

# OLyC2

## Gramática

Andrés Ricardo Ismael Guzmán - 201010425

---

Gramáticas	1
<b>J#Jison</b>	<b>2</b>
Expresiones Regulares.	2
Precedencia utilizada.	3
Cantidad de símbolos terminales.	3
Enumeración de los símbolos terminales.	3
Cantidad de símbolos no terminales.	4
Explicación de cada uno de los símbolos no terminales (cuál fue su uso dentro de la gramática)	4
Gramática funcional describiendo cada una de las acciones	7

---

# Gramáticas

## J#jison

### Expresiones Regulares.

IDENTIFIER	<code>[_a-zA-Z][a-zA-Z0-9_]*</code>
INTEGER LITERAL	<b>Macros Utilizadas</b> <code>D</code> <code>[0-9]</code> <code>NZ</code> <code>[1-9]</code> <code>Ds</code> <code>("0"   {NZ}{D}*)</code> <code>{DS}</code>
STRING LITERAL	<b>Macros Utilizadas</b> <code>escapechar</code> <code>[\\"\\bfnrtv]</code> <code>escape</code> <code>\\{escapechar}</code> <code>acceptedcharsdouble</code> <code>[^\"\\]+</code> <code>stringdouble</code> <code>{escape} {acceptedcharsdouble}</code> <code>stringliteral</code> <code>(\"{stringdouble}*\" )</code>
DOUBLE LITERAL	<b>Macros Utilizadas</b> <code>D</code> <code>[0-9]</code> <code>NZ</code> <code>[1-9]</code> <code>Ds</code> <code>("0"   {NZ}{D}*)</code> <code>{Ds}"."{Ds}+</code>
CHARACTER LITERAL	<b>Macros Utilizadas</b> <code>escapechar</code> <code>[\\"\\bfnrtv]</code> <code>escape</code> <code>\\{escapechar}</code> <code>acceptedcharsdouble</code> <code>[^\"\\]+</code> <code>stringdouble</code> <code>{escape} {acceptedcharsdouble}</code> <code>charliteral</code> <code>(\'{stringdouble}\')</code>
BOOLEAN LITERAL	<code>"true"   "false"</code>

NULL LITERAL	"null"
SEMICOLON, COMMA, LBRACE, RBRACE, LPAREN, RPAREN, COLON, LBRACK, RBRACK, ARROW, QUESTION, PEQ, DOLLAR, DOT	"," "," "{" "}" "(" ")" ":" "[" "]" "?" "!=" "\$" "."
EQ	"=="
PLUSPLUS, MINUSMINUS	"++" "--"
CONST, VAR, GLOBAL	"const" "var" "global"
PLUS, MINUS, NOT, DIV, MOD, POW, MULT	"+" "-" "!" "/" "%" "^" "*"
LT, LTEQ, GT, GTEQ, AND, OR, XOR, EQEQEQ	"<" "<=" ">" ">=" "&&" " " "^^" "==="
IF, ELSE, SWITCH, CASE, DEFAULT, FOR, BREAK, CONTINUE, WHILE, DO, THROW, TRY, CATCH, IMPORT, DEFINE, AS, STRC, RETURN	"if" "else" "switch" "case" "default" "for" "break" "continue" "while" "do" "throw" "try" "catch" "import" "define" "as" "strc" "return"
BOOLEAN, INTEGER, DOUBLE, CHAR, VOID	"boolean" "integer" "double" "char" "void"
Comentarios	<pre> "/" /" .* "/ * " &lt;comment&gt; "*" /" &lt;comment&gt; . \s+</pre>

## Precedencia utilizada.

**%left** PEQ, EQ

**%left** ELSE

**%left** LBRACK

**%left** DOT

---

## Cantidad de símbolos terminales.

66

## Enumeración de los símbolos terminales.

1. PEQ	18. IF	35. RETURN	53. PLUSPLUS
2. LBRACE	19. ELSE	36. BOOLEAN	54. PLUS
3. RBRACE	20. WHILE	37. INTEGER	55. MINUSMIN
4. LPAREN	21. DO	38. DOUBLE	US
5. RPAREN	22. FOR	39. CHAR	56. MINUS
6. LBRACK	23. BREAK	40. LTEQ	57. MULT
7. RBRACK	24. CONTINUE	41. LT	58. DIV
8. COMMA	25. SWITCH	42. REEQ	59. MOD
9. QUESTION	26. CASE	43. EQEQ	60. DOT
10. COLON	27. DEFAULT	44. GTEQ	61. NULL_LITERAL
11. SEMICOLON	28. TRY	45. GT	AL
12. DOLLAR	29. CATCH	46. NOTEQ	62. IDENTIFIER
13. FINAL	30. THROW	47. OROR	63. DOUBLE_LITERAL
14. VAR	31. BOOLEAN_LITERAL	48. POT	64. INTEGER_LITERAL
15. GLOBAL	32. DEFINE	49. XOR	TERAL
16. VOID	33. AS	50. ANDAND	65. STRING_LITERAL
17. IMPORT	34. NEW	51. NOT	ERAL
		52. EQ	66. CHARACTER_LITERAL

