



## DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

Celaya, Guanajuato, 24 / febrero / 2025

ASUNTO: **SOLICITUD DE ACTIVIDADES**

### LENGUAJES Y AUTÓMATAS II

DOCENTE DESIGNADO: ISC. RICARDO GONZÁLEZ GONZÁLEZ  
**SEMESTRE ENERO-JUNIO 2025**

#### ACTIVIDAD 4 (VALOR 44 PUNTOS)

LEA CUIDADOSAMENTE, Y REALICE LAS SIGUIENTES 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 PROponer 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 FLEX**, 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.





### ¿ 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.



## **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, SUPONDRÍA 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.





**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	: LAII, 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 PERÍODO SEMESTRAL EN CURSO: <b>ENE-JUN</b>

EJEMPLO :

SI EL TRABAJO SE SOLICITÓ EN EQUIPO.

2025-02-24\_TNM\_CELAYA\_LAII-A\_A4\_EQUIPO\_99\_9999999\_PEREZ\_PEREZ\_JUAN\_ENE-JUN25.PDF

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

SI EL TRABAJO SE SOLICITÓ INDIVIDUALMENTE.

2025-02-24\_TNM\_CELAYA\_LAII-A\_A4\_9999999\_PEREZ\_PEREZ\_JUAN\_ENE-JUN25.PDF





**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. DESPUES DE LA HORA INDICADA EN LA PLATAFORMA VIRTUAL ( AUN 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.



# ACTIVIDAD 4

análisis  
léxico y sintáctico

21030223 Macías Sevilla Diana Natasha

21030104 Castañeda Ruiz Cristian Eduardo

21031027 Aguilar Flores Abel

Profesor : Ricardo González González

Fecha : 03 de Marzo de 2025

semestre : Ene - Jun 2025

# Análisis léxico

Para recordar, un análisis léxico es un proceso que consiste en dividir el código fuente de un programa en unidades básicas llamadas tokens. Estos pueden ser palabras reservadas, identificadores, números, operadores, etc.

## — GENERADORES DE ANALISADORES LÉXICOS —

Son herramientas fundamentales en el desarrollo de compiladores e intérpretes. Estas herramientas permiten automatizar el proceso de creación de analizadores léxicos.

### Importancia

- Eficiencia: permiten crear analisadores eficientes que puedan procesar grandes cantidades de código rápidamente.
- Flexibilidad: suelen basarse en expresiones regulares lo que facilita la definición de reglas léxicas.
- Integración: pueden unirse fácilmente a otras partes.

En esta monografía veremos algunos de los más destacados:

# JTLEX

JTlex permite la especificación conjunta de sintaxis y semántica, lo que permite computar

atributos, utilizando expresiones regulares traductoras que introduce un nuevo formalismo que permite reconocer simbolos para formar atributos.

El diseño e implementación están orientados a objetos facilitando la integración con proyectos modernos.

Ej.: para un nombre de usuario: `/^[\w{3,16}]$/`

# FLEX

Flex produce código en C. Es ampliamente usado en UNIX y otros sistemas

Produce analisadores

lexicos que son generalmente mas rápidos que los de Lex, gracias a optimizaciones en la generación de código y en los algoritmos de búsqueda de patrones. Está diseñado para ser compatible con Lex, así que se puede migrar de Lex a Flex fácilmente.

```
#define TOKEN_IN 1  
#define TOKEN_OUT 2  
%  
"in" {printf ("Token entrada");  
return TOKEN_IN;}  
"Salida" {print ("Token Salida");}
```

# JFLEX

JFLEX es un generador de analisadores léxicos para Java, conocido por su eficiencia y facilidad

guillermo  
2023

de uso. Utiliza expresiones regulares para definir reglas léxicas y genera automáticamente el código del analizador léxico en Java.

Se basa en automátomas finitos deterministas (DFAs).

Diseñado con la capacidad de adaptarse a proyectos Java existentes.



# LEX

LEX está incluido originalmente en el ambiente de desarrollo UNIX, genera código fuente C a partir de una serie

de especificaciones escritas en el lenguaje lex.

Ese código contiene siempre una función `yylex()`

Permite definir la sintaxis de los símbolos mediante expresiones regulares.

Aunque FLEX es más moderno y eficiente, LEX sigue siendo compatible con muchos sistemas y puede ser preferido en ciertos contextos



# PRÁCTICA DE LABORATORIO

CARRERA	NOMBRE DE LA ASIGNATURA
INGENIERIA EN SISTEMAS COMPUTACIONALES	Lenguajes y autómatas II.

AUTORES		
• AGUILAR FLORES ABEL	21031027	
• CASTAÑEDA PÉREZ CRISTIAN EDUARDO	21030140	
• MACÍAS SEVILLA DIANA NATASHA	21030223	

PRACTICA NO.	NOMBRE DE LA PRACTICA	DURACIÓN (HORAS)
4	Actividad-4. Generadores de analizadores léxico.	3 hora(max)

## INTRODUCCIÓN

En distintos lenguajes de programación es necesario la compilación para poder generar un código objeto, de esta manera el código podrá ser procesado por la máquina. Durante el proceso de compilación pasa por varias etapas dentro de las que se encuentra en análisis léxico y análisis sintáctico.

El análisis léxico es la primera fase en el procesamiento de un lenguaje. Su propósito es leer el código fuente y dividirlo en tokens, que son las unidades mínimas significativas del lenguaje, como palabras clave, para esta práctica se lleva a cabo con las siguientes palabras clave: while, for y if ; además los analizadores léxicos también corresponden a los tokens de operadores (+,-,\*,/ ) y otros símbolos de puntuación ( {,}, ; ).

Para realizar esta tarea de manera eficiente, se emplea Flex, una herramienta que genera analizadores léxicos basados en

# PRÁCTICA DE LABORATORIO

expresiones regulares. Además de la realización con un ejemplo con YACC, el cual permite que FLEX solo devuelva los Tokens mientras que YACC realiza la segunda fase del proceso de compilación, el análisis sintáctico.

## OBJETIVO

Realizar un ejercicio demostrativo sobre al menos tres definiciones léxicas propias del lenguaje C, con la ayuda del Software Flex.

## FUNDAMENTO

Flex es una herramienta que permite generar analizadores léxicos. A partir de un conjunto de expresiones regulares, Flex busca concordancias en un fichero de entrada y ejecuta acciones asociadas a estas expresiones. Es compatible casi al 100% con Lex, una herramienta clásica de Unix para la generación de analizadores léxicos, pero es un desarrollo diferente realizado por GNU bajo licencia GPL.

Los ficheros de entrada de Flex (normalmente con la extensión .l) siguen el siguiente esquema:

# PRÁCTICA DE LABORATORIO

%%

patrón1 {acción1}

patrón2 {acción2}

...

donde:

patrón: expresión regular

acción: código C con las acciones a ejecutar cuando se encuentre concordancia del patrón con el texto de entrada

Flex lee los ficheros de entrada dados, o la entrada estándar si no se le ha indicado ningún nombre de fichero, con la descripción de un escáner a generar. La descripción se encuentra en forma de parejas de expresiones regulares y código C, denominadas reglas. Flex genera como salida un fichero fuente en C, 'lex.yy.c', que define una función 'yylex()'. Este fichero se compila y se enlaza con la librería de Flex para producir un ejecutable. Cuando se arranca el fichero ejecutable, este analiza su entrada en busca de casos de las expresiones regulares. Siempre que encuentra uno, ejecuta el código C correspondiente.

El fichero de entrada de Flex está compuesto de tres secciones, separadas por una línea donde aparece únicamente un '%%' en esta:

definiciones

%%

reglas

%%

código de usuario

La sección de reglas en la entrada de Flex contiene una serie de reglas de la forma: patrón acción.

# PRÁCTICA DE LABORATORIO

## Patrones

Los patrones en la entrada se escriben utilizando un conjunto extendido de expresiones regulares y usando como alfabeto cualquier carácter ASCII. Cualquier símbolo excepto el espacio en blanco, tabulador, cambio de línea y los caracteres especiales se escriben tal cual en las expresiones regulares (patrones) de Flex.

- x empareja el carácter 'x'
- . cualquier carácter excepto una línea nueva
- [xyz] un conjunto de caracteres; en este caso, el patrón empareja una 'x', una 'y', o una 'z'
- [abj-oZ] un conjunto de caracteres con un rango; empareja una 'a', una 'b', cualquier letra desde la 'j' hasta la 'o', o una 'Z'
- [^A-Z] cualquier carácter menos los que aparecen en el conjunto. En este caso, cualquier carácter EXCEPTO una letra mayúscula.
- [^A-Z\n] cualquier carácter EXCEPTO una letra mayúscula o una línea nueva.
- r\* cero o más r's, donde r es cualquier expresión regular
- r+ una o más r's
- r? cero o una r (es decir, "una r opcional")
- r{2,5} entre dos y cinco concatenaciones de r
- r{4} exactamente 4 r's
- {nombre} la expansión de la definición de "nombre" (ver más abajo) "[xyz]\\"foo" la cadena literal: [xyz]"foo
- \x si x es una 'a', 'b', 'f', 'n', 'r', 't', o 'v', entonces la interpretación ANSI-C de \x (por ejemplo \t sería un tabulador). En otro caso, un literal 'x' (usado para la concordancia exacta de caracteres especiales (\\\ . \?)

## PRÁCTICA DE LABORATORIO

- (r) empareja una R; los paréntesis se utilizan para anular la precedencia
- rs la expresión regular r seguida por la expresión regular s; se denomina "concatenación"
- r|s bien una r o una s
- r/s una r pero sólo si va seguida por una s.
- ^r una r, pero sólo al comienzo de una línea
- r\$ una r, pero sólo al final de una línea (es decir, justo antes de una línea nueva). Equivalente a "r/\n".
- <s>r una r, pero sólo en la condición de arranque s
- <s1,s2,s3>r lo mismo, pero en cualquiera de las condiciones de arranque s1, s2, o s3.

El funcionamiento de un analizador léxico se puede definir como un reconocedor de lenguaje, es decir, es un programa que toma como entrada una cadena y devuelve verdadero o falso en función de si la cadena pertenece o no al lenguaje. Los lenguajes regulares (descritos por gramáticas regulares o por expresiones regulares) pueden ser reconocidos por medio de Autómatas Finitos (Finite State Machines).

Si se necesita demostrar que para cada expresión regular, existe un autómata finito que acepta el mismo lenguaje. Se selecciona el autómata más apto. Si, por el contrario, se necesita demostrar que por cada autómata finito, hay una expresión regular definiendo su lenguaje se selecciona el autómata con mayores restricciones: DFA. Los lenguajes aceptados por DFA, NFA, €-NFA, RE son llamados lenguajes regulares.

El análisis sintáctico es la segunda fase en el procesamiento de un lenguaje y su objetivo es verificar que los tokens obtenidos del análisis léxico sigan una estructura válida de acuerdo con la gramática del lenguaje.

# PRÁCTICA DE LABORATORIO

Para lograr esto, se utiliza Yacc (Yet Another Compiler Compiler) o su versión moderna Bison, que permite definir gramáticas libres de contexto mediante reglas de producción. Mientras que Flex simplemente reconoce tokens, Yacc interpreta y estructura el código fuente, asegurando que siga las reglas del lenguaje de programación.

## REQUISITOS BÁSICOS

- Una computadora
- Instalación de Flex y Bison instalados
- Block de notas o algún editor de texto

## DESARROLLO

Para el desarrollo de esta práctica es necesario crear un archivo .lex o .yy.c. (Imagen 1)

```
nath@nath:~/flex$ nano actividad4.lex
```

Imagen 1. Creación de archivo .lex

El inicio del código solo se anexa la librería estándar stdio.h, el cual permite usar printf() para imprimir en pantalla (Imagen 2)

```
%{  
#include <stdio.h>  
%}  
%%
```

Imagen 2. Librería standar

Se describe a continuación el código donde se identifican 3 palabras claves: while, for y if. Además, también se reconoce identificadores, numero entero y en dado caso si es un punto un carácter desconocido. Esto se reconoce a través de expresiones regulares y las acciones que se ejecutara cuando se reconozca un patrón. En la imagen 3 se muestra el código de las expresiones

## PRÁCTICA DE LABORATORIO

regulares con su acción en este caso solo imprime si es una palabra, clave o si no se identifica. (Imagen 3)

```

while      { printf("PALABRA CLAVE: while\n"); }
for       { printf("PALABRA CLAVE: for\n"); }
if        { printf("PALABRA CLAVE: if\n"); }
[a-zA-Z_][a-zA-Z0-9_]*  { printf("IDENTIFICADOR: %s\n", yytext); }
[0-9]+    { printf("NÚMERO ENTERO: %s\n", yytext); }
.         { printf("CARÁCTER DESCONOCIDO: %s\n", yytext); }

%%

```

Imagen 3. Sección de reglas léxicas.

