GRAMATICAS LIBRES DEL CONTEXTO

Estas gramáticas, conocidas también como gramáticas de tipo 2 o gramáticas independientes del contexto, son las que generan los lenguajes libres o independientes del contexto. Los lenguajes libres del contexto son aquellos que pueden ser reconocidos por un autómata de pila determinístico o no determinístico.

Como toda gramática se definen mediante una cuadrupla G = <N, T, P, S>, siendo

- N es un conjunto finito de símbolos no terminales
- T es un conjunto finito de símbolos terminales $N \cap T = \varnothing$
- P es un conjunto finito de producciones
- S es el símbolo distinguido o axioma $S \notin (N \cup T)$

En una gramática libre del contexto, cada producción de P tiene la forma

$$A \rightarrow \omega \qquad \begin{cases} A \in N \cup \{S\} \\ \omega \in (N \cup T)^* - \{\varepsilon\} \end{cases}$$

Es decir, que en el lado izquierdo de una producción pueden aparecer el símbolo distinguido o un símbolo no terminal y en el lado derecho de una producción cualquier cadena de símbolos terminales y/o no terminales de longitud mayor o igual que 1.

La gramática puede contener también la producción $S \rightarrow \varepsilon$ si el lenguaje que se quiere generar contiene la cadena vacía.

Los ejemplos mostrados a continuación corresponden a los lenguajes libres del contexto que fueron introducidos en los ejemplos 1, 2 y 3 del apunte de autómatas de pila.

Ejemplo 1:

La siguiente gramática genera las cadenas del lenguaje $L_1 = \{wcw^R / w \in \{a, b\}^*\}$

$G_1 = <\{A\}, \{a, b, c\}, P_1, S_1>$, y P_1 contiene las siguientes producciones

$S_1 \rightarrow A$

$A \rightarrow aAa$

$A \rightarrow bAb$

$A \rightarrow c$

Ejemplo 2:

La siguiente gramática genera las cadenas del lenguaje $L_2 = \{0^i 1^{i+k} 2^k 3^{n+1} / i, k, n \geq 0\}$

Casos	*Cadenas de L_2*
si n, i, k > 0	$0^i 1^{i+k} 2^k 3^{n+1}$
si n=0 y i, k > 0	$0^i 1^{i+k} 2^k 3$
si i=0 y n, k >0	$1^k 2^k 3^{n+1}$
si k=0 y n, i >0	$0^i 1^i 3^{n+1}$
si n, i=0 y k >0	$1^k 2^k 3$
si n, k=0 y i >0	$0^i 1^i 3$
si i, k =0 y n >0	3^{n+1}
si n,i,k=0	3

$G_2 = <\{A, B, C\}, \{0, 1, 2, 3\}, P_2, S_2>$, y P_2 contiene las producciones

$S_2 \rightarrow ABC$ $B \rightarrow 1B2$
$S_2 \rightarrow AC$ $B \rightarrow 12$
$S_2 \rightarrow BC$ $C \rightarrow 3C$
$S_2 \rightarrow C$ $C \rightarrow 3$
$A \rightarrow 0A1$
$A \rightarrow 01$

Ejemplo 3:
La siguiente gramática genera las cadenas del lenguaje $L_3 = \{h^n g^j e^{2n} d^{3i} / i, j, n \geq 0 \}$

Casos	*Cadenas de L_3*
si n, i, j > 0	$h^n g^j e^{2n} d^{3i}$
si n=0 y i, j > 0	$g^j d^{3i}$
si i=0 y n, j >0	$h^n g^j e^{2n}$
si j=0 y n, i >0	$h^n e^{2n} d^{3i}$
si n, i=0 y j >0	g^j
si n, j=0 y i >0	d^{3i}
si i, j =0 y n >0	$h^n e^{2n}$
si n,i,j=0	ε

$G_3 = <\{A, B, C\}, \{h, g, d, e\}, P_3, S_3>$, y P_3 contiene las producciones
$S_3 \rightarrow \varepsilon$ $A \rightarrow B$
$S_3 \rightarrow AC$ $B \rightarrow gB$
$S_3 \rightarrow A$ $B \rightarrow g$
$S_3 \rightarrow C$ $C \rightarrow dddC$
$A \rightarrow hAee$ $C \rightarrow ddd$
$A \rightarrow hee$

