMPLOS:



PASO 3) Se transforman las reglas que contengan a la derecha un solo símbolo no terminal. Se hace “heredando” al símbolo no terminal de la izquierda de la regla en cuestión, las composiciones que produce el no terminal de la derecha y se conservan estas últimas.

EJEMPLO:

Las composiciones originales:

$\mathbf{A \rightarrow B},$
 $B \rightarrow BD, B \rightarrow CC, B \rightarrow CA, B \rightarrow 1.$

Son actualizadas por las siguientes:

$$\boxed{\begin{aligned} & \mathbf{A \rightarrow BD, A \rightarrow CC, A \rightarrow CA, A \rightarrow 1,} \\ & \mathbf{B \rightarrow BD, B \rightarrow CC, B \rightarrow CA, B \rightarrow 1.} \end{aligned}}$$

La utilidad que tiene el expresar una Gramática Libre de Contexto en su Forma Normal de Chomsky se conocerá cuando se curse la asignatura de Compiladores.

También existe un tipo de estructuración para las Gramáticas Libres de Contexto en la llamada Forma Normal de Greibach (GNF). Existe un Teorema que expresa que cualquier Gramática Libre de Contexto se puede transformar en otra equivalente, expresada en la GNF. Se deja al alumno que investigue en qué consiste esta representación y su aplicación.

EJEMPLO:

Convertir la siguiente Gramática Libre de Contexto a otra equivalente, en su Forma Normal de Chomsky (CNF):

$$\begin{aligned} P = \{ & S \rightarrow zS \mid AB \mid C, & C \rightarrow BA \mid xA \mid yz, \\ & A \rightarrow CE \mid CDE \mid zD, & D \rightarrow xyz, \\ & B \rightarrow ABA \mid E, & E \rightarrow EE \mid zB \}. \end{aligned}$$

PASO 1:

$$\begin{aligned} P = \{ & S \rightarrow ZS \mid AB \mid C, & C \rightarrow BA \mid XA \mid YZ, \\ & A \rightarrow CE \mid CDE \mid ZD, & D \rightarrow XYZ, \\ & B \rightarrow ABA \mid E, & E \rightarrow EE \mid ZB, \\ & X \rightarrow x, & Y \rightarrow y, & Z \rightarrow z \}. \end{aligned}$$

PASO 2:

$$\begin{aligned} P = \{ & S \rightarrow ZS \mid AB \mid C, & C \rightarrow BA \mid XA \mid YZ, \\ & A \rightarrow CE \mid CM \mid ZD, & D \rightarrow X\tilde{N}, \\ & B \rightarrow AN \mid E, & E \rightarrow EE \mid ZB, \\ & X \rightarrow x, & Y \rightarrow y, & Z \rightarrow z, \\ & M \rightarrow DE, & N \rightarrow BA, & \tilde{N} \rightarrow YZ \}. \end{aligned}$$

PASO 3:

$$\begin{array}{ll} P = \{ & S \rightarrow ZS \mid AB \mid BA \mid XA \mid YZ, \quad C \rightarrow BA \mid XA \mid YZ, \\ & A \rightarrow CE \mid CM \mid ZD, \quad D \rightarrow X\tilde{N}, \\ & B \rightarrow AN \mid EE \mid ZB, \quad E \rightarrow EE \mid ZB, \\ & X \rightarrow x, \quad Y \rightarrow y, \quad Z \rightarrow z, \\ & M \rightarrow DE, \quad N \rightarrow BA, \quad \tilde{N} \rightarrow YZ \}. \end{array}$$

EJEMPLO:

Convertir la siguiente Gramática Libre de Contexto a su CNF equivalente.

$$\begin{array}{ll} P = \{ & A \rightarrow xy \mid Bx \mid xExz, \quad D \rightarrow xx \mid yB \mid xE, \\ & B \rightarrow zC \mid DE \mid C, \quad E \rightarrow FA \mid yCy \mid A, \\ & C \rightarrow DC \mid xyF \mid FA \mid y, \quad F \rightarrow AC \mid BB \}. \end{array}$$

Queda para que el alumno lo resuelva, aunque no aparezca en los problemas finales.

CAPÍTULO 2

LAS GRAMÁTICAS FORMALES EN LA COMPUTACIÓN

FORMA NORMAL DE BACKUS - NAUR (BNF)

Fue la primera notación formal cuyo empleo se ha generalizado para describir la sintaxis de los Lenguajes de programación y fue creada por John Backus en 1960. Los Lenguajes de alto nivel como Fortran, Pascal, C, C++, Java y Algol pueden representarse en BNF.

La notación se adoptó de inmediato para describir Algol 60. Después se supo que Panini había usado una notación similar a la BNF cinco siglos antes de Cristo para describir las complejas reglas de la Gramática del sánscrito.

Las siglas BNF comenzaron como abreviatura de Backus Normal Form, y después se transformó su interpretación en Backus - Naur Form, para reconocer las contribuciones de este último como editor del informe de ALGOL 60. Una variante surgida posteriormente fue la BNF Extendida. Estas dos representaciones, en combinación con los diagramas o esquemas de sintaxis (que son en sí reglas de producción en modo gráfico), pueden utilizarse para expresar la forma como se construyen las partes de los programas en un Lenguaje de programación de alto nivel y no son sino representaciones alternas de las reglas de producción de un Lenguaje Formal que se estudiaron en el módulo anterior.

LA BNF EN ESENCIA ES UN CAMBIO EN LA NOMENCLATURA.

Consiste en la aplicación de los siguientes tres criterios para modificar el aspecto que presenta la Gramática, cambiando la notación aprendida en el capítulo anterior:

1. Símbolos no terminales: Se delimitan con los picoparéntesis < >.
2. Composiciones: Se cambia la flecha de la producción por ::=.
3. Composiciones múltiples: Se pueden agrupar con el carácter | (que significa **o**), cuando varias reglas tienen el mismo valor en el lado izquierdo.

EJEMPLO:

Transformar la notación de la siguiente Gramática a la BNF correspondiente.

$$G = (\{R, S\}, \{a, b\}, \{R \rightarrow bR, R \rightarrow aS, S \rightarrow bS, S \rightarrow b\} , \{R\})$$

$$G = (\{<R>, <S>\}, \{a, b\}, \{<R> ::= b<R> \mid a<S>, <S> ::= b<S> \mid b\} , \{<R>\})$$

En el ejemplo anterior se empleó la conversión a la notación de BNF de una Gramática cuyo enfoque es más matemático que computacional para fines didácticos, lo cual resultó inadecuado, ya que se complicó la representación; nótese que la descripción en BNF resultó ser más extensa en su longitud.

En realidad a la BNF se la debe utilizar solamente en casos en los que para nombrar a los símbolos no terminales se deba describir su significado, en vez de emplear una sola letra mayúscula, es decir, cuando tiene un propósito computacional.

EJEMPLO:

Mostrar que la cadena $s = -318 \in L(G)$, donde $G = (N, T, P, S)$ con los conjuntos componentes siguientes:

$$N = \{ <\text{entero}>, <\text{entero con signo}>, <\text{entero sin signo}>, <\text{dígito}> \}$$

$$T = \{ +, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 \}$$

$$\begin{aligned} P = \{ & \quad <\text{entero}> ::= <\text{entero con signo}> \mid <\text{entero sin signo}>, \\ & <\text{entero con signo}> ::= + <\text{entero sin signo}> \mid - <\text{entero sin signo}>, \\ & <\text{entero sin signo}> ::= <\text{dígito}> \mid <\text{dígito}> <\text{entero sin signo}>, \\ & <\text{dígito}> ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \} \end{aligned}$$

$$S = \{ <\text{entero}> \} \quad // \text{ Indica lo que produce la Gramática.}$$

La Gramática genera todos los números enteros válidos y nada más. Por tanto produce, por ejemplo 1649, -65, +1000000, entre otros muchos, pero no -36.2, ni 12., ni 67+.

NOTA: Existen otras variadas Gramáticas equivalentes a la anterior para el mismo propósito. *De hecho la anterior tiene el grave inconveniente de que no es regular, cuando debiera serlo por cuestión de emplearse en una definición lexicográfica.*

Derivando a partir del símbolo inicial y basados en la descripción contenida en cada símbolo no terminal, se sabrá cómo debe ser la derivación sin posibilidad de equivocación.

$$\begin{aligned} <\text{entero}> & \Rightarrow <\text{entero con signo}> \Rightarrow -<\text{entero sin signo}> \\ & \Rightarrow -<\text{dígito}><\text{entero sin signo}> \Rightarrow -<\text{dígito}><\text{dígito}><\text{entero sin signo}> \\ & \Rightarrow -<\text{dígito}><\text{dígito}><\text{dígito}> \Rightarrow -3<\text{dígito}><\text{dígito}> \Rightarrow -31<\text{dígito}> \Rightarrow -318. \end{aligned}$$

Recomendación: Al símbolo inicial se lo debería denominar con una o varias palabras descriptivas sobre lo que se desea producir en el Lenguaje; por ejemplo, en este caso se lo llamó entero y ese es el tipo de dato que se obtiene al derivar.

La ventaja de emplear la BNF en estas Gramáticas, es que se puede describir el tipo de cadena que va a ser producida; esto se logra, dándole el nombre correspondiente al símbolo no terminal. No importa que la descripción sea muy extensa, ya que con los caracteres < y > se delimita lo que es un único símbolo; por ejemplo, <entero sin signo> es un único no terminal, descrito con tres palabras.

EJEMPLOS DE COMPOSICIONES DE LENGUAJE PASCAL EXPRESADAS EN BNF

```
<Programa> ::= <Cabecera> ; <Cláusula Uses> <Cuerpo de Programa>.  
<Sentencia> ::= <Sentencia Simple> | <Sentencia Compuesta>  
<Sentencia Compuesta> ::= Begin <Sentencia> End  
<Asignación> ::= <Identificador> := <Expresión>  
<Selectiva Simple> ::= If <Condición> Then <Sentencia>  
<Selectiva Doble> ::= If <Condición> Then <Sentencia> Else <Sentencia>  
<Selectiva Múltiple> ::= Case <Selector> Of  
    <Lista de Constantes>: <Sentencias 1>;  
    <Lista de Constantes> : <Sentencias 2>;  
    ...  
    [Else <Sentencia por Defecto> ]  
    End;  
<Para> ::= For <Índice> := <Valor Inicial> To <Valor Final> Do <Sentencia>  
<Mientras> ::= While <Condición> Do <Sentencia> ;  
<Repetir> ::= Repeat <Sentencia> Until <Condición>  
<Condición> ::= ( <Condición> ) | <Expresión> <Op_Relacional> <Expresión>
```

A los símbolos no terminales se les llama también constructores, y a los terminales tokens o componentes léxicos.

De la misma manera, cada Lenguaje de Programación tiene su colección de reglas de producción que definen su sintaxis, además de otras reglas donde se indica la forma como se construyen sus tokens o producciones léxicas. Esta colección de composiciones constituye el punto de partida para el diseño de un compilador de acuerdo a las especificaciones deseadas. También se pueden emplear eficazmente para comprender programas escritos en Lenguajes no conocidos por el estudiante y para comparar la sintaxis de las instrucciones en diversos Lenguajes de Programación.

TAREA PARA EL ESTUDIANTE

Investigar en alguna referencia válida en internet o en un texto sobre un Lenguaje de programación típico, acerca de la forma como se expresan sus reglas sintácticas, ya sea en BNF, BNFE o en diagramas de sintaxis, además de comprender cómo se pueden transformar estas representaciones entre sí.

EJEMPLO:

¿De qué frases consta la siguiente Gramática? Considerar como símbolo inicial a <oración>.

```
P = { <oración> ::= <sujeto> <predicado>,
       <sujeto> ::= <sustantivo>,
       <predicado> ::= <verbo intransitivo> | <verbo transitivo> <objeto>,
       <sustantivo> ::= Mario | Sandra,
       <verbo transitivo> ::= odia | golpea,
       <verbo intransitivo> ::= duerme,
       <objeto> ::= a <sustantivo> }
```

Respuesta:

L = { Mario odia a Mario.	Mario odia a Sandra.
Mario golpea a Mario.	Mario golpea a Sandra.
Sandra odia a Mario.	Sandra odia a Sandra.
Sandra golpea a Mario.	Sandra golpea a Sandra.
Mario duerme.	Sandra duerme. }

Nótese que la misma Teoría expuesta en este capítulo nos ha permitido producir un Lenguaje, ya no formado de palabras, sino ahora de oraciones o sentencias; en este caso, el Alfabeto contiene símbolos que son las palabras. De esa misma forma se pueden formar Lenguajes de párrafos o de otras cadenas muy diversas.

Por otra parte, se observa que aún los Lenguajes Formales *finitos*, tienen una Gramática Formal que los produce.

En el *Procesamiento de los Lenguajes Naturales*, área de la *Inteligencia Artificial* que nos permitirá conversar con un Agente inteligente inerte en nuestro propio idioma, se emplea la Teoría de los Lenguajes Formales para su diseño. Investíguese sobre el trabajo desarrollado por Noam Chomsky para establecer sus primeras bases.

Se recomienda también consultar sobre el tema *Agentes que se Comunican* en un texto sobre Inteligencia Artificial o en internet.

NOTA: Para el diseño de las reglas sintácticas del compilador en un Lenguaje de programación, se utiliza una Gramática no ambigua, aunque históricamente se ha dado el caso de emplear una ambigua con reglas adicionales para resolver las ambigüedades. Esta afirmación es muy importante, ya que al presentarse el caso de una ambigüedad, un analizador sintáctico podría confundirse al jerarquizar el orden en el que se realizan las operaciones.

Por ejemplo, en la Gramática del ejemplo anterior (donde había ambigüedad), en el primer árbol se indica que se hace primero la resta y luego la suma, al contrario de lo que se observa en el segundo.

Cuando un Árbol de Derivación se emplea para definir la sintaxis en una sentencia válida en un Lenguaje de programación, o bien en alguna expresión computacional y esto se muestra de una manera detallada para cada paso de la derivación, se le da el nombre de Árbol de Análisis Sintáctico.

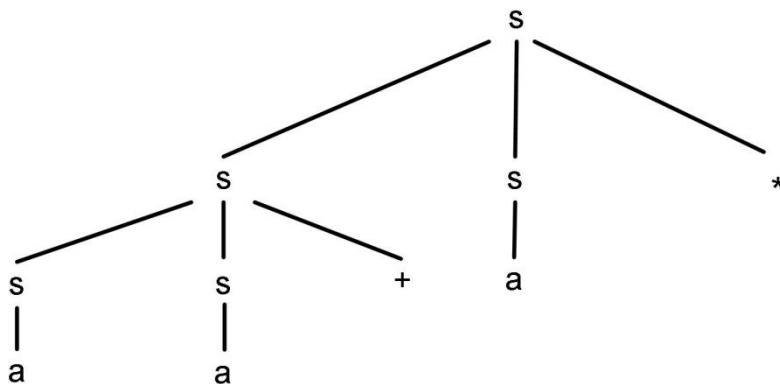
EJEMPLO:

Considérese la Gramática Libre de Contexto definida por las reglas de $P = \{S \rightarrow SS^* \mid a\}$.

a) ¿Cómo se puede generar la cadena $aa+a^*$ por medio de derivaciones sucesivas?

$$S \Rightarrow SS^* \Rightarrow SA^* \Rightarrow SS+A^* \Rightarrow SA+A^* \Rightarrow aa+A^* \Rightarrow aa+a^*.$$

b) Construir un árbol de derivación para esta misma cadena.



Este árbol de derivación constituye la única forma posible de producir dicha cadena. No existe otro árbol que la produzca.

Esta situación no es suficiente para afirmar que la Gramática, en general, será unívoca, ya que todas las cadenas que produzca la misma, deben ser producidas por un único árbol cada una de ellas.

La Gramática de este problema es única porque todas las cadenas que produce, se asocian a *un único árbol cada una*.

c) *Caracterizar, si es posible, la Gramática. ¿Con qué utilidad se la puede aplicar?*

Veamos primero la utilidad. Las cadenas producidas en este caso, se refieren a expresiones válidas que corresponden a la sintaxis que se conoce como NOTACIÓN POSFIJA, porque el operador aparece precisamente como tal.

En este ejemplo se han considerado sólo dos de las operaciones aritméticas básicas, pero podrían haberse contemplado también la resta y la división; no importa si se toman las dos o las cuatro operaciones, ya que las conclusiones que habremos de obtener son las mismas.

Algunas calculadoras de la marca HP emplean esta notación; para multiplicar 6 por 8, por ejemplo, se teclea la secuencia *6 Enter 8 Enter **, y se obtiene el resultado al oprimir el mencionado operador para la multiplicación.

La Notación Posfijo se emplea para definir expresiones en las que no existe ambigüedad en la interpretación sintáctica, aún sin el empleo de paréntesis a diferencia de la infija. Por ejemplo, en esta última nomenclatura, la expresión **a-b-c** puede tener dos diferentes formas de interpretarse. Como somos occidentales, y leemos de izquierda a derecha, nos parece natural que las operaciones se realicen de izquierda a derecha, es decir, primero se resta a-b y luego a este resultado se le resta c. Pero ese es un convencionalismo que la gente que no sabe mucho de matemáticas no descubre tan evidentemente, y menos si es un oriental de Japón o bien de Arabia, por ejemplo, donde se lee de derecha a izquierda. Si se desea que la interpretación sea la de restar primeramente b-c, entonces sí se deben emplear, definitivamente, los paréntesis.

Curiosamente, en la notación posfija, no se requieren los paréntesis para nada. Las dos interpretaciones posibles se pueden expresar sin que exista riesgo de confusión. En la expresión **a - b - c**, si se desea que primero se realice la resta de la izquierda ($a-b$) se debe expresar como **a b - c -**. Pero en cambio, si se desea indicar que primero se realiza la resta de la derecha que es $(b-c)$ entonces se tiene **a b c - -**. Sin necesidad de emplear paréntesis, una persona que conoce la notación, interpreta correctamente la sintaxis para evaluar la expresión correctamente, aunque sea japonés o árabe. Dicha situación no es de extrañar, ya que desde el inciso b) se había deducido que la Gramática era única y no ambigua.

Como se vio hace un par de páginas en este documento, la Gramática que sí es ambigua es la infija, misma que es la que usamos de manera habitual.

En caso de que el lector no entienda bien lo anteriormente expuesto, le conviene consultar más información sobre esta notación llamada posfija o polaca inversa.

La caracterización resulta más fácil al entender lo expuesto anteriormente. Sin haber analizado la utilidad, resultaría complicado el obtener una expresión regular para definir la respuesta. Aún ahora, parece ser que lo más conveniente

será caracterizar en Lenguaje natural y no en formal, y basados en nuestro conocimiento sobre la notación posfija, más que analizando las reglas gramaticales.

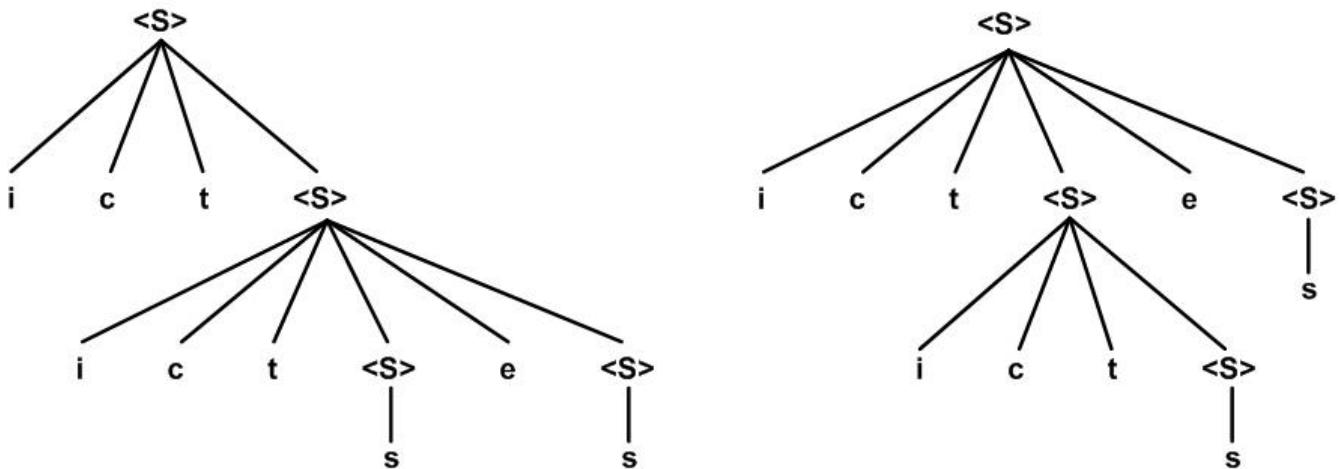
Las cadenas del Lenguaje producido, tienen las siguientes características:

- Inician con dos operandos al menos (las **a**) y terminan con operador (+ ó *), excepto la cadena a.
- Si contienen n operadores, entonces contendrán n+1 operandos.
- Todo posible prefijo de las mencionadas cadenas obtenidas en este Lenguaje, excluyendo a λ, contienen más operandos que operadores.

EJEMPLO:

Demostrar que la Gramática con $N = \{ S \}$, $T = \{ i, c, t, e, s \}$, $P = \{ S \rightarrow ictS \mid ictSeS \mid s \}$ y $S = \{ S \}$ es ambigua y explicar las implicaciones que tiene ese hecho.

Se hace la de mostración de una manera sencilla por medio de los siguientes dos árboles de derivación que generan la misma cadena **(ict)²ses**. En el diagrama se expresará S como símbolo no terminal delimitado por <> debido a que como se expondrá más adelante se trata de una definición sintáctica de un Lenguaje de Programación y en estos casos es recomendable emplear BNF.



Respecto a las implicaciones de la anterior ambigüedad se menciona que esta es una Gramática que corresponde a tres de las reglas del lenguaje de programación Pascal, donde **i** significa *if*, **c** sería *condición*, la **t** es por *then*, y la **e** es de *else*; se aclara que la **S** es para determinar *sentencia en construcción* y la **s** es para una sentencia ya específica, por ejemplo un write o un read, que son sendas instrucciones de Pascal. De esta forma se tiene que las composiciones corresponden a la Gramática con las siguientes:

$$P = \{ \text{<Sentencia>} ::= \text{if } \text{<condición>} \text{ then } \text{<Sentencia>} \mid \text{if } \text{<condición>} \text{ then } \text{<Sentencia>} \text{ else } \text{<Sentencia>} \mid \text{sentencia} \}.$$

Los dos árboles de análisis sintáctico corresponden a una misma expresión donde se anidan dos instrucciones *if*, una con *else* y la otra sin esta opción, pero empleando ambas en diferente orden. Como se verá puede presentarse el problema de la interpretación de a cuál de los dos *if* pertenece el segundo *else* para un programador en este lenguaje. La conclusión sería que Pascal sería un Lenguaje ambiguo.

Los fabricantes del compilador se dieron cuenta de este detalle y solucionaron el problema estableciendo el criterio de que cuando se presentara esta ambigüedad se considerara como que el segundo *else* corresponde a la instrucción *if* más cercana que no hubiera sido ya vinculada con una de esas instrucciones. Por esa razón se considera que de los dos árboles representados anteriormente, el de la izquierda sería el que el compilador considera como el correcto, ya que corresponde al segundo *if*.

REGLAS DE LENGUAJES DE PROGRAMACIÓN PARA OPERADORES ASOCIATIVOS POR DERECHA E IZQUIERDA

Los diversos operadores de los Lenguajes de programación se jerarquizan por el orden de aplicación, cuando en una expresión aparecen varios de ellos; ese nivel de jerarquía se enumera en tablas de operadores que se incluyen en textos sobre la materia. Además, cuando en una sentencia aparece varias veces un mismo operador, o bien diferentes pero de la misma jerarquía, se debe considerar la asociatividad; ésto es, que siempre se deberá operar iniciando por el que se encuentra más a la derecha o por el de más a la izquierda, según sea el caso, y seguir ese orden.

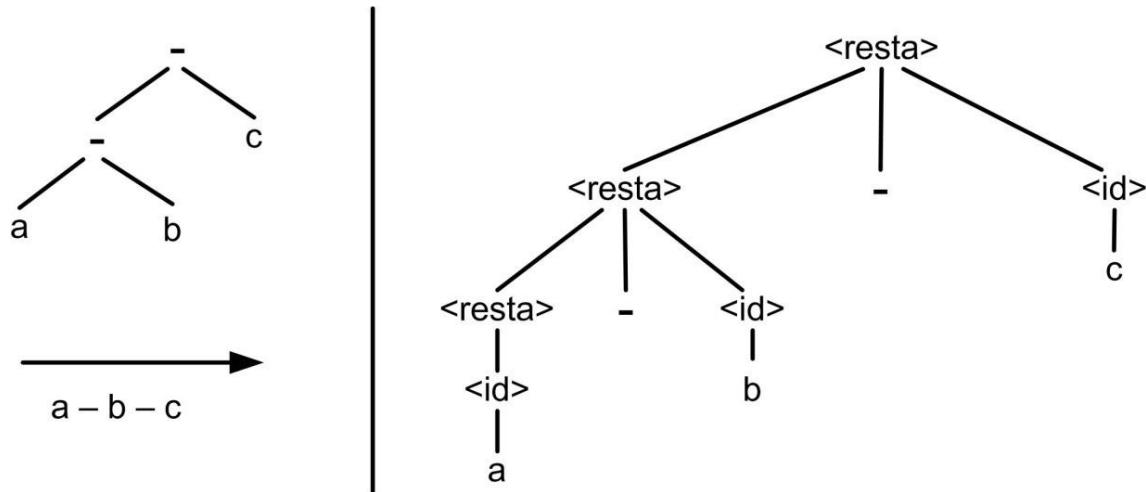
En este caso se abordará el problema de diseñar las reglas de producción correctas, considerando la asociatividad. Se justificará por partes, desde el planteamiento de los árboles correspondientes. Si no se diseñan las reglas en forma correcta, un compilador diseñado en base a ellas, realizará las operaciones en forma incorrecta, dando resultados erróneos al ejecutar un programa creado en ese Lenguaje.

Se tomarán como modelo dos típicas operaciones múltiples válidas en Lenguaje C, y se diseñarán los árboles, tanto el de sintaxis como el de análisis sintáctico, para que la asociatividad se realice correctamente; para simplificar el estudio, se considera un caso simple donde sólo aparece dos veces el operador. Este análisis se aplicará de manera similar para cualquier otra operación válida de un Lenguaje de programación, que tendrá que corresponder a alguno de los dos casos.

Posteriormente, en un problema propuesto, se propondrá un caso donde se deben jerarquizar las operaciones con diferente nivel de prioridad en una misma expresión.

EJEMPLO DE ASOCIATIVIDAD POR LA IZQUIERDA

Tomemos como referencia una operación múltiple en la que se presente la asociatividad por la izquierda. Todas las demás operaciones con este mismo caso, siguen el mismo esquema.

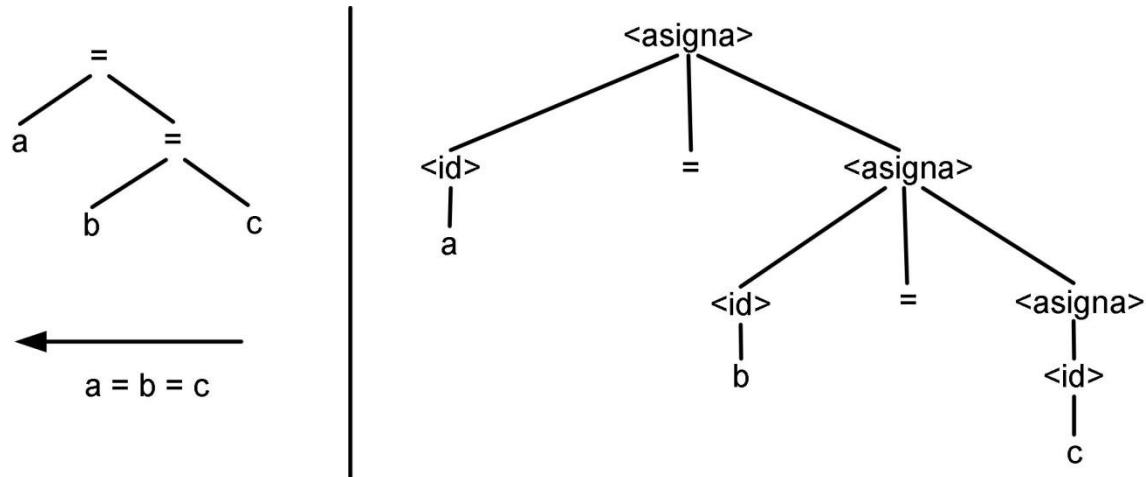


Reglas para sustracción múltiple (ASOCIATIVA POR LA IZQUIERDA):

$$P = \{ \langle \text{resta} \rangle ::= \langle \text{resta} \rangle - \langle \text{id} \rangle \mid \langle \text{id} \rangle, \quad \langle \text{id} \rangle ::= a \mid b \mid c \mid \dots \}$$

EJEMPLO DE ASOCIATIVIDAD POR LA DERECHA

Tomemos como referencia una operación múltiple en la que se presente la asociatividad por la derecha. Todas las demás operaciones con este mismo caso, siguen el mismo esquema.



Reglas para asignación múltiple (ASOCIATIVA POR LA DERECHA):

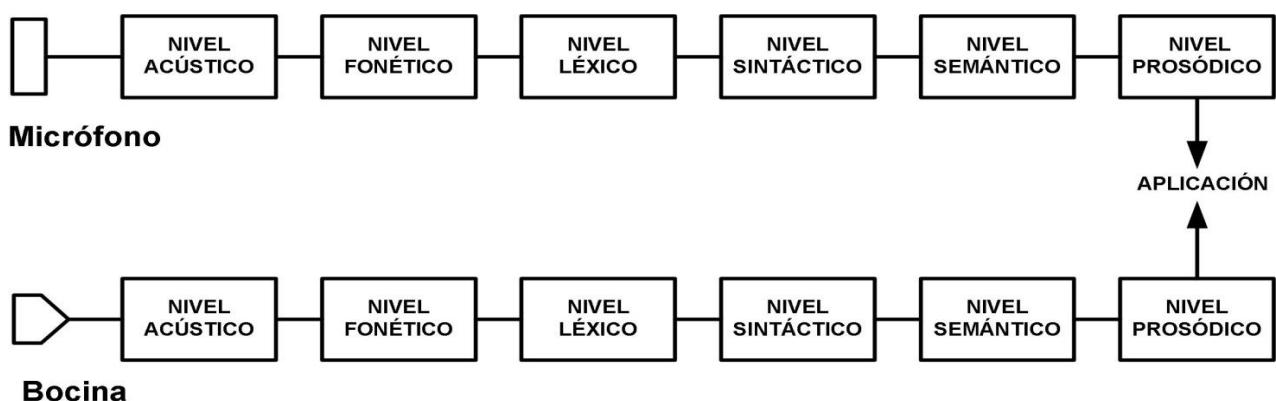
$$P = \{ \langle \text{asigna} \rangle ::= \langle \text{id} \rangle = \langle \text{asigna} \rangle \mid \langle \text{id} \rangle, \quad \langle \text{id} \rangle ::= a \mid b \mid c \mid \dots \}$$

Para ambos casos se realiza el diseño del árbol de sintaxis, posteriormente el de análisis sintáctico y finalmente las composiciones obtenidas.

APLICACIÓN DE LOS ANALIZADORES DE LENGUAJES

A continuación, se muestra un diagrama de un potencial proyecto, en el que se podría dar una aplicación más amplia a los temas de este capítulo, aclarando que se le puede dar un enfoque diverso, ya que no solamente los compiladores para un Lenguaje de programación contienen las etapas de análisis que se han de mencionar más adelante.

La aplicación indicada podría ser, por ejemplo, una silla de ruedas activada por comandos de voz; antes del nivel acústico superior se ubica un micrófono como receptor. Cuando se recibe una entrada que no corresponde a una instrucción válida, no es aceptada.