Dentro de las reglas se anexo el poder ignorar espacios en blancos y saltos de línea, esto debido a que los reconocería como caracteres desconocidos.

```

[ \t\n] ; /* Ignorar espacios y saltos de línea */
.     { printf("CARÁCTER DESCONOCIDO: %s\n", yytext); }

```

Imagen 4. Regla para ignorar los espacios y saltos de línea

Una manera para obtener las expresiones regulares es a través de autómatas. Como ejemplo se anexa un autómata para reconocer un numero entero. (Imagen 5.a)

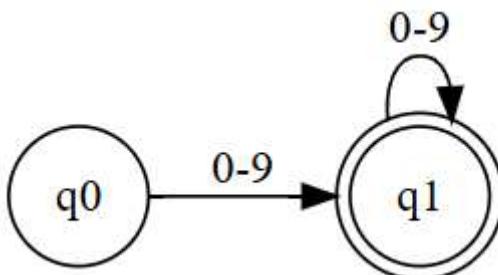


Imagen 5.a. Autómata de la expresión regular de numero entero

De igual manera como la construcción de numero entero también se puede realizar la construcción de la expresión regular para reconocer un identificador. (Imagen 5.b)

## PRÁCTICA DE LABORATORIO

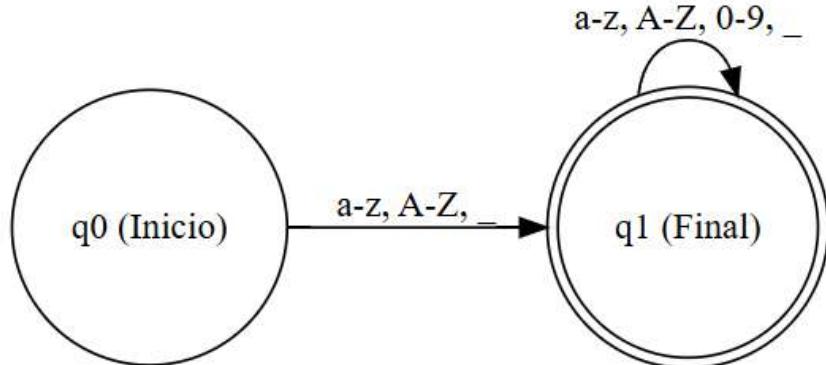


Imagen 5.b. Autómata de la expresión regular par identificadores

Por ultimo se agrega al código .lex a función para generar el analizador léxico. Donde la función yylex es generada automáticamente por Flex y se encarga de leer la entrada y aplicar las reglas definidas (Imagen 6)

```

int main() {
    yylex(); // Iniciar el analizador
    return 0;
}
    
```

Imagen 6. Función main

Se compila el programa y ejecuta. (Imagen 7)

```

nath@nath:~/flex$ flex actividad4.lex
nath@nath:~/flex$ gcc lex.yy.c -o actividad4 -lfl
nath@nath:~/flex$ ./actividad4
    
```

Imagen 7. Compilación y ejecución del programa

A continuación, se muestran varios ejemplos de como funciona el analizador léxico, con varias pruebas.

```

while count<10
PALABRA CLAVE: while
IDENTIFICADOR: count
CARÁCTER DESCONOCIDO: <
NÚMERO ENTERO: 10
    
```

## PRÁCTICA DE LABORATORIO

Imagen 8. Prueba de numero entero, identificador y la palabra clave while

```
if while 189 Idenficator for *
PALABRA CLAVE: if
PALABRA CLAVE: while
NÚMERO ENTERO: 189
IDENTIFICADOR: Idenficator
PALABRA CLAVE: for
CARÁCTER DESCONOCIDO: *
```

Imagen 9. Se reconocen todas las reglas definidas

A continuación, se muestra un ejemplo con sintaxis utilizando en conjunto con yacc, la diferencia del código es que ahora la reglas léxicas, en lugar de imprimir ahora solo devuelven el tokens, y yacc verifica la sintaxis para ver si while, for o if están escritos correctamente.

Un token o componente léxico es una cadena de caracteres que tiene un significado coherente en cierto lenguaje de programación. Ejemplos de tokens, podrían ser palabras clave (if, while, int), identificadores, números, signos, o un operador de varios caracteres. O bien definido de otra manera un token es un par que consiste en un nombre de token y un valor de atributo opcional. El nombre del token es un símbolo abstracto que representa un tipo de unidad léxica.

En la imagen 10 se muestra el código .lex que trabajara en conjunto con el código .yacc de esta manera solo deberá del volver los tokens. Cabe recalcar que es este caso se deben anexar la librería el cual incluye los tokens predefinidos en YACC

# PRÁCTICA DE LABORATORIO

```
%{
#include "y.tab.h" // Incluye los tokens definidos en Yacc
%}

%%

while    { return WHILE; }
for     { return FOR; }
if      { return IF; }
[a-zA-Z_][a-zA-Z0-9_]* { return IDENTIFIER; }
[0-9]+   { return NUMBER; }
[ \t\n] ; // Ignorar espacios
.        { return yytext[0]; }

%%
```

Imagen 10. Código .lex solo devuelve los tokens

El código de yacc se agrega la librería stdio.h para poder usar printf, el cual se agrega en código c, de igual manera que en el primer ejemplo de esta práctica. Asimismo, se define la gramática, donde tanto el while, for y if solo funcionaran con identificadores.

```
= actividad4.yacc.y
1  %}
2  #include <stdio.h>
3  %
4
5  %token WHILE FOR IF IDENTIFIER NUMBER
6
7  %%
8
9  stmt : WHILE IDENTIFIER { printf("Estructura WHILE válida\n"); }
10 | FOR IDENTIFIER { printf("Estructura FOR válida\n"); }
11 | IF IDENTIFIER { printf("Estructura IF válida\n"); }
12 ;
13
14 %%
```

Imagen 11. Código yacc donde se define la gramática

Una de las partes finales es la función main(), la función yyparse() es generada por Yacc/Bison y se encarga de analizar la entrada según las reglas sintácticas definidas. Cuando el analizador léxico (Flex) detecta tokens lo regresa, yypase() los recibe y los evalúa con reglas sintácticas. (Imagen 12)

# PRÁCTICA DE LABORATORIO

```
int main() {
    yyparse();
    return 0;
}
```

Imagen 12. Función main en el código .yacc

Por ultimo se maneja la función de error, donde se muestra un mensaje de error, si la regla gramatical no coincide. (Imagen 13)

```
void yyerror(const char *s) {
    printf("Error de sintaxis: %s\n", s);
}
```

Imagen 13. Función de error

Ejecución y compilación de ambos programas (Imagen 14)

```
nath@nath:/Flex$ yacc -d actividad4_yacc.y
nath@nath:/Flex$ flex actividad4.lex
nath@nath:/Flex$ gcc lex.yy.c y.tab.c -o actividad4_yacc -lfl
cc1: fatal error: y.tab.c: No such file or directory
compilation terminated.
nath@nath:/Flex$ gcc lex.yy.c y.tab.c -o actividad4_yacc -lfl
y.tab.c: In function 'yyparse':
y.tab.c:1021:16: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
1021 |     yychar = yylex ();
           ^
y.tab.c:1174:7: warning: implicit declaration of function 'yyerror'; did you
mean 'yyerrok'? [-Wimplicit-function-declaration]
1174 |     yyerror (YY_("syntax error"));
           ^
           yyerrok
actividad4_yacc.y: At top level:
actividad4_yacc.y:21:6: warning: conflicting types for 'yyerror'; have 'void
(const char *)'
   21 | void yyerror(const char *s) {
           ^
y.tab.c:1174:7: note: previous implicit declaration of 'yyerror' with type v
oid(const char *)
  1174 |     yyerror (YY_("syntax error"));
           ^
nath@nath:/Flex$ ./actividad4
-bash: ./actividad4: No such file or directory
nath@nath:~/Flex$ ./actividad4_yacc
-bash: ./actividad4_yacc: No such file or directory
nath@nath:~/Flex$ ./actividad4_yacc
```

Imagen 14. Ejecución y compilación de programa .y y lex

Nos podemos dar cuenta que muestra errores al copilar, esto se debe a que indica que hay la declaración de yyerror previamente definida.

Se muestra el funcionamiento del programa, con varios ejemplos. Destacando que solo se declaró tres reglas sintácticas donde será el while, if y for con un identificador Si bien, no representa la

# PRÁCTICA DE LABORATORIO

estructura exacta del lenguaje c, demuestra como es importante la fase léxica. (Imagen 15- Imagen 18).

```
nath@nath:~/flex$ ./actividad4_yacc
while x
Estructura WHILE válida
while 8931
Error de sintaxis: syntax error
```

Imagen 15. Prueba 1 de las reglas sintácticas.

```
nath@nath:~/flex$ ./actividad4_yacc
for id
Estructura FOR válida
```

Imagen 16. Regla sintáctica de for

```
nath@nath:~/flex$ ./actividad4_yacc
if identificador
Estructura IF válida
```

Regla sintáctica de if

## INCIDENCIAS

- Sin incidencias.

## OBSERVACIONES

La realización de esta práctica proporciona una base para entender los fundamentos en la construcción de los compiladores y procesadores de lenguajes. Además, durante la investigación se encontró otras herramientas para el análisis sintáctico y léxico como ANTLR y LLVM que son herramientas muy similares, pero con enfoques más modernos y mayores capacidades.

ANTLR es un potente generador de analizadores sintácticos que puedes usar para leer, procesar, ejecutar o traducir archivos binarios o de texto estructurado. Se usa ampliamente en el ámbito académico y en la industria para crear todo tipo de lenguajes, herramientas y marcos.

Por otro lado LLVM (Low Level Virtual Machine) es una infraestructura de compilación de código abierto que proporciona un conjunto de herramientas y bibliotecas para crear

# PRÁCTICA DE LABORATORIO

compiladores, depuradores y otras herramientas de desarrollo de software.

## CONCLUSIONES

Flex y Yacc son herramientas fundamentales en el procesamiento y análisis de lenguajes de programación. Flex, como generador de analizadores léxicos, se encarga de reconocer patrones dentro del código fuente mediante expresiones regulares, clasificando las secuencias de caracteres en tokens predefinidos, como palabras clave (while, for, if), identificadores (variable, funcion1), números (123, 42), operadores (+, =) y otros elementos del lenguaje. Una vez identificados estos tokens, se envían al siguiente nivel de análisis.

Por otro lado, Yacc (Yet Another Compiler Compiler) cumple la función de analizador sintáctico, asegurándose de que los tokens generados por Flex conformen estructuras gramaticales válidas según las reglas de producción definidas. Esto permite verificar si una secuencia de tokens corresponde a una estructura válida del lenguaje, como una declaración condicional, un bucle o una asignación. Yacc utiliza gramáticas libres de contexto para definir estas reglas y genera un parser capaz de interpretar correctamente la estructura del código fuente.

El uso combinado de Flex y Yacc es esencial en la construcción de compiladores, intérpretes, parsers y analizadores de código en sistemas complejos. Su conocimiento no solo es útil para diseñar lenguajes de programación, sino también para desarrollar herramientas avanzadas en procesamiento de texto, validación de formatos, análisis de datos y automatización de tareas relacionadas con la manipulación de código fuente. En la práctica, estas herramientas son utilizadas en áreas como bases de datos, seguridad informática, optimización de código, depuración de errores y transformación de archivos estructurados.

Un aspecto clave del análisis léxico es el uso de expresiones regulares, que permiten definir patrones de texto que corresponden a estructuras válidas dentro de un lenguaje.

## PRÁCTICA DE LABORATORIO

Gracias a estas expresiones, un analizador léxico puede distinguir con precisión entre palabras clave y variables, diferenciar operadores matemáticos de identificadores y detectar errores en la sintaxis básica del código antes de pasarlo al análisis sintáctico. A pesar de ser herramientas clásicas, Flex y Yacc siguen siendo ampliamente utilizadas en la enseñanza de teoría de compiladores, ya que proporcionan una base sólida para comprender cómo funcionan los sistemas de procesamiento de lenguajes. Sin embargo, con el tiempo han surgido alternativas más modernas como ANTLR, Bison, LLVM, entre otras, que ofrecen más flexibilidad y facilidad de uso.

### BIBLIOGRAFIA

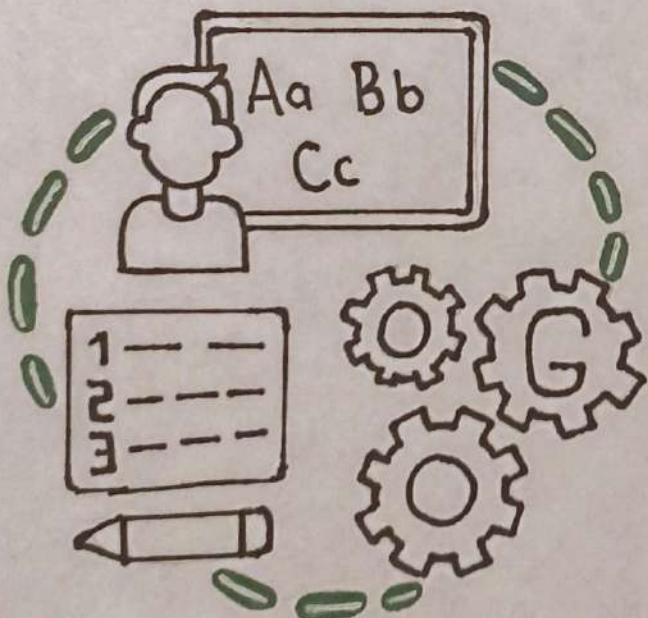
- *AIX 7.2.* (s. f.).  
<https://www.ibm.com/docs/es/aix/7.2?topic=information-creating-parser-yacc-program>
- *AIX 7.3.* (s. f.).  
<https://www.ibm.com/docs/es/aix/7.3?topic=information-generating-lexical-analyzer-lex-command>
- GeeksforGeeks. (2024, 17 septiembre). *FLEX (Fast Lexical Analyzer Generator)*. GeeksforGeeks.  
<https://www.geeksforgeeks.org/flex-fast-lexical-analyzer-generator/>
- Universitat Politècnica de València - UPV. (2011, 27 septiembre). *Flex. Desarrollo de un analizador léxico usando Flex | | UPV* [Vídeo]. YouTube.  
[https://www.youtube.com/watch?v=\\_zbIOMp63mo](https://www.youtube.com/watch?v=_zbIOMp63mo)
- *Webgraphviz.* (s. f.). <http://www.webgraphviz.com/>

# Gramáticas

## Libres de CONTEXTO

Primero que nada, la gramática es el conjunto de reglas del lenguaje que regulan el uso de una lengua determinada, y existen de 4 tipos:

- Tipo 0. De estructura de frase o sin restricciones.
- Tipo 1. Sensible al contexto (dependientes).
- Tipo 2. Libre de contexto.
- Tipo 3. Regulares.



En este trabajo nos enfocaremos a las de tipo 2, que son una clase de gramática formal utilizada en la teoría de lenguajes formales para describir lenguajes que no dependen del contexto en el que se encuentran los símbolos.

Leyla  
Leyla

Este tipo de gramática se define mediante una 4-tupla matemática

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

donde:

$V_n$  es un conjunto finito de símbolos no terminales

$V_t$  es un conjunto finito de símbolos terminales ( $V_n \cap V_t = \emptyset$ )

$P$  es un conjunto finito de producciones de la forma  
 $\alpha \rightarrow \beta$  donde  $\alpha \in V_n$  y  $\beta \in (V_t \cup V_n)^*$

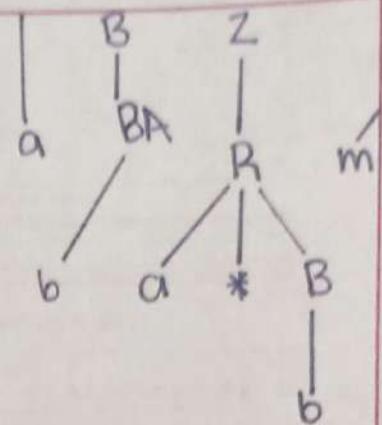
$S \in V_n$  es el símbolo inicial o axioma de la gramática.

