1ª Guía Compiladores

Nombre: Santos Méndez Ulises Jesús Grupo: 3CM14 Fecha: 19/04/2023

-Defina **compilador**

Programa que traduce programa escrito en lenguaje fuente a uno equivalente en lenguaje objeto

-Cuales son las dos partes de la compilación

1.- Análisis
2.- Sintesis

-Describa las 6 fases de un compilador

- **Analizador léxico:** En el que la cadena de caracteres que constituye el programa fuente se lee de izquierda a derecha y se agrupa en componentes léxicos.

- **Analizador sintáctico**: En el que los componentes léxicos se agrupan jerárquicamente en colecciones anidadas con un significado colectivo.

- **Analizador semántico:** En el que se realizan ciertas revisiones para asegurar que los componentes de un programa se ajustan de un modo significativo.

- **Generador de código intermedio:** Se puede considerar esta representación intermedia como un programa para una máquina abstracta.

- **Optimizador de código:** Trata de mejorar el código intermedio, de modo que resulte un código de máquina más rápido de ejecutar.

- **Generador de código:** La fase final de un compilador es la generación de un código objeto, por lo general consiste en código de máquina relocalizable o código ensamblador.

-Cuales son los 8 modulos de un compilador

- **Administrador de la tabla de símbolos.**
- **Analizador léxico.**
- **Analizador sintáctico.**
- **Analizador semántico.**
- **Generador de código intermedio.**
- **Optimizador de código.**
- **Generador de código.**
- **Manejador de errores.**

Falso o verdadero (F/V)

0.-A los terminales se les llama asi porque no pueden ser sustituidos	(V)
1.-Que una secuencia de caracteres concreta sea un token depende del lenguaje	(F)
2.-Las cadenas que pertenecen al lenguaje generado por una gramatica estan hechas solo de terminales	(V)
3.-El **análisis léxico** lee la cadena de entrada de derecha a izquierda	(V)
4.-El **análisis léxico** construye el árbol de análisis sintáctico	(F)
5.-La secuencia de caracteres que forma un componente léxico es el **lexema** del componente	(V)
6.-La gramática S → **a**S | S**a** | **a** se puede analizar con un **análizador sintáctico predictivo descendente recursivo**	(V)
7.-El tipo de **yylval** no es el mismo que el de los elementos en la pila de YACC	(V)
8.-La unica forma de indicar el tipo de los elementos en la pila de YACC es usando #define YYSTYPE	(V)
9.-El **código intermedio** debe ser fácil de generar	(V)
10.- Un **esquema de traducción** es una **GLC** + **reglas semanticas**	(F)
11.- Arbol de análisis sintáctico con anotaciones es sinonimo de **árbol decorado**	(V)
12-Análisis sintáctico **descendente** es donde la construcción del árbol de análisis sintáctico se inicia en las hojas y avanza hacia la raíz	(F)
13-Análisis sintáctico **ascendente** es donde la construcción del árbol de análisis sintáctico se inicia en las hojas y avanza hacia la raíz	(V)
14.-yylex() llama a yyparse()	(F)
15.-yyparse() llama a yylex()	(V)
16.-yylex() retorna el **tipo** de **token**	(V)
17.-yylval almacena el **lexema**	(V)
18-HOC1 es una **calculadora**	(V)
19-Las **variables** en HOC son de tipo entero	(F)
20.-La notación **posfija** es una notación matemática libre de paréntesis y en esta notación los operadores aparecen después de los operandos	(V)
21.-La **raiz** del árbol de análisis sintáctico se etiqueta con el simbolo inicial	(V)
22.- Las **hojas** del árbol de análisis sintáctico se etiquetan con no terminales	(F)
23.-En la notación **infija** la **asociatividad** y la **precedencia** se usan para determinar en que orden hay que realizar las operaciones para **evaluar** una expresion	(V)

Para que sirve el **Análisis Léxico**

a) Para generar el código en lenguaje objeto b) Nos dice si una cadena pertenece al lenguaje generado por una gramática (C)

c) Para dividir una cadena en tokens d) Los compiladores no lo necesitan nunca

El_____ comprueba que el orden en que el **analizador léxico** le va entregando los tokens es válido.

a) analizador semantico b) analizador sintáctico c) optimizador d) generador de codigo (B)

Es una gramática que tiene cuatro componentes:

1. Un conjunto de componentes léxicos.
2. Un conjunto de no terminales.
3. Un conjunto de producciones, en el que cada producción consta de un no terminal, llamado lado izquierdo de la producción, una flecha y una secuencia de componentes léxicos y no terminales, o ambos, llamado lado derecho de la producción.
4. .La denominación de uno de los no terminales como símbolo inicial.

a) Gramática Asociativa por la izquierda b) Gramática recursiva (C)

c) Gramática libre de contexto (GLC) d) Gramática ambigua

Cual de las sigs. opciones no es sinónimo de las otras

a) Componente léxico b) no terminal c) token d) Simbolo gramatical (B)

Es una gramática donde en el lenguaje que genera existe una cadena que tiene mas de un árbol de análisis sintáctico.