---

## Cantidad de símbolos no terminales.

92

## Explicación de cada uno de los símbolos no terminales (cuál fue su uso dentro de la gramática)

1. translation_unit	1. Este es el root, es una lista de posibles opciones
2. literal	2. producción para todas las hojas de las expresiones
3. primitive_type	3. Tipos primitivos
4. numeric_type	4. Producción para tipo numérico
5. integral_type	5. Tipo entero
6. floating_point_type	6. Tipo double
7. array_type	7. Nombres y primitivos en la forma de arreglo
8. name	8. Producción para simple name y qualified name
9. simple_name	9. IDENTIFICADOR
10. qualified_name	10. Producción para nombres del tipo A.B
11. import_declarations	11. Declaración para import
12. type_declarations	12. Lista para los tipos
13. type_declaration	13. Declaración de dos posibilidades
14. struct_declaration	14. Declaración de struct
15. struct_body	15. Cuerpo del struct
16. struct_variable_declaration	16. Definición de la forma en que vienen las variables
17. class_member_declaration	17. Declaración de los miembros que pueden venir: definición de variable y función
18. modifiers	18. const, var y global
19. field_declaration	19. Declaración de variable
20. variable_declarators	20. Declaración de variable sin punto y coma
21. variable_declarator	21. Id de la variable
22. variable_declarator_id	22. Id de la variable //hotfix para evitar conflictos
23. method_declaration	23. method_header y method_body
24. method_header	24. tipo y method_declarator
25. method_declarator	25. Id y lista de argumentos
26. formal_parameter_list	26. Lista de parámetros formales

27. formal_parameter	27. parámetro formal (tipo id)
28. method_body	28. body de la función
29. array_initializer	29. dos tipos de inicialización de arreglo
30. variable_initializers	30. Lista de tipos de inicialización de variable
31. variable_initializer	31. Inicialización de variable
32. block	32. lista de sentencias delimitada por { }
33. block_statements	33. lista de sentencias de un bloque
34. block_statement	34. switch con todas las sentencias posibles
35. variable_declaration_statement	35. declaración de variable con ;
36. variable_declaration	36. declaración de variable sin ;
37. special_declaration	37. tipo de declaración con :=
38. statement	38. Todas las posibles sentencias dentro de un método
39. statement_without_trailing_substatement	39. hotfix para evitar problemas de if colgante (no funcionó)
40. if_then_statement	40. if sin sentencia else
41. if_then_else_statement	41. if con sentencia else
42. while_statement	42. sentencia while
43. for_statement	43. sentencia for
44. for_init	44. primer hijo del for
45. for_update	45. tercer hijo del for
46. throw_statement	46. sentencia throw
47. try_statement	47. sentencia try catch
48. catches	48. lista de catches
49. catch_clause	49. sentencia catch como tal
50. statement_expression	50. expresión con delimitador ;
51. empty_statement	51. ; como tal
52. expression_statement	52. expresión
53. switch_statement	53. sentencia switch
54. switch_block_statement_groups	54. grupo de sentencias de switch
55. switch_labels	55. etiquetas case default de switch
56. do_statement	56. sentencia do while
57. expression_list	57. lista de expresiones con comas intercaladas
58. break_statement	58. sentencia break
59. continue_statement	59. sentencia continue
60. return_statement	60. sentencia return
61. primary	61. expresión que agrupa ciertas no terminales
62. primary_no_new_array	62. expresión con el grupo de no terminales faltantes
63. class_instance_creation_expression	63. setencia str IDENTIFIER()
64. argument_list	64. lista de argumentos para función
65. array_creation_expression	65. creación de arreglo
66. dim_exprs	66. lista dimensiones con expresiones

67. dim_expr 68. dims 69. field_access  70. method_invocation 71. array_access 72. postfix_expression 73. postincrement_expression 74. unary_expression 75. predecrement_expression 76. cast_expression 77. unary_expression_not_plus_minus 78. power_expression 79. preincrement_expression 80. multiplicative_expression 81. additive_expression 82. relational_expression  83. equality_expression 84. exclusive_or_expression 85. conditional_and_expression 86. conditional_or_expression 87. assignment_expression 88. assignment 89. left_hand_side 90. assignment_operator 91. expression 92. constant_expression	adentro 67. dimensión con expresión 68. lista de dimensión sin expresion [] 69. acceso a un objeto 70. llamada a función 71. acceso a un arreglo 72. expresión postfija 73. expresión postfija ++ 74. expresiones unarias 75. expresión prefija -- 76. casteos 77. expresión unaria sin menos o mas 78. potencia 79. expresión preija ++ 80. producción para producto, división y módulo 81. producción para suma, resta 82. producción con todos los relacionales >,>=,<=,< 83. Producción para igualdad ==, != 84. Producción para xor 85. Producción para and 86. Producción para or 87. asignación 88. asignación 89. lado izquierdo de la asignación 90. operador = 91. expresión 92. expresión nuevamente
--	---

