



DEPARTAMENTO DE SISTEMAS COMPUTACIONALES E INFORMÁTICA

ASUNTO: SOLICITUD DE ACTIVIDADES

Celaya, Guanajuato, 13 / septiembre / 2022

LENGUAJES Y AUTÓMATAS II

DOCENTE DESIGNADO: ISC. RICARDO GONZÁLEZ GONZÁLEZ
SEMESTRE AGOSTO-DICIEMBRE 2022

ACTIVIDAD 4 (VALOR 42 PUNTOS)

LEA CUIDADOSAMENTE, Y REALICE LAS SIGUIENTE ACTIVIDADES, CONSIDERANDO LOS CRITERIOS DE CALIDAD PROPUESTOS EN LOS DOCUMENTOS DE LA [GUÍA TUTORIAL](#), Y LA [RÚBRICA DE EVALUACIÓN](#),

EL LECTOR DEBE TOMAR MUY EN CUENTA QUE ESTA ACTIVIDAD ES UN EXAMEN, Y NO UNA SIMPLE TAREA, PUES DEMANDA DEDICACIÓN PARA INVESTIGAR, LEER, ANALIZAR, REDACTAR, ILUSTRAR Y PROPOSER DE MANERA PROFESIONAL LOS TEMAS PROPUESTOS EN LA ESTRUCTURA TEMÁTICA DE ESTA ASIGNATURA.

2. ANÁLISIS LÉXICO.

INVESTIGUE, LEA, COMPREnda Y ELABORE UNA **MONOGRAFÍA TÉCNICA** COMPLETAMENTE APEGADA A LO SOLICITADO EN LA [GUÍA TUTORIAL](#) (PUNTO 3, INCISO a) ACERCA DE LOS SIGUIENTES TEMAS :

- GENERADORES DE ANALIZADORES LÉXICOS
- APLICACIONES (CASO DE ESTUDIO)

CONSIDERACIÓN :

DEBE USTED ENTENDER EL VALOR QUE TIENE ESTA ACTIVIDAD Y QUE LOS TEMAS ANTES REFERIDOS, PARA NADA DEBEN SER ABORDADOS COMO SIMPLES CONCEPTOS REDACTADOS CON LA LIGEREZA QUE YA SE HA OBSERVADO EN ACTIVIDADES PREVIAS.

ANALICE CADA TEMA, SUS CARACTERÍSTICAS, SU IMPORTANCIA, SUS CONCEPTOS, SUS EJEMPLOS, SUS ILUSTRACIONES, Y LOS TIPOS DE EVIDENCIAS QUE USARÁ PARA DEMOSTRAR QUE USTED HA ADQUIRIDO UN VERDADERO CONOCIMIENTO ACERCA DE ÉSTOS.





IMPORTANTE : SI LO REQUIERE PUEDE CONSULTAR EL [SIGUIENTE DOCUMENTO](#) PARA ORIENTAR SU TRABAJO EN CONOCER QUÉ ES Y CÓMO HACER UNA MONOGRAFÍA CON EL RIGOR ACADÉMICO REQUERIDO.

EN LO REFERENTE AL TEMA APLICACIONES (CASO DE ESTUDIO), ARRIBA REFERIDO Y A MANERA DE PRÁCTICA, ELABORE CON EL APOYO DEL SOFTWARE LIBRE DENOMINADO **ANALIZADOR LÉXICO YACC**, UN EJERCICIO DEMOSTRATIVO SOBRE AL MENOS TRES DEFINICIONES LÉXICAS PROPIAS DEL LENGUAJE PROGRAMACIÓN C.

3. ANÁLISIS SINTÁCTICO.

INVESTIGUE, LEA, COMPREnda Y ELABORE UNA **MONOGRAFÍA TÉCNICA** COMPLETAMENTE APEGADA A LO SOLICITADO EN LA [GUÍA TUTORIAL](#) (PUNTO 3, INCISO a) ACERCA DE LOS SIGUIENTES TEMAS :

- GRAMÁTICAS LIBRES DE CONTEXTO Y ÁRBOLES DE DERIVACIÓN
- DIAGRAMAS DE SINTAXIS

CONSIDERACIÓN :

DEBE USTED ENTENDER EL VALOR QUE TIENE ESTA ACTIVIDAD Y QUE LOS TEMAS ANTES REFERIDOS, PARA NADA DEBEN SER ABORDADOS COMO SIMPLES CONCEPTOS REDACTADOS CON LA LIGEREZA QUE YA SE HA OBSERVADO EN ACTIVIDADES PREVIAS.

ANALICE CADA TEMA, SUS CARACTERÍSTICAS, SU IMPORTANCIA, SUS CONCEPTOS, SUS EJEMPLOS, SUS ILUSTRACIONES, Y LOS TIPOS DE EVIDENCIAS QUE USARÁ PARA DEMOSTRAR QUE USTED HA ADQUIRIDO UN VERDADERO CONOCIMIENTO ACERCA DE ÉSTOS.

A MODO DE PRÁCTICA REALICE ESTE PUNTO Y ELABORE EJERCICIOS PRÁCTICOS CON LOS CUÁLES USTED DEMUESTRE

- ¿ QUÉ SON LAS GRAMÁTICAS LIBRES DE CONTEXTO ?
- ¿ QUÉ ES UN CONTEXTO ?, HAGA UNA MUY BUENA ILUSTRACIÓN DE ESTE CONCEPTO.
- ¿ SIRVE DICHO CONTEXTO A LOS PROPÓSITOS DE UNA GRAMÁTICA QUE DEFINA UN LENGUAJE FORMAL ?

ADEMÁS INCLUYA EJEMPLOS (AL MENOS TRES) DE ÁRBOLES DE DERIVACIÓN Y DEMUESTRE CÓMO AYUDAN A LOS PROCESOS DE ANÁLISIS SINTÁCTICOS.

IMPORTANTE : SI LO REQUIERE PUEDE CONSULTAR EL [SIGUIENTE DOCUMENTO](#) PARA ORIENTAR SU TRABAJO EN CONOCER QUÉ ES Y CÓMO HACER UNA MONOGRAFÍA CON EL RIGOR ACADÉMICO REQUERIDO.

POR ÚLTIMO, RECUERDE LEER LA [GUÍA TUTORIAL](#) PARA EL CORRECTO TRATAMIENTO DE ESTE INCISO.





EDUCACIÓN

SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO
NACIONAL DE MÉXICO

Instituto Tecnológico de Celaya
Departamento de Sistemas y Computación

¿ QUÉ SE CALIFICARÁ ?

LA RÚBRICA PARA EVALUAR ESTA ACTIVIDAD ESTARÁ INTEGRADA POR LOS SIGUIENTES CRITERIOS.

- a. LA OPORTUNIDAD. SI EL TRABAJO FUE ENTREGADO OPORTUNAMENTE.
- b. LA COMPRENSIÓN. SE VALORARÁ EL GRADO DE COMPRENSIÓN DEL TEMAS ANALIZADOS.
- c. LA CALIDAD. SI LAS EVIDENCIAS ENVIADAS CORRESPONDEN A LA CALIDAD ESPERADA PARA ESTE NIVEL PROFESIONAL QUE SE CURSA.
- d. LA CAPACIDAD DE SÍNTESIS. SI LAS EVIDENCIAS ENTREGADAS TIENEN EL NIVEL DE DETALLE Y PROFUNDIDAD REQUERIDA, O EN BIEN SI SE OMITIERON CONCEPTOS CON EL AFÁN DE SIMPLIFICAR Y ENTREGAR UN MATERIAL ACADÉMICA Y TÉCNICAMENTE POBRE.
- e. LA CREATIVIDAD. LA MANERA EN QUE SE EXPRESAN LOS CONCEPTOS Y EL TRATAMIENTO QUE SE DA A LA INFORMACIÓN ANALIZADA PARA QUE ÉSTA SEA COMPRESIBLE EN SU ESENCIA.

IMPORTANTE : CUENTA CON EL TIEMPO SUFFICIENTE PARA REALIZAR ESTA ACTIVIDAD Y SUMAR PUNTOS IMPORTANTES A SU CALIFICACIÓN DE ESTA EVALUACIÓN.

IMPORTANTE : TODO EL MATERIAL ESCRITO DEBERÁ SER HECHO A MANO.



Av. Antonio García Cubas #600 esq. Av. Tecnológico,
Colonia Alfredo V. Bonfil, C.P. 38010 Celaya, Gto.
Tel. 01 (461) 611 75 75
e-mail: lince@celaya.tecnm.mx
tecnm.mx | celaya.tecnm.mx





CONSIDERACIONES.

CADA UNO DE LOS PUNTOS ANTERIORES DEBE SER DESARROLLADO CON LA PROFUNDIDAD ACORDE A UN NIVEL PROFESIONAL, Y APEGÁNDOSE COMPLETAMENTE A LAS DIRECTRICES DE LA GUÍA TUTORIAL.

NO CONCIBA ESTE TRABAJO, COMO UN SIMPLE RESUMEN O EJERCICIO DE TRANSCRIPCIÓN, PUES EL VALOR INDICADO AL INICIO DE ESTA ACTIVIDAD LE DARÁ A USTED UNA BUENA IDEA DE LO QUE SE ESPERA DE ELLA, EN CUANTO A CALIDAD Y EL APRENDIZAJE OBTENIDO, MISMO QUE SERÁ PUESTO A PRUEBA MEDIANTE UN EXAMEN ESCRITO O BIEN ORAL EN CLASE.

SI DECIDIÓ ELABORAR ESTA ACTIVIDAD EN EQUIPO, CADA INTEGRANTE DE ÉSTE DEBERÁ POSEER EL MISMO NIVEL DE CONOCIMIENTO, PUES TAN SOLO REPARTIR TEMAS ENTRE LOS INTEGRANTES DEL EQUIPO, SUPONDRIÁ UN GRAVE ERROR DE INTERPRETACIÓN A LA INTENCIÓN DIDÁCTICA REAL DE ESTA ACTIVIDAD.

POR ÚLTIMO, ESTA ACTIVIDAD SOLO SE PODRÁ DESARROLLAR EN EQUIPO, SI SE REGISTRÓ EN UNO PREVIAMENTE, UTILIZANDO EL FORMATO ENTREGADO EN LA ACTIVIDAD INICIAL. DE LO CONTRARIO DEBERÁ ELABORAR Y ENTREGAR LA ACTIVIDAD DE FORMA INDIVIDUAL.

LA ENTREGA DE DICHO REGISTRO SE HARÁ VÍA CORREO ELECTRÓNICO ENVIANDO ÉSTE AL PROFESOR DESIGNADO, Y POSTERIORMENTE EN CLASE ENTREGANDO LA HOJA EN FÍSICO.

OBSERVACIONES:

- CADA HOJA QUE ENTREGUE DE SU ACTIVIDAD, DEBERÁ ESTAR FIRMADA AL MARGEN DERECHO, INCLUIDA LA PROPIA SOLICITUD DE LA ACTIVIDAD.
- **INTEGRE TODO SU TRABAJO EN UN SOLO ARCHIVO DE TIPO .PDF, Y ASIGNE EL NOMBRE QUE A CONTINUACIÓN SE INDICA.**

NO OLVIDE ANEXAR LAS HOJAS DE ESTA ACTIVIDAD Y DE SU TRABAJO DESPUÉS DE SU PORTADA.