a) Gramática recursiva por la izquierda b) Gramática recursiva (D)

c) Gramática libre de contexto d) Gramática ambigua

Si Una gramática contiene una regla de producción de la forma $A \rightarrow A\ \alpha$ entonces es una

a) Gramática recursiva por la izquierda b) Gramática ambigua (A)

c) Gramática libre de contexto d) ninguna de las anteriores

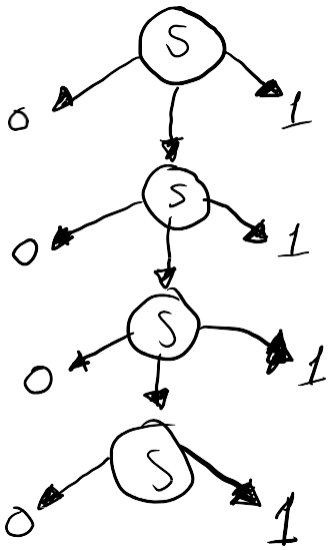
Considere la siguiente gramática

S→ **0** S **1** | **01**

a) Mostrar una derivación de **00001111**

S → 0S1
→ 00S11
→ 000S111
→ 00001111

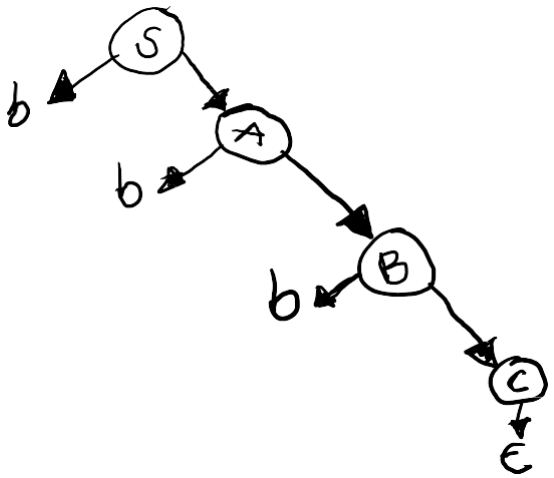
b) Dibuje el árbol de análisis sintáctico para la entrada **00001111**



Considere la siguiente gramática
S → **b**A
A → **b**B
B → **b**C
C → **ε**

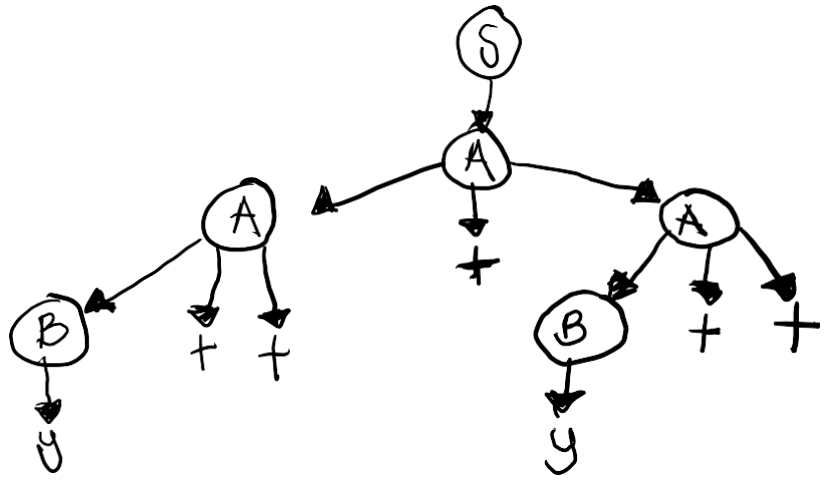
a) Mostrar una derivación de **bbb**
S → bA
→ bbB
→ bbbC
→ bbb

b) Dibuje el árbol de análisis sintáctico para la entrada **bbb**



Considere la siguiente gramática
S → *A*
A → *A*+*A* | *B*++
B → ***y***
a) Mostrar una derivación de **y + + + y + +**
S → A
→ A + A
→ B + + + B + +
→ y + + + y + +