---

## Gramática funcional describiendo cada una de las acciones

Básicamente las acciones tomadas fueron para la generación de un AST. Haciendo uso de las clases de javascript, en cada terminal se fue creando un nodo y según fuera el caso este nodo se sintetiza o se agrupa con otros para dar la estructura del árbol deseada.

### Estructura del lenguaje

```
translation_unit :  
  import_declarations SEMICOLON type_declarations EOF {  
    $$ = new AST("PROGRAM", null, @1.first_line, @1.first_column, $1, ...$3);  
    JSharpRoot = $$;  
  }  
  | type_declarations EOF {  
    $$ = new AST("PROGRAM", null, @1.first_line, @1.first_column, ...$1);  
    JSharpRoot = $$;  
  }  
  ;
```

En la primera producción se referencia al primer nodo y se obtiene la raíz del árbol.

Se toma el caso que sólo pueda venir una declaración ya que eso mencionaba el enunciado. Que pudiese venir import únicamente o la lista de declaraciones.

Esta producción contiene las 2 posibles declaraciones para una función. Como los parámetros pueden ser opcionales, se comprueba si los nodos traen algo o no. Se toma de referencia el terminal y se le cambia el tipo para construir el AST



## Tipos de Literales

```
141 // 19.3) Lexical Structure.
142 literal :
143     INTEGER_LITERAL {
144         $$ = new AST("INTEGER_LITERAL", parseInt($1), @1.first_line, @1.first_column);
145     }
146     | DOUBLE_LITERAL {
147         $$ = new AST("DOUBLE_LITERAL", parseFloat($1), @1.first_line, @1.first_column);
148     }
149     | BOOLEAN_LITERAL {
150         $$ = new AST("BOOLEAN_LITERAL", $1 == 'true', @1.first_line, @1.first_column);
151     }
152     | CHARACTER_LITERAL {
153         $$ = new AST("CHARACTER_LITERAL", $1, @1.first_line, @1.first_column);
154     }
155     | STRING_LITERAL {
156         $$ = new AST("STRING_LITERAL", $1.substring(1,$1.length-1), @1.first_line, @1.first_column);
157     }
158     | NULL_LITERAL {
159         $$ = new AST("NULL_LITERAL", null, @1.first_line, @1.first_column);
160     }
161 ;
```

Se crean nodos de los tipos de datos del lenguaje, estos se crean a partir de las reglas y expresiones regulares definidas anteriormente. Son un

## Sentencias

```
615 /* statements */
616 statement :
617     statement_without_trailing_substatement
618     $$ = $1;
619     |
620     | if_then_statement {
621         $$ = $1;
622     }
623     | if_then_else_statement {
624         $$ = $1;
625     }
626     | while_statement {
627         $$ = $1;
628     }
629     | for_statement {
630         $$ = $1;
631     }
632     ;
```

statement sirve para agrupar para la recursividad de la lista como tal. Así que lo único que se hace es reducir el nodo y agregarlo a la lista o al nodo necesario

---

## Bloques de Instrucciones

```
538 // 19.11) Blocks and Statements
539
540 block : LBRACE block_statements RBRACE {
541     $$ = $2;
542     $$.$changeType("BLOCK");
543 }
544 | LBRACE RBRACE {
545     $$ = new AST("BLOCK",null,@1.first_line, @1.first_column);
546 }
547 ;
548
549 block_statements :
550     block_statement {
551         $$ = new AST("STMT_LIST",null,@1.first_line, @1.first_column,$1) ;
552     }
553     | block_statements block_statement {
554         $$ = $1;
555         $$.$addChild($2);
556     }
557 ;
558
559 block_statement :
560     variable_declaration_statement {
561         $$ = $1;
562     }
563     | statement {
564         $$ = $1;
565     }
566 ;
567
```

block\_statements es simplemente una lista de sentencias, al igual que las otras sentencias, en el inicio de la recursividad se crea el contenedor que agrupará el resto de sentencias, solo se van agregando los bloques a la lista de hijos.