- UNA VEZ ELABORADA SU ACTIVIDAD, RECUERDE DIGITALIZARLA Y NOMBRARLA EN BASE A LA NOMENCLATURA QUE SE INDICA MÁS ADELANTE EN ESTE DOCUMENTO.
- SI SUS EVIDENCIAS ENVIADAS POR CORREO, NO CUMPLEN CON LA NOMENCLATURA SOLICITADA, NO SERÁN CONSIDERADAS COMO EVIDENCIAS PARA SU EVALUACIÓN.
- **POR ÚLTIMO, POR FAVOR GESTIONE APROPIADAMENTE SU TIEMPO, Y SEA PUNTUAL EN SU ENTREGA Y ASÍ EVITAR PROBLEMAS DE NULIDAD POR EXTEMPORANEIDAD.**





EDUCACIÓN

SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO
NACIONAL DE MÉXICO

Instituto Tecnológico de Celaya
Departamento de Sistemas y Computación

LA NOMENCLATURA SOLICITADA PARA ENVIAR SU TRABAJO ES LA SIGUIENTE :

AAAA-MM-DD_TNM_CELAYA_MATERIA_DOCUMENTO_[EQUIPO]_NOCTROL_APELLIDOS_NOMBRE_SEM.PDF

(NOTA : *** TODO DEBE SER ESCRITO USANDO LETRAS MAYÚSCULAS ***)

DONDE :

TNM_CELAYA	: INSTITUCIÓN ACADÉMICA
AAAA	: AÑO
MM	: MES
DD	: DÍA
MATERIA	: LAI , LI MÁS EL GRUPO (-A , -B, -C)
DOCUMENTO	: A1-ACTIVIDAD 1, P1-PRACTICA 1, R1-REPORTE 1, T1-TAREA 1, PG1-PROGRAMA, ETC. (CAMBIANDO EL NÚMERO CONSECUТИVO POR EL QUE CORRESPONDA)
[EQUIPO]	: NÚMERO DEL EQUIPO QUE CORRESPONDA SEGÚN INDICACIÓN DEL PROFESOR. [OPCIONAL]
NOCTROL	: SU NÚMERO DE CONTROL
APELLIDOS	: SUS APELLIDOS
NOMBRE	: SU NOMBRE
SEM	: EL PERIODO SEMESTRAL EN CURSO: AGO-DIC

EJEMPLO :

SI EL TRABAJO SE SOLICITÓ EN EQUIPO.

2022-09-13_TNM_CELAYA_LAI-A_A4_EQUIPO_99_9999999_PEREZ_PEREZ_JUAN_AGO-DIC22.PDF

DONDE EL NOMBRE DEBERÁ CORRESPONDER AL JEFE DE EQUIPO QUE HACE LA ENTREGA DEL TRABAJO.

SI EL TRABAJO SE SOLICITÓ INDIVIDUALMENTE.

2022-09-13_TNM_CELAYA_LAI-A_A4_9999999_PEREZ_PEREZ_JUAN_AGO-DIC22.PDF



Av. Antonio García Cubas #600 esq. Av. Tecnológico,
Colonia Alfredo V. Bonfil, C.P. 38010 Celaya, Gto.
Tel. 01 (461) 611 75 75
e-mail: lince@celaya.tecnm.mx
tecnm.mx | celaya.tecnm.mx





FECHA Y HORA DE ENTREGA:

LA INDICADA EN LA PLATAFORMA VIRTUAL.

EN CASO DE QUE EL TRABAJO SE HAYA SOLICITADO EN EQUIPO, EL JEFE DEL MISMO SERÁ EL ÚNICO RESPONSABLE DE ENVIAR LA ACTIVIDAD EN LA PLATAFORMA VIRTUAL.

MUY IMPORTANTE:

1. DESPUÉS DE LA HORA INDICADA EN LA PLATAFORMA VIRTUAL (AÚN CUANDO SOLO SEA UN MINUTO O VARIOS), LA ACTIVIDAD SERÁ CONSIDERADA COMO EXTEMPORÁNEA Y NO CONTARÁ COMO EVIDENCIA PARA SU EVALUACIÓN.

SE LE SUGIERE ENVIAR CON ANTICIPACIÓN SU ACTIVIDAD A FIN DE EVITAR CONFLICTOS POR NO ENTREGAR ÉSTA A TIEMPO.

BAJO NINGÚN PRETEXTO O JUSTIFICACIÓN SE ACEPTARÁN LOS TRABAJOS EXTEMPORÁNEOS, EVITE LA PENA DE RECORDAR A USTED QUE EL VALOR DE LA PUNTUALIDAD ES PARTE IMPORTANTE DE SUS EVIDENCIAS Y ES EL PRIMER PUNTO QUE SE HA DE EVALUAR.

2. NO OLVIDE ANEXAR A SU ARCHIVO .PDF DE EVIDENCIAS UNA PORTADA PROFESIONAL, Y ESTA SOLICITUD DE ACTIVIDADES CON TODAS LAS HOJAS FIRMADAS EN EL MARGEN DERECHO.
3. POR ÚLTIMO, TODA EVIDENCIA GENERADA QUE CONTENGA AL MENOS UNA TRANSCRIPCIÓN DE CUALQUIER FUENTE Y DE CUALQUIER TIPO, ES DECIR CON MATERIAL PLAGIADO SERÁ ANULADA DE FORMA INCONTROVERTIBLE.



INSTITUTO TECNOLOGICO NACIONAL DE MEXICO EN CECILAYA

LENGUAJES Y AUTOMATAS II

ACTIVIDAD 4. ANÁLISIS LÉXICO Y SINTÁCTICO.

Dto. de Sistemas Computacionales.

Docente: I.S.C. González González Ricardo.

Equipo 3:

- Ramirez García Marco Isaias (19030260)
- García Ramirez Luis David (19030263)
- Pérez Cabrera José Eduardo (19030985)

A 22 de Septiembre de 2022

I Introducción

A lo largo del tiempo que llevan existiendo las ciencias computacionales, siempre se han tomado como base a otras ciencias como base. Un ejemplo de esto son las matemáticas, ya que gracias a modelos matemáticos podemos representar algo tan complejo como lo son los automatas, grafos o árboles.

Pero no solo han sido las matemáticas las base para las ciencias computacionales. La lingüística ha formado parte de estas bases desde el inicio, ya que sin esta no habría lenguajes de programación de bajo y alto nivel, por lo que sería imposible la comunicación entre humano y máquina. De esta ciencia se rescatan conceptos muy importantes como lo son gramática, sintaxis, semántica y léxico, de este último es sobre lo que se abordará en este trabajo, más en particular en los generadores de análisis léxico, se abarcarán conceptos claves como: ¿Qué son?, ¿Para qué sirven?, ¿Cómo funcionan? y algunos ejemplos de los mismos y por último se propone un ejercicio en el generador léxico lex sobre las definiciones léxicas de lenguaje C.

ANALIZADOR LÉXICO.

Antes que nada hay que poner una definición clara sobre qué es un analizador léxico en las Ciencias Computacionales. Un analizador léxico lee los caracteres del archivo de entrada, en dicho archivo se encuentran las Cadenas (tokens) que el mismo se encargará de analizar, después de eso tratará de separar la cadena en subcadenas de ese modo retornará los distintos tokens.

La tarea antes descrita puede sonar una tarea "simple" pero nada más alejado de la realidad, hacer un analizador léxico es una tarea exhaustiva y sumamente compleja, por lo que se cuentan con herramientas que generan de manera automática el analizador léxico con las reglas descritas por el usuario.

Dichas herramientas permiten al usuario definir el alfabeto, reglas léxicas y Sintácticas mediante expresiones regulares. De esta forma se libera al usuario de la responsabilidad de tener que computar una linea de código para generar él mismo su analizador léxico.

A continuación se mencionarán y se describirán tres analizadores léxicos, que se podría decir que son los más conocidos por la comunidad, de este modo encontrar información acerca de los mismo es relativamente sencilla y accesible para cualquier interesado.

LEX

El primer analizador léxico que se describirá es Lex, también quizás el más conocido para realizar dicha tarea.

Lex, este analizador acepta especificaciones para resolver problemas de alto nivel, el mismo produce un programa en lenguaje C y acepta expresiones regulares, dichas expresiones son definidas por el usuario.

El código Lex reconoce estas expresiones y trata de dividir la cadena de entrada con base a las expresiones definidas.

El archivo lex hace una asociación entre las expresiones regulares y los fragmentos de programa. Ya que, cada expresión se encuentra en la salida del programa escrito en lex, después de eso se ejecuta el fragmento de código correspondiente.

El usuario establecerá el código necesario para completar estas funciones. dicho código tendrá que estar escrito en lenguaje C y el programa se encargará de reconocer las expresiones que generarán dicho código. Debemos tener en cuenta que Lex no es lenguaje, si no, un generador que añade un nuevo lenguaje al lenguaje de programación C.

El programa yylex reconoce expresiones, conforme va detectando estas expresiones ejecutará las acciones específicas. todo esto gracias a que lex puede convertir las acciones y expresiones del usuario a yylex.

Los programas escritos en lex sólo aceptan expresiones regulares. Yaac se encarga de reconocer las gramáticas de texto libre, por lo que hacer lex y yaac trabajen juntos es relativamente sencillo. Para hacer que yaac reconozca las gramáticas requiere un analizador de bajo nivel para reconocer a los tokens de entrada. Lex se usa para dividir la entrada. Ya que el generador de reconocimiento le asigna una estructura a la piezas resultantes, de este modo es fácil hacer que programas escritos en otros generadores se han añadidos a programas escritos en Lex.

A partir de las expresiones regulares lex genera un autómata finito, dicho autómata es interpretado y no compilado esto para ahorrar espacio. El tiempo que es utilizado por un programa lex para reconocer y dividir una cadena de entrada es totalmente proporcional al tamaño de la cadena, mientras que las órdenes o la complejidad de las mismas no es factor a tener en cuenta para la velocidad. La única forma para que suceda lo contrario es si las órdenes tengan contexto posterior o incluyan una cantidad significativa de exploración. El autómata finito crece proporcionalmente en el número y complejidad de las órdenes.

Lex no está limitado a fuente de un posible carácter. Un ejemplo de esto es el siguiente: Una orden que busque la cadena abc y otra que busca abcd. lex reconocerá la cadena abc y el puntero de entrada apuntará a d.

Formato de lex

El formato de lex es el siguiente:

{definiciones}

%;

{ordenes}

%;

{sus rutinas del usuario}

Por lo regular las Subrutinas del usuario son omitidas, esto quiere decir que el segundo % es opcional pero el primero no, ya que este marca el principio de las órdenes.

Las órdenes son las decisiones de control definidas por el usuario, estas forman una tabla con dos columnas, la columna de la izquierda tiene las expresiones regulares, mientras que la columna de la derecha contiene las acciones, estas acciones son fragmentos de código que se ejecutarán cuando la expresión regular de la misma fila sea reconocido.

String printf("Se encontró una palabra reservada
llamada String");

Expresiones de lex

Una expresión define el conjunto de literales que serán comparadas, estas tienen caracteres de texto, las letras del alfabeto y los dígitos son caracteres de texto. Entonces:

String va coincidir siempre que aparezca la expresión String.

Los caracteres Operador son los siguientes

" , \[] ^ - ? . * + | () \$ / { } % < >

Si uno de estos caracteres va ser usado es necesario poner la diagonal invertida (\) entre cada caracter de operador o encerra entre Comillas dobles para delimitar los caracteres de operador. Ejemplos:

abc"++" → abc++ ← abc\+\+

i">" a → i>a ← i\>\a

x"? " z → x?z ← x\?\a z

Salidas de escape C normales con la diagonal invertida (\)

\n	Salto de linea
\t	Tabulador
\b	Espacio en blanco
\\\	Diagonal invertida.