TEORÍA DE COMPILADORES

Las Gramáticas tipo 0 y 1 muestran un buen número de propiedades indecidibles. En las primeras, por ejemplo, puede suceder que no se pueda determinar algorítmicamente si una cadena dada es producto de ella o no; es decir, no se podría saber si una Máquina de Turing diseñada para reconocer esas cadenas se detendrá o no durante el transcurso de esa computación. En este texto no son considerados los capítulos correspondientes a sus Autómatas y el empleo de ambas Gramáticas se considera esporádicamente.

La Teoría de la Computabilidad facilita descubrir muchas propiedades de los Lenguajes y sus reconocedores, y ha permitido también el diseño de los Lenguajes de programación y de sus correspondientes Autómatas encargados de reconocerlos y traducirlos a expresiones más simples: los analizadores de los compiladores.

Para diseñar las etapas de análisis de un compilador, es necesario saber diseñar e implementar los Autómatas de este curso, ya que el analizador léxico es un Autómata Finito y los analizadores sintáctico y semántico son diseñados en base a Autómatas de Pila.

La Gramática que describe la sintaxis de los Lenguajes de programación es Libre de Contenido. Así mismo, el analizador lexicográfico de un compilador es un Autómata Finito, porque la Gramática que se emplea en la definición léxica de un Lenguaje como C, C#, Java, o Fortran es Regular. Se puede diseñar una Gramática de ese tipo que mostraría como se pueden obtener los nombres de los componentes

léxicos para alguno de esos Lenguajes, con excepción de las palabras reservadas, mismas que se analizan por otro mecanismo.

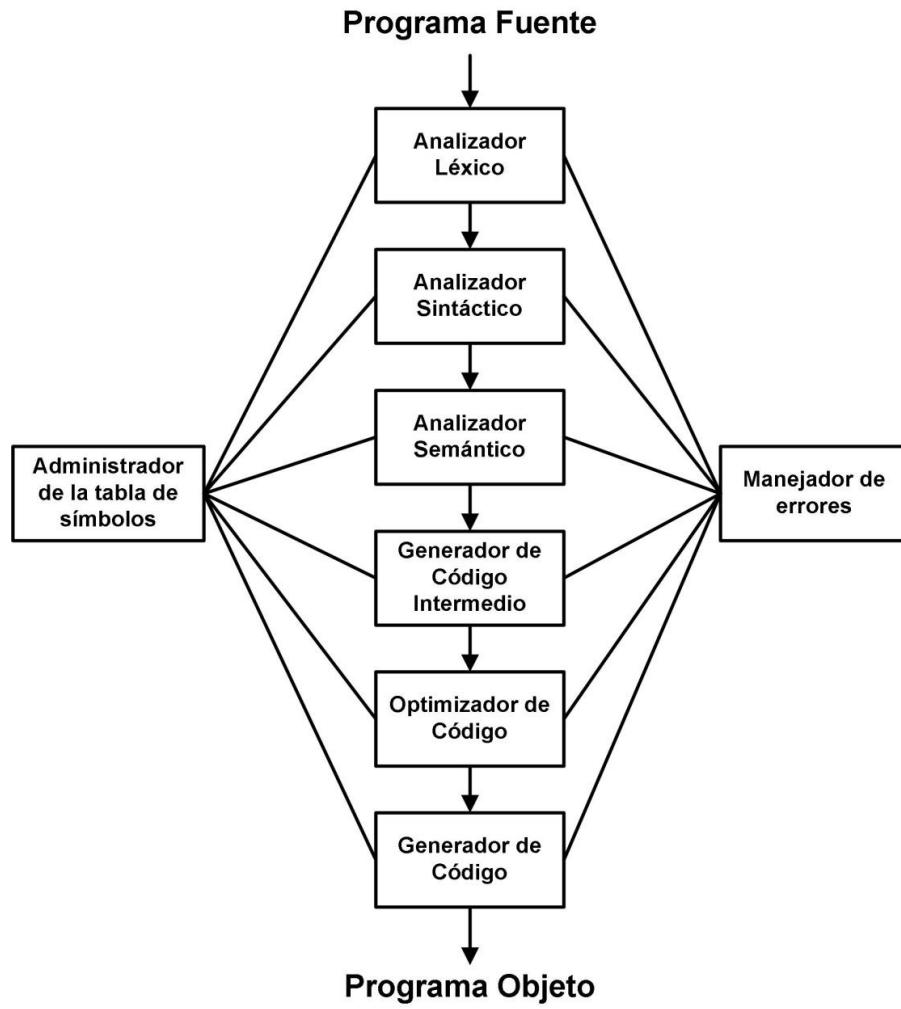
El conocimiento sobre los conceptos teóricos de los Lenguajes Formales se remonta a mediados del siglo XX, antes de que se desarrollaran los modernos Lenguajes de programación.

Como dato interesante, se puede comentar que el primer compilador de Fortran requirió para su implantación de 18 años-hombre de trabajo en grupo. Con las nuevas técnicas, estudiadas en este curso, un compilador de calidad aceptable se puede diseñar en un curso de un semestre.

No es posible diseñar las etapas de análisis de un compilador sin el conocimiento de los temas estudiados en este curso. Precisamente, el curso de Teoría de la Computación es prerequisito para cursar la materia de Compiladores.

En la siguiente figura se muestran las etapas de un compilador, consideradas por Alfred V. Aho en su texto sobre esta materia.





TOKEN O COMPONENTE LÉXICO: Es cualquiera de los símbolos terminales que se manejan en la construcción de sentencias de un programa en un Lenguaje de alto nivel. Se define como un par ordenado:

Token = (Tipo, Valor).

EJEMPLO:

Sea **discrim = b * b - 4 * a * c ; if (discrim > 0)**. Determinar los tokens de este fragmento de programa en Lenguaje C.

(Identificador, discrim)	(operador, =)	(identificador, b)
(operador, *)	(operador, -)	(constante, 4)
(identificador, a)	(identificador, c)	(separador, ;)
(palabra reservada, if)	(agrupación izq., ()	(op. relación, >)
(constante, 0)	(agrupación der.,))
...		

CASO DE ESTUDIO

Hacer un análisis de la línea de código que forma parte de un programa, expresada como **tiempo = minutos + segundos / 60**, y explicar cómo se va procesando en cada una de las etapas de un compilador. Enmarcada se irá mostrando la transformación que va sufriendo la cadena de entrada al ir pasando por cada etapa de compilación.

ANALIZADOR LÉXICO

Función: Comprobar la correcta construcción de cada uno de los tokens. A ciertos componentes léxicos, como los identificadores, se les introduce en la tabla de símbolos, si es que no están ya incluidos.

Errores típicos:

Identificadores mal estructurados: 4AC, aga@quantum, yo.tu, posición, etc.

Constantes mal estructuradas: +4.6789, 6900-, 90+25, etc.

Operadores mal estructurados: :=, =>, <>, etc. // todos ellos en C

Palabras reservadas mal estructuradas: PRINTF, print, suitch, etc.

Tabla de Símbolos						
tiempo = minutos + segundos / 60						
✓	✓	✓	✓	✓	✓	✓
id₁	=	id₂	+	id₃	/	60
...

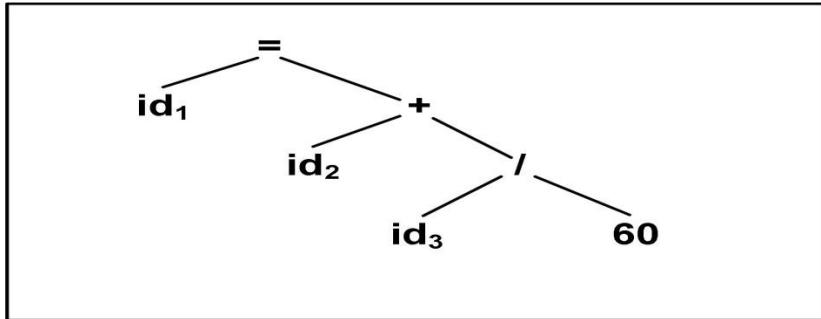
ANALIZADOR SINTÁCTICO

Función: Comprobar el correcto acomodo de los tokens dentro del código. En esta etapa se genera el árbol sintáctico de la expresión.

Errores típicos:

Paréntesis mal anidados: ((() , ()) (()), () (()) (()), etc.

Expresiones incorrectas, tales como:
c , a + b = c, a + b * -



ANALIZADOR SEMÁNTICO

Función: Evalúa el programa fuente para tratar de encontrar errores en los tipos de datos que se emplean y reúne información sobre los mismos. Si fuera necesario, hace un ajuste de los mismos, siempre que no afecte al algoritmo en sus resultados.

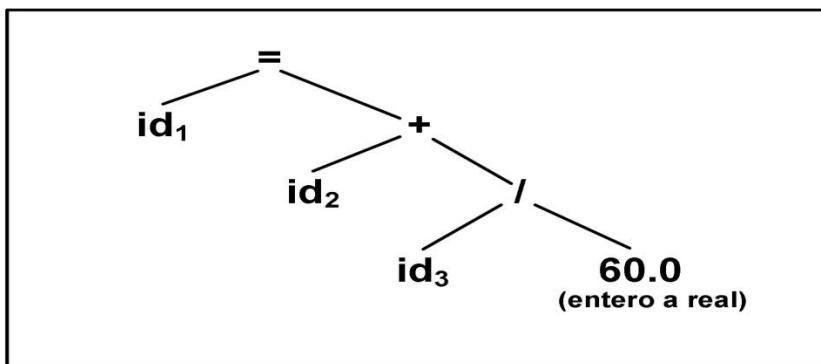
Errores típicos:

Los indicados con el mensaje *Type mismatch*.

Cuando se usa una variable real como argumento de un *for*, o como subíndice de una matriz.

Si se suman matrices bidimensionales con vectores que son unidimensionales.

Si en la instrucción alternativa múltiple (como *switch*) un mismo valor de la constante corresponde a dos rangos diferentes.



GENERADOR DE CÓDIGO INTERMEDIO

Función: Algunos compiladores producen una representación intermedia explícita del programa fuente que se puede considerar como un programa para una máquina abstracta. Se propone como Lenguaje el llamado *código de tres direcciones*, en el que cada instrucción tiene como máximo tres operandos (considerando inclusive la asignación).

```

temp1 := enteroareal(60)
temp2 := id3 / temp1
temp3 := id2 + temp2
id1 := temp3

```

OPTIMIZADOR DE CÓDIGO

Función: Sirve para mejorar el código intermedio, de modo que se obtenga un código de máquina más rápido de ejecutar. Existe mucha variación en la magnitud de la optimización que realizan los distintos compiladores, pudiendo suceder que algunas optimizaciones resultan triviales. Es importante que el diseñador especifique qué entiende por este término.

Considérese el siguiente ejemplo donde se compara cómo se tienen dos líneas de código, antes y después de la optimización. En este segundo caso se ahorra una operación de multiplicación:

Antes de optimizar:

```

...
temp4 := id8 * temp2
...
temp6 := id8 * temp2
...

```

Después de optimizar:

```

...
temp4 := id8 * temp2
...
temp6 := temp4
...
```

Para el caso de estudio:

```

temp1 := id3 / 60.0
id1 := id2 + temp1

```

GENERADOR DE CÓDIGO

Función: Produce el código objeto, que por lo general consiste en código relocalizable o ensamblador. Las posiciones de memoria se seleccionan para cada una de las variables usadas por el programa; después, cada una de las instrucciones intermedias se traduce a una secuencia de instrucciones de máquina que ejecuta la misma tarea.

	destino, origen
MOVF	Bx, id3
DIVF	Bx, #60.0
MOVF	Ax, id2
ADDF	Ax, Bx
MOVF	id1, Ax

Se recomienda al estudiante que consulte mayor información sobre el tema de diseño de las etapas de análisis de un compilador en algún texto orientado a esta asignatura.

CAPÍTULO 3

MÁQUINAS DE ESTADO FINITO

DEFINICIÓN FORMAL

Es un Modelo abstracto de una máquina con memoria interna primitiva muy básica, que se representa como **M** y que consiste en:

- a) Un conjunto finito I de símbolos de entrada.
- b) Un conjunto finito O de símbolos de salida.
- c) Un conjunto finito S de estados.
- d) Una función estado siguiente f de $S \times I$ en S .
- e) Una función salida g de $S \times I$ en O .
- f) Un estado inicial σ_0 , incluido en S .

Se expresa con una *descripción formal* con la 6-tupla $\mathbf{M} = (I, O, S, f, g, \sigma_0)$.

En este caso, los símbolos de entrada y salida no deben ser necesariamente diferentes, sino más bien según sea requerido en la especificación del diseño. Nótese que se trata de símbolos, como los que se emplearon en los capítulos anteriores.

DESCRIPCIÓN DE LA MÁQUINA DE ESTADO FINITO

Se trata de un modelo matemático, representado con recursos formales (que se especificarán posteriormente) y que puede emplearse para representar o **simular** el funcionamiento de un sistema real, que puede ser electrónico o computacional o de otro tipo. Esto es muy útil, ya que posteriormente al diseño formal, se puede **implementar** en forma sencilla por medio de un programa escrito en cualquier Lenguaje de programación.

Los sistemas que se pueden representar por medio de este modelo matemático contienen una única entrada y una sola salida, y son todos aquellos en los que no basta con conocer el valor de la entrada para conocer la salida. En este caso, existe un parámetro muy importante, llamado estado actual del sistema, y que en combinación con la entrada, ambos sí determinan una salida bien definida.

En una Máquina de Estado Finito, ocurre lo que se conoce como una **Transición**, una acción que consta de tres partes. Consiste en que al llegar un símbolo de entrada, el modelo responda primero con la producción de un símbolo en la salida y posteriormente (pero de manera inmediata) exista un cambio a un

nuevo estado que podría ser el mismo. Dicha *transición* es la unidad de operación en una Máquina de Estado Finito.

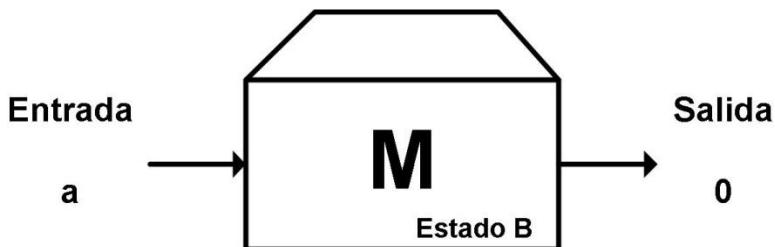
Es muy importante que primero se produzca la salida, antes del cambio de estado.

En este modelo se presentan dos funciones, ambas de las mismas dos variables, de las cuales una será llamada f y la otra g :

Símbolo de salida = f (Estado actual, Símbolo de entrada) = g // Función g .

Estado siguiente = f (Estado actual, Símbolo de entrada) = f // Función f .

Veamos un ejemplo, como si la Máquina fuera una "caja negra":



En la Máquina de la figura anterior supondremos, para ejemplificar, que si la entrada es una **a**, y el estado actual fuese **B**, la salida sería un **0**. Sin embargo, si el estado actual fuese **D**, por ejemplo, la salida podría ser **2**, *aún con la misma entrada a*.

Se recalca que se emplea este modelo para SIMULAR todos aquellos sistemas en los que la salida no es función exclusiva de la entrada, sino también de un estado o situación actual implícita en el mismo.

EJEMPLOS DE APLICACIONES DE MÁQUINAS DE ESTADO FINITO:

1 **Máquina para venta de refrescos.**

Ya que el *estado actual* (cantidad de dinero que se ha depositado hasta el momento en ella) junto con la *entrada* actual (la denominación de la moneda que se está depositando en un cierto momento) determinan la *salida* que el aparato entrega (nada, cambio, refresco, refresco y cambio). La misma moneda produce diferentes *salidas* según el estado en el que se encuentra la máquina.

2 **Cajero automático.**

En este caso la *entrada* (tarjeta, NIP, botones en el tablero) debe combinarse con una correcta serie de transiciones por los *estados* (espera usuario, espera NIP, espera requisición de servicio, etc.) para que se pueda obtener la *salida* deseada (billetes, impresiones de estado de cuenta, etc.). La *entrada* "solicito \$100" en el *estado* "espera NIP" no produce el efectivo requerido.

3 Circuitos secuenciales digitales.

Pues en los circuitos de este tipo es necesario conocer el potencial o voltaje que guardan ciertos puntos dentro de los mismos (es decir, los *estados*) para que en combinación con los valores de los bits de *entrada*, se conozcan los de *salida*. En un simple flip-flop, se requiere de conocer el *estado* del mismo para saber qué *salida* presenta.

4 Juegos de combate.

Al simular una pelea en un juego de video, se debe considerar que las *entradas* (golpes, agresiones, recuperaciones y otros más) en combinación con el *estado* (nivel de energía del peleador) determinan el efecto que se manifestará como *salida* (tambalearse, caerse, sangrar, ser noqueado, etc.) y al igual que en los casos que se mencionan anteriormente se actualiza el estado actual.

EJEMPLO:

Representar la Máquina de Estado Finito de la siguiente descripción formal por conjuntos, por medio de un diagrama de transición y por una tabla de transición.

$$\begin{array}{llll} I = \{ a, b \} & O = \{ x, y, z \} & S = \{ q_0, q_1, q_2 \} & \sigma_0 = \{ q_0 \} \\ f = \{ f(q_0, a) = q_1, & f(q_0, b) = q_2, & & \\ & f(q_1, a) = q_2, & f(q_1, b) = q_1, & \\ & f(q_2, a) = q_0, & f(q_2, b) = q_1 \} & & \\ g = \{ g(q_0, a) = x, & g(q_0, b) = y, & & \\ & g(q_1, a) = x, & g(q_1, b) = z, & \\ & g(q_2, a) = z, & g(q_2, b) = y \} & & \end{array}$$

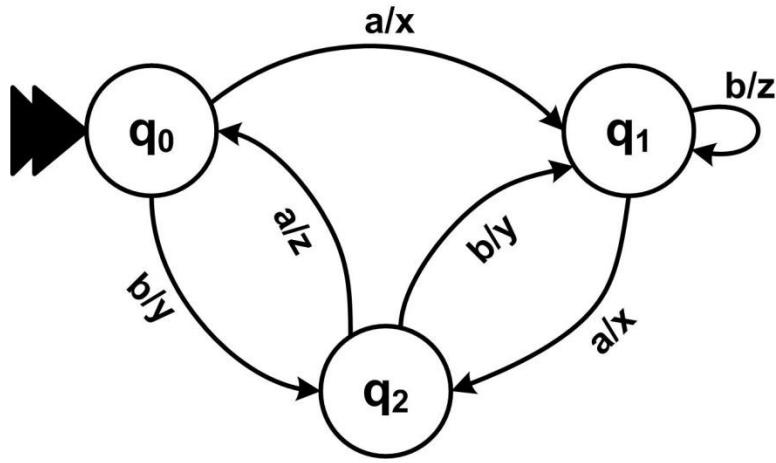
Las funciones **f** y **g** deben ser especificadas para cada combinación de cualquier símbolo de entrada con cada estado actual posible.

Es muy común denominar a los estados con una **q** y un subíndice, considerando que q_0 sería el estado inicial, aunque este criterio es mas bien una costumbre y no una regla.

DIAGRAMA DE TRANSICIÓN DE LA MÁQUINA DE ESTADO FINITO

Es un grafo dirigido G , en donde los nodos son los estados; el estado inicial se señala con una flecha gruesa sin origen. Si se parte del estado q_m y una entrada **i** proporciona una salida **o**, y se cambia al estado q_n , se traza un arco dirigido de q_m a q_n y se marca con una ponderación o peso como **i / o**.

Para este ejemplo se tiene el siguiente diagrama de transición:



Esta descripción gráfica es muy útil, ya que permite observar fácilmente las transiciones que se presentan en la Máquina. Considérese el siguiente análisis, donde se hace un *recorrido del digrafo* y en donde cada columna representa una transición:

Cadena de entrada: $S_{in} = a \ b \ b \ a \ a \ a \ a \ b$

Cadena de salida: $S_{out} = x \ z \ z \ x \ z \ x \ x \ y$

Estados recorridos: $q_0 \ q_1 \ q_1 \ q_1 \ q_2 \ q_0 \ q_1 \ q_2 \ q_1$

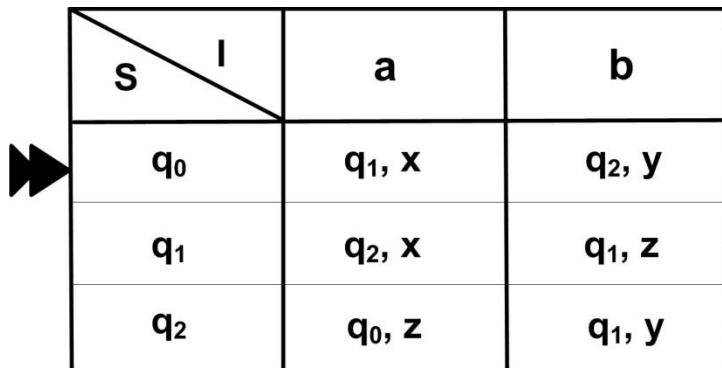
Una ventaja significativa del diagrama de transición es que el comportamiento de la Máquina de Estado Finito se puede analizar en una forma sumamente "amigable" y sencilla.

Por cada símbolo de entrada, hay un símbolo de salida producido.

TABLA DE TRANSICIÓN DE LA MÁQUINA DE ESTADO FINITO

Se hace la tabla, en la cual se consideran todas las combinaciones de $S \times I$ colocando en la columna izquierda los estados y en el renglón superior las entradas. En la intersección de los valores particulares de esos dos valores, se coloca el que será el siguiente estado y el símbolo de salida, separados por una coma; en ocasiones, se separan los valores de las funciones f y g en tablas contiguas separadas. Se indica de manera especial el estado inicial con una flecha gruesa \Rightarrow .

A continuación se muestran los dos formatos empleados para la tabla de transición.



A diagram illustrating the conversion of a state transition table into a state transition diagram. On the left, there is a large black arrow pointing from the top table to the bottom one. The top table has three columns labeled 'a' and 'b'. The first column contains states q_0 , q_1 , and q_2 . The second column contains sets q_1, x , q_2, x , and q_0, z . The third column contains sets q_2, y , q_1, z , and q_1, y . The bottom table has five columns labeled 'f' and 'g'. The first column contains states q_0 , q_1 , and q_2 . The second column contains states q_1 , q_2 , and q_0 . The third column contains states q_2 , q_1 , and q_1 . The fourth column contains symbols x , x , and z . The fifth column contains symbols y , z , and y .

	a	b
s	q_0	q_1, x
i	q_1	q_2, x
	q_2	q_0, z
		q_1, y

	f	g		
s	a	b	a	b
i	q_1	q_2	x	y
	q_2	q_1	x	z
	q_0	q_1	z	y

Las dos variaciones anteriores representan la tabla de transición; el alumno deberá elegir la que le resulte más adecuada. Esta representación es necesaria para la implementación del software que corresponde a la Máquina, ya que resulta obvio que es más factible programar un arreglo, que un gráfico.

Además es imprescindible en los casos en los que se cuenta con muchos estados y muchas entradas, y se corre el riesgo de que un diagrama de transición se convierta en una auténtica "telaraña" por tantos trazos.

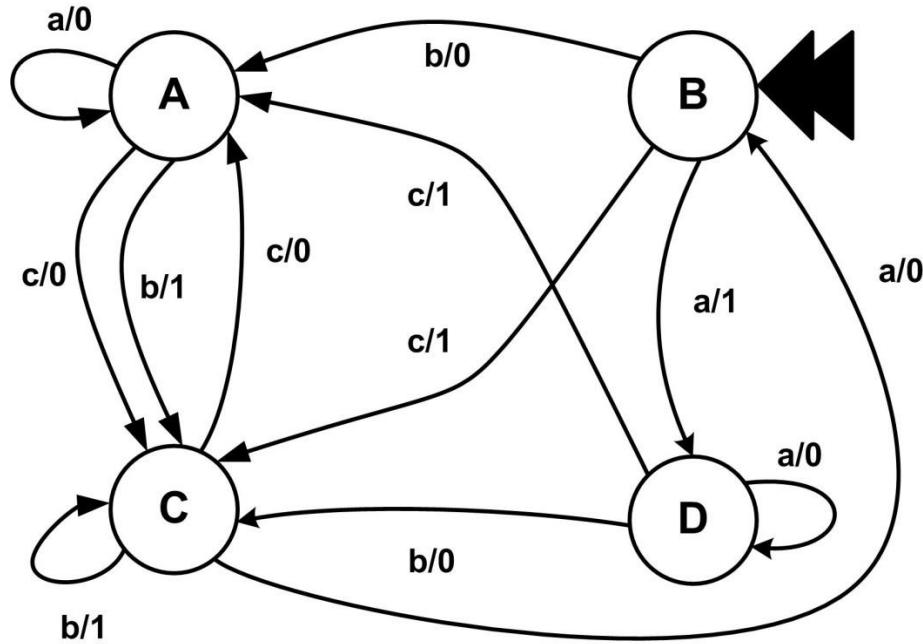
Dada una cualquiera de las tres representaciones posibles de M debe ser posible, sin ambigüedades, de obtenerse las otras dos.

EJEMPLO:

Diseñar el diagrama de transición y los seis conjuntos que definen formalmente a la Máquina de Estado Finito de la siguiente tabla de transición:

$S \setminus I$	a	b	c
A	A, 0	C, 1	C, 0
B	D, 1	A, 0	C, 1
C	B, 0	C, 1	A, 0
D	D, 0	C, 0	A, 1

Primeramente se tiene el siguiente Diagrama de transición equivalente:



Por otra parte se tiene que $M = (I, O, S, f, g, \sigma_0)$, en donde los conjuntos contienen:

$$I = \{a, b, c\} \quad O = \{0, 1\} \quad S = \{A, B, C, D\} \quad \sigma_0 = \{B\}$$

$$f = \{f(A, a) = A, f(A, b) = C, f(A, c) = C, f(B, a) = D, f(B, b) = A, f(B, c) = C,$$

$$f(C, a) = B, f(C, b) = C, f(C, c) = A, f(D, a) = D, f(D, b) = C, f(D, c) = A\}$$

$$g = \{g(A, a) = 0, g(A, b) = 1, g(A, c) = 0, g(B, a) = 1, g(B, b) = 0, g(B, c) = 1,$$

$$g(C, a) = 0, g(C, b) = 1, g(C, c) = 0, g(D, a) = 0, g(D, b) = 0, g(D, c) = 1\}$$

En resumen: Es muy común denominar a los estados con una q y un subíndice, pero no es una regla, como se acaba de observar. Tampoco es obligado que el primer estado en orden de aparición sea el inicial. Los símbolos de entrada y los de salida

no tienen necesariamente una relación entre ellos y pueden ser iguales o diferentes entre sí.

EJEMPLO:

Diseñar una Máquina de Estado Finito, considerando como dato de entrada posible cualquier cadena de bits válida. Si el dato de entrada contiene una cantidad par de 1 la salida es 1; 0 en caso contrario.

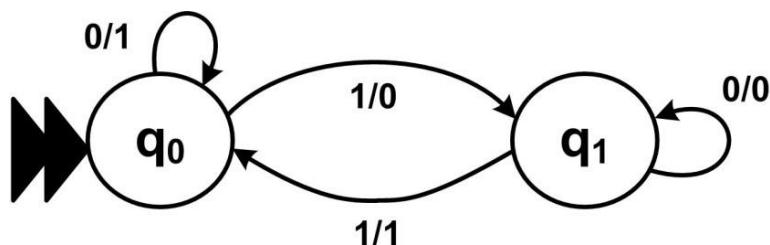
En este caso, emplearemos la notación más amigable, que es la del diagrama de transición. Consideremos que los bits de la cadena de entrada van llegando uno por uno (en serie) ya que la Máquina de Estado Finito contiene una única entrada y no recibe varios bits a la vez.

Es importante definir los estados, como punto de partida; en este caso existen dos posibles: o la cantidad de unos es par o dicha cantidad es impar.

La cantidad de ceros en la entrada es totalmente intrascendente, por lo que no hay cambio de estado alguno; éstos se traducen en lazos en el diagrama, sin importar cuál es el estado actual.

Sin embargo, la cantidad de unos en la entrada, sí es extremadamente importante, ya que son los que se nos pide que cumplan con ser una cantidad par. Es evidente que en estos casos sí debería de haber cambio de estado actual, de un estado al otro.

Llamemos q_0 al estado en el cual la cantidad de unos es par, y q_1 al estado donde es impar dicha cantidad. Al inicio tenemos cero unos, y siendo cero un número par, definamos a q_0 como el estado inicial.



Al iniciar el funcionamiento de la Máquina, la salida debe ser 1, ya que sí se cumple la condición; mientras que no exista ningún 1, sigue estando en esa condición. Cuando aparece el uno, la salida se convierte en 0, y así se quedará hasta que vuelva a aparecer otro 1, quedando la situación igual que al principio (tener 0, 2, 4, 6, 8, ... unos, es equivalente, ya que siempre se cumple con una cantidad par).

Finalmente, hay que observar ***cuál es el bit de salida, cuando llega el último bit de entrada.***

EJEMPLO:

Diseñar una Máquina de Estado Finito que represente un sumador binario en serie.

Primeramente habrá que considerar que la mencionada realizará una suma de cadenas de bits, en la que éstos se sumarán de dos en dos a la vez, uno del primer sumando y el otro del segundo. La longitud de las dos cadenas a sumar es la misma para este ejemplo y puede ser cualquier cantidad de bits. Un ser humano realiza la suma en serie cuando la realiza sin calculadora, evaluando uno por uno, cada bit del resultado. Veamos un ejemplo de una suma binaria en serie, como muestra:

$$\begin{array}{r}
 & & 1 & 1 \\
 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & + \\
 & \underline{1} & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1
 \end{array}$$

Ejemplo de una suma binaria en serie

Nótese que al sumar dos bits, el bit resultado varía; por ejemplo al hacerlo con $0 + 0$, tenemos que da 0 (cuando no hay acarreo) ó 1 (cuando sí hay un acarreo). El existir o no acarreo representa un *estado del sistema*, que influye en el bit de salida.

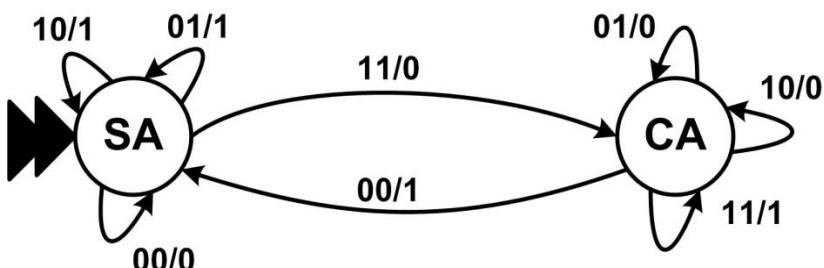
De acuerdo al conocimiento, tanto de la operación de la suma como de las Máquinas de Estado Finito, se determinan los conjuntos:

$$I = \{ 00, 01, 10, 11 \} \quad O = \{ 0, 1 \} \quad S = \{ SA, CA \} \quad \sigma_0 = \{ SA \}$$

Recuérdese que un símbolo de entrada se puede formar por más de un carácter, por lo que las entradas pueden expresarse con un par de caracteres, que representan los bits que se están sumando (por ejemplo 00 significa $0 + 0$). Este hecho permite que se puedan tener Máquinas que representen sistemas en los que hay varias entradas, pero canalizadas todas ellas como una sola. Si no fuese así, entonces tendríamos que definir modelos de Máquinas con varias entradas, lo cual sería un tanto complicado.

En este caso SA significa "Sin Acarreo" y CA es por "Con Acarreo". Cuando empezamos a sumar lo hacemos desde un estado inicial sin acarreo. Cuando se suman dos bits el resultado es un solo bit y si el resultado excede tal longitud, de todas formas se escribe un solo bit y se pasa a un estado con acarreo.