Las gramáticas libres de contexto permiten describir la mayoría de los lenguajes de programación, de hecho la sintaxis de la mayoría está definida mediante estas gramáticas.

Por otro lado, estas gramáticas son suficientemente simples como para permitir el diseño de eficientes algoritmos de análisis sintáctico que, para una cadena de caracteres dada, determinen como puede ser generada desde la gramática.

# Árboles

\* \* de derivación \* \*

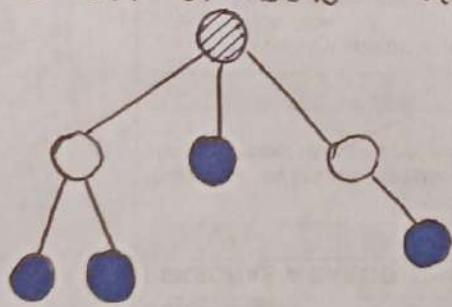


Un árbol de derivación es una representación gráfica que muestra como se puede derivar cualquier cadena de un lenguaje a partir del símbolo distinguido de una gramática. Este árbol consta de nodos unidos por arcos, donde cada nodo puede ser una hoja (sin hijos) o un nodo interior (con hijos).

## 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.



- Nodo raíz
- Nodos interiores
- Hojas

Para explicar mejor este concepto, pondremos un ejemplo de árbol de derivación.

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

$\langle \text{Sent\_asig} \rangle ::= \langle \text{Var} \rangle := \langle \text{expresión} \rangle$

$\langle \text{expresión} \rangle ::= \langle \text{expresión} \rangle + \langle \text{termino} \rangle \mid$   
 $\langle \text{expresión} \rangle - \langle \text{termino} \rangle \mid$   
 $\langle \text{termino} \rangle$

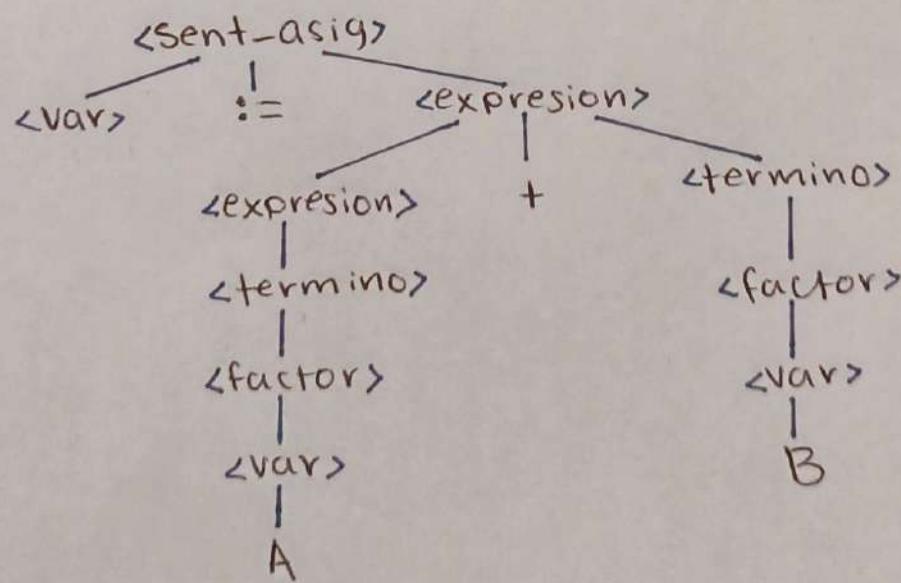
$\langle \text{termino} \rangle ::= \langle \text{termino} \rangle * \langle \text{factor} \rangle \mid$   
 $\langle \text{termino} \rangle / \langle \text{factor} \rangle \mid$   
 $\langle \text{factor} \rangle$

$\langle \text{factor} \rangle ::= (\langle \text{factor} \text{ expresión} \rangle) \mid \langle \text{Var} \rangle \mid \langle \text{num} \rangle$

$\langle \text{Var} \rangle ::= A \mid B \mid C \mid D \mid \dots \mid Z$

$\langle \text{num} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

Para la sentencia, propia del lenguaje,  $A := A + B$ ;  
su árbol de derivación sería de la siguiente forma:



# DIAGRAMAS DE SINTAXIS

Juanjo 100% BSC

## GRAMÁTICA

Conjunto de reglas que definen la secuencia correcta de los elementos de un lenguaje de programación. (Informática)

## INTRODUCCIÓN

Los diagramas de sintaxis o del ferrocarril son una forma de representar una gramática libre de contexto. Representa una alternativa gráfica para la forma Backus-Naur (BNF, por sus siglas en inglés) o la forma extendida de Backus Naur (EBNF, por sus siglas en inglés).

Es decir, es un formato visual que indica qué opciones puede proporcionar con el mandato y cómo especificarlos.

## IMPORTANCIA EN LA MATERIA DE LENGUAJES Y AUTÓMATAS II

Los diagramas de sintaxis permiten visualizar de manera intuitiva la estructura gramatical de una expresión, facilitando su compresión e implementación.

Además, tienen una gran importancia en la informática y los lenguajes debido a las siguientes razones.

- Análisis sintáctico: son esenciales en esta fase de los compiladores e intérpretes

Agustín 100% 100%

Herramientas: Son fundamentales para entender y aprender sobre la estructura de los lenguajes formales.

Optimización de código: Facilita la detección de redundancia y mejora de la eficiencia del código generado.

Los diagramas de sintaxis son una herramienta fundamental en la materia de lenguajes y Automatas II, ya que, permite visualizar, analizar y optimizar la estructura de los lenguajes formales.

## DESARROLLO.

### "FUNDAMENTOS TEÓRICOS"

#### LENGUAJES FORMALES Y GRAMÁTICAS

Un lenguaje formal es un lenguaje artificial que no se centra en la comunicación humana habitual ni se caracteriza por su ausencia de ambigüedad. Está definido y aplicado según un conjunto de reglas específicas dentro de un entorno particular. Este lenguaje consta de una serie de símbolos básicos que se combinan encadenados para formar expresiones complejas.

El conjunto de reglas que definen los lenguajes formales se conoce como gramáticas formales. Estas gramáticas determinan la estructura válida de las expresiones dentro del lenguaje y pueden representarse mediante diversas notaciones.

La gramática libre de contexto (CFG por sus siglas en inglés), es un superconjunto de gramática regular. (Diagrama 1).

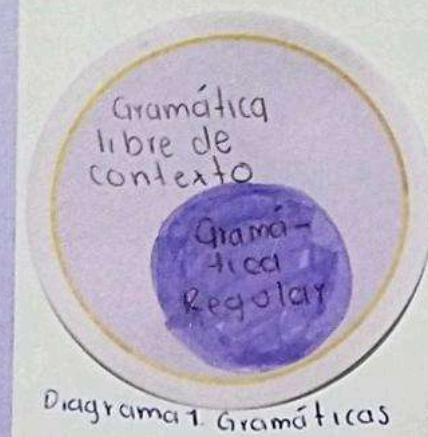


Diagrama 1. Gramáticas

Esto implica que toda gramática regular es independiente del contexto, pero existen algunos problemas que van más allá del alcance de la gramática regular.

### DIFERENCIAS ENTRE GRÁMATICAS REGULARES Y LIBRES DE CONTEXTO.

Los lenguajes libres de contexto, definidos por gramáticas libres de contexto, generan estructuras más complejas que las gramáticas regulares. Estas estructuras se representan mediante diagrama de sintaxis, que muestran cómo se combinan los símbolos no terminales y terminales para formar expresiones válidas.

A continuación, se muestra una tabla comparativa de las diferentes gramáticas. (Tabla 1).

GRAMÁTICA CARACTERÍSTICA	REGULARES	LIBRES DE CONTEXTO.
Definición	Los gramáticas regulares son un tipo de gramática formal que generan lenguajes regulares. Utilizan reglas de producción muy simples	Las gramáticas libres de contexto son un tipo más poderoso de gramática formal. Son utilizados en la mayoría de los lenguajes de programación.
Reglas de producción	Las reglas de producción tienen la forma: $A \rightarrow aB$ ó $A \rightarrow a$ Donde A y B son variables (no terminales) y a es un símbolo terminal. Pueden ser descritas mediante expresiones regulares.	Tienen la forma. $A \rightarrow \alpha$ Donde A es una variable (no terminal) y $\alpha$ es una cadena de variable y/o terminal.
Automata	Autómatas finitos	Autómatas de pila.
Ejemplo	<ul style="list-style-type: none"> <li>• <math>S \rightarrow aS</math></li> <li>• <math>S \rightarrow b</math></li> </ul>	<ul style="list-style-type: none"> <li>• <math>S \rightarrow aSb</math></li> <li>• <math>S \rightarrow \epsilon</math> (donde <math>\epsilon</math> representa la cadena vacía).</li> </ul>

Tabla 1. Comparación entre gramáticas

## NOTACIONES PARA REPRESENTAR LA SINTAXIS.

### ■ Backus-Naur Form (BNF)

Es una notación para definir la sintaxis de los lenguajes formales. Fue introducida por John Backus y Peter Naur. Utiliza reglas de producción para describir cómo se pueden formar expresiones válidas en el lenguaje.

Tiene la forma:  $\langle \text{simbolo no terminal} \rangle ::= \langle \text{expresión} \rangle$ , donde  $::=$  se lee como "se define como".

Ejemplo:

```
 $\langle \text{expresión} \rangle ::= \langle \text{término} \rangle | \langle \text{expresión} \rangle + \langle \text{término} \rangle$ 
 $\langle \text{término} \rangle ::= \langle \text{factor} \rangle | \langle \text{término} \rangle * \langle \text{factor} \rangle$ 
 $\langle \text{factor} \rangle ::= \langle \text{número} \rangle | \langle \text{expresión} \rangle$ 
 $\langle \text{número} \rangle ::= 0|1|2|3|4|5|6|7|8|9$ 
```

### ■ Extended Backus-Naur Form (EBNF)

Es una extensión de BNF que añade más notaciones para hacer la descripción de la sintaxis más concisa y expresiva. Introduce operadores adicionales como la repetición, la opción y la agrupación.

Utiliza  $::=$  para definiciones,  $|$  para alternativas,  $\{\}$  para repetición,  $[ ]$  para opción y  $( )$  para agrupación.

Ejemplo:

```
 $\langle \text{expresión} \rangle ::= \langle \text{término} \rangle \{ "+" \langle \text{término} \rangle \}^*$ 
 $\langle \text{término} \rangle ::= \langle \text{factor} \rangle \{ "*" \langle \text{factor} \rangle \}^*$ 
 $\langle \text{factor} \rangle ::= \langle \text{número} \rangle | "(" \langle \text{expresión} \rangle ")"$ 
 $\langle \text{número} \rangle ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"$ 
```

### ■ Diagramas de sintaxis.

Son representaciones gráficas de las reglas de producción de una gramática formal. Utilizan símbolos gráficos y flechas para conectar elementos, ilustrando la aplicación de las reglas de producción de manera visual.

"ELEMENTOS Y CONSTRUCCIÓN DE UN  
DIAGRAMA DE SINTAXIS"

Son diagramas con caminos y conexiones que muestran cómo se pueden combinar los símbolos (terminales y no terminales) según las reglas gramaticales. Cada diagrama de sintaxis lineal emplea un anclaje doble hacia la derecha y finaliza con un par de flechas a la derecha y a la izquierda. Las líneas de continuación.

A continuación, se muestra la tabla de convenio de los diagramas de sintaxis. (Tabla 2)

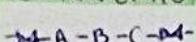
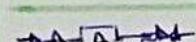
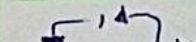
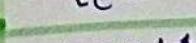
CONVENIO	SIGNIFICADO
	Debe especificar los valores A, B, C. Los valores necesarios se muestran en la línea principal de un diagrama de ferrocarril.
	Puede especificar el valor A. Los valores opcionales se muestran debajo de la línea principal.
	Los valores A, B, C son alternativas
	Esto muestra que se debe seleccionar un valor (A, o B, o C) y si se va a seleccionar otro, se debe utilizar una coma entre los valores.
	Debe especificar el valor de A varias veces. El separador de este ejemplo es opcional.
	Los valores A, B, C son alternativas, una de las cuales puede especificar. Si no se especifica ninguno de los valores mostrados, se utiliza el valor predeterminado D.
	El fragmento de sintaxis lineal Nombre se muestra separadamente del diagrama de sintaxis lineal principal.

Tabla 2. Convenio para los diagramas de sintaxis.

Guillen Pachón

## RELACION ENTRE ANALISIS SINTACTICOS

Se ha visto en otras actividades que un analizador léxico puede identificar tokens con la ayuda de una expresión regular y de patrones. Pero un analizador léxico no puede verificar la sintaxis de una oración debido a las limitaciones de las expresiones regulares. Las expresiones regulares no pueden verificar tokens de equilibrio como los paréntesis. Por lo tanto la fase análisis sintáctico, utiliza la gramática libre de contexto (CFG), que es reconocida por los autómatas.

Un analizador sintáctico o parser toma la entrada de un analizador léxico en forma de flujos de tokens. El parser analiza el código fuente (flujo de tokens) comparándolo con las reglas de producción, para detectar cualquier error del código. El resultado de esta fase es un árbol de análisis. (Diagrama 2).



Diagrama 2. Analizador léxico y sintáctico

De esta manera, el analizador realiza dos tareas, es decir, analiza el código, busca errores y genera un árbol de análisis como salida de fase.

Los diagramas de sintaxis son herramientas visuales que representan la regla de la gramática.