Llamar a lex

Para Compilar un programa lex hay dos pasos, la primera de ellas es convertir el programa escrito en flex a un lenguaje de propósito general. Entonces después de esto el programa tiene que ser compilado y cargado con una de las librerías de subrutinas de lex. El programa generado se encuentra en un archivo llamado lex.yy.c

Para acceder al programa resultante se usan los siguientes Comandos :

lex Source
C lex.yy.c -l

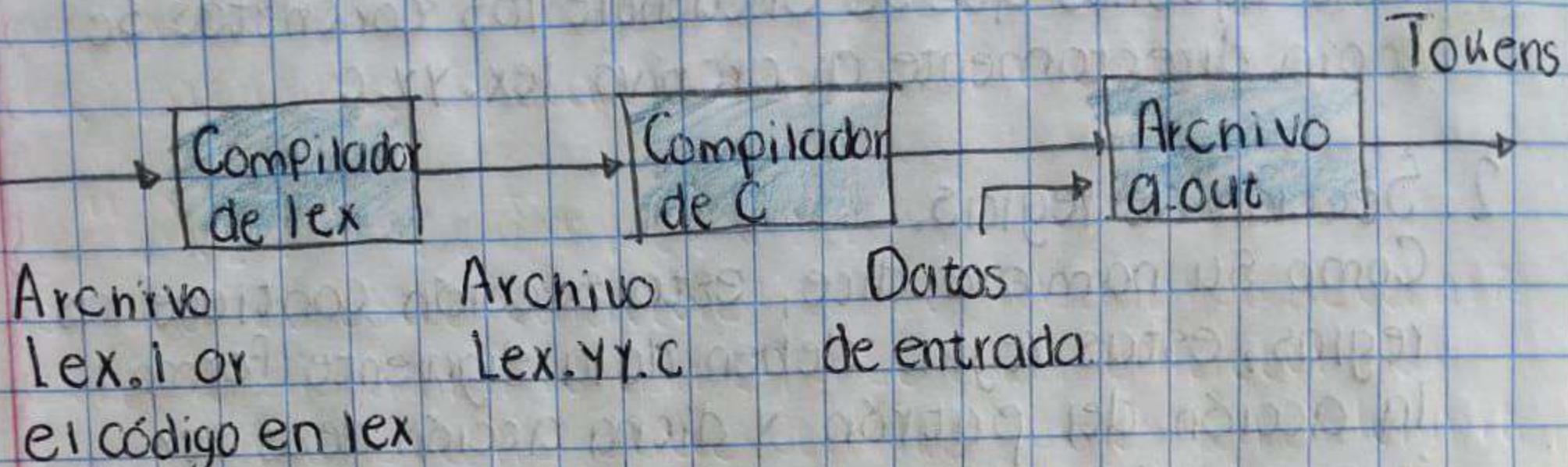
El programa resultante se pone en el fichero usual a.out para ser ejecutado posteriormente.

FILEX

Flex traducido al español como Generador de Analizador léxico rápido al igual que lex es una herramienta de software para generar un analizador léxico, Flex fue creado por Vern Paxson. Este analizador al igual que lex su usa conjunto con Yacc.

Flex a comparación de Yacc y Lex es más flexible y genera un código que se ejecuta más rápido. La función yylex() se genera automáticamente por el mismo generador, esto sucede cuando se le alimenta con un archivo .l, el analizador espera a que se llame a esta función para recuperar los tokens.

A continuación por medio de un diagrama se explica como funciona Flex:



La función yylex() es la encargada de ejecutar las reglas, mientras que la extensión .l sirve para guardar los programas.

De acuerdo al diagrama anterior hay tres pasos.

1. Se tiene un archivo de entrada el cual contiene las reglas para la construcción del analizador léxico llamado lex.l, el compilador de Lex transforma el archivo lex.l a un programa C llamado lex.yy.c.
2. El compilador de C compila el lex.yy.c el cual generará el a.out que es un archivo ejecutable.
3. Por último por medio del archivo de entrada se generan los tokens.

Estructura del programa

1. Sección de definición.

Esta sección contiene las variables, definiciones regulares y constantes manifiestas.

El texto que vaya a ser introducido se incluye entre Signos de porcentajes y corchetes (% y %) Todo aquello que se encuentre los corchetes se copia directamente al archivo lex.yy.c

2. Sección de reglas.

Como su nombre dice, esta sección contiene las reglas, estas reglas deben de la siguiente forma: la acción del patrón y dicha acción debe ser involuntaria y la acción debe estar entre paréntesis, debe estar delimitada de inicio y fin por los caracteres % %.

% %

Patrón Acción

% %

Ejemplo:

% %

[0-9] printf("Es un número del 0 al
9");

%

[a-z] printf("Es una letra");

%

3. Sección de Código de Usuario.

Esta sección contiene declaraciones en lenguaje C y funciones adicionales.

Entonces habiendo revisado las tres partes para formar un programa en Flex, la estructura básica del programa quedará de la siguiente manera:

%{

// Definiciones

%}

%%

Preguntas

%%

Sección de código para el usuario

Ejecutar el programa.

Primero debemos guardar el programa con la extensión .l o .lex y ejecutar los siguientes comandos en el siguiente orden.

gcc lex.yyy.c
.la.out

Después proporcionar los datos al analizador léxico

A continuación se mostrará un ejemplo demostrativo de tres definiciones de lenguaje C con el analizador léxico Lex.

```
#include <Studio.h>
#include <stdlib.h>
#include <Conio.h>
#include <String.h>
```

```
File *yyin;
```

```
%{
```

```
%option noyywrap
%option yylineno
```

```
/* Tokens */
```

```
DIGITO
```

```
[0-9]
```

```
LETRA
```

```
[a-zA-Z]
```

```
ESPACIO
```

```
[" " ]
```

```
GUIONES
```

```
["-"]
```

```
C_COMA
```

```
[" , " ]
```

```
C_PUNTO
```

```
[" . " ]
```

```
C_PUNTOCOMA
```

```
[" ; " ]
```

```
C_DOSPUNTOS
```

```
[" ; ; " ]
```

```
IDENTIFICADOR
```

```
{LETRA}({LETRA}|{DIGITO}|{GUIONES})*
```

```
CONST_INTEGER
```

```
(-?[1-9][0-9]{0-4})10
```

```
CONST_FLOAT
```

```
{DIGITO}{0-8}{C_PUNTO}{DIGITO}{1-8}
```

```
CONST_CADENA
```

```
"[^\\n]*\"
```

```
/* Operadores lógicos y aritméticos */
```

```
O_SUMA
```

```
["+"]
```

```
O_RESTA
```

```
["-"]
```

```
O_MULTI
```

```
["*"]
```

O - DIV	[" / "]
O - IGUAL	[" = "]
O - MAYOR	[" > "]
O - MENOR	[" < "]
O - MAYORIGUAL	[" >= "]
O - MENORIGUAL	[" <= "]
O - COMPIGUAL	[" == "]
O - NEGACION	[" ! "]
O - DISTINTO	[" != "]

* Palabras reservadas *

IF	" IF" " IF"
WHILE	" While" " WHILE"
DECVAR	" DECVAR"
ENDDEC	" ENDDC"
INTEGER	" integer" " INTEGER"
FLOAT	" float" " FLOAT"
WRITE	" write" " WRITE"

% %

```
{ O - DOSPUNTOS }
{ C - COMA }
{ CONST - ENTEROS }
{ CONST - FLOAT }
{ CONST - CADENA }
```

```
{ printf( "\n O - DOSPUNTOS (%.5s)", yytext); }
{ printf( "\n C - COMA (%.5s)", yytext); }
{ printf( "\n CONST - ENTEROS (%.5s)", yytext); }
{ printf( "\n CONST - FLOAT (%.5s)", yytext); }
{ printf( "\n CONST - CADENA (%.5s)", yytext); }
```

```
{ O - SUMA }
{ O - RESTA }
{ O - MULTI }
{ O - DIV }
```

```
{ printf( "\n O - SUMA (%.5s)", yytext); }
{ printf( "\n O - RESTA (%.5s)", yytext); }
{ printf( "\n O - MULTI (%.5s)", yytext); }
{ printf( "\n O - DIV (%.5s)", yytext); }
```

{0-IGUAL}

```
i printf("In O-IGUAL (%s)", yytext); }
```

{0-MAYOR}

```
i printf("In O-MAYOR (%s), yytext); }
```

{0-MENOR}

```
i printf("In O-MENOR (%s), yytext); }
```

{0-MAYORIGUAL}

```
i printf("In O-MAYORIGUAL (%s), yytext); }
```

{0-MENORIGUAL}

```
i printf("In O-MENORIGUAL (%s), yytext); }
```

{0-COMP-IGUAL}

```
i printf("In O-COMP-IGUAL (%s), yytext); }
```

{0-NEGACION}

```
i printf("In O-NEGACION (%s), yytext); }
```

{0-DISTINTOS}

```
i printf("In O-DISTINTO (%s), yytext); }
```

{0-DOSPUNTOS}

```
i printf("In O-DOSPUNTOS (%s), yytext); }
```

{IF}

```
{ printf("In Palabra reservada if, yytext); }
```

{WHILE}

```
{ printf("In Palabra reservada While, yytext); }
```

{INTEGER}

```
{ printf("In palabra reservado int, yytext); }
```

{FLOAT}

```
{ printf("In palabra reservado float, yytext); }
```

{WRITE}

```
{ printf("In palabra reservada write, yytext); }
```

{DECVAR}

```
{ printf("In palabra reservado decvar, yytext); }
```

{ENDDEC}

```
{ printf("In palabra reservado enddec, yytext); }
```

{FLOAT}

```
{ printf("In palabra reservado float, yytext); }
```

{INTEGER}

```
{ printf("In palabra reservado int, yytext); }
```

%^%

}

Ejemplos de Salida.

if Palabra reservada if

== O-COMP-IGUAL

while Palabra reservada While

> O-COMP-MAYOR

CONCLUSIÓN

Para finalizar, como se vió a lo largo de este trabajo los analizadores léxicos están presentes en cualquier lenguaje de programación, por lo que es de suma importancia saber como funcionan, puesto que esto le dará un mejor entendimiento al programador de cómo es que funcionan el proceso de compilación o de interpretación.

Sin duda alguna esta información le servirá al equipo de trabajo para crear su propio analizador léxico, debido a que ahora mismo se tiene una mejor noción de como es que funcionan los analizadores. También el equipo ya tiene una gran idea de que lenguaje de programación utilizará para crear el analizador, ya que un analizador léxico se apoya mucho de estructura de datos, entonces utilizar un lenguaje que soporte y emplee estructuras de datos será un candidato.

Por último creemos que las herramientas Flex y Lex son herramientas muy útiles que nos puede servir para nuestra futura vida laboral, no solo para crear compiladores o intérpretes, si no para comunicarnos con cualquier tipo de máquina por medio de estas herramientas.

