1a Guía Compiladores

Nombre: Cruz López Adrián Grupo: 3CM15 Fecha: 13/09/2021

* Defina **compilador**

Programa que lee un programa escrito en un lenguaje fuente y lo traduce a un programa equivalente o en lenguaje objeto.

* ¿Cuáles son las dos partes de la compilación?

1. Análisis
2. Síntesis

* Describa las 6 fases de un compilador

Análisis lineal o léxico, análisis sintáctico, análisis semántico, generación de código intermedio, optimización de código, generación de código.

* ¿Cuáles son los 8 módulos de un compilador?

Análisis léxico, Análisis sintáctico, Análisis semántico, Generación de código intermedio, Optimización de código, Generación de código objeto, Tabla de símbolos y Manejo de errores.

* A partir de hoc4 se usan dos etapas en hoc. ¿Cuáles son y qué hacen?

1. Generación de código
2. Ejecución

Falso o verdadero (F/V)

|  |  |
| --- | --- |
| 1. A los terminales se les llama así porque no pueden ser sustituidos | (V) |
| 1. Que una secuencia de caracteres concreta sea un token depende del lenguaje | ( F ) |
| 1. Las cadenas que pertenecen al lenguaje generado por una gramática están hechas solo de terminales | ( V ) |
| 1. El **análisis léxico** lee la cadena de entrada de derecha a izquierda | ( F ) |
| 1. El **análisis léxico** construye el árbol de análisis sintáctico | ( F ) |
| 1. La secuencia de caracteres que forma un componente léxico es el **lexema** del componente | ( V ) |
| 1. La gramática S → aS |S**a** | **a** se puede analizar con un **analizador sintáctico predictivo descendente recursivo** | ( V ) |
| 1. El tipo de **yylval** no es el mismo que el de los elementos en la pila de YACC | ( F ) |
| 1. La única forma de indicar el tipo de los elementos en la pila de YACC es usando #define YYSTYPE | ( V ) |
| 1. El **código intermedio** debe ser fácil de generar | ( V ) |
| 1. Un **esquema de traducción** es una **GLC** + **reglas semánticas** | ( V ) |
| 1. Árbol de análisis sintáctico con anotaciones es sinónimo de **árbol decorado** | ( V ) |
| 1. Análisis sintáctico **descendente** es donde la construcción del árbol de análisis sintáctico se inicia en las hojas y avanza hacia la raíz | ( F ) |
| 1. Análisis sintáctico **ascendente** es donde la construcción del árbol de análisis sintáctico se inicia en las hojas y avanza hacia la raíz | ( V ) |
| 1. yylex () llama a yyparse () | ( F ) |
| 1. yyparse () llama a yylex () | ( V ) |
| 1. yylex () retorna el **tipo** de **token** | ( V ) |
| 1. yylval almacena el **lexema** | ( V ) |
| 1. HOC1 es una **calculadora** | ( V ) |
| 1. Las **variables** en HOC son de tipo entero | ( F ) |
| 1. La notación **posfija** es una notación matemática libre de paréntesis y en esta notación los operadores aparecen después de los operandos | ( V ) |
| 1. La **raíz** del árbol de análisis sintáctico se etiqueta con el símbolo inicial | ( V ) |
| 1. Las **hojas** del árbol de análisis sintáctico se etiquetan con no terminales | ( V ) |
| 1. En la notación **infija** la **asociatividad** y la **precedencia** se usan para determinar en qué orden hay que realizar las operaciones para **evaluar** una expresión | ( V ) |

¿Para qué 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 semántico | b) analizador sintáctico | c) optimizador | d) generador de código | ( 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 |  |

¿Cuál de las sigs. opciones no es sinónimo de las otras?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a) Componente léxico | b) no terminal | c) token | d) Símbolo gramatical | ( A ) |