## "EJEMPLOS PRÁCTICOS Y CASOS DE USO"

Se presenta un ejemplo del bloque if-else (Diagrama 3)  
if-else

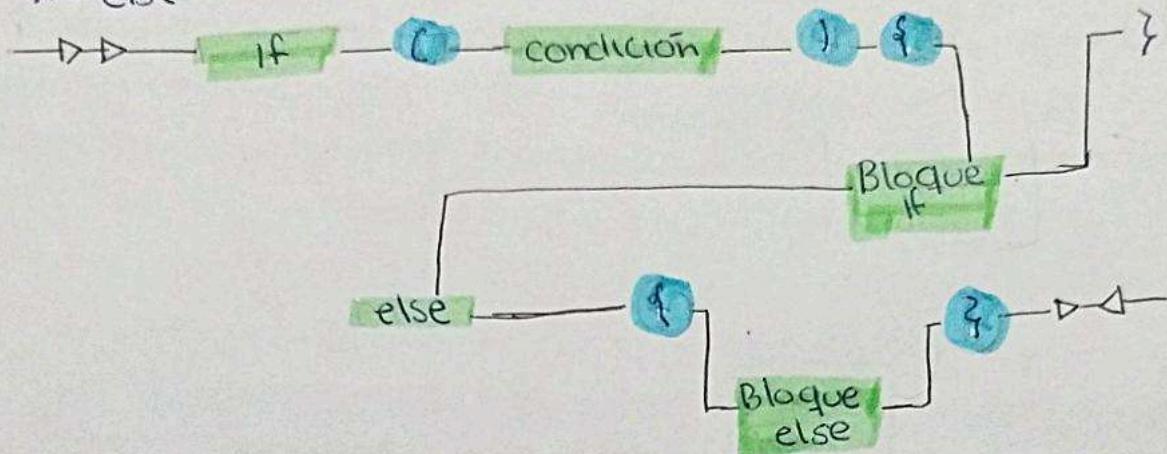


Diagrama 3. Diagrama de sintaxis if-else

Otro ejemplo es de la estructura JSON, donde se muestra el diagrama de sintaxis de un objeto; un objeto es un conjunto desordenado de pares nombre / valor. Un objeto comienza con llave izquierda { y termina con llave derecha }. Cada nombre va seguido de dos puntos y los pares nombre / valor están separados por coma. (Diagrama 4)

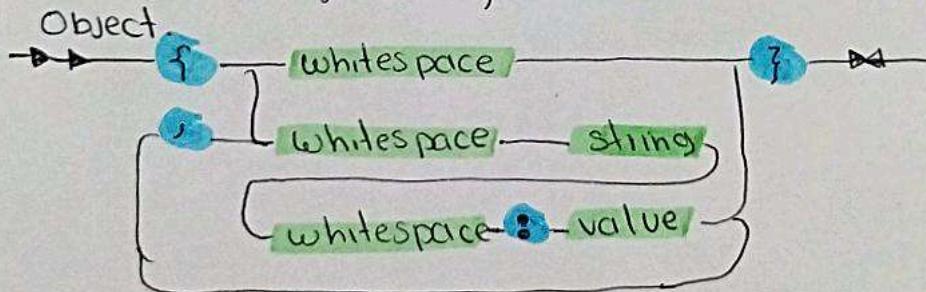


Diagrama 4. Diagrama de sintaxis de estructura JSON

## CONCLUSIÓN

Los diagramas de sintaxis son herramienta fundamental en la fase de análisis sintáctico de la compilación, donde se verifica la estructura gramatical.

del código fuente. En el contexto de los lenguajes formales, las gramáticas regulares y libres de contexto son esenciales para definir las reglas que rigen la sintaxis de estos lenguajes. Las notaciones como la Backus-Naur (BNF) y sus variantes se utilizan para representar estas reglas de manera formal. Los diagramas de sintaxis visualizan la estructura jerárquica del código, facilitando la identificación de errores y la compresión de la organización de la gramática, destacando cómo se aplican las reglas gramaticales en cada paso. En conjunto, estas herramientas y conceptos son cruciales para asegurar la correcta interpretación de los programas por parte de los compiladores e intérpretes.

Julián Pérez  
100%

# =Práctica=

## 1- ¿Qué son las gramáticas libres de contexto?

Así como se abordó en la monografía, las gramáticas libres de contexto (GLC) son aquellas donde su regla se encuentra definida por la forma  $A \rightarrow \alpha$ , donde A corresponde a un símbolo no terminal, mientras que  $\alpha$  es cualquier cadena formada por símbolos no terminales y/o terminales.

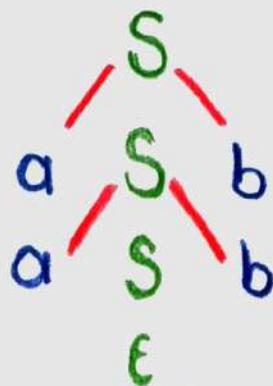
Estas gramáticas son fundamentales en la teoría de lenguajes formales y se usan para describir la sintaxis de lenguajes de programación y lenguajes naturales.

En el ejercicio práctico se creó una gramática libre de contexto simple, cuyo símbolo inicial es S y las reglas de producción son:

- $S \rightarrow aSb$
- $S \rightarrow \epsilon$  ( $\epsilon$  representa la cadena vacía)

→ Esta gramática genera cadenas como  $\epsilon$ , ab, aabb, aaabbb, etc.

A continuación se presenta un árbol de derivación para la cadena aaabb:



## 2: ¿Qué es un contexto?

El contexto en una gramática se refiere a la dependencia que puede existir entre los símbolos que rodean a un no terminal y las reglas de producción que se aplican.

En las gramáticas dependientes de contexto, las reglas de producción pueden ser aplicadas dependiendo de los símbolos que aparecen antes o después del no terminal.

A diferencia de las GLC, donde las reglas se aplican independientemente del entorno, las gramáticas dependientes del contexto requieren de ciertas condiciones sobre los símbolos adyacentes sean satisfechas para que se pueda realizar una producción.

En el documento se plantea un ejercicio que considero una buena demostración práctica de cómo el contexto influye en la interpretación de un identificador.

En el contexto del programar, el contexto se refiere al entorno o situación en la que un identificador (como una variable, función, close, etc), es interpretado.

En el código se aprecia que **X** cambia de significado.

- 1 Inicia como una función
- 2 Luego se redefine como una variable con valor 5
- 3 Finalmente vuelve a ser variable.

Código

```
def x():  
    return "Soy una función"
```

en ese contexto  $\boxed{x}$  se interpreta como función.

$\hookrightarrow x=5$

Después  $\boxed{x}$  ya no es una función, ahora una variable.

! En el código al intentar usar  $\boxed{x}$  como una función se lanza un tipo de error (TypeError).

Finalmente  $x$  se restaura para volver a ser función.

del x

def x():

return "Soy una función nuevamente"

Este ejercicio demuestra la importancia del contexto al cambiar  $x$  entre una función y una variable.

3- ¿Sirve dicho contexto a los propósitos de una gramática que define un lenguaje formal?

Considero que la respuesta a esta pregunta va más allá de un simple sí o un no, ya que depende del propósito y del lenguaje que se pretende generar.

El contexto es fundamental cuando se requiere expresar lenguajes que tienen relaciones de dependencia entre diferentes partes de la cadena. En lenguajes formales simples, como los definidos por gramáticas libres de contexto, el contexto no es necesario.

Sin embargo en lenguajes más complejos que requieren una correspondencia precisa entre diferentes símbolos o

donde las reglas dependen de lo que ocurre antes o después de un símbolo dado, el contexto es indispensable.

Existen lenguajes de programación que no pueden ser descritos sin el uso de gramáticas dependientes de contexto.

En los programas, el primero (GLC) no fue necesario el contexto para definir el lenguaje "a<sup>n</sup>b", ya que la gramática puede generar el lenguaje sin restricciones adicionales. Sin embargo para el segundo fue necesario el contexto.

## Árboles de derivación

~~Ejemplo 1~~

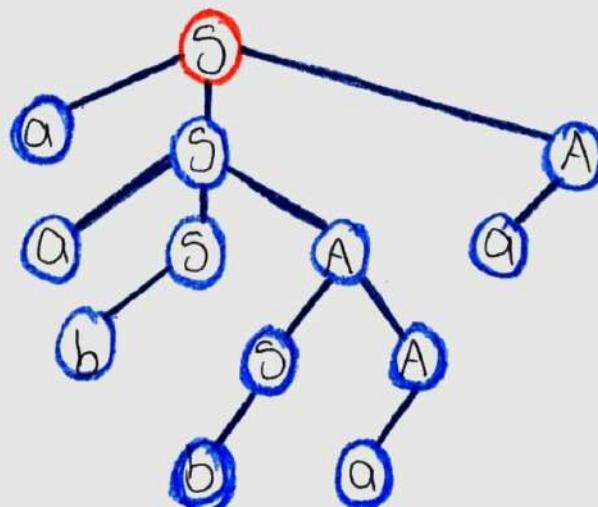
Reglas de producción:

- 1:  $S \rightarrow aSA$
- 2:  $S \rightarrow b$
- 3:  $A \rightarrow SA$
- 4:  $A \rightarrow a$

Derivando la cadena abbaa utilizando las reglas anteriores dadas.

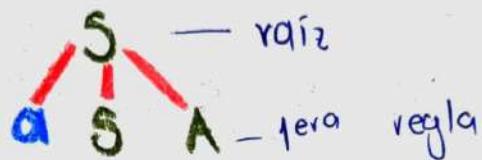
Simbolos

- No terminales: S, A
- Terminales: a, b

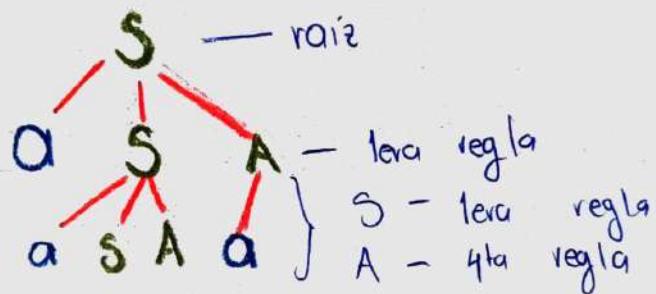


## Explicación:

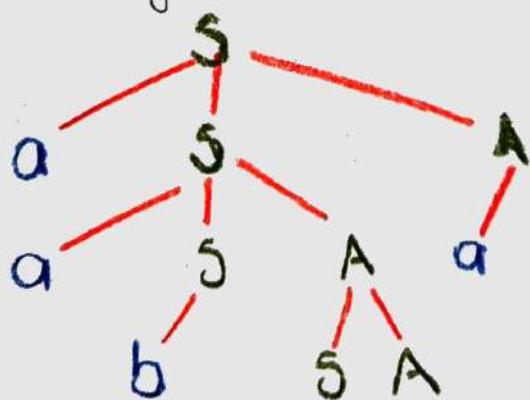
Tenemos la cadena  $aabbba$  y las reglas de producción. Comenzamos colocando nuestra raíz ( $S$ ) el cual es un nodo no terminal que se deriva en  $ASA$ , utilizando la primera regla, siendo  $a, S, A$  nodos dentro del árbol;  $a$  y  $s$  no terminales y  $A$  una hoja.



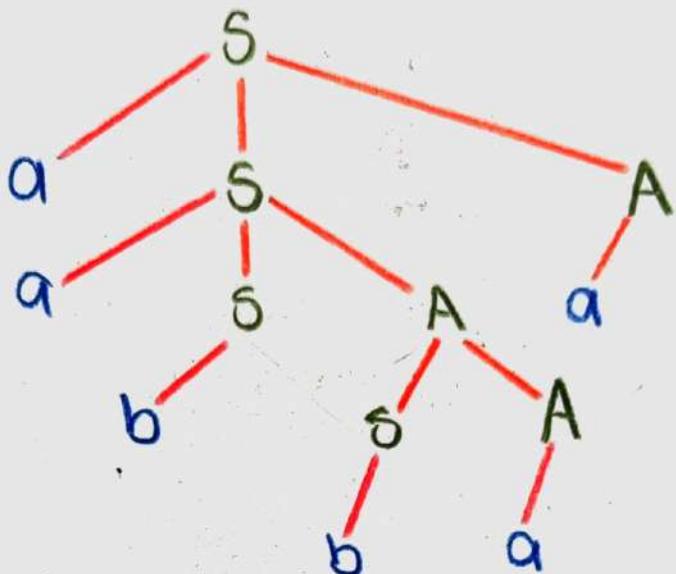
Se procede a derivar  $A$  y  $S$  que son nodos no terminales e intermedios.



Se continua derivando a  $S$  con la segunda regla + a  $A$  con la tercera regla de producción



Por último se deriva  $S$  según la segunda regla de producción y a  $A$  según la cuarta regla.



Julián HF 2010

Como se observa las hojas del árbol leídas de izquierda a derecha forman la cadena aabbaai, la cual era el objetivo a derivar.

## — Ejemplo 2 —

### Reglas de producción

- 1: O  $\rightarrow$  SN SV
- 2: SN  $\rightarrow$  Det N
- 3: SV  $\rightarrow$  VS N
- 4: Det  $\rightarrow$  El | La | los | las
- 5: N  $\rightarrow$  gato | pescado
- 6: V  $\rightarrow$  come
- 7: S N  $\rightarrow$  N

### Símbolos

- Terminales: El, la, los, las, gato, pescado, come
- No terminales: O, SN, SV, Det, N, V

### Significado de símbolos

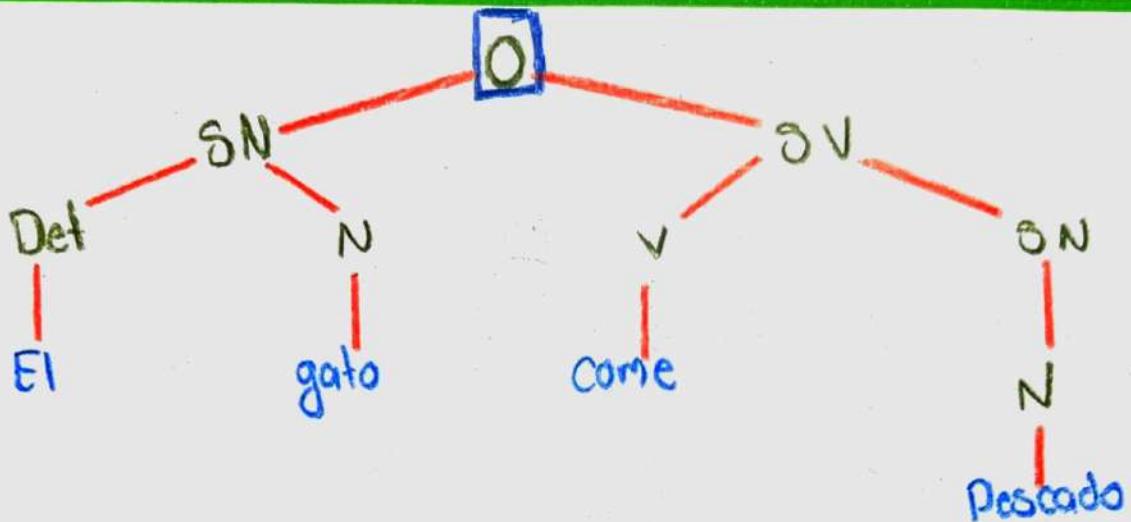
O  $\rightarrow$  Oración    SN  $\rightarrow$  sintagma nominal

SV  $\rightarrow$  sintagma verbal    Det  $\rightarrow$  Determinante

N  $\rightarrow$  Nombre

V  $\rightarrow$  verbo

Derivando la cadena El gato come pescado usando las reglas de producción.



