3. Análisis Sintáctico.

Tabla de Contenidos:

- Introducción:
- Desarrollo:
  - Gramáticas formales
  - Componentes de una GLC
  - Diferencias entre GLC y otras Gramáticas
    - > GLC vs. GR
    - > GLC vs GSC
  - Producción y derivación en GLC
    - > Reglas de producción en una GLC
  - Derivación
    - > Derivación por izquierda
    - > Derivación por derecha
  - Árboles de derivación
    - > Definición
    - > Relación entre árboles y una GLC
    - > Ejemplo
    - > Ambigüedad
- Conclusión
- Referencias

MONOGRAFÍA TÉCNICA: Gramáticas Libres de Contexto y Árboles de Derivación.

INTRODUCCIÓN:

Las gramáticas libres de contexto (GLC) son un concepto fundamental en la teoría de Lenguajes formales y son de gran importancia en diversas áreas de la computación, especialmente en el análisis y procesamiento de lenguajes. Estas gramáticas fueron introducidas por el lingüista Noam Chomsky en los 50's y se han convertido en una herramienta esencial para modelar la estructura sintáctica de lenguajes formales, incluidos los lenguajes de programación.

Las GLC se caracterizan por tener reglas de producción que permiten describir cómo se generan las cadenas en un lenguaje, utilizando símbolos terminales y no terminales. A diferencia de las gramáticas regulares, las GLC pueden describir lenguajes más complejos, lo que las convierte en un modelo adecuado para representar la sintaxis de lenguajes naturales y formales.

Una de las principales herramientas para visualizar las derivaciones en las GLC son los árboles de derivación, que representan gráficamente la estructura jerárquica de una cadena derivada en la gramática. Estos arboles no solo permiten entender cómo se generan las cadenas en un lenguaje, sino que también juegan un papel fundamental en el proceso de análisis sintáct-

ico, utilizado en compiladores y otras aplicaciones.

En esta monografía, exploraremos a profundidad el concepto de las gramáticas libres de contexto, sus características, aplicaciones y el uso de árboles de derivación.

DESARROLLO

Gramáticas Formales:

Para entender el concepto de GLC, primero debemos recordar que una gramática formal es un conjunto de reglas que definen cómo se pueden formar las cadenas de un lenguaje. Estas gramáticas proporcionan un método para describir la estructura sintáctica de un lenguaje, ya sea natural (como el lenguaje humano) o formal (como los lenguajes de programación). Las gramáticas formales están compuestas por símbolos y reglas de producción que determinan cómo estos símbolos pueden combinarse para formar cadenas válidas dentro de un lenguaje.

Las gramáticas formales se clasifican en diferentes tipos, de acuerdo con el nivel de complejidad y el tipo de reglas que emplean. Esta clasificación se conoce como la jerarquía de Chomsky, y las gramáticas se dividen en:

- Tipo 0 (gramáticas recursivamente enumerables): Máq. Turing
- Tipo 1 (gramáticas sensibles al contexto): Autómata Linealmente acotado
- Tipo 2 (gramáticas libres de contexto): Autómata con Pila
- Tipo 3 (gramáticas regulares): Autómata finito

Las gramáticas libres de contexto (GLC) pertenecen al tipo 2 de esta jerarquía y son particularmente importantes por que permiten

describir la mayoría de los lenguajes que se utilizan en programación y otras áreas de la informática.

Componentes de una GLC:

Una gramática libre de contexto se define formalmente como una cuádrupla:

$$G = (V, \Sigma, P, S)$$

Donde:

- V es el conjunto de símbolos no terminales
- Σ es el conjunto de símbolos terminales
- P es el conjunto de reglas de producción
- S es el símbolo inicial

Las reglas de producción en una GLC tienen la siguiente forma:

$$A \rightarrow \alpha \qquad (A \text{ deriva en alfa})$$

Donde:

- A es un símbolo No terminal (V)
- α es una cadena de símbolos terminales y/o no terminales (Σ y V).

Una característica fundamental de las GLC es que el lado izquierdo de una regla de producción es un único símbolo no terminal, lo que lo distingue de otros tipos de gramáticas más complejas.

Diferencias entre GLC y Otras Gramáticas.

• GLC vs GR:

Las gramáticas regulares (tipo 3) son un subconjunto más restringido que las GLC. Estas gramáticas solo permiten reglas de producción donde el símbolo no terminal se encuentra en una posición fija en la cadena (al inicio o al final). Las gramáticas regulares son más simples y solo pueden describir lenguajes que tienen una estructura muy sencilla, como los lenguajes reconocidos por autómatos finitos.

En cambio, las gramáticas libres de contexto son más expresivas y permiten generar estructuras sintácticas más complejas. Por ejemplo, los lenguajes que permiten el uso de paréntesis anidados o el análisis de expresiones aritméticas son ejemplos de lenguajes que no pueden ser descritos por gramáticas regulares, pero sí por gramáticas libres de contexto.

• GLC vs GSC:

Las gramáticas sensibles al contexto (tipo 1) son más poderosas que las GLC, ya que permiten que las reglas de producción dependan del contexto en el que se encuentra un símbolo. Es decir, las reglas pueden tener en cuenta los símbolos que rodean a un símbolo no terminal. Las GLC no tienen esta dependencia, las reglas de producción se aplican sin importar el contexto.

Aunque las GSC pueden describir lenguajes más complejos que las GLC, son menos utilizadas en la práctica debido a su mayor complejidad y a la dificultad para analizarlas eficientemente, se podría decir que las GLC son un tipo intermedio entre complejidad y capacidad descriptiva.

Producción y Derivación en GLC:

Reglas de producción en una GLC:

Las reglas de producción son la base de las gramáticas libres de contexto (GLC). Estas reglas describen cómo se pueden transformar o expandir los símbolos terminales y no terminales, con el objetivo de generar todas las cadenas válidas que pertenecen al lenguaje definido por la gramática.

En una GLC, una regla de producción tiene la forma:

$$\boxed{A \rightarrow \alpha}$$

Donde:
- A = No terminal (V)
- α = $(\Sigma \cup V)$

Una característica importante de las GLC es que el lado izquierdo de cualquier regla de producción siempre es un solo símbolo no terminal, independientemente del contexto en el que se encuentre. Esto es lo que define la propiedad "libre de contexto": las reglas pueden aplicarse sin depender del entorno de los símbolos.

Por ejemplo: La siguiente es una gramática libre de contexto para expresiones enteras algebraicas sintácticamente correctas sobre las variables x, y, z.

$$S \rightarrow x \mid y \mid z \mid S+S \mid S-S \mid S*S \mid S/S \mid (S)$$

Generaría por ejemplo:

$$(x+y)*x-z*y/(x+x)$$

Derivación

La derivación en una GLC es el proceso de aplicar las reglas de producción de manera sucesiva, comenzando desde el símbolo inicial, para generar una cadena de símbolos terminales. Durante este proceso, se selecciona un símbolo no terminal y se reemplaza por el lado derecho de una regla de producción aplicable.

Existen 2 tipos principales de derivación:

- Derivación izquierda:

  Se selecciona siempre el símbolo no terminal más a la izquierda en la cadena para aplicar una regla de producción. El proceso continúa reemplazando ese símbolo por una cadena según las reglas de producción, y luego se repite para el siguiente no terminal más a la izquierda. Por ejemplo, considerando la siguiente gramática:

(1) $S \to S + S$

(2) $S \to 1$

y queremos derivar la cadena:

$1 + 1 + 1$

① S — Símbolo inicial

② $S + S$ — regla (1) en el único S (más a la izq)

③ $S + S + S$ — Aplicar (1) en la S más a la izq.