Gramáticas Libres de Contexto

Las "gramáticas libres de contexto" o también llamadas "gramáticas tipo 2" según la clasificación de Chomsky. Son aquellas que cuyo nombre indica no dependen del contexto para ser desarrolladas y su reconocedor es un autómata de pila que puede ser determinístico o no determinístico, está definida por una cuádrupla como todos las gramáticas, dicha gramática se representa como: $G = (N, T, S, P)$; donde " N " representa los valores NO Terminales, " T " representa los valores Terminales, " S " el símbolo inicial de " P " que representa el conjunto finito de producciones posibles para realizar con la gramática que sea evaluada, lo que vuelve diferente a la gramática son las restricciones aplicadas a las reglas de producción representadas por $A \rightarrow w$; donde A representa la parte izquierda de la producción y pertenece a la unión de cualquier valor no terminal con el símbolo inicial, representado como $A \in U\{S\}$ y w representa la parte derecha de la producción y pertenece a la combinación de todas las posibles uniones entre los valores terminales y los no terminales que es representado como $w \in (N \cup T)^*$

Las gramáticas libres de contexto son parte de la técnica más común para definir lenguajes de programación por sus grandes similitudes con los lenguajes naturales aunque con las reglas de escritura necesarias para volverla un lenguaje formal, se utiliza una notación conocida como BNF (Backus-Naov Form) utilizada para definir la sintaxis de un lenguaje de programación alimentado por el tipo de gramática que genera un lenguaje independiente de contexto o también llamado lenguaje libre de contexto que dada la gramática formada por la cuadriga ya mencionada es representado por $L(G) = \{w \in T^* \mid S \Rightarrow^* w\}$; donde el lenguaje generado por la gramática cumple con las restricciones de que w pertenece a los posibles terminales mientras que el símbolo inicial S debe de tener que en sus producciones se evalúe cualquier producción de w , recapitulando se tiene que las gramáticas tienen derivaciones producidas por las reglas de producción por que pueden ser evaluadas por un árbol de derivaciones que forma parte de las técnicas posibles para evaluar los lenguajes producidos y determinar el cumplimiento de las reglas que hayan sido inicializadas.

Contexto

El contexto forma parte indispensable del mensaje que se quiere transmitir cuando se establece una comunicación, para contextualizar la interpretación de un mensaje recibido. Se toman ciertos parámetros externos como el entorno y las diferentes situaciones que se presenten, el contexto a su vez es una poderosa herramienta para transmitir mensajes capaces de ser interpretados reduciendo a su vez la posibilidad de ambigüedades en lo que se quiere transmitir, su funcionalidad más importante es que al mensaje a transmitir se organiza y se le busca el sentido de ser, para que tenga lógica; la comunicación tiene los elementos de el emisor que expresa un mensaje, el receptor que recibe los mensajes y puede enviar respuestas, el contenido del mensaje que se quiera comunicar, el canal que será el medio de transmisión del mensaje emitido, el código es el sistema como el tipo de idioma y el contexto que representa el entorno que rodea al emisor y al receptor para establecer la comunicación y que a su vez está teniendo lógica para que el mensaje transmitido sea comprendido correctamente.



Ejemplos de Gramáticas libres de Contexto y su representación

Para las exemplificaciones se estará trabajando con el alfabeto

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

Para expresar una GLC se puede hacer el uso de fórmulas o descripciones para determinar sus reglas, es decir

$$L = \{a^n b^n \mid n \geq 0\}$$

Puede ser expresado como la expresión que forma el lenguaje que se pide de acuerdo a la descripción dada

El lenguaje es formado por una cantidad de simbolos a seguidos por una cantidad de simbolos b teniendo mínimo cero datos.

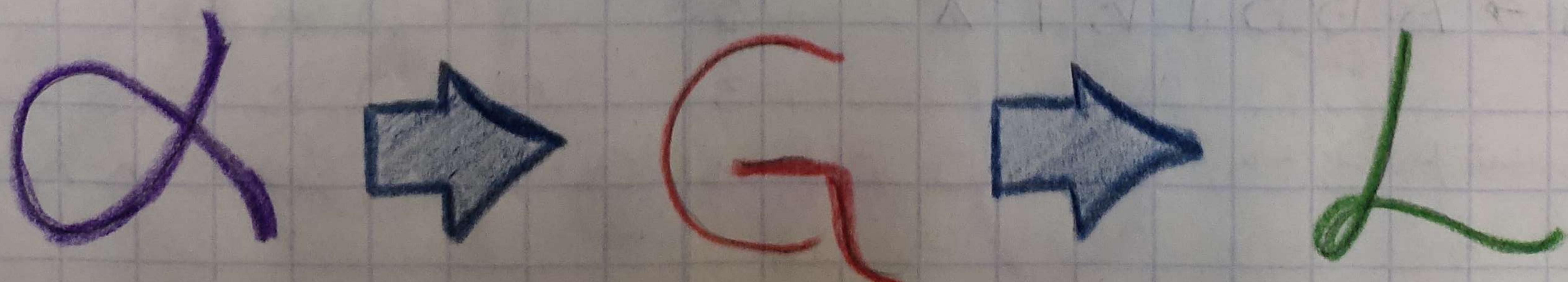
Para finalmente ser expresado por la gramática que genera dicho lenguaje siendo

n	
0	λ
1	aab
2	aabb
3	aaabb
4	aaaabb

Gramáticos independientes de contexto
y lenguajes formales

Un lenguaje formal está compuesto por una gramática formal la cual tiene parámetros específicos sin importar el contenido anterior, por lo que para una gramática que defina un lenguaje formal no importa el contexto cotidiano como si puede estar representado en los lenguajes informales teniendo términos coloquiales y contextualización por el entorno que rodea al receptor, los lenguajes formales tienen las restricciones necesarias para volver al lenguaje concreto evitando así las ambigüedades, en resumen el contexto está más relacionado con los lenguajes naturales y por consiguiente con los PLN, teniendo en los lenguajes formales contextos diferentes como la lógica matemática, las ciencias de la computación y la lingüística teórica para obtener formulas bien formadas.

Alfabeto → Gramática → Lenguaje



Árboles de Derivación

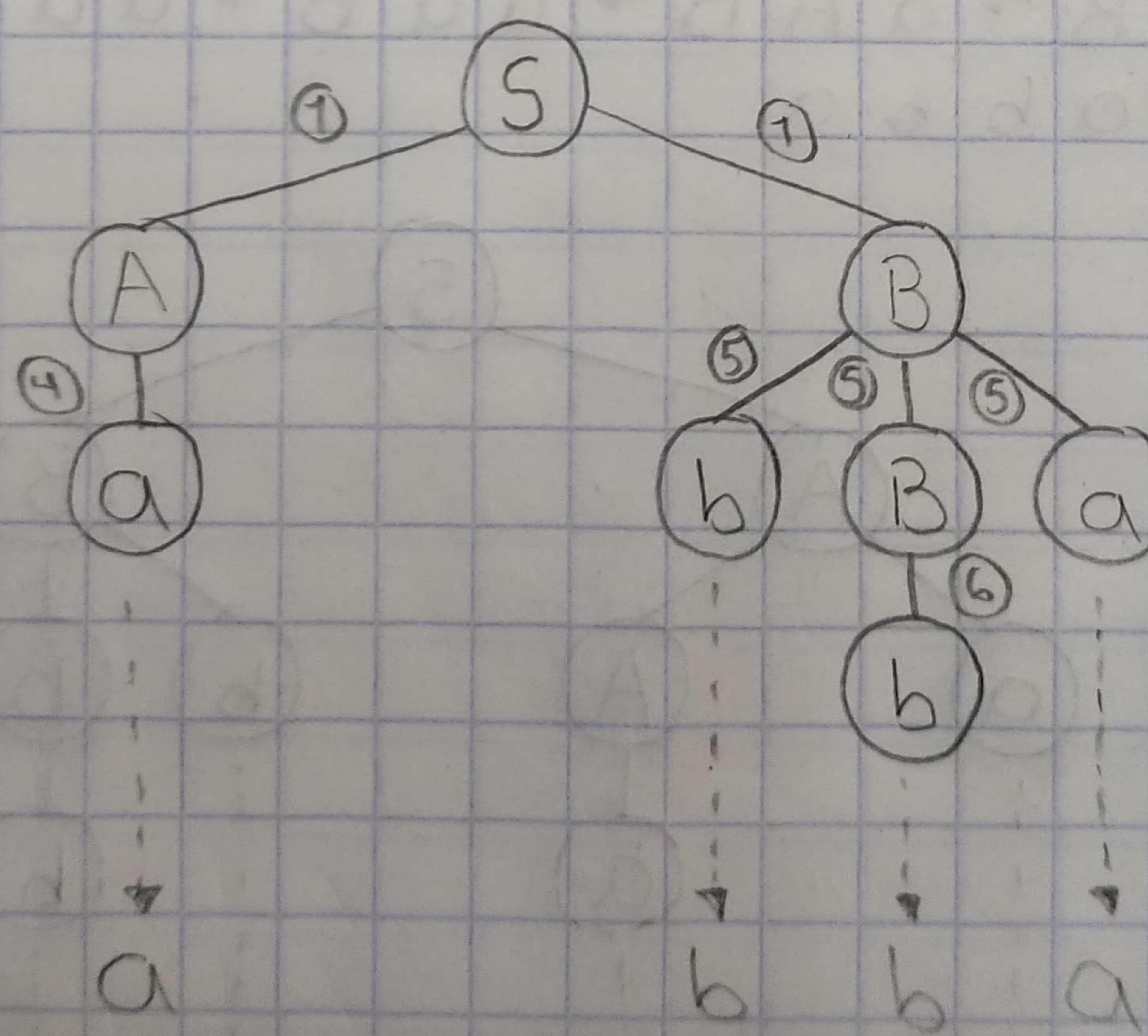
Ayudan a representar de manera gráfica el recorrido de las derivaciones resultantes de las reglas de producción aplicadas para formar una cadena, a continuación se muestra como formar árboles de derivación a partir de las cadenas de una gramática

$$G = (\{A, B, S\}, \{a, b\}, P, S)$$

$$P = \{ \begin{array}{ll} ① S \rightarrow A B ; & ② S \rightarrow A a B \\ ③ A \rightarrow a A ; & ④ A \rightarrow a \\ ⑤ B \rightarrow b B a ; & ⑥ B \rightarrow b \end{array} \}$$

① ahba

$$S \xrightarrow{1} A B \xrightarrow{5} A b B a \xrightarrow{4} a b B a \xrightarrow{6} a b b a$$