b) Dibuje el árbol de análisis sintáctico para la entrada **y + + + y + +**

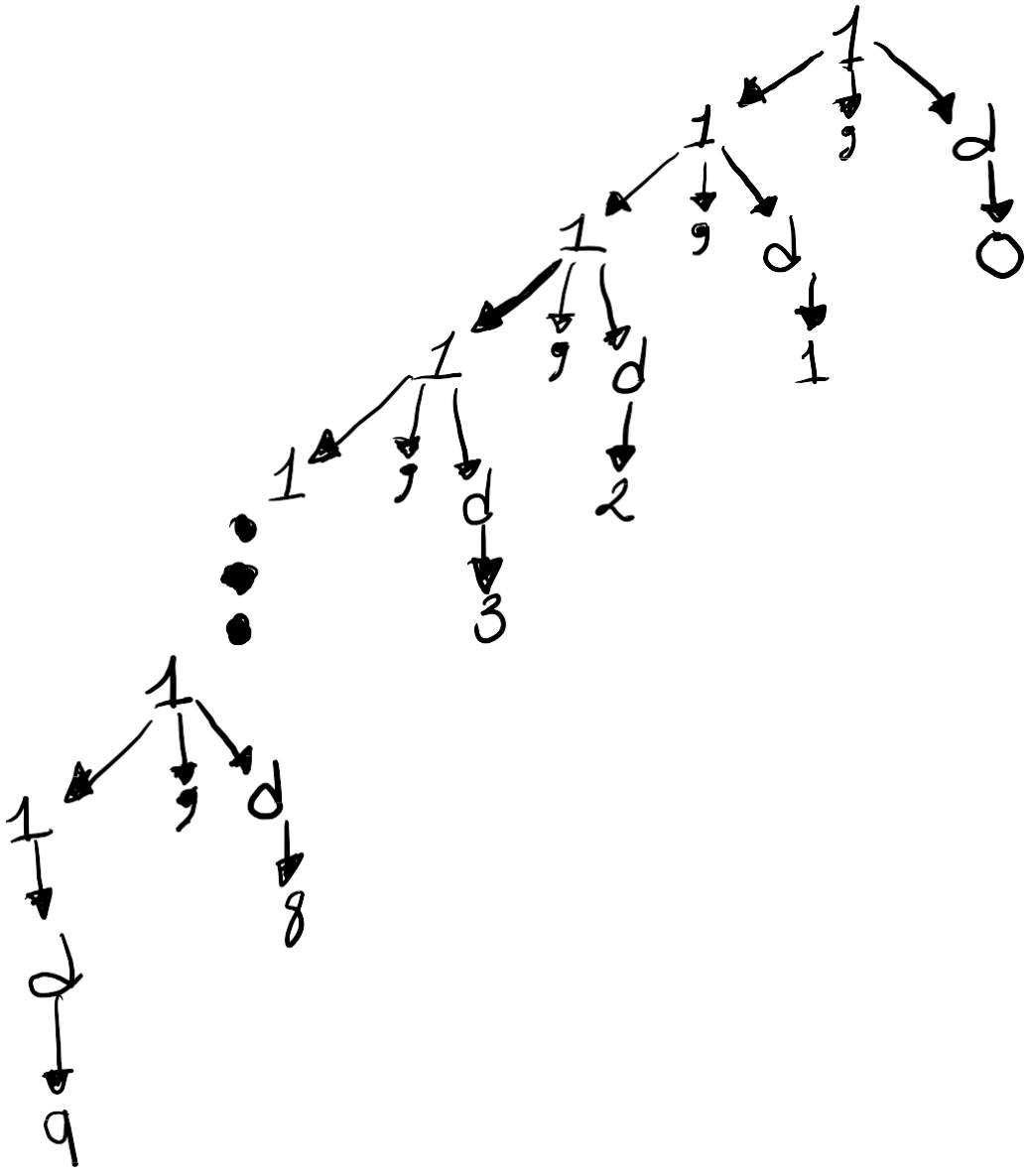


Considere la siguiente gramática
l→ *l* **,** *d* | *d*
d→ ***0*** | ***1*** | ***2*** | ***3*** | ***4*** | ***5*** | ***6*** | ***7*** |***8*** | ***9***

a) Mostrar una derivación de ***9,8,7,6,5,4,3,2,1,0***

l → l, d
→ l, d, d
→ l, d, d, d
→ l, d, d, d, d
→ l, d, d, d, d, d
→ l, d, d, d, d, d, d
→ l, d, d, d, d, d, d, d
→ l, d, d, d, d, d, d, d, d
→ d, d, d, d, d, d, d, d, d, d
...
→ 9, 8, 7, 6, 5, 4, 3, 2, 1, 0

b) Dibuje el árbol de análisis sintáctico para la entrada ***9,8,7,6,5,4,3,2,1,0***



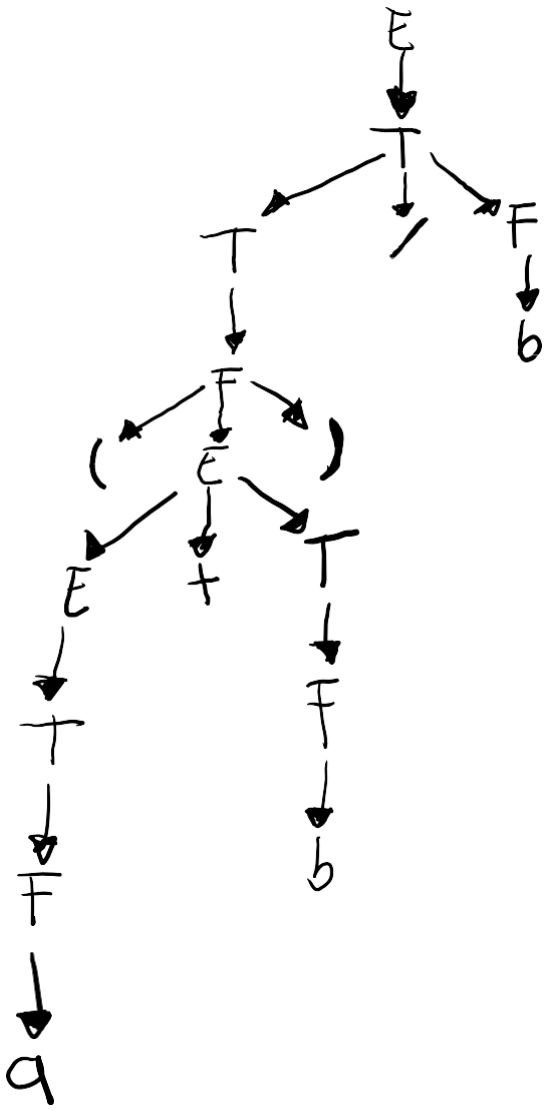
Dada la gramática
T={**a**, **b**, +, -, *, /, (,)}, N={E, T, F} S={E}
P={ E->T | E+T | E-T
T->F | T*F | T/F
F-> **a** | **b** | (E) }
y la cadena **(a+b)/b**