④ $1 + S + S$ — Aplicar Regla (2) en la S más a la izq

⑤ $1 + 1 + S$ — repetir

⑥ $1 + 1 + 1$ — repetir

En este caso, la derivación por izq. sigue el orden de las reglas [(1), (1), (2), (2), (2)]

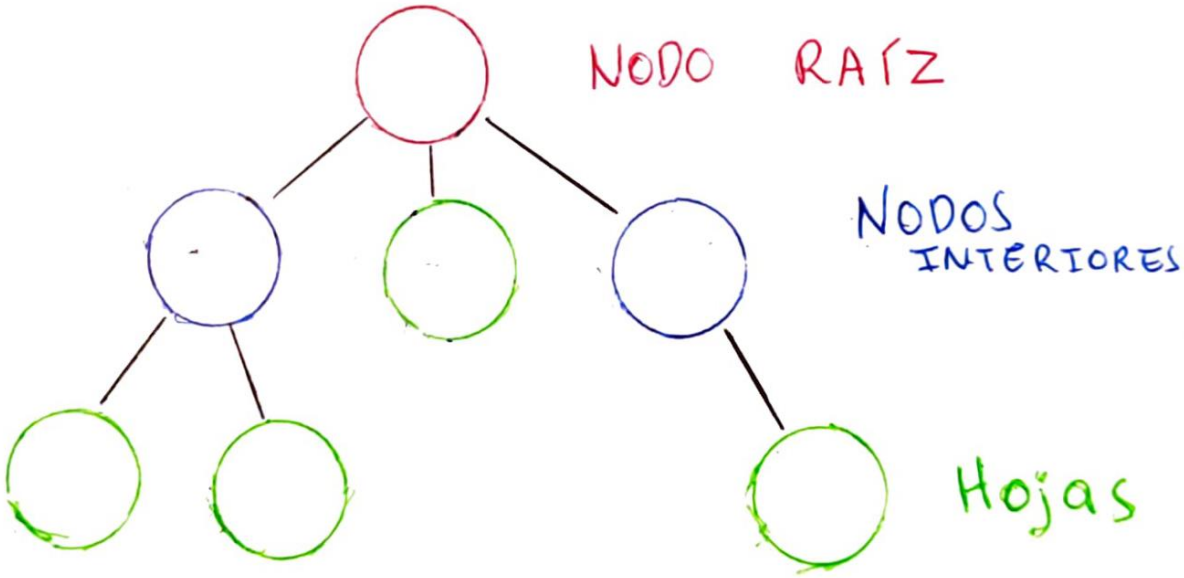
- Derivación Derecha:

Se selecciona siempre el no terminal más a la derecha en la cadena para aplicar una regla de producción. El proceso es igual a la derivación por izquierda, pero con el orden invertido. A este tipo de derivación también se le conoce como derivación canónica.

Árboles de Derivación:

Definición de Árboles de derivación:

Un árbol de derivación es una representación gráfica de la estructura sintáctica de una cadena generada por una GLC.



En este árbol, los nodos representan los símbolos (terminales y/o no terminales), y las ramas muestran cómo se aplican las reglas de producción de la gramática para generar la cadena final. El nodo raíz es el símbolo inicial y las hojas representan los símbolos terminales que forman la cadena derivada.

Para cada cadena del lenguaje generado por una gramática es posible construir (al menos) un árbol de derivación. Los árboles de derivación permiten visualizar de forma clara la forma en que una GLC genera una cadena específica, desglosando las derivaciones en pasos jerárquicos.

Relación entre árboles de Derivación y una GLC

El proceso de derivación en una GLC, ya sea por izquierda o por derecha, se puede visualizar en forma de árbol. Cada paso de la derivación, donde se reemplaza un símbolo no terminal por una secuencia de símbolos terminales y/o no terminales, corresponde a una expansión de un nodo en el árbol. El nodo que representa el símbolo no terminal se expande en una serie de hijos, que corresponden a los símbolos en el lado derecho de la regla de producción aplicada.

Ejemplo:

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>

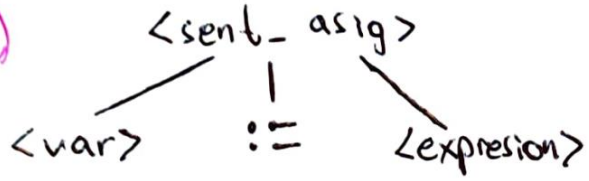
<termino> ::= | <termino> * <factor> | <termino> / <factor> | <factor>

<factor> ::= (<expresion>) | <var> | <num>

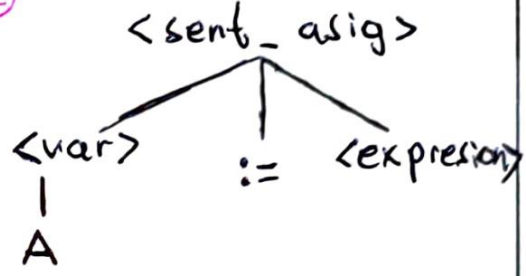
<var> ::= A|B|C|D|...|Z <num> ::= 0|1|2|3|...|9

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:

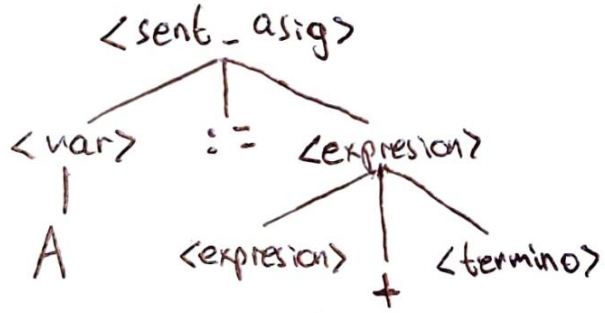
1



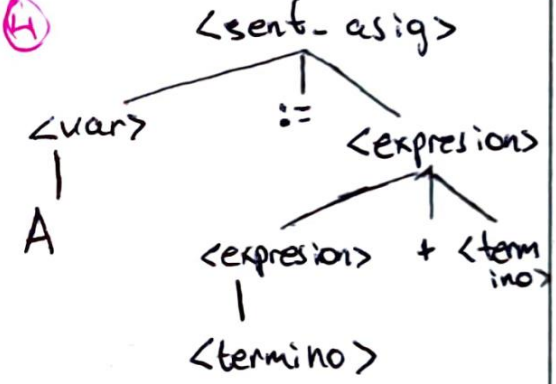
2



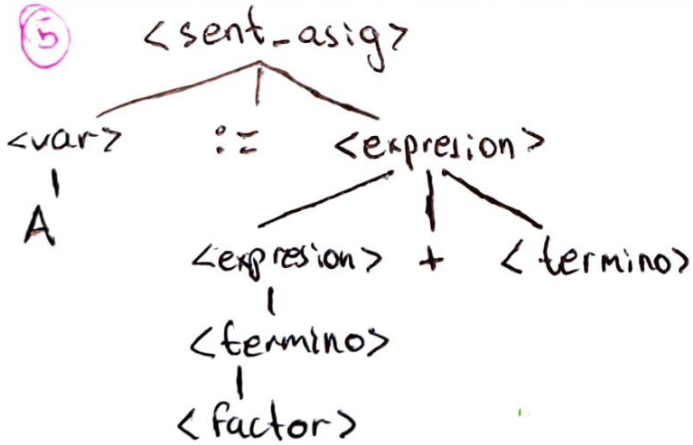
3



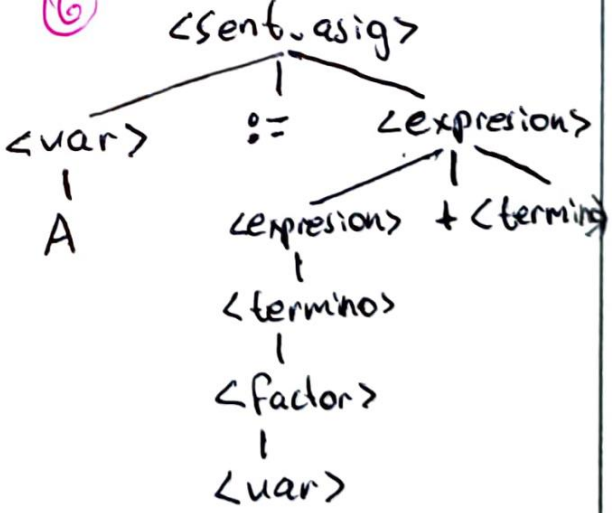
4



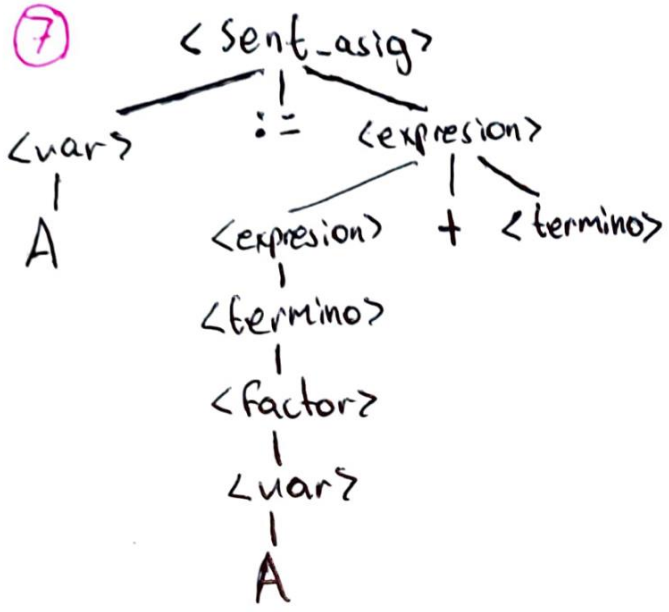
5



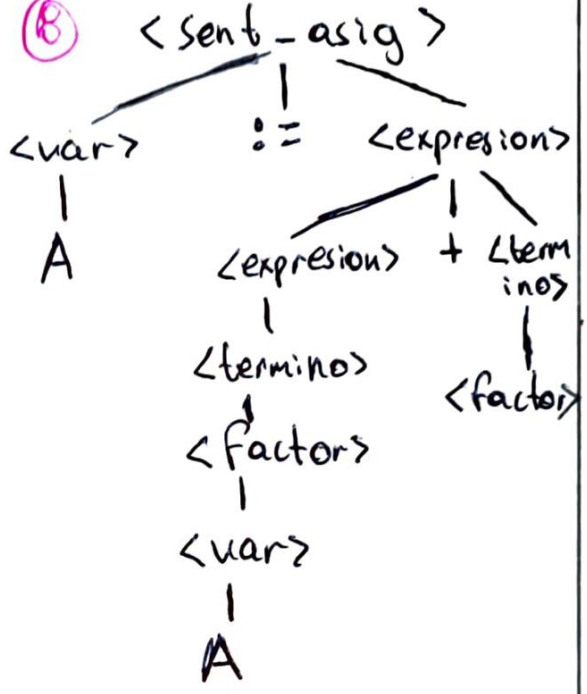
6



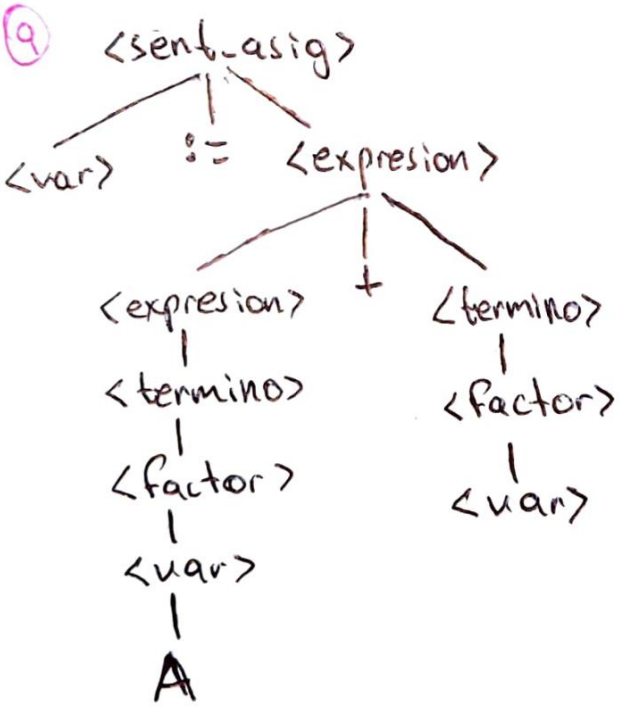
7


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

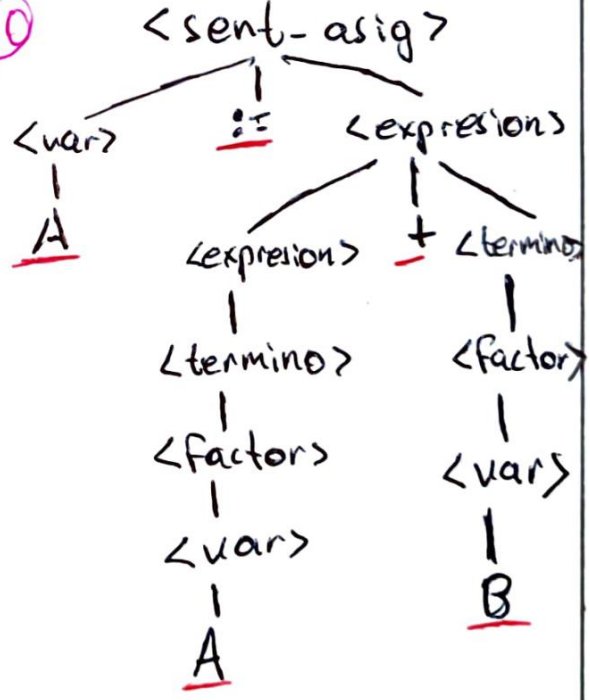
8


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

9


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

10


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

Ambigüedad:

Una gramática es ambigua si una misma cadena puede ser derivada de más de una manera, es decir, si existen múltiples árboles de derivación de una misma cadena. La ambigüedad no está permitida y es un problema crítico en el diseño de lenguajes de programación, ya que pueden generar diferentes interpretaciones para una expresión, afectando el análisis y ejecución de un programa. Por ejemplo:

Dada la gramática:

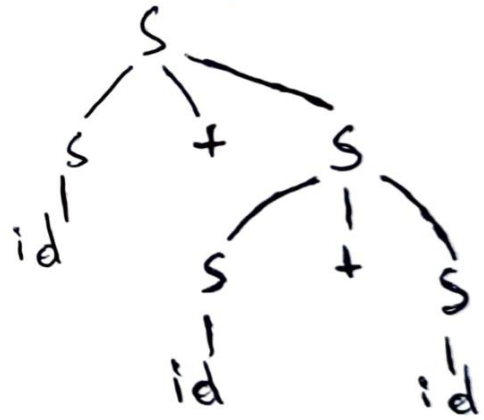
(1) $S \to S + S$

(2) $S \to id$

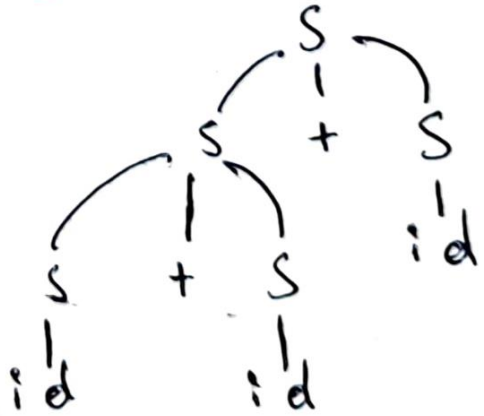
y derivamos la cadena id+id+id

Los árboles de derivación posibles son:

① Por izq.



② Por der.



La ambigüedad en esta gramática surge porque no especifica cómo se deben agrupar las operaciones de suma. En el primer árbol, la interpretación sería (id + id) + id, mientras el segundo sería id + (id + id). Esto puede generar problemas en un lenguaje de programación.