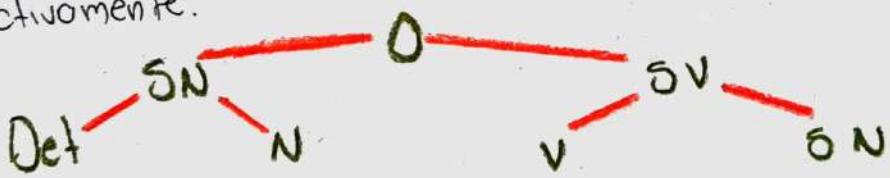
Explicación:

Se tiene que derivar la oración "El gato come pescado".

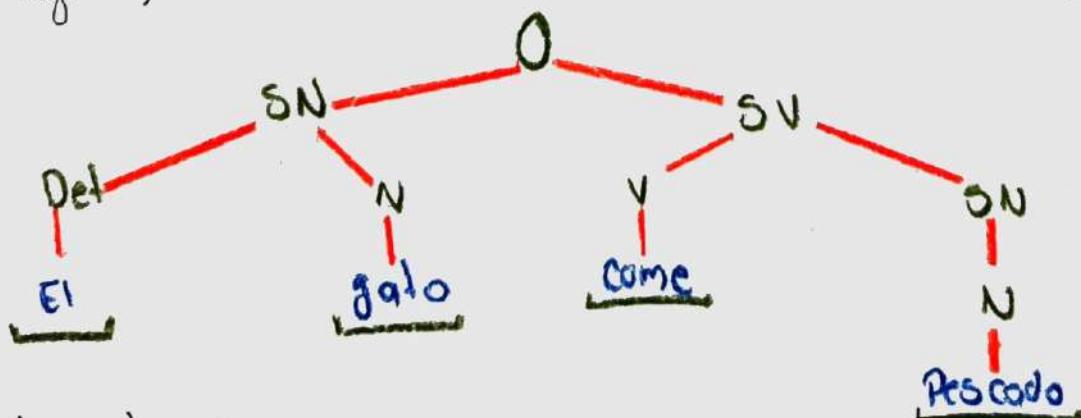
Comenzamos con la raíz del árbol O y se utilizará la primera regla de derivación



Se continua derivando a cada nodo con las reglas 2, 3 respectivamente.



De izquierdo a derecho se utilizan las reglas 4, 5 y 6 que ya son nodos terminales. El nodo más alejado se deriva con la regla 7, y a su vez este es derivado por la regla 5.



Así el árbol queda completo.

### Ejemplo 3

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

$\langle \text{Sent-asig} \rangle ::= \langle \text{Var} \rangle ::= \langle \text{expression} \rangle$

$\langle \text{expression} \rangle ::= \langle \text{expression} \rangle + \langle \text{termino} \rangle \mid \langle \text{expression} \rangle - \langle \text{termino} \rangle \mid \langle \text{termino} \rangle$

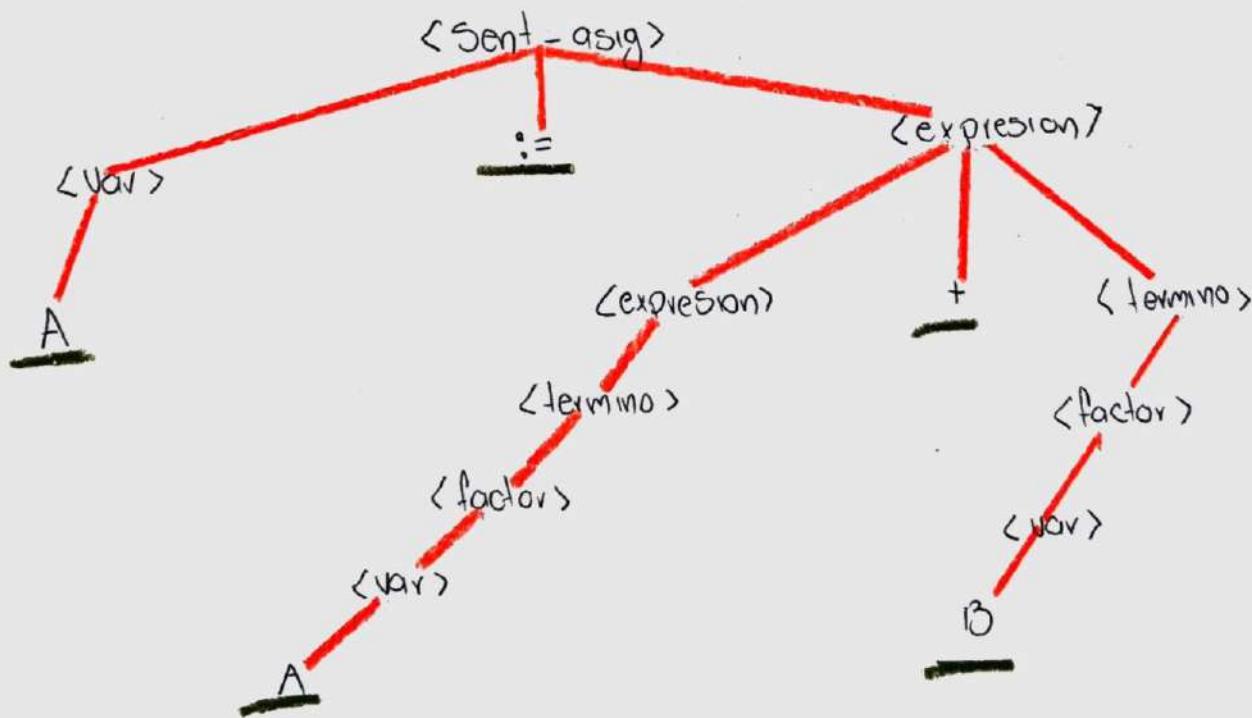
$\langle \text{termino} \rangle ::= \langle \text{termino} \rangle * \langle \text{factor} \rangle \mid \langle \text{termino} \rangle / \langle \text{factor} \rangle \mid \langle \text{factor} \rangle$

$\langle \text{factor} \rangle ::= (\langle \text{expression} \rangle) \mid \langle \text{var} \rangle \mid \langle \text{num} \rangle$

$\langle \text{var} \rangle ::= A \mid B \mid C \mid D \mid E \mid \dots \mid z$

$\langle \text{num} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

Derivando la sentencia  $A := A + B$  utilizando la definición de BNF:



## ► Explicación:

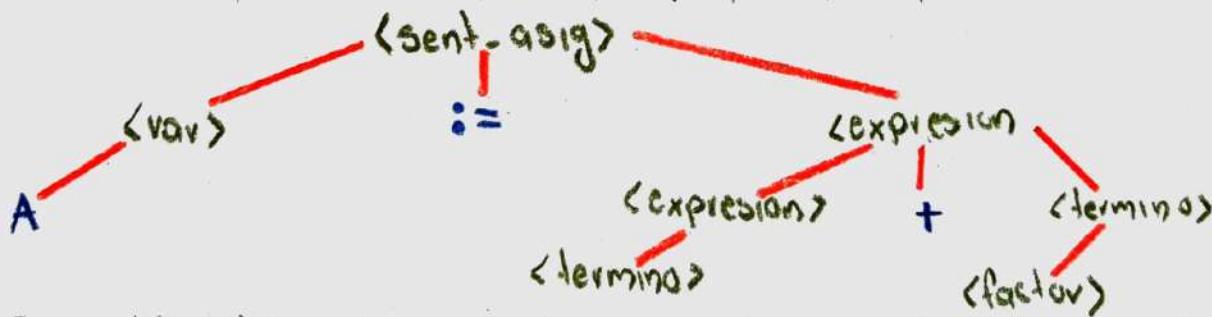
Deseamos realizar una sentencia donde asignamos o A el resultado de la expresión  $A+B$ . Identificamos dentro de la definición BNF el bloque o sentencia correspondiente a una asignación en nuestro lenguaje tipo poscal:

<sent-asig>

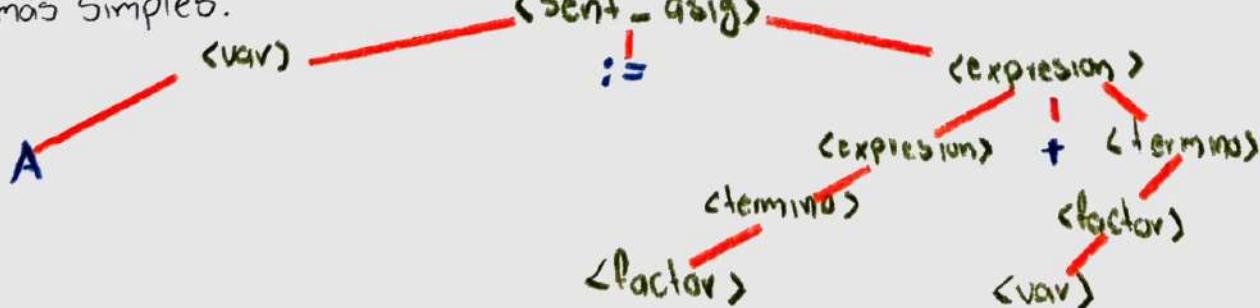
Esta sentencia se compone de una variable, el signo  $:=$  y una expresión. Las variables se identifican como <var> y pueden contener una letra mayúscula de la A a la Z. Lo que ahora nos interesa es <expresión>+<termino>, referente a la expresión.



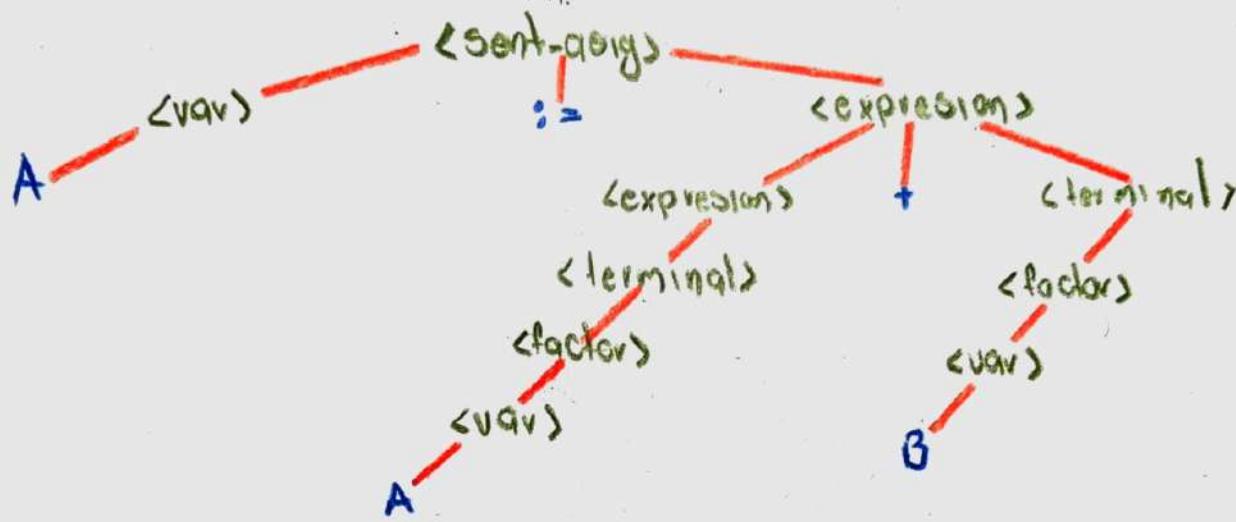
En este punto hay 3 nodos hoja, aún quedan nodos que pueden derivarse. Continuando tendríamos <expresion> y a <termino>



Es posible volver a derivar los nodos no terminales nuevos, al derivar su complejidad disminuye, es decir se reduce a estructuras más simples.



El <factor> puede volver a derivarse y (var) da como resultado un nodo terminal.



Si se lee de izquierdo a derecha se muestra la cadena  
 $A := A + B$ .

Es posible concluir lo siguiente: La definición BNF, así como las reglas de producción y otras formas de representar la gramática de un lenguaje nos ayudan durante la construcción de símbolos de derivación para representar el comportamiento de una gramática libre de contexto cuando validamos cadenas.

CARRERA	NOMBRE DE LA ASIGNATURA
Ingeniería en Sistemas Computacionales	Lenguajes y Autómatas II

DOCENTE DESIGNADO
ISC. RICARDO GONZÁLEZ GONZÁLEZ
EQUIPO / INTEGRANTES
AGUILAR FLORES ABEL 21031027
MACÍAS SEVILLA DIANA NATASHA 21030223
CASTAÑEDA PÉREZ CRISTIAN EDUARDO 21030140

PRACTICA No.	NOMBRE DE LA PRACTICA	DURACION(HORAS)
4	Ejercicios de gramática y la importancia del contexto	5 horas

1	INTRODUCCIÓN
En el estudio de los lenguajes formales y la teoría de la computación, las gramáticas libres de contexto y el concepto de contexto desempeñan un papel fundamental. Estos conceptos no solo son esenciales para comprender cómo se estructuran y analizan los lenguajes de programación, sino que también son herramientas clave en el diseño de compiladores, intérpretes y otras aplicaciones relacionadas con el procesamiento de lenguajes. En este trabajo, se abordarán ejercicios que nos permitirán ejemplificar las preguntas que se mencionan en el objetivo.	

2	OBJETIVOS (COMPETENCIAS)
Elaborar ejercicios prácticos con los que se pueda demostrar lo siguiente: <ul style="list-style-type: none"> <li>• ¿Qué son las gramáticas libres de contexto?</li> <li>• ¿Qué es un contexto?</li> <li>• ¿Sirve dicho contexto a los propósitos de una gramática que defina un lenguaje formal?</li> </ul>	

4

## MARCO TEÓRICO REFERENCIAL

El estudio de los lenguajes formales y las gramáticas es fundamental en áreas como la teoría de la computación, el diseño de compiladores y el procesamiento de lenguajes naturales. Las gramáticas libres de contexto, el contexto y su relación con la definición de lenguajes formales son conceptos que son pilares teóricos, y que también tienen aplicaciones prácticas en el desarrollo de herramientas para el análisis y procesamiento de lenguajes.