a) Obtenga una derivación de dicha cadena

E → T
T → T/F
E → T/F
T → F
E → F / F
F → (E)
E → (E)/F
F→ b
E → (E)/b
E → E + T
E → (E+T)/b

E →T
E →(T+T)/b
T→F
E → (F+T)/b
T→F
E → (F+F) / b
F → a
E → (a+F)/b
F → b
E →(a+b)/b

b) Dibuje el árbol de análisis sintáctico que corresponde a la cadena mencionada



Análisis sintáctico predictivo descendente recursivo

Considere la siguiente gramática

S →**a** | **(** S **)**

Escriba el analizador sintáctico predictivo descendente recursivo

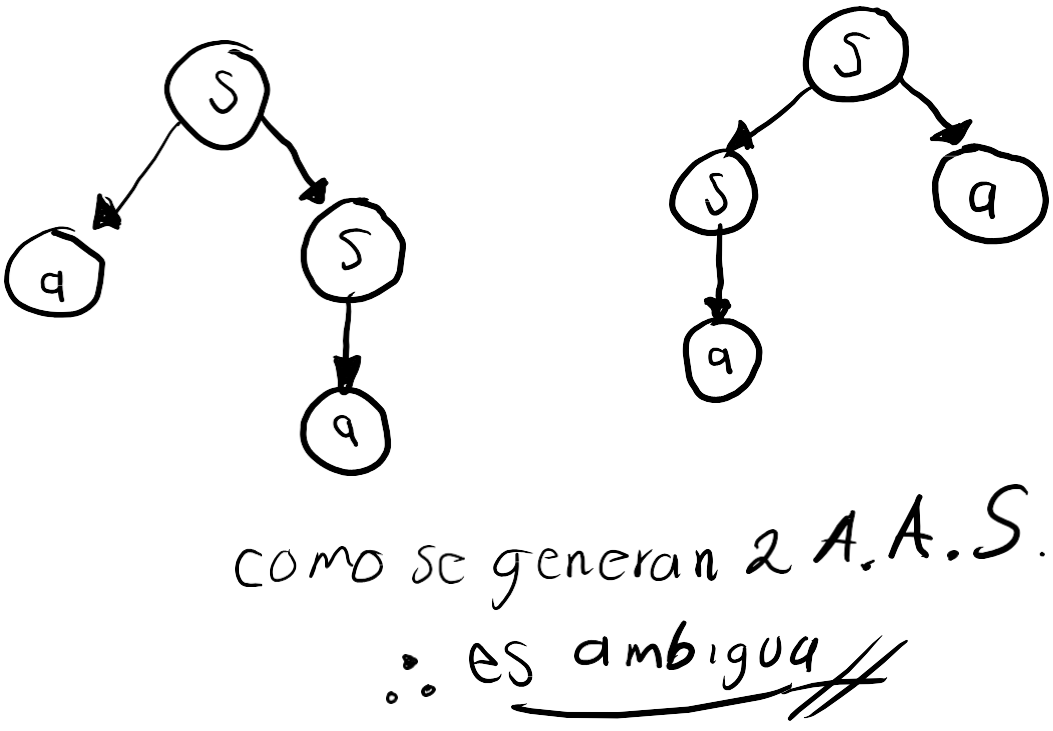
```
void parea(complex t){
        if(preana == t) preana = sigcomplex();
        else error();
}
void S(){
        if(preana == 'a')
                parea('a');
        else if(preana == '('){
                parea('(');S();parea(')');
        }else
                error();
}
void main(){
        preana = sigcomplex();

}
```

Ambigüedad

Demostrar que la siguiente gramática es ambigua

S → **a**S | S**a** | **a**

usando la cadena **aa**



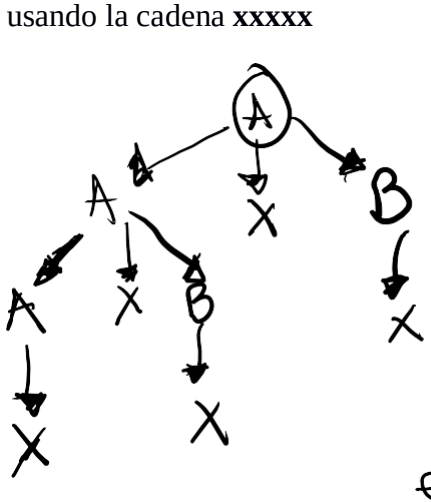
como se generan 2 A.A.S.