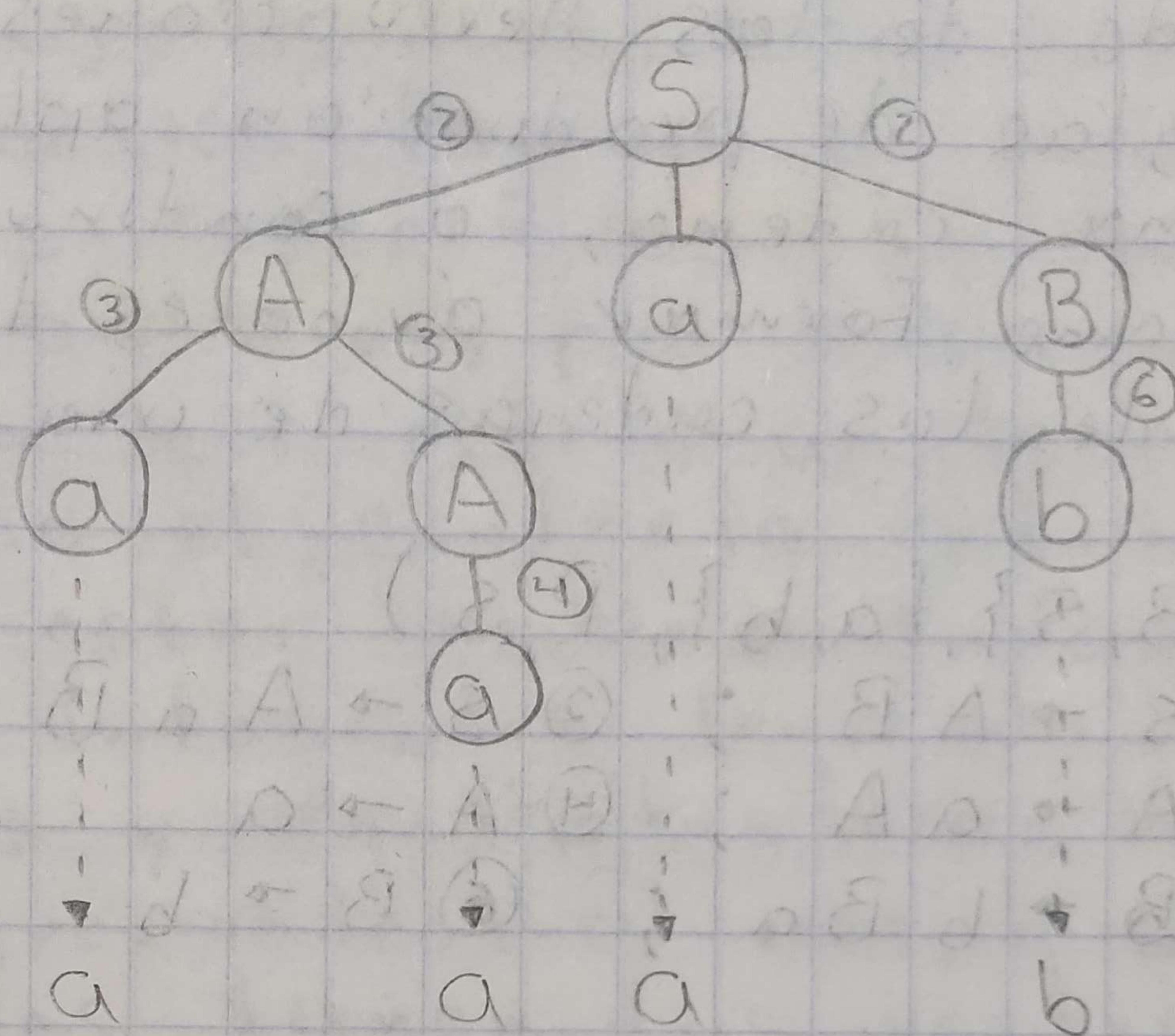


Terminales ✓

✓ ✓ ✓ ✓

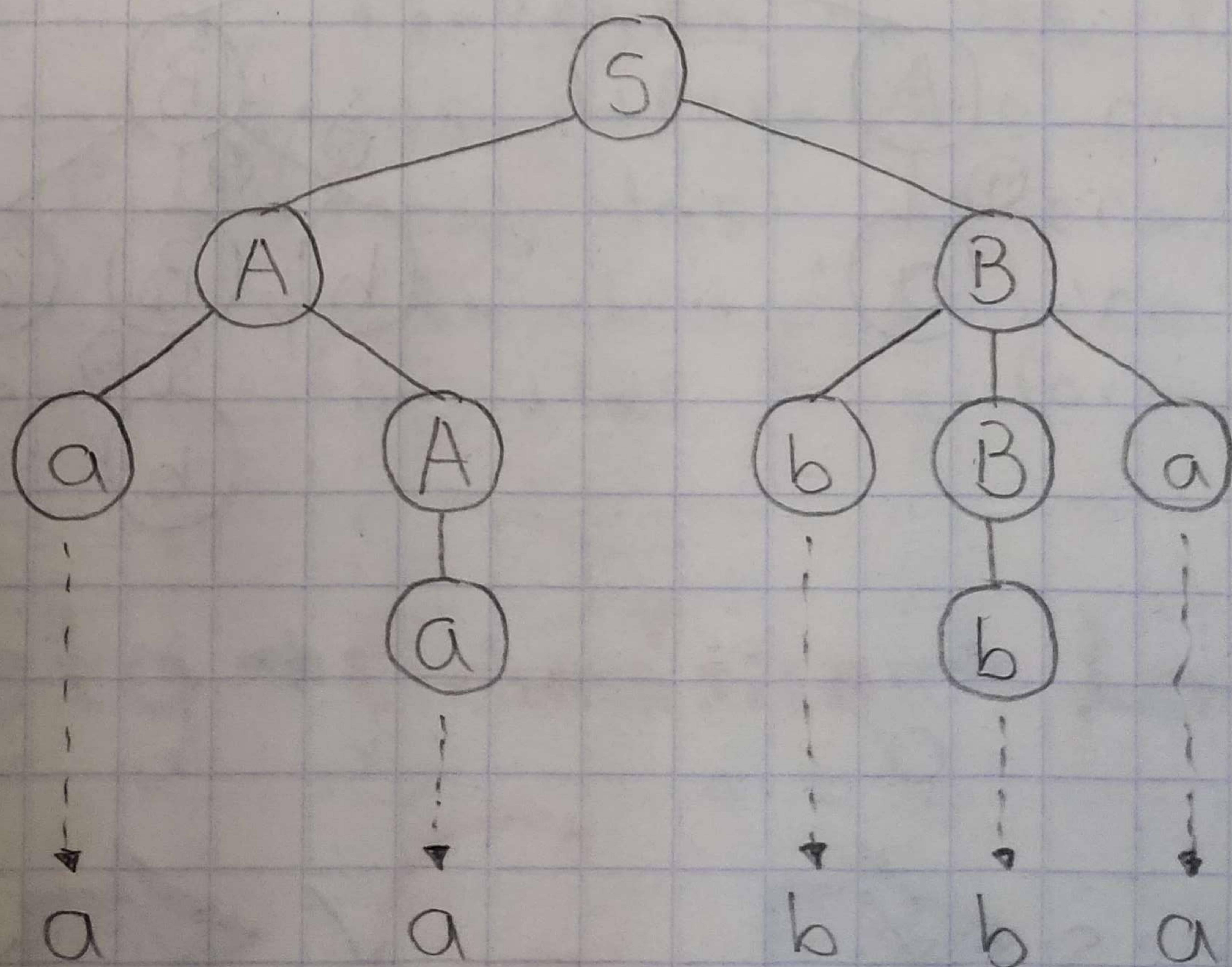
② aaab

$S \xrightarrow{2} AaB \xrightarrow{3} aAaB \xrightarrow{4} aaab \xrightarrow{6} aaab$



③ aabbba

$S \xrightarrow{1} AB \xrightarrow{3} aAB \xrightarrow{4} aaB \xrightarrow{5} aabBa$
 $\xrightarrow{5} aabbba$



Gramáticas libres de contexto

Una gramática libre de contexto es una gramática formal, también conocida como gramática de tipo 2 o gramática independiente del contexto, es una gramática la cual sigue una regla de producción, que consiste en que un símbolo no terminal (v) es seguido de una cadena de símbolos terminales o no terminales (w), dando como resultado un lenguaje, también podemos identificar las gramáticas libres de contexto porque siempre en la parte izquierda de las producciones solo aparece una variable.

Se le llama gramática libre de contexto porque el símbolo no terminal (v) puede ser siempre sustituido por la cadena de símbolos terminales o no terminales (w) sin importar el contexto en el que esto ocurra.

Como las demás gramáticas, la gramática libre de contexto está compuesta por 4 elementos:

1. $T \cup V_t$: Símbolos terminales.

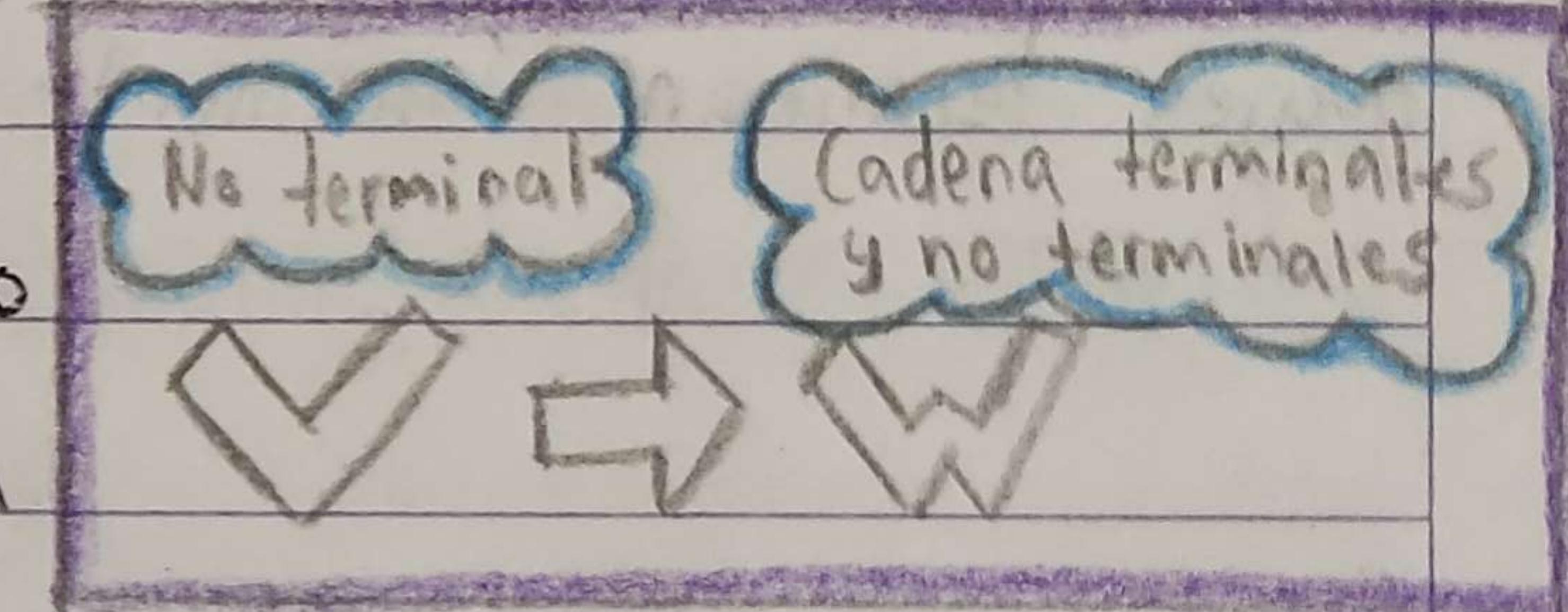
2. $V \cup V_n$: Símbolos no terminales.

3. P : Conjunto de producciones.

4. S : Símbolo inicial.

Estos elementos nos dan como resultado la gramática ($G = (V, T, P, S)$) o también denotada como ($G = \langle V_n, V_t, P, S \rangle$).

El lenguaje generado por las gramáticas libres de contexto es denotado como $L(G)$ y lleva por nombre Lenguaje independiente del contexto (LLC) o también conocido como Lenguaje libres del contexto. Los lenguajes libres del contexto son aquellos lenguajes que pueden ser reconocidos por



La notación más fre-

cientemente utilizada

para expresar gramáticas

libres de contexto es la

forma Backus-Naur.

un autómata de pila determinista o no determinista.

Estos lenguajes llevan como leyenda "Un lenguaje formal es libre de contexto si hay una gramática libre de contexto que lo genera".

