Universidad Autónoma del Estado de México
Centro Universitario UAEM Texcoco

Departamento de Ciencias Aplicadas.

Ingeniería en Computación.

Autómatas y Lenguajes Formales.

Unidad de competencia IV: "Conocer, utilizar y diseñar gramáticas de libre contexto"

Presenta:

M. en C. C. J. Jair Vázquez Palma.

Autómatas y Lenguajes Formales

Objetivos de la Unidad de la Unidad de Aprendizaje

- Aplicar las estructuras algebraicas fundamentales en el manejo de estructuras de datos.
- Diseñar y simplificar circuitos lógicos.
- Manejo de autómatas y su relación con los lenguajes de programación.
- Representar en modelos formales de cómputo explicando su importancia desde puntos de vista teóricos y prácticos.

Unidad IV: Conocer, utilizar y diseñar gramáticas de libre contexto.

Contenido:

- Gramáticas regulares y gramáticas libres de contexto.
- Árboles de derivación.
- Autómata con pila.
- Equivalencia entre gramáticas y autómatas con pila.

Objetivo de la Unidad de Competencia IV

- El discente diseñara las gramáticas libres de contexto, así como el diaño de autómatas de pila y sus conversiones.

Conocer, utilizar y diseñar gramáticas de libre contexto.

Gramáticas regulares y gramáticas libres de contexto

- Los Lenguajes Libres de Contexto (abreviado LLC) forman una clase de lenguajes más amplia que los Lenguajes Regulares, de acuerdo con la Jerarquía de Chomsky.

- Estos lenguajes son importantes tanto desde el punto de vista teórico, por relacionar las llamadas Gramáticas Libres de Contexto con los Autómatas de Pila, como desde el punto de vista practico, ya que casi todos los lenguajes de programación están basados en los LLC.

- Por otra parte, el análisis automático de los LLC es computacionalmente mucho más eficiente que el de otras clases de lenguajes más generales.

Gramáticas regulares y gramáticas libres de contexto

- Una ***gramática*** es básicamente un conjunto de reglas.

Por ejemplo, consideremos, la siguiente gramática para producir un pequeño subconjunto del idioma español:

1.-<frase> → <sujeto> <predicado>

2.-<sujeto> → <articulo> <sustantivo>

3.-<articulo> → el | la

4.-<sustantivo> → perro | luna

5.-<predicado> → <verbo>

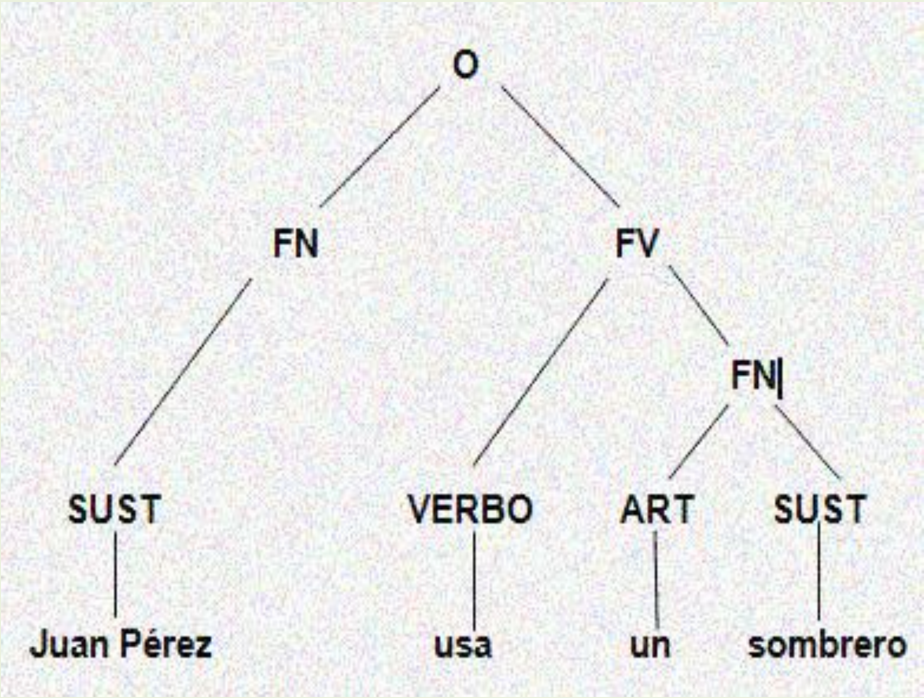
6.-<verbo> → brilla | corre

Donde el símbolo “|” separa varias alternativas.

En esta gramática se supone que las variables son <frase>, <sujeto>, <articulo>, <sustantivo>, <predicado> y <verbo>, mientras que las constantes son ***el , la, perro y luna***. La variable <frase> será considerada el símbolo inicial.

Gramáticas regulares y gramáticas libres de contexto

Ejemplo, consideremos, el siguiente árbol de una oración en español:



O → *FN FV*
FV → VERBO FN| VERBO
FN → SUST |
ART SUT |
SUST ADJ |
ART SUST ADJ|

SUST → cosa | árbol | nombre
ADJ → azul | grande
VERBO → tiene | estudia
ART → la | los | el | un

Gramáticas regulares y gramáticas libres de contexto

DEFINICION FORMAL DE GRAMATICA

Definición.- Una gramática regular es un cuádruplo (V, Σ,R, S) en donde:

V es un alfabeto de variables,

Σ es un alfabeto de constantes,

R, el conjunto de reglas, es un subconjunto finito de V × (Σ U Σ).

