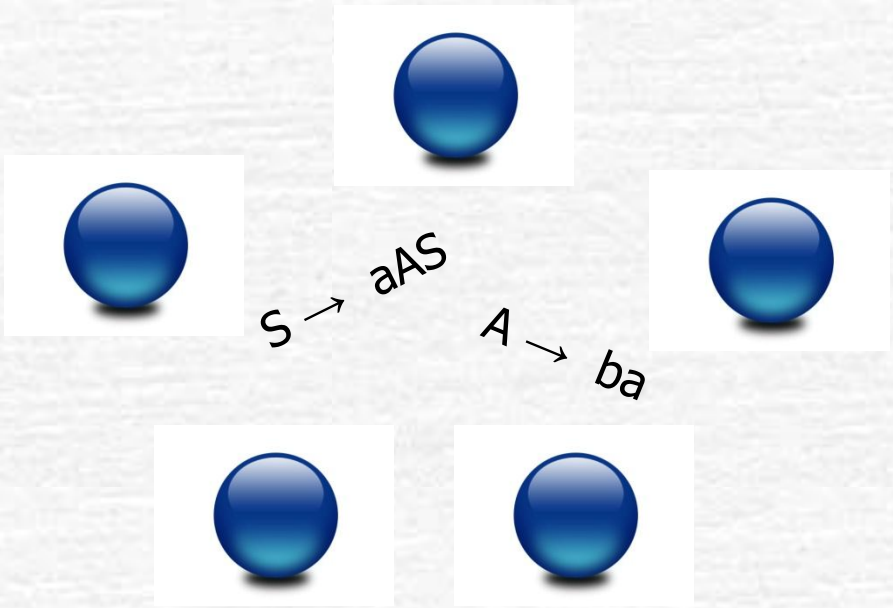
Notas para la Materia
Lenguajes de Programación

Gramáticas



Universidad de Sonora
Departamento de Matemáticas
Licenciatura en Ciencias de la Computación
Dra. María de Guadalupe Cota Ortiz

Lenguaje

- Un **lenguaje natural** es "un conjunto de palabras y métodos de combinaciones de palabras utilizado y comprendido por una comunidad de tamaño considerable"
- Los **lenguajes formales** se utilizan para modelar los lenguajes naturales y para implementarse en computadoras. Un lenguaje formal es un lenguaje con símbolos primitivos (**alfabeto**), y reglas (**producciones**) formalmente especificadas a través de una gramática formal que permite especificar las reglas sintácticas de los lenguajes formales o naturales; se compone de un conjunto de **fórmulas bien formadas (fbf).** Una fórmula bien formada es una cadena (**palabra**) formada por símbolos, y se constituyen de acuerdo a las reglas especificadas en la gramática correspondiente.

Definiciones

Alfabeto

- Es un conjunto finito de símbolos no vacío.

Cadena

- Se forma por un conjunto finito de símbolos que forman parte de un alfabeto.
- **Si A es un alfabeto, L es el lenguaje formado por las cadenas aceptadas sobre A (L(A)).**

Lenguajes Regulares

- Los lenguajes regulares como uno de los tipos de lenguajes formales, son útiles para especificar la construcción de analizadores léxicos y es generado por una Gramática Regular. Todo lo que es verdad para lenguajes regulares también lo es para lenguajes aceptados por un autómata finito y viceversa.
- Las expresiones regulares o patrones y los autómatas finitos, proporcionan los medios para especificar lenguajes.

Máquinas de Estado Finito (MEF)

Son modelos abstractos de máquinas con una memoria interna primitiva.

M = (I, O, S, f, g, σ)

I = Alfabeto de entrada

O = Alfabeto de salida

S = Conjunto de estados

σ = Símbolo inicial

f: S x I $\Rightarrow$ S (función de transición de estados)

g: S x I $\Rightarrow$ O (función de transición símbolos de salida)

Tabla de transición.- Es la base para el diagrama de transición.

Una MEF traduce un alfabeto de entrada en un alfabeto de salida, partiendo de un estado inicial y siguiendo las transiciones 'f' y 'g'.

Ejemplo de una MEF

En base al diseño de un sumador binario en serie, diseñar la Máquina de Estado Finito correspondiente:

M = (I, O, S, f, g, σ)

I = {00, 01, 10, 11} **#valores de entrada**

O = {0,1} **#valores resultantes de la suma binaria**

S = {C, NC} **#Estados de Acarreo (C) y Sin acarreo (NC)**

σ = {NC} **#Estado inicial**

Las especificaciones de las funciones de transición 'f' y 'g' se forman de acuerdo al comportamiento del sumador.

