**COMPILADORES 2017-II UNIVERSIDAD TECNOLÓGICA DE PEREIRA**

**SEGUNDA ENTREGA “ANALIZADOR SINTÁCTICO”**

**Christian Herrera Rodríguez 9860957**

**DOCUMENTACIÓN**

**1. Introducción**

Se implementó un analizador sintáctico usando la librería ply.yacc en python.

El analizador sintáctico está diseñado para detectar las reglas gramaticales de un lenguaje "MiniJava" que es una versión Java simplificada para los propósitos de la materia, y cuyas especificaciones se encuentran en el documento **"Descripción proyecto compiladores"** enviado por el profesor.

El analizador sintáctico usa el analizador léxico que se desarrollo en la primera entrega para identificar los tokens.

**2. Arbol sintáctico**

Para construir el arbol sintáctico se utilizó una sola clase de nodo. Aprovechando la versatilidad que ofrece Python se definió una clase nodo que solo tiene dos atributos:

-Una lista de hijos: Un nodo puede o no tener hijos. Y esos hijos pueden ser de distintos tipos de datos. Esto lo podemos representar fielmente usando una lista.

-Tipo: es una cadena con la descripción del nodo, la cual se usa para imprimir el tipo de nodo o regla dentro del arból sintáctico

**2. Reglas gramaticales**

Luego de convertir las reglas en su forma EBN (Extended Backus Naur) a BN (Backus Naur) y usando el método LALR para el análisis, resultó solo 1 conflicto desplaza/reduce y 94 reglas gramáticales que se listan a continuación:

Rule 0 S' -> PROGRAMA

Rule 1 PROGRAMA -> CLASS\_DECL\_LIST

Rule 2 PROGRAMA -> epsilon

Rule 3 CLASS\_DECL\_LIST -> CLASS\_DECL

Rule 4 CLASS\_DECL\_LIST -> CLASS\_DECL CLASS\_DECL\_LIST

Rule 5 CLASS\_DECL -> class id extends id { FIELD\_OR\_METHOD\_DECL\_LIST }

Rule 6 CLASS\_DECL -> class id extends id { }

Rule 7 CLASS\_DECL -> class id { FIELD\_OR\_METHOD\_DECL\_LIST }

Rule 8 CLASS\_DECL -> class id { }

Rule 9 FIELD\_OR\_METHOD\_DECL\_LIST -> FIELD\_DECL

Rule 10 FIELD\_OR\_METHOD\_DECL\_LIST -> FIELD\_DECL FIELD\_OR\_METHOD\_DECL\_LIST

Rule 11 FIELD\_OR\_METHOD\_DECL\_LIST -> METHOD\_DECL

Rule 12 FIELD\_OR\_METHOD\_DECL\_LIST -> METHOD\_DECL FIELD\_OR\_METHOD\_DECL\_LIST

Rule 13 FIELD\_DECL -> TYPE id ;

Rule 14 FIELD\_DECL -> TYPE id LIST\_AUX\_IDS ;

Rule 15 LIST\_AUX\_IDS -> , id

Rule 16 LIST\_AUX\_IDS -> , id LIST\_AUX\_IDS

Rule 17 METHOD\_DECL -> TYPE id ( ) BLOCK

Rule 18 METHOD\_DECL -> TYPE id ( FORMALS ) BLOCK

Rule 19 METHOD\_DECL -> void id ( ) BLOCK

Rule 20 METHOD\_DECL -> void id ( FORMALS ) BLOCK

Rule 21 FORMALS -> TYPE id

Rule 22 FORMALS -> TYPE id , FORMALS

Rule 23 TYPE -> int

Rule 24 TYPE -> boolean

Rule 25 TYPE -> string

Rule 26 TYPE -> id

Rule 27 TYPE -> TYPE [ ]

Rule 28 BLOCK -> { }

Rule 29 BLOCK -> { VAR\_DECL\_STATEMENTS\_LIST }

Rule 30 VAR\_DECL\_STATEMENTS\_LIST -> VAR\_DECL

Rule 31 VAR\_DECL\_STATEMENTS\_LIST -> VAR\_DECL VAR\_DECL\_STATEMENTS\_LIST

Rule 32 VAR\_DECL\_STATEMENTS\_LIST -> STATEMENT

Rule 33 VAR\_DECL\_STATEMENTS\_LIST -> STATEMENT VAR\_DECL\_STATEMENTS\_LIST

Rule 34 VAR\_DECL -> TYPE id LIST\_IDS\_EXPRESSIONS ;

Rule 35 VAR\_DECL -> TYPE id ;

Rule 36 VAR\_DECL -> TYPE id = EXPRESSION LIST\_IDS\_EXPRESSIONS ;

Rule 37 VAR\_DECL -> TYPE id = EXPRESSION ;

Rule 38 LIST\_IDS\_EXPRESSIONS -> , id

Rule 39 LIST\_IDS\_EXPRESSIONS -> , id = EXPRESSION

Rule 40 LIST\_IDS\_EXPRESSIONS -> , id LIST\_IDS\_EXPRESSIONS

Rule 41 LIST\_IDS\_EXPRESSIONS -> , id = EXPRESSION LIST\_IDS\_EXPRESSIONS

Rule 42 STATEMENT -> ASSIGN ;