CONCLUSIÓN

En esta monografía se ha aprendido cómo las gramáticas libres de contexto (GLC) son esenciales para entender la estructura y formación de los lenguajes de programación, tanto en el ámbito teórico como práctico.

Las GLC nos permiten entender cómo se generan las oraciones dentro de un lenguaje a través de reglas de producción claras y precisas. Además, los árboles de derivación demostraron ser una gran herramienta visual que ayuda a comprender de manera estructurada el proceso de derivación, facilitando la identificación de posibles ambigüedades en las gramáticas.

Estos conceptos no solo fortalecen nuestro conocimiento de la teoría de lenguajes formales, sino que también tienen aplicaciones directas en la creación de lenguajes de programación, por lo que para nosotros como estudiantes de sistemas es de vital importancia dominar estos temas.

REFERENCIAS:

- https://www.cs.buap.mx/~mtovar/doc/LFA/GLC.pdf
- https://ccc.inaoep.mx/~emorales/Cursos/Automatas/GramsLibresContexto.pdf.
- https://users.exa.unicen.edu.ar/catedras/ccompl/Apunte5.pdf
- https://es.wikipedia.org/wiki/Gramatica_libre_de_contexto.
- https://www.youtube.com/watch?v=UU-j2_xyFU
- https://es.wikipedia/wiki/jerarquia_de_Chomsky.

Introducción

El presente trabajo es acerca de los diagramas de sintaxis, los cuales son un diagrama de flujo que indica la secuencia que puede seguir una gramática y poder construir adecuadamente las frases.

Para ello se ondará en las componentes de los diagramas, que son los nodos y las direcciones, en el como se leen y otras características.

Así mismo, también se toma en cuenta su estrecha relación con las reglas de producción y el como ambos desempeñan un rol importante en el desarrollo y construcción de un lenguaje.

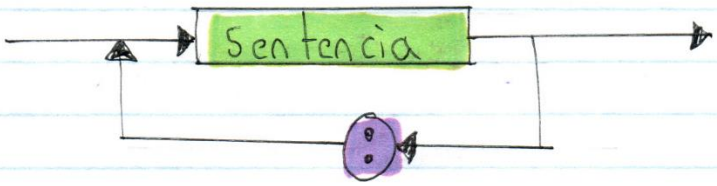
Para ello se dibujaran multitud de diagrama inspirados en distintas reglas de producción tanto de las matemáticas como de algunos lenguajes de programación.

Índice

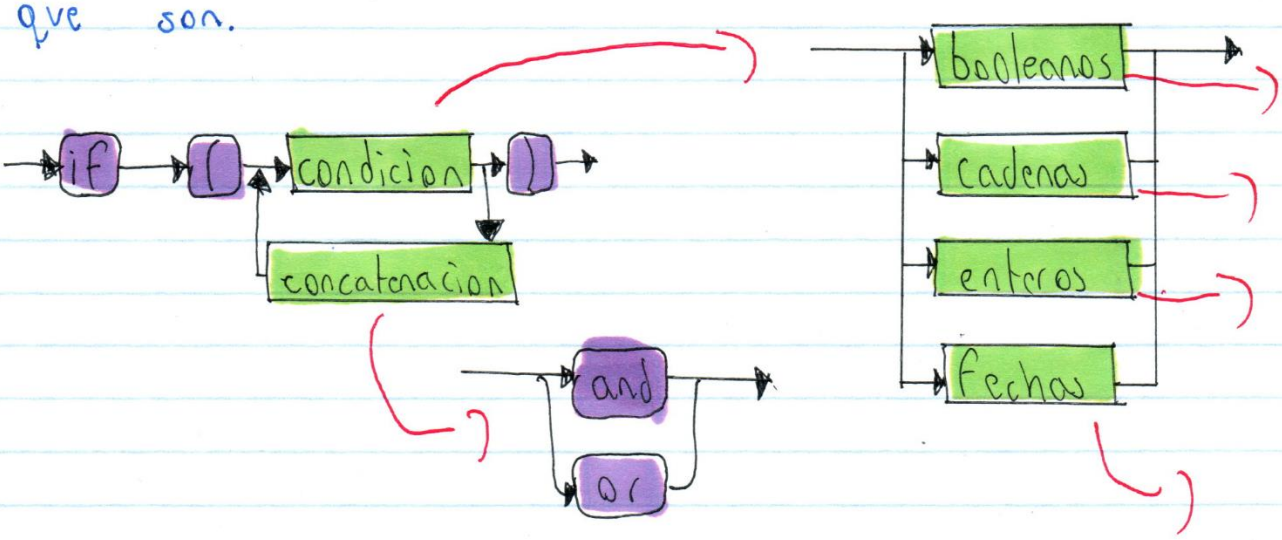
Definición
Características
Para que sirven
Como crearlos

Diagramas de sintaxis

Los diagramas de sintaxis son un grafo direccionado en el que los nodos que contiene representan los símbolos tanto terminales como no terminales y los arcos indican la secuencia que se puede tomar en estos para poder formar frases que sean válidas para la gramática.

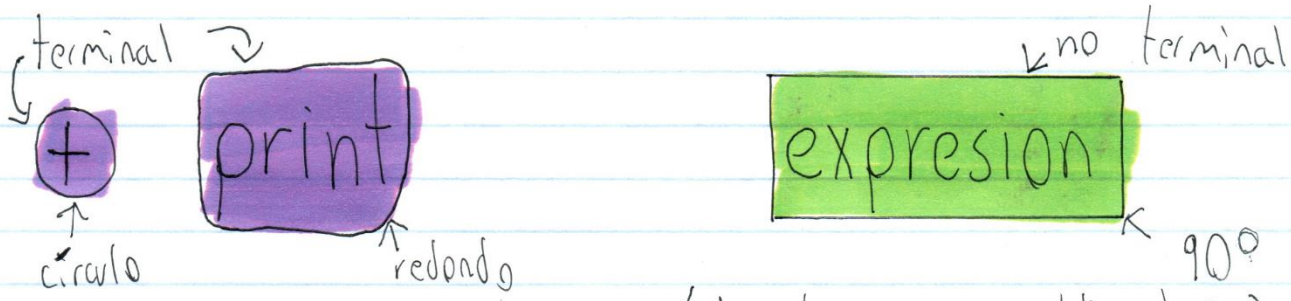


Cada uno de estos es la representación de un símbolo no terminal, el cual se puede expandir siempre que las reglas de producción lo permitan, con lo cual, una gramática completa será conformada por una multitud de estos diagramas y podrán estar interconectados como los símbolos no terminales que son.



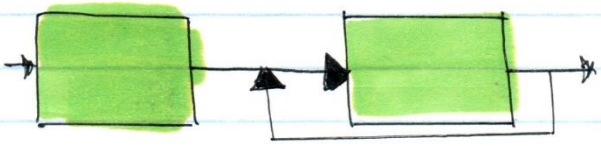
Ahora revisaremos las partes de estos diagramas, que son los nodos y los arcos.

Los nodos representan ambos tipos de símbolos; los terminales se dibujan en formas redondas, ya sean círculos, elipses o rectángulos con las puntas redondas. Mientras tanto, los no terminales se dibujan en rectángulos con las puntas de 90°.