S, el símbolo inicial, es un elemento de V .

Por ejemplo, la gramática que presentamos arriba se representaría formalmente como:

({S, A,B}, {a, b}, {(S, aA), (S, bA), (A, aB), (A, bB), (A, a), (B, aA), (B, bA)}, S)

Gramáticas regulares y gramáticas libres de contexto

Ejemplo.- Sea una gramática con las siguientes reglas:

1. S → aA
2. S → bA
3. A → aB
4. A → bB
5. A → a
6. B → aA
7. B → bA

- La idea para aplicar una gramática es que se parte de una variable, llamada símbolo inicial, y se aplican repetidamente las reglas gramaticales, hasta que ya no haya variables en la palabra.

- En ese momento se dice que la palabra resultante es generada por la gramática, o en forma equivalente, que la palabra resultante es parte del lenguaje de esa gramática.

Gramáticas regulares y gramáticas libres de contexto

({S, A,B}, {a, b}, {(S, aA), (S, bA), (A, aB), (A, bB), (A, a), (B, aA), (B, bA)}, S)

V es un alfabeto de variables,

Σ es un alfabeto de constantes,

R, el conjunto de reglas, es un subconjunto finito de V × (Σ U Σ).

S, el símbolo inicial, es un elemento de V .

1. S → aA
2. S → bA
3. A → aB
4. A → bB
5. A → a
6. B → aA
7. B → bA

Lenguaje Generado

Definición.- El lenguaje generado por una gramática G, *L(G)*, es igual al conjunto de las palabras derivables a partir de su símbolo inicial.

Esto es $L(G) = \{ w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w \}$

Gramáticas regulares y gramáticas libres de contexto

Lenguaje Generado

Definición.- El lenguaje generado por una gramática G, *L(G)*, es igual al conjunto de las palabras derivables a partir de su símbolo inicial.

Esto es → $L(G) = \{w \in \Sigma^* | S \stackrel{*}{\Rightarrow} w\}$

Nota: Más adelante veremos como probar rigurosamente que una gramática efectivamente corresponde a un lenguaje dado.

Gramáticas regulares y gramáticas libres de contexto

Ejemplo.- Proponer una gramática que genere el lenguaje de las palabras en {a, b} que contienen la subcadena ***bb***, como:

abb, ababba, etc.

Se proponen las siguientes reglas:

1. A → aA
2. A → bB
3. B → aA
4. B → bC
5. C → aC
6. C → bC

En efecto, con tales reglas podemos producir, por ejemplo, la palabra ***abba***, mediante la derivación siguiente:

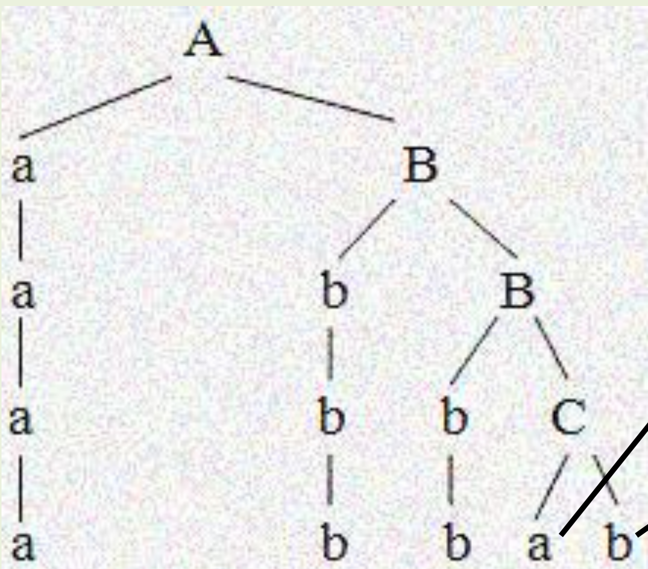
A ⇒ aA ⇒ abB ⇒ abbC ⇒ abba

Y

A ⇒ aA ⇒ abB ⇒ abbC ⇒ abbb

Gramáticas regulares y gramáticas libres de contexto

El árbol generado queda de la siguiente manera:



abba

abbb

Gramáticas regulares y gramáticas libres de contexto

Ejemplo.- Verificar que lenguaje se genera con las siguientes reglas.

Dado:

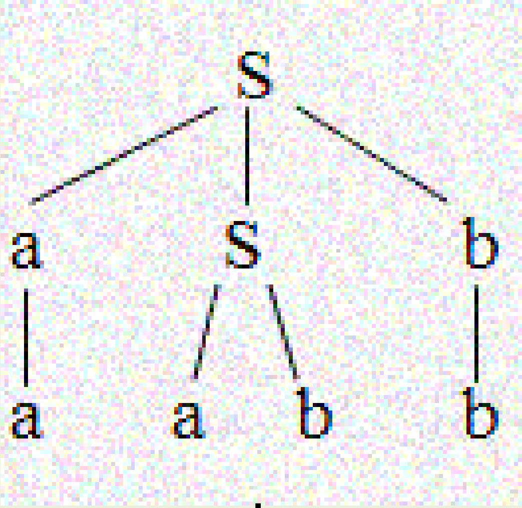
$$P = \begin{cases} S \rightarrow aSb \\ S \rightarrow ab \end{cases}$$

Gramáticas regulares y gramáticas libres de contexto

Ejemplo.- Verificar que lenguaje se genera con las siguientes reglas.