Ejemplo de una MEF

Función de transición 'f':

f(NC, 00) = NC f(C, 00) = NC
f(NC, 01) = NC f(C, 01) = C
f(NC, 10) = NC f(C, 10) = C
f(NC, 11) = C f(C, 11) = C

Interpretación:

f(estado_actual, valores_entrada) = nuevo_estado

Función de transición 'g':

g(NC, 00) = 0 g(C, 00) = 1
g(NC, 01) = 1 g(C, 00) = 0
g(NC, 10) = 1 g(C, 00) = 0
g(NC, 11) = 0 g(C, 00) = 1

Interpretación:

g(estado_actual, valores_entrada) = valor_salida

Op: Valor1 de entrada + Valor2 de entrada = valor_salida

Ej.: En estado No Carry, 1 + 1 = 0, y se lleva 1 de acarreo

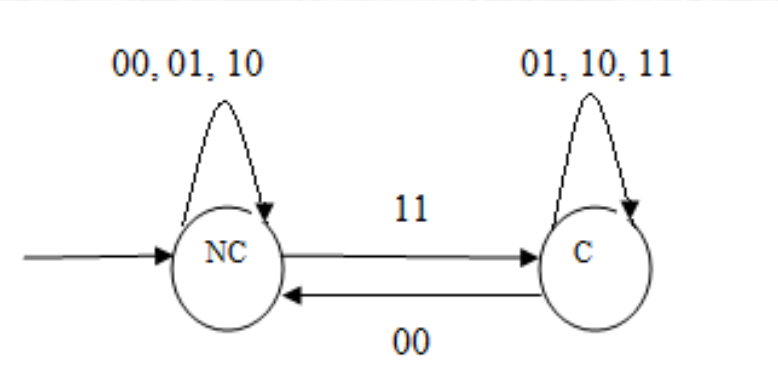
Ej: En etado Carry, 1 + 1 = 1, y se lleva 1 de acarreo

Ejemplo de una MEF

Tabla de Transición

Estados	f				g			
	00	01	10	11	00	01	10	11
C	NC	C	C	C	1	0	0	1
→ NC	NC	NC	NC	C	0	1	1	0

Diagrama de Transición



Pruebas del ejemplo MEF Sumador en Serie

Con estado_actual NC:

valores de entrada: 11 **Valor de salida:** 0 **nuevo_estado:** C

valores de entrada: 10 **Valor de salida:** 0 **nuevo_estado: NC**

Con estado_actual C:

valores de entrada: 11 **Valor de salida:** 1 **nuevo_estado: C**

valores de entrada: 00 **Valor de salida:** 1 **nuevo_estado: NC**

NOTA: En Lenguajes de programación una máquina de estado finito puede utilizarse en el proceso de traducción de un lenguaje a otro, ya que dada una cadena de entrada escrita en un lenguaje, se genera una cadena de salida escrito en otro lenguaje distinto.

Autómata de Estado Finito (AF)

Es un tipo particular de máquina de estado finito que está íntimamente ligado a un tipo particular de lenguaje.

Determinista. (AFD)- Permite crear solo una arista de un estado a otro, por cada símbolo que se evalúa.

No determinista. (AFND)- Permite crear varias aristas de un estado hacia otros estados por cada símbolo que se evalúa.

Nota: **Las diferencias entre una MEF y un AF son:**

En una AF no se maneja un alfabeto de salida, y por lo tanto la función de transición de salida ('g'), también se omite.

En un AF se agrega un elemento de Estados de Aceptación para identificar cadenas aceptadas o válidas.

Definición de un AF

A = (I, S, AC, f, σ)

I = Alfabeto de entrada

S = Conjunto de estados

AC = Conjunto de estados de aceptación

f = función de transición de estados

σ = Símbolo inicial

Para construir la gráfica correspondiente es necesario conocer las definiciones especificadas en la tabla de transición.