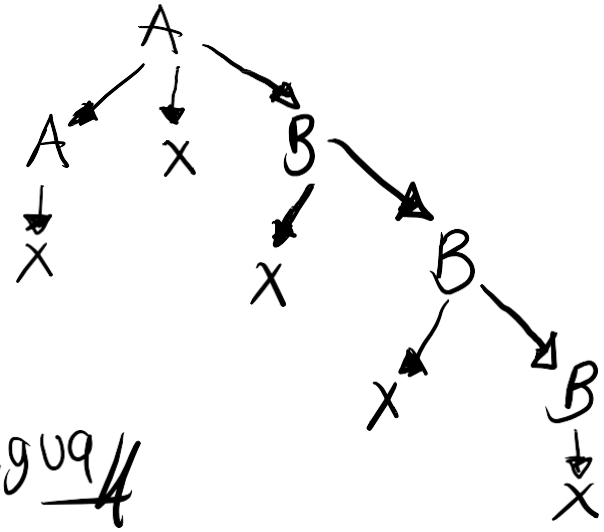
∴ es ambigua //

Demostrar que la siguiente gramática es ambigua

A →A **x** B | **x**
B → **x** B | **x**

usando la cadena **xxxxx**



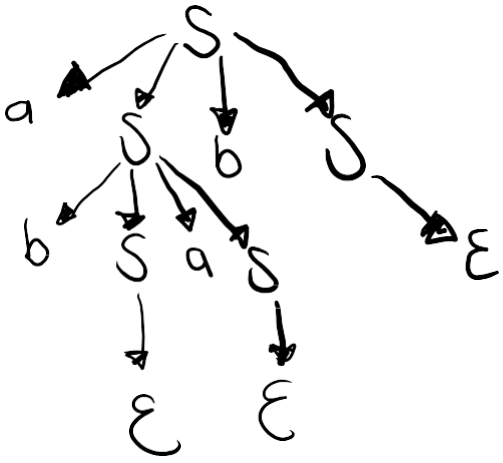


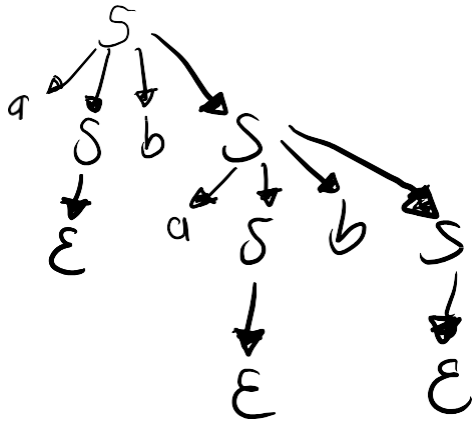
es ambigua

Demostrar que la siguiente gramática es ambigua

S→ **a** S **b** S | **b** S **a** S | **ε**

usando la cadena **abab**



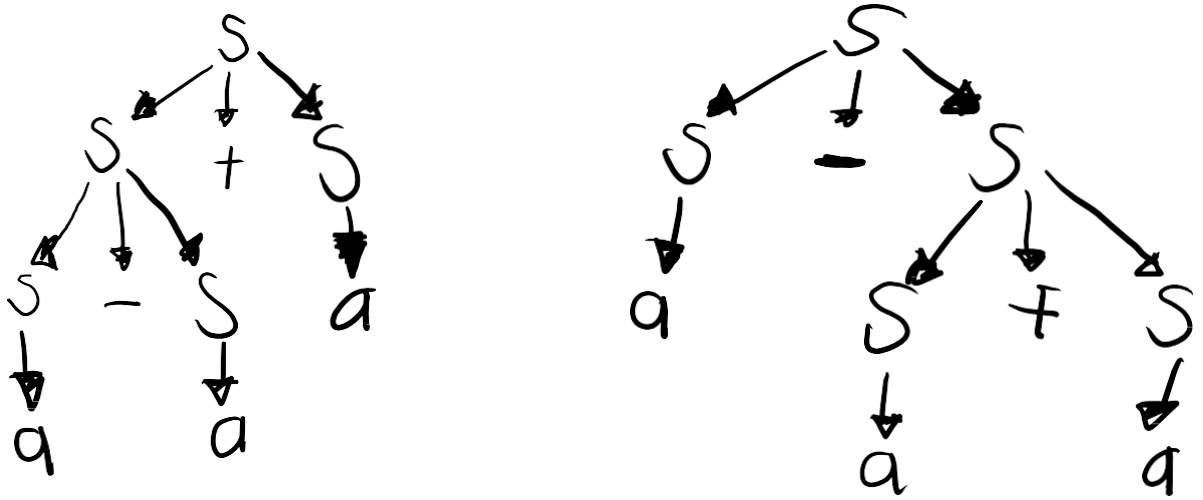


es ambigua

Verificar si las siguientes gramáticas son ambiguas

$S \rightarrow S + S \mid S - S \mid \mathbf{a}$

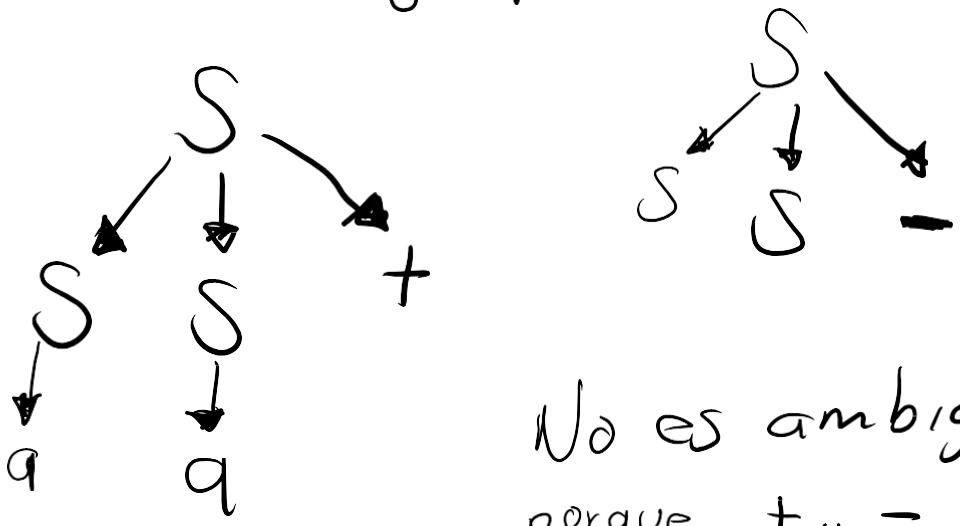
considerando a-a+a



es ambigua.

$S \rightarrow SS + \mid SS - \mid \mathbf{a}$

considerando aa+



No es ambigua porque + y - al estar al final obligan a que las cadenas solo tengan un A.A.S.

Recursividad por la **izquierda**
Para eliminar la recursividad por la izquierda

A → A**a** | **b**

se transforma en

A → **b** | **b**R
R → **a**R | **ε**

Ahora considere las siguientes gramáticas

A→ **1** | A **0**

y

S→ **(** L **)** | **a**
L→ L **,** S | S

Elimine la recursividad por la izquierda de dichas gramáticas.