Dado:

$$P = \begin{cases} S \rightarrow aSb \\ S \rightarrow ab \end{cases}$$



$S \Rightarrow aSb \Rightarrow aabb$

¿Esta reglas son recursivas?

Lenguaje generado por una gramática:

El lenguaje generado por una gramática es el conjunto de palabras hechas exclusivamente de constantes que son derivables a partir del símbolo inicial.

Se define: $L(G) = \{w \in \Sigma^* | S \overset{*}{\Rightarrow} w\}$

Dado:

$$P = \begin{cases} S \rightarrow aSb \\ S \rightarrow ab \end{cases}$$

Gramáticas regulares y gramáticas libres de contexto

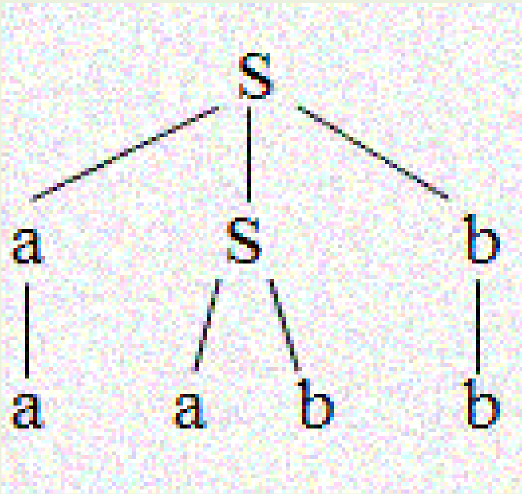
Dado:

$$P = \begin{cases} S \rightarrow aSb \\ S \rightarrow ab \end{cases}$$

$$\begin{aligned} S &\Rightarrow aSb \\ &\Rightarrow aaSbb \\ &\Rightarrow a^{n-1} S b^{n-1} \\ &\Rightarrow a^{n-1} ab b^{n-1} \\ &\Rightarrow a^n b^n \end{aligned}$$

Lenguaje generado

$$L(G) = \{a^n b^n \mid n>0\}$$



Gramáticas regulares y gramáticas libres de contexto

Dado las siguientes reglas gramaticales, generar el lenguaje:

$S \rightarrow bS$

$S \rightarrow ax$

$X \rightarrow bx$

$X \rightarrow b$

Generar árbol de derivaciones.

$S \Rightarrow bS$

$\Rightarrow b\ ax$

$\Rightarrow b\ abx$

$\Rightarrow b\ ab^m$

$L(G) = \{b\ ab^m \mid m > 0\}$

GERARQUIA DE GRAMATICAS.

Noam Chomsky estableció la siguiente jerarquía de gramáticas:

Gramáticas regulares, o de tipo 3: las reglas son de la forma

$A \rightarrow aB$ o bien $A \rightarrow a$, donde A y B son variables y a es constante. Estas gramáticas son capaces de describir los lenguajes regulares.

Gramáticas Libres de Contexto (GLC), o de tipo 2: las reglas son de la forma $X \rightarrow \alpha$, donde ***X*** es una variable y es una cadena que puede contener variables y constantes. Estas gramáticas producen los lenguajes Libres de Contexto (abreviado "LLC").

GERARQUIA DE GRAMATICAS.

Gramáticas sensitivas al contexto o de tipo 1: las reglas son de la forma $\alpha A \beta \rightarrow \alpha \Gamma \beta$, donde A es una variable y , α, β y Γ son cadenas cualesquiera que pueden contener variables y constantes.

Gramáticas no restringidas, o de tipo 0, con reglas de la forma $\alpha \rightarrow \beta$, donde **α** no puede ser vacío, que generan los lenguajes llamados “recursivamente enumerables”.

Gramáticas regulares y gramáticas libres de contexto

- Desde el punto de vista de el diseño de construcción de compiladores solo las gramáticas de tipo 2 y 3 pueden ser tomadas en la practica como fundamento formal.

- Las gramáticas de tipo 3 son el fundamento del análisis léxico mediante la utilización de expresiones regulares.

- Las gramáticas de tipo 2 son el fundamento del análisis sintáctico.

Gramáticas regulares y gramáticas libres de contexto

➢**Aunque los lenguajes de programación pueden ser sensibles al contexto, la construcción de un analizador para una gramática de tipo 1 es mucho mas compleja que la de tipo 2, por ello la dependencia del contexto se resuelve en el análisis semántico.**

EJERCICIO

Como un ejercicio adicional, generar la gramática con reglas que permitan generar expresiones aritméticas con sumas y multiplicaciones de enteros. Función a derivar: **25 + 3 * 2.**

?

Gramáticas regulares y gramáticas libres de contexto

EJERCICIO

Solución:

La gramática con las reglas siguientes permite generar expresiones

1. E → E + T

2. E → T

3. T → T * F

4. T → F

5. F → CF

6. F → C

7. C → 0|1|2|3|4|5|6|7|8|9 aritméticas con sumas y multiplicaciones de enteros:

Gramáticas regulares y gramáticas libres de contexto

EJERCICIO: Derivar: 25 + 3 * 2. El símbolo inicial aquí es E, las constantes son +, * y las cifras 0 - 9; E, T, F,C son variables.