Ejemplo de un AF

Sea

$A = (I, S, AC, f, \sigma)$

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

$S = \{A,B,C\}$

$AC = \{B\}$

$\sigma = \{A\}$

y

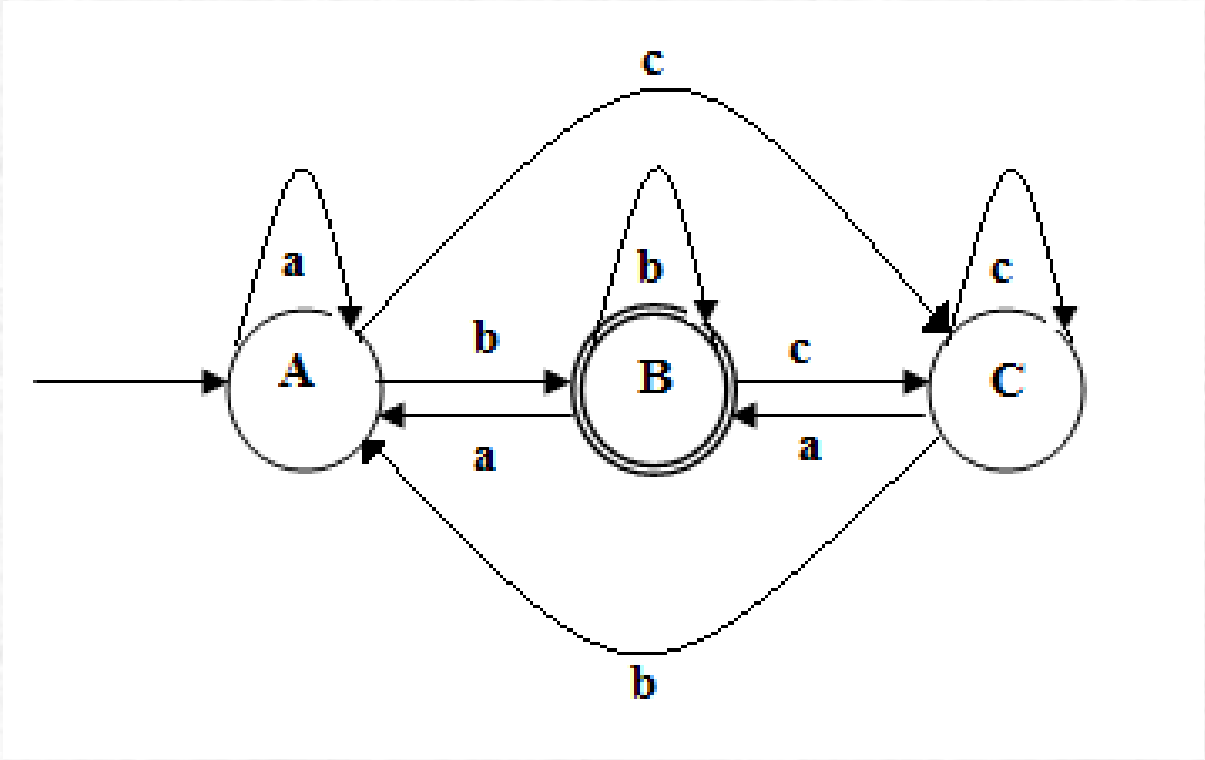
Función de transición 'f':

f(A,a) = A	f(B,a) = A	f(C, a) = B
f(A,b) = B	f(B,b) = B	f(C, b) = A
f(A,c) = C	f(B,c) = C	f(C,c) = C

Interpretación:

f(estado_actual, valores_entrada) = nuevo_estado

Diagrama de Transición



Pruebas del AF:

Cadena de entrada: **accbab (CADENA ACEPTADA)**

Cadena de entrada: **accbac (CADENA RECHAZADA)**

Máquina de Turing

Es un tipo particular de máquina de estado finito, pero con elementos adicionales.

MT = (I,O, S, f, g, σ, Movimientos)