Símbolos terminales.

Los símbolos terminales son aquellos símbolos los cuales no pueden ser cambiados por ninguna regla de producción, pero si pueden aparecer como resultado de estas reglas. Las salidas de las cadenas generadas en los lenguajes terminan únicamente con símbolos terminales. Ejemplo:

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

Símbolos no terminales.

Los símbolos no terminales son aquellos símbolos los cuales se pueden reemplazar por medio de producciones, ya sea por otro símbolo no terminal o un símbolo terminal.

Estos símbolos también son conocidos como variables sintácticas. Ejemplo:

$$\underline{A} \rightarrow \underline{AB} \quad \underline{A} \rightarrow \underline{C} \quad \underline{B} \rightarrow \underline{b}$$

Producción.

Una producción se puede definir como una reescritura o reemplazo que indica como sustituir a los símbolos no terminales por alguna expresión o símbolos, estos símbolos son definidos en la parte derecha de las producciones. Las producciones son utilizadas para generar o analizar las cadenas. Ejemplo:

$$\underline{A} \rightarrow \underline{b} \quad \underline{A} \rightarrow \underline{c} \quad \underline{A} \rightarrow \underline{AB}$$

Símbolo inicial.

El símbolo inicial es un símbolo no terminal con el cual se empieza a aplicar las reglas de la gramática para así ir generando las distintas cadenas del

Un autómata de pila es una configuración concreta de las sucesivas, por las que para el autómata durante el procesamiento de una cadena.

lenguaje. Ejemplo:

$$\underline{S} \rightarrow AB$$

$$\underline{S} \rightarrow D$$

Convenciones de la gramática.

Las convenciones de la gramática son alguna de las reglas que se tienen que respetar para así hacer el uso correcto de la gramática.

1. Las letras mayúsculas (A, B, C, D, E, S) representan símbolos no terminales, también conocidas como variables.
2. Las letras minúsculas (a, b, c, d, e), dígitos o expresiones encerradas entre comillas simples ('') representan símbolos terminales.
3. X, Y, Z, W representan cadenas de símbolos terminales, también conocidas como frases.
4. Las letras griegas minúsculas (α, β, γ) representan formas de frases.

Derivación.

La gramática libre de contexto consta de un proceso llamado derivación, el cual trata de determinar si una palabra pertenece a un lenguaje y también a determinar la estructura sintáctica de esta. En cada paso de la derivación se puede aplicar las reglas de producción en cualquier orden, pero antes de esto se debe elegir que símbolo no terminal reducir y elegir que alternativa de las reglas de producción usar en caso de que tenga más de una alternativa.

Existen 2 tipos de derivación, la derivación más a la izquierda y la derivación más a la derecha. La derivación más a la izquierda trata de escoger el símbolo no terminal más a la izquierda mientras que la derivación más a la derecha trata de escoger el símbolo no terminal más a la derecha. No existe alguna regla que nos indique cuál de estos 2 tipos de derivación utilizar, entonces, ya queda a decisión de la persona cuál de estos 2 tipos es mejor de utilizar.

Ejemplo: Verificar que la cadena abab pertenezca al lenguaje por medio de la derivación más a la izquierda.

$$G = (V_n, V_t, S, P)$$

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

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

$$P: S \rightarrow A$$

$$A \rightarrow AA \mid a \mid b \mid AB$$

$$B \rightarrow b$$

	Cadena	Producción	Derivación
S	S	$S \rightarrow A$	A
A	A	$A \rightarrow AA$	AA
AA	AA	$A \rightarrow AA$	AAA
AAA	AAA	$A \rightarrow AB$	ABAA
ABAA	ABAA	$A \rightarrow a$	aBAA
aBAA	aBAA	$B \rightarrow b$	abAA
abAA	abAA	$A \rightarrow a$	abaA
abaA	abaA	$A \rightarrow b$	<u>abab</u>

Con el resultado final de la tabla de derivación podemos comprobar que efectivamente la cadena abab pertenece al lenguaje. Este resultado debería ser el mismo sin importar que utilicemos el método de la derivación más a la izquierda o la derivación más a la derecha, lo único que si puede cambiar es el procedimiento.

Se puede presentar algunos problemas cuando se hace uso de la derivación, uno de ellos es que es posible encontrar más de una derivación, ya sea como por la izquierda o como por la derecha, en estos casos se dice que la gramática es ambigua.

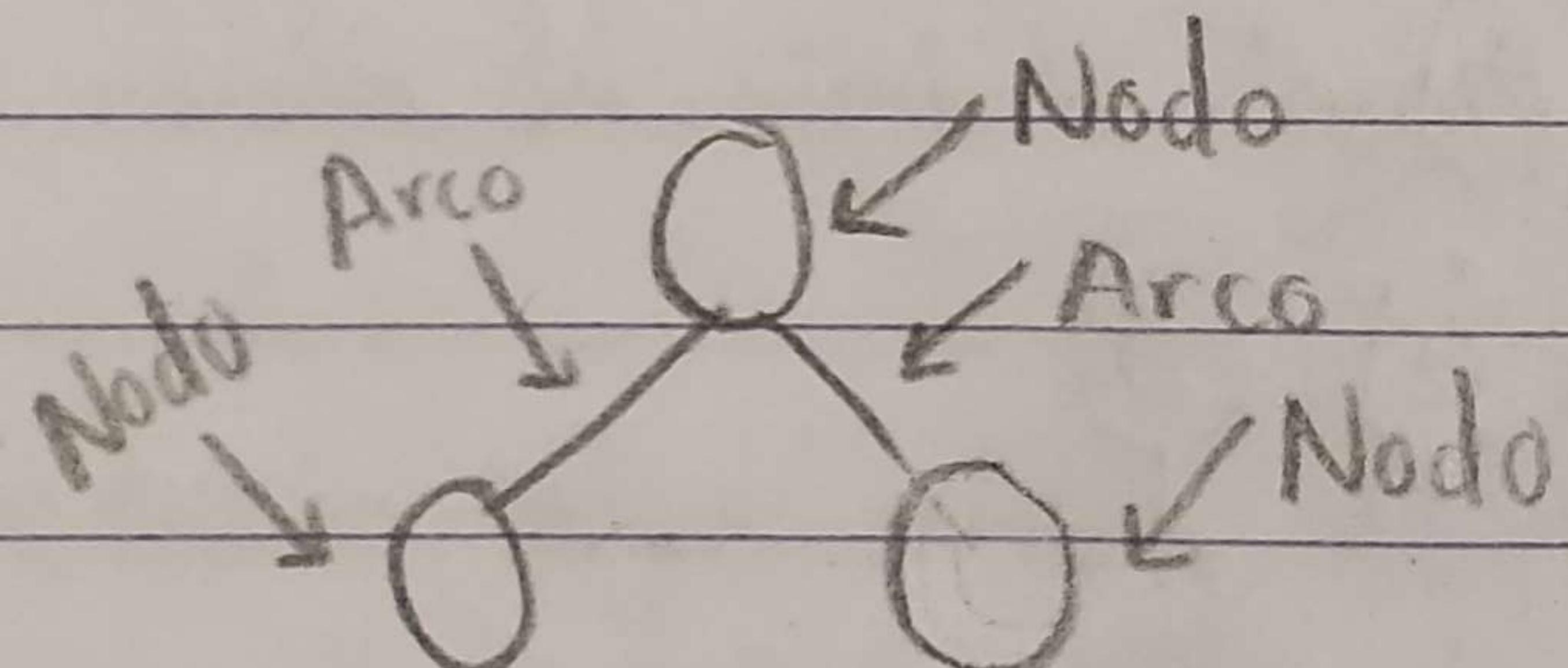
Para definir la estructura sintáctica por medio de la derivación se construye el árbol de derivación, el cuál es construido a partir de la cadena de derivaciones.

La ambigüedad significa que una expresión del lenguaje puede tener más de una interpretación, lo cual no está permitido.

Arbol de derivación

Un árbol de derivación es una estructura que permite ilustrar como se puede衍生 una cadena generada de un lenguaje, a partir de los símbolos de una gramática. Estos árboles son utilizados en la construcción de compiladores para representar el análisis sintáctico de los programas fuente y sirviendo como base para la generación del código. Los árboles de derivación también son conocidos como árbol de análisis, árbol de análisis gramatical o árbol de análisis sintáctico.

Un árbol es un conjunto de nodos, los cuales están unidos por líneas, que a su vez estas líneas son llamadas arcos. Los arcos son los encargados de conectar dos distintos nodos.

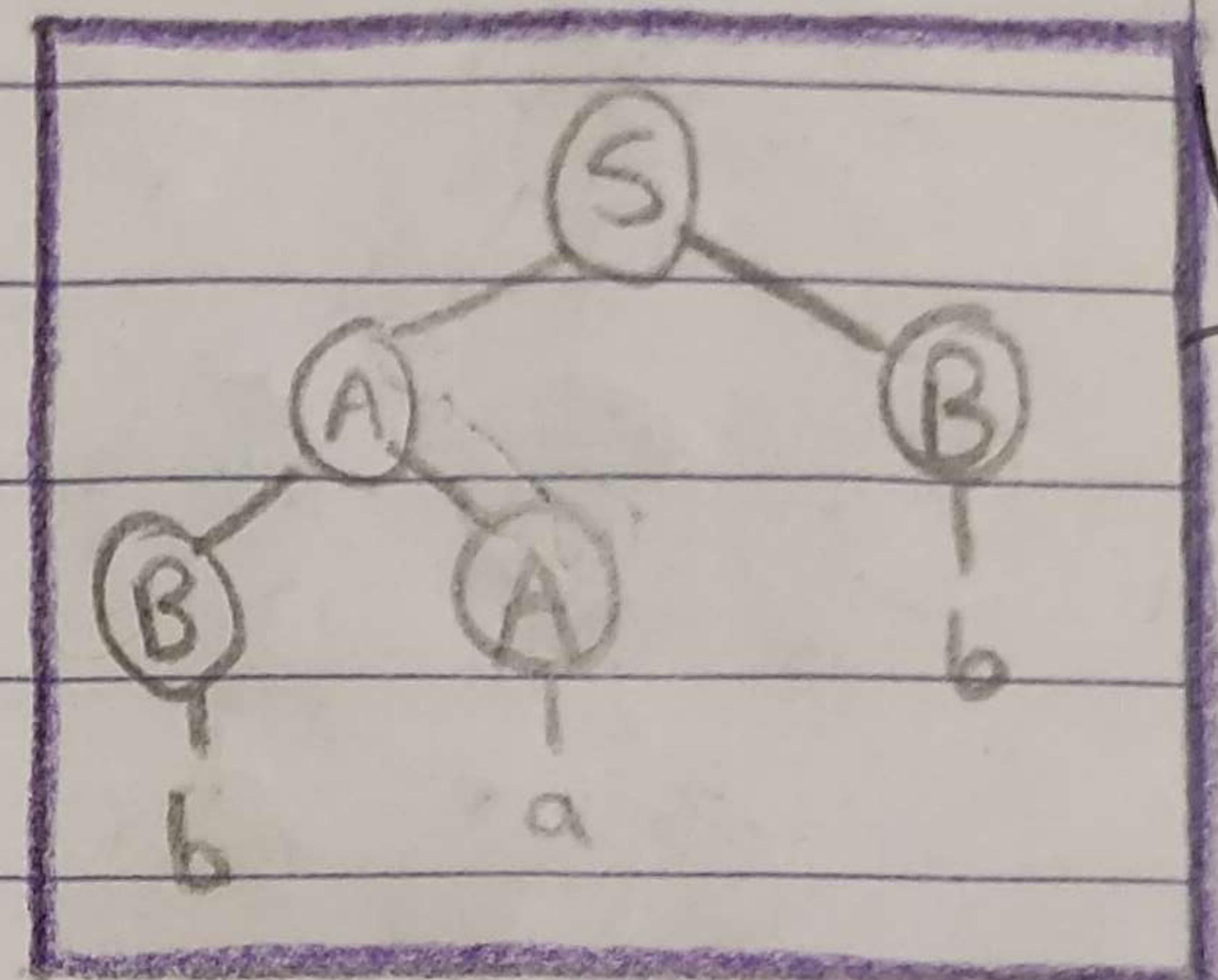


La cadena de símbolos terminales que se obtiene al recorrer las hojas del árbol en orden es llamado producto de árbol.