Es una gramática donde en el lenguaje que genera existe una cadena que tiene más 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 *→* A entonces es una

|  |  |  |
| --- | --- | --- |
| *a)* Gramática recursiva por la izquierda | *b) Gramática ambigua* | ( C ) |
| c) Gramática libre de contexto | d) ninguna de las anteriores |  |

Considere la siguiente gramática

*S→ 0 S* ***1*** *|* ***01***

1. Mostrar una derivación de ***00001111***

*S* *0 S 1*

*0 0 S 1 1*

*0 0 0 S 1 1 1*

*0 0 0 0 1 1 1 1*

1. Dibuje el árbol de análisis sintáctico para la entrada ***00001111***

Diagrama

Descripción generada automáticamente

Considere la siguiente gramática

S **b**A

A **b**B

B **b**C

C **ε**

1. Mostrar una derivación de **bbb**

**S b A**

**b b B**

**b b b C**

**b b b**

1. Dibuje el árbol de análisis sintáctico para la entrada **bbb**

Un reloj de aguja

Descripción generada automáticamente con confianza media

Considere la siguiente gramática

*S*  *A*

*A*  *A***+***A* | *B****++***

*B*  ***y***

1. Mostrar una derivación de **y + + + y + +**

S A

A + A

B + + + B + +

y + + + y + +

1. Dibuje el árbol de análisis sintáctico para la entrada **y + + + y + +**

Forma

Descripción generada automáticamente

Considere la siguiente gramática

*l l, d | d*

*d* ***0*** *|* ***1*** *|* ***2*** *|* ***3*** *|* ***4*** *|* ***5*** *|* ***6*** *|* ***7*** *|****8*** *|* ***9***

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

l l , d l

l , d , d l

l , d , d , d l

l , d , d , d , d l

l , d , d , d , d , d l

l , d , d , d , d , d , d l

l , d , d , d , d , d , d , d l

l , d , d , d , d , d , d , d , d l

l , d , d , d , d , d , d , d , d , d l

9, 8, 7, 6, 5, 4, 3, 2, 1, 0

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

Diagrama

Descripción generada automáticamente

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**

1. Obtenga una derivación de dicha cadena

E T

T/F

F/b

(E)/b

(E + T)/b

(F + F)/b

(a + b)/b

1. Dibuje el árbol de análisis sintáctico que corresponde a la cadena mencionada

Imagen en blanco y negro

Descripción generada automáticamente con confianza media

**Análisis sintáctico predictivo descendente recursivo**

Considere la siguiente gramática

S **a** | **(**S)

Escriba el analizador sintáctico predictivo descendente recursivo

void parea(complex +){

if(preanalisis == +)

preanalisis == sigcomplex();

else error();

}

void S() {

if( preanalisis == ‘(’){

parea (‘(’);

S ( );

parea ( ‘)’);

}

else if (preanalisis==’a’)

parea(‘a’);

else

error();

}

**Ambigüedad**

Demostrar que la siguiente gramática es ambigua

**Un dibujo de una persona

Descripción generada automáticamente con confianza baja** S aS |S**a** | **a**

usando la cadena **aa**

S aS | Sa | a

aS

aa

S Sa

aa

Demostrar que la siguiente gramática es ambigua

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

B **x** B | **x**

usando la cadena **xxxxx**

A A x B | x

B x B | x

A A x B

A x x B

A x x x B

A x x x x

x x x x x

A A x B

A x B x B

x x x x x

Imagen en blanco y negro

Descripción generada automáticamente con confianza media

Demostrar que la siguiente gramática es ambigua

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

usando la cadena abab

S a S b S | b S a S | ε

S a S b S

a b S a S b ε

a b ε a ε b

a b a b

Diagrama

Descripción generada automáticamente

S a S b S

a ε b a S b S

a b a ε b ε

a b a b

Imagen en blanco y negro

Descripción generada automáticamente con confianza baja

Verificar si las siguientes gramáticas son ambiguas

S S **+** S | S **-** S | **a**

S S S **+** | S S **-** | **a**

S → S+S

→ a+a

S → SS+

→ aS+

→ aa+

NO SON AMBIGUAS

**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

A 1 | S

S 0 | A

S ( L ) | a

*No tiene recursividad*

L L , S | S

L L , S | S

Beta = S

Alfa = , S

L SR

R , S | ε

Escriba el analizador sintáctico predictivo descendente recursivo para dichas gramáticas

void parea(complex +){

if(preanalisis == +)

preanalisis == sigcomplex();

else

error();