EXPRESION	JUSTIFICACION
E	Símbolo inicial, inicia derivación
$\Rightarrow E + T$	Aplicación 1a. regla
$\Rightarrow T + T$	2a. regla, sobre la E
$\Rightarrow F + T$	4a. regla, sobre la T izquierda
$\Rightarrow CF + T$	5a. regla, sobre F
$\Rightarrow 2F + T$	7a. regla
$\Rightarrow 2C + T$	6a. regla
$\Rightarrow 25 + T$	7a. regla
$\Rightarrow 25 + T * F$	3a. regla
$\Rightarrow 25 + F * F$	4a. regla
$\Rightarrow 25 + C * F$	6a. regla, sobre la F izquierda
$\Rightarrow 25 + 3 * F$	7a. regla
$\Rightarrow 25 + 3 * CF$	5a. regla
$\Rightarrow 25 + 3 * 1F$	7a. regla
$\Rightarrow 25 + 3 * 1C$	6a. regla
$\Rightarrow 25 + 3 * 12$	7a. regla

Generar el árbol de derivaciones

FORMALIZACION DE LAS GLC

Definición.- Una gramática libre de contexto es un cuádruplo (V, Σ,R, S) en donde:

V es un alfabeto de variables, también llamadas ***no-terminales***.

Σ es un alfabeto de constantes, también llamadas ***terminales.***

R, el conjunto de reglas, es un subconjunto finito de V × (Σ U Σ)*.

S, el símbolo inicial, es un elemento de V .

Por ejemplo, tenido las siguientes reglas gramaticales:

1. S → aSb
2. S → ab

ARBOLES DE DERIVACION

Las GLC tienen la propiedad de que las derivaciones pueden ser representadas en forma arborescente.

Por ejemplo, considérese la gramática siguiente para producir el lenguaje de los paréntesis bien balanceados, que tiene palabras como (()), ()(), (()())(), pero no a (() ni)(:

1. S → SS
2. S → (S)
3. S → ()

Árboles de derivación

FORMALIZACION DE LAS GLC

Por ejemplo, tenido las siguientes reglas gramaticales:

1. S → aSb
2. S → ab

La gramática de $\{a^n b^n\}$ que presentamos antes se representa formalmente como:

({S}, {a, b}, {(S, aSb), (S, ab)}, S)

V es un alfabeto de variables

Σ es un alfabeto de constantes

R, el conjunto de reglas

S, el símbolo inicial, es un elemento de V .

Árboles de derivación

ARBOLES DE DERIVACION

1. S → SS
2. S → (S)
3. S → ()

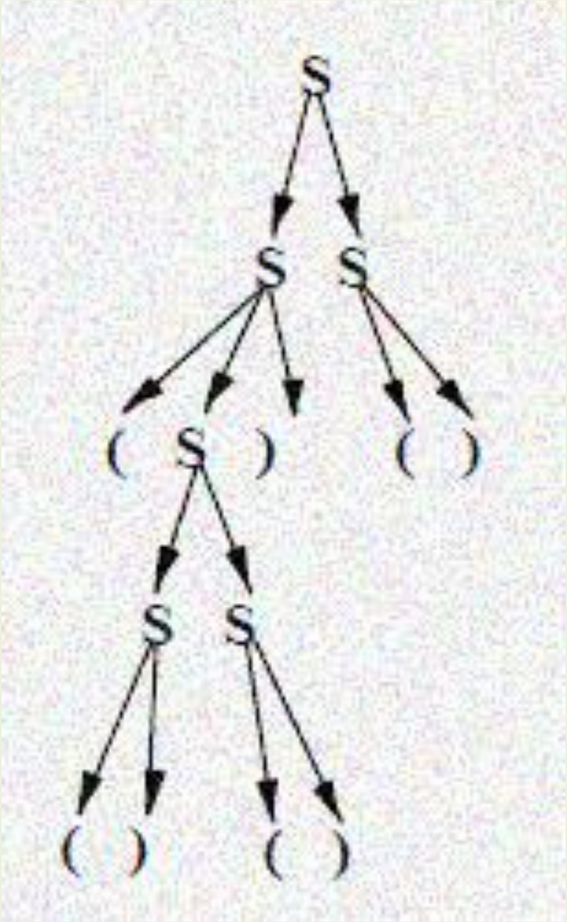
Usando la gramática anterior como podemos derivar la palabra (()())():

Árboles de derivación

ARBOLES DE DERIVACION: Usando esta gramatical, la palabra (()())() puede ser derivada de la siguiente manera:

- En dicha figura se puede apreciar la estructura que se encuentra implícita en la palabra (()())().

- A estas estructuras se les llama ***árboles de derivación***, o también ***árboles de compilación*** (por usarse extensivamente en los compiladores) y son de vital importancia para la teoría de los compiladores de los lenguajes de programación.



ARBOLES DE DERIVACION

Soluciones:

Para el árbol de la figura anterior, hay al menos las derivaciones siguientes (anotamos como subíndice de ⇒ el número de regla aplicado):

1. $S \Rightarrow_1 SS \Rightarrow_2 (S)S \Rightarrow_3 (S)() \Rightarrow_1 (SS)() \Rightarrow_3 (S())() \Rightarrow_3 (()())().$
2. $S \Rightarrow_1 SS \Rightarrow_3 S() \Rightarrow_2 (S)() \Rightarrow_1 (SS)() \Rightarrow_3 (()S)() \Rightarrow_3 (()())().$

Ejercicio.