I = Alfabeto de entrada

O = Alfabeto de salida

S = Conjunto de estados

f = función de transición de estados

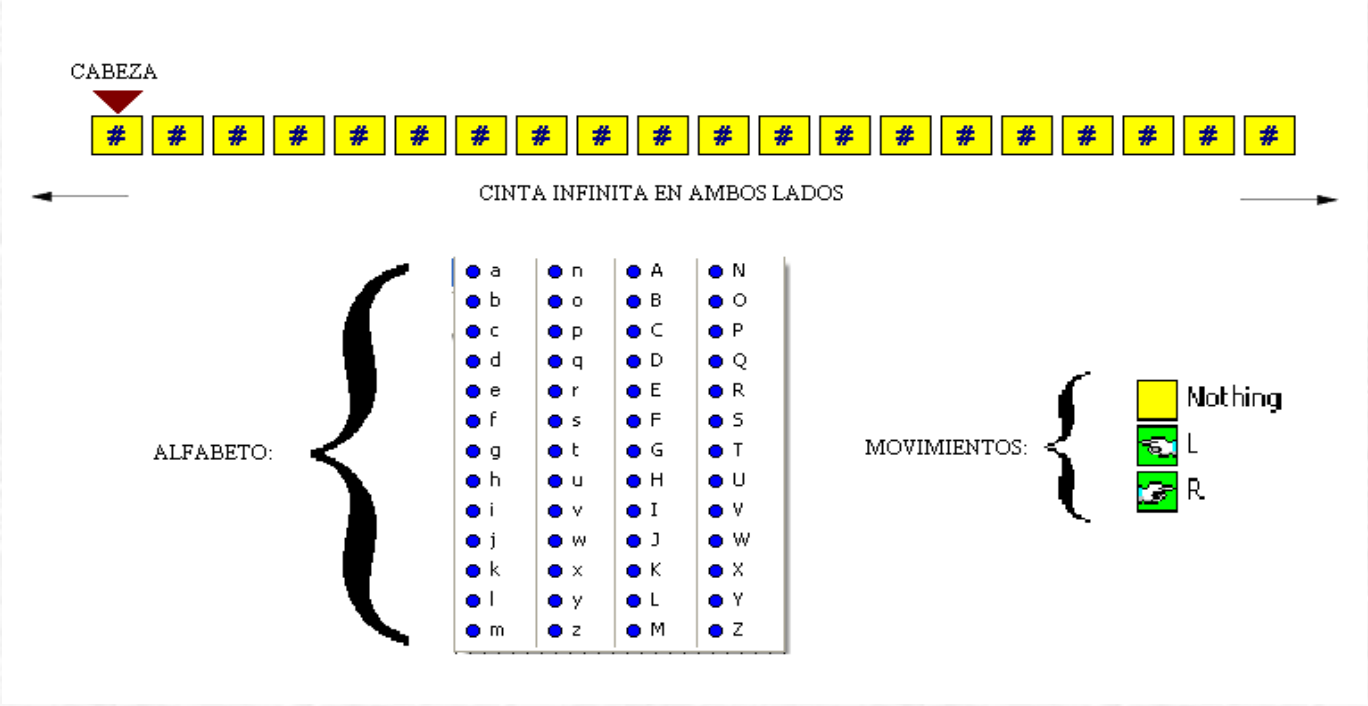
g = función de transición de salida

σ = Posición de la cabeza lectora/escritora (head) sobre la cinta

Movimientos = Derecha, Izquierda, Neutro.

Máquina de Turing

- Posee una cinta infinita compuesta de celdas donde pueden leerse o escribirse los símbolos definidos en el alfabeto de la cinta.



Nota: Utilizar Visual Turing para los ejercicios.

Gramática

La gramática es un ente formal para especificar, de una manera finita, el conjunto de cadenas de símbolos que constituyen un lenguaje.

Una gramática es una cuádrupla **G = (T , NT, σ , P)** donde:

T {conjunto finito de símbolos terminales}

NT {conjunto finito de símbolos no terminales}

σ es el *símbolo inicial* y pertenece a NT.

P {conjunto de producciones o de reglas de derivación}

Gramática

Símbolos Terminales de un lenguaje de programación:

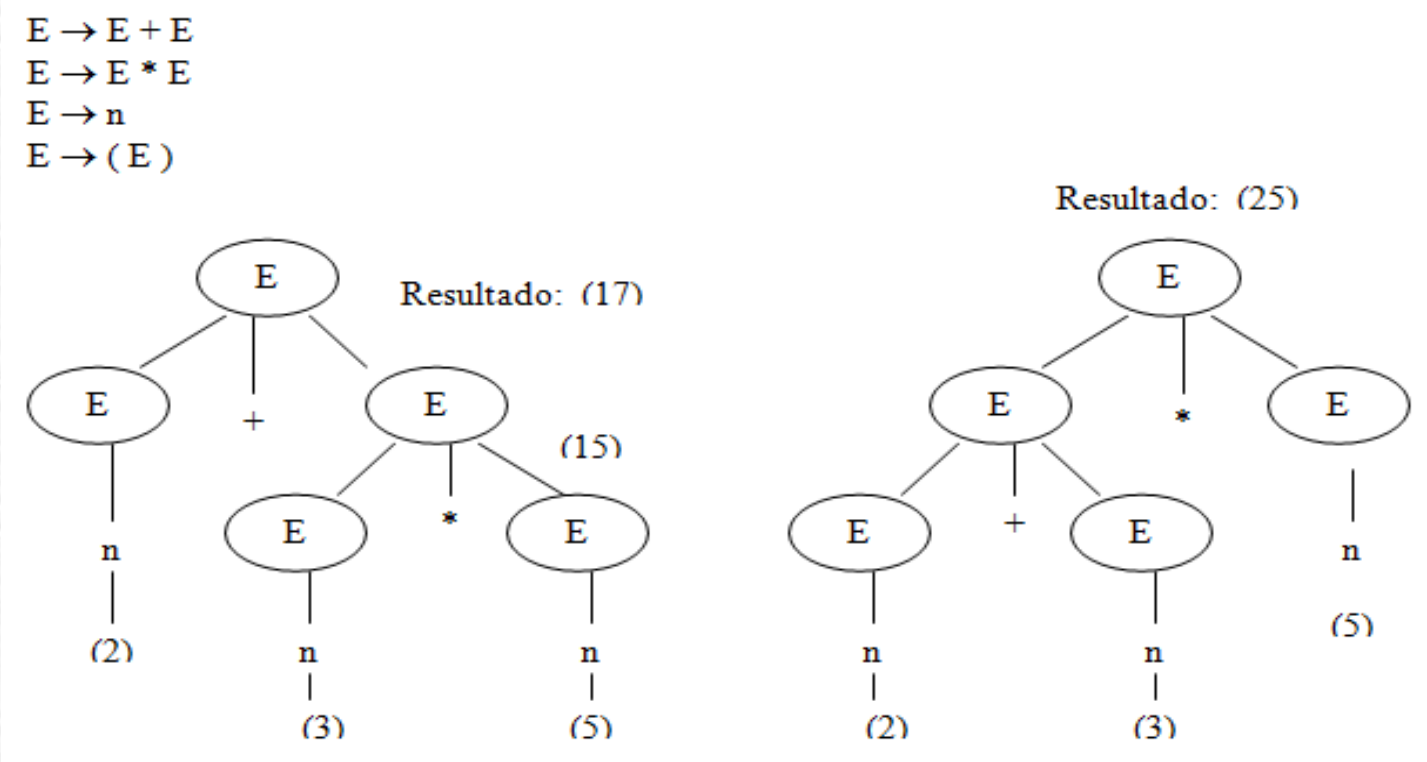
- Conjunto de palabras reservadas de un Lenguaje de Programación
- Conjunto de operadores aritméticos
- Conjunto de operadores lógicos
- Conjuntos de símbolos especiales
- Letras minúsculas y mayúsculas.
- Dígitos
- Etc.