Finalmente, resulta el siguiente diagrama de transición para el sumador:



Es muy recomendable que el alumno, una vez que resuelva un problema, haga una comprobación del funcionamiento del modelo formal, para ver si funciona con diversos y muy variados *casos de prueba*. He aquí la verificación de la suma ya anteriormente evaluada, ahora con el Diagrama de Transición que se acaba de diseñar:

$$\begin{array}{l}
 \text{cadena}_{\text{in}} = \quad 0 \ 1 \quad 1 \ 1 \quad 0 \ 1 \quad 0 \ 0 \quad 1 \ 0 \quad 0 \ 1 \quad 0 \ 0 \quad 1 \ 1 \\
 \text{cadena}_{\text{out}} = \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \\
 \text{SA} \quad \text{SA} \quad \text{CA} \quad \text{CA} \quad \text{SA} \quad \text{SA} \quad \text{SA} \quad \text{SA} \quad \underline{\text{CA}}
 \end{array}$$

La versión previa de sumador tiene el inconveniente de que el bit más significativo se pierde si el resultado excede en un bit a la longitud de los sumandos. Lo anterior no es un problema serio, ya que en algunos casos así sucede con el modelo real que estamos emulando; por ejemplo, el chip de un circuito integrado sumador digital tiene como terminales o pines para cada entrada, la misma cantidad que la de los pines donde se muestra la suma de las dos cadenas de bits, mostrando el posible bit adicional más significativo en el pin de acarreo (que no es propiamente una terminal de dato del resultado en el sumador). El circuito que realiza la función de sumar dos cadenas de cuatro bits es el 7483 en tecnología TTL.

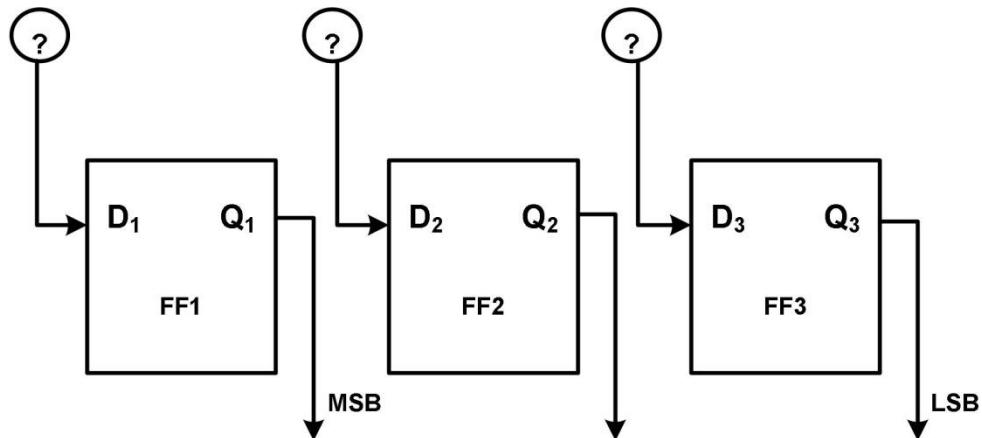
Como comprobación del aprendizaje se pide al alumno proponer una variante al sumador anterior tal que el resultado lo entregue íntegro. Además se puede modificar el diseño para obtener una Máquina de Estado Finito en la que se puedan sumar cadenas de longitudes diferentes. Considerar como posibles entradas 0 + símbolo vacío, 1 + símbolo vacío, símbolo vacío + 0, símbolo vacío + 1.

Sin embargo, no existe una Máquina de Estado Finito que pueda realizar la multiplicación binaria en serie. ¿Por qué?

Entre los casos de aplicación más comunes de este modelo se tiene el diseño de sistemas digitales secuenciales, como ya se ha indicado anteriormente. Se pueden proponer diversos ejemplos específicos como el que se refiere a continuación. Se recomienda al alumno que consulte bibliografía adicional sobre este importante campo de utilización de estos modelos.

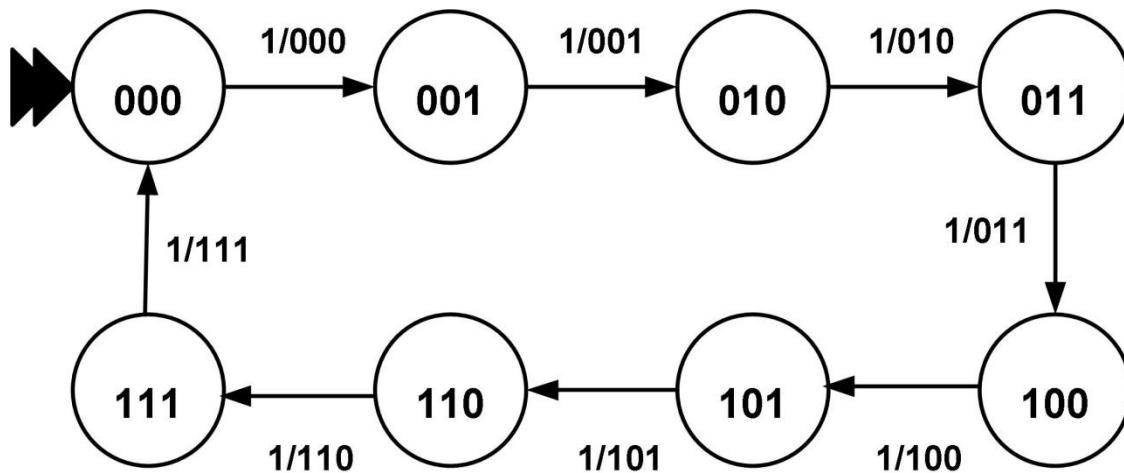
EJEMPLO: CONTADOR SÍNCRONO DE 3 BITS.

Diseñar un contador síncrono de 3 bits con flip flops tipo D. En este caso es adecuado utilizar una Máquina de Estado Finito (porque la salida no es un simple bit, sino tres). De hecho, se deben evaluar los valores que debe haber en las entradas de cada uno de los flip flop tipo D que se van a emplear para obtener el conteo requerido.



Si se evalúa la expresión lógica de las entradas a los tres flip-flops, se podrán obtener, combinando las tres salidas, los valores binarios deseados. Para mayor claridad en los conceptos referidos en este problema, apoyarse de un texto sobre Sistemas Digitales Secuenciales.

El Diagrama y la Tabla de Transición obtenidos para las especificaciones de diseño de dicho contador de tres bits serían los siguientes:



En la siguiente página se muestra la *Tabla de Transición de una Máquina de Estado Finito*, conocida en electrónica como *tabla de verdad*.

Nótese que la función siguiente estado indica el próximo valor numérico en el conteo y la función salida muestra precisamente el número que se representa. La salida y el siguiente estado no se consideran cuando la entrada (que sería la señal del reloj) es 0, porque éstos no ocasionan acción alguna en el circuito

Los diagramas de Karnaugh para determinar las entradas de los 3 flip flops, deducidas de la tabla, se muestran después de la tabla de verdad también a partir de la página siguiente.

S			I			f			g					
Q1 Q2 Q3			Q1 Q2 Q3			Q1 Q2 Q3			Q1 Q2 Q3			Q1 Q2 Q3		
0 0 0				0 0 0		0 0 1			0 0 0			0 0 0		
0 0 1				0 0 1		0 1 0			0 0 1			0 0 1		
0 1 0				0 1 0		0 1 1			0 1 0			0 1 0		
0 1 1				0 1 1		1 0 0			0 1 1			0 1 1		
1 0 0				1 0 0		1 0 1			1 0 0			1 0 0		
1 0 1				1 0 1		1 1 0			1 0 1			1 0 1		
1 1 0				1 1 0		1 1 1			1 1 0			1 1 0		
1 1 1				1 1 1		0 0 0			1 1 1			1 1 1		

Q ₁ Q ₂		0 0	0 1	1 1	1 0
Q ₃					
0				1	1
1			1		1

FF1

$$D_1 = \overline{Q}_1 Q_2 Q_3 + Q_1 \overline{Q}_3 + Q_1 \overline{Q}_2$$

Q ₁ Q ₂		0 0	0 1	1 1	1 0
Q ₃					
0			1	1	
1		1			1

FF2

$$D_2 = Q_2 \overline{Q}_3 + \overline{Q}_2 Q_3 = Q_2 \oplus Q_3$$

$Q_1 Q_2$	0 0	0 1	1 1	1 0
Q_3	0	1	1	1
	1			

FF3

$$D_3 = \overline{Q_3}$$

El circuito contador de tres bits se obtiene a partir de las ecuaciones para las entradas en los tres flip flops: D_1 , D_2 y D_3 , recientemente obtenidas. Cada uno de los tres flip-flops debe recibir la entrada según la expresión lógica correspondiente. Queda como trabajo para el estudiante diseñar el diagrama electrónico resultante, a partir de las tres ecuaciones.

Nótese que las Tablas de Transición son ampliamente utilizadas en estas aplicaciones en cursos de electrónica digital secuencial, aunque sin haberlas justificado desde un punto de vista formal. Para obtener mayor información de estas aplicaciones consultense libros sobre la materia.

EJEMPLO: MASCOTA VIRTUAL.

Una especie animal que podría ser mascota, puede ser simulada por el modelo descrito en este capítulo. En este caso tenemos que el ser vivo cuenta con un estado de ánimo presente en todo momento de su vida y que estos son determinados por el estado anterior en combinación con la entrada, además del resultado causado que es la salida. La entrada es la causa y la salida es el efecto.

Se podrían expresar los conjuntos componentes de la siguiente forma:

- Entradas tales como un regaño, una felicitación, un abrazo recibido, un beso, descansar, etc.
- Salidas como gritar, sonreir, llorar, bostezar o acalorarse entre otros.
- Estados que serían los estados de ánimo tales como enojado, feliz, deprimido, eufórico, ilusionado, desanimado o cualquiera que tenga según la especie animal que se busque representar.

La implementación de mascotas virtuales es muy aplicada en la actualidad para aquellas personas que quieren tener compañía pero no pueden o no quieren mantener un animal vivo.

También una persona podría ser representada por medio de un modelo similar como el lector podrá imaginarse.

Un caso particular lo sería el mostrado en la siguiente tabla de transición que por razones del amplio espacio requerido se muestra solo un fragmento.

ESTADO VS ENTRADA	0) ·DESPERTAR·	1) ·ACARICIAR·	2) ·ALIMENTAR·	3) ·AGUA·	4) ·TRANQUIL
0) NACIENDO	2/0	0/7	0/7	0/7	0/7
1) ENOJADO	1/7	3/1	1/4	1/4	3/1
2) CONTENTO	2/7	2/9	2/2	2/2	2/8
3) TRANQUILO	3/7	2/9	3/2	3/2	5/12
4) ASUSTADO	4/7	3/1	4/4	4/4	3/1
5) EN CELO	4/7	4/12	5/4	5/4	5/12
6) HAMBRIENTO	6/7	6/12	17/1	6/2	6/1
7) SEDIENTO	7/7	7/3	7/2	2/1	7/9
8) CANSADO	8/7	8/9	8/2	8/2	8/10
9) ACALORADO	9/0	9/10	9/4	2/1	9/10
10) DORMIDO	2/7	10/9	10/5	10/5	10/5
11) FRIO FENITO	11/7	11/9	11/2	11/2	11/1

Obviamente que los valores mostrados en las celdas de la tabla hacen referencia primero a los estados siguientes sin mostrar la q, y posterior a la diagonal se escribe la salida, renombrada con un número entero de acuerdo a una relación de equivalencias. Por ejemplo la salida marcada como 0 sería silencio, el 1 sería gruñir, 2 sería aullar y así sucesivamente.

EJEMPLO: RECORRIDO EN UN LABERINTO VIRTUAL.

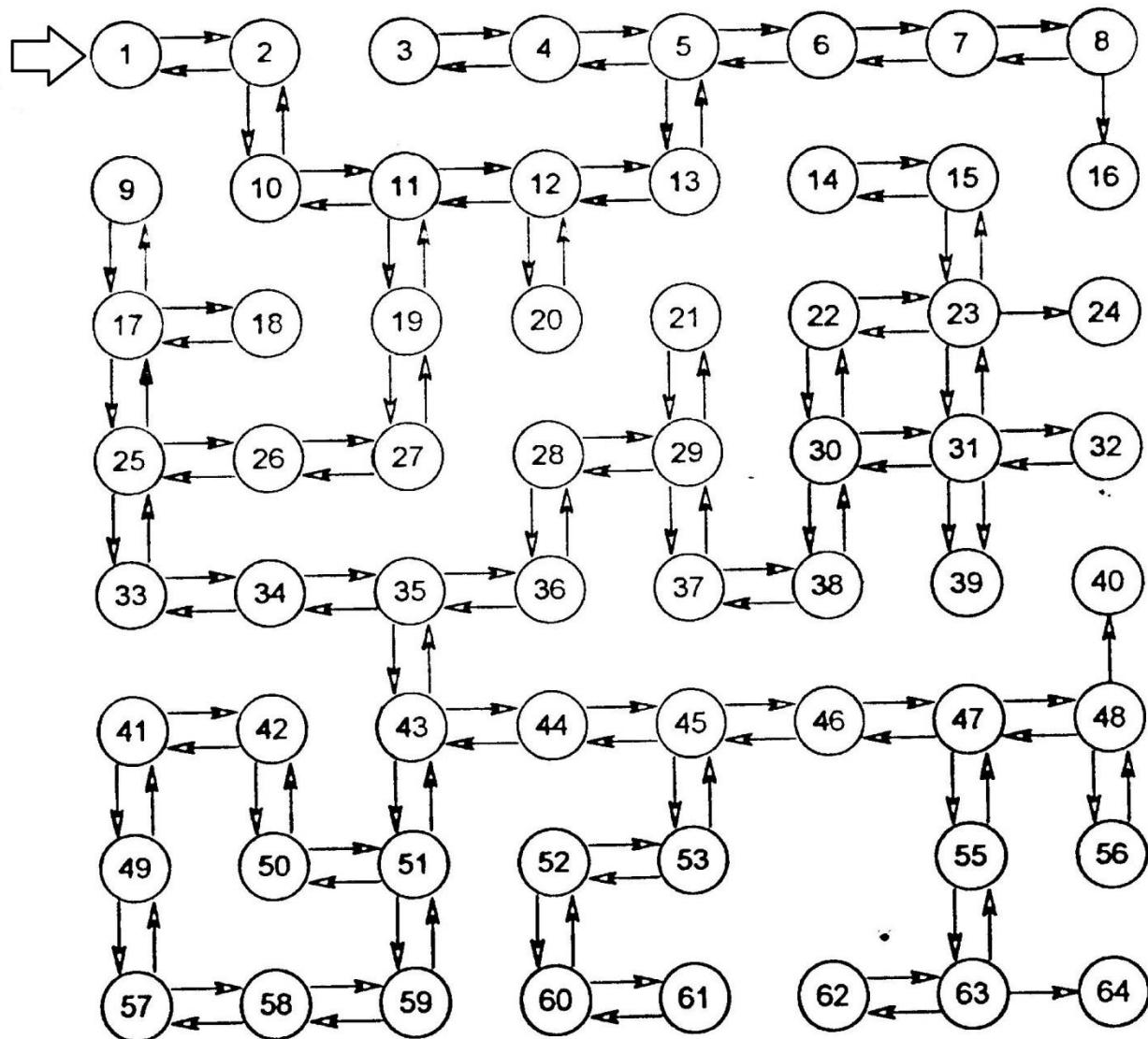
Una aplicación que puede tener una Máquina de Estado Finito es la de representar un laberinto que puede ser recorrido por un personaje ficticio en un sistema que pudiera ser un videojuego o como parte de un experimento de simulación. Recordamos que en 1950 Claude Shanon desarrolló un experimento con un laberinto en el cual se colocaba un ratón electromecánico llamado Theseus que intentaba encontrar la salida. Las formas en las que el ratón podía realizar la exploración dieron lugar a los primeros métodos de búsqueda que se utilizaron en Inteligencia Artificial.

Un diagrama que podría corresponder al referido laberinto podría ser el que se muestra en la siguiente página.

En el ejemplo se observan los posibles 64 estados que podría tener y que corresponden a posiciones dentro del laberinto. Las cuatro entradas posibles corresponden a las direcciones ortogonales (arriba, abajo, derecha e izquierda) y las salidas podrían ser “ver pasillo”, “ver muro” o “ver salida” que corresponden estos últimos a los casos en los que ya no hay regreso hacia atrás.

Por cuestión de claridad no se muestra en cada transición la entrada porque se deduce del diagrama y la salida se evidencia por lo que un observador tendría frente a sí.

Al implementar esta Máquina de Estado Finito se puede hacer la simulación que podría emplearse para aplicar en un juego, incluido el famoso ejemplo de pacman, o bien el ratón virtual ya referido.



EJEMPLO: ENCRYPTADOR O CIFRADOR DE CADENAS.

Un problema a resolver muy importante dentro de las aplicaciones prácticas de las ciencias computacionales consiste en contar con algoritmos que permitan hacer una encriptación o cifrado de cadenas alfanuméricas, con lo que se hacen ilegibles para personas que no conocen la técnica empleada.

En el caso de las aplicaciones de comercio electrónico se sabe que mucha gente tiene desconfianza de comprar algunos productos por este medio al imaginarse que la información confidencial que envía puede ser interceptada y empleada con fines de cometer un fraude con estos datos. Si los mismos fueran adquiridos en una manera que no fueran legibles para el interceptor y no pudieran modificarlos para obtener los reales, se tendría confianza plena en el uso de este tipo de portales.

Es muy conocida la historia de Enigma, la cual era una máquina que disponía de un mecanismo de cifrado rotatorio, que permitía usarla tanto para cifrar como para descifrar mensajes y que fue empleada por los alemanes para sus tácticas militares durante la Segunda Guerra Mundial. El empleo de este equipo les permitió tener mucha ventaja durante una buena parte de esta guerra hasta que su sistema de cifrado fue finalmente descubierto y el conocimiento por parte de los aliados de la información que contenían los mensajes supuestamente protegidos es considerado por algunos como la causa de haber podido concluir este conflicto armado al menos dos años antes de lo que hubiera sucedido si no se hubiera descubierto su operación.

Existen muchos algoritmos para encriptar un mensaje siendo el más común el de cambiar el código ASCII de cada uno de los caracteres, sumando o restando un número entero. De esta manera se tiene que por ejemplo si el valor numérico que se considera es +3, la a se encripta como d, la b como e, la c como f, la d como g, la e como h, la f como i y así sucesivamente. Ciertamente se cambian los caracteres pero la técnica empleada es sumamente arcaica y fácil de descubrir hasta cierto punto.

Si se deseara cifrar una cadena como por ejemplo *Guadalajara*, con la técnica mencionada en el párrafo anterior la a se transformaría siempre igual y la cadena encriptada tendría muchas veces la misma letra, dando la pista a los hackers de que en esa posición original habría una misma vocal, ya que son éstas las letras que más aparecen en una palabra cualquiera. Si la misma letra a se cifrara de manera distinta cada vez y de una manera controlada entonces sería más difícil conocer la cadena original.

El encriptador se puede diseñar en base a un modelo de una Máquina de Estado Finito que inicia en el estado q_0 y que cada vez que va recibiendo un carácter de la cadena a cifrar va produciendo un carácter de salida y cambia a un estado nuevo para que así vaya transitando por varios de ellos. Si se intentara encriptar la cadena *Guadalajara* cada vez que van llegando las a se estaría en un estado diferente y por lo tanto se cifraría de manera distinta cada una de ellas.

El mismo modelo formal se podría emplear para posteriormente hacer la desencriptación al aplicarse de manera inversa, es decir, ahora dado el carácter de salida, se debe determinar para el estado actual cuál sería el símbolo de entrada que la generó. Por lo mismo, cuando se diseña el encriptador se debe tener cuidado de que dos diferentes entradas para un estado actual, no generen el mismo símbolo de salida, ya que al desencriptar existiría ambigüedad.

Queda para el alumno el diseño de una tabla de transición para este caso.

CONCEPTOS ADICIONALES

Existen otros diferentes tipos de Máquinas de Estado Finito para modelar máquinas de cómputo, que son llamadas Máquinas secuenciales. Hay un tipo tradicional en el que las salidas corresponden a transiciones entre estados y estas representaciones son conocidas como **Máquinas de Mealy**, ya que fueron primeramente estudiadas por George H. Mealy en 1955.

Adicionalmente existe otro tipo importante de Máquina de Estado Finito con salida, donde esta última es determinada sólo por el estado y se la conoce como **Máquina de Moore**, ya que Edward Forrest Moore introdujo este tipo de Máquina en 1956.

Se observa una estrecha relación de las Máquinas de Mealy con las de Estado Finito y de la de Moore con los Autómatas de Estado Finito que se estudiarán el siguiente capítulo. Una Máquina de Moore puede convertirse en una de Mealy equivalente añadiendo más estados y haciendo las modificaciones necesarias.

Queda para el estudiante investigar más sobre estas dos variantes de Máquinas de Estado Finito, muy usuales en el área de aplicación de los *Circuitos Digitales Secuenciales*.

Un proyecto final de este curso consistirá en diseñar e implementar un programa que represente una aplicación del modelo de una Máquina de Estado Finito en un caso práctico que generalmente tiene relación con una simulación de un modelo del mundo real, tal como serían los ejemplos mencionados al inicio de este módulo.

Hay una relación muy estrecha entre Máquinas y Autómatas de Estado Finito. De hecho, éstos últimos son un subconjunto de las primeras. La forma como se relacionan se estudiará en el siguiente capítulo, donde se tratarán con detalle sus características.

Queda pendiente el empleo de las Máquinas de Estado Finito para representar sistemas electrónicos, computacionales o, en general, del mundo real. En el siguiente capítulo se analizarán en forma detallada ciertos casos de aplicación, como la máquina despachadora de refrescos, diseñada como Máquina y como Autómata de Estado Finito.

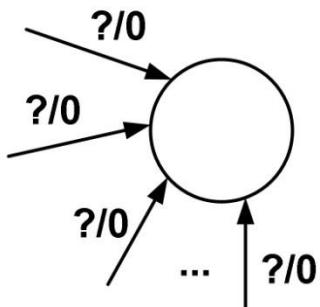
CAPÍTULO 4

AUTÓMATAS DE ESTADO FINITO

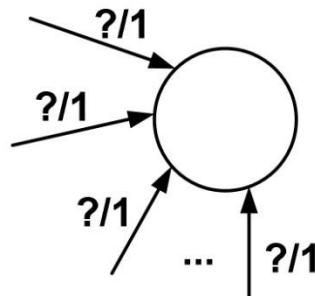
CONCEPTOS GENERALES

Un Autómata Finito, también llamado Autómata de Estado Finito, es toda Máquina de Estado Finito en la que el conjunto de símbolos de salida es exclusivamente $O = \{ 0, 1 \}$ y dónde el estado actual determina cuál de ellos fue el último dato de salida. Aquellos estados para los cuales el último dato de salida fue 1, se denominan estados de aceptación. En todo Autómata Finito, representado como **A**, debe haber cuando menos un estado de aceptación y por sentido común se recomienda que no todos lo sean.

En forma gráfica se muestra la forma como se identifican los dos tipos de estado que se pueden presentar en este Autómata. El carácter ? significa que no importa cuál es el símbolo en la entrada.



Estado de no aceptación



Estado de aceptación

El motivo por el que el conjunto O debe ser $\{ 0, 1 \}$ resulta razonable debido a que se los utiliza en la computación y en la electrónica digital, donde se opera con el sistema binario y que los símbolos resultan ser adecuados como indicación de aceptación o rechazo.

La segunda característica surge de la necesidad práctica de saber cuál fue el último bit de la cadena de salida, sabiendo en qué estado se *termina* el recorrido en el diagrama de transición para una cadena de entrada, y sin importar los estados intermedios que se van obteniendo. Si se conoce el estado final, entonces también se deberá conocer cuál fue el último bit de la cadena de salida.

En síntesis, es fácil notar que en muchos casos, lo importante no son las salidas momentáneas del modelo, sino únicamente el último bit de salida. Lo

anterior es muy importante, ya que se presenta un *0* en la salida cuando la cadena de entrada no cumple un enunciado, puesto como **especificación** de diseño del **Autómata**; por otro lado se presenta el **1 en la salida** cuando **sí lo cumple**, tal y como se observa en el reconocedor de cantidades pares de unos que se presentó en el capítulo anterior.

Se podría decir que un Autómata Finito es un aceptador o rechazador de cadenas.

Desde el inicio del capítulo anterior se afirmó que uno de estos modelos tenía una memoria interna primitiva. Esa afirmación se justifica en el hecho de que al operar, ya sea una Máquina de Estado Finito o un Autómata Finito se "sabe" con toda certeza cuál es su estado actual, pero "no recuerda" cómo llegó al mismo. Como se verá en un ejemplo a resolver, esa situación impide que pueda reconocer cadenas con la misma cantidad de diferentes símbolos, tales como los paréntesis cuando se anidan correctamente o el Lenguaje $L = \{ 0^n1^n \mid n \in \mathbb{N} \}$. Esa limitación es muy importante en aplicaciones computacionales.

EJEMPLO:

Trazar el diagrama de transición de la Máquina de Estado Finito definida en la tabla; en caso de que sea también un Autómata Finito, trazar su diagrama correspondiente en la notación respectiva; determinar el conjunto de estados de aceptación.



S	I	w	x	y	z
q_0		$q_1, 0$	$q_2, 1$	$q_0, 1$	$q_1, 0$
q_1		$q_1, 0$	$q_0, 1$	$q_1, 0$	$q_0, 1$
q_2		$q_2, 1$	$q_1, 0$	$q_1, 0$	$q_0, 1$

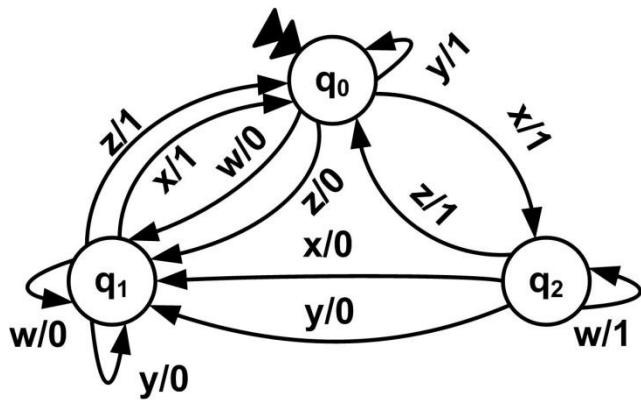


Diagrama de transición de M

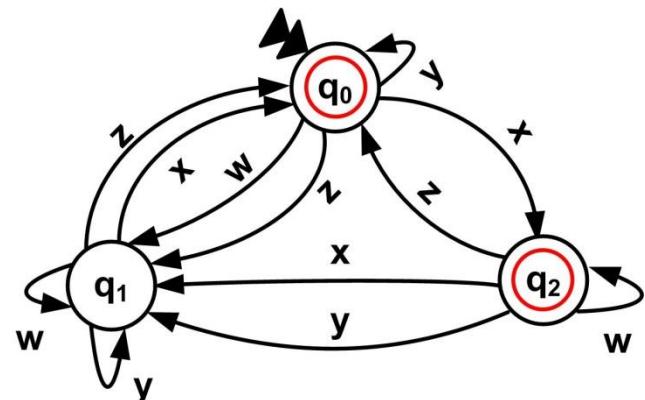


Diagrama de transición de A

Los Estados de aceptación son q_0 y q_2 mientras que el Estado de no aceptación es q_1 . Para distinguirlos, se remarca con un círculo doble cada estado de aceptación al representarlo ya como Autómata Finito. Obsérvese que no es necesario describir los bits de salida en cada arco, ya que tales símbolos son evidentemente deducibles.

El anterior es un Autómata de Estado Finito en razón de que en todos los arcos que llegan a los estados de aceptación se tiene la indicación de bit de salida en 1, y por otro lado se tiene un 0 en los de no aceptación.

En el Autómata podremos analizar qué es lo que sucede cuando llega una cadena de entrada, tal como $yxxzwyxw$, y en cuyo caso podremos especificar la secuencia de estados por los que pasa el Autómata, de la siguiente forma:

Cadena = $y \ x \ x \ z \ w \ y \ x \ w$ // cadena de entrada.
 $q_0 \ q_0 \ q_2 \ q_1 \ q_0 \ q_1 \ q_1 \ q_0 \ q_1$ // estados recorridos.

Si un Autómata de Estado Finito recibe como dato de entrada una cadena o arreglo, se puede concluir el recorrido en el modelo formal o en un estado de aceptación o en uno de no aceptación; la clase de estado en el cual se termina, establece cuando una cadena es aceptada por el Autómata. Como el recorrido finalizó en este caso en un estado de no aceptación (q_1), la cadena $y \ x \ x \ z \ w \ y \ x \ w$ se considera que no es aceptada por el Autómata; esto significa que no cumple con las especificaciones que se dieron en el diseño.

En este caso, al analizar las transiciones no se consideran los símbolos de salida. Una *transición de un Autómata Finito* se considera que consta de dos etapas únicamente: ingresa un símbolo de entrada, y hay un cambio de estado. La tabla de transición, ya simplificada para el reconocedor, quedaría:

S	I	w	x	y	z
q0	q1	q2	q0	q1	
q1	q1	q0	q1	q0	
q2	q2	q1	q1	q0	

Representación en la que se observa que con una flecha se apunta el estado inicial, y que se encierra en un círculo cada estado de aceptación. No se incluye la indicación del símbolo de salida, ya que tal valor se deduce según el tipo de estado al que se accede.

MUY IMPORTANTE: El Autómata de Estado Finito se emplea en vez de la Máquina, cuando no es importante considerar las diversas salidas obtenidas al ir recibiendo los símbolos de la cadena de entrada; solamente importa conocer el último bit de salida, para verificar el cumplimiento o no de una característica específica. También es fundamental la existencia de estados que influyen en la operación del modelo.

El alumno no debe inventar un modelo en el que aparezcan tanto estados de aceptación como salidas en las transiciones, ya que se distinguen bien los dos casos de modelos formales y no se pueden mezclar elementos de ambos.

DEFINICIÓN FORMAL DE AUTÓMATA DE ESTADO FINITO

UN AUTÓMATA DE ESTADO FINITO CONSISTE EN:

- a) Un conjunto finito I de símbolos de entrada.
- b) Un conjunto finito S de estados.
- c) Una función estado siguiente f de $S \times I$ en S .
- d) Un subconjunto $A \subseteq S$ de estados de aceptación.
- e) Un estado inicial $\sigma_0 \in S$.

Se expresa en notación de conjuntos con la 5-tupla: $\mathbf{A} = (I, S, f, A, \sigma_0)$. A ésta se la llamará la *descripción formal del Autómata Finito*.

Para el Autómata expresado en el ejemplo anterior, se tiene:

$$\begin{aligned} I &= \{w, x, y, z\} & S &= \{q_0, q_1, q_2\} & A &= \{q_0, q_2\} & \sigma_0 &= \{q_0\} \\ f &= \{f(q_0, w) = q_1, f(q_0, x) = q_2, f(q_0, y) = q_0, f(q_0, z) = q_1, f(q_1, w) = q_1, \end{aligned}$$

$$f(q_1, x) = q_0, f(q_1, y) = q_1, f(q_1, z) = q_0, f(q_2, w) = q_2, f(q_2, x) = q_1, \\ f(q_2, y) = q_1, f(q_2, z) = q_0 \}.$$

Nótese que en las tres representaciones de un Autómata Finito, el especificar el conjunto de los estados de aceptación A, hace innecesario definir la función g de una Máquina, por obviedad; las transiciones que llevan a un estado de aceptación implicarían un 1 en la salida, por ejemplo. Por lo mismo, el conjunto de símbolos de salida (0 y 1) se omite también.

EJEMPLO:

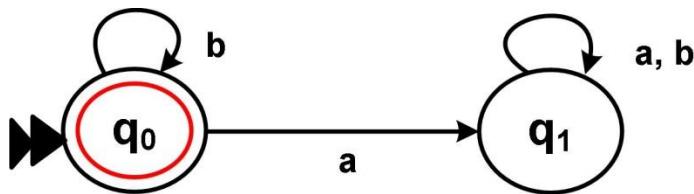
Diseñar un Autómata Finito que acepte únicamente los arreglos con $I = \{a, b\}$ pero que no contengan a , lo que también se podría expresar como las cadenas de $L = \{b^n \mid n \in \mathbb{N}_0\}$.

Es un caso algo extraño, ya que nos podremos preguntar, por qué si la entrada a no se debe presentar en ninguna cadena, entonces ¿para qué la definimos como posible entrada?