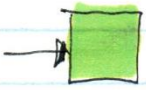
(el color no es obligatorio)

Los arcos indican el sentido del diagrama, aunque este suele ser de izquierda a derecha, y siempre que el sentido y dirección de los arcos esté bien definido, indica la validez del diagrama. Adicionalmente estos tienen un inicio y un fin, pero se caracterizan por no tener origen y destino, respectivamente



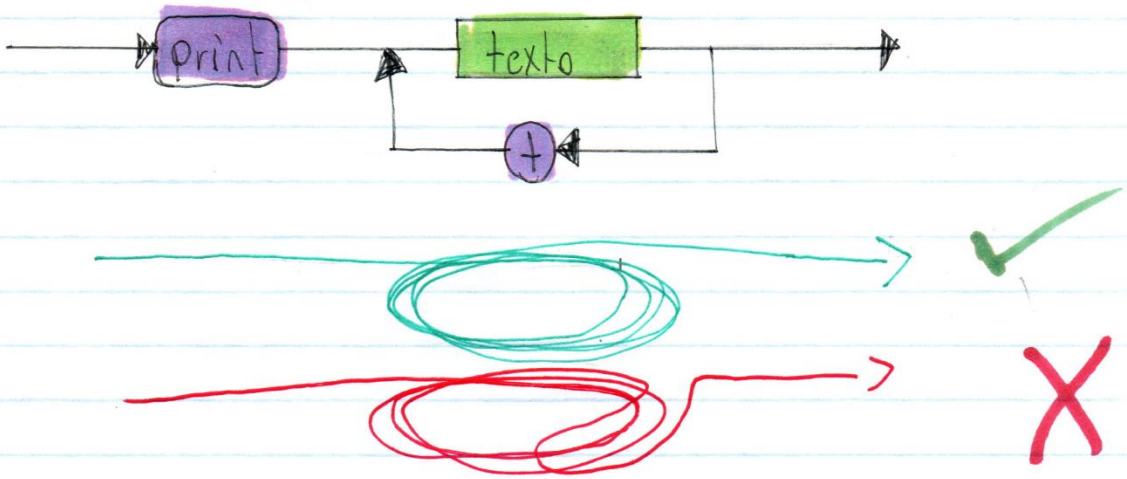
indica dirección
permiten hacer ciclos

origen -> puede venir de cualquier lado

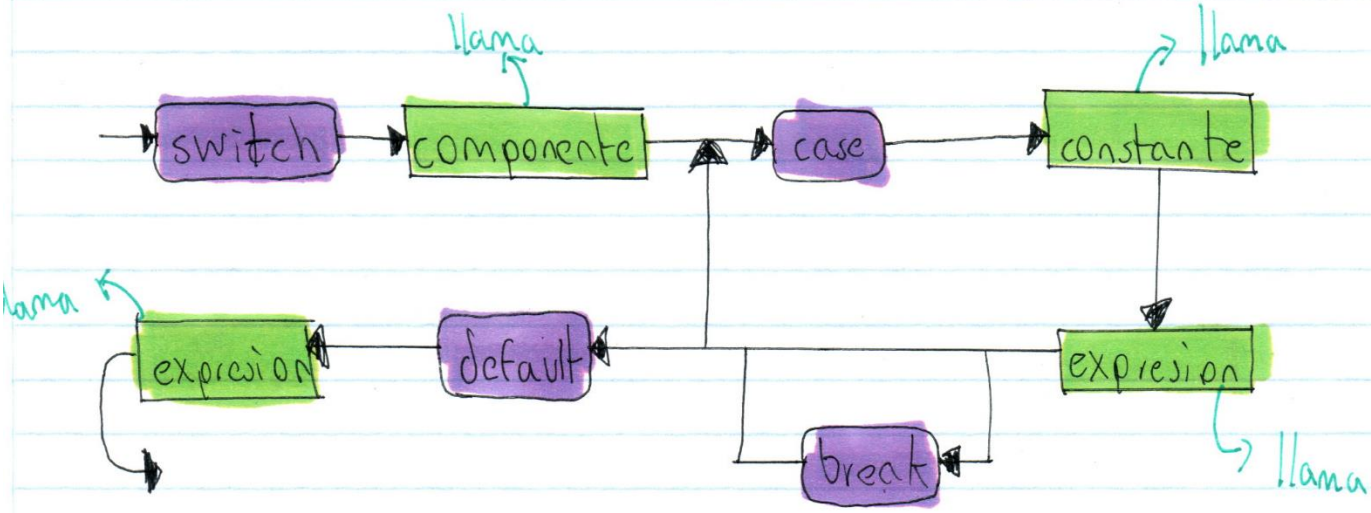


destino -> puede ir a cualquier lado

Con todo lo anterior se tiene que para reconocer una frase se debe partir del punto de entrada y seguir hasta la salida, solamente entonces la frase será reconocida, en caso contrario, será denegada.

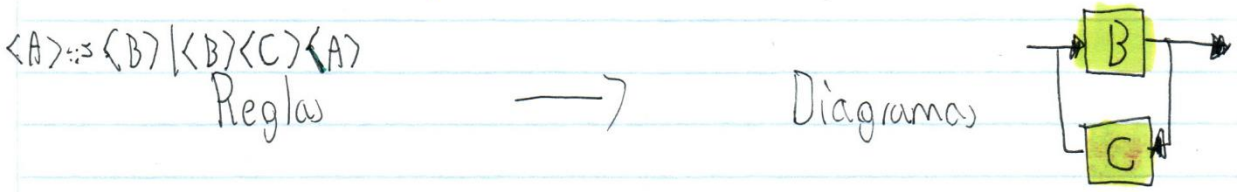


Y cuando en el recorrido se encuentra, se viaja a estos de forma recursiva las veces que sea necesario. Debido a ello un diagrama puede expandir N diagramas.

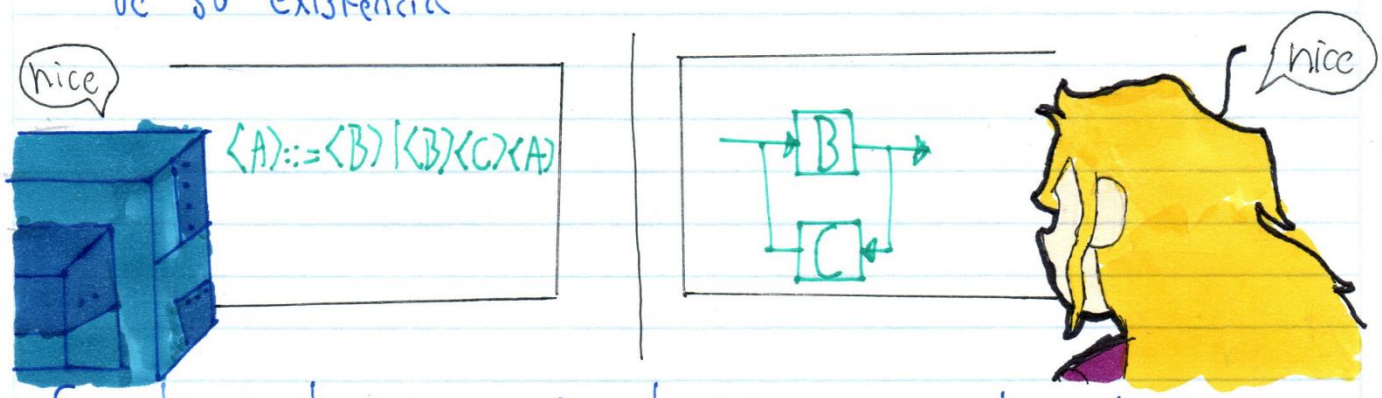


¿Para que sirven?

Como todo lenguaje, para poder darle un uso hay que saber que es lo que se está representando, y en este caso es una alternativa a la notación de las reglas de producción, como BNF, por ejemplo.



Las reglas en las notaciones tradicionales se componen de texto, por lo que el compilador/intérprete los comprenderá.
¿Y los diagramas de sintaxis? Para que el ser humano pueda comprender más fácilmente las reglas del lenguaje. Esa es la principal razón de su existencia



Con lo cual, se puede traducir una regla de producción a diagrama y viceversa.

¿Cómo crearlos?

Para usar bien esta herramienta hay que preguntarnos ¿qué fue primero, la gallina o el huevo?, aunque en este caso sería las reglas de producción o los diagramas de sintaxis.