Símbolos No Terminales:

- Símbolo inicial
- Producciones que se sustituyen hasta ser reemplazadas por símbolos terminales

Gramática

- **Equivalencia entre gramáticas:** Dos gramáticas son equivalentes si generan el mismo lenguaje.
- **Ambigüedad.-** Una gramática es ambigua si permite generar distintas derivaciones para una misma expresión.
- Ejemplo:



Derivación por la izquierda

Dada una cadena de entrada, se revisa de izquierda a derecha (L = Left), reemplazando progresivamente los símbolos No Terminales hasta llegar al fin de cadena de izquierda a derecha (L = Left), de ahí el nombre de Parsers LL. (Este método es equivalente al recorrido en Pre-Orden de un Arbol). Los parsers que implementan este tipo de revisión se llaman 'LL' y son de tipo 'Top-Down'.

Ejemplo

Dada la gramática con las producciones:

S → aAS

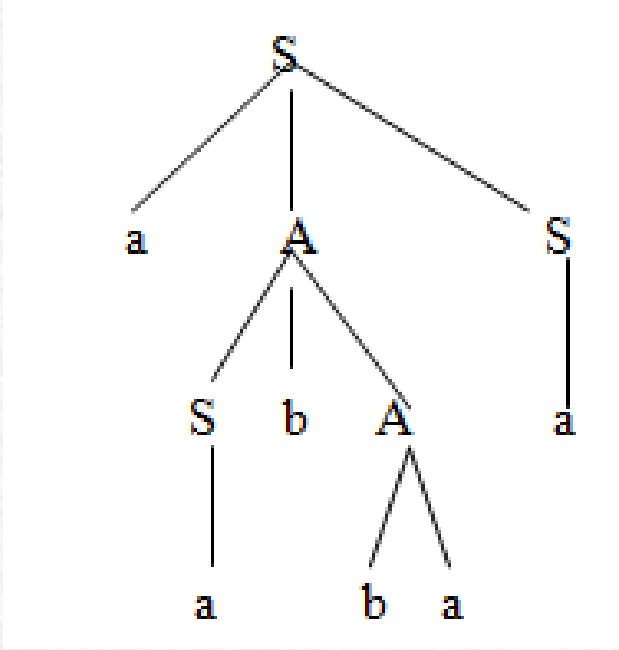
S → a

A → SbA

A → b

A → ba

Arbol de derivación: →



Con derivación por la izquierda, evaluar la cadena: 'aabbaa'

$$S \Rightarrow a\underline{A}S \Rightarrow a\underline{SbA}S \Rightarrow a\underline{a}bAS \Rightarrow aab\underline{ba}S \Rightarrow aabba\underline{a}$$
$$\qquad\qquad\quad A \qquad\quad S \qquad\quad A \qquad\quad S$$

Se obtiene el árbol que se muestra en la Figura 5.

Derivación por la derecha

Dada una cadena de entrada, se revisa de derecha a izquierda (L = left), reemplazando progresivamente la parte derecha de una regla por los símbolos No Terminales hasta llegar al símbolo inicial, caso en que se supone una cadena esta correcta (R = right). (Equivalente al recorrido en Pos-Orden Inverso en un Arbol). Los parsers que implementan este tipo de revisión son denominados 'LR' y son de tipo 'Bottom-Up'.