Este caso debe corresponder a una situación en la que el Autómata serviría para representar un dispositivo de control, instalado en una empresa para monitorear fallos en distintas partes de las instalaciones. A dicho sistema llegan señales provenientes de sensores ubicados en sitios estratégicos; el llegar una a , podría significar **avería**, mientras que la b significaría **bien**. Si no llega ninguna de las dos, entonces no habría problema.

Esa podría ser la razón por la que es posible recibir una a , aunque tal situación sería no aceptada.

El diseño del Autómata requerido quedaría como sigue:



Cuando se inicializa el Autómata se acepta la palabra, ya que la cadena vacía no contiene a ; evidentemente, el estado inicial es de aceptación. Inclusive, mientras se sigan recibiendo las b en la entrada, se sigue aceptando la cadena.

La situación cambia cuando estando en el estado q_0 se recibe una a . En ese caso, ya no puede ser aceptada ninguna cadena, sin importar cuál es el resto de la misma. Por eso, se pasa al estado q_1 , en el cual, sin importar lo que llegue, ya no puede ser aceptada la entrada. **Dicho estado puede ser omitido en el diagrama de transición, si se desea.**

ASPECTOS IMPORTANTES EN DISEÑO DE UN AUTÓMATA FINITO

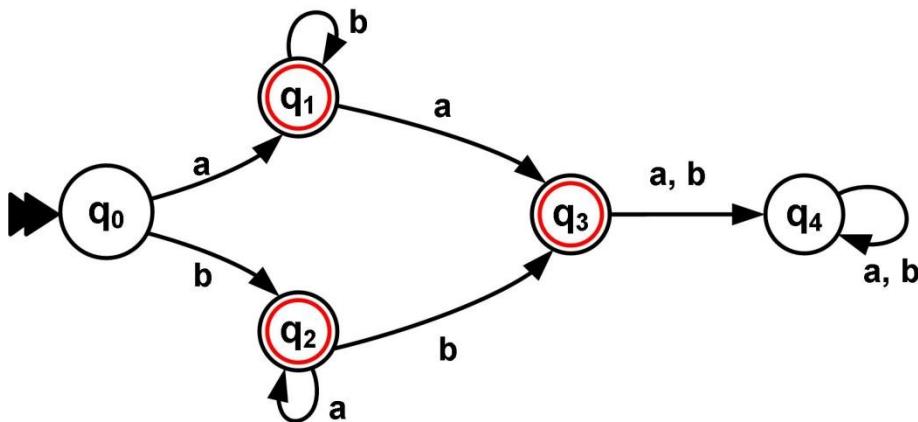
▫ Verificar si al querer representar el sistema real con este modelo, se puede (¿se identifican estados?) y se debe (¿no sería mejor con una Máquina de Estados?).

▫ Al inicio del diseño de este modelo, debemos preguntarnos: ¿la cadena vacía debe ser aceptada? Si la respuesta es afirmativa, entonces el estado inicial debe ser de aceptación. No lo es, en caso contrario.

▫ Cuando una entrada es trascendental, se tiene un cambio a un estado distinto al actual; cuando una entrada, es intrascendente, se representa como un “cambio” al mismo estado (gráficamente, como un lazo).

EJEMPLO:

Construir una expresión regular que describa el Lenguaje aceptado por el siguiente Autómata:

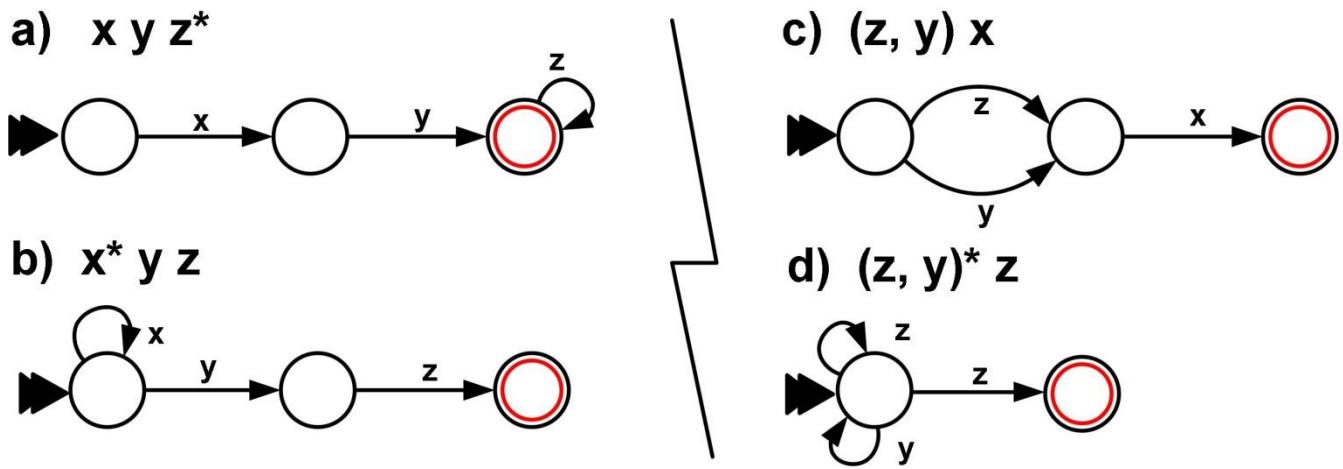


Para resolver este problema, se definen las posibles trayectorias que llevan desde el estado inicial, a todos y cada uno de los estados de aceptación, que en este caso son tres. Para llegar a dos de ellos, hay uno solo de esos caminos, pero para llegar al tercero, existen dos posibilidades, pasando por los dos estados mencionados anteriormente, obteniéndose, en total, cuatro de esas posibles trayectorias. Finalmente, se unen todas las respuestas individuales.

De esa forma, se tiene que la respuesta es $L = \{ ab^* \} \cup \{ ba^* \} \cup \{ ab^*a \} \cup \{ ba^*b \}$.

EJEMPLO:

¿Cuáles de los Lenguajes descritos por las siguientes expresiones regulares para el Alfabeto $\Sigma = \{ x, y, z \}$ son infinitos?



En este caso, los Autómatas Finitos que aceptan Lenguajes infinitos, son los indicados en los incisos a, b y d. Eso, por el simple hecho de tener lazos en su estructura.

Además de ser esa una característica para identificar los Autómatas que aceptan este tipo de Lenguajes, también ocurre en los casos en los que se observa en el diagrama la presencia de circuitos, es decir, de caminos cerrados. Por tanto, si se invirtiera el sentido de uno solo de los dos arcos que se dirigen del estado inicial al siguiente estado en el modelo del inciso c, ya no aceptaría un Lenguaje Finito como ahora.

AUTÓMATAS EQUIVALENTES

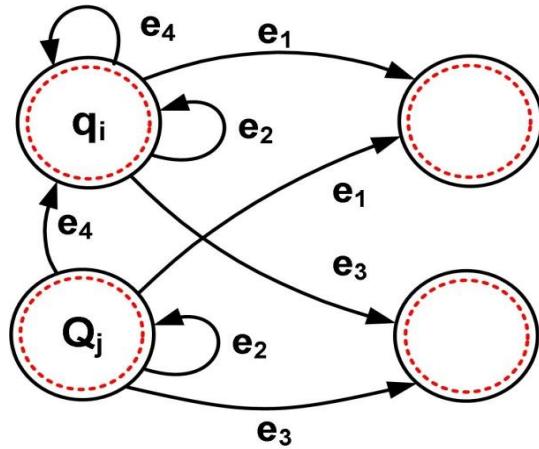
Si dos Autómatas aceptan exactamente las mismas palabras, se dice que tales son equivalentes. El conjunto I debe ser el mismo para ambos, mientras que σ_0 y S pueden variar y la función f necesariamente cambia entre ellos.

Es evidente que siempre es mejor hacer el diseño más simple, ya que su implementación deberá ser más fácil y con menor riesgo de cometer errores al diseñarlo y sobre todo al implementarlo. Queda para el estudiante proponer algunos ejemplos de Autómatas equivalentes al resolver los problemas propuestos al final de este texto.

CONSIDERACIÓN IMPORTANTE PARA SIMPLIFICAR UN AUTÓMATA FINITO

Si en un Autómata Finito se tienen dos estados, siendo ambos o de aceptación o de no aceptación, y las transiciones que salen de ambos se dirigen a los mismos estados siguientes para cada una de las entradas (o si son equivalentes), entonces se pueden reducir, quedando un único estado en lugar de los dos estados originales.

En el caso del Autómata Finito que se muestra en la figura, se pueden simplificar q_i y q_j como un único estado, pero solamente si ambos son aceptación o de no aceptación.



IMPLEMENTACIÓN DE UN AUTÓMATA FINITO

Los Autómatas de Estado Finito, al igual que las Máquinas, tienen la gran ventaja de que SE PUEDEN PROGRAMAR, con lo que se transforman en una herramienta muy valiosa para todo el profesional que intenta simular el funcionamiento de un sistema real, inicialmente representado con este modelo matemático.

EJEMPLO:

Diseñar e *implementar* por medio de un Autómata Finito, un analizador que acepte las cadenas que representan los nombres de identificadores válidos de un Lenguaje de programación típico como C o Java. Este diseño es un pequeño subconjunto de un analizador léxico.

Se tomará como base el criterio de que un identificador inicia con una letra o un guión bajo (grupo que se denominará simplemente como letra por comodidad) y después puede llevar dígitos y/o letras y/o guiones bajos, pero nunca caracteres diferentes a estos tipos.

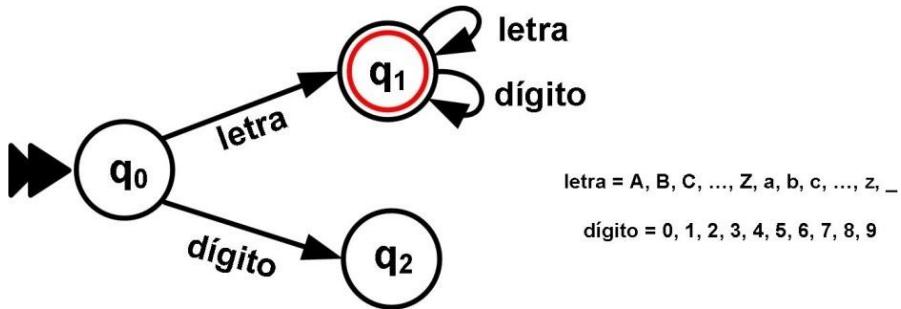
Se definen las clases para los símbolos de entrada que se consideran en los arcos como tipos de datos para simplificar la representación del diagrama:

dígito: 0, 1, 2, ..., 9.

letra: A, B, C, ..., Z, a, b, c, ..., z, _ . // letras, además de _.

otro: +, -, @, ;, sp, ~, \$, %, ... // los cuales no son aceptados.

Las transiciones se realizan una por cada carácter de la cadena a evaluar.



Al dibujar el diagrama de transición se debe tomar en cuenta que por conveniencia, los arcos que corresponderían a situaciones de no aceptación se omiten, quedando como entendido que irían todos a un estado de rechazo (como q_2) del que no se podría salir. En realidad el estado q_2 se puede omitir debido a que llegando a él, de todos modos la cadena no va a ser aceptada, pero se incluye, para que con tres estados el ejemplo sea más demostrativo.

La primera versión de la implementación puede ser en base a la siguiente propuesta de un pseudocódigo:

```

Estado ← 0
LEER siguiente símbolo de entrada
MIENTRAS no ( fin de cadena ) HACER
    OPCIÓN ( Estado ) DE
        0: SI ( Símbolo actual = letra )
            ENTONCES ( Estado ← 1 )
            SINO SI ( Símbolo actual = dígito )
                ENTONCES ( Estado ← 2 )
                SINO ( Error_rutina )
        1: SI ( Símbolo actual = letra )
            ENTONCES ( Estado ← 1 )
            SINO SI ( Símbolo actual = dígito )
                ENTONCES ( Estado ← 1 )
                SINO ( Error_rutina )
        2: Error_rutina
    FINOPCIÓN
    LEER Siguiente símbolo de entrada
FINMIENTRAS
SI ( Estado ≠ 1 ) ENTONCES (Error_rutina)

```

Sin embargo, la opción de diseño con anidamiento de instrucciones *if*, no debe utilizarse para implementar un Autómata, pues en un caso en el que existan varios estados con muchas entradas posibles, resultaría en una instrucción alternativa múltiple (*switch*) con un rango gigantesco como se podrá notar y sería extremadamente difícil su mantenimiento.

Una segunda versión, mucho más práctica, utiliza una tabla de transiciones en la que se consulta cuál será el valor del siguiente estado. La gran ventaja de esta variante es que para cambios en el diseño original del Autómata, las modificaciones se hacen en la tabla solamente. A esta representación se la denominará **Autómata Matricial**.



I	letra	dígito	FDC
S			
q₀	q₁	q₂	x
q₁	q₁	q₁	✓
q₂	x	x	x

El siguiente paso, consiste en traducir esta tabla a un formato de estructura de datos bidimensional simple, por ejemplo un arreglo, para que pueda ser consultada por el programa principal, que se menciona a continuación.

Estado ← 0

REPETIR

LEER siguiente símbolo de entrada

OPCIÓN (símbolo) DE

letra: Entrada ← “letra”

dígito: Entrada ← “dígito”

FDC : Entrada ← “FDC”

Otro : Error_rutina

FINOPCIÓN

Estado ← Tabla (Estado, Entrada)

SI (Estado = “error”) ENTONCES Error_rutina

HASTA (Estado = “aceptar”).

La transición del Autómata Finito se determina en la sentencia marcada en negritas en el algoritmo anterior, y que permite actualizar la variable que define el estado actual. Los valores de la tabla de transición se pueden incluir como parte del algoritmo en forma de un arreglo bidimensional, como se ha referido anteriormente. Dicha estructura tendría tantos renglones como estados resultaran en el modelo, y tantas columnas como tipos de datos diferentes. Se recomienda que "aceptar" (\checkmark) y "error" (X) se vinculen con un entero cada uno, para hacerlos homogéneos con el resto del contenido de la tabla, mismo que puede ser representado por el entero que indique el subíndice de la q correspondiente.

El arreglo tendría los siguientes elementos: $f(0, 0) = 1$, $f(0, 1) = 2$, $f(0, 2) = X$, $f(1, 0) = 1$, $f(1, 1) = 1$, $f(1, 2) = \checkmark$, $f(2, 0) = X$, $f(2, 1) = X$ y $f(2, 2) = X$. Para fines prácticos se elimina la fila correspondiente al estado q_2 debido a que no hay posible aceptación de la cadena.

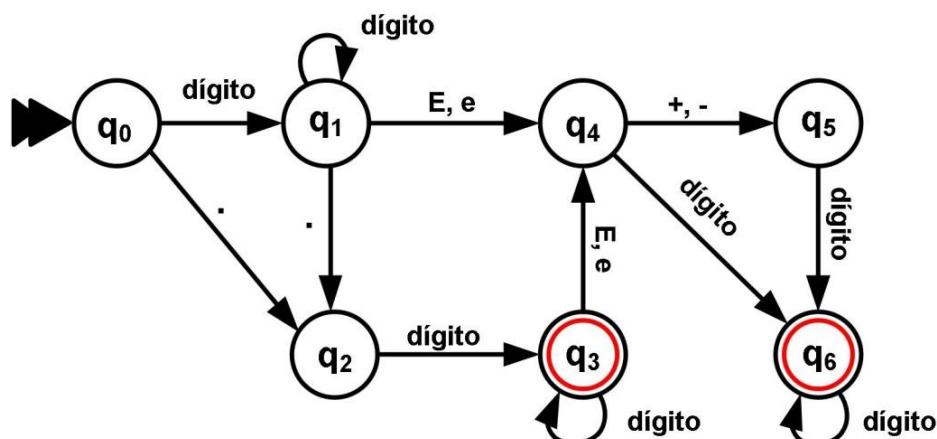
Este es un pseudocódigo y por ello se le asignan valores tipo cadena a la variable llamada *Entrada*, pero es evidente que sería mejor asignarle valores de tipo entero, según sea el *tipo* de símbolo de entrada.

En realidad se está haciendo la misma implementación de software que corresponde a un grafo dirigido, tema que se estudió en el curso de Matemáticas Discretas.

EJEMPLO:

Diseñar e implementar por medio de un Autómata matricial, un analizador que reconozca cadenas que representen tanto a los números reales sin signo, como a los exponentiales también sin signo.

Se hace el diseño del diagrama de transición y se transforma en el formato de la tabla de transición equivalente tal y como se hizo en el ejemplo anterior.





S	I	dígito	.	E, e	+, -	FDC
q_0	q_1	q_2	x	x	x	x
q_1	q_1	q_2	q_4	x	x	x
q_2	q_3	x	x	x	x	x
q_3	q_3	x	q_4	x	x	✓
q_4	q_6	x	x	q_5	x	x
q_5	q_6	x	x	x	x	x
q_6	q_6	x	x	x	x	✓

A la tabla anterior se le da un formato de arreglo bidimensional o alguna otra estructura de datos de dos dimensiones (por ejemplo con varias listas). Enseguida se implementa el Autómata de la siguiente forma:

Estado $\leftarrow 0$

REPETIR

LEER siguiente símbolo de entrada

OPCIÓN (símbolo) DE

0, ..., 9 : Entrada \leftarrow “dígito”

- : Entrada \leftarrow “punto”

- E, e : Entrada \leftarrow “letra E”

- +, - : Entrada \leftarrow “signo”

- FDC : Entrada \leftarrow “FDC”

- Otro : Error_rutina

FINOPCIÓN

Estado \leftarrow Tabla (Estado, Entrada)

SI (Estado = “error”) ENTONCES (Error_rutina)

HASTA (Estado = “Aceptar”).

En este caso, la asignación en negritas es la parte más importante del algoritmo, y consiste en accesar en la celda correspondiente del arreglo que representa la Tabla de Transición previamente diseñada, para actualizar la variable del *estado actual*.

Sin embargo un compilador no debe aceptar números con exponentes tan exagerados como el de la cadena 2.34E-385261. ¿Cómo se modificarían el diagrama y la tabla de transición para que el exponente tenga 1 ó 2 dígitos solamente?

Una especificación que no es tomada en cuenta en este diseño es el rango de los números que se están considerando, pues hay que tomar en cuenta que un número flotante o exponencial tiene un rango dentro del cual debe estar acotado.

Nótese que, en esencia, cualquier Autómata Finito se diseña bajo el mismo formato básico. Lo único que cambia de un caso a otro es el contenido del rango de la instrucción *Opción*, según sean los símbolos que componen el Alfabeto de las cadenas que se deben aceptar, y distribuidas según sea su tipo. ***Inclusive, cuando los símbolos de entrada no se pueden clasificar en grupos con una misma transición para todos los casos de entrada, no se requiere la instrucción Opción en el algoritmo.***

¿Cómo se modificaría el algoritmo anterior para implementar una Máquina de Estado Finito, considerando que en ella se deben actualizar no uno, sino dos parámetros (*estado* y *salida*) y que en la transición la que ocurre primero es la salida?

Es obvio que se deberán definir dos tablas o arreglos en el algoritmo, una para el siguiente estado (como en el Autómata) y otra aparte para evaluar la salida. En el lugar adecuado, antes de que cambie el estado actual, se deberá insertar la sentencia:

Salida ← Tabla_g (Estado, Entrada).

Quedando la asignación con **Tabla_f (Estado, Entrada)** para evaluar el cambio de estado.

La indicación FDC significa un *Fin De Cadena* y se incluye como si fuese entrada, pero solamente en la Tabla de Transición. Su función es indicarle al programa que la cadena ha sido introducida en su totalidad y que deseamos conocer el diagnóstico correspondiente.

El referido fin de cadena no sería necesario considerar en el diseño de una Máquina de Estado Finito en donde se desea ir mostrando la salida y el nuevo Estado cada vez que se ingresa un símbolo de Entrada.

La clave para implementar diversos Autómatas Finitos consiste en adaptar y en aplicar esta estructura básica del programa en seudocódigo:

```

Estado ← 0
HACER
    LEER siguiente símbolo de entrada
    OPCIÓN (símbolo)
        A, ..., Z, a, ..., z, _ : Entrada ← "letra"
        0, ..., 9 : Entrada ← "dígito"
        +, - : Entrada ← "signo"
        . : Entrada ← "punto"
        ...      ...      // y así con todas las entradas, si es necesario.
        FDC : Entrada ← "FDC"
        Otro : Error_rutina
    FINOPCIÓN
Estado ← Tabla ( Estado, Entrada )
    SI (Estado = "error") ENTONCES (Error_rutina)
    HASTA (Estado = "ACEPTAR")

```

Existen casos en los que hay que separar en partes una misma clase de símbolos de entrada, como sucede por ejemplo con las letras; esto se debe a que en reconocedores de ciertos tokens se emplean con diferente finalidad. Por ejemplo, la O se utiliza exclusivamente en identificadores, pero la A se emplea además en los números hexadecimales, y por otro lado está el caso de la E que se emplea aparte de los casos anteriormente referidos también en los exponentiales.

Como parte de los criterios de evaluación, el alumno deberá desarrollar la implementación de algunos proyectos, que permitirán demostrar que tiene la habilidad de poderlos programar. Los mismos se desarrollarán por equipos para que los estudiantes adquieran la habilidad de trabajar eficientemente en grupos, misma que requiere desarrollar para cuando egrese y se inserte en el campo laboral.

Un proyecto que cada ciclo escolar se ha solicitado a los alumnos consiste en el diseño e implementación de un reconocedor de diversos componentes léxicos, todos ellos por un mismo programa que se diseña en base a especificaciones definidas. Aunque este no es todavía un analizador lexicográfico, al desarrollar este programa un alumno no debe tener problemas para elaborar uno completo en el curso de compiladores.

De esta misma forma se pueden implementar proyectos con Autómatas que funcionen como reconocedores de ciertas secuencias alfanuméricas válidas, tales como fechas, direcciones de internet, correos electrónicos o CURPs entre otras.

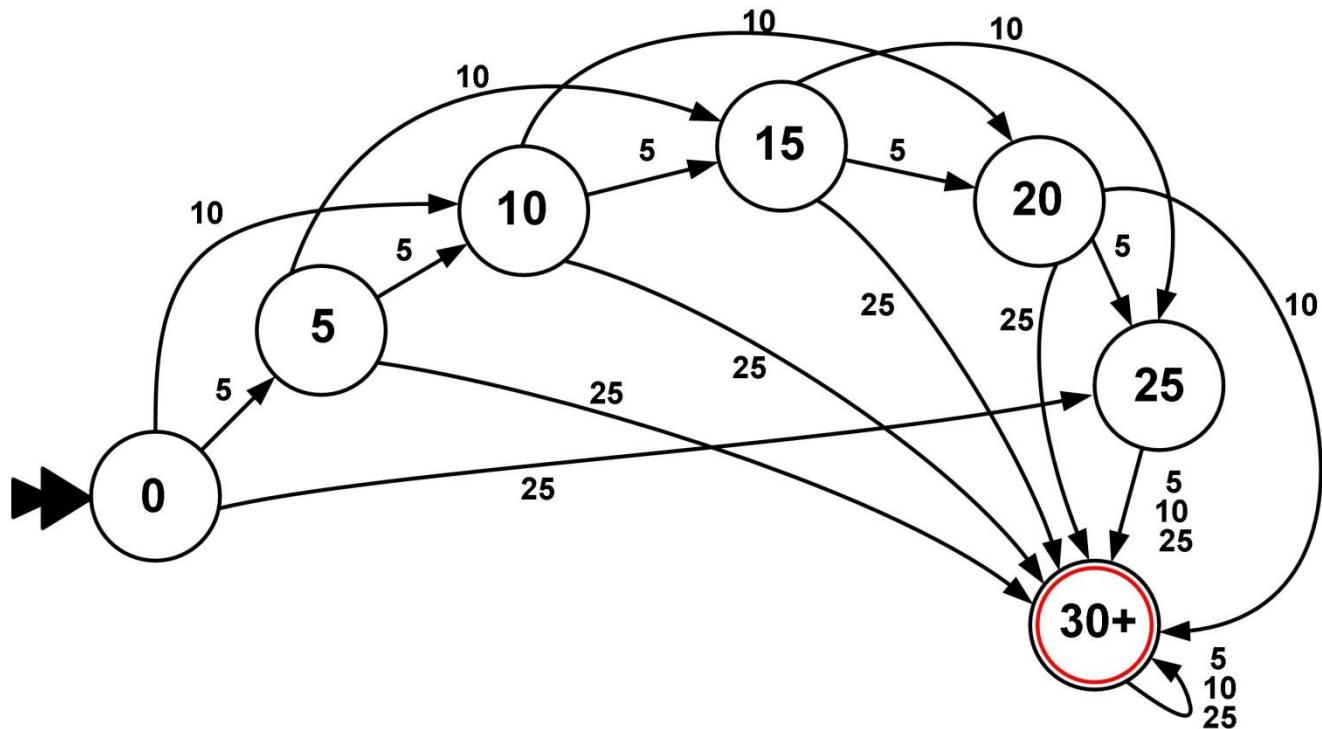
APLICACIONES DE LOS AUTÓMATAS DE ESTADO FINITO SIMULACIÓN DE UN MODELO DEL MUNDO REAL (MÁQUINA PARA VENTA DE REFRESCOS)

Diseñar una máquina despachadora de bebidas que reciba monedas de 5, 10 y 25 centavos y que cuando complete 30¢ entregue al usuario una lata con refresco.

Primera versión (Autómata Finito):

$I = \{ 5, 10, 25 \}$	// la denominación de las monedas.
$S = \{ 0, 5, 10, 15, 20, 25, 30+ \}$	// es la cantidad de dinero acumulada.
$\sigma_0 = \{ 0 \}$	// su operación inicia con 0¢.
$A = \{ 30+ \}$	// significa 30 ¢ o más (no da cambio).

En vez de incluir los valores de las funciones f y g , incluiremos el diagrama de transición.



LIMITACIONES DEL MODELO PROPUESTO ANTERIORMENTE:

- No regresa el cambio, pues la salida es exclusivamente, entrega o no entrega una lata de refresco, dependiendo si se llega o no al estado de aceptación. Recordar que los Autómatas Finitos no muestran salidas.
- La salida consiste en un solo tipo de refresco por esa misma razón.
- Para regresar al estado inicial cuando se complete la venta, se debe considerar una entrada que reinicialice el sistema.

Se podría considerar que el estado de aceptación es el que activa un mecanismo electromecánico que permite seleccionar la marca de refresco y que ya no forma parte del modelo. Los diferentes estados de aceptación son matemáticamente equivalentes.

Segunda versión (Máquina Finita):

$$I = \{ 5, 10, 25, Bn, Bd \}$$

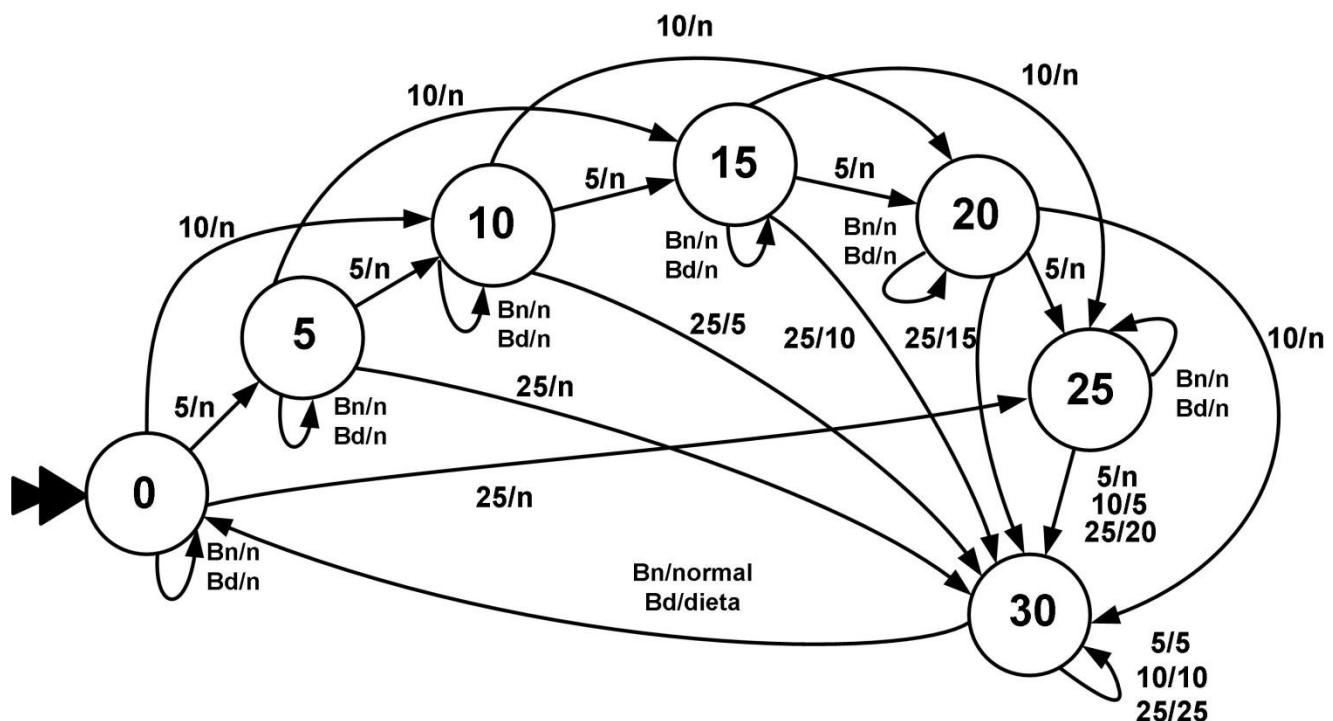
$$O = \{ n, 5, 10, 15, 20, 25, \text{normal}, \text{dieta} \}$$

$$S = \{ 0, 5, 10, 15, 20, 25, 30 \}$$

$$\sigma_0 = \{ 0 \}$$

Bn es el botón para elegir refresco normal (regular), Bd es botón para la bebida dietética, n significa nada (salida vacía); normal y dieta son los dos posibles tipos de refrescos que se ofrecen al usuario. Los demás elementos son similares al ejemplo anterior.

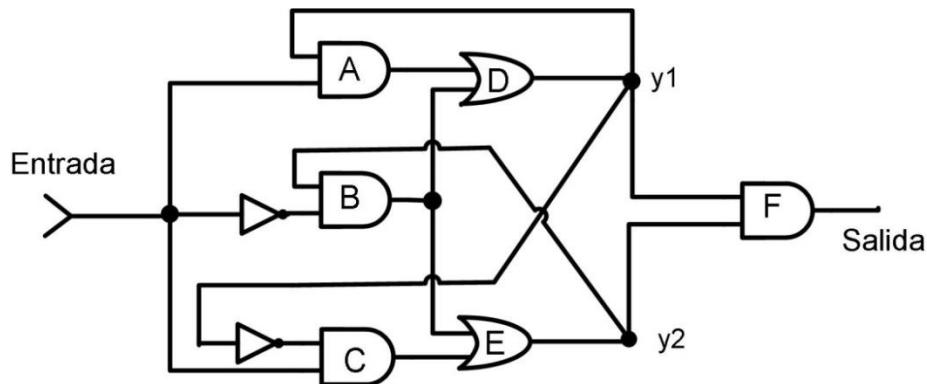
Se propone el siguiente diagrama de transición.



Nótese que en este caso, el modelo de la Máquina es más recomendable, debido a que se deben considerar salidas variadas y no solamente aceptación o rechazo de ciertas condiciones. Todas las limitaciones ya referidas que se presentan en el modelo del Autómata Finito se ven eliminadas en su totalidad en esta segunda propuesta. **Obsérvese que la Máquina no tiene estados de aceptación o de no aceptación**; es común que el alumno se confunda y haga en ocasiones una extraña mezcla de los dos modelos.

APLICACIONES EN CIRCUITOS DIGITALES SECUENCIALES (RECONOCEDOR DE IMPARIDAD DE PULSOS)

Encontrar un Autómata Finito cuyo comportamiento simule al sistema digital del circuito anexo, considerando cuáles son las condiciones que harían que se encienda un diodo emisor de luz conectado en la salida del circuito. Supóngase que existe el tiempo suficiente entre cada cambio de los valores de la entrada para que las señales se propaguen y para que la red de compuertas alcance una configuración estable.