A→ **1** | A **0**
Pasa a

A→ 1R | 1
R → 0R | ε

S→ **(** L **)** | **a**
L→ L **,** S | S
Pasa a

L → S R | S
R → , S R | ε
S → (L) | a

Escriba el analizador sintáctico predictivo descendente recursivo para dichas gramáticas
Escriba la sección de reglas de la especificación de YACC para dichas gramáticas
Primer gramática

```
void A(){
        if(preana=='1R')
                parea('1'); R();
        elseif(preana=='1')
                parea('1');
        else
                error();
}
void R(){
        if(preana=='0')
                parea('0'); R();
        else;
}
```

Segunda gramática

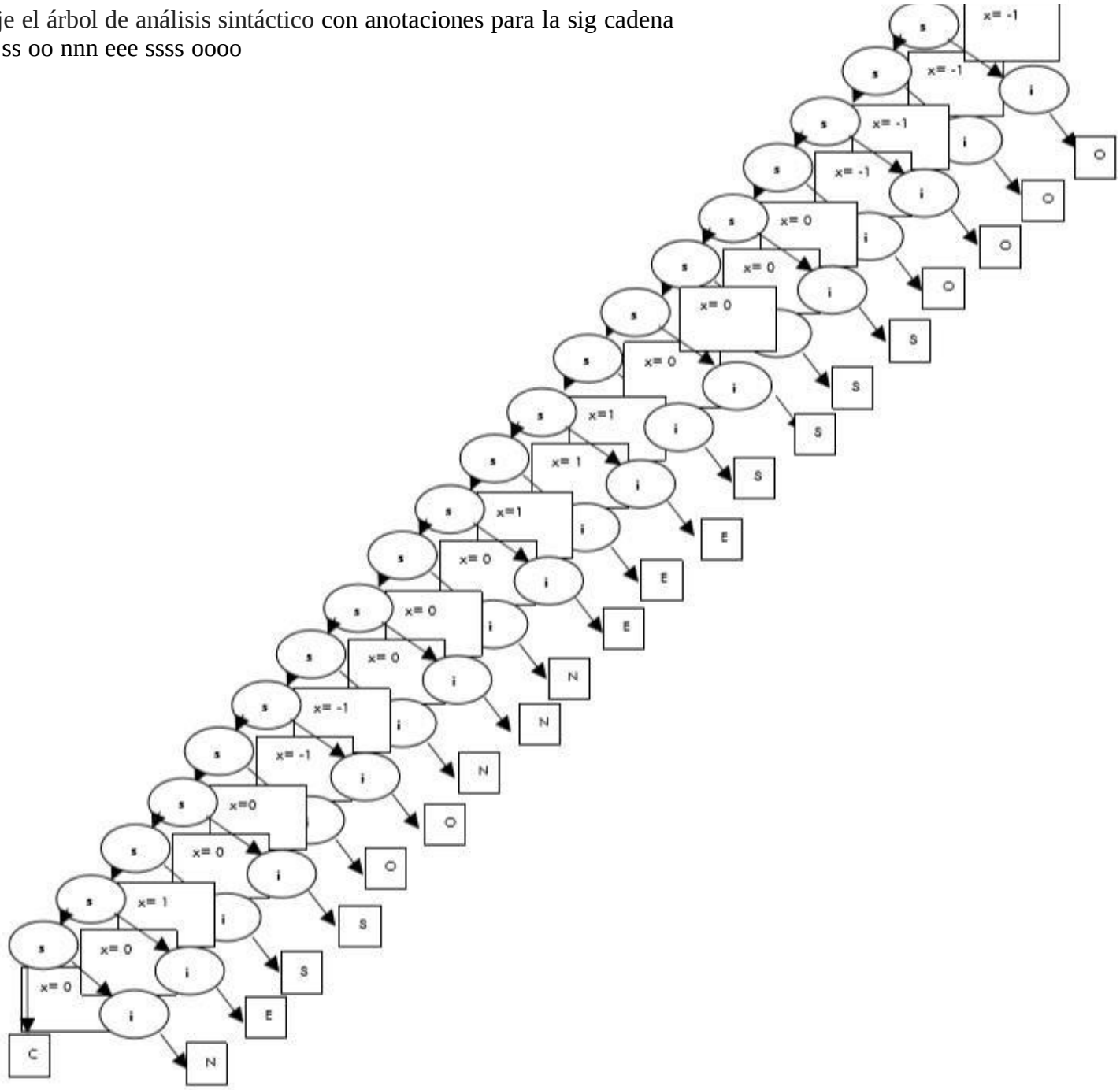
```
void L(){
        if(preana=='SR')
                S(); R();
        elseif(preana=='S')
                parea('S');
        else
                error();
}
```

```
void R(){
        if(preana==’S’)
                S(); R();
        else;
}
void S(){
        if(preana==’(’)
                parea(‘(’); L();parea(‘)’);
        elseif(preana==’a’)
                parea(‘a’);
        else
                error();
}
```