Realizar el árbol de derivaciones.

Árboles de derivación

ARBOLES DE DERIVACION: Formalmente, un árbol de derivación es un grafo dirigido arborescente; definido de la manera siguiente:

Definición.- Sea G = (V, ∑,R, S) una GLC. Entonces un ´árbol de derivación cumple las siguientes propiedades:

1. Cada nodo tiene una etiqueta.

2. La raíz tiene etiqueta S.

3. La etiqueta de los nodos que no son hojas debe estar en V, y las de las hojas en ∑ U { ξ }.

4. Si un nodo n tiene etiqueta ***A***, y los nodos $n_1, \ldots, n_m$ son sus hijos (de izquierda a derecha), con etiquetas respectivamente $A_1, \ldots, A_m$, entonces $A \rightarrow A_1, \ldots, A_m \in R$.

ARBOLES DE DERIVACION. Definición teórica.

Definición.- La cadena de caracteres que resulta de concatenar los caracteres terminales encontrados en las etiquetas de los nodos hoja, en un recorrido en orden del árbol de derivación, se llama el **producto del árbol**.

Es decir, al efectuar un recorrido en orden del árbol de derivación recuperamos la cadena a partir de la cual se construyó dicho árbol.

Así, el problema de "compilar" una cadena de caracteres consiste en construir el árbol de derivación a partir del producto de éste.

AMBIGÜEDAD

Hay GLC en las cuales para ciertas palabras hay más de un árbol de derivación. Sea por ejemplo la siguiente GLC, para expresiones aritméticas sobre las variables ***x*** y ***y***.

1. E ⇒ E + E
2. E ⇒ E * E
3. E ⇒ x
4. E ⇒ y

Con esta gramática, para la expresión

x + y * x existen los dos árboles de derivación. Verificar cuales son esos dos arboles.

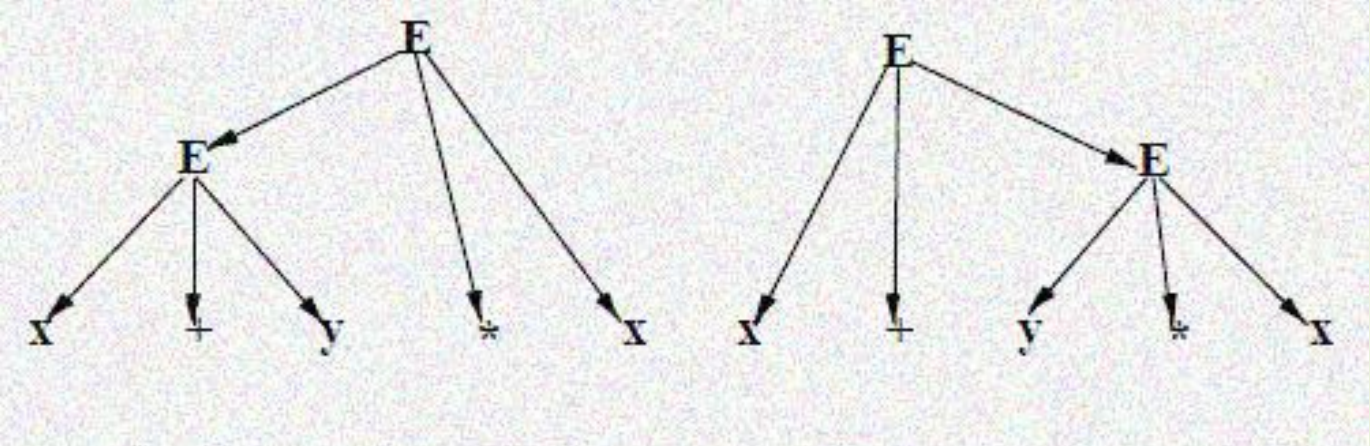
Árboles de derivación

AMBIGÜEDAD

Solución:

Con esta gramática, para la expresión **x + y * x** existen los dos árboles de derivación las cuales se muestran a continuación.

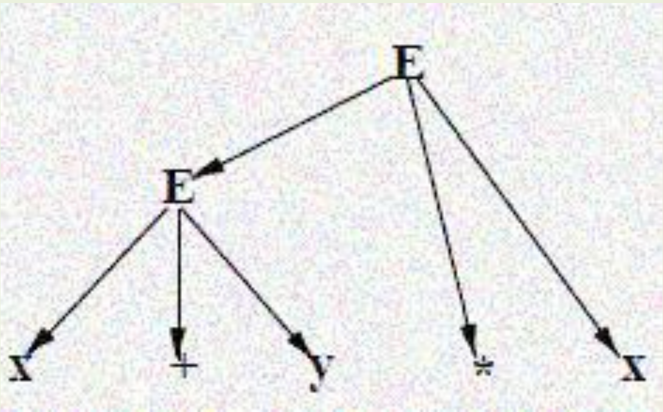
1. E ⇒ E + E
2. E ⇒ E * E
3. E ⇒ x
4. E ⇒ y