### Gramáticas Libres de Contexto (GLC)

Las **gramáticas libres de contexto** son un tipo de gramática formal en la que cada regla de producción tiene la forma  $A \rightarrow \alpha A \rightarrow \alpha$ , donde  $AA$  es un símbolo no terminal y  $\alpha\alpha$  es una cadena de símbolos terminales y/o no terminales. Estas gramáticas son ampliamente utilizadas para describir la sintaxis de lenguajes de programación y lenguajes naturales debido a su equilibrio entre expresividad y simplicidad.

- **Componentes de una GLC:**

- **Símbolos no terminales:** Representan categorías sintácticas (por ejemplo, "expresión", "declaración").
- **Símbolos terminales:** Son los elementos básicos del lenguaje (por ejemplo, palabras clave, operadores).
- **Reglas de producción:** Definen cómo los no terminales pueden ser reemplazados por cadenas de terminales y no terminales.
- **Símbolo inicial:** Es el punto de partida para generar cadenas válidas en el lenguaje.

- **Aplicaciones:**

- En compiladores, las GLC se utilizan para definir la estructura sintáctica de los lenguajes de programación, en el procesamiento de lenguajes naturales, se emplean para modelar la gramática de oraciones.

### El Concepto de Contexto

El contexto se refiere al entorno o situación en la que un elemento (como un identificador, una palabra o un símbolo) es interpretado. En el ámbito de los

2023  
Julián  
TecNM

lenguajes formales y la programación, el contexto determina el significado y el uso de los elementos sintácticos.

- **Contexto en Gramáticas:**

En una gramática libre de contexto, las reglas de producción no dependen del contexto en el que aparecen los símbolos no terminales. Esto significa que un no terminal siempre se reemplaza de la misma manera, independientemente de su entorno.

En una gramática dependiente del contexto, las reglas de producción pueden variar según el contexto en el que aparecen los símbolos.

- **Contexto en Programación:**

En lenguajes de programación, el contexto determina si un identificador es una variable, una función, una clase, etc. Por ejemplo, en Python, el mismo nombre puede referirse a una función en un contexto y a una variable en otro.

4

### MATERIALES UTILIZADOS

Para desarrollar los ejercicios prácticos se utilizarán:

- Lenguaje de Programación Python:
  - Para la implementación de analizadores sintácticos.
  - Para la construcción de árboles de derivación.
- Editor de Código:
  - PyCharm.

### **1. Ejercicio sobre gramáticas libres de contexto**

**Pregunta:** ¿Qué son las gramáticas libres de contexto?

Este programa verifica si una cadena pertenece al lenguaje generado por la gramática libre de contexto:  $S \rightarrow aSb \mid \epsilon$

Donde cada 'a' debe tener un 'b' correspondiente después, asegurando una estructura balanceada de la forma  $a^n b^n$ . Se utiliza una pila para llevar un seguimiento de los símbolos.

#### **Pseudocódigo**

FUNCION es\_valida(cadena):

    Crear una pila vacía

    PARA CADA símbolo EN cadena:

        SI símbolo ES 'a':

            Apilar 'a' EN la pila

        SINO SI símbolo ES 'b':

            SI la pila está vacía O el elemento superior de la pila NO ES 'a':

                RETORNAR FALSO # No hay 'a' para emparejar con 'b'

            Desapilar el elemento superior de la pila

        SINO:

            RETORNAR FALSO # Símbolo no válido

    SI la pila está vacía:

        RETORNAR VERDADERO # Todos los 'a' se emparejaron con 'b'

    SINO:

        RETORNAR FALSO # No se emparejaron todos los 'a' con 'b'

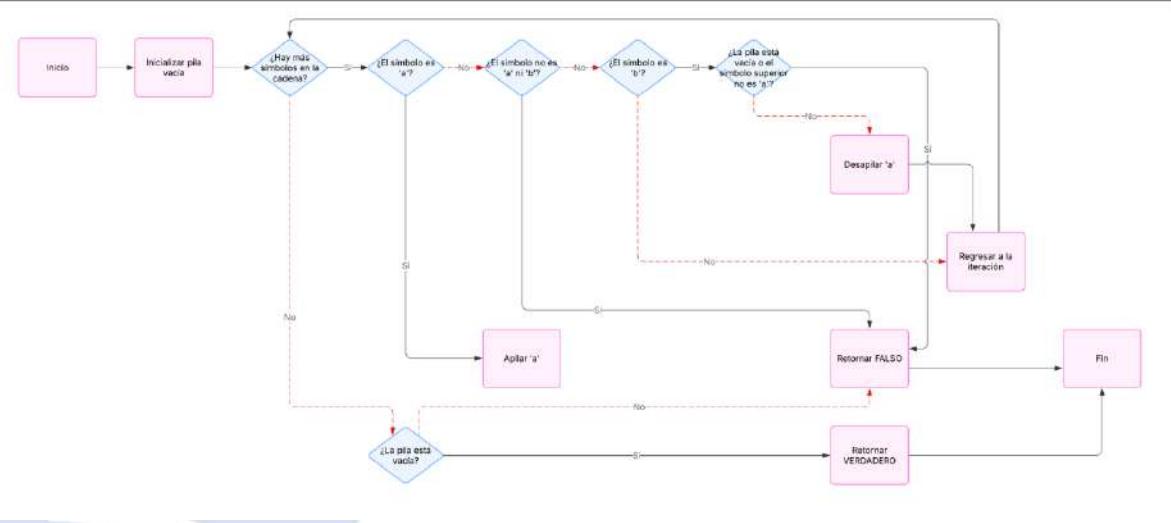
# Pruebas

cadenas = ["ab", "aabb", "aaabbb", "aab", "abb", "abc"]

PARA CADA cadena EN cadenas:

    Imprimir "cadena es válida: es\_valida(cadena)"

## Diagrama de flujo



## Código

```

# Gramática libre de contexto: S -> aSb | ε
def es_valida(cadena):
    pila = []
    for simbolo in cadena:
        if simbolo == 'a':
            pila.append(simbolo) # Apilar 'a'
        elif simbolo == 'b':
            if not pila or pila[-1] != 'a':
                return False # No hay 'a' para emparejar con 'b'
            pila.pop() # Desapilar 'a'
        else:
            return False # Símbolo no válido
    return not pila # La pila debe estar vacía al final
  
```

### # Pruebas

```

cadenas = ["ab", "aabb", "aaabbb", "aab", "abb", "abc"]
for cadena in cadenas:
    print(f"'{cadena}' es válida: {es_valida(cadena)}")
  
```

## Explicación del algoritmo

### Uso de la Pila

- Cada vez que encontramos un 'a', se apila.
- Cada vez que encontramos un 'b', se desapila un 'a', asegurando que cada 'b' tenga un 'a' antes de él.
- Si encontramos un 'b' sin 'a' en la pila o un símbolo no válido, la cadena no es aceptada.

Inicialmente se crea una pila vacía, luego se recorre la cadena símbolo por símbolo.

Después se apilan las 'a' y se desapilan cuando aparece un 'b'. Si hay errores (un 'b' sin 'a', un símbolo inválido o 'a' sin emparejar), se retorna False. Si al final la pila está vacía, la cadena es válida.

Las pruebas fueron: cadenas = ["ab", "aabb", "aaabbb", "aab", "abb", "abc"]

## Resultados

```
C:\Users\lalo2\PycharmProjects\opencv1\.venv\Scripts\python.exe
'ab' es válida: True
'aabb' es válida: True
'aaabbb' es válida: True
'aab' es válida: False
'abb' es válida: False
'abc' es válida: False
```

## 2. Ejercicio sobre contexto

**Pregunta:** ¿Qué es un contexto?

Este programa demuestra cómo el contexto influye en la interpretación de un identificador (por ejemplo, si es una variable o una función).

### Pseudocódigo

INICIO

FUNCION x()

    RETORNAR "Soy una función"

FIN FUNCIÓN

x ← 5 # Ahora x es una variable con valor 5

INTENTAR

    IMPRIMIR x() # Intentar llamar a x como función

    CAPTURAR ERROR TipoError COMO e

        IMPRIMIR "Error: ", e, ". x es ahora una variable con valor ", x

FIN INTENTAR

# Restaurar x como función

ELIMINAR x # Se elimina la variable x

FUNCION x()

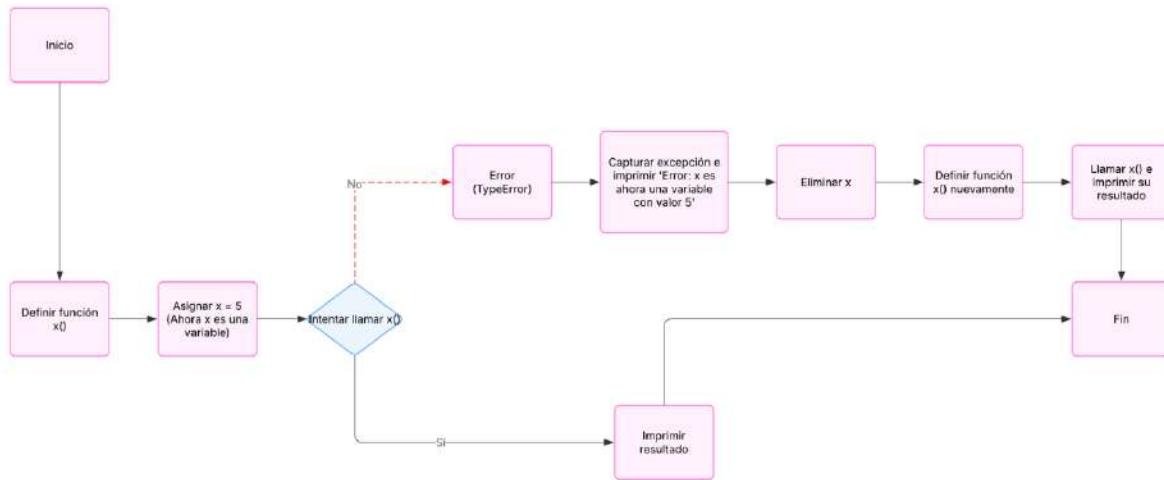
    RETORNAR "Soy una función nuevamente"

FIN FUNCIÓN

IMPRIMIR x() # Se imprime el resultado de la función x

FIN

### Diagrama de flujo



### Código

```

# Ejemplo de contexto en Python
def x():
    return "Soy una función"

x = 5 # Ahora x es una variable

# Dependiendo del contexto, x puede ser una variable o una función
try:
    print(x()) # Esto fallará porque x ahora es un entero
except TypeError as e:
    print(f"Error: {e}. x es ahora una variable con valor {x}.")

# Restaurar x como función
del x # Eliminar la variable x
def x():
    return "Soy una función nuevamente"

print(x()) # Ahora x es una función de nuevo
  
```

## Explicación del algoritmo

1. Inicialmente se define x como función que retorna el mensaje "Soy una función". En este contexto, x se interpreta como una función y puede ser llamada con x().

2. Redefinición de x como variable -> x = 5

- x se redefine como una variable que almacena el valor 5. Ahora, x ya no es una función, sino un entero.

3. Intento de usar x como función

- Se intenta llamar a x como si fuera una función (x()), pero como x ahora es un entero, Python lanza un error de tipo (TypeError).
- El bloque try-except captura el error y muestra un mensaje indicando que x es ahora una variable con valor 5.

4. Restauración de x como función

- ```
del x # Eliminar la variable x
def x():
    return "Soy una función nuevamente"
```
- del x elimina la variable x, lo que permite redefinirla.
  - Luego, x se redefine nuevamente como una función que retorna "Soy una función nuevamente".

5. Uso de x como función: Ahora que x es una función, la llamada x() funciona correctamente y muestra el mensaje "Soy una función nuevamente".

Este algoritmo ilustra cómo el contexto determina el significado de un identificador (x). En un momento, x es una función; en otro, es una variable. Esto demuestra la flexibilidad de Python y la importancia del contexto en la interpretación de código.

## Resultados

```
C:\Users\lalo2\PycharmProjects\opencv1\.venv\Scripts\python.exe C:\Users\la
Error: 'int' object is not callable. x es ahora una variable con valor 5.
Soy una función nuevamente

Process finished with exit code 0
```

**6**

### **BITÁCORA DE INCIDENCIAS**

- No se presentaron incidencias

**7**

### **OBSERVACIONES**

- A través de la investigación y de los ejercicios y ejemplos propuestos es importante destacar que las GLC tienen una expresividad limitada, sin embargo son fáciles de implementar.
- En el ejercicio sobre el contexto, un identificador puede cambiar de significado (función, variable) según el contexto, aunque si hubiera usado otro lenguaje de programación como Java, no habría sido posible el ejercicio pues al ser más rígido no permite que los identificadores cambien de tipo durante la ejecución.