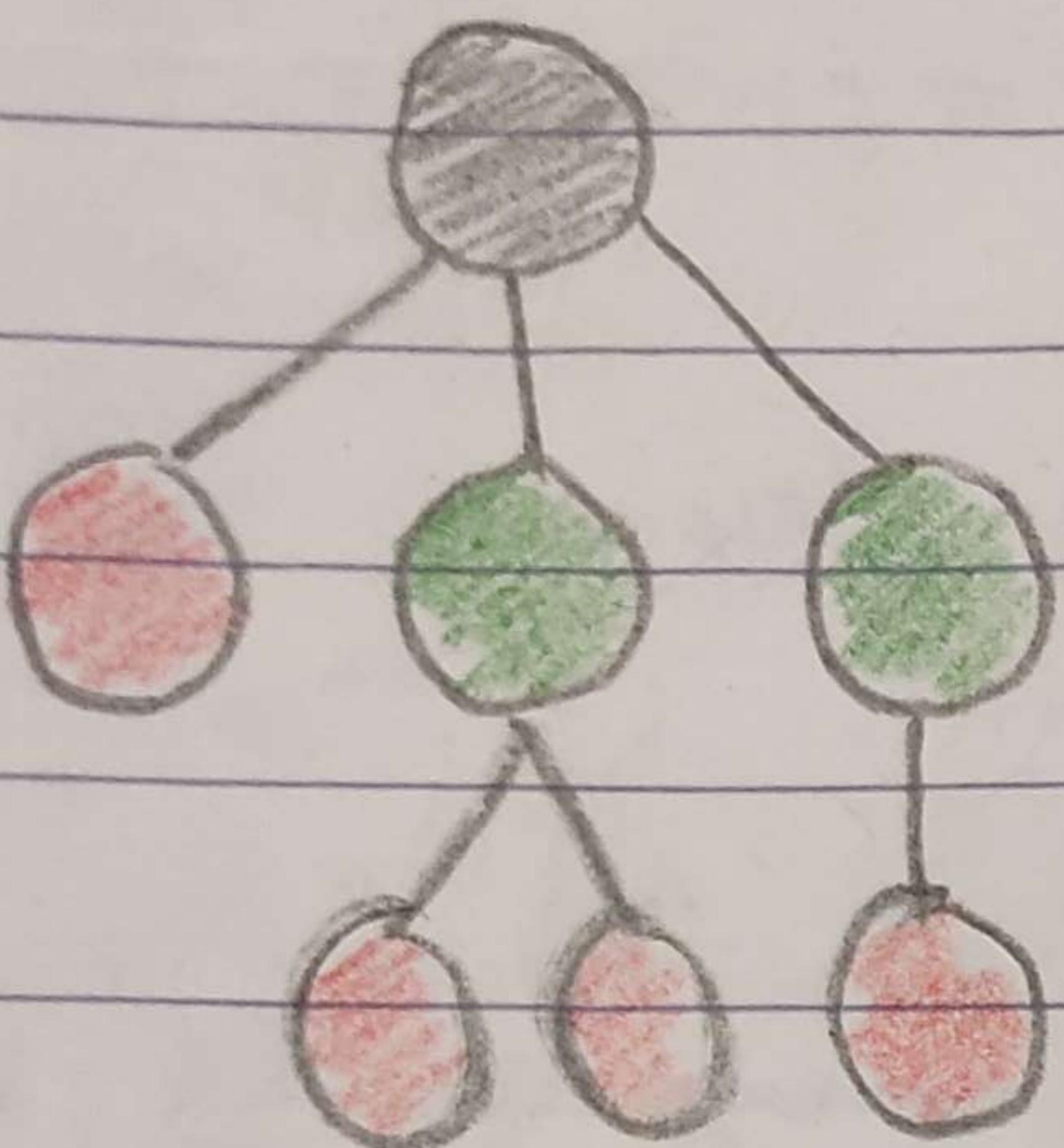
Propiedades

Las propiedades del árbol de derivación son:

- El nodo raíz está ubicado al principio del árbol y es el símbolo inicial de la gramática.
- Cada hoja del árbol corresponde a un símbolo terminal, incluyendo la cadena vacía (ϵ) solo si son hijos únicos.
- Cada nodo interior del árbol corresponde a un símbolo no terminal.
- Todos los nodos se conforman por un símbolo de la gramática.



- Si los hijos de un nodo interior A son X_1, X_2, \dots, X_n , entonces, $A \rightarrow X_1 X_2 \dots X_n$ es una regla de derivación.



• Nodo raíz
 • Nodo interior
 • Nodo hoja

A continuación se hará muestra de un ejemplo de un árbol de derivación.

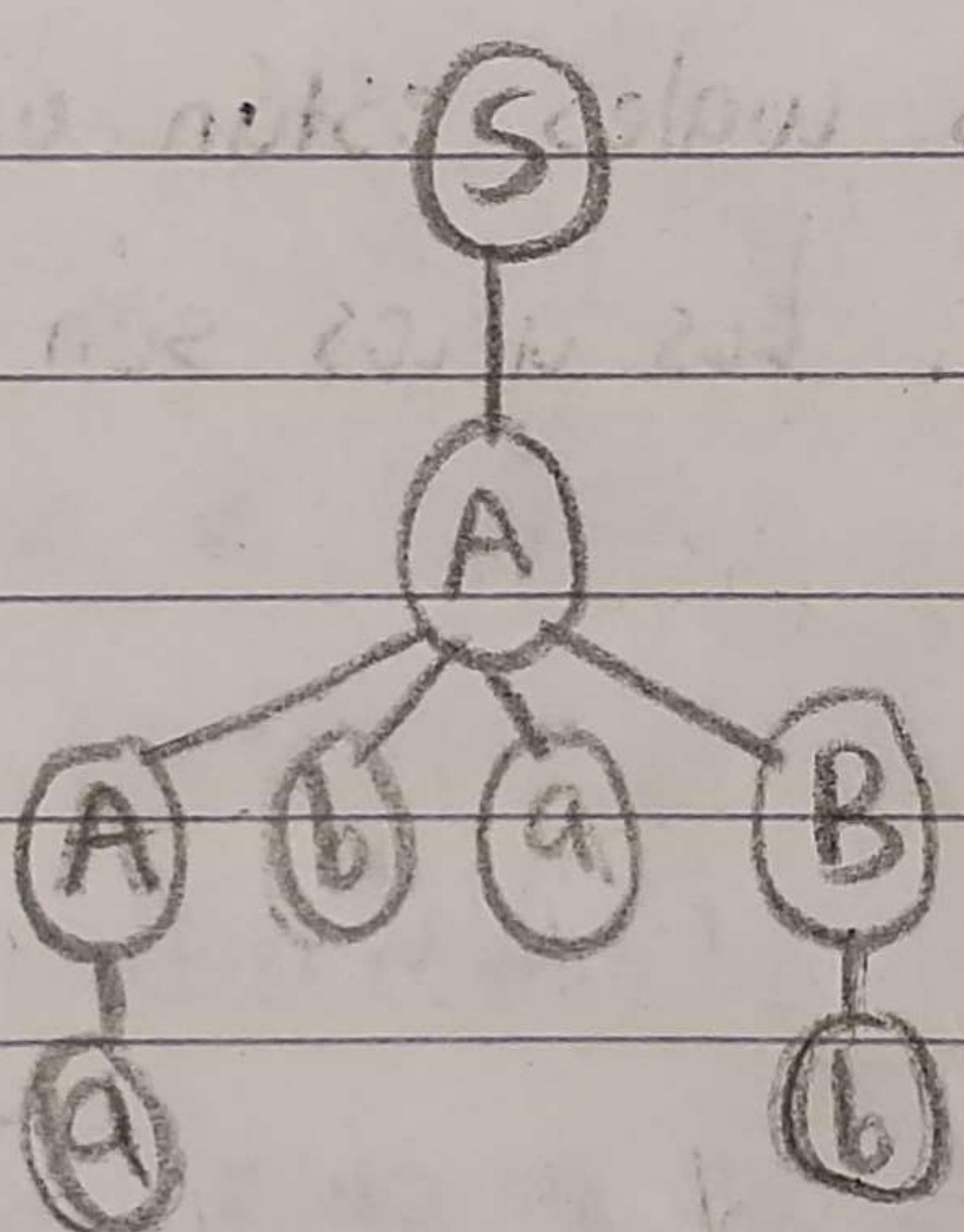
La cadena abab tiene la siguiente derivación:

$$S \rightarrow A \rightarrow AA \rightarrow AAA \rightarrow ABAA \rightarrow aBAA \rightarrow abAA \rightarrow abaA \rightarrow abab$$

P: $S \Rightarrow A$

$$A \rightarrow AA \text{ | } aB \text{ | } bB$$

$$B \rightarrow b$$



Cómo podemos comprobar el producto del árbol nos da como resultado la cadena abab que es la misma cadena que obtenemos al realizar la derivación.

Ambigüedad en árboles de derivación

Cuando se obtienen distintos árboles de derivación para una misma cadena, se dice que la gramática es ambigua, al contrario, cuando solo se obtiene un árbol de derivación para una cadena entonces no es ambigua.

Para eliminar la ambigüedad en una gramática se requiere de un proceso de análisis, esto para verificar que no se pueda generar más de un árbol de derivación en cada cadena. Algunos tips para eliminar la ambigüedad es utilizar nuevas variables que simplifiquen el proceso y así se obtengan las mismas cadenas pero

eliminando los problemas de derivación.