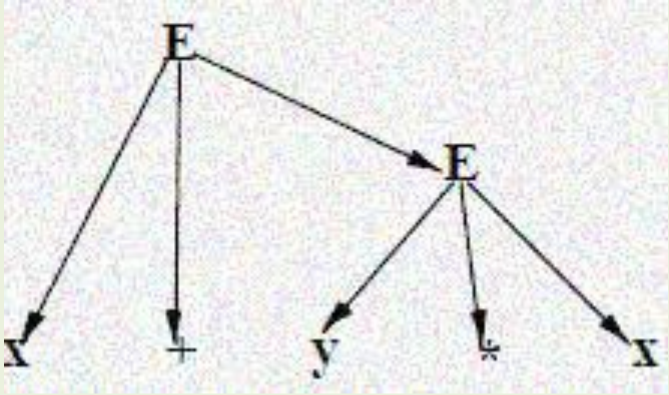


(a) (b)

Árboles de derivación

Ambigüedad:





En este ejemplo, el hecho de que existan dos árboles de derivación para una misma expresión es indeseable, pues cada árbol indica una manera distinta de estructurar la expresión.

En efecto, en el árbol superior, al resultado de la suma (x + y) se multiplica con x, mientras que en el inferior sumamos x al resultado de multiplicar x con y; por lo tanto el significado que se asocia a ambas expresiones puede ser distinto.

Árboles de derivación

AMBIGÜEDAD:

- Se dice que una gramática es ***ambigua*** si alguna palabra del lenguaje que genera tiene más de un árbol de derivación.

- Nótese que la ambigüedad, como la estamos definiendo, es una propiedad de la gramática, no de su lenguaje generado.

- Para un mismo lenguaje puede haber una gramática ambigua y una no ambigua.

Árboles de derivación

AMBIGÜEDAD:

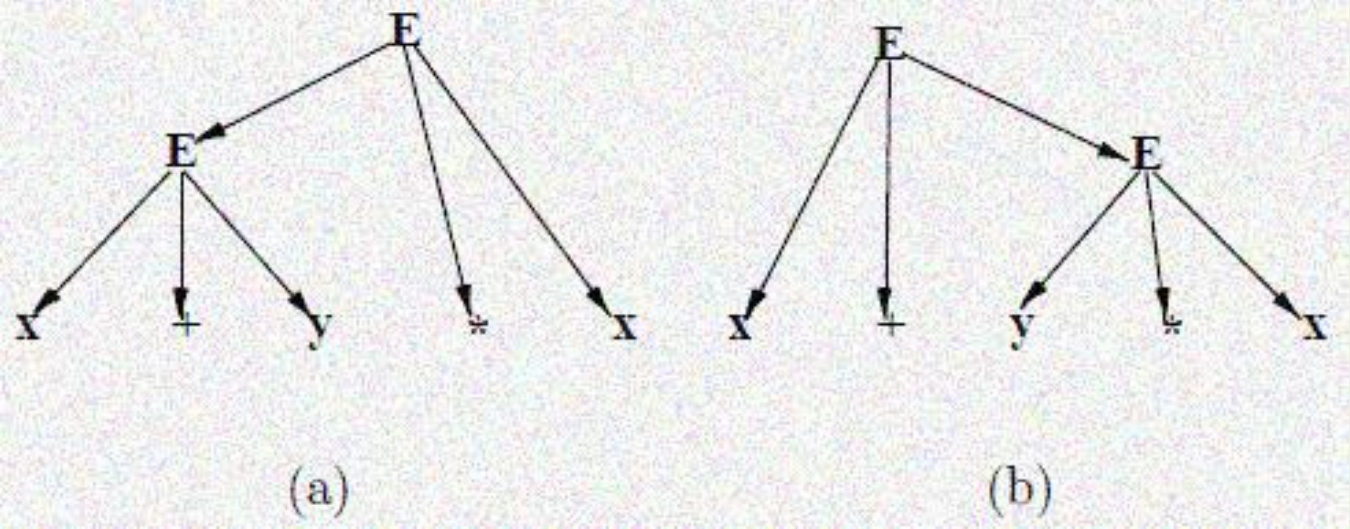
Existen técnicas para eliminar la ambigüedad de una GLC; en general estas técnicas consisten en introducir nuevos ***no-terminales*** de modo que se eliminen los árboles de derivación no deseados.

Para nuestro ejemplo de los operadores aritméticos, es clásica la solución que consiste en introducir, además de la categoría de las Expresiones (no-terminal E), la de los términos (T) y factores (F), dando la gramática con las reglas siguientes:

Árboles de derivación

AMBIGÜEDAD: Dando la gramática con las reglas siguientes:

1. E → E + T
2. E → T
3. T → T * F
4. T → F
5. F → (E)
6. F → x
7. F → y



Con esta nueva GLC, el árbol de derivación de la figura (a) se elimina, quedando finalmente una adaptación del árbol de (b) a la GLC con términos y factores, lo cual se deja como ejercicio.

Autómatas con pila

- La pila funciona de manera que el último carácter que se almacena en ella es el primero en salir (“LIFO” por las siglas en ingles), como si empiláramos platos uno encima de otro, y naturalmente el primero que quitaremos es el ´ultimo que hemos colocado.
- Un aspecto crucial de la pila es que sólo podemos modificar su “tope”, que es el extremo por donde entran o salen los caracteres. Los caracteres a la mitad de la pila no son accesibles sin quitar antes los que están encima de ellos.
- La pila tendrá un alfabeto propio, que puede o no coincidir con el alfabeto de la palabra de entrada. Esto se justifica porque puede ser necesario introducir en la pila caracteres especiales usados como separadores, según las necesidades de diseño del autómata.