Definiciones dirigidas por la sintaxis

PRODUCCIÓN	REGLA SEMÁNTICA
sec ⟶ **comienza**	*sec.x = 0* *sec.y = 0*
sec ⟶ *sec₁ instr*	*sec.x = sec₁.x + instr.dx* *sec. y = sec₁.y + instr.dy*
instr ⟶ **este**	*instr.dx = 1* *instr.dy = 0*
instr ⟶ **norte**	*instr.dx = 0* *instr.dy = 1*
instr ⟶ **oeste**	*instr.dx = -1* *instr.dy = 0*
instr ⟶ **sur**	*instr.dx = 0* *instr.dy = -1*

Dibuje el árbol de análisis sintáctico con anotaciones para la sig cadena
c n e ss oo nnn eee ssss oooo



Escribir la sección de reglas de la especificación de yacc para calcular la posición final del robot.

```
%{
struct cord{
int x, y,dx,dy;
}
;
typedef struct cord cordenada;
#define struct cord cordenada
#define YYSTYPE struct cord
%}

%token  comienza este oeste norte sur
%%
        sec:    comienza {$$.x = 0; $$.y = 0;}
        |       sec instr {$$.x = $1.x + $2.dx; $$.y = $1.y + $2.dy;}
        instr:  este {$1.dx = 1; $1.dy = 0;}
        |       oeste{$1.dx = -1; $1.dy = 0;}
        |       norte{$1.dx = 0; $1.dy = 1; }
        |       sur{$1.dx = 0; $1.dy = -1;}
        ;
%%
```

Escriba una definición dirigida por la sintaxis para evaluar expresiones booleanas.

Esquemas de traducción

Escriba un esquema de traducción para convertir una expresión en:

infijo a postfijo **expr -> expr + termino { print (‘ + ’)}** **expr -> expr + termino { print (‘ - ’)}** **expr -> termino** **termino -> 0 { print (‘ 0 ’)}** **termino -> 1 { print (‘ 1 ’)}** **termino -> 9 { print (‘ 9 ’)}**	postfijo a infijo **expr -> + expr termino + { print (‘ + ’);}** **expr -> - expr termino - { print (‘ - ’);}** **expr -> termino** **termino -> 0 { print (‘ 0 ’)}** **termino -> 1 { print (‘ 1’)}** **termino -> 9 { print (‘ 9’)}**
infijo a prefijo **expr -> expr termino + { print (‘+’expr , termino}** **expr -> expr termino - { print (‘-‘ exp, termino) }** **expr -> termino** **termino -> 0 { print (‘ 0 ’)}** **termino -> 1 { print (‘ 1’)}** **termino -> 9 { print (‘ 9’)}**	prefijo a infijo **expr ->+ expr termino { print (expr , ‘+’, termino}** **expr -> - expr termino { print (exp, ‘-‘, termino) }** **expr -> termino** **termino -> 0 { print (‘ 0 ’)}** **termino -> 1 { print (‘ 1’)}** **termino -> 9 { print (‘ 9’)}**

Escriba un esquema de traducción para evaluar expresiones booleanas

Para cada esquema de traducción de arriba escriba la sección de reglas de la especificación de YACC

Escritura de Gramaticas

Escribir una gramática que genere todas las cadenas de longitud 4 formadas con los símbolos del alfabeto {a,b,c}

T={a,b,c}
NT={I,X}
S={I}
I → X X X X
X → a | b | c

Escribir una gramática que sirva para generar las siguientes cadenas

Especie perro	**Especie gato**	**Especie perro**	**Especie gato**
Edad 1	**Edad 2**	**Edad 2**	**Edad 2**
Sexo macho	**Sexo macho**	**Sexo hembra**	**Sexo macho**
Tamaño grande	**Tamaño mediano**	**Tamaño pequeño**	**Tamaño grande**
Colores negro , blanco	**Colores negro , blanco , café**	**Colores canela , gris**	**Colores blanco**
Soy rápido , activo, alegre	**Soy tranquilo , sociable**	**Soy fuerte , alegre, activo.**	**Soy listo , obediente**
Aficiones correr, comer	**Aficiones dormir, parrandear, comer**	**Aficiones aullar**	**Aficiones jugar, haraganear**