Diagramas de sintaxis

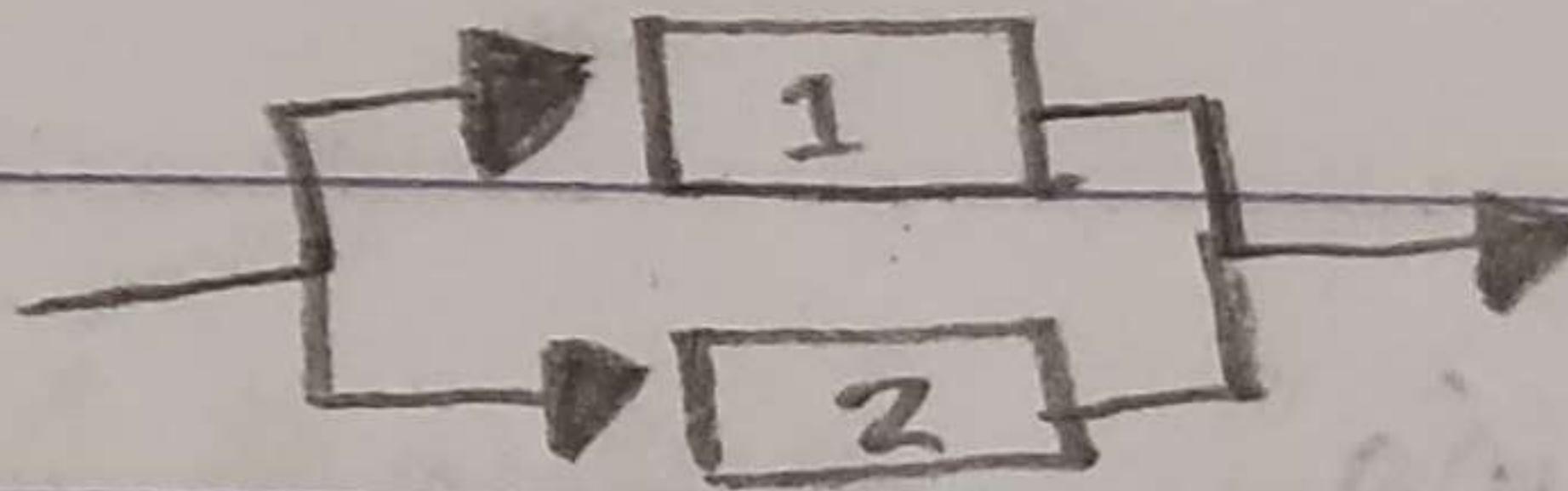
Los diagramas de sintaxis son una forma de representar las gramáticas libres de contexto, también son conocidas como diagramas sintácticos o diagramas de ferrocarril.

Estos diagramas tienen como elementos una serie de cajas o bien vistos como símbolos geométricos conectados por arcos dirigidos. La construcción de estos diagramas se basan en reglas, estas reglas son:

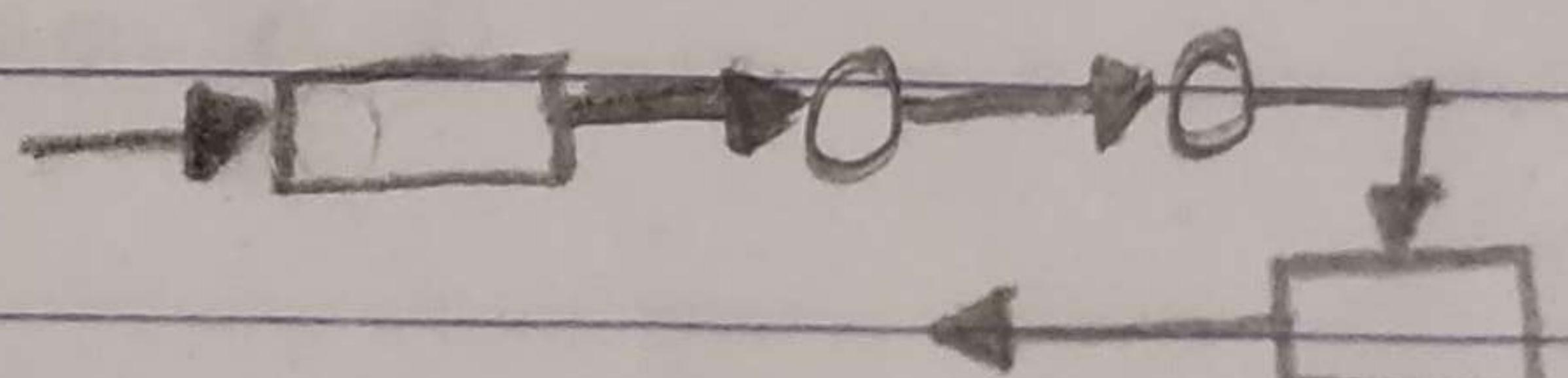
1. Cada símbolo terminal se representa con un círculo y el nombre del símbolo o símbolo dentro de este círculo. Terminal

2. Cada símbolo no terminal se representa con un rectángulo y el nombre del símbolo dentro del rectángulo. No terminal

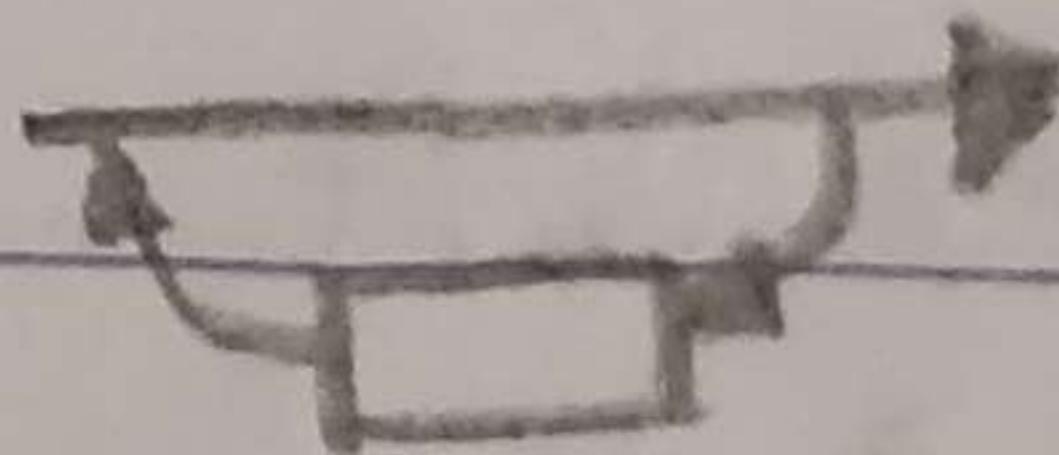
3. Cuándo una producción tenga varias alternativas, se deberá representar de la siguiente manera en el digrama.



4. Cuándo una producción tenga una concatenación de símbolos, basta con representarla de la siguiente manera en el diagrama.



5. Para las producciones que realicen un bucle, es decir, que tengan cero, una o más repeticiones de un símbolo se representara de la siguiente manera en el diagrama.



Cómo se puede notar las reglas para la construcción no son difíciles de entender ya que estas reglas no pueden cambiar, además de que estos diagramas de sintaxis son más fáciles de comprender para los usuarios, ya que, se pueden ver las

sustituciones de forma dinámicas y esto hace que el usuario entienda más el funcionamiento de las producciones de la gramática.

Una regla de oro en los diagramas es que el

inicio o final del diagrama no es señalizado,

pero siempre se sabe que el origen siempre es

la figura más a la izquierda mientras que el destino es la figura más a la derecha.

Otro aspecto a favor de los diagramas de sintaxis es que las operaciones básicas de BNF (yuxtaposición, opción y repetición) son muy amigables y fáciles de representar, a continuación serán mostrados algunos ejemplos de la representación de estas operaciones en el diagrama.

Yuxtaposición $A|B|C = \rightarrow [A] \rightarrow [B] \rightarrow [C]$

Opción $A|B = \rightarrow [A] \rightarrow [B]$ $\epsilon|B = \rightarrow [B]$

Repetición 1 o más $[A]$ $\rightarrow [A] \rightarrow [A]^*$

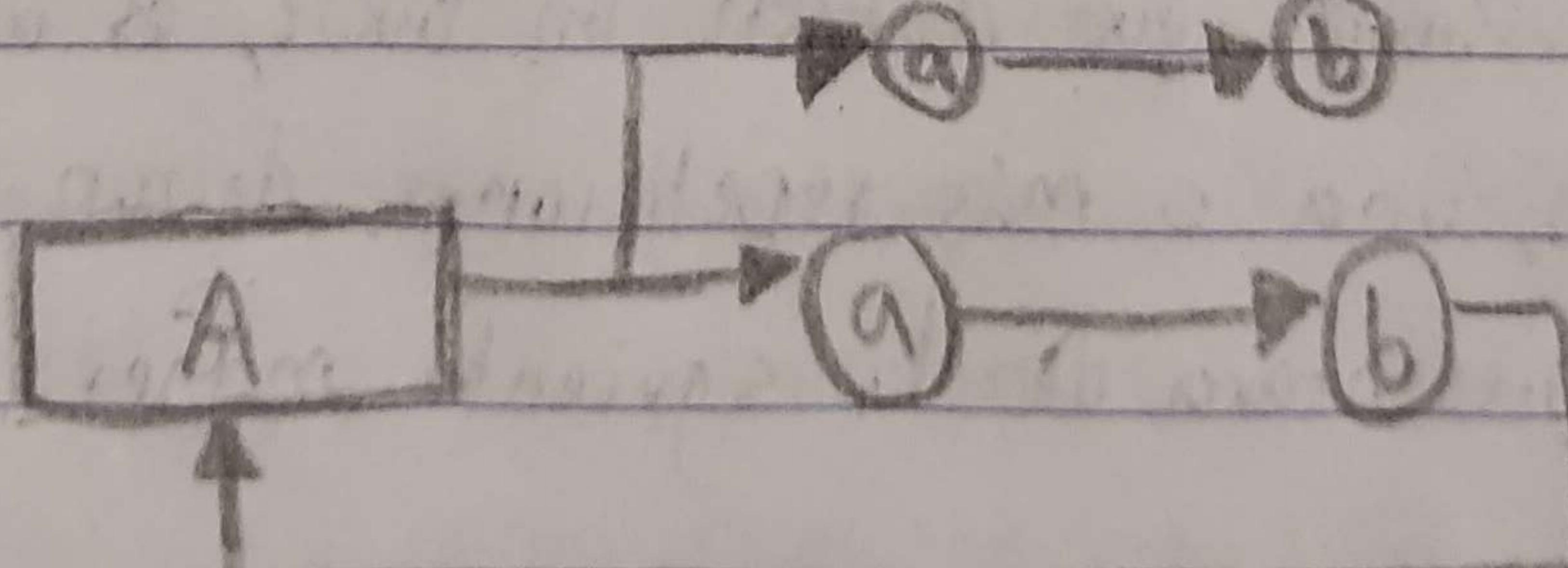
0 o más $[A]$ $\rightarrow [A]^*$

Ejemplo: Se realizará un diagrama de sintaxis de la producción:

$$\rho: S \rightarrow A$$

$$A \rightarrow abA$$

$$A \rightarrow ab$$



Bibliografía

Universidad de Valladolid (s.f.). El generador de analizadores léxicos lex.
Recuperado el 21 de Septiembre de 2022

<https://www.infor.uva.es/~mluisa/talf/docs/lab0/L3.pdf>

Unicen. (s.f.). Gramáticas libres de contexto
Unicen.edu. Recuperado el 21 de Septiembre de 2022

<https://users.exa.unicen.edu.ar/catedras/ccomp1/Apunte5.pdf>

Contexto en comunicación - Información, importancia y ejemplos (s.f.). Concepto Recuperado el 21 de Septiembre de 2022

<https://concepto.de/contexto-en-comunicacion/>

Pedro Alvarez & Rubén Bejar (2004). Gramáticas libres de contexto. Recuperado el 21 de Septiembre de 2022

<https://es.wikidat.com/info/gramatica-libre-de-contexto>

Buap (s.f.). Gramáticas libres de contexto
Recuperado el 21 de Septiembre de 2022

https://gaz.wiki/wiki/es/Terminal_and_nonterminal_symbols#:~:text=S%C3%ADmbolos%20terminales%20y%20no%20terminales%201%20S%C3%ADmbolos%20terminales,3%20Reglas%20de%20producci%C3%B3n%20...%204%20Ejemplo%20