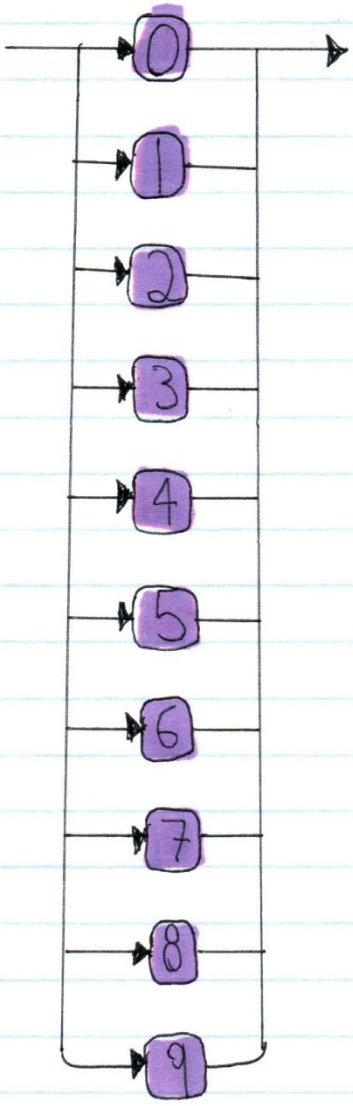
Y la respuesta es: cualquiera de los dos, ya que depende meramente del desarrollador, por lo que hay que ver cómo hacer reglas a partir de los diagramas y viceversa.

Primero		Luego
Diagrama	→	Regla
Regla	→	Diagrama

Todo esto ya que si bien el código usará únicamente las reglas, una documenta si que se puede ver ayudada por estos diagramas

Empezaremos por las reglas, y usaremos esta como ejemplo, donde solamente tenemos un digito con sus 10 posibles resultados. En este caso solamente ponemos el origen y desplegamos los números

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

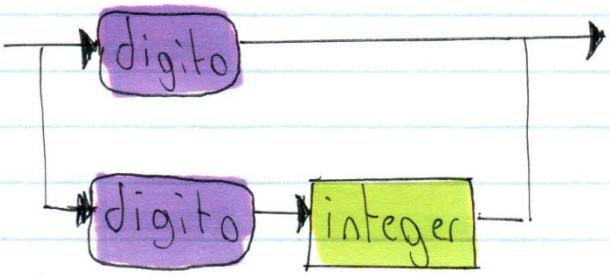


Con esto podemos ver que del origen pasa por alguno de los 10 digitos y sale del diagrama.

Aunque este por si mismo no nos sirve del todo, por lo que lo usaremos como estado no final de otra regla.

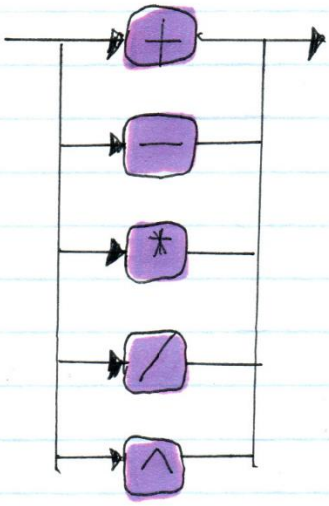
<integer> ::= <digito> | <digito><integer>

Este se puede hacer de dos formas, recursivo o con un ciclo, ambos para indicar que se repetirá múltiples veces.



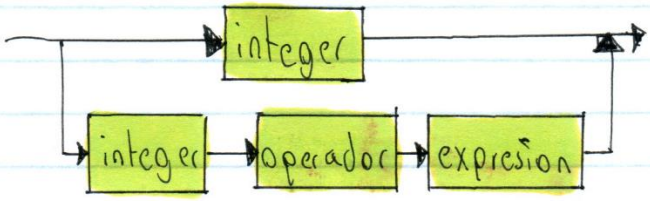
Con esto ya se tiene una expresión que puede determinar números enteros. Ahora bien, para los operadores se hace algo similar a los dígitos.

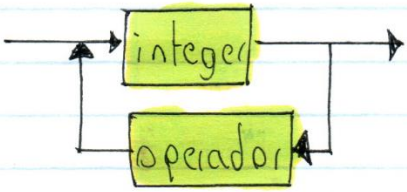
<operador>::= + | - | * | / | ^



Y la cereza sobre el pastel será una regla que haga expresiones aritméticas, usando los diagramas construidos anteriormente.

<expresion>::= <integer> | <integer><operador><expresion>





Estos diagramas pueden validar todo tipo de expresiones, como, por ejemplo:

1 ✓
1 + 1 ✓
30666 ✓
123/14 ✓
342 + 27 * 48 ✓
35 + 35 * 35/35 - 35 ✓

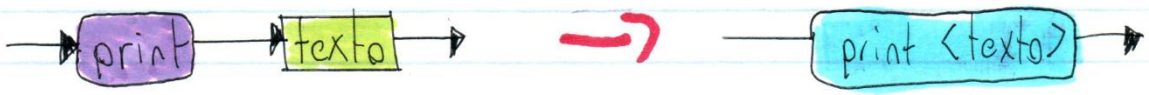
434 + ✗
* 876 ✗
123 + 123 - 432 * ✗
* ✗
/ ✗
+ ✗

Nota: en estos casos el llamado recursivo si que parece inútil, pero en expresiones más complejas si que es útil.

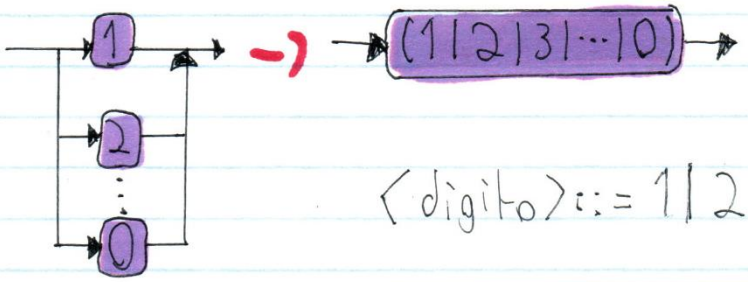
Nota 2: la regla de las expresiones tampoco es perfecta, pero es un buen ejemplo.

Ahora para convertir un diagrama a regla, estos se pueden compactar para acercarse a la regla original.

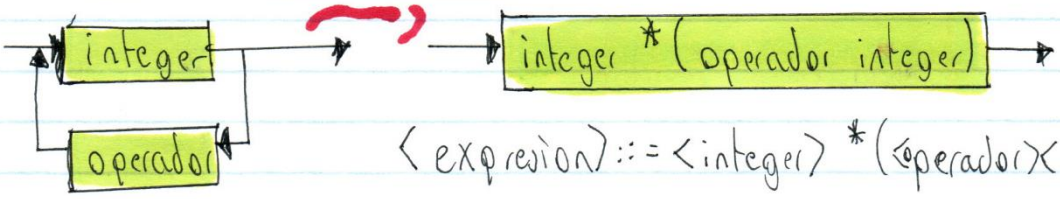
Las reducciones consisten en agrupar las diversas componentes para evitar redundancias, y en lugar de usar flechas, se usan las reglas de las notaciones, como si fueran un punto intermedio.



<imprimir>::= print <texto>



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



<expresion>::= <integer> *(<operador><integer>)

<expresion>::= <integer> | <integer><operador><expresion>

Estos son solo ejemplos simples, pero que permiten ver la utilidad de esta herramienta.

Conclusión

En conclusión, los diagramas de sintaxis son una versión más bonita de las reglas de producción en notaciones como BNF o Shorthand, y son hechas con el único motivo de ser comprendidas más facilmente por los seres humanos.

La principal función es ofrecer una versión gráfica de dichas reglas para comprender su funcionamiento de forma más básica y aumenta el público que pueda comprender las reglas.

También es de suma importancia saber como leerlos y aún más importante, saber traducirlos, tanto si haces primero el diagrama como si haces primero la regla, como si lo haces al revés.

Lo importante es comprender adecuadamente las virtudes que proveen los diagramas y saber usarlos adecuadamente en conjunto con las reglas de producción y hacer que estas puedan comprender de forma más sencilla con el proposito de que sea más claro.