Ejemplo

Dada la gramática ejemplo anterior:

Con derivación por la derecha, evaluar la cadena: ‘aabbaa’

$$S \Rightarrow aA\underline{S} \Rightarrow aA\underline{a} \Rightarrow a\underline{SbA}a \Rightarrow aSb\underline{ba}a \Rightarrow a\underline{a}bbaa$$

S A A S

Forma Normal de Backüs

Para representar las *reglas sintácticas de una gramática*, se utiliza la Forma Normal de Backüs en su forma básica y extendida.

Para identificar los símbolos no terminales, se usa el par de símbolos **< >** entre los cuales se debe especificar el identificador correspondiente.

Por ejemplo:

<inicio>, <main>, <numero>

se identifican como ***unidades sintácticas o símbolos no terminales*** que deben ser reemplazados hasta ser sustituidos completamente por símbolos terminales.

Forma Normal de Backüs

El símbolo

::= o ➔

Se utiliza después de la identificación de un símbolo terminal, y sirve para fijar una definición que puede interpretarse como "se define" o "produce" o como una implicación.

El símbolo **"|"** indica opcionalidad ("or")

El conjunto finito de todas las reglas sintácticas forman una *gramática*

Ejemplo

La definición para una estructura básica de un lenguaje de programación con la sintaxis BNF para especificar la gramática para la identificación de números sería:

<inicio> ::= <numero>

<numero> ::= <digito> | <digito> [{ <numero> }]

<digito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Otras especificaciones

Los paréntesis se usan para agrupar e indicar a quién se aplica una determinada operación.

Las llaves, { }, representan cero o más *repeticiones*.

Los corchetes, [ ], expresan *opcionalidad.*

Cuando en los dos casos anteriores se desea incluir un símbolo de separación, este se antepone al enunciado seguido de una coma.

Por ejemplo:

{ ‘,’, <numero>}, indica una lista de números separados por comas.

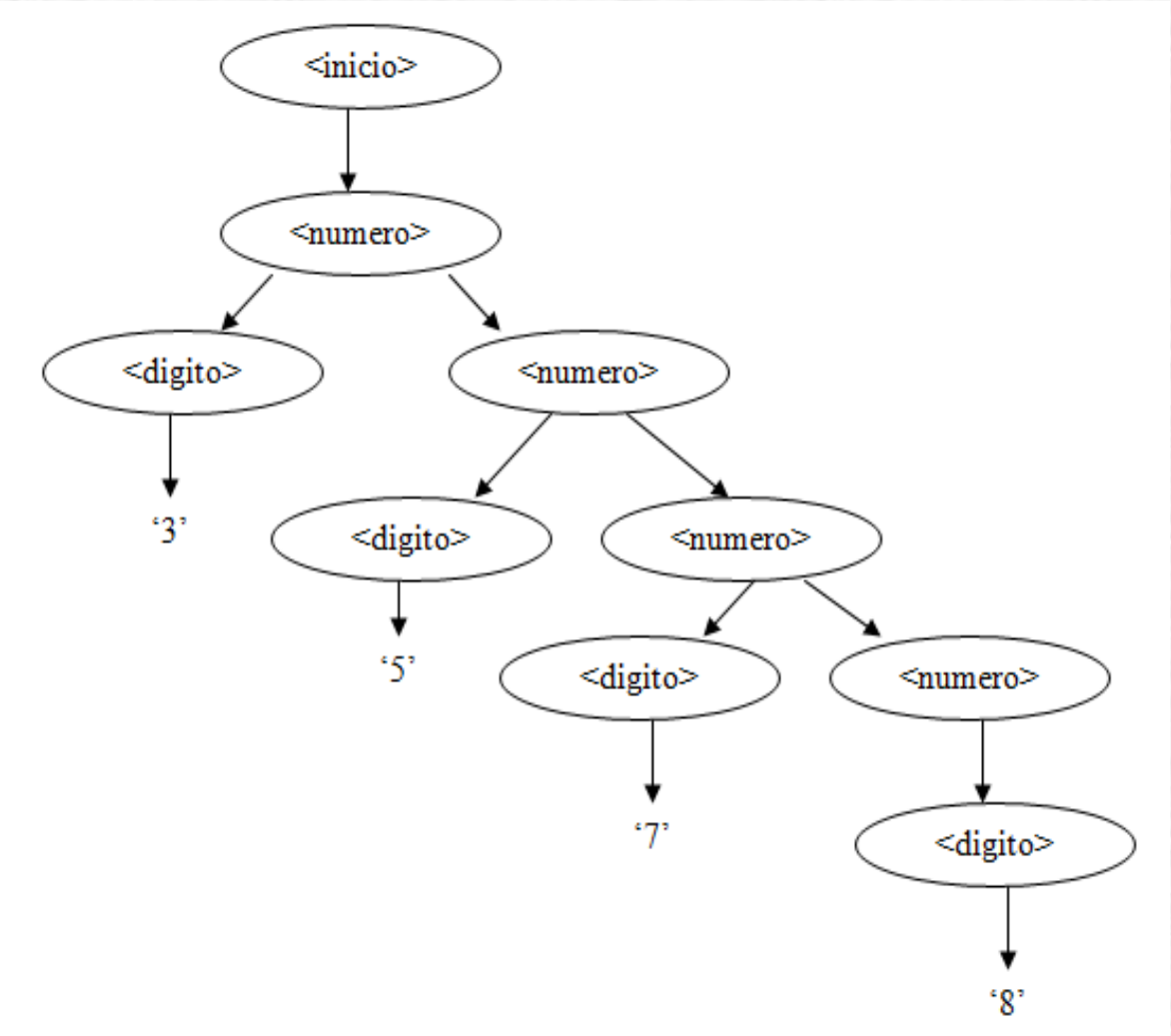
[‘|’, <numero>], indica una lista de números separados por el operador ‘or’.