Rule 43 STATEMENT -> CALL ;

Rule 44 STATEMENT -> return EXPRESSION ;

Rule 45 STATEMENT -> return ;

Rule 46 STATEMENT -> if ( EXPRESSION ) STATEMENT else STATEMENT

Rule 47 STATEMENT -> if ( EXPRESSION ) STATEMENT

Rule 48 STATEMENT -> while ( EXPRESSION ) STATEMENT

Rule 49 STATEMENT -> break ;

Rule 50 STATEMENT -> continue ;

Rule 51 STATEMENT -> BLOCK

Rule 52 ASSIGN -> LOCATION = EXPRESSION

Rule 53 LOCATION -> id

Rule 54 LOCATION -> EXPRESSION . id

Rule 55 LOCATION -> id [ EXPRESSION ]

Rule 56 CALL -> METHOD ( ACTUALS )

Rule 57 CALL -> METHOD ( )

Rule 58 METHOD -> id

Rule 59 METHOD -> EXPRESSION . id

Rule 60 ACTUALS -> EXPRESSION

Rule 61 ACTUALS -> EXPRESSION , ACTUALS

Rule 62 EXPRESSION -> LOCATION

Rule 63 EXPRESSION -> CALL

Rule 64 EXPRESSION -> this

Rule 65 EXPRESSION -> new id ( )

Rule 66 EXPRESSION -> new TYPE [ EXPRESSION ]

Rule 67 EXPRESSION -> EXPRESSION . length

Rule 68 EXPRESSION -> BINARY\_EXPRESSION

Rule 69 EXPRESSION -> ! EXPRESSION

Rule 70 EXPRESSION -> - EXPRESSION

Rule 71 EXPRESSION -> LITERAL

Rule 72 EXPRESSION -> ( EXPRESSION )

Rule 73 BINARY\_EXPRESSION -> EXPRESSION + EXPRESSION

Rule 74 BINARY\_EXPRESSION -> EXPRESSION - EXPRESSION

Rule 75 BINARY\_EXPRESSION -> EXPRESSION \* EXPRESSION

Rule 76 BINARY\_EXPRESSION -> EXPRESSION / EXPRESSION

Rule 77 BINARY\_EXPRESSION -> EXPRESSION % EXPRESSION

Rule 78 BINARY\_EXPRESSION -> EXPRESSION y EXPRESSION

Rule 79 BINARY\_EXPRESSION -> EXPRESSION o EXPRESSION

Rule 80 BINARY\_EXPRESSION -> EXPRESSION < EXPRESSION

Rule 81 BINARY\_EXPRESSION -> EXPRESSION menorIgual EXPRESSION

Rule 82 BINARY\_EXPRESSION -> EXPRESSION > EXPRESSION

Rule 83 BINARY\_EXPRESSION -> EXPRESSION mayorIgual EXPRESSION

Rule 84 BINARY\_EXPRESSION -> EXPRESSION comparador EXPRESSION

Rule 85 BINARY\_EXPRESSION -> EXPRESSION noIgual EXPRESSION

Rule 86 LITERAL -> numero

Rule 87 LITERAL -> binario

Rule 88 LITERAL -> hexa

Rule 89 LITERAL -> cientifico

Rule 90 LITERAL -> cadenaCaracteres

Rule 91 LITERAL -> true

Rule 92 LITERAL -> false

Rule 93 LITERAL -> null

Rule 94 epsilon -> <empty>

**3. Bugs**

- Si el token número tiene un número negativo, entonces el analizador puede arrojar un error porque no es capaz de distinguir el número negativo -1 de la expresión '-' 1.

- Si la regla gramátical LOCATION = EXPRESSION '[' EXPRESSION '] se deja así, no reconoce un statement que sea un índice de un arreglo. Ejemplo

//no reconoce estas expresiones

a[i] = a[j];

a[j] = tmp;

Para solucionarlo, tocó modificar la gramática y definir explicitamente que LOCATION = id '[' EXPRESSION ']' y ahí sí funciona perfectamente. Pero entonces toca hacer esto para cada una de las posibles derivaciones que tome la producción EXPRESSION

**4. Observaciones**

La parte más díficil del trabajo fue definitivamente transformar la gramática de la forma EBN a BN porque reglas que sobre el papel deberían funcionar y que tienen lógica, el programa no las reconocía en algunas ocasiones. Esto puede ser debido al manejo de conflictos desplaza/reduce, precedencias, o detalles internos del módulo PLY.yacc. Pero fue definitivamente díficil porque se intentaron varias transformaciones que deberían funcionar en teoría pero luego en el programa no funcionaban, y el ejercicio se convirtió al final en una serie de actos de prueba y error donde no se actuaba siguiendo una lógica, sino se experimentaba modificando algunas reglas hasta obtener el resultado deseado.