S-> especie + edad + sexo + tamaño + colores +soy + aficiones
Especie-> perro|gato
Edad -> 1|2
Sexo -> macho|hembra
Tamaño -> grande|mediano|pequeño
Colores -> colores,colores| colores|negro|blanco|café|canela|gris
Soy -> soy, soy|soy|rápido|activo|alegre|tranquilo|sociable|fuerte|listo|obediente
Aficiones -> aficiones, aficiones|aficiones|correr|comer|dormir|parrandear|aullar|jugar|haraganear

12.-Escribir una gramática que sirva para generar las siguientes cadenas

Etiquetado Nerd	Etiquetado Geek	Etiquetado Nerd	Etiquetado Freak
Nivel Junior	Nivel Senior	Nivel Junior	Nivel Senior
Sexo Hombre	Sexo Mujer	Sexo Mujer	Sexo Hombre
Lenguajes Java , C , Logo	Lenguajes Pascal , Prolog , SQL	Lenguajes PHP , Perl, Java	Lenguajes Ensamblador, C
Aficiones programar, videogames, comics, hackear, googlear	Aficiones chatear, videogames, programar	Aficiones hackear, googlear, gotcha, dormir	Aficiones gotcha, dormir, chatear, comics

S-> etiquetado + nivel + sexo + lenguajes + aficiones
Etiquetado -> nerd|geek|freak
Nivel -> junior|senior
Sexo -> hombre|mujer
Lenguajes -> lenguajes,lenguajes|lenguajes|java|c|logo|pascal|prolog|php|pearl|ensamblador
Aficiones-> Aficiones, aficiones | aficiones | programar | videogames | comics | hackear | googlear | chatear | dormir | gotcha

YACC

.-Para que sirve $$

Para asignar valores de una produccion

.-Dentro de una accion gramatical $n se refiere a la variable asociada al elemento n de lado derecho de una produccion

1.-Los %% se usan para indicar

a)inicio de la sección de declaraciones b)inicio de la sección de reglas (B)

c)precedencia de los operadores d)fin del código de soporte

2.-%token sirve para indicar

a)inicio de la sección de declaraciones d)los no terminales de la gramática (D)

c)precedencia de los operadores d)los terminales de la gramática

3.-Como le indica el analizador léxico (yylex) al analizador sintáctico (yyparse) que ya no hay mas tokens en la entrada

a) retornando cero b) retornando -1 (D)

c) almacenando -1 en yylval d) almacenando 0 en yylval

4.-Una acción gramatical debe ir entre

a) comillas b) paréntesis c) corchetes d) llaves (D)

5.-Considere la producción

S : S ' a' S 'b'

$4 a cual de los miembros del lado derecho de la producción se refiere?

a)la 'a' b)la 1er S (D)

c)la segunda S d)la 'b'

Si el codigo de yylex es el siguiente

int yylex() { return getchar(); }

de cuantos caracteres son los tokens

a) 0 b) 1 c) 2 d) la cantidad de caracteres del token varia (B)

Considere la siguiente gramática (los terminales se indican en negritas)

L-> L**,** D | D

D-> **0** | **1**

Escriba la sección de reglas de la especificación de yacc para dicha gramática

```
%%
L:      L ',' D
        | D
        ;
D:      0
        | 1
        ;
%%
```

Escriba la especificación de yacc para la gramática

S → U | V
U → T**a**U | T**a**T
V → T**b**V | T**b**T
T → **a**T**b**T | **b**T**a**T | **ε**

```
%%
S:      U
        | V
        ;
U:      T ‘a’ U
        | T ‘a’ T
        ;
V:      T ‘b’ V
        | T ‘b’ T
        ;
T       /* ε*/
        | ‘a’ T ‘b’ T
        | ‘b’ T ‘a’ T
        ;
%%
```

Escriba las acciones gramaticales para que imprima el numero de b's en la cadena de entrada

```
%{
/*escriba el tipo de los elementos en la pila de yacc */
#define YYSTYPE
%}

%%
S : ’(’ B ’)’ {$$=$2}
;
B : ’(’ B ’)’ {$$=$2 }
  | D { $$=$1; }
;
D : { }
  | ’b’ D { $$.numb++;$$=$2}
;
%%
```

Considere la siguiente gramática (los terminales se indican en negritas)
lista->lista **,** figura | figura
figura-> triangulo | cuadrilatero
triangulo-> **lado lado lado**
cuadrilatero-> **lado lado lado lado**
Escriba la sección de reglas de la especificación de yacc para dicha gramática y las acciones semánticas respectivas para que se imprima si un triangulo es equilátero y si un cuadrilátero es un cuadrado

```
%%
        lista: lista ‘,’ figura
        |       figura
        ;
        figura: triangulo
        |       cuadrilátero
        ;
        triangulo: lado lado lado {if($1==$2 && $2==$3) printf(“Equilatero”);}
        ;
        cuadrilátero: lado lado lado lado {if($1 == $2 && $2 == $3 && $3 == $4)
        printf(“Cuadrialtero”);}
        ;
%%
```