Fuentes

Yatish Parmar (2016, 24 de enero). Syntax diagrams - A Level Computer Science. Youtube.
https://www.youtube.com/watch?v=5Rwd6jumo1I

Sergio G. David T. Antonio G. Antonio L. (Recuperado el 2024, 15 de septiembre). Generación completa de compiladores mediante diagramas de sintaxis extendidos. Universidad de Málaga.
http://info.iaia.lcc.uma.es/lcc/publicaciones/LCC829.pdf

Anónimo. (2024, 31 de enero). Diagramas de sintaxis. IBM.
https://www.ibm.com/docs/es/ibm-mq/9.2?topic=reference-syntax-diagrams

CARRERA	NOMBRE DE LA ASIGNATURA
Ingeniería en sistemas computacionales	Lenguajes y autómatas II

DOCENTE DESIGNADO
ISC. RICARDO GONZÁLEZ GONZÁLEZ

PRACTICA No.	NOMBRE DE LA PRACTICA	DURACIÓN (HORAS)
2	Gramáticas libres de contexto 15 de septiembre de 2024	3

1 INTRODUCCIÓN

En el presente trabajo consiste en definir un poco mas a detalle que son las gramáticas libres de contexto, para lo cual se harán árboles de derivación con un conjunto de reglas de producción definidas de antemano.

El propósito de esto es definir en mayor profundidad que son las gramáticas libres de contexto, comparándolas con las gramáticas regulares y haciendo énfasis en el contexto y el funcionamiento que cumple dentro de la definición de un lenguaje.

2 OBJETIVO (COMPETENCIAS)

General

- Definir mediante ejemplos que son las gramáticas libres de contexto

Específicos

- Desarrollar o buscar un conjunto de reglas de producción de una gramática libre de contexto
- Desarrollar árboles de derivación usando unas reglas de producción ya establecidas.
- Definir que es un contexto
- Indicar la función del contexto

3	MARCO TEÓRICO REFERENCIAL

- ✓ Gramáticas de programación: Conjunto finito de reglas que describen la secuencia que se puede llevar a cabo con los símbolos que pertenecen al lenguaje, con el fin de generar sentencias validas.
- ✓ Compilador: Programa que verifica y traduce código de medio-alto nivel para ser compilado, y con ello poder ser optimizado y comprendido para el dispositivo en cuestión.
- ✓ Gramáticas libres al contexto: Es un tipo de gramática en la que las reglas de producción se le pueden aplicar al símbolo no terminal siempre que se quiera.
- ✓ Sintaxis: Conjunto de reglas que sirven para dar estructura a un lenguaje, con lo cual estas tendrán control sobre los símbolos y las palabras reservadas.
- ✓ Léxico: Conjunto de reglas para definir el diccionario de palabras reservadas de un lenguaje, así como definir las nuevas a medida que se vaya necesitando
- ✓ BNF: notación usada para describir la sintaxis de los lenguajes formales. También puede definirse como una metasintaxis para definir gramáticas libres de contexto.

4	MATERIALES UTILIZADO

- Material acerca de las gramáticas libres de contexto
- Herramienta para hacer árboles de derivación, en este caso la utilizada fue jsSyntaxTree de la página http://ironcreek.net/syntaxtree/
- Útiles de escritura

5	REQUISITOS BÁSICOS.

Contar con un conjunto de reglas de producción
Saber controlar el software jsSyntaxTree
Comprender el funcionamiento de los árboles de derivación

6	DESARROLLO DE LA PRÁCTICA (PASO A PASO)

Los árboles de derivación representan la estructura sintáctica de una gramática, por lo que, para poder hacer los ejercicios primero hay que tener un conjunto de reglas.

Las que encontré son para generar expresiones matemáticas dado que no son ejemplos muy complejos y son sencillos de entender,

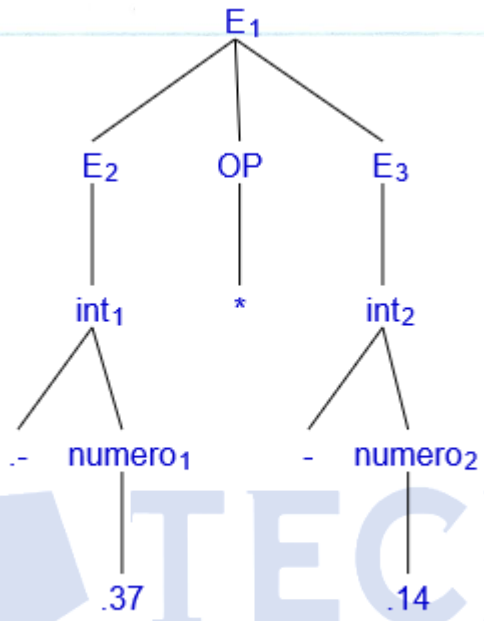
La notación en cuestión es la siguiente:

<expresion> ::=

<expresion> ::= <int>
<expresion> ::= <expresion> <operador> <expresion>
<expresion> ::= (<expresion>)
<operador> ::= +
<operador> ::= -
<operador> ::= *
<operador> ::= /
<operador> ::= ^
<int> ::= - <numero>
<int> ::= <numero>
<numero> ::= (0|1|2|3|4|5|6|7|8|9) <numero>
<numero> ::= (0|1|2|3|4|5|6|7|8|9)

Una operación sencilla

-37 * - 14



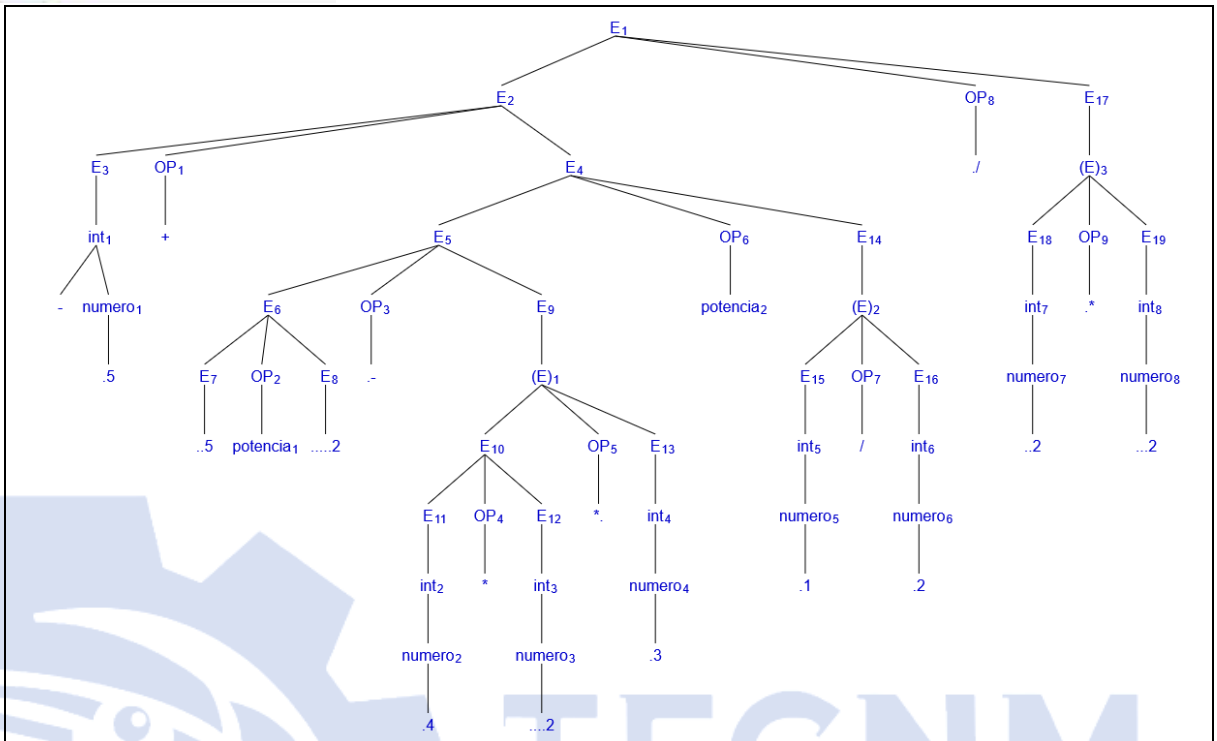
Formula general $\frac{-b \pm \sqrt{b^2-4ac}}{2a}$