Ejemplo 4:
La siguiente gramática genera las cadenas de $L_4 = \{a^m b^p c^{p+m} / m, p \geq 1\} \cup \{a^i b^{2i} / i \geq 1 \}$
$G_4 = <\{A, B, C\}, \{a, b, c\}, P_4, S_4>$, y P_4 contiene las producciones
$S_4 \rightarrow A$
$S_4 \rightarrow C$
$A \rightarrow aAc$
$A \rightarrow aBc$
$B \rightarrow bBc$
$B \rightarrow bc$
$C \rightarrow aCbb$
$C \rightarrow abb$

BNF
Las gramáticas libres del contexto se escriben, frecuentemente, utilizando una notación conocida como BNF (Backus-Naur Form). BNF es la técnica más común para definir la sintaxis de los lenguajes de programación.
En esta notación se deben seguir las siguientes convenciones:
- los no terminales se escriben entre paréntesis angulares < >
- los terminales se representan con cadenas de caracteres sin paréntesis angulares

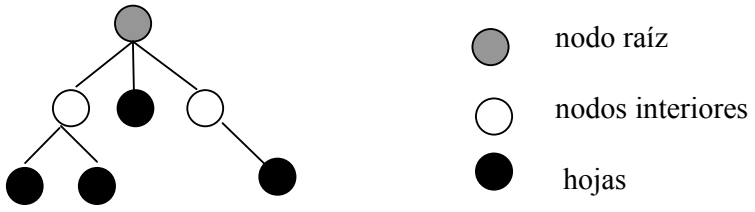
- el lado izquierdo de cada regla debe tener únicamente un no terminal (ya que es una gramática libre del contexto)
- el símbolo ::=, que se lee "se define como" o "se reescribe como", se utiliza en lugar de →
- varias producciones del tipo
  <A> ::= <B_1>
  <A> ::= <B_2>
  .
  .
  .
  <A> ::= <B_n>
  se pueden escribir como <A> ::= <B_1> | <B_2> | ... |<B_n>

Ejemplo 5:
La siguiente es una definición BNF del lenguaje que consiste de cadenas de paréntesis anidados:
<cadena_par> :: = <cadena_par> <paréntesis> |<paréntesis>
<paréntesis> ::= **(**<cadena_par> **)** | **( )**

Por ejemplo las cadenas ( ) (( )) y ( ) ( ) ( ) son cadenas válidas. En cambio las cadenas (( ) y ( ))) no pertenecen al lenguaje.

Arbol de derivación
Un árbol de derivación permite mostrar gráficamente cómo se puede derivar cualquier cadena de un lenguaje a partir del símbolo distinguido de una gramática que genera ese lenguaje.
Un árbol es un conjunto de puntos, llamados nodos, unidos por líneas, llamadas arcos. Un arco conecta dos nodos distintos. Para ser un árbol un conjunto de nodos y arcos debe satisfacer ciertas propiedades:
- hay un único nodo distinguido, llamado raíz (se dibuja en la parte superior) que no tiene arcos incidentes.
- todo nodo c excepto el nodo raíz está conectado con un arco a otro nodo k, llamado el padre de c (c es el hijo de k). El padre de un nodo, se dibuja por encima del nodo.
- todos los nodos están conectados al nodo raíz mediante un único camino.
- los nodos que no tienen hijos se denominan hojas, el resto de los nodos se denominan nodos interiores.

El árbol de derivación tiene las siguientes propiedades:
- el nodo raíz está rotulado con el símbolo distinguido de la gramática;
- cada hoja corresponde a un símbolo terminal o un símbolo no terminal;
- cada nodo interior corresponde a un símbolo no terminal.



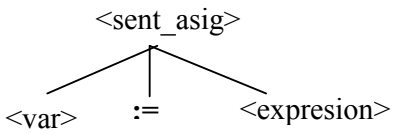
Para cada cadena del lenguaje generado por una gramática es posible construir (al menos) un árbol de derivación, en el cual cada hoja tiene como rótulo uno de los símbolos de la cadena.