Arbol sintáctico o árbol de Derivación

Los **árboles sintácticos** son representaciones jerárquicas de las derivaciones que forma una expresión.

Cada nodo no terminal tendrá nodos hijos que pueden ser otros símbolos no terminales símbolos terminales.

Para evaluar la cadena 3578, se obtiene el árbol:



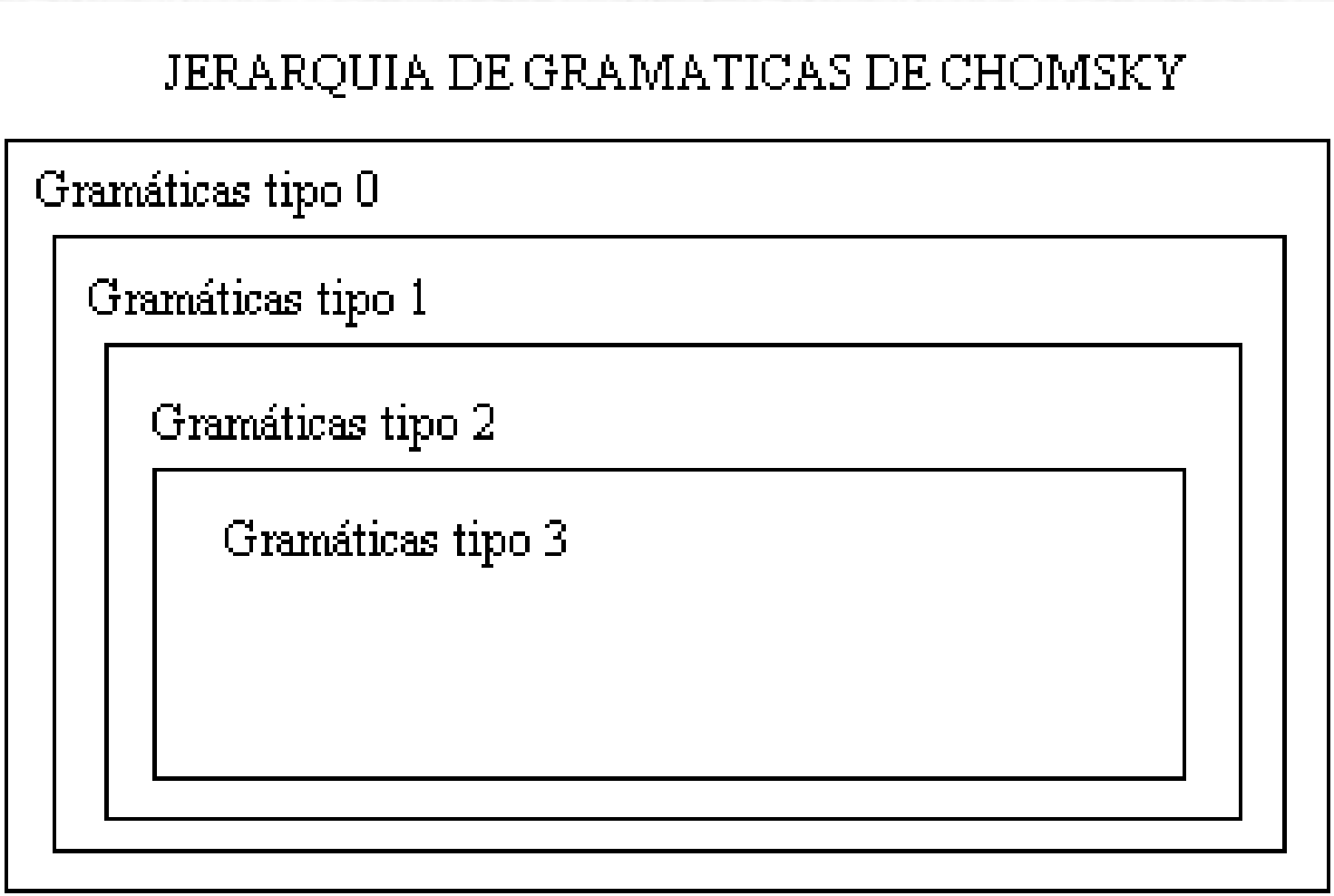
Jerarquía de Gramáticas de Noam Chomsky

Chomsky definió cuatro tipos distintos de gramáticas en función de la forma de las reglas de derivación (*Chomsky, 1959*).

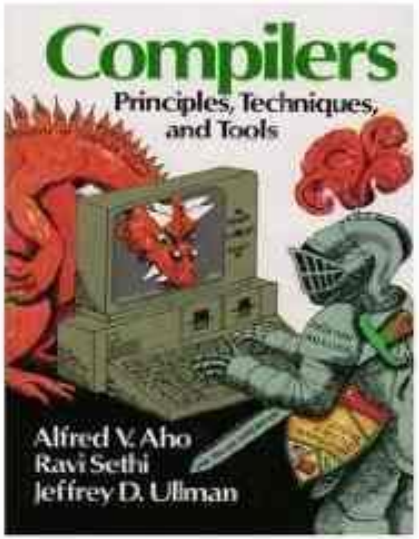
La clasificación comienza con un tipo de gramáticas que pretende ser universal, y aplicando restricciones a sus reglas de derivación se van obteniendo los otros tres tipos de gramáticas.