En este caso se tiene un circuito digital *secuencial* en virtud de que para saber cuál sería la salida obtenida, se debe conocer además de la entrada, lo que sucede en los puntos marcados como **y₁** y **y₂** que constituyen la retroalimentación. Los valores de voltajes presentes en esos puntos, en combinación, determinarían el estado del sistema.

La tabla de transición o tabla de verdad que se obtiene del análisis del circuito sería la siguiente:

S	I	0	1
y₁	y₂	y₁	y₂
0	0	0	0
0	1	1	1
1	0	0	0
1	1	1	1

→

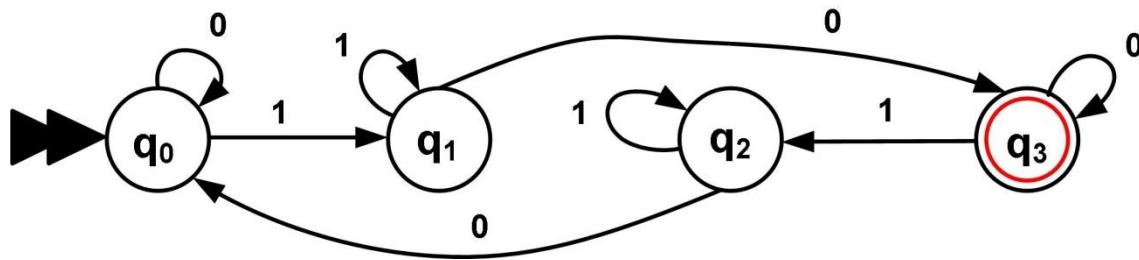
S	I	0	1
q₀	q₀	q₁	
q₁	q₃	q₁	
q₂	q₀	q₂	
q₃	q₃	q₂	

→

La primera es la versión original y la segunda es la misma pero reducida y expresada de acuerdo a su correspondiente equivalencia en la nomenclatura de este curso. Los valores de los dos bits originales se traducen a decimal y constituyen el subíndice de los estados.

Aquí q_3 es el estado de aceptación porque en ese caso las dos entradas para la compuerta AND etiquetada como F son 1 y de esa manera es activada la salida del circuito.

Traduciendo los valores de la segunda versión de la tabla a un diagrama de transición se obtiene el Autómata Finito que se muestra a continuación.



En este problema se muestra el circuito original, las tablas de transición obtenidas (considerando la operación de las compuertas lógicas) y el diagrama de transición resultante.

¿Qué ventajas tiene el modelo representado en este Autómata y que no tiene el diagrama del circuito original? ¿Será más fácil el análisis del funcionamiento del circuito en el Autómata que en el diagrama electrónico digital?

Como ya se había referido, los circuitos electrónicos secuenciales, son muy propicios para ser representados por Máquinas y Autómatas de Estado Finito, ya que en ellos la entrada por sí sola no determina la salida, sino que ésta se define también por los estados internos presentes en el circuito. Consultar textos sobre la materia y ver cómo se emplean esos modelos, no sólo para la representación de estos circuitos, sino también para su diseño.

Una ventaja de modelar matemáticamente los circuitos digitales secuenciales es que el análisis es más sencillo en el diagrama de transición que en el diagrama electrónico. Además, este hecho ha permitido la creación de aplicaciones como Workbench, Multisim o bien OrCAD Pspice, donde se trabaja con un modelo matemático del circuito real.

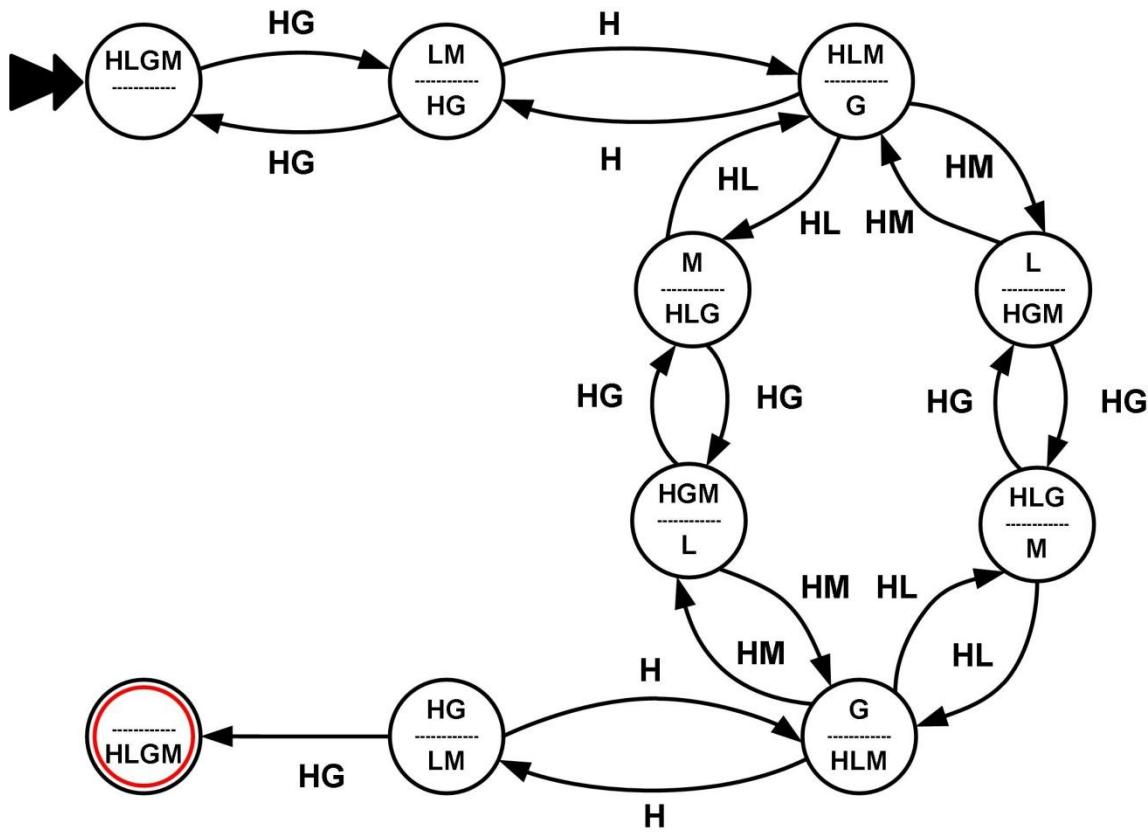
APLICACIONES EN INTELIGENCIA ARTIFICIAL (ESPACIOS DE ESTADOS)

En la orilla norte de un río se encuentra un hombre junto con un lobo, una gallina y una cubeta con maíz. Hay un bote con capacidad suficiente para transportar al hombre y uno solo de los otros tres elementos. Todos ellos deben cruzar al río sin que queden solos lobo-gallina o gallina-maíz o lobo-gallina-maíz por razones obvias. ¿Cómo se resuelve el problema? ¿Cuántas soluciones posibles existen en este caso?

Representamos a los protagonistas con su letra inicial: Hombre (H), Lobo (L), Gallina (G) y Maíz (M). Las entradas serían los elementos del conjunto $I = \{ H, HL, HG, HM \}$ que se interpretan como las siglas de los pasajeros que están cruzando el río en cualquiera de los dos sentidos posibles, debiendo en todos los casos viajar el hombre, solo o acompañado.

Existen combinaciones imposibles para los símbolos de entrada, como LM, porque el lobo no puede conducir la embarcación y transladar el maíz sin la intervención del humano. Aquí se indica quién está cruzando el río en cualquiera de los dos sentidos.

En cada estado se muestra al separarlos en dos renglones, qué personajes están en cada una de las orillas, tal que en el estado inicial todos se encuentran del mismo lado, y en el de aceptación se encuentran en el otro; de esta manera mostramos que existen dos soluciones mínimas posibles para este problema. A esta representación, que sirve para representar la búsqueda de las soluciones, se le llamaría en Inteligencia Artificial un *Espacio de Estados*.

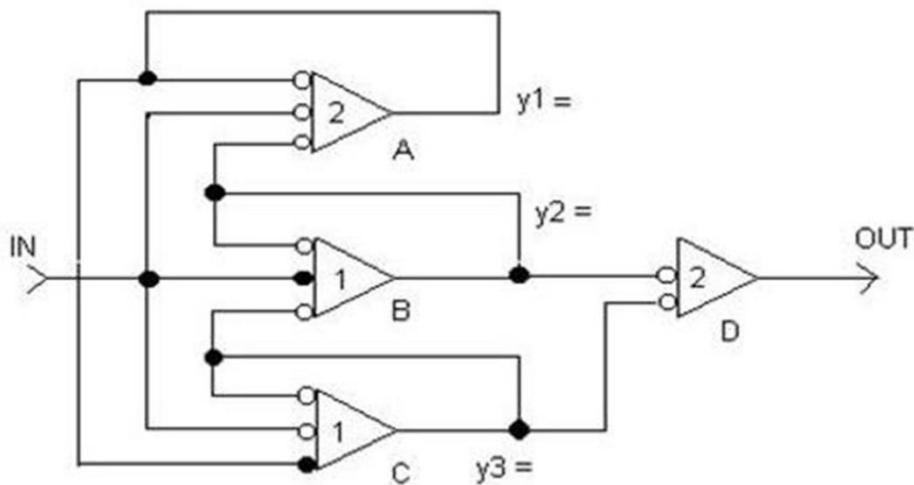


APLICACIONES EN INTELIGENCIA ARTIFICIAL (REDES NEURONALES)

Históricamente, los Autómatas Finitos se utilizaron por primera vez para modelar Redes de Neuronas. Diseñar un Autóma Finito cuyo comportamiento sea equivalente a la Red Neuronal de la figura mostrada al inicio de la siguiente página. Los estados de aceptación del modelo corresponden a una salida 1 de la red.

Cada neurona puede tener sinapsis excitantes (○) o inhibitorias (●) en sus diversas entradas, de acuerdo al modelo fisiológico correspondiente a las conexiones entre las neuronas en el cerebro humano. Se recomienda al lector que consulte en una referencia sobre la materia de manera conjunta a este problema para entenderle mejor.

Una neurona produce una salida 1 si el número de sinapsis excitantes con entrada 1 excede al de las inhibitorias con entrada 1, por al menos el valor del umbral de la neurona (el número que se encuentra dentro del triángulo). Supóngase que existe tiempo suficiente entre cada cambio de valor de entrada para que las señales se propaguen y para que la Red alcance una configuración estable.



En realidad la mayoría de las Redes Neuronales actualmente no son tan sencillas como la que aparece en este ejemplo, pero de aquí se pueden obtener conceptos muy importantes para entender cómo opera una conexión más compleja. Obsérvese la gran similitud que existe en su funcionamiento con los circuitos digitales secuenciales.

El factor más importante, que determina en gran medida el hecho de poder emplear un Autómata Finito para este caso, es el de presentarse una retroalimentación, lo cual implica que los valores actuales de los puntos marcados como y_1 , y_2 y y_3 determinan un estado actual de la red de neuronas, el cual influye en el estado siguiente de los puntos referidos así como en el valor de la salida.

$S \setminus I$	0	1
$y_1 \ y_2 \ y_3$	$y_1 \ y_2 \ y_3$	$y_1 \ y_2 \ y_3$
0 0 0	0 0 0	0 0 1
0 0 1	0 1 1	0 0 1
0 1 0	0 1 0	1 0 1
0 1 1	0 1 1	1 1 1
1 0 0	0 0 0	1 0 0
1 0 1	0 1 0	1 0 1
1 1 0	1 1 0	1 0 0
1 1 1	1 1 0	1 1 1

$S \setminus I$	0	1
$q_0 \ q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7$	$q_0 \ q_3 \ q_2 \ q_3 \ q_0 \ q_2 \ q_6 \ q_7$	$q_1 \ q_1 \ q_5 \ q_7 \ q_4 \ q_5 \ q_4 \ q_7$
q_0	q_0	q_1
q_1	q_3	q_1
q_2	q_2	q_5
q_3	q_3	q_7
q_4	q_0	q_4
q_5	q_2	q_5
q_6	q_6	q_4
q_7	q_6	q_7

Se considera como estado inicial al caso en que los tres puntos considerados tienen un valor de 0 o, lo que es lo mismo, se encuentran con ausencia de potencial.

Cuando y_2 y y_3 están en un valor de 1, sin importar el valor presente en y_1 , se activa la neurona D, porque sólo esos dos potenciales están alimentando a la entrada de la misma. Esta situación se presenta cuando se está en el estado q_3 (011) o en el q_7 (111) exclusivamente, por lo que tales son los estados de aceptación.

Como ejercicio para el alumno queda diseñar el diagrama de transición y analizar cómo se puede interpretar el hecho de que dos estados queden inconexos (q_2 y q_5). **Eso significa que dichos estados, considerados en un principio como posibles, en realidad son inalcanzables.**

AUTÓMATA FINITO DETERMINISTA (AFD)

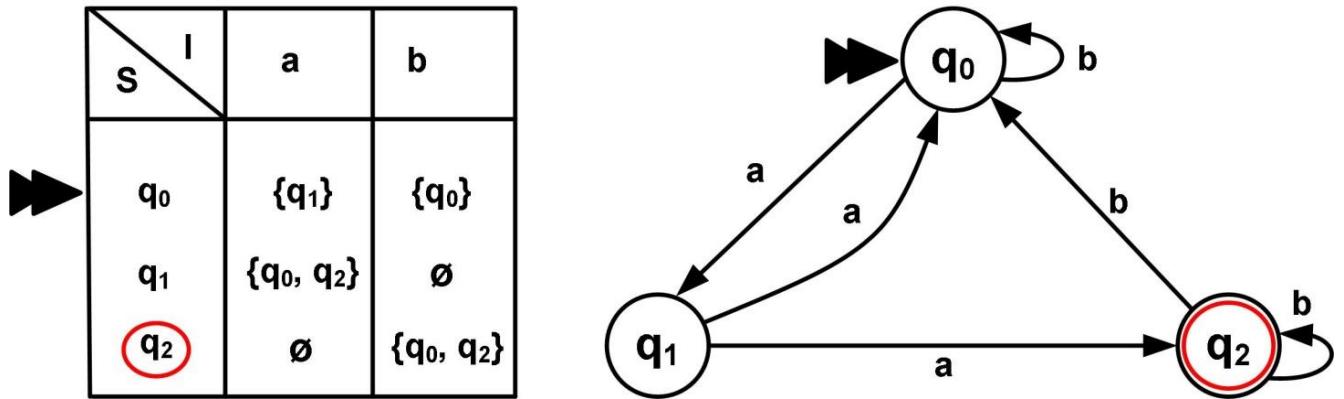
Es un dispositivo formal que puede estar en uno cualquiera de un número finito de estados, uno de los cuales es el estado inicial y por lo menos uno es un estado de aceptación. A este dispositivo está asociado un flujo de entrada que consiste en una secuencia de símbolos de un Alfabeto determinado.

El modelo tiene la capacidad de detectar los símbolos conforme llegan y, basándose en el estado actual y el símbolo recibido, ejecutar una transición, que puede consistir en un cambio a otro estado o la permanencia en el actual.

La representación de un AFD no debe contener ambigüedades, de lo cual le viene el nombre de determinista; el nombre de finito es debido a que aunque analice una cantidad infinita de posibles cadenas de entrada, la cantidad de estados es limitada. **Los Autómatas Finitos estudiados en el curso hasta este momento son de este tipo.** Por cada símbolo de entrada hay un arco que sale de cada estado del modelo.

AUTÓMATA FINITO NO DETERMINISTA (AFND)

Se llama así cuando en el Autómata se presenta en por lo menos una ocasión el siguiente caso: de algún estado parten arcos múltiples con el mismo símbolo x, si x es un dato de entrada; la situación es no determinista, ya que no es seguro el siguiente estado. Un ejemplo de un AFND, con su tabla de transición correspondiente, sería:



El no determinismo ocurre en virtud de que estando en q_1 y llegando una a, o bien, estando en q_2 y llegando una b, no sabemos con certeza cuál será el siguiente estado de entre los dos posibles.

Dada una cadena de entrada S_{in} , se considera que tal arreglo **es aceptado si existe por lo menos un recorrido en el Autómata que lleve a un estado de aceptación**, sin importar que existan otros para la misma cadena que terminen en uno de no aceptación.

Por ejemplo, sea la cadena de entrada $S_{in} = aab$. En este caso hay tres formas de recorrer el Autómata y como en una de ellas sí se finaliza en un estado de aceptación, entonces se considera que es una cadena aceptada. Sin embargo, si se recibe una cadena como abbaab, obsérvese que cuando llega el segundo símbolo de entrada (b) se tiene una situación en la que se está en un estado (q_1) en el cual no está definido lo que sucede para esa b; en ese caso la cadena de entrada se rechaza.

Este tipo de Autómatas no puede implementarse con el algoritmo que se estudió anteriormente en este documento. Sin embargo, es imprescindible estudiar este modelo, ya que se presenta en muchos casos prácticos y no se puede evitar su aparición en diversos casos; afortunadamente, existen procedimientos para transformarlos en Autómatas Finitos Determinísticos equivalentes, como se estudiará más adelante. Además existen algunos problemas en los que el AFND ofrece una mejor solución que su versión equivalente en AFD.

Consultar con el profesor o en otra fuente de consulta, a qué se refieren los términos *Determinístico*, o *No Determinístico*. Originalmente implican connotaciones filosóficas muy importantes, referentes al destino de los seres vivos.

RELACIÓN ENTRE AUTÓMATAS Y GRAMÁTICAS

Una Gramática Regular y un Autómata de Estado Finito se relacionan entre sí, ya que cualquiera de ellos se vincula directamente con un determinado Lenguaje Regular. Hay una estrecha relación entre una **Gramática Regular** que produce un determinado **Lenguaje Regular** y el **Autómata Finito** que acepta exactamente las cadenas de ese mismo Lenguaje.

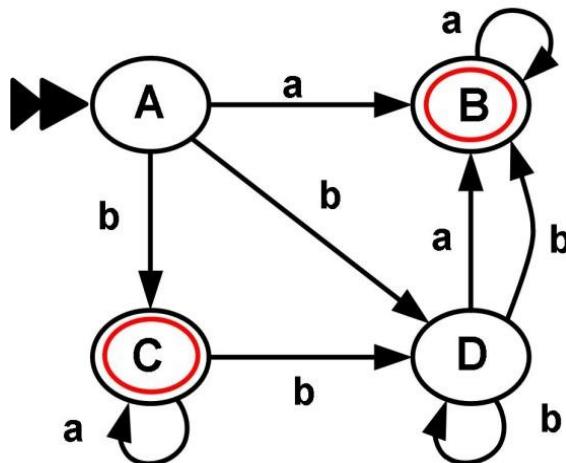
Dado el Autómata que reconoce números reales y exponentiales sin signo, analizado previamente, se puede construir una Gramática Regular que produce tal Lenguaje, por ejemplo. Este hecho favorece a los alumnos de un curso de compiladores, ya que pueden diseñar con el Autómata un compilador personal y de ahí se deducen las reglas gramaticales.

Para definir la Gramática del Lenguaje Regular aceptada por un Autómata Finito, se sigue el siguiente procedimiento:

- Los símbolos de **entrada** de A son los **terminales** de G.
- Los **estados** se convierten en los símbolos **no terminales**.
- El **estado inicial** se transforma en el **símbolo inicial**.
- Las **composiciones** corresponden a los **arcos dirigidos**. Si existe un arco con la entrada **x de A hacia B** se incluye en la Gramática Regular la regla de producción **$A \rightarrow xB$** . Además, si esa misma transición se dirige **hacia un estado de aceptación**, por ejemplo si B lo fuera en este caso, se incluye la composición **$A \rightarrow x$** , aparte de la anterior.

EJEMPLO:

Determinar la Gramática Regular que produce el Lenguaje Regular aceptado por el Autómata Finito que se muestra a continuación.



$$N = \{ A, B, C, D \}$$

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

$$\sigma_0 = \{ A \}$$

$$P = \{ A \rightarrow aB \mid bC \mid bD, \quad B \rightarrow aB,$$

$$\begin{array}{ll} C \rightarrow aC \mid bD, & D \rightarrow aB \mid bB \mid bD, \\ A \rightarrow a, A \rightarrow b, B \rightarrow a, C \rightarrow a, D \rightarrow a, D \rightarrow b \end{array}$$

Si solamente existieran los Autómatas Finitos Determinísticos, ¿Cómo se podría definir el Autómata Finito asociado con la Gramática Regular anterior, por ejemplo?

Es muy recomendable en los casos en que una Gramática Regular sea difícil de obtener, mejor se diseñe el Autómata Finito que acepta las cadenas y posteriormente obtener las reglas gramaticales del diagrama de transición como se ha expuesto.

PROBLEMA INVERSO:

PARTIENDO DE UNA GRAMÁTICA REGULAR, DETERMINAR EL AUTÓMATA FINITO QUE ACEPTE LAS CADENAS DE $L(G)$.

Aplicando el procedimiento a la inversa se puede resolver con los mismos conocimientos debiendo siempre poder obtener el Autómata, aunque en muchos casos se deberán hacer algunas consideraciones para que la respuesta sea adecuada y también considerar que pudiera resultar un AFND que debiera ser transformado en otro equivalente pero como AFD.

EJEMPLO:

Sea la Gramática $G = (N, T, P, \sigma_0)$.

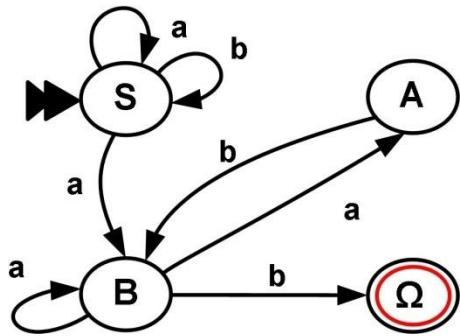
$$N = \{ S, A, B \} \quad T = \{ a, b \} \quad \sigma_0 = \{ S \}.$$

$$P = \{ S \rightarrow aS \mid bS \mid aB, A \rightarrow bB, B \rightarrow aB \mid aA \mid b \}$$

Diseñar el AFND que acepta los arreglos de $L(G)$ aplicando el procedimiento anterior en sentido inverso y obtener conclusiones.

En este caso se observa que sería imposible diseñar un Autómata Finito siguiendo el procedimiento anterior al pie de la letra y en sentido inverso, ya que debería haber un arco que sale del nodo B hacia un estado de aceptación cualquiera cuando ocurre la entrada b, pero esto obligaría a que existiera una composición en el conjunto P con el formato $B \rightarrow b?$ (donde el signo de interrogación correspondería a un no terminal) pero ésta no existe.

Se resuelve el problema agregando un estado adicional, que denominaremos Ω por llamarle de alguna manera y que es una especie de "estado vacío", además que será el único de aceptación. Si el estado Ω no es agregado, no podrá resolverse el problema. Por cierto, su nombre es determinado de manera arbitraria y no existe razón para que se llame de esa manera en particular.



No obstante que parece que no corresponde este diagrama con la Gramática de acuerdo a lo expuesto anteriormente, no existe incongruencia en el diseño, ya que si se hiciera el diseño de la Gramática Regular que produce el Lenguaje aceptado por el AFND anterior, la regla $B \rightarrow b \Omega$ se deberá eliminar del resultado, ya que Ω no produciría símbolos.

El algoritmo general para hacer la conversión se determina de la siguiente forma:

Se tiene $G = (N, T, P, \sigma_0)$, la cual es una Gramática Regular. Sean

$$I = T$$

$$S = N \cup \{ \Omega \}, \text{ donde } \Omega \notin (N \cup T) \text{ si fuera necesario } \Omega.$$

$$f(A, x) = \{ B \mid A \rightarrow xB \in P \} \cup \{ \Omega \mid A \rightarrow x \in P \}$$

$$A = \{ \Omega \}$$

$$\sigma_0 = \{ \sigma_0 \}$$

Entonces el Autómata Finito obtenido acepta precisamente los arreglos de $L(G)$.

En este caso los símbolos de entrada se toman del conjunto T , mientras que los estados son los no terminales junto con el nuevo estado de aceptación, si es que éste fuera necesario de incluir, **pues hay ocasiones en que uno o más de los mismos símbolos de N pudieran ser de aceptación y no se requeriría de agregar a Ω** . Además el estado inicial es el mismo símbolo inicial. Finalmente la función f se define de acuerdo a las composiciones pero consideradas de manera inversa de cuando se hacia la conversión del Autómata Finito en Gramática Regular.

Con esto, queda completada la relación que ya se había mencionado anteriormente: dada una **Gramática Regular**, que produce un **Lenguaje Regular**, existe también un **Autómata de Estado Finito** que acepta las cadenas producidas por dicha Gramática.

TEOREMA: Un Lenguaje L es Regular si y sólo si existe un AFD que acepta únicamente las cadenas de ese Lenguaje. Debido a que en este curso se aceptan solamente las reglas $A \rightarrow xB$ y $A \rightarrow x$ para diseñar una Gramática Regular, entonces el Autómata no podrá tener su estado inicial como de aceptación al no aceptar la cadena vacía.

Si se empleara el criterio de que se pueden emplear las reglas $A \rightarrow \lambda$ en el diseño, entonces el AFD sí podrá tener el ya referido estado como de aceptación.

CONVERSIÓN DE UN AFND EN UN AFD

Los Autómatas Finitos No Determinísticos son imprescindibles en muchas aplicaciones de la Teoría de la Computación y no solamente cuando se emplean para diseñar Gramáticas Regulares. Sin embargo, es evidente que no pueden traducirse en un programa de computadora como el que se expuso anteriormente, ya que un algoritmo que se pretende programar no debe presentar ambigüedades y en este caso en algunas transiciones hay dos o más posibles estados siguientes y no uno solo. Sin embargo, esto no es problema, ya que todo AFND puede transformarse en un AFD equivalente, el cual sí puede implementarse.

TEOREMA: Dado un AFND cualquiera llamado A , puede construirse un AFD denominado A' que sea equivalente a A .

PROCEDIMIENTO: Sea $A = (I, S, f, A, \sigma_0)$ un AFND. Evalúense los siguientes conjuntos:

$$I' = I$$

$$S' = P(S)$$

$$A' = \{ X \subseteq S' \mid X \cap A \neq \emptyset \}$$

$$\sigma'_0 = \{ \sigma_0 \}$$

$$f(X, x) = \begin{cases} \emptyset & \text{si } X = \emptyset \\ \cup f(S, x) \quad \forall S \in X & \text{si } X \neq \emptyset \end{cases}$$

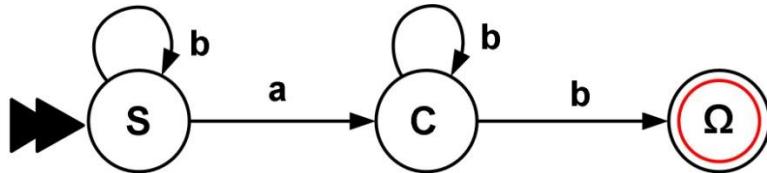
Entonces se obtiene el AFD llamado $A' = (I', S', f, A', \sigma'_0)$, el cual es equivalente al Autómata A . En este caso A' consiste en los subconjuntos de S' que contienen al menos un estado de aceptación del AFND original.

En este caso, debido a que el procedimiento para la conversión es algo extenso, se ha determinado que en estas notas no se muestre en detalle cómo se resuelven diversos ejemplos específicos, más que el que se muestra a continuación y que es muy sencillo, pero que si se comprende bien servirá perfectamente para resolver otros problemas más complejos.

EJEMPLO:

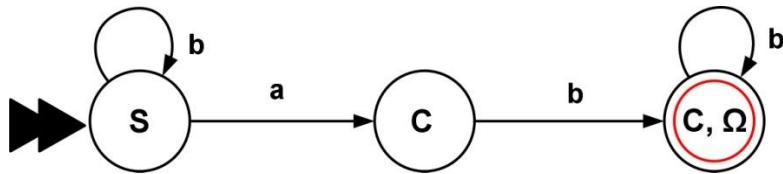
Diseñar un AFD que reconozca las cadenas del Lenguaje generado por la siguiente Gramática: $G = (\{S, C\}, \{a, b\}, \{S \rightarrow bS \mid aC, C \rightarrow bC \mid b\}, \{\Omega\})$. Encontrar la caracterización del Lenguaje $L(G)$.

El AFND obtenido por el procedimiento aprendido anteriormente es:



Se podría pensar que haciendo que C fuera estado de aceptación, no se requeriría de Ω , pero eso no es posible porque entonces la Gramática G debería contener la regla $S \rightarrow a$ y eso no sucede.

La conversión de AFND a su equivalente de acuerdo con el procedimiento referido produce como resultado el siguiente AFD:



El Lenguaje aceptado, tanto por el AFND original como por el AFD equivalente, es $L = \{b^*ab^+\}$. El procedimiento para llegar a ese resultado consiste de los siguientes criterios:

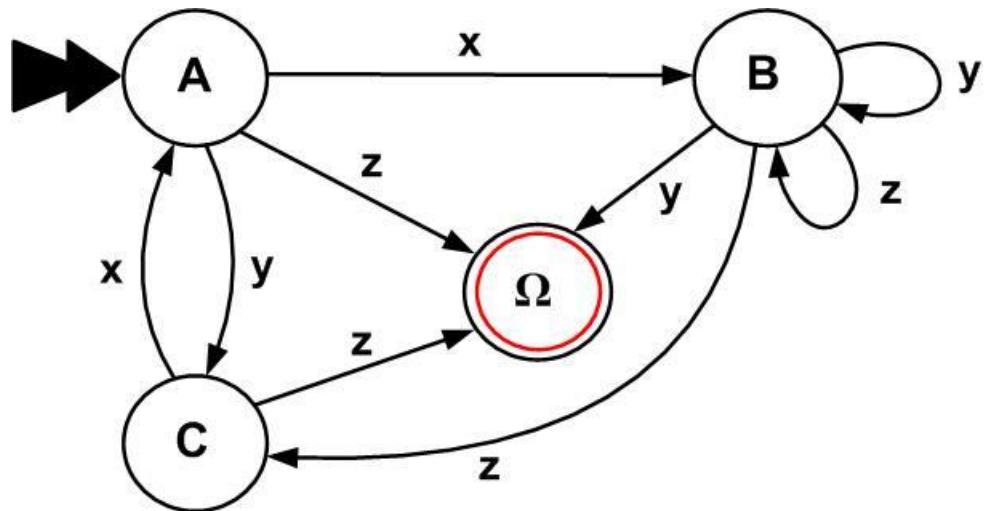
- En los casos en los que no existe problema de no determinismo se redibuja el diagrama igual a como aparece el original AFND.
- Cuando se presenta el caso de que exista el no determinismo se dibuja un nuevo estado combinado a partir de los posibles estados siguientes. Para este caso $f(C, b) = \{C, \Omega\}$ por lo que se creó un nuevo estado combinado formado por tales estados. Éste debe ser de aceptación si alguno de los estados componentes lo es, como en este caso en que $\{C, \Omega\}$ es de aceptación porque Ω lo define de esta manera.
- Completar el diseño considerando los estados siguientes para cada entrada diferente en los estados combinados que surgen en el paso anterior, como en este caso sucede con $\{C, \Omega\}$. Verificar los posibles estados siguientes como si fuera por separado cada uno de los estados para cada entrada involucrada y luego combinarlos. En este caso $f(C, a) = \emptyset$ y $f(\Omega, a) = \emptyset$, por lo que $f(\{C, \Omega\}, a) = \emptyset$; por otra parte $f(C, b) = \{C, \Omega\}$ y $f(\Omega, b) = \emptyset$, por lo que $f(\{C, \Omega\}, b) = \{C, \Omega\}$.

Nótese que el estado $\{C, \Omega\}$ hereda de C el lazo y de Ω el hecho de ser de aceptación por lo que cada uno de ellos se toma en cuenta de alguna manera.

EJEMPLO:

Diseñar un AFD que acepte las cadenas del Lenguaje generado por la siguiente Gramática: $G = (\{A, B, C\}, \{x, y, z\}, \{A \rightarrow xB \mid yC \mid z, B \rightarrow yB \mid zB \mid zC \mid y, C \rightarrow xA \mid z\}, \{A\})$.