Ejemplo 6:
La siguiente definición BNF describe la sintaxis (simplificada) de una sentencia de asignación de un lenguaje tipo Pascal:

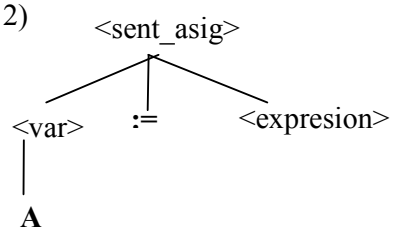
<sent_asig> ::= <var> **:=** <expresion>
<expresion> ::= <expresion> + <termino> | <expresion> - <termino> | <termino>
<termino> ::= <termino> * <factor> |<termino> / <factor> |<factor>
<factor> ::= (<expresion>) |<var> | <num>
<var> ::= A | B | C | D | ... | Z
<num> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Por ejemplo, la sentencia A := A + B es una sentencia de asignación que pertenece al lenguaje definido por la definición BNF dada, y cuyo árbol de derivación se construye como se muestra a continuación:

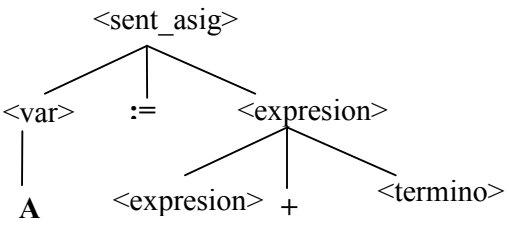
1)



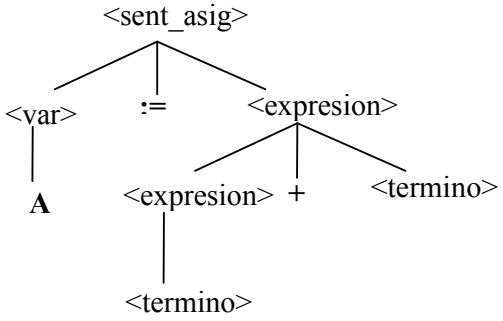
2)



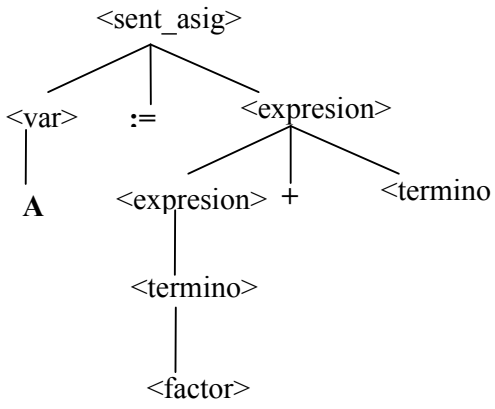
3)



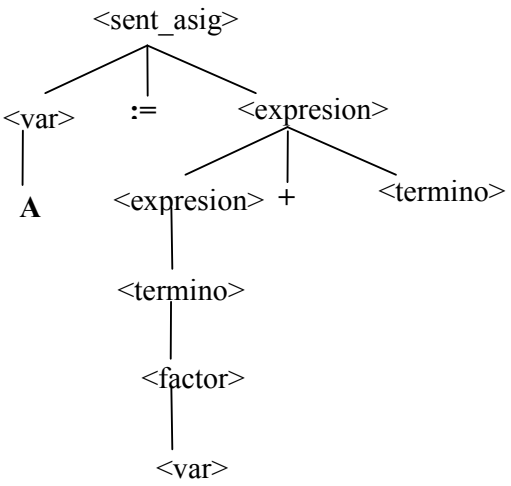
4)



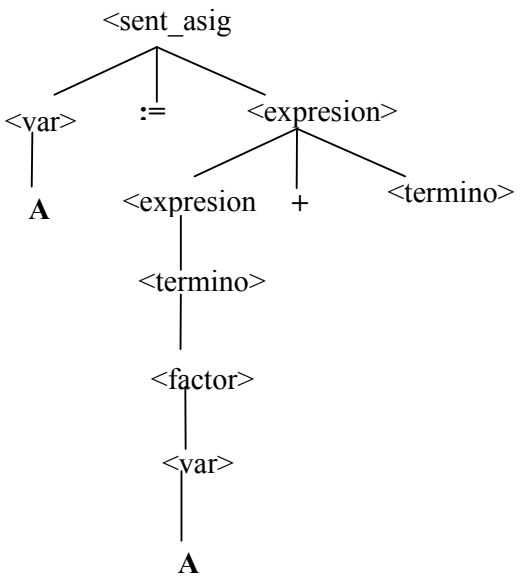
5)