**8**

### **ANEXOS**

- Sin anexos

**9**

### **REFERENCIAS**

- ✓ Un Profe de Informática. (2020, 6 mayo). *Lenguajes y Autómatas - Módulo 2.1 (Gramáticas libres de contexto)* [Vídeo]. YouTube.  
<https://www.youtube.com/watch?v=S7CIUL-4b-k>
- ✓ AxolotechAcademy. (2021, 6 febrero). Teoría de la computación - Capítulo 6 - Clasificación de las gramáticas según Chomsky. [Video]. YouTube.  
<https://www.youtube.com/watch?v=q7Rq3pu1oFY>
- ✓ Hilda Yelitza Contreras Zambrano. (2013, 10 mayo). Gramáticas Libres de Contexto [Video]. YouTube. <https://www.youtube.com/watch?v=V8a2-O2oEbo>

## FUENTES DE INFORMACIÓN

- Compiler Design - Syntax analysis (s.f). [https://www.tutorialspoint.com/compiler\\_design-syntax\\_analysis.htm](https://www.tutorialspoint.com/compiler_design-syntax_analysis.htm).
- Fariás G. (2024, 25 marzo). Lenguajes formales-Concepto, características, tipos y ejemplos. Concepto. <https://concepto.de/lenguajes-formales/>
- IBM MQ 9.2 (s.f) <https://www.ibm.com/docs/es/ibm-mq/9.2?topic=reference-syntax-diagrams>
- JSON. (s.f) <http://www.json.org/json-en.html>



| CARRERA | NOMBRE DE LA ASIGNATURA  |
|---------|--------------------------|
| ISC     | Lenguajes y Automatas II |

| AUTORES:                         |                                                                                    |                  |
|----------------------------------|------------------------------------------------------------------------------------|------------------|
| MACÍAS SEVILLA DIANA NATASHA     |                                                                                    | 21030223         |
| AGUILAR FLORES ABEL              |                                                                                    | 21031027         |
| CASTAÑEDA PÉREZ CRISTIAN EDUARDO |                                                                                    | 21030140         |
| <br>                             |                                                                                    |                  |
| PRÁCTICA NO.                     | NOMBRE DE LA PRÁCTICA                                                              | DURACIÓN (HORAS) |
| 2                                | Tarea en equipo (Video:Por qué ASML tiene el Monopolio Más COLOSAL de la Historia) | 2 Hora           |

| 1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | INTRODUCCIÓN |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|
| <ul style="list-style-type: none"> <li>La industria de los semiconductores es uno de los pilares fundamentales del mundo moderno, permitiendo el desarrollo de dispositivos cada vez más rápidos y eficientes. En este contexto, el video analiza el papel de ASML, una empresa que ha revolucionado la fabricación de chips a través de su tecnología de litografía ultravioleta extrema (EUV). A lo largo del contenido, se expone cómo esta compañía ha superado obstáculos, ha apostado por la innovación y ha logrado consolidarse como un monopolio en la producción de las máquinas más avanzadas para la manufactura de semiconductores.</li> <li>En este análisis, se abordará el impacto de ASML en la industria tecnológica, la importancia de la litografía en la fabricación de chips y cómo la constante miniaturización de los transistores ha sido clave en el avance de la computación. Además, se incluirá una opinión crítica sobre los desafíos que enfrenta la compañía y el futuro de esta tecnología.</li> </ul> |              |

| 2                                                                                                                                                                                                                                                                                                                                                        | OBJETIVO (COMPETENCIA) |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------|
| <ul style="list-style-type: none"> <li>Conocer los principios físicos en los que se basa la litografía EUV así como sus componentes clave de las máquinas de litografía EUV, los diferentes pasos del proceso de fabricación de chips con tecnología EUV y el impacto de la tecnología EUV en el desarrollo de chips más pequeños y potentes.</li> </ul> |                        |

3

### MARCO TEÓRICO REFERENCIAL

En esta práctica estamos revisando un vídeo con diferentes conceptos importantes para la asignatura. No solo para la asignatura, sino también para un ingeniero en sistemas, realmente se revisan temas que son fundamentales para la industria de las computadoras que se debe entender plenamente si se quiere utilizar esta información el futuro para algún proyecto profesional o para desarrollo personal del ingeniero.

Entre los **conceptos** revisados se encuentran:

- **Litografía:** Técnica antigua de impresión en la cual reproducimos un dibujo o grabado que hemos realizado sobre la superficie de una piedra al estamparlo sobre una hoja de papel.
- **ASML:** La compañía holandesa lidera la producción de equipos de fotolitografía para la industria de los semiconductores.
- **Espectro Electromagnético:** En física, se denomina espectro electromagnético al conjunto de todos los tipos de radiación que se desplazan en ondas, es decir, al conjunto de todas las ondas electromagnéticas
- **Monopolio:** Situación del mercado en la que un productor o vendedor es el único que explota un bien o un servicio.

4

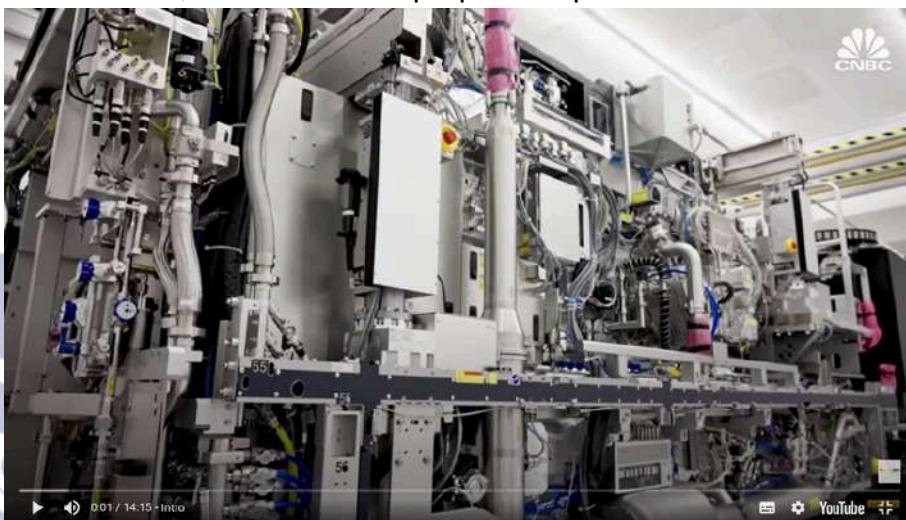
### MATERIALES UTILIZADOS

- Computadora
- Video de youtube “Por qué ASML tiene el Monopolio Más COLOSAL de la Historia”
- Documento compartido, para el trabajo colaborativo

5

## DESARROLLO

En la introducción del vídeo se explica lo que es a grandes rasgos una maquina avanzada de litografía y para que es usada, lo realmente importante que es hoy en día y que es capaz de generar transistores 10,000 veces más pequeños que un cabello humano.



Img 1. Máquina litográfica

Este profesor cuenta lo que representa la empresa holandesa ASML para la industria actual, básicamente un monopolio de estas máquinas indispensables para el desarrollo de microprocesadores y microchips.

Esta empresa es la única en el mundo que fabrica estas maquinas.



Img 2. Profesor explicando

5

## DESARROLLO

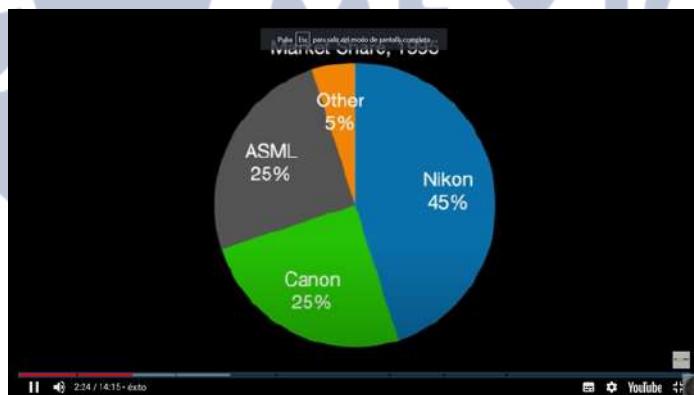
En 1984 dos empresas dedicadas a la tecnología se unieron para crear ASML.



Desde de este inicio ya se había fijado la meta de que esta empresa fuese dedicada a la litografía avanzada.

*Img 3: Creación de ASML*

Estas empresas eran Phillips y ASM, en esta época apenas aparecían los primeros teléfonos móviles y surgían las primeras computadoras de escritorio. Para 1986 ya habían lanzado la primera máquina de fotolitografía, para imprimir diseños sobre silicio y así fabricar chips.



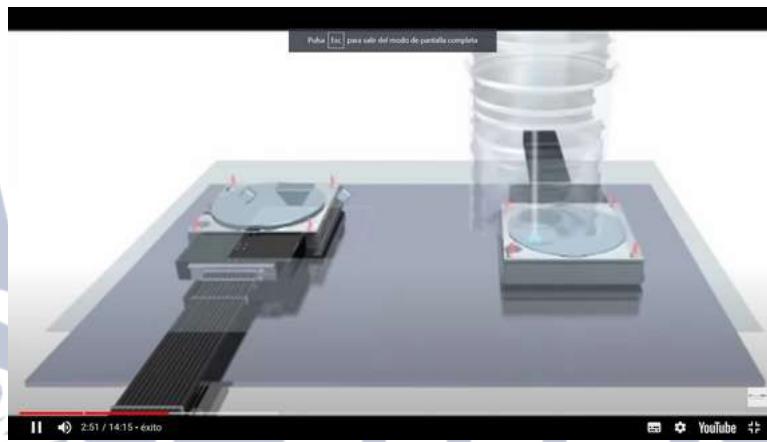
*Img 4. Representación gráfica de las empresas que dominaban el mercado*

Para ese entonces no era muy popular este mercado y tan solo otros dos fabricantes podían competir con ASML, las cuales fueron Nikon y Canon, estas dos marcas se apoderaron de la mayoría del mercado 12 años después del nacimiento de ASML

5

## DESARROLLO

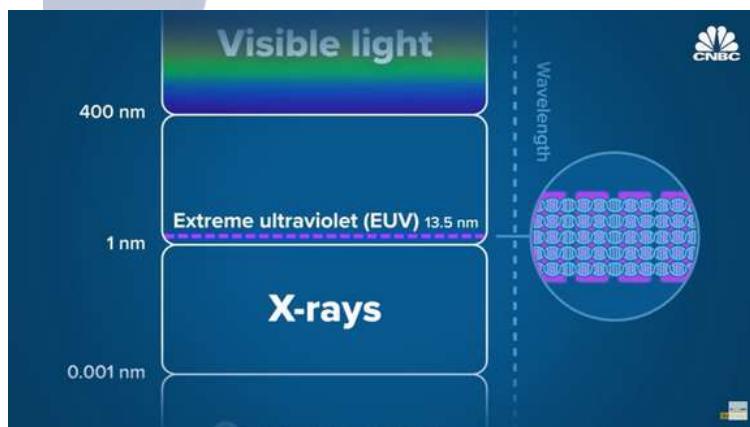
Para superar a sus competidores ASML, invirtió todos sus recursos en la fabricación de una nueva máquina más avanzada y eficiente, llamada Twin Scan, esta máquina era más rápida que la de la competencia debido a que podía medir e imprimir al mismo tiempo, logrando que se fabricaran chips 3 veces más rápido.



Img 5. Máquina Twin Scan

Al cabo del tiempo la competencia no pudo seguir el paso de ASML

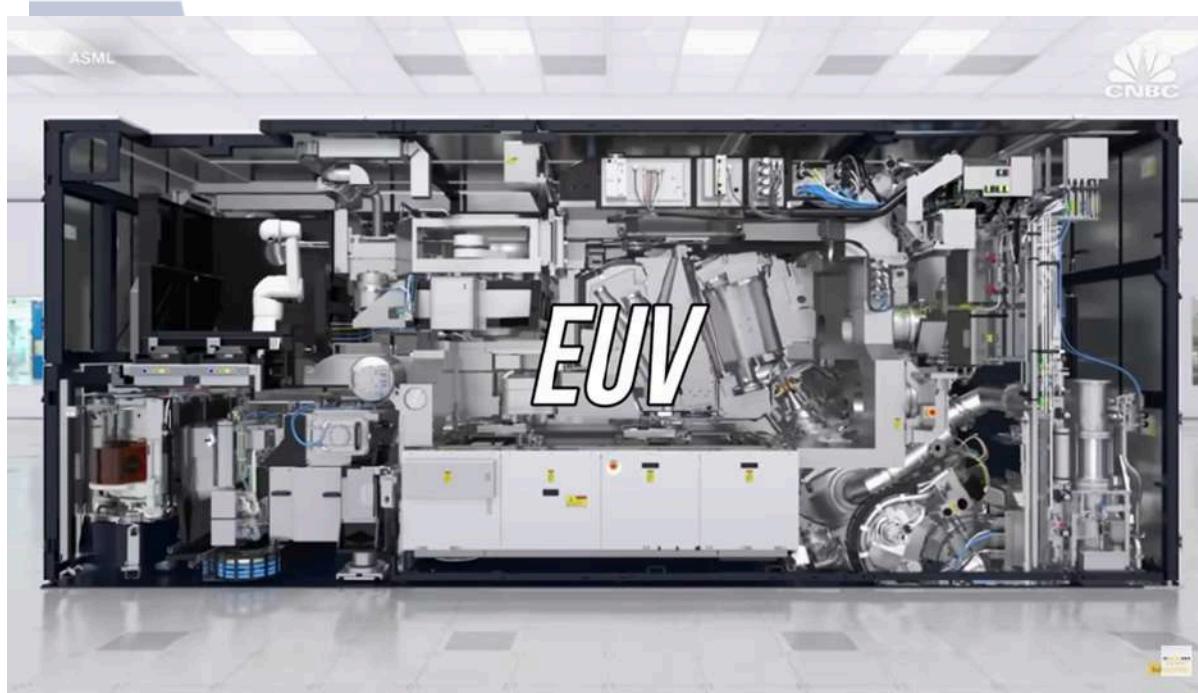
Más tarde ASML ideó otra estrategia para ganar ventaja, consistía en crear chips más pequeños y de esta manera estos fueran más rápidos, supuestamente el tamaño se relacionaba con la velocidad del mismo. Para lograr esto creyeron que era mejor usar una luz con una mayor frecuencia (13.5 nm).



ASML quiso generar una máquina capaz de usar una luz de esta frecuencia más alta con el objetivo de imprimir transistores muy pequeños, lo que resulta en chips más rápidos.

Img 6: Frecuencias de luz

ASML se enfrentaba a otro reto, fabricar esta máquina suponía un riesgo debido al alto costo y la investigación que requería, a su vez no era seguro que esto iba a funcionar o si los clientes estarían contentos, por lo que involucraron a sus clientes más grandes en esto, para finalmente poder construirla. Teniendo a clientes como parte de las acciones de mercado de ASML.



*Img 7.;Máquina EUV*

La máquina fue nombrada EUV, y con esta fue posible que ASML se convirtieran en el monopolio de estas máquinas y en la fabricación de chips de alta velocidad

### La maquina EUV

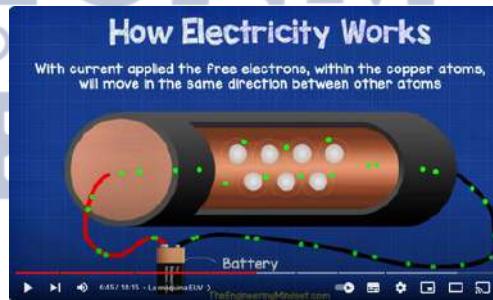
para entender la importancia de la maquina EUV o Extreme Ultraviolet, tenemos que verlo a nivel de la fisica y la computacion.

Los chips son una de las mayores invenciones del siglo XX, pero si hacemos zoom en uno de estos chips, encontramos billones de pasadisos diseñados cuidadosamente para el envio de unos y ceros, pues cada numero, cada letra, cada instruccion incluso cada pixel es un numero en codigo binario (unos y ceros), con los cuales se pueden formar letras, imagenes, videos, etc. es por eso que cada que usamos un celular o una comutadora, el microprocesador debe realizar sienos de miles de operaciones binarias para darnos la informacion que solicitamos, cuantas mas operaciones haga el chip, mas rapido sera.