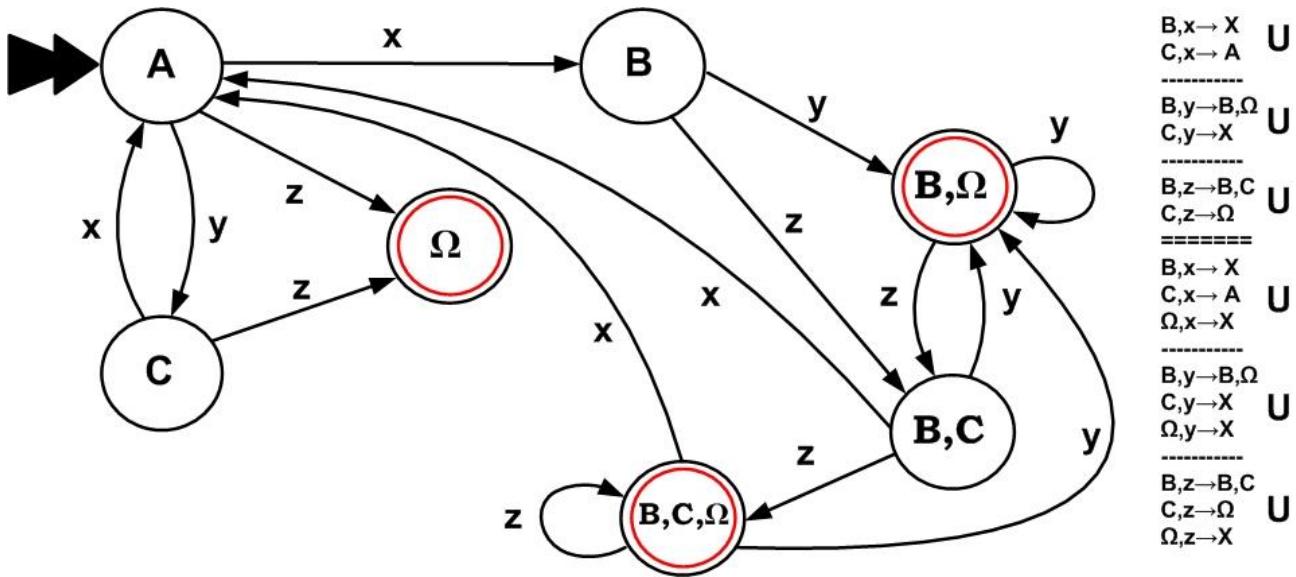
El diseño del AFND que resulta aplicando el procedimiento ya expuesto produce el siguiente diagrama de transición:



Como ya se podría prever el modelo resultó ser No Determinístico. Los dos casos que lo ocasionan son $f(B, y) = \{B, \Omega\}$ y $f(B, z) = \{B, C\}$ por lo que ya desde este momento se puede conocer que surgirán dos estados nuevos combinados que serán de B con Ω y de B con C, que se llamarán respectivamente (B, Ω) y (B, C) . El primero de ellos será de aceptación debido a que contiene a Ω que como estado individual sí lo es. El segundo se compone de dos estados de no aceptación por lo que éste no lo será.

Al desarrollar el AFD equivalente surge un tercer estado combinado que es el (B, C, Ω) que por razones ya explicadas anteriormente se observa que es de aceptación.

El diseño del AFD equivalente queda como sigue:



Cuando una de las transiciones consiste en el rechazo de la cadena se expresa con una X en el análisis de la transición compuesta y que se muestra a la derecha del diagrama.

Cuando en la siguiente transición se hace la unión de uno o más estados componentes con la mencionada X, se descarta el rechazo de la cadena porque así se considera el criterio para los AFND.

Es importante considerar que cuando surge un estado combinado se heredan características de los estados componentes. Por ejemplo cuando se tiene el caso de (B, Ω) se observa que este estado es de aceptación como lo es Ω pero no lo es B. Este último sin embargo es quien decide cual es el estado siguiente ya que las transiciones son idénticas si se observa para B que para (B, Ω); lo anterior ocurre porque Ω no tiene transiciones de salida.

Inclusive el lector observara que las transiciones son iguales para (B, C) que para (B, C, Ω) en el caso de las tres entradas como se señala:

$$\begin{aligned}
 f((B, C), x) &= \{A\} \\
 f((B, C, \Omega), x) &= \{A\} \\
 f((B, C), y) &= \{B, C\} \\
 f((B, C, \Omega), y) &= \{B, C\} \\
 f((B, C), z) &= \{B, C, \Omega\} \\
 f((B, C, \Omega), z) &= \{B, C, \Omega\}
 \end{aligned}$$

Sin embargo (B, C) es un estado de no aceptación y (B, C, Ω) sí lo es.

Pudieron haber surgido varios estados combinados aparte de los tres que aparecieron, pero la cantidad máxima posible es de $2^4 = 16$ estados en total incluido el que se omite por ser de no aceptación y que no se puede salir del mismo con cualquier entrada.

AUTÓMATAS PARA EXPRESIONES REGULARES

1. Unión de Lenguajes: $L_1 \cup L_2$. La unión de dos Lenguajes Regulares es otro Lenguaje Regular.

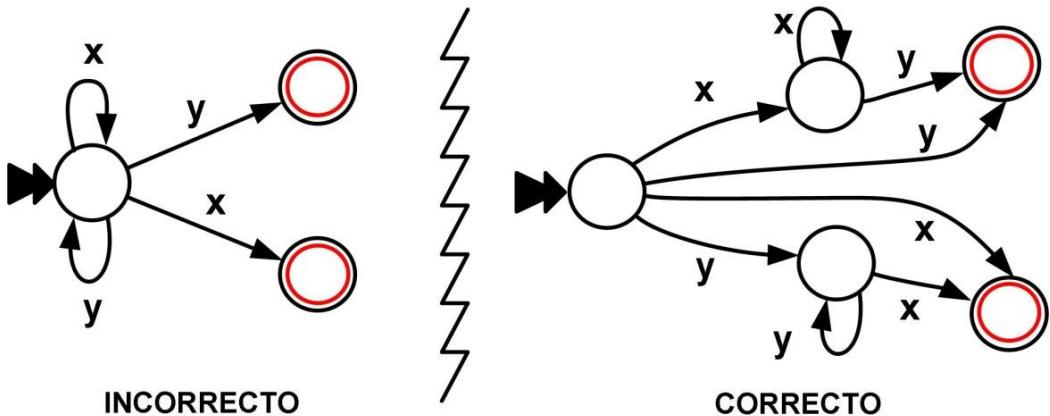
EJEMPLO:

Sean $L_1 = \{ x^*y \}$ y $L_2 = \{ y^*x \}$. Evaluar $L_1 \cup L_2$, y diseñar el Autómata Finito que acepta esas cadenas.

$$L_1 \cup L_2 = \{ x^*y \vee y^*x \}.$$



Observemos las dos opciones que normalmente los alumnos presentan como posible respuesta, de las cuales solamente la segunda sería la correcta:



Entrando a la parte correspondiente del diagrama según el primer símbolo de la cadena, ya no se puede pasar a la otra parte. El nuevo estado inicial sería de aceptación sólo si uno de los iniciales originales lo fuera.

El procedimiento indica que se debe dibujar un arco con la misma etiqueta que tienen los que salen de los que eran los estados iniciales, pero ahora partiendo del nuevo estado inicial, lo cual se considera por si los símbolos afectados por los asteriscos no aparecieran.

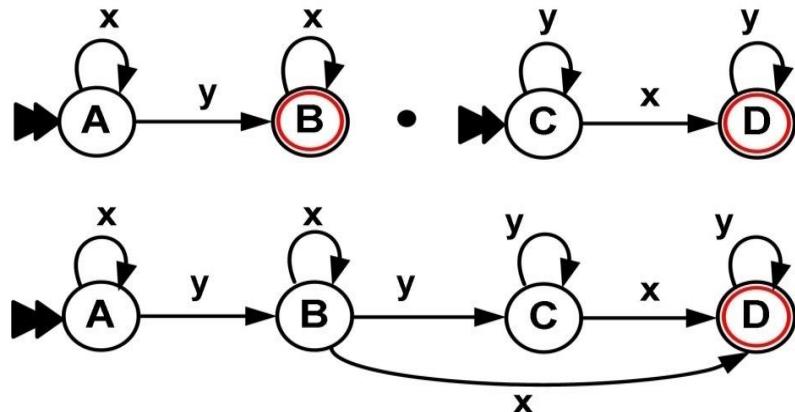
2. Concatenación de Lenguajes: $L_1 \bullet L_2$. La concatenación de dos Lenguajes Regulares es otro Lenguaje Regular.

EJEMPLO:

Sean $L_1 = \{ x^*yx^* \}$ y $L_2 = \{ y^*xy^* \}$. Evaluar $L_1 \bullet L_2$, y diseñar el Autómata Finito que acepta estas cadenas.

$$L_1 \bullet L_2 = \{ x^*yx^*y^*xy^* \}.$$

El siguiente diagrama muestra cómo se haría la integración de los diagramas.



El procedimiento indica que a partir del estado de aceptación del primer diagrama, se debe dibujar un arco hacia cada estado del segundo que sea el destino de un arco del estado inicial de éste último. Rotular cada uno de estos arcos con la etiqueta del arco correspondiente en el segundo esquema. El estado de aceptación del primero seguiría siéndolo sólo en el caso de que el estado inicial del segundo lo fuera.

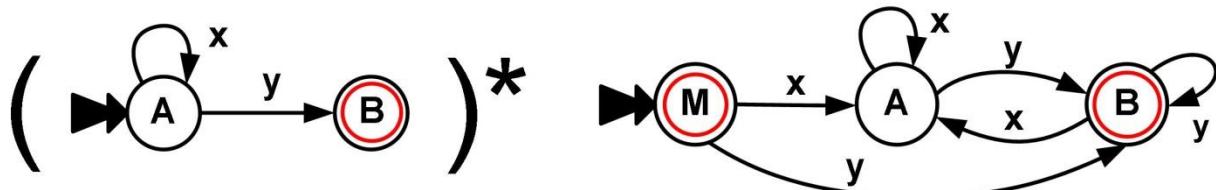
3. Estrella de Kleene o Cerradura de un Lenguaje: (L_1)^{*}. La cerradura de cualquier Lenguaje Regular es también un Lenguaje Regular.

EJEMPLO:

Sea $L = \{ x^*y \}$. Evaluar L^* y diseñar el Autómata Finito que acepta esas cadenas.

$$L^* = \{ (x^*y)^* \}.$$

El Autómata Finito que se obtiene como resultado sería el siguiente:



El procedimiento indica que el nuevo diagrama de transición implica la concatenación del esquema original hacia atrás consigo mismo, además de que debe aceptar λ .

Como debe aceptar la cadena vacía se antepone al estado inicial otro nuevo y que sea de aceptación. Para conectar ambos estados se emplea la primera entrada potencial que sería la x , por lo que dibujamos un arco que con tal entrada señale la primera transición.

Sin embargo, como se considera que podría no haber tales símbolos al inicio, como sucede cuando se tiene la cerradura, se conecta del nuevo estado inicial hacia el final con la entrada y .

Para que puedan aceptarse las repeticiones de las subcadenas se añade un arco del estado de aceptación final a cada estado que es el destino de un arco del estado inicial. Cada uno de estos nuevos arcos se rotula con la misma etiqueta que corresponda al arco del estado inicial.

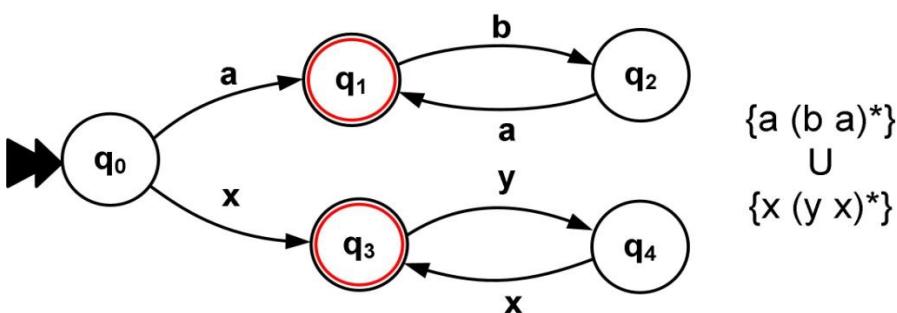
Dado un Alfabeto particular y usando las tres operaciones señaladas anteriormente, se pueden construir varios Lenguajes a partir de bloques de construcción, siendo todos ellos de tipo 3.

TEOREMA:

Si p, q, r son expresiones regulares, entonces $(p \bullet q)^*$, $(q \cup p) \bullet r$, $q^* \cup (r \bullet q)$ y otras diversas combinaciones también son expresiones regulares.

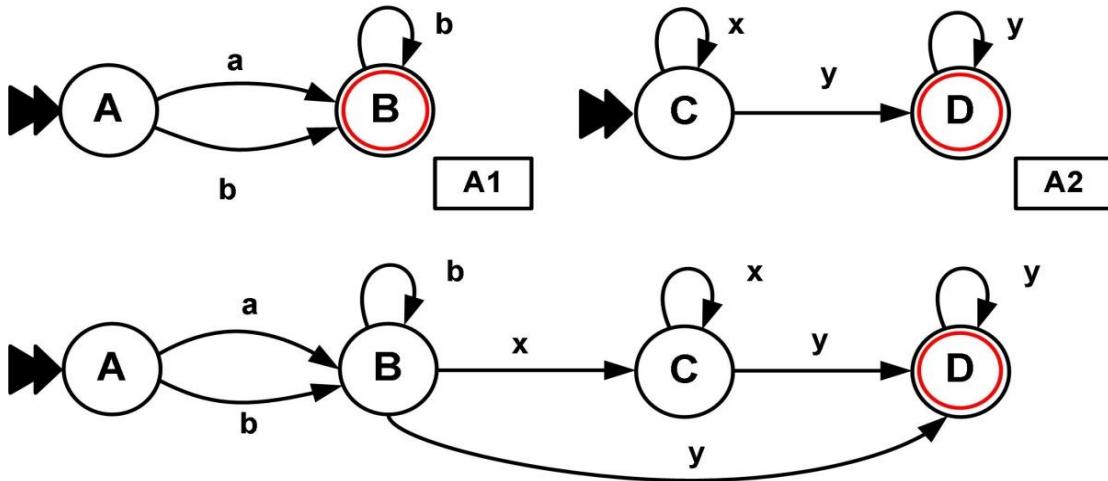
EJEMPLO:

Representar gráficamente el Autómata de Estado Finito que acepta las cadenas de la unión de $a \bullet (b \bullet a)^*$ con $x \bullet (y \bullet x)^*$.



EJEMPLO:

Dibujar un diagrama de transiciones que acepte la concatenación del Lenguaje aceptado por A_1 con el aceptado por A_2 .



El estado B del Autómata Finito resultante hubiera sido de aceptación solamente si C en el Autómata del segundo operando del enunciado de los datos lo fuera.

COMENTARIOS ADICIONALES

Los Autómatas Finitos carecen del poder suficiente para analizar y aceptar expresiones aritméticas que contengan paréntesis correctamente anidados. Uno de estos mecanismos aceptaría expresiones, tanto incorrectas como correctas, debido a que no recuerdan cuantos paréntesis se han abierto al no contar con memoria. Solamente se podría tener un registro de dichas apariciones cambiando de estado, pero podría tenerse una cantidad enorme de paréntesis y se tendrían demasiados de ellos. Para resolver el problema anterior se requiere de un Autómata de Pila.

Se espera que el alumno tenga la iniciativa de investigar acerca de las diversas aplicaciones que tienen los Autómatas de Estado Finito y no le quede la creencia de que solamente sirven para el diseño de un analizador lexicográfico para un compilador.

CAPÍTULO 5

AUTÓMATAS DE PILA

DESCRIPCIÓN GENERAL

Como se ha hecho referencia en el capítulo anterior, *el potencial de los Autómatas de Estado Finito es limitado en razón de que no pueden ser empleados para analizar Lenguajes que no son Regulares, tales como los Libres de Contexto.* En este caso se requiere de un dispositivo formal que posea memoria para poder tener el registro de las apariciones de ciertos símbolos de entrada y compararlas en cantidad con otros símbolos.

Por ejemplo, cuando se analiza una expresión que forma parte de un programa escrito en un Lenguaje de programación y que contiene paréntesis anidados, se debe llevar un conteo de los que se han abierto para cotejar con los que se cierran y verificar que sus cantidades son iguales aparte de comprobar que no apareció uno de cierre sin que haya surgido el que corresponde a su apertura.

El nuevo modelo de Autómata tiene muchos elementos y criterios en común con el que se estudió el capítulo anterior y como se verá enseguida, se los puede utilizar para analizar cadenas en forma similar a como se usan los Autómatas Finitos. ***Una cadena que ingresa al Autómata de Pila puede ser aceptada o rechazada según el tipo de estado en el cual se puede terminar el recorrido al hacer su análisis.***

En ambos casos se define el estado inicial como de aceptación si se reconoce la cadena vacía, o no en caso contrario; si se desea se numeran los estados con la nomenclatura que los define con el nombre de q_n , siendo q_0 el inicial o bien se los puede llamar de cualquier otra manera, como por ejemplo con letras mayúsculas.

Es muy común que en los Autómatas de Pila la cadena vacía λ se emplee como símbolo de entrada para varias transiciones.

Por otra parte, los Autómatas de Pila serían NO DETERMINISTAS.

Los símbolos que pueden almacenarse en la pila (símbolos de pila del Autómata) constituyen un conjunto finito que puede incluir alguno(s) o todos los símbolos del Alfabeto de la máquina y algunos símbolos adicionales que la máquina utiliza como marcas internas, como la indicación de la **pila vacía (#)**.

Las transiciones que ejecutan los Autómatas de Pila deben ser basadas en la siguiente secuencia básica: leer un símbolo de entrada, extraer un símbolo de la pila, insertar un símbolo en la pila y pasar a un nuevo estado (o continuar en el mismo). Nótese que al trabajar con más información que el Autómata Finito, también se hacen más complejas sus transiciones.

Es importante recalcar que en este caso la realización de una transición no depende exclusivamente de la presencia del símbolo de entrada, pues para que ésta se pueda consumar es preciso que en el tope de la pila exista el símbolo que se pretende extraer y de no ocurrir así, no será posible llevarla a cabo.

El estado actual, el símbolo de entrada y el símbolo que se pretende extraer (ubicado en la cima de la pila), determinan conjuntamente el símbolo que se insertará en la pila y el nuevo estado.

La adición de la Pila incrementa de manera considerable el potencial de procesamiento del Lenguaje de Autómata y proporciona un marco en el cual se formulan diversos algoritmos eficientes para el análisis sintáctico.

Precisamente, los analizadores sintácticos, además de los semánticos, se desarrollan principalmente a partir de Autómatas de Pila.

DEFINICIÓN FORMAL DEL AUTÓMATA DE PILA

Un Autómata de Pila es una 6-tupla ordenada de la forma: $S = (I, S, \Gamma, T, A, \sigma_0)$ donde:

I es el conjunto de símbolos de entrada o Alfabeto de la máquina.

S es una colección finita de estados.

Γ es la colección finita de símbolos de la pila.

T es una colección finita de Transiciones del modelo.

A es un subconjunto de S y es la colección de estados de aceptación.

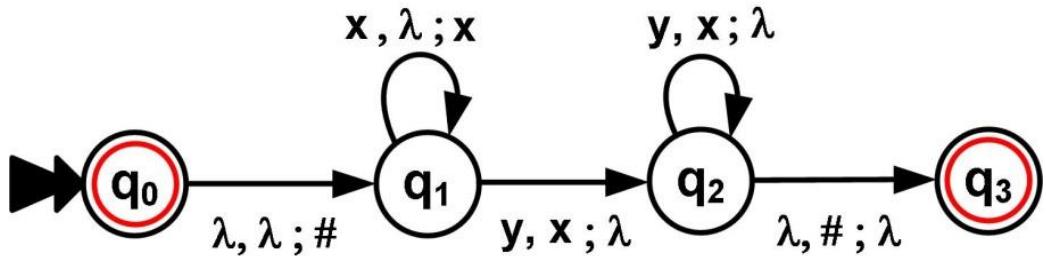
σ_0 es un elemento tomado de S y se reconoce como el estado inicial.

DISEÑO DE AUTÓMATAS DE PILA

A continuación se ofrecen ejemplos para que el lector comprenda mejor cómo se hace el diseño de casos para este nuevo modelo. Primeramente se hace el desarrollo a partir de un Lenguaje Formal y en el ejemplo final se ofrece como dato una Gramática Libre de Contexto.

EJEMPLO:

Diseñar el Autómata de Pila que acepta el Lenguaje $L = \{ x^n y^n \mid n \in \mathbb{N}_0 \}$. Observar que en este caso se deben aceptar cadenas con la misma cantidad de x y de y por lo tanto no existe un Autómata Finito que acepte estas cadenas.



Para interpretar una transición, tal como por ejemplo $y, x; \lambda$, se toma el primer elemento (y) como el símbolo de entrada que ocasiona la transición, el segundo (x) como el que se extrae de la pila y el tercero (λ) es el símbolo de pila que se inserta en ella.

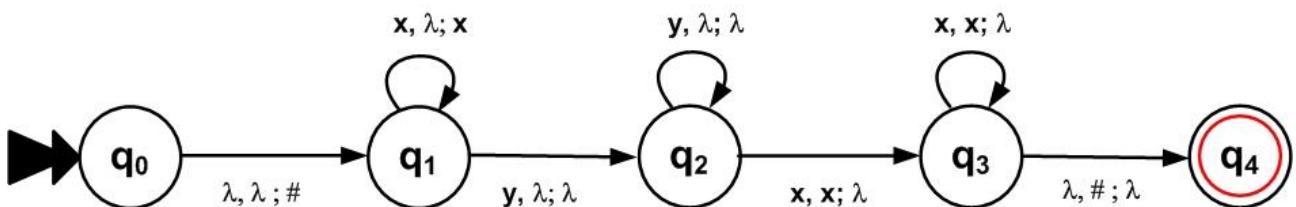
El lazo que aparece sobre q_1 sirve para insertar el símbolo x en la pila cada vez que llega uno de estos mismos símbolos. Como no deben mezclarse las x con las y se debe hacer un cambio a q_2 cuando llega la primera y , y en este nuevo estado se reciben las siguientes y ; en estos últimos casos se extraen las mismas x que se habían insertado al evaluar las x .

Las transiciones lambda que aparecen en el diagrama, son muy importantes en este diseño. La que lleva de A hacia B es para inicializar la pila con el símbolo vacío antes de empezar a recibir las x y las y . La que lleva de C a D se emplea para verificar que realmente haya quedado la pila vacía, lo cual se cumple si es posible efectuar la transición en la que se puede extraer el símbolo $\#$ una vez que se han dejado de recibir las x y las y .

En caso de que se reciban más x que y la pila no quedaría vacía y no se podría realizar la última de las transiciones, puesto que habría todavía algunas x por encima del símbolo $\#$. Si la cadena contiene más y que x entonces faltarían x en la pila, pues el símbolo λ que lleva de q_2 a q_3 sería exclusivamente el fin de cadena y no se puede efectuar la misma hasta que dejan de llevar las x y las y . Como se observará solamente se puede aceptar una cadena cuando contiene exactamente la misma cantidad de x que de y y en ese orden de aparición.

EJEMPLO:

Diseñar el Autómata de Pila que acepta las cadenas del Lenguaje $L = \{ x^m y^n z^m \mid m, n \in \mathbb{N} \}$, el cual es un Lenguaje Libre de Contexto.



En este problema se deben relacionar las cantidades de **x** y de **z**, por lo que se requiere de un Autómata de Pila para aceptar este Lenguaje. Precisamente por esa razón en el diseño se interactúa con la pila solamente cuando se reciben ambos. La cantidad de **y** es independiente de los otros dos símbolos y por ese motivo se ocasiona la extracción y la inserción simplemente de λ en la pila.

En resumen, cuando ingresan las **x** se inserta este mismo símbolo en la pila, después cuando llegan las **y** no se afectan esas **x** pero cuando llegan al final las **z** entonces sí son retiradas, logrando extraer el símbolo de pila vacía solo si se cumple con la igualdad ya mencionada.

Una duda que puede surgir en este momento es sobre los lazos en un Autómata de Pila. En uno Finito recordamos que el recorrerlos es opcional, pero si en el modelo que se está estudiando en este capítulo también fuera el caso, queda la duda de por qué cuando llegan las **x** se tiene solamente un lazo en q_1 y sin embargo debe haber al menos una de ellas.

La respuesta mencionada anteriormente es correcta, ya que aunque recorrer un lazo en un Autómata de Pila es opcional, si no se hiciera su recorrido no habría **x** en la pila y no se podría realizar la transición de q_2 a q_3 . De esta manera el recorrer el lazo para insertar la **x** se hace necesario.

Pero si no se recorriera el lazo la cadena sería rechazada... Esto podría suceder a causa del no determinismo que tienen estos modelos.

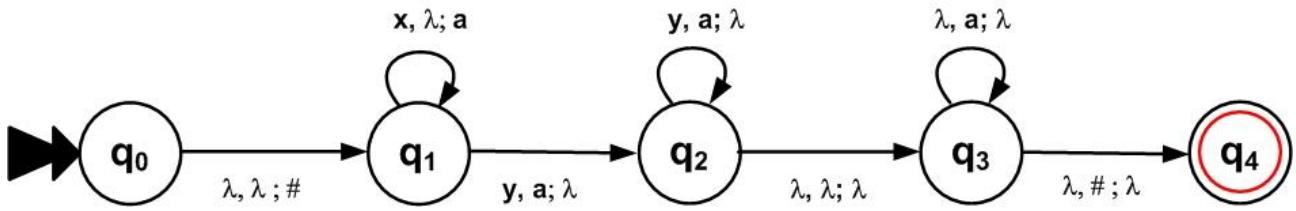
Los Autómatas de Pila que no tienen su estado inicial como de aceptación pueden aceptar Lenguajes Libres de Contexto como el del ejemplo anterior. Como los Lenguajes Regulares también cumplen con las condiciones para ser Libres de Contexto, teóricamente pueden ser manejados por Autómatas de Pila (aunque esta estructura de datos no sería empleada para algo útil durante el análisis de la cadena).

Una forma de mostrar como un Lenguaje es Libre de Contexto sería diseñando un Autómata de Pila que deberá ser el idóneo para analizar las cadenas del mismo. *Es muy importante considerar que si se cumple con el criterio en el que nos hemos basado en este curso, no se aceptaría la cadena vacía λ como parte del Lenguaje y q_0 debería ser un estado de no aceptación, pero habrá que recordar que diversos autores sí aceptan dicha palabra como parte de un Lenguaje de Tipo 2, de acuerdo a la aclaración hecha en el capítulo correspondiente.*

Es importante considerar que no todos los Lenguajes no Regulares son Libres de Contexto, por lo que se da el caso de que para algunos problemas de diseño la respuesta es que no existe un Autómata de Pila que acepte un Lenguaje determinado que se ofrece como dato, por no ser Libre de Contexto. Un ejemplo simple podría ser en el caso de un Lenguaje Formal en el que se deben relacionar las cantidades de tres símbolos.

EJEMPLO:

¿Qué Lenguaje Formal acepta el siguiente Autómata de Pila?



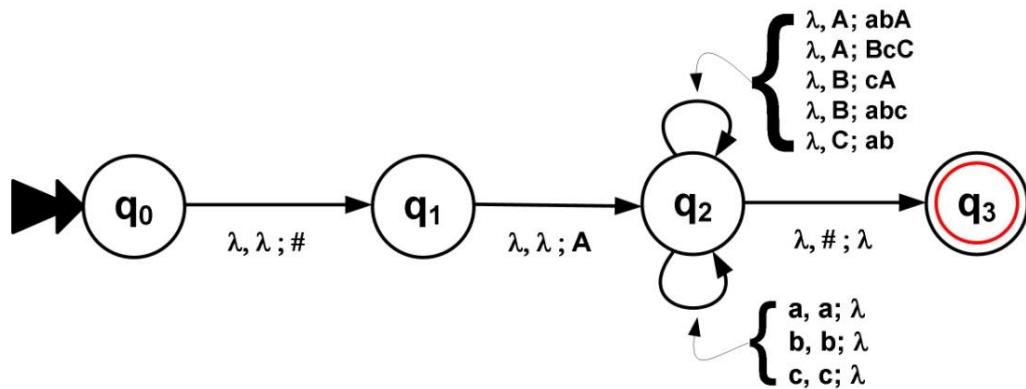
El presente modelo acepta las cadenas del Lenguaje Formal definido por $L = \{x^m y^n \mid m \geq n ; m, n \in \mathbb{N}\}$.

Este Autómata de Pila desde q_0 hasta q_2 se parece al expuesto en el primer ejemplo y es normal ya que algunas de las cadenas que debe aceptar son aquellas donde son iguales las cantidades de **x** y de **y**. Sin embargo en el segundo de los estados mencionados pueden presentarse dos casos posibles, que son que la pila esté vacía o bien que aún contenga al símbolo de pila que para este ejemplo es la **a**. Así como se emplea **a** como símbolo de pila podría haberse utilizado algún otro.

La transición vacía que lleva a q_3 es para dedicarnos en caso de que sea necesario a vaciar la pila, pero solo si fuera el caso. Se pretende entregar al final la pila vacía como estaba al principio y por ello se extraen todas las **a** que hubieran quedado según lo mostrado en el lazo del último estado mencionado pero sin recibir ni a **x** ni a **y** porque ya se agotaron en la entrada.

EJEMPLO:

Diseñar el Autómata de Pila que acepta el Lenguaje producido por la Gramática Libre de Contexto con las reglas del conjunto $P = \{ A \rightarrow abA, A \rightarrow BcC, B \rightarrow cA, B \rightarrow abc, C \rightarrow ab \}$ si se tiene que **A** es el símbolo inicial.



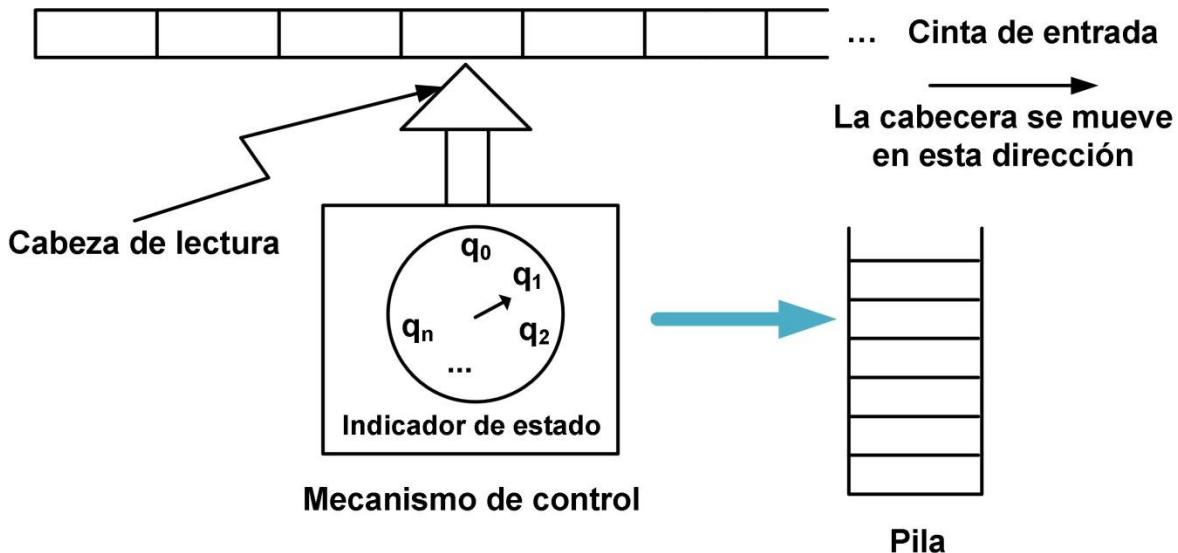
En este caso existe una propuesta de un Autómata de Pila que se diseña de manera directa **sin tener que caracterizar la Gramática** y en ella se inicia con la inserción de la pila vacía y enseguida se hace lo mismo con el símbolo inicial

cambiando de estado actual y llegando hasta q_2 ; posteriormente se incluyen en dicho estado diversas transiciones como lazos.