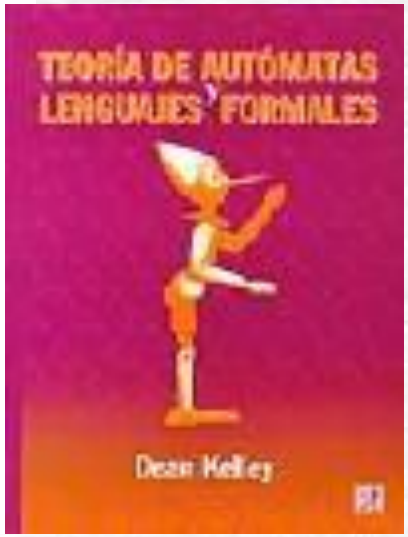
La jerarquía de gramáticas se clasifica en cuatro tipos: 0, 1, 2, y 3, donde la mas general es la 0, y conforme avanza el número que las identifica, aumentan las restricciones. Las gramáticas de tipo 0, contienen a todas las demás. Las de tipo 1 contienen a las de tipo 2, y por último las de tipo 2 contienen a las de tipo 3. Por lo tanto se define una *jerarquía de gramáticas respecto de la relación de inclusión.*

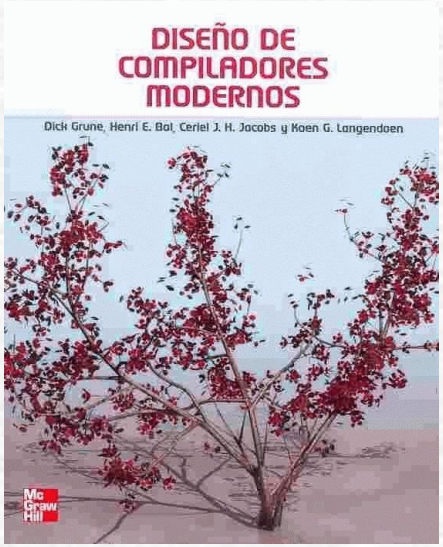
Jerarquía de Chomsky Gráfica

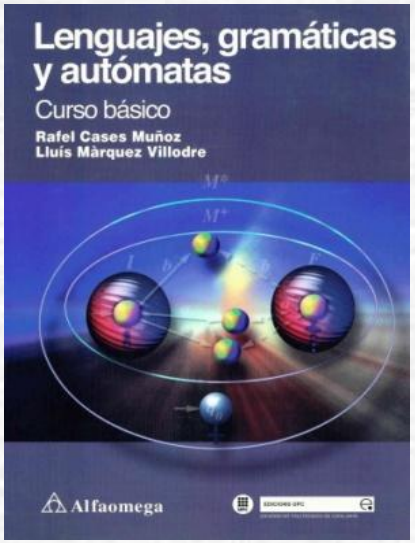


Bibliografía de apoyo a la Unidad









Datos de contacto:

Dra. María de Guadalupe Cota Ortiz

E-Mail:

lcota@gauss.mat.uson.mx

lcota@hades.mat.uson.mx