}

void S() {

if(parea == ‘(’ ){

parea (‘(’);

L( );

parea ( ‘)’);

}

else if(parea == ‘a’)

parea(‘a’);

else

error();

}

void L( ) {

S( );

R( );

}

void R() {

if(parea == ‘ , ’ ){

parea (‘,’);

S( );

}

else ;

}

Escriba la sección de reglas de la especificación de YACC para dichas gramáticas

%left ‘,’

%%

S: (‘ L ’)

| a ;

L: S R ;

R: ‘,’ S

| ;

%%

**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 sintácticocon anotaciones para la sig. cadena

c n e ss oo nnn eee ssss oooo

Imagen que contiene Gráfico

Descripción generada automáticamente

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;}

;

%%

**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

expr V expr termino { print (expr , ‘V’, termino}

expr ∧ expr termino { print (exp, ‘ ∧‘, termino)}

expr ~ expr termino { print (exp, ‘ ~‘, termino)}

expr termino

termino T { print (‘ T ’)}

termino F { print (‘ F’)}

**Escritura de Gramáticas**

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

T = {a,b,c}

N = {A,S}

S = {S}

P = { S AAAA

A a|b|c}

|  |  |  |  |
| --- | --- | --- | --- |
| **Escribir una gramática que sirva para generar las siguientes cadenas** | | | |
| Especie perro  Edad 1  Sexo macho  Tamaño grande  Colores negros, blanco  Soy rápido, activo, alegre  Aficiones correr, comer | Especie gato  Edad 2  Sexo macho  Tamaño mediano  Colores negro, blanco, café  Soy tranquilo, sociable  Aficiones dormir, parrandear, comer | Especie perro  Edad 2  Sexo hembra  Tamaño pequeño  Colores canela, gris  Soy fuerte, alegre, activo.  Aficiones aullar | Especie gato  Edad 2  Sexo macho  Tamaño grande  Colores blancos  Soy listo, obediente  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  Nivel Junior  Sexo Hombre  Lenguajes Java, C, Logo  Aficiones programar, videogames, comics, hackear, googlear | Etiquetado Geek  Nivel Senior  Sexo Mujer  Lenguajes Pascal, Prolog, SQL  Aficiones chatear, videogames, programar | Etiquetado Nerd  Nivel Junior  Sexo Mujer  Lenguajes PHP, Perl, Java  Aficiones hackear, googlear, gotcha, dormir | Etiquetado Freak  Nivel Senior  Sexo Hombre  Lenguajes Ensamblador, C  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 qué sirve $$

Para indicar un símbolo no terminal

Dentro de una acción gramatical $n se refiere a la posición del token

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 |

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

|  |  |  |
| --- | --- | --- |
| a) retornando cero | b) retornando -1 | ( D ) |
| c) almacenando -1 en yylval | d) almacenando 0 en yylval |

Una acción gramatical debe ir entre

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

Considere la producción

S: S ' a' S ‘b'

$4 a cuál 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 código 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 /\*nada\*/

| ‘a’ T ‘b’ T

| ‘b’ T ‘a’ T

;

%%

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

%{

#define YYSTYPE

%}

%%

S:’ (’ B’)’ { }

;

B:’ (’ B’)’ { }

| D {$$=$1;}

;

D: { }

| ’b’ D { }

;

%%

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

listalista, **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 triángulo 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”);}

;

%%