cuando el +

a = 2
b = 5
c = 3

$\frac{-5 + \sqrt{5^2 - 4*2*3}}{2*2}$

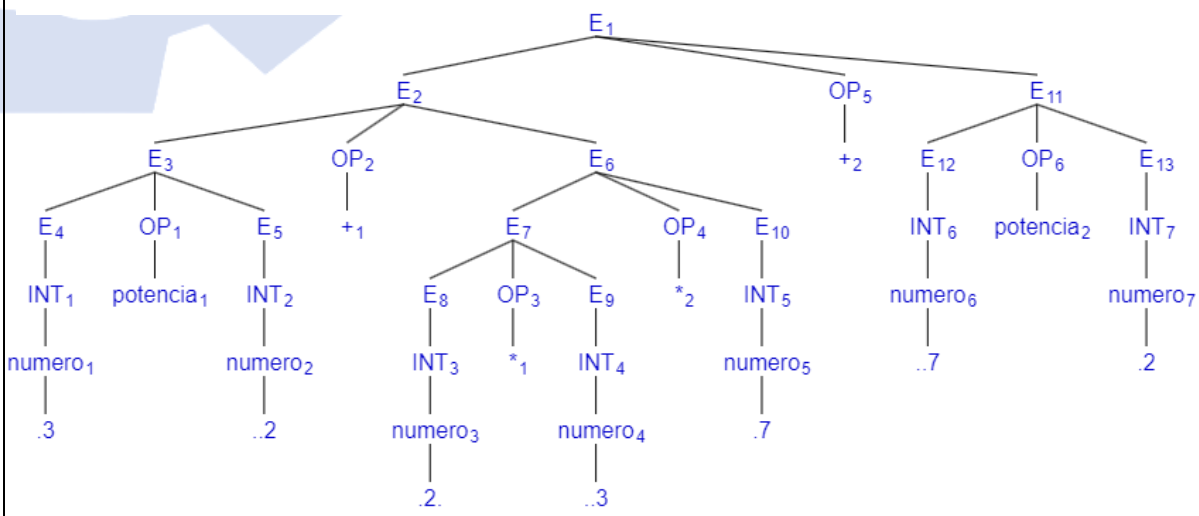
con ello quedaría



El último que me pareció interesante representar es el desarrollo del mítico binomio al cuadrado, el cual llega a ser bastante utilizado en materias de matemática.

$$a^2 + 2ab + b^2$$

tomando en cuenta $a = 3$, $b = 7$



Estos tres ejemplos nos permiten ver el como se usan los árboles cuando se encuentran las reglas adecuadas

Para definir a detalle que es el contexto me pareció adecuado comparar las gramáticas regulares y las libres de contexto.

	Gramática Regular	Gramática Libre de Contexto
Componentes	(V, T, P, S) V -> No terminales T -> Terminales P -> Reglas de producción S -> Símbolo de inicio	
Derivación	Muy malo	Excelente
Autómata	Finito determinista	De pila
Tipo según Chomsky	Tipo 3	Tipo 2
Lenguajes de programación	No pueden ser representados	Se pueden representar
Reglas	S -> a S -> aA	S -> a S -> AaA S -> aAb

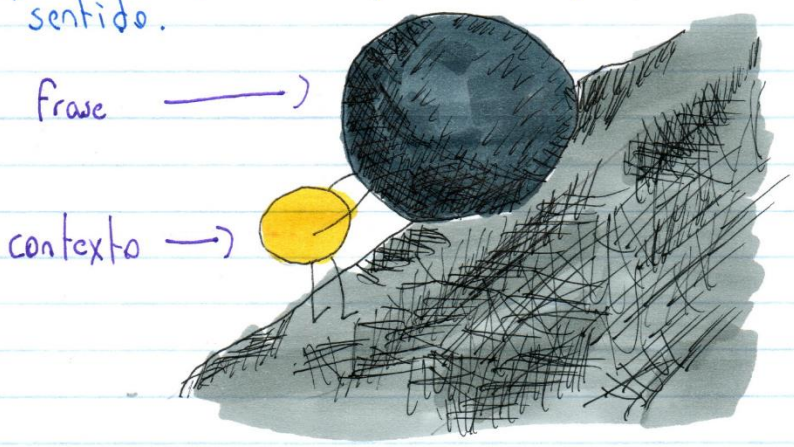
Con esto podemos ver que la diferencia son las reglas de producción, ya que en las gramáticas libres de contexto realmente no importa que está antes del componente, siempre y cuando se cumpla la regla, mientras que las regulares son dependientes del componente anterior.

Ya solo nos queda responder, ¿Qué es el contexto y para qué sirve?

Dejar claro que, dependiendo el tema, la palabra contexto tiene significados diferentes tales como:

- lugar
- evento
- edad
- variables
- jerga

Y un largo etc, pero todos ellos tienen algo en común: es algo que los precede y hace que la frase tenga sentido.



Por lo que para el tema que nos compete no es nada más ni nada menos que las palabras reservadas, expresiones, estructuras, signos, parentesis, funciones, etc. que van antes del componente.

```
def main():
```

main podría ser cualquier cosa, pero al estar precedido por def lo convierte en el nombre de una función.

```
print("lilith")
```

lilith podría ser el nombre de una variable, pero debido a donde se encuentra solo va a ser mostrado en pantalla.

Con todo esto solo resta preguntar, ¿sirven para algo?

Y la respuesta es: SI

El contexto permite definir todo tipo de cosas dentro de un lenguaje de programación: estructuras lógicas, operaciones, pilas, nodos, etc, los cuales solo tienen sentido en determinadas situaciones.

```
import os
```

Por ejemplo, podrías colocar el nombre de un paquete, pero si no colocas el import no servirá de nada y solamente ocasionará problemas por estar fuera de contexto.

5 BITÁCORA DE INCIDENCIAS

PROBLEMA	FECHA	HORA	SOLUCIÓN	FECHA	HORA
El software de los arboles no funcionaba correctamente con los números	15-09	13:30	Anteceder un carácter, para que piense en el como una cadena	15-09	13:34
Perdí el código realizado para la realización de los primeros dos árboles de derivación	15-09	15:21	No hubo, la página no los guarda		

6 OBSERVACIONES

La derivación de los árboles pensé que se podía hacer desde las gramáticas de tipo 3, pero ahora veo lo impráctico que son para dichas estructuras, ya que más bien parecen una lista, dado que solo van hacia izquierda o derecha.

En cambio, las libres de contexto no están atadas a esta convención, por lo que pueden ir en cualquier dirección y se pueden derivar con suma facilidad.

7 ANEXOS

7 REFERENCIAS BIBLIOGRÁFICAS

https://softwarerecs.stackexchange.com/questions/74012/tool-for-drawing-parse-trees

- Por indicar un software útil para desarrollar árboles de derivación.

Sundeep Saradhi Kanthety. (2021, 3 de noviembre), CONTEXT FREE GRAMMAR IN AUTOMATA THEORY || CFG IN ATFL || TOC || AUTOMATA THEORY. YouTube. https://www.youtube.com/watch?v=UV-j2_xyF-U

Sundeep Saradhi Kanthety. (2021, 8 de noviembre), DIFFERENCE BETWEEN REULAR GRAMMAR AND CONTEXT FREE GRAMMAR IN AUTOMATA THEORY || TOC. YouTube. https://www.youtube.com/watch?v=p64A328JZho

Anonimo. (recuperado el 15 de septiembre de 2024). Context-Free Grammars. Stanford University. https://web.stanford.edu/class/archive/cs/cs103/cs103.1164/lectures/18/Small18.pdf