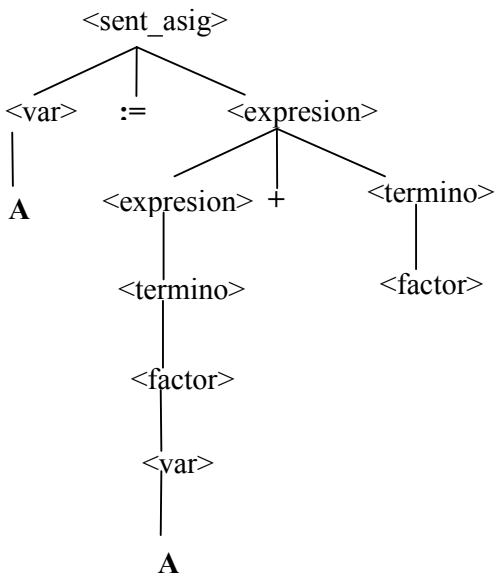
6)



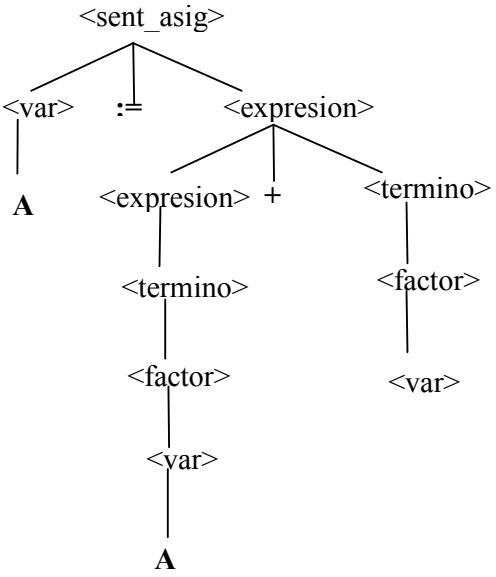
7)


<sent_asig
<var>
:=
<expresion>
A
<expresion
+
<termino>
<termino>
<factor>
<var>
A

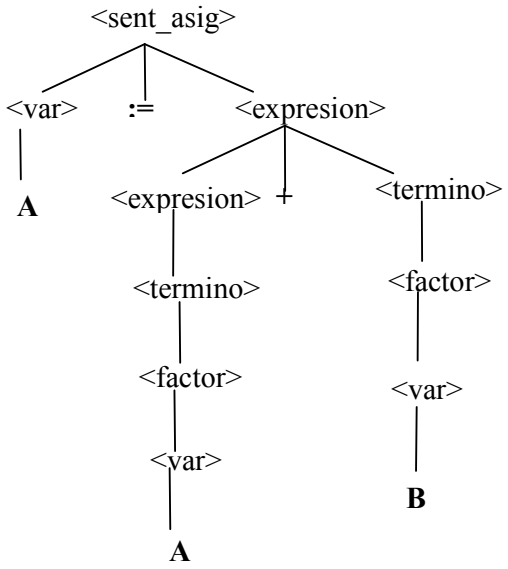
8)


<sent_asig>
<var>
:=
<expresion>
A
<expresion>
+
<termino>
<termino>
<factor>
<factor>
<var>
A

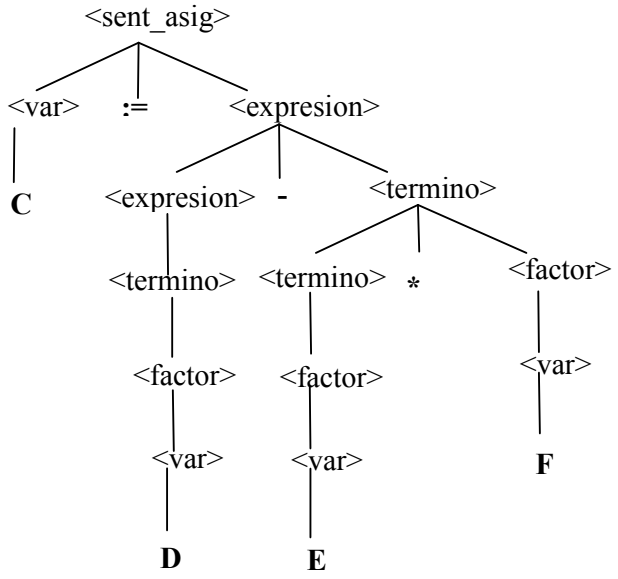
9)


<sent_asig>
<var>
:=
<expresion>
A
<expresion>
+
<termino>
<termino>
<factor>
<factor>
<var>
<var>
A

10)


<sent_asig>
<var>
:=
<expresion>
A
<expresion>
+
<termino>
<termino>
<factor>
<factor>
<var>
<var>
B
A

El árbol de derivación correspondiente a la sentencia C := D – E * F es el siguiente



Gramáticas ambiguas

Una gramática es ambigua si permite construir dos o más árboles de derivación distintos para la misma cadena. Por lo tanto, para demostrar que una gramática es ambigua lo único que se necesita es encontrar una cadena que tenga más de un árbol de derivación.