Unas tienen el formato λ , <lado izquierdo de la regla>; <lado derecho de la misma regla> para cada composición presente en G y en las otras se define como <símbolo terminal>, <mismo símbolo terminal>; λ para cada uno de ellos considerado en el conjunto de dichos elementos. Esta es la respuesta más simple y no requiere de mucho análisis.

TEOREMA: Para toda Gramática Libre de Contexto G , existe un Autómata de Pila S que acepta exclusivamente las mismas cadenas del Lenguaje $L(G)$.

REPRESENTACIÓN ALTERNA DE UN AUTÓMATA DE PILA



En la cinta de entrada de la máquina se coloca la cadena que se analizará, con la cabeza de lectura sobre la celda del extremo izquierdo de la cinta. Luego se pone en marcha la máquina desde su estado inicial, con la pila vacía, y se declara que la cadena se aceptará si es posible que la máquina llegue a un estado de aceptación después de leer toda la cadena. Existe un diagrama similar, pero sin la pila de memoria que corresponde al Autómata Finito.

En el transcurso de las clases ordinarias se verán más ejemplos sobre este modelo formal, el cual es menos empleado que el anterior, pero que no por ello es menos importante.

A P É N D I C E S

1.
EL PROBLEMA DE LA
FALTA DE CREATIVIDAD
DEL ESTUDIANTE DE
MATEMÁTICAS

2.
PROGRAMA VIGENTE DE
LA ASIGNATURA CC209
(TEORÍA DE LA
COMPUTACIÓN).

3.
PROBLEMAS
PROPUESTOS PARA
RESOLVER EN CLASE.

EL PROBLEMA DE LA FALTA DE CREATIVIDAD DEL ESTUDIANTE DE MATEMÁTICAS

Artículo publicado por el Mtro. Abelardo Gómez Andrade

Centro Universitario de Ciencias Exactas e Ingenierías

Universidad de Guadalajara

El contenido de este documento está relacionado con el curso CC209

I. Introducción

En el transcurso de nuestra labor como docentes, los profesores nos preocupamos porque el conocimiento que impartimos origine un aprendizaje significativo en nuestros alumnos.

Sin embargo, es triste advertir que en el área de las Ciencias Exactas, y más específicamente en las Matemáticas, en la mayoría de los casos, los estudiantes aunque comprenden muy bien los conocimientos que les impartimos, no tienen la menor creatividad para solucionar otros problemas relacionados con los que resolvemos en clase. En tal caso surge un conflicto debido a que el maestro exige que en tareas, trabajos especiales y exámenes, el alumno resuelva esos problemas "tan difíciles".

Es por ello necesario que investiguemos la razón de esa enorme falta de creatividad de los alumnos, para luego proponer soluciones; recordemos que un *Ingeniero* es tal en cuanto a que cuenta con un *ingenio* que le permite resolver cuestiones totalmente imprevisibles, en las cuales, como en los problemas escolares, deberá demostrar esa herramienta tan importante, llamada **creatividad**.

En matemáticas, de muy poco sirve la simple posesión de información; lo más importante es "saber como usarla". Saber matemáticas significa tener fluidez en el uso del lenguaje matemático, resolver problemas a partir de enunciados, encontrar demostraciones, y la que pudiera ser la actividad fundamental, que hace que ésta sea una asignatura aplicable y no solamente teórica: reconocer un concepto matemático a partir de una situación concreta determinada.

II. Deficiencias en la Enseñanza de la Matemática en Nuestros Días.

Para atacar este problema, es importante que primero analicemos cual es la causa que lo origina. Al investigar nos damos cuenta de que los planes de estudios tradicionales han recibido muchas críticas de los conoedores de la pedagogía; entre las más importantes se señalan las siguientes:

- a) Se obliga al alumno a memorizar antes que a comprender, a través de una gran cantidad de ejercicios y algunas veces se evalúan como incorrectas las respuestas dadas por el estudiante cuando se sale del patrón que le hemos enseñado.
- b) Se presentan los conocimientos desconectados entre sí, representando para el alumno temas aislados.
- c) No siempre recibe los antecedentes necesarios que marcan los planes de estudios, en razón de pérdida de clases, tratamiento extenso de algunos temas y descuido de otros, etc.
- d) Se tratan temas que han perdido actualidad, tomados de libros viejos, o de textos pobremente escritos con fines comerciales y enfocados a otro entorno diferente al del estudiante.

Morris Kline señala como el más grave defecto del plan tradicional la falta de motivación, ya que el alumno nunca ve la necesidad real de estudiar matemáticas. Se ofrecen razonamientos como los siguientes, acerca de la utilidad del estudio de esta rama del conocimiento humano que lo único que ocasionan es una sonrisa de compasión hacia el profesor:

- a) "Son temas que pueden necesitarse en cursos posteriores ...", no obstante las aplicaciones que se podrían ofrecer, en el momento de su estudio, no tienen ningún interés para los jóvenes, además de que en ocasiones el maestro no sabe explicarles por qué lo dice.
- b) "Sirven de entrenamiento mental ...", pero podría pensarse en otras técnicas más agradables para lograr el mismo fin (como ingerir un producto para agilizar la mente).
- c) "Las matemáticas son estéticamente bellas ...", pero es lógico darnos cuenta que no es posible apreciar algo que no se comprende.
- d) "Presenta un estímulo intelectual ...", pero son pocos los que pueden apreciarlo, pues para la mayoría representa más bien síntomas de masoquismo.
- e) "Se aprende resolviendo problemas... ", aunque la mayoría de éstos son inútiles, irreales y rebuscados, por lo que no convencen a nadie.

Para colmo de males, el profesor de matemáticas no les explica a sus alumnos la manera en que su asignatura se relaciona con las de computación, por ejemplo, porque no le corresponde y les dice que eso se los ha de explicar el maestro de la mencionada materia. ¡Y cuando le llega su turno al maestro de computación, nunca utiliza las bases matemáticas pues les tiene repulsión como la mayoría de la gente! No ocurre siempre, pero sí en muchas ocasiones.

En síntesis, la motivación anterior resultaría negativa y el alumno estudia matemáticas por obligación. Y esta afirmación se evidencia en el bajo nivel académico de la mayoría del alumnado de los Centros Universitarios y en la aversión hacia tales asignaturas.

III. El Problema de Saber Resolver los Problemas.

Es natural que antes de empezar a buscar la manera de solucionar problemas debemos definir qué entendemos por lo que significa tal vocablo. Un problema “plantea una situación que debe ser modelada para encontrar la respuesta a una pregunta que se deriva de una situación” (Parra, 1989). Bouvier señala que un problema deberá permitir “derivar preguntas nuevas, pistas nuevas, ideas nuevas”.

Al intentar resolver un problema, el sujeto al que se le plantea debe disponer de los elementos para comprender la situación que el problema describe y no disponer de un sistema de respuestas totalmente constituido que le permita responder de manera casi inmediata.

La resolución de un problema ocurre cuando el sujeto que lo debe resolver cree, explícita o implícitamente que ha obtenido la solución correcta. Aquí se manifiesta la coordinación de experiencias previas, conocimiento e intuición, en un esfuerzo por encontrar la solución verdadera.

Normalmente, el proceso de resolución consta de tres pasos:

- a) Entender bien el problema, y no otra cosa diferente, para lo cual el planteamiento del problema no debe contener ambigüedades.
- b) Desarrollar y llevar a cabo una estrategia eficaz para encontrar un resultado.
- c) Evaluar la solución para demostrarla o, en su caso, descartarla y buscar otra.

El alumno típico del CUCEI puede tener problemas serios para poder entender que es lo que debe hacer con un problema, y es peor si el problema está expresado en forma de enunciado. Generalmente no le encuentra conexión con la base de conocimientos que ha adquirido en las aulas, solo le queda mirar al cielo esperando la respuesta, y hasta parece como que “le doliera la cabeza al pensar”. Si ya comenzamos mal, pues indiscutiblemente que todo el desarrollo va a ser erróneo si es que se intenta alguno. Éste problema se presenta en los cursos de cálculo, álgebra, geometría, matemática finita, entre otros y curiosamente también lo pueden padecer los profesores de física y de computación. ¡Nos falta creatividad!

Curiosamente, el inciso b no es en muchos casos el que nos preocupa, pues el algoritmo para llegar a un resultado se llega a dominar en base a la realización de ejemplos típicos variados, o del conocimiento profundo de las bases matemáticas que el alumno recibe en las sesiones en el aula de clases. Aquí decididamente les va mal a los que no le ponen ganas.

Un hábito que pocas veces les inculcamos a nuestros alumnos, indebidamente, es el de no evaluar la solución encontrada. Al terminar un examen, es muy común que nuestros alumnos nos pregunten que ¿cómo vemos el resultado? o ¿es correcto? Y es digno de darnos tristeza, pues el alumno debía estar capacitado para hacer una validación, que en muchas ocasiones no es tan difícil.

Son pocos los alumnos que al resolver un sistema de ecuaciones verifican el cumplimiento de las igualdades, o los que al calcular una matriz inversa multiplican el resultado por la matriz original para ver si obtienen la matriz identidad, por ejemplo. El proceso de evaluación se puede hacer, incluso, durante el desarrollo. ¿Cuántos alumnos no se equivocan cuando utilizan el método numérico de bisección, y el punto medio les resulta fuera del intervalo a particionar, y así continúan su cálculo?

IV. Importancia del Papel Desempeñado por el Profesor.

Un factor esencial para que la resolución de problemas se convierta en una actividad interesante y productiva para los alumnos es, sin duda, el maestro. Sus acciones y el ambiente que logre crear dentro de su clase darán significado a la práctica de la resolución de problemas.

Charles (1982) afirma que "la componente ambiente del aula identifica comportamientos que el docente debiera modelar para desarrollar una atmósfera de clase propicia para la resolución de problemas de matemáticas. Debe identificar algunos comportamientos útiles para ayudar a desarrollar las habilidades del alumno para seleccionar y utilizar estrategias de resolución".

Entre los comportamientos con los que el maestro puede ayudar a crear este ambiente, merecen destacarse:

- a) Invitar a los estudiantes a explorar cualquier idea o estrategia que pueda ayudarles a entender y/o resolver un problema, sin censurar las ideas generadas. En el periodo ordinario de clases se puede hacer una comparación de las diferentes estrategias.
- b) Reconocer y estimular los diferentes tipos de habilidad o excelencia de los estudiantes.
- c) Resolver ejemplos con cierto grado de similaridad y mencionarle al estudiante que los diferentes problemas a los que se habrá de enfrentar se pueden resolver en base al mismo conjunto de conocimientos.
- d) Lograr la buena disposición del alumno frente a la tarea de resolver un problema y motivar a la perseverancia al intentar la resolución.
- e) Buscar problemas en que el alumno tenga la certeza que son aplicables al área del conocimiento específico al que se va a dedicar. Esto conlleva a resaltar el lugar de las matemáticas como una parte importante de las manifestaciones de conocimiento humano.
- f) Hacer que el alumno tenga un razonamiento lógico adecuado. He aquí un aspecto que ha sido muy descuidado por los profesores, y que en opinión personal del autor de este documento es fundamental. Donovan Johnson, en los 60's, planteaba: "En matemáticas enfatizamos la lógica y la demostración, el razonamiento deductivo, así como el inductivo, y sin embargo no las hemos usado en forma consistente y planeada para hacer nuestros nuevos programas".

V. Conclusiones.

Es agradable para un profesor responsable saber que el problema de la falta de creatividad en sus alumnos, por lo menos teóricamente, tiene solución. El problema surge cuando se intenta poner en práctica y nos damos cuenta de que ésto no es tan sencillo.

Algunos inconvenientes que impiden desarrollar en forma fácil las ideas anteriores son:

- a) El tiempo que se le dedica a un curso es breve como para asignarle tiempo suficiente a cada problema, y con mucha mayor razón en el caso de los profesores con mucha actividad extraclase.
- b) Los grupos son muy numerosos y en el momento en que hay que evaluar es demasiado lento el proceso de revisar los exámenes si los alumnos inventan métodos muy extravagantes. Hay que recordar que los resultados se requieren en el menor tiempo posible.
- c) Definitivamente que no a todos los estudiantes les gusta pensar. Hay algunos que requieren de mucho esfuerzo para convencerles de que deben tener creatividad para triunfar como profesionistas. Un facilitador que aplique estos principios, puede no ver sus frutos de manera inmediata y en la totalidad de sus pupilos.

La profesión de docente es a veces mal pagada para un profesionista, y le requiere muchos sacrificios y exigencias; a cambio de eso, es una de las que más satisfacción pueden hacer sentir a cualquier ser humano, por la enorme trascendencia que dejaremos en la vida de miles de alumnos en los que algo habremos de aportar para su formación profesional. Así las cosas, tenemos que cumplir esta vocación con dignidad y valorando la importancia de nuestra adecuada preparación para capacitar al futuro de nuestra patria.

¡Vale la pena preparar adecuadamente a quienes en nuestras manos están para ello!



DATOS DE IDENTIFICACIÓN DEL CURSO					
DEPARTAMENTO:	Ciencias Computacionales				
ACADEMIA A LA QUE PERTENECE:	Estructuras y Algoritmos				
NOMBRE DE LA MATERIA:	Teoría de la Computación				
CLAVE DE LA MATERIA:	CC209				
CARÁCTER DEL CURSO:	Obligatorio				
TIPO DE CURSO:	Teórico				
No. DE CRÉDITOS:	11				
No. DE HORAS TOTALES:	80	Presencial	64	No presencial	16
ANTECEDENTES:	MATEMÁTICAS DISCRETAS (MT260)				
CONSECUENTES:	COMPILEDORES (CC317) y ANÁLISIS Y DISEÑO DE ALGORITMOS (CC316)				
CARRERAS EN QUE SE IMPARTE:	Ingeniería en Computación, Licenciatura en Informática y Licenciatura en Matemáticas				
FECHA DE ÚLTIMA REVISIÓN:	Enero de 2012				

PROPÓSITO GENERAL

Al concluir este curso se habrá introducido al estudiante en las competencias necesarias para el diseño de modelos formales fundamentales en las Ciencias de la Computación, principalmente los Autómatas y los Lenguajes Formales, tanto de manera formal como aplicada.

OBJETIVO TERMINAL

Comprender y analizar modelos matemáticos tales como Autómatas y Lenguajes Formales para representar con ellos sistemas del mundo real, enfocando el conocimiento de manera principal al diseño de sistemas computacionales.

CONOCIMIENTOS PREVIOS

Elementos de teoría de grafos y teoría de conjuntos, además de conceptos de computación, de programación estructurada y de pilas de memoria.

HABILIDADES Y DESTREZAS A DESARROLLAR

Capacidad de razonamiento lógico y de búsqueda de soluciones en problemas formales, así como relacionar conocimiento puro con aplicado.

ACTITUDES Y VALORES A FOMENTAR

Honradez, autoformación didáctica, confianza en sí mismo y competencias para desarrollar trabajo en equipo.

CONTENIDO TEMÁTICO

MÓDULO 1. GRAMÁTICAS Y LENGUAJES FORMALES.		18 HRS
<i>OBJETIVO DEL MÓDULO:</i> Comprender la forma en que los lenguajes naturales se pueden definir matemáticamente como lenguajes formales, enfatizando los aspectos que hacen similares o diferentes a ambas representaciones.		
1.1	Conceptos introductorios.	2 HRS
<i>OBJETIVOS DEL TEMA:</i>		
	<ul style="list-style-type: none"> * <i>El alumno identificará los conceptos de Lenguaje, Gramática, Alfabeto, símbolo y de palabra, arreglo o cadena.</i> * <i>El alumno distinguirá los términos naturales y formales.</i> * <i>El alumno reconocerá la importancia de los Lenguajes Formales en las Ciencias Computacionales.</i> 	
1.2	Lenguajes Formales y sus operaciones.	3 HRS
<i>OBJETIVOS DEL TEMA:</i>		
	<ul style="list-style-type: none"> * <i>El alumno conocerá la estructura de un Lenguaje Formal.</i> * <i>El alumno diferenciará los conceptos en los Lenguajes Formales y en los Naturales.</i> * <i>El alumno conocerá las operaciones aplicables en los Lenguajes Formales.</i> * <i>El alumno resolverá operaciones en las que se involucren Lenguajes Formales.</i> 	
1.3	Gramáticas Formales y su diseño.	4 HRS
<i>OBJETIVOS DEL TEMA:</i>		
	<ul style="list-style-type: none"> * <i>El alumno conocerá e identificará los elementos que conforman una Gramática Formal en su representación como tupla.</i> * <i>El alumno diferenciará los elementos que conforman una Gramática Formal</i> * <i>El alumno diseñará Gramáticas Formales a partir de Lenguajes Formales.</i> 	
1.4	Caracterización de una Gramática.	3 HRS
<i>OBJETIVOS DEL TEMA:</i>		
	<ul style="list-style-type: none"> * <i>El alumno definirá el Lenguaje Formal a partir de una Gramática Formal.</i> * <i>El alumno identificará el concepto de Gramáticas equivalentes.</i> * <i>El alumno diseñará un Lenguaje Formal a partir de la caracterización de una Gramática Formal.</i> 	
1.5	Jerarquía de Chomsky.	3 HRS
<i>OBJETIVOS DEL TEMA:</i>		
	<ul style="list-style-type: none"> * <i>El alumno conocerá la clasificación de las Gramáticas Formales.</i> * <i>El alumno identificará los criterios empleados para distinguir los 4 tipos de Gramáticas Formales.</i> * <i>El alumno será capaz de identificar a qué tipo de Gramática pertenece un determinado Lenguaje Formal.</i> * <i>El alumno clasificará los Lenguajes Formales según este mismo criterio.</i> 	

	<ul style="list-style-type: none"> * El alumno identificará los reconocedores para cada tipo de Lenguaje Formal. * El alumno diseñará Gramáticas Formales que cumplan con un tipo específico de acuerdo a esta clasificación. 	
1.6	Conceptos adicionales sobre Gramáticas Formales.	1 HRS
	<p>OBJETIVOS DEL TEMA:</p> <ul style="list-style-type: none"> * El alumno conocerá e identificará el concepto de Gramática decidible. * El alumno diseñará una Gramática decidible. * El alumno conocerá e identificará el concepto de las Gramáticas lineales. * El alumno diseñará una Gramática lineal. 	
1.7	Forma Normal de Chomsky (CNF).	2 HRS

MÓDULO 2. LAS GRAMÁTICAS FORMALES EN LA COMPUTACIÓN.

11 HI

OBJETIVO DEL MÓDULO: Aplicar los conocimientos sobre Gramáticas Formales en la descripción de los lenguajes de programación, con el propósito de desarrollar parte de un compilador, o para entender su operación.

2.1	Forma Normal de Backus-Naur (BNF)	2 HI
	<p>OBJETIVOS DEL TEMA:</p> <ul style="list-style-type: none"> * El alumno comprenderá la aplicación de esta nomenclatura ahora desde el punto de vista formal. * El alumno conocerá la representación de las reglas en BNF para algunos lenguajes de programación. 	
2.2	Árboles de derivación.	4 HI
	<p>OBJETIVOS DEL TEMA:</p> <ul style="list-style-type: none"> * El alumno será capaz de construir árboles de derivación para la producción de cadenas en un Lenguaje formal. * El alumno comprenderá el uso y aplicación de estos árboles. * El alumno conocerá la relación que tienen los árboles de derivación con los de sintaxis y los de análisis sintáctico en la teoría para el diseño de compiladores. 	
2.3	Gramáticas ambiguas y unívocas	2 HI
	<p>OBJETIVOS DEL TEMA:</p> <ul style="list-style-type: none"> * El alumno distinguirá entre las Gramáticas ambiguas y las unívocas y comprenderá la importancia de estos conceptos en el diseño de las especificaciones sintácticas de un lenguaje de programación. * El alumno tendrá la capacidad de diseñar Gramáticas unívocas equivalentes a partir de otras que son ambiguas. 	
2.4	Antecedentes para el diseño de lenguajes de programación.	3 HI

	OBJETIVOS DEL TEMA:	
	<ul style="list-style-type: none"> * El alumno diseñará Gramáticas que consideran la asociatividad correspondiente de acuerdo a los criterios de los lenguajes de programación. * El alumno aplicará sus conocimientos de desarrollo de las reglas de un compilador para proponer un prototipo de un lenguaje de programación. * El alumno relacionará las Gramáticas formales con las definiciones léxicas y sintácticas de un lenguaje de programación. * El alumno distinguirá el analizador correspondiente para las etapas de análisis de un compilador y las vinculará con los Autómatas de la Jerarquía de Chomsky. 	

	MÓDULO 3. MÁQUINAS DE ESTADO FINITO.	6 HI
<i>OBJETIVO DEL MÓDULO:</i> Obtener las bases necesarias para el diseño de Máquinas de Estado Finito y simular con este modelo algunos sistemas reales.		
3.1	Concepto.	2 HI
<i>OBJETIVOS DEL TEMA:</i>		
	<ul style="list-style-type: none"> * El alumno conocerá el concepto formal de las Máquinas de Estado Finito, así como su descripción como tupla de conjuntos. * El alumno entenderá el significado del modelo y en qué casos se puede aplicar. 	
3.2	Representación.	2 HI
<i>OBJETIVOS DEL TEMA:</i>		
	<ul style="list-style-type: none"> * El alumno dominará las representaciones alternas para este modelo, incluyendo la tabla de transición y el diagrama de transición. * El alumno será competente para transformar una Máquina de este tipo de una forma de representación a otra. 	
3.3	Desarrollo de una aplicación.	2 HI
<i>OBJETIVOS DEL TEMA:</i>		
	<ul style="list-style-type: none"> * El alumno descubrirá los modelos del mundo real que se pueden representar por este modelo y su descripción. * El alumno aprenderá a implementar las aplicaciones que se representan de esta manera. 	

	MÓDULO 4. AUTÓMATAS DE ESTADO FINITO.	19 HI
<i>OBJETIVO DEL MÓDULO:</i> Obtener las bases necesarias para el diseño de analizadores lexicográficos y comprender la gran variedad de aplicaciones que se pueden representar y simular con este modelo.		
4.1	Concepto y representación del modelo.	2 HI
<i>OBJETIVOS DEL TEMA:</i>		
	<ul style="list-style-type: none"> * El alumno conocerá los criterios que hacen que una Máquina de Estado Finito puede ser transformada en un Autómata Finito. * El alumno conocerá la representación de un Autómata Finito por medio de una tupla de conjuntos. * El alumno dominará las representaciones alternas para este modelo, incluyendo la tabla de transición y el diagrama de transición. 	

	* El alumno será competente para transformar un Autómata de este tipo de una forma de representación a otra.	
4.2	Desarrollo de un Autómata Finito. <i>OBJETIVOS DEL TEMA:</i>	3 HRS
	* El alumno tendrá la competencia para desarrollar Autómatas Finitos en términos generales. * El alumno conocerá el concepto de Autómatas equivalentes.	
4.3	Autómatas de expresiones Regulares. <i>OBJETIVOS DEL TEMA:</i>	3 HRS
	* El alumno tendrá la capacidad de diseñar una gran diversidad de Autómatas Finitos en base a una expresión regular determinada.	
4.4	Relación entre Autómatas Finitos y Gramáticas Regulares. <i>OBJETIVOS DEL TEMA:</i>	3 HRS
	* El alumno confirmará sus conocimientos sobre la relación que existe entre los Lenguajes Regulares y sus reconocedores, que son los Autómatas Finitos. * El alumno diseñará Autómatas Finitos de manera directa a partir de una Gramática Regular determinada. * El alumno encontrará de manera directa la Gramática Regular que produce las cadenas aceptadas por un Autómata Finito determinado.	
4.5	Autómatas Finitos Deterministas (AFD) y Autómatas Finitos No Deterministas (AFND). <i>OBJETIVOS DEL TEMA:</i>	3 HRS
	* El alumno comprenderá las diferencias entre los dos tipos de Autómatas Finitos. * El alumno conocerá la importancia y las ventajas de cada uno de ellos. * El alumno será capaz de convertir un Autómata Finito No Determinístico en otro Determinístico equivalente.	
4.6	Desarrollo de una aplicación de Autómatas de Estado Finito. <i>OBJETIVOS DEL TEMA:</i>	3 HRS
	* El alumno tendrá la capacidad de representar diversos modelos del mundo real por medio de Autómatas Finitos. * El alumno podrá implementar los modelos del mundo real representados por este medio.	
4.7	Limitaciones de los Autómatas de Estado Finito. <i>OBJETIVOS DEL TEMA:</i>	1 HRS
	* El alumno comprenderá que existen algunos casos de Lenguajes Formales que no pueden ser aceptados por Autómatas Finitos al no ser regulares.	
4.8	Relación y diferencias de los Autómatas de Estado Finito con las Máquinas de Estado Finito. <i>OBJETIVOS DEL TEMA:</i>	1 HRS
	* El alumno comparará los dos modelos y verá las diferencias significativas que determinan cuál modelo es más conveniente para un caso de aplicación determinado.	

MÓDULO 5. AUTÓMATAS DE PILA.		7 HRS
<i>OBJETIVO DEL MÓDULO:</i> Obtener las bases del diseño de estos dispositivos para aplicarse como analizadores sintácticos en un compilador.		
5.1	Concepto y representación de Autómatas de Pila.	2 HRS
<i>OBJETIVOS DEL TEMA:</i>		
	<ul style="list-style-type: none"> * <i>El alumno conocerá la representación de un Autómata de Pila por medio de una tupla de conjuntos.</i> * <i>El alumno repasará los conceptos básicos de una pila de memoria, resaltando la inserción y extracción de símbolos.</i> * <i>El alumno comprenderá los elementos que se emplean en este nuevo modelo formal.</i> 	
5.2	Relación entre Autómatas de Pila y Gramáticas Libres de Contexto.	3 HRS
<i>OBJETIVOS DEL TEMA:</i>		
	<ul style="list-style-type: none"> * <i>El alumno tendrá la habilidad de diseñar Autómatas de Pila en base a un Lenguaje Libre de Contexto determinado.</i> * <i>El alumno tendrá la habilidad de diseñar Autómatas de Pila en base a una Gramática Libre de Contexto determinada.</i> 	
5.3	Implementación de un Autómata de Pila.	1 HRS
<i>OBJETIVOS DEL TEMA:</i>		
	<ul style="list-style-type: none"> * <i>El alumno podrá implementar Autómatas de Pila empleados en aplicaciones computacionales.</i> 	
5.4	Limitaciones de los Autómatas de Pila.	1 HRS
<i>OBJETIVOS DEL TEMA:</i>		
	<ul style="list-style-type: none"> * <i>El alumno comprenderá que existen algunos Lenguajes que no son aceptados por Autómatas de Pila al no ser Libres de Contexto.</i> 	

MÓDULO 6. MÁQUINAS DE TURING.		10 HRS
<i>OBJETIVO DEL MÓDULO:</i> Conocer el poder computacional de estas máquinas en el contexto de la solución de problemas de reconocimiento de lenguajes.		
6.1	Concepto y representación de Máquinas de Turing.	3 HRS
<i>OBJETIVOS DEL TEMA:</i>		
	<ul style="list-style-type: none"> * <i>El alumno conocerá el concepto de Máquina de Turing como reconocedor universal de Lenguajes Formales.</i> * <i>El alumno será capaz de representar formalmente una Máquina de Turing como una tupla.</i> 	
6.2	Máquinas de Turing como aceptadores de Lenguajes.	2.5 HRS
<i>OBJETIVOS DEL TEMA:</i>		
	<ul style="list-style-type: none"> * <i>El alumno tendrá la habilidad de relacionar este modelo con los Lenguajes Formales que puede reconocer.</i> 	
6.3	Construcción de Máquinas de Turing.	3 HRS
<i>OBJETIVOS DEL TEMA:</i>		
	<ul style="list-style-type: none"> * <i>El alumno tendrá la capacidad de diseñar Máquinas de Turing en base a especificaciones concretas y su posterior implementación.</i> 	

6.4	El problema de la parada.	1.5 HRS
	<i>OBJETIVOS DEL TEMA:</i>	
	* <i>El alumno conocerá el problema de la parada y su implicación en las ciencias computacionales.</i>	
MÓDULO 7. COMPUTABILIDAD.		
9 HI		
<i>OBJETIVO DEL MÓDULO:</i> Entender que el diseño de algoritmos presenta limitaciones en ciertos casos, que impiden su representación adecuada.		
7.1	Complejidad de los cálculos.	1.5 HI
	<i>OBJETIVOS DEL TEMA:</i>	
	* <i>El alumno analizará la complejidad de los cálculos.</i>	
7.2	Complejidad de los algoritmos.	1.5 HI
	<i>OBJETIVOS DEL TEMA:</i>	
	* <i>El alumno analizará la complejidad de los algoritmos.</i>	
7.3	Complejidad de los problemas.	1.5 HI
	<i>OBJETIVOS DEL TEMA:</i>	
	* <i>El alumno analizará la complejidad de los problemas.</i>	
7.4	Problemas NP.	2 HI
	<i>OBJETIVOS DEL TEMA:</i>	
	* <i>El alumno comprenderá la importancia de los problemas NP en las ciencias computacionales.</i>	
7.5	Problemas irresolubles.	2.5 HI
	<i>OBJETIVOS DEL TEMA:</i>	
	* <i>El alumno conocerá sobre la existencia de algoritmos que no pueden ser programados al no existir una adecuada representación del modelo.</i>	

METODOLOGÍA DE ENSEÑANZA APRENDIZAJE								
Método	Método tradicional de exposición	Método Audiovisual	Aula Interactiva	Multimedia	Desarrollo de proyecto	Dinámicas	Estudio de casos	Otros (Especificar)
%	10	0	0	0	15	65	10	0

CRITERIOS DE EVALUACIÓN	
Examen Parcial no Departamental.....	20%
Examen Final Departamental.....	30%
Tareas individuales (resolución de ejercicios) y trabajos especiales.....	30%
Implementación de aplicaciones.....	20%

BIBLIOGRAFÍA

BÁSICA

TITULO	AUTOR	EDITORIAL	AÑO DE EDICIÓN	% DE COBERTURA DEL CURSO
Introducción a la Teoría de los Autómatas y Lenguajes Formales	Abelardo Gómez Andrade et al.	Servicios Editoriales Tranco	2012	100
Lenguajes Formales y Teoría de la Computación	John Martin	Mc Graw Hill	2004 3ra. Ed	80

COMPLEMENTARIA

TITULO	AUTOR	EDITORIAL	AÑO DE EDICIÓN	% DE COBERTURA DEL CURSO
Introducción a la Teoría de Autómatas, Lenguajes y Computación	John E. Hopcroft y Jeffrey D. Ullman	Addison Wesley Iberoamericana	2002	75
Teoría de la Computación	J. Glenn Brookshear	Addison Wesley Iberoamericana	1995	75
Teoría de Autómatas y Lenguajes Formales	Dean Kelley	Prentice Hall	1995	80

REVISIÓN REALIZADA POR:

NOMBRE DEL PROFESOR		FIRMA
GÓMEZ ANDRADE ABELARDO		*Consta en el original
MURILLO LEAÑO MARÍA MAGDALENA		*Consta en el original
VÁZQUEZ ÁVILA JORGE		*Consta en el original

Vo.Bo. Presidente de Academia

*Consta en el original

MTRO. MIGUEL ÁNGEL GUERRERO SEGURA R.

Vo.Bo. Jefe del Departamento

*Consta en el original

ING. PATRICIA MENDOZA SÁNCHEZ

Para obtener el archivo de este mismo programa y considerar posibles actualizaciones futuras, consultar en la dirección <http://dcc.cucei.udg.mx>.

REACTIVOS DE TEORÍA DE LA COMPUTACIÓN

01.- Dado el Lenguaje $L = \{ \lambda, cb, abbc, babac \}$ determinar:

- a) Los posibles prefijos, infijos y posfijos de cada cadena.
- b) El Alfabeto Σ .
- c) La longitud de cada cadena de L .
- d) El contenido del conjunto L^2 .
- e) ¿ L es un Lenguaje finito o infinito?