Autómatas con pila

•Al iniciar la operación de un AP, la pila se encuentra vacía. Durante la operación del AP, la pila puede ir recibiendo (y almacenando) caracteres, según lo indiquen las transiciones ejecutadas. Al final de su operación, para aceptar una palabra, la pila debe estar nuevamente vacía.

•En los AP las transiciones de un estado a otro indican, además de los caracteres que se consumen de la entrada, también lo que se saca del tope de la pila, así como también lo que se mete a la pila.

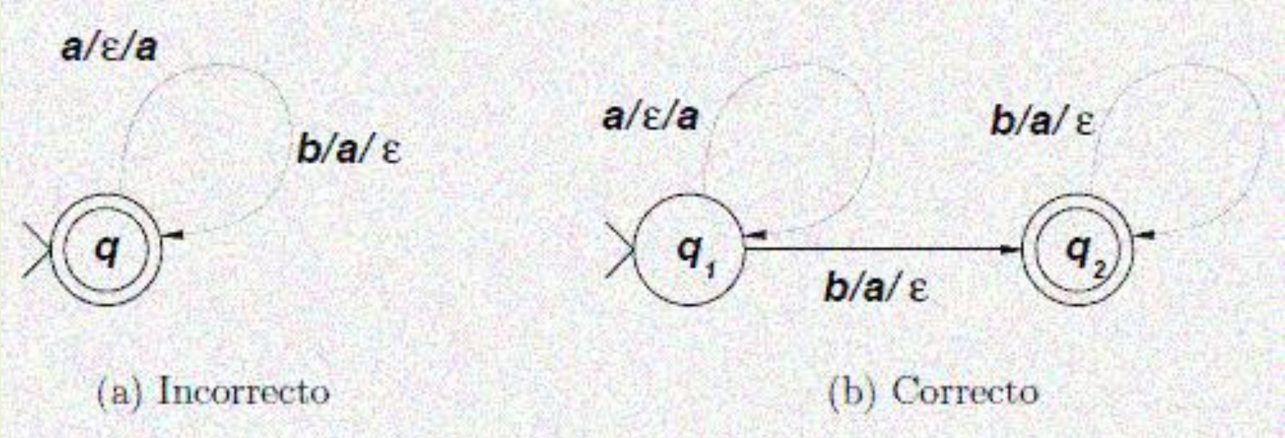
Autómatas con pila

•Al iniciar la operación de un AP, la pila se encuentra vacía. Durante la operación del AP, la pila puede ir recibiendo (y almacenando) caracteres, según lo indiquen las transiciones ejecutadas. Al final de su operación, para aceptar una palabra, la pila debe estar nuevamente vacía.

•En los AP las transiciones de un estado a otro indican, además de los caracteres que se consumen de la entrada, también lo que se saca del tope de la pila, así como también lo que se mete a la pila.

Autómatas con pila

•Antes de formalizar los AP, vamos a utilizar una notación grafica, parecida a la de los diagramas de los autómatas finitos. Para las transiciones usaremos la notación "*w/α/β*", donde ***w*** es la entrada (secuencia de caracteres) que se consume, ***α*** es lo que se saca de la pila, y ***β*** lo que se mete a la pila.



AP para el lenguaje $a^n b^n$

Autómatas con pila

Al igual que los AF, los AP tienen estados finales, que permiten distinguir cuando una palabra de entrada es aceptada.

De hecho, para que una palabra de entrada sea aceptada en un AP se deben cumplir todas las condiciones siguientes:

- La palabra de entrada se debe haber agotado (consumido totalmente).
- El AP se debe encontrar en un estado final.
- La pila debe estar vacía.

Autómatas con pila

Al igual que los AF, los AP tienen estados finales, que permiten distinguir cuando una palabra de entrada es aceptada.

De hecho, para que una palabra de entrada sea aceptada en un AP se deben cumplir todas las condiciones siguientes:

- La palabra de entrada se debe haber agotado (consumido totalmente).
- El AP se debe encontrar en un estado final.
- La pila debe estar vacía.

Formalización de los AP.

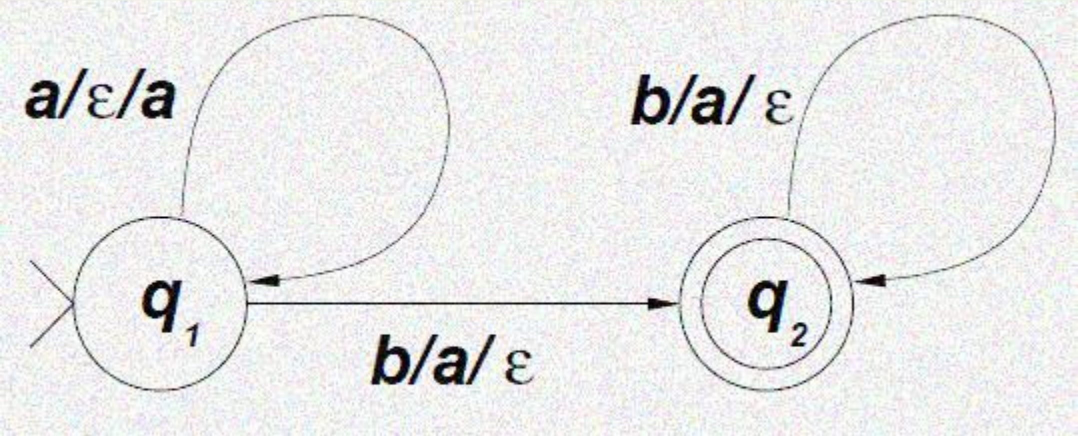
Un autómata de pila es un séxtuplo $(K, \Sigma, \Gamma, \Delta, S, F)$, donde:

- K es un conjunto de estados,
- Σ es el alfabeto de entrada,
- Γ es el alfabeto de la pila,
- $S \in K$ es el estado inicial,
- $F \subseteq K$ es un conjunto de estados finales,
- $\Delta \subseteq (K \times \Sigma^* \times \Gamma^*) \times (K \times \Gamma^*)$ es la relación de transición.

Ahora describiremos el funcionamiento de los AP. Si tenemos una transición de la forma $((\rho, \upsilon, \beta), (q, \Upsilon)) \in \Delta$, el AP hace lo siguiente:

- Estando en el estado $\boldsymbol{\rho}$, consume $\boldsymbol{\upsilon}$ de la entrada;
- Saca $\boldsymbol{\beta}$ de la pila;
- Llega a un estado **q**;
- Mete $\boldsymbol{\Upsilon}$ en la pila

Ejemplo.- Formalizar el AP de la siguiente figura, que acepta el lenguaje $\{ww^R\}$, w ∈ {a, b}.



Ejemplo.- Formalizar el AP de la siguiente figura, que acepta el lenguaje $\{ww^R\}$, w ∈ {a, b}.

•Solución.- El AP es el séxtuplo (K, Σ, Γ, Δ, S, F), donde:

K = {s, f}, F = {f}, Σ = {a, b, c}, Γ = {a, b}

Δ está representada en la siguiente tabla:

(s, a, ε)	(s, a)
(s, b, ε)	(s, b)
$(s, \varepsilon, \varepsilon)$	(f, ε)
(f, a, a)	(f, ε)
(f, b, b)	(f, ε)

Relación entre AP y GLC

- Ahora vamos a establecer el resultado por el que iniciamos el estudio de los AP, es decir, verificar si son efectivamente capaces de aceptar los LLC.

Teorema.- Los autómatas de pila aceptan exactamente los LLC.

Equivalencia entre GLC y AP

- Vamos a examinar la prueba de esta afirmación, no solamente por el interés por la rigurosidad matemática, sino sobre todo porque provee un método de utilidad práctica para transformar una GLC en un AP. La prueba de este teorema se puede dividir en dos partes:

- 1. Si M es un AP, entonces L(M) es un LLC
- 2. Si L es un LLC, entonces hay un AP M tal que L(M) = L

Equivalencia entre GLC y AP

Sea una gran ática G = (V, Σ,R, S). Entonces un AP *M* que acepta exactamente el lenguaje generado por *G* se define como sigue:

$$M = (\{p, q\}, \Sigma , V \cup \Sigma , p, \{q\})$$

donde Δ contiene las siguientes transiciones:

- 1. Una transición ((p, ε, ε), (q, S))
- 2. Una transición ((q, ε,A), (q, x)) para cada A → x ∈ R
- 3. Una transición ((q, σ, σ), (q, ε)) para cada σ ∈ *Σ.*

Equivalencia entre GLC y AP

- Ejemplo.- Obtener un AP que acepte el LLC generado por la gramática con reglas:

1. S → aSa
2. S → bSb
3. S → c

Equivalencia entre GLC y AP

Las transiciones del AP correspondiente están dadas en la tabla siguiente:

1. S → aSa
2. S → bSb
3. S → c

1	$(p, \varepsilon, \varepsilon)$	(q, S)
2	(q, ε, S)	(q, aSa)
3	(q, ε, S)	(q, bSb)
4	(q, ε, S)	(q, c)
5	(q, a, a)	(q, ε)
6	(q, b, b)	(q, ε)
7	(q, c, c)	(q, ε)

Equivalencia entre GLC y AP

El funcionamiento de este AP ante la palabra ***abcba*** aparece en la siguiente tabla:

Estado	Falta leer	Pila
p	$abcba$	ε
q	$abcba$	S
q	$abcba$	aSa
q	$bcba$	Sa
q	$bcba$	$bSba$
q	cba	Sba
q	cba	cba
q	ba	ba
q	a	a
q	ε	ε

Equivalencia entre GLC y AP

- La equivalencia de los AP y de las GLC permite aplicar todas las propiedades de los LLC para resolver problemas de diseño de AP.

BIBLIOGRAFÍA

1.- Aho V. Alfred 2008, “Compiladores; principios, técnicas y herramientas” 2da Edición, Pearson Addison Wesley.

2.- Hopcroft E. Jhon 2008, “Teoría de autómatas lenguajes y computación” 3ra Edición, Ed. Pearson Addison Wesley.

3.- Brena R. 2003 “Autómatas y lenguajes”, Tecnológico de Monterrey.