Una gramática en la cual, para toda cadena generada w, todas las derivaciones de w tienen el mismo árbol de derivación es no ambigua.

En algunos casos, dada una gramática ambigua, se puede encontrar otra gramática que produzca el mismo lenguaje pero que no sea ambigua.

Si todas las gramáticas independientes del contexto para un lenguaje son ambiguas, se dice que el lenguaje es un lenguaje independiente del contexto inherentemente ambiguo.

Por ejemplo, el lenguaje L = { $a^i b^j c^k$ / i = j ó j = k }

Ejemplo 7:

La siguiente es otra definición BNF simplificada para sentencias de asignación de un lenguaje tipo Pascal:

<sent_asig> ::= <var> := <expresión>

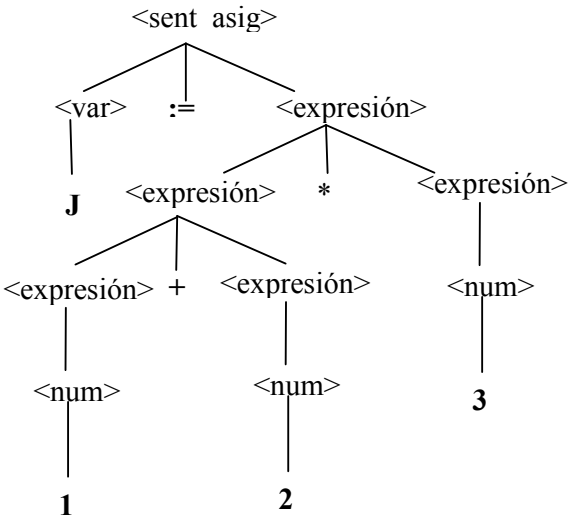
<expresión> ::= <expresión> + <expresión> | <expresión> - <expresión> | (<expresión>) |
<expresión> * <expresión> | <expresión> / <expresión> | <var> | <num>

<var> ::= A | B | C | D | ... | Z

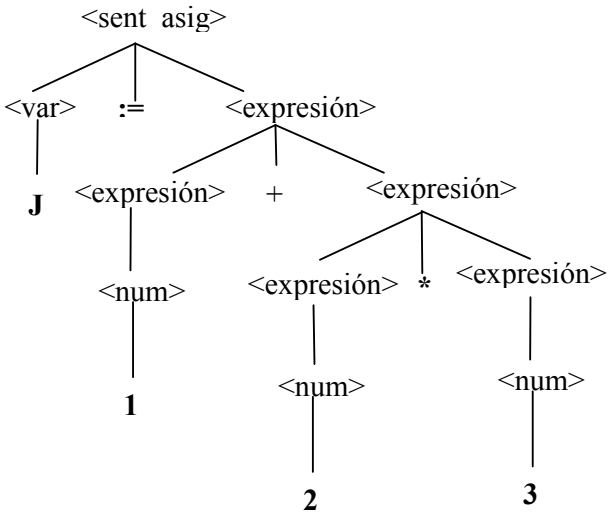
<num> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

La cadena J := 1 + 2 * 3 es una cadena de este lenguaje de sentencias de asignación. Se pueden construir dos árboles de derivación distintos para ella. Esto determina que la gramática es ambigua. Sin embargo, este lenguaje NO es inherentemente ambiguo ya que es posible construir una gramática no ambigua que lo genere (como se mostró en el Ejemplo 6).

Árbol 1



Árbol 2



Si se pretende determinar cómo se calcula el valor de la derecha del operador de asignación, se obtienen dos resultados posibles. El árbol 1 agrupa la expresión como (1 + 2) * 3, dando como resultado el valor incorrecto 9; esto se debe a que en este agrupamiento no se tiene en cuenta la precedencia de los operadores. El árbol 2 agrupa la expresión como 1 + (2 * 3), y da como resultado 7 que es el valor correcto.

Este ejemplo muestra que la ambigüedad puede ser un problema para ciertos lenguajes en los que su significado depende, en parte, de su estructura, como ocurre con los lenguajes naturales y los lenguajes de programación. Como una cadena que tiene más de un árbol de derivación suele tener más de un significado, para aplicaciones de compilación es necesario diseñar, cuando sea posible, gramáticas no ambiguas.