Cuando los electrones de los atomos se mueven en un flujo constante le llamamos electricidad, pero un flujo continuo no lo podemos interpretar como unos y ceros, para eso necesitamos materiales que nos ayuden a conducir la electricidad (conductor) para interpretar esto como un uno, pero tambien sirvan como aislantes y no conduzcan la electricidad para interpretar esto como cero, a estos materiales se les denomina semiconductores



Img. 8. Nivel de fisica/computación



Img. 9. Funcionamiento de la electricidad



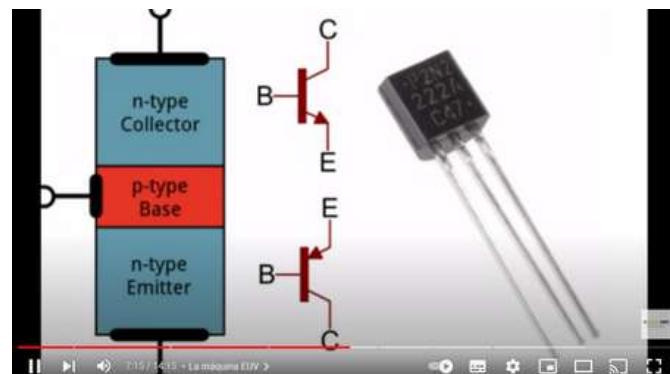
Img. 10. Funcionamiento de los semiconductores

5

## DESARROLLO

Pero debemos poder crear unos y ceros de forma controlada, para eso creamos los transistores.

Los transistores se diseñan como una serie de caminos por donde circula la electricidad, para que puedan darle paso o no, así se generan pulsos eléctricos, así que funcionan como un interruptor

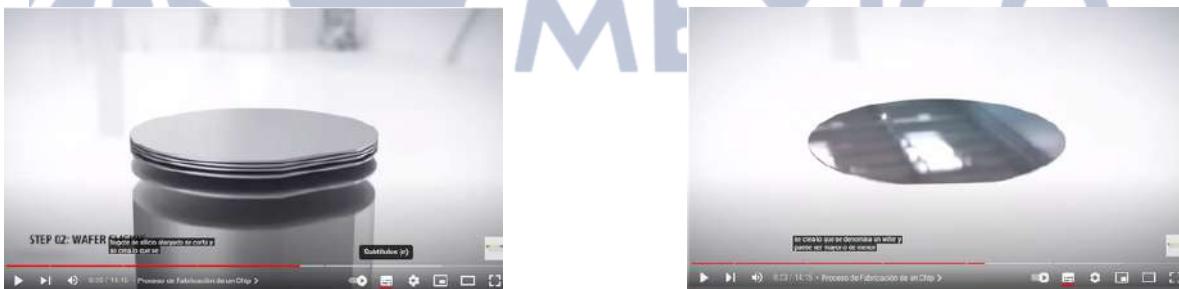


Img. 11 Transistores

Cuantos mas transistores o interruptores tenga el microprocesador, mas rápido será. entonces si queremos mas rapidez, necesitamos crear transistores más pequeños; es ahí donde entra la máquina EUV.

El proceso de fabricación de un chip, comienza recolectando arena, por sus altos niveles de silicio, se usa el silicio por su disponibilidad, y por su estructura atómica ya que al combinarse con otros elementos, fácilmente se puede hacer semiconductor.

Tras un proceso de purificación de la arena, se obtiene un lingote de silicio alargado, se corta y se obtiene un WIFER que pueden ser de distintos tamaños



Img. 12 y 13 Fabricación de un chip

Aquí es donde se van a ir depositando los materiales capa a capa, la primera capa que se agrega es una de resistencia a la luz, para que cuando reciba la luz se endurezca, pero no se trata de enfocar una luz a todo el wifer, sino que la luz pasa por una fotomáscara

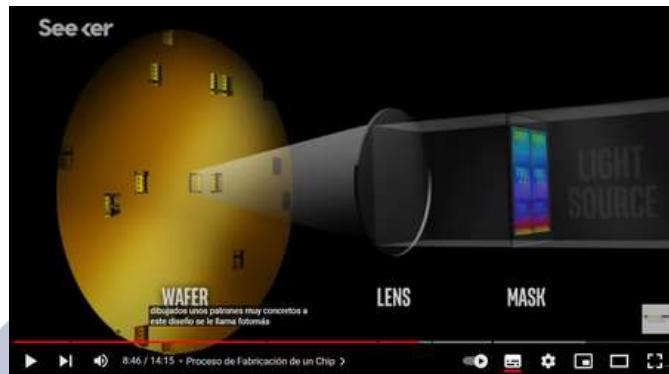


Img. 14. Fotomáscara

5

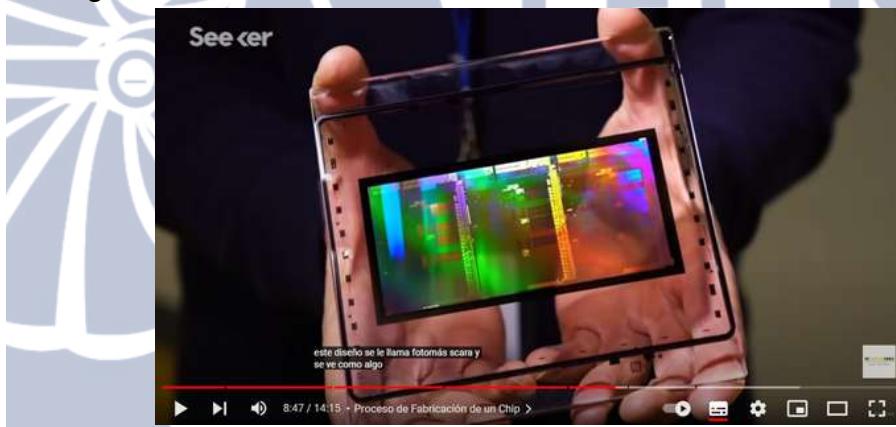
## DESARROLLO

Esta fotomáscara es una especie de cristal con unos diseños muy concretos, por donde pasa la lus UV y genera estos diseños en el wifer



Img. 15. Fotomáscara.

Y se ve algo asi



Img. 16 Diseño de fotomáscara

Se aplicara la luz UV por la  
mascara y cuando hacemos  
zoom, vemos como la luz va  
dejando unos caminos



Img. 17; Realización de fotomáscara

5

## DESARROLLO

Como es posible la maquina de ASML (EUV) de hacer esta luz UV



Img. 18 Luz que genera la máquina ASML

se lanza unas 50,000 gotas de estaño por segundo y son golpeadas por láser provocando pequeñas explosiones que generan plasma

Este proceso es conocido como ablación láser. Las altas temperaturas generadas en el plasma permiten que el estaño se evapore rápidamente, creando una nube de partículas cargadas que se condensan para formar una capa delgada y uniforme sobre la superficie deseada. La precisión y velocidad de este método lo convierten en una técnica ampliamente utilizada en la industria para la deposición de películas delgadas en diversos sustratos.

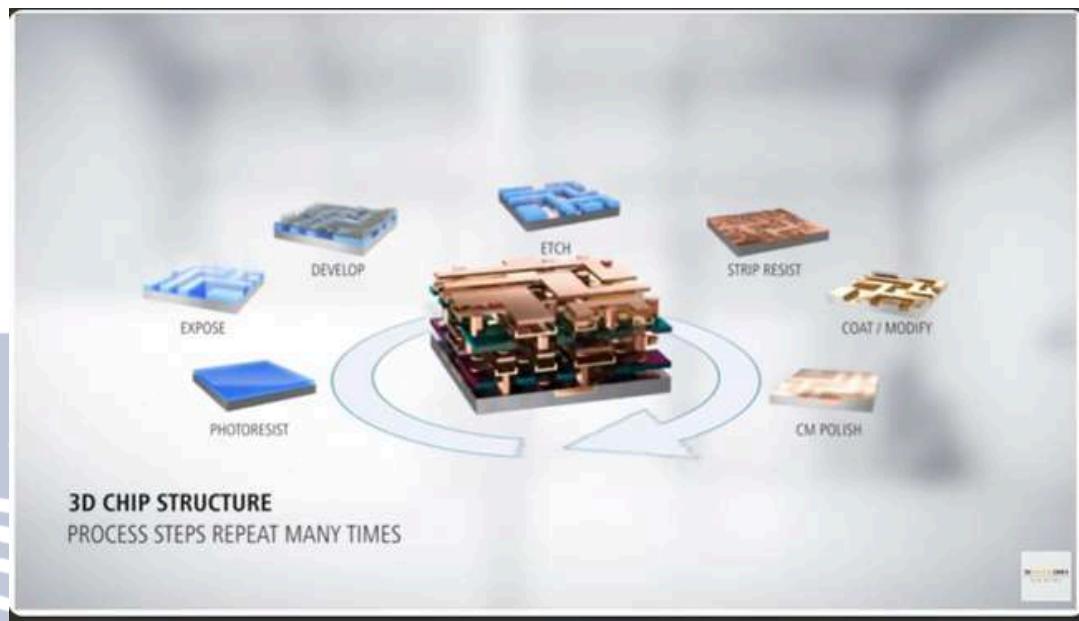


Img. 19 Construcción del chip

La luz ultravioleta imprime los caminos de la información dentro del chip

y permite que los datos fluyan a través de los circuitos minúsculos con una velocidad impresionante. Este proceso de grabado de información es fundamental para el funcionamiento de dispositivos electrónicos cada vez más avanzados. La tecnología detrás de este fenómeno es fascinante, ya que la luz ultravioleta tiene la capacidad de alterar las propiedades de los materiales en una escala microscópica, creando así un intrincado entramado de información que impulsa nuestra sociedad hacia un futuro cada vez más digitalizado y conectado.

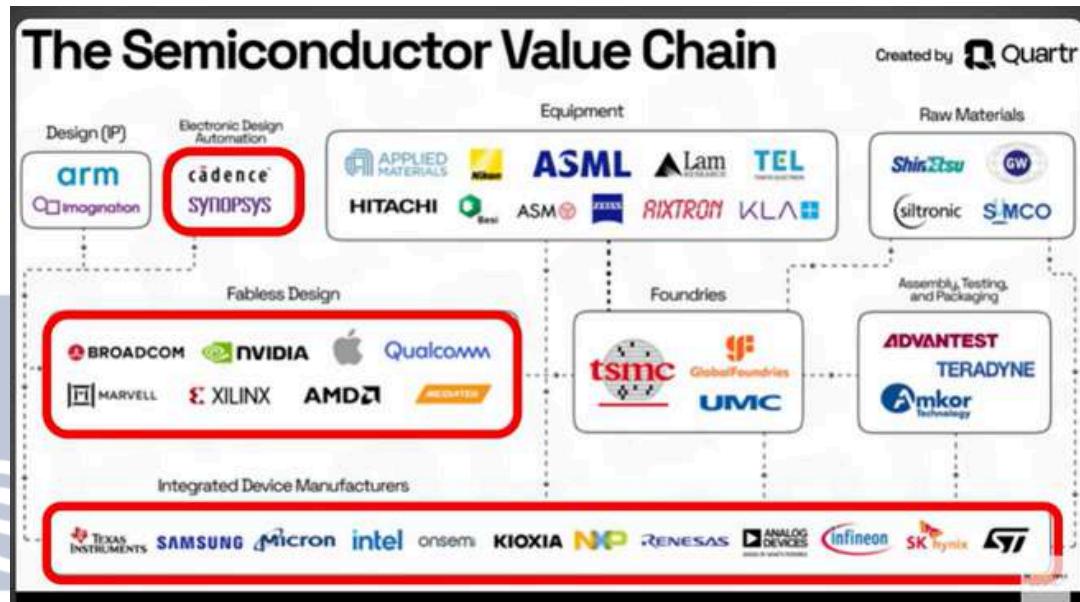
DESARROLLO



Img. 20. Estructura del 3D chip

se aplican capas de distintos materiales para aislar, ionizar silicio y para que circule la electricidad

Una técnica común utilizada en la fabricación de dispositivos electrónicos es la deposición de capas de distintos materiales para aislar, ionizar el silicio y permitir el flujo de electricidad de manera controlada. Este proceso es fundamental para garantizar el correcto funcionamiento de los componentes y la eficiencia de los circuitos integrados. La combinación de estos materiales ayuda a proteger los componentes delicados de posibles daños, asegurando así la integridad y estabilidad de los dispositivos electrónicos.



Img. 21 Bolsa de valor de los semiconductores

A pesar de que existen otras empresas que se encargan de la creación de otros componentes ASML es líder por la calidad y rapidez de sus componentes y la tecnología que desarrolla los principales clientes son TSMC y Samsung

ASML se destaca en la industria por su compromiso con la excelencia en la creación de componentes de alta calidad y su enfoque en la tecnología. Su liderazgo se manifiesta en la velocidad con la que desarrolla sus productos, satisfaciendo las demandas de sus principales clientes, como TSMC y Samsung. La confianza depositada en ASML por estas empresas líderes es un testimonio de su reputación y su capacidad para ofrecer soluciones avanzadas en el mercado de componentes tecnológicos.



Img. 22. ASML tiene contratos únicos con proveedores

En un negocio comúnmente tiene éxito mientras la demanda es más grande que la oferta pero en los semiconductores se maneja de diferente forma ASML se le ha permitido continuar con su monopolio ya que cuenta con clientes que invierten en sus equipos principalmente cuando la oferta es baja por ello no hay competencia ya que necesitarían contratos con proveedores menos especializados para tener un lugar en la industria.



Img. 23. Monopolio de ASML

La industria de los semiconductores aún sigue siendo muy rentable especialmente desde el uso de los celulares inteligentes hoy hasta la industria 4.0 con campos como inteligencia artificial y el internet de las cosas seguir haciendo muy vigente hasta que existan nuevas técnicas o tecnologías capaces de reemplazar las ahorita vistas

**6**

**BITÁCORA DE INCIDENCIAS**

Sin incidencias, todos los integrantes trabajaron de forma colaborativa.

**7**

**OBSERVACIONES**

Hubo muy poco tiempo para la realización de la actividad

**8**

**ANEXOS**

**9**

**REFERENCIAS**

- [https://www.youtube.com/watch?v=zAYCfw\\_syFc](https://www.youtube.com/watch?v=zAYCfw_syFc)