02.- Dados los Lenguajes $L_1 = \{ a, aa, ab, aba \}$ y $L_2 = \{ \lambda, b, ab, bab, bbb \}$ evaluar:

- a) $L_1 \cup L_2$
- b) $L_1 \cap L_2$
- c) $L_1 \bullet L_2$
- d) $L_2 - L_1$
- e) $L_1 - L_2$
- f) $L_2 \cup L_1 \bullet L_2$

03.- Dado el Lenguaje $L = \{ \lambda, a, ab, c \}$, determinar el contenido de los conjuntos Σ , L^0 , L^1 , L^2 , L^+ y L^* .

04.- Considérese el Lenguaje L^* , surgido a partir de $L = \{a, b\}$ con un Alfabeto $\Sigma = \{a, b\}$.

- a) ¿Cuántas palabras de ese L^* tienen longitud de 2, de 3, de 4 y de n , respectivamente?
- b) Evaluar lo mismo que en el inciso anterior, pero ahora con $L = \{a, ab, bc\}$ si $\Sigma = \{a, b, c\}$.

05.- Considérese al Lenguaje $L = \{ a, ab, bb \}$. Las cadenas aabab, babbbab, abbabab, abbaababb, ¿son palabras de un L^n ?

06.- Dado el Lenguaje $L = \{ aa, aba, baa \}$, responder:

- a) Las cadenas $a^2ba^2, (ba^3)^2, (ba^2)^3b^2a, ba^5(ba)^2a^3$ ¿son palabras del Lenguaje L^+ ?

b) ¿Qué características tendrían las cadenas de este L^+ , expresadas en un Lenguaje natural?

07.- Describir en un Lenguaje natural las cadenas que contiene cada uno de los siguientes Lenguajes Formales:

- a) $L = \{ 1^*0 \}$
- b) $L = \{ 1^*00^* \}$
- c) $L = \{ 1^20^* \} \cup \{ 0^21^* \}$
- d) $L = \{ (1, 00)^* \}$
- e) $L = \{ (0^+1)^* \}$
- f) $L = \{ (0,1)^1 \bullet (0,1)^* 00 \}$

08.- Determinar en cuáles de los siguientes Lenguajes está contenida la cadena 1011.

- a) $L = \{ 10^*1^* \}$
- b) $L = \{ 0^* (11,10)^* \}$
- c) $L = \{ (10)^* 101 \}$
- d) $L = \{ 1^* 01 (0,1)^1 \}$
- e) $L = \{ (11)^* (10)^* \}$
- f) $L = \{ (11)^*, (10)^* \}$
- g) $L = \{ 1^+ (01)^* 1^* \}$
- h) $L = \{ (1,00) \bullet (01,0) \bullet 1^* \}$
- i) $L = \{ 10, 11 \}^3$
- j) $L = \{ 0, 1, 10 \}^n$

09.- Considerando a $\Sigma = \{ a, b, c \}$, ¿son equivalentes las parejas de expresiones formales de cada inciso?

- a) $((a \cup b)c)^* \text{ y } (ac \cup bc)^*$.
- b) $b(ab \cup ac) \text{ y } (ba \cup ba)(b \cup c)$.
- c) $(a, b)^* \text{ y } (a^*, b^*)$.
- d) $((a \cup b)a)^* \text{ y } (a \cup b)^* a^*$.

10.- Especificar la expresión formal que corresponda a cada uno de los siguientes Lenguajes sobre $\Sigma = \{ a, b \}$:

- a) El Lenguaje de las cadenas que inician con a^2 ***o*** que finalizan con b^2 .
- b) El Lenguaje de las cadenas que inician con a^2 ***y*** que finalizan con b^2 .

11.- Sea $G = \{(A, B, C), (a, b, c), (A \rightarrow aaA, A \rightarrow bB, B \rightarrow cCb, B \rightarrow cb, C \rightarrow bbC, C \rightarrow bb), (A)\}$.

Establecer cuáles de las siguientes cadenas pertenecen a L(G).

- a) aabcb.
- b) abcbc.
- c) bcbbbb.
- d) aabccbbb.
- e) aaaabcbc.
- f) aaaabcbbbb.

12.- Dada G = ($\{a, b, c\}$, $\{+, -\}$, $\{a \rightarrow a+, a \rightarrow -c-, b \rightarrow +, b \rightarrow +, b \rightarrow a+c, c \rightarrow cc, a \rightarrow +\}$, $\{a\}$).

- a) ¿Qué observaciones se le pueden hacer a G?
- b) Caracterizar la Gramática anterior.
- c) Encontrar 6 cadenas del Lenguaje L(G).

13.- Caracterizar la Gramática con composiciones $P = \{S \rightarrow AB, A \rightarrow aA, A \rightarrow a, B \rightarrow Bb, B \rightarrow b\}$ y $S = \{S\}$.

14.- Sean $N = \{S, A, B\}$, $T = \{a, b\}$, $S = \{S\}$. Para cada inciso caracterizar el Lenguaje generado por la Gramática señalada y mencionar si existe algún error en las especificaciones y cómo se puede corregir.

- a) $P = \{S \rightarrow A \mid B, A \rightarrow abA \mid c, B \rightarrow ccB \mid ab\}$.
- b) $P = \{S \rightarrow AA \mid B, A \rightarrow aaA \mid aa, B \rightarrow bB \mid b\}$.
- c) $P = \{S \rightarrow AB \mid AA, A \rightarrow aB \mid ab, B \rightarrow b\}$.
- d) $P = \{S \rightarrow AS \mid aA, A \rightarrow a, B \rightarrow ba\}$.

15.- Caracterizar la Gramática con $P = \{A \rightarrow xAz, A \rightarrow yAz, A \rightarrow \lambda\}$ y $S = \{A\}$. ¿Es la anterior una Gramática decidable?

16.- Diseñar la Gramática que produce cada uno de los L(G) enunciados.

- a) $L(G) = \{0^n1^n \mid n \in \mathbb{N}\}$.
- b) $L(G) = \{0^n1^n \mid n \in \mathbb{N}_0\}$.
- c) $L(G) = \{0^m1^n \mid m, n \in \mathbb{N}\}$.

17.- Diseñar la Gramática Formal que produce el Lenguaje $L = \{(ab)^*c^2\}$.

18.- Diseñar para cada inciso una Gramática que genere todas las cadenas de bits con las siguientes características:

- a) Con igual cantidad de 0 que de 1.
- b) Con más 0 que 1.
- c) Con desigual cantidad de 0 que de 1.

19.- Diseñar una Gramática Regular que produzca el Lenguaje $L = \{01, 10\}^*$ con $T = \{0, 1\}$.

20.- Diseñar la Gramática Regular que produce el Lenguaje $L = \{a^*b\} \cup \{a\}$ si el Alfabeto es $\Sigma = \{a, b\}$.

21.- Demostrar que $L(G) = \{0^n1^n \mid n \in \mathbb{N}\}$ no es Lenguaje Regular.

22.- Expresar al menos 5 de las reglas de producción más comunes de Lenguaje C en BNF y otras 4 reglas de Java o Pascal en Diagramas de Sintaxis.

23.- Diseñar las composiciones en BNF para una Gramática que genere todos los números reales o flotantes válidos, en este caso sin tomar en cuenta el tipo de Gramática a emplear.

24.- Diseñar una Gramática Regular en BNF para cada inciso, que produzca los siguientes tokens:

- a) Identificadores válidos en Lenguaje C.
- b) Números enteros decimales sin signo.
- c) Números enteros decimales (con o sin signo).
- d) Números reales decimales (con o sin signo).

- e) Números enteros hexadecimales sin signo en ensamblador para microprocesadores de Intel.

25.- Dada la Gramática con las composiciones del conjunto

$$P = \{ \begin{array}{l} \langle \text{número} \rangle ::= \langle \text{entero} \rangle \mid \langle \text{real} \rangle, \\ \langle \text{entero} \rangle ::= \langle \text{dígito} \rangle \mid \langle \text{entero} \rangle \langle \text{dígito} \rangle, \\ \langle \text{real} \rangle ::= \langle \text{entero} \rangle . \langle \text{entero} \rangle \mid \\ \quad \langle \text{entero} \rangle . \langle \text{entero} \rangle E \langle \text{exp} \rangle \mid \langle \text{entero} \rangle E \\ \quad \langle \text{exp} \rangle, \\ \langle \text{exp} \rangle ::= \langle \text{entero} \rangle \mid \langle \text{signo} \rangle \langle \text{entero} \rangle, \\ \langle \text{signo} \rangle ::= + \mid -, \\ \langle \text{dígito} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \end{array} \}$$

en donde $\langle \text{número} \rangle$ es el símbolo inicial; especificar qué inconvenientes tiene la Gramática, respecto a la producción de números válidos según los criterios empleados en el desarrollo de compiladores actuales.

26.- Empleando la misma Gramática del inciso anterior evaluar tanto la derivación por la izquierda como por la derecha, para las cadenas 0.27 y 6E-3.

27.- Demostrar con un árbol de derivación, si esto fuera posible, que la cadena $a^2b^2a^2$ es una cadena del Lenguaje producido por la Gramática con $P = \{ S \rightarrow aAS \mid a, A \rightarrow SS \mid SbA \mid ba \}$, con símbolo inicial S .

28.- Dada la Gramática con $G = (\{S\}, \{a, b, c\}, \{S \rightarrow abS, S \rightarrow bcS, S \rightarrow bbS, S \rightarrow a, S \rightarrow cb\}, \{S\})$, construir los árboles de derivación para cada una de las cadenas:

- | | |
|-----------------|------------------|
| a) b^3cb^2a | d) ab^2c^2ab |
| b) ab^2ac^2ba | e) bcb^5ca |
| c) $bcab^5cb$ | f) $(ab)^3b^4cb$ |

29.- Desarrollar una oración o sentencia válida de un Lenguaje natural, como el castellano, por medio de un árbol de derivación y utilizando las reglas de producción adecuadas, aprendidas durante la primaria.

30.- Diseñar el árbol de derivación para la expresión tomada del código de un programa $(a + b)^* c$. ¿Cuáles serían su árbol de análisis sintáctico y su árbol sintáctico?

Las composiciones a emplear deben ser deducidas por el alumno en base a sus conocimientos sobre programación.

31.- Diseñar un árbol de sintaxis y otro de análisis sintáctico que correspondan a la siguiente cadena tomada de un programa en Lenguaje C: $\mathbf{discrim} = b^* b - 4 * a * c$. Definir las reglas usadas.

32.- Determinar si G es ambigua o unívoca: $G = (\{E\}, \{(,), +, *\}, \{E \rightarrow E + E \mid E * E \mid (E) \mid id\}, \{E\})$. En este caso, como la variedad de identificadores válidos es muy grande, se considera en la teoría de compiladores, para generalizarlos, como si id fuera un terminal definido.

33.- Dada la Gramática con $P = \{ A \rightarrow 0B B, B \rightarrow 1A \mid 0A \mid 0 \}$ con A como símbolo inicial, determinar si es ambigua o unívoca y justifíquese la respuesta.

34.- Demostrar que la Gramática con $P = \{ \langle \text{entero} \rangle ::= \langle \text{dígito} \rangle \mid \langle \text{entero} \rangle \langle \text{signo} \rangle \langle \text{entero} \rangle, \langle \text{dígito} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \}$ es ambigua y encontrar otra equivalente y no ambigua.

35.- Determinar si son ambiguas las siguientes Gramáticas:

- a) $P = \{ S \rightarrow + S S \mid - S S \mid a \}$.
- b) $P = \{ S \rightarrow (S) S \mid \epsilon \}$. // ϵ es cadena vacía
- c) $P = \{ S \rightarrow S \wedge S \mid a \}$.

36.- Diseñar las reglas de producción para un compilador que evalúe sin que exista ambigüedad la exponenciación múltiple ($a^b^c^d \dots$) y considerando la asociatividad.

37.- Diseñar una Gramática Libre de Contexto en la cual se jerarquicen las cuatro operaciones aritméticas básicas, además de incluir los paréntesis y considerando también su asociatividad. Llámese <expresión> al símbolo inicial.

38.- Convertir la Gramática Libre de Contexto a su Forma Normal de Chomsky (CNF):

$$P = \{ A \rightarrow xy \mid Bx \mid xEx, \quad D \rightarrow xx \mid yB \mid xE, \\ B \rightarrow zC \mid DE \mid E, \quad E \rightarrow EAx \mid yCy, \\ C \rightarrow DC \mid xyF \mid FA \mid y, F \rightarrow AC \mid B \}.$$

39.- Convertir a la Forma Normal de Chomsky (CNF) equivalente la siguiente Gramática Libre de Contexto:

$$N = \{ A, B, C, D, E, F \} \quad T = \{ x, y \} \quad S = \{ A \}$$

$$P = \{ A \rightarrow xy, \quad B \rightarrow DEF, \quad C \rightarrow y, \quad E \rightarrow FA, \\ A \rightarrow Bx, \quad C \rightarrow DC, \quad D \rightarrow xx, \quad E \rightarrow yCy, \\ A \rightarrow xEx, \quad C \rightarrow xyF, \quad D \rightarrow yB, \quad F \rightarrow AC, \\ B \rightarrow zC, \quad C \rightarrow FA, \quad D \rightarrow xE, \quad F \rightarrow B \}.$$

40.- (*Tarea Adicional de la primera parte*).- Redactar una investigación de al menos dos cuartillas para cada caso sobre las aplicaciones de las Gramáticas Formales en:

a) Diseño de las etapas de análisis en un compilador para un Lenguaje de Programación, como C, Java o Pascal.

b) Procesamiento de Lenguajes Naturales en el área de la Inteligencia Artificial.

c) Expresiones Regulares en algoritmos de búsqueda de cadenas en Lenguajes de programación como Perl.

FIN DE LA PRIMERA PARTE DEL CURSO
Aplicar examen departamental parcial

41.- Dibujar el diagrama de transición de la Máquina de Estado Finito que corresponde a cada inciso, además de expresar los seis conjuntos que forman la representación de la tupla para M.

a)

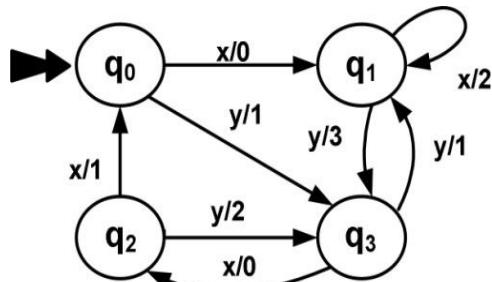
		f			g			
		a	b	c	a	b	c	
S	I	q ₀	q ₁	q ₀	q ₀	z	y	y
→		q ₁	q ₀	q ₁	q ₂	y	x	w
		q ₂	q ₂	q ₁	q ₁	x	z	x

b)

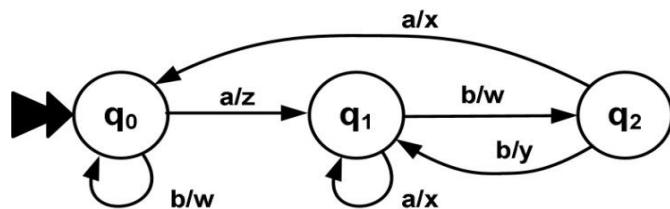
		f			g			
		x	y	z	x	y	z	
S	I	q ₀	q ₁	q ₃	q ₂	a	b	a
→		q ₁	q ₀	q ₂	q ₁	d	a	c
		q ₂	q ₂	q ₀	q ₃	c	b	b
		q ₃	q ₂	q ₃	q ₁	d	c	b

42.- Determinar cuáles son los conjuntos que determinan la descripción formal para las siguientes Máquinas de Estado Finito, además de las tablas de transición para cada inciso.

a)



b)



43.- Establecer las transiciones (estados y cadenas de salida) que resultan al introducir las siguientes cadenas de entrada, referidas a las Máquinas de los problemas 41 y 42.

- a) babccab en 41a)
- b) zxyzxxyz en 41b)

- c) xyxyyxyxxy en 42a)
- d) bbabbbaaba en 42b).

44.- Diseñar una Máquina de Estado Finito que funcione como restador binario en serie y otra que se comporte como un multiplicador binario en serie, para operar con cadenas de bits de la misma longitud. Justificar las respuestas.

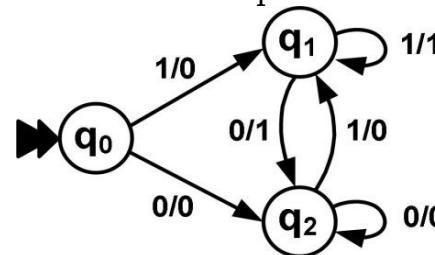
45.- Diseñar una Máquina de Estado Finito, que también sea Autómata de Estado Finito, con las siguientes características:

$$I = \{a, b, c\}, \quad S = \{A, B, C, D\}, \quad \sigma_0 = \{A\}.$$

46.- Dado el siguiente diagrama de transición determinar:

- a) ¿Se puede transformar esta Máquina en un Autómata Finito?

- b) ¿Qué sistema real representa?



47.- Diseñar un Autómata Finito con $I = \{a, b\}$ que acepte cadenas con una cantidad impar de letras a .

48.- Diseñar un Autómata de Estado Finito que acepte las cadenas de bits con una cantidad de 0 igual a un múltiplo de 2 ó de 3.

49.- Diseñar un Autómata Finito que acepte cadenas no nulas formadas con los símbolos del conjunto $I = \{a, b, c\}$ y que no contengan ni a ni c .

50.- Diseñar un Autómata Finito con $I = \{a, b\}$ que acepte las palabras en las que siempre que aparezca una b , esté seguida necesariamente por una a .

51.- Diseñar un Autómata de Estado Finito que acepte las cadenas de bits que contienen el infijo 10.

52.- Diseñar un Autómata de Estado Finito que acepte las cadenas de bits que contienen el infijo 101.

53.- Diseñar un Autómata Finito con $I = \{ 0, 1 \}$ que acepte los arreglos que inician con 011.

54.- Diseñar un Autómata Finito que acepte todas las cadenas de bits que finalicen con 00 y mostrar el resultado por medio de las tres representaciones posibles para el modelo formal.

55.- Diseñar el Autómata Finito que acepta cada uno de los siguientes Lenguajes, considerando $I = \{ a, b, c \}$

- | | |
|--------------------|----------------------------------|
| a) \emptyset | d) $\{ (a, b, c)^* \}$ |
| b) $\{ \lambda \}$ | e) $\{ a^*b^+c^* \}$ |
| c) $\{ a, b, c \}$ | f) $\{ bca^* \} \cup \{ ac^* \}$ |

56.- Dado el conjunto $I = \{ a, b \}$ diseñar un diagrama de transición del Autómata Finito para cada uno de los Lenguajes expresados en los siguientes incisos:

- $L = \{ (ab)^+ \} \cup \{ a \}$.
- $L = \{ ab^* \} \cup \{ ab^*a \}$.
- $L = \{ a (a,b)^* \} \cup \{ b (a,b)^* \}$

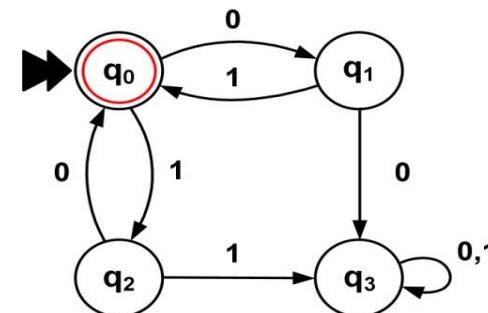
57.- Diseñar un Autómata Finito en cada inciso de acuerdo a las entradas y las cadenas aceptadas que se indican.

- $I = \{ +, * \}$. Cadenas que contienen ** ó $*^{++}$.
- $I = \{ w, z \}$. Cadenas que terminen en wz o z^2w .

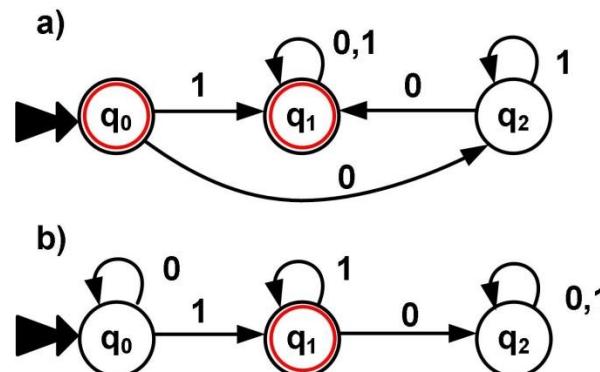
58.- Demostrar que no es posible construir un Autómata de Estado Finito que al recibir como entrada cualquier cadena de bits, acepte la cadena que contiene exactamente la misma cantidad de 0 que de 1.

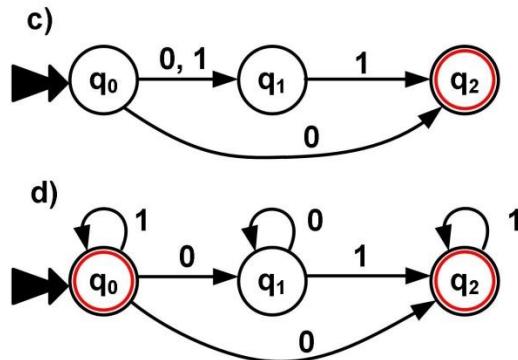
59.- Comparar entre el diseño de una máquina para venta de refrescos implementada con los modelos de Autómata y de Máquina de Estado Finito para descubrir la diferencia entre las dos herramientas matemáticas. Para fines didácticos considérese el caso de un refresco con un coste de \$10 y empleando monedas de 1\$, \$2, \$5 y \$10 como posibles entradas.

60.- ¿Qué cadenas acepta el siguiente Autómata Finito?

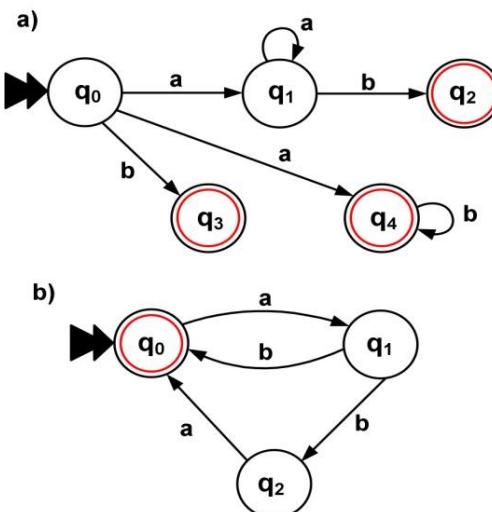


61.- Determinar el Lenguaje aceptado por cada uno de los siguientes Autómatas Finitos, observando que dos de ellos son No Determinísticos.

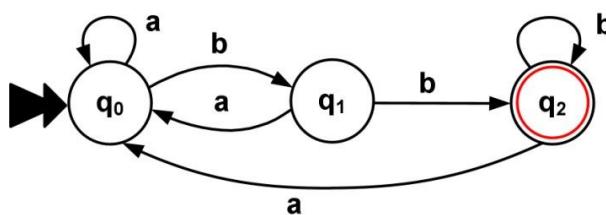




62.- Caracterizar el Lenguaje aceptado por los siguientes Autómatas Finitos No Determinísticos:



63.- ¿Qué cadenas acepta el siguiente AFD?



64.- Demostrar que el Lenguaje de todas las cadenas de unos y ceros que tienen al menos dos ceros consecutivos es un Lenguaje Regular.

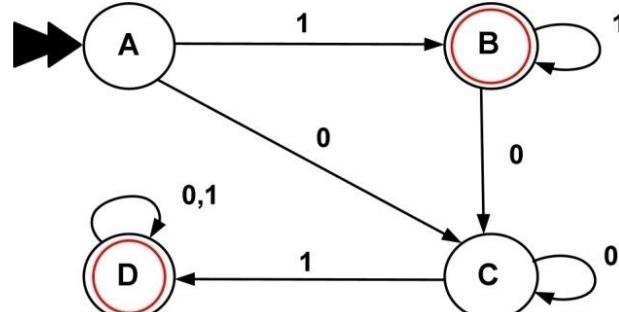
65.- Diseñar una Gramática Regular con $T = \{ a, b \}$ que genere todos los arreglos que finalicen con ba. El diseño se deberá hacer a partir de un Autómata Finito.

66.- Diseñar por medio de un Autómata Finito una Gramática Regular con $T = \{ a, b \}$ que genere todas las cadenas que contengan el infijo ba.

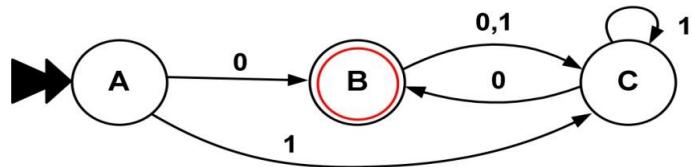
67.- Diseñar por medio de un Autómata Finito la Gramática Regular que produce todas las cadenas que representan los números enteros hexadecimales sin signo en la notación de ensamblador para microprocesadores de Intel.

68.- Construir una Gramática Regular que genere el Lenguaje reconocido por el Autómata Finito determinado.

a)



b)



69.- Diseñar el AFD que acepte las cadenas producidas por la Gramática con las composiciones $P = \{ A \rightarrow xB \mid yA, B \rightarrow yC \mid x, C \rightarrow xB \mid y \}$ si $\sigma_0 = \{ A \}$.

70.- Diseñar un AFD que acepte las cadenas del Lenguaje $L(G)$ para la Gramática con reglas $P = \{ A \rightarrow mA \mid nB, B \rightarrow nC \mid m, C \rightarrow mC \mid n \}$ con A como inicial. Caracterizar el Lenguaje producido.

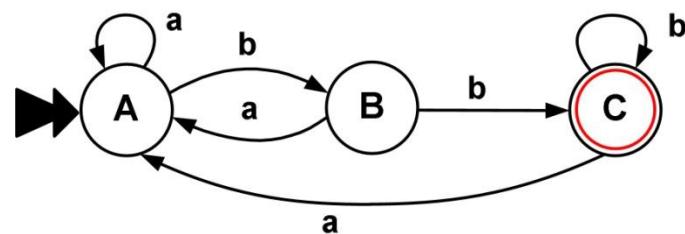
71.- Diseñar un AFD que reconozca las cadenas del Lenguaje generado por la Gramática $G = (\{ A, B \}, \{ 0, 1 \}, \{ A \rightarrow 0A \mid 0B \mid 1, B \rightarrow 1B \mid 0 \}, \{ A \})$. Encontrar la caracterización del Lenguaje $L(G)$.

72.- Diseñar un AFD que acepte las cadenas del Lenguaje Formal $L(G)$ para la Gramática con reglas $P = \{ A \rightarrow mB \mid nC \mid m, B \rightarrow mA \mid n, C \rightarrow nB \mid nC \mid m \mid n \}$. El símbolo inicial es A .

73.- Diseñar un AFD que acepte las cadenas del Lenguaje $L(G)$ para la Gramática con reglas $P = \{ A \rightarrow aA \mid uB \mid a, B \rightarrow oA \mid a \mid o \mid u \}$ con $\sigma_0 = \{ A \}$.

74.- Diseñar un AFD que acepte las cadenas del Lenguaje $L(G)$ para la Gramática con reglas $P = \{ X \rightarrow aX \mid bY \mid b, Y \rightarrow cY \mid a \mid c \}$ con X como símbolo inicial. Caracterizar el Lenguaje $L(G)$.

75.- Sea L el conjunto de arreglos aceptados por el AFD mostrado al final del enunciado. Constrúyase un AFD que acepte los arreglos $L^R = \{ x_n \dots x_2 x_1 \mid x_1 x_2 \dots x_n \in L \}$ además de caracterizar L y L^R . Se inicia el diseño en el que era el estado de aceptación original y se traza el recorrido en sentido opuesto hasta llegar al que se tenía como el estado inicial. Finalmente se convierte el AFND en un AFD.



76.- Diseñar para cada inciso, el diagrama de transición del Autómata Finito que acepta las palabras, resultado de las operaciones indicadas en los correspondientes Lenguajes.

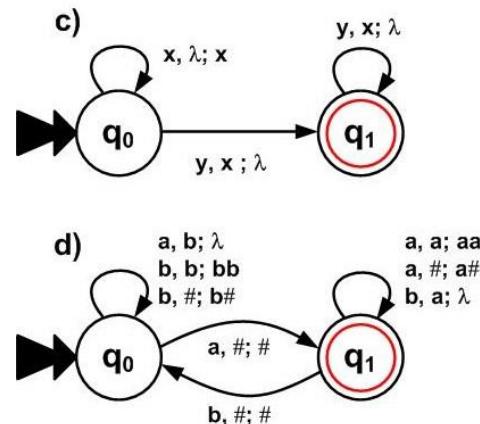
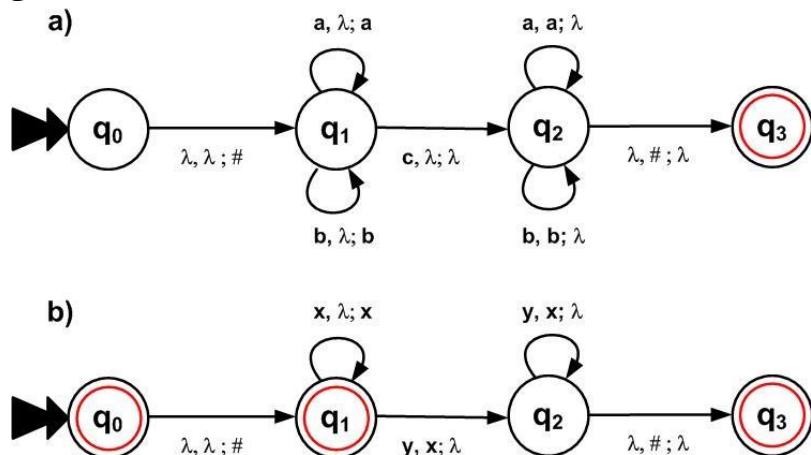
- a) \cup
- b) \bullet
- c)

77.- Diseñar un Autómata Finito con transiciones λ en el que con $I = \{a, b\}$ se acepte el Lenguaje $L = \{ab^*\} \cup \{(ab)^*\}$. ¿Cómo sería la versión del Autómata Finito que acepta ese mismo Lenguaje pero sin transiciones λ ?

78.- Diseñar para cada inciso el Autómata de Pila que acepta el Lenguaje que se describe:

- $L = \{0^n 1^n \mid n \in \mathbb{N}\}$.
- $L = \{s \mid s \text{ sea cualquier expresión aritmética que contenga paréntesis correctamente anidados}\}$.
- $L = \{x^n y^n z^n \mid n \in \mathbb{N}\}$.
- $L = \{w^m x^n y^n z^m \mid m, n \in \mathbb{N}\}$.
- $L = \{w^m x^n y^m z^n \mid m, n \in \mathbb{N}_0\}$.
- $L = \{x^m y^m \mid m \in \mathbb{N}\} \cup \{x^n y^{2n} \mid n \in \mathbb{N}\}$.
- $L = \{(a, b)^* \mid \text{Cantidad de } a \geq \text{Cantidad de } b\}$.

79.- ¿Qué Lenguaje Formal acepta cada de los siguientes Autómatas de Pila?



80.- Sea la Gramática Libre de Contexto definida por las composiciones $P = \{A \rightarrow zBCx, B \rightarrow xAz, B \rightarrow zBw, B \rightarrow z, C \rightarrow Cy, C \rightarrow zy\}$ y símbolo inicial A. Diseñar el Autómata de Pila que acepta el Lenguaje $L(G)$.

81.- Sea la Gramática Libre de Contexto definida por las composiciones $P = \{A \rightarrow aBA \mid a, B \rightarrow AbB \mid AA \mid ba\}$ y símbolo inicial A. Diseñar el Autómata de Pila que acepta el Lenguaje $L(G)$.

Problemas propuestos por el Mtro. Abelardo Gómez Andrade.

Profesor de Tiempo Completo en el Centro Universitario de Ciencias Exactas e Ingenierías de la Universidad de Guadalajara.

e-mail: abelardo.gomez@red.cucei.udg.mx

Tómense como referencia para el curso Teoría de la Computación y los exámenes departamentales de la asignatura.