---

## Instrucciones Especiales

```
633 |
634 | statement_without_trailing_substatement :
635 |     block {
636 |         $$ = $1;
637 |     }
638 |     | empty_statement {
639 |         $$ = $1;
640 |     }
641 |     | expression_statement {
642 |         $$ = $1;
643 |     }
644 |     | switch_statement {
645 |         $$ = $1;
646 |     }
647 |     | do_statement {
648 |         $$ = $1;
649 |     }
650 |     | break_statement {
651 |         $$ = $1;
652 |     }
653 |     | continue_statement {
654 |         $$ = $1;
655 |     }
656 |     | return_statement {
657 |         $$ = $1;
658 |     }
659 |     | throw_statement {
660 |         $$ = $1;
661 |     }
662 |     | try_statement {
663 |         $$ = $1;
664 |     }
665 |     ;
666 |
```

Este es el workaround para el if-else colgante. Al igual que en los anteriores sólo se sube el nodo.

```
790 |
791 | statement_expression :
792 |     expression {
793 |         $$ = $1;
794 |     }
795 |     ;
796 |
797 | empty_statement :
798 |     SEMICOLON {
799 |         $$ = new AST("NO_OP", null, @1.first_line, @1.first_column, $1);
800 |     }
801 |     ;
802 |
803 | expression_statement :
804 |     expression SEMICOLON {
805 |         $$ = new AST("EXPRESSION_STMT", null, @1.first_line, @1.first_column, $1);
806 |     }
807 |     ;
808 |
809 |
810 |
```

De las producciones vistas en la foto anterior se toman sólo los nodos necesarios. En el caso del If tomamos el nodo del IF y en la primera casilla agregamos la condición de verdadero o falso seguido de la sentencia. Este caso es para un if sin else colgante

## Sentencia IF

```
666 if_then_statement :  
667     IF LPAREN expression RPAREN statement {  
668         $$ = new AST('IF', null, @1.first_line, @1.first_column, $3, $5);  
669     }  
670 ;  
671 ;  
672 if_then_else_statement :  
673     IF LPAREN expression RPAREN statement  
674     ELSE statement {  
675         $$ = new AST('IF', null, @1.first_line, @1.first_column, $3, $5, $7);  
676     }  
677 ;  
678 ;  
679 ;
```

Aca se crea el nodo para la sentencia IF se crea un nodo principal como padre. Se añaden los hijos: hijo 1 la condición, hijo 2 la lista de sentencias, hijo 3 la lista de sentencias else. En caso de que hijo 3 sea nulo, no se traduce nada al momento de interpretar.

## Sentencia SWITCH

```
811 switch_statement :  
812     SWITCH LPAREN expression RPAREN switch_block {  
813         $$ = new AST('SWITCH', null, @1.first_line, @1.first_column, $3, $5);  
814     }  
815 ;  
816 switch_block :  
817     LBRACE switch_block_statement_groups switch_labels RBRACE {  
818         $$ = $2;  
819         $2.addChild($3);  
820     }  
821     LBRACE switch_block_statement_groups RBRACE {  
822         $$ = $2;  
823     }  
824     LBRACE switch_labels RBRACE {  
825         $$ = new AST("SWITCH_BODY", null, @1.first_line, @1.first_column, $2);  
826     }  
827     LBRACE RBRACE {  
828         $$ = new AST("SWITCH_BODY", null, @1.first_line, @1.first_column);  
829     }  
830 ;  
831 ;  
832 switch_block_statement_groups :  
833     switch_block_statement_group {  
834         $$ = new AST("SWITCH_BODY", null, @1.first_line, @1.first_column, $1);  
835     }  
836     switch_block_statement_groups switch_block_statement_group {  
837         $$ = $1;  
838         $$.addChild($2);  
839     }  
840 ;  
841 ;  
lock statement_group :  
ch_labels block_statements {  
    $$ = $1;  
    $1.addChild($2);  
}  
labels :  
ch_label {  
    $$ = new AST('CASE_LABEL_LIST', null, @1.first_line, @1.first_column, ...$1);  
}  
switch_labels switch_label {  
    $$ = $1;  
    $$.addChild(...$2);  
}  
label :  
constant_expression COLON {  
    $$ = [$2];  
}  
DEFAULT COLON {  
    $$ = [new AST("DEFAULT", null, @1.first_line, @1.first_column)]
```

Se crea un nodo para la sentencia SWITCH. Este contiene dos hijos: el primero la expresión que se evaluara y el segundo que es un switch\_block.

La producción de switch\_block agrupa los diferentes tipos de cuerpos que el switch puede contener desde una lista de expresiones hasta un cuerpo vacío.

La producción de switch\_block\_statement\_groups es una lista de expresiones junto a sus sentencias.

La producción de switch\_block\_statement\_group define la estructura de una lista de expresiones junto a sus sentencias.

La producción de switch\_labels define una lista de casos con su expresión (case 1: case 2

case 22-3: case n:).

La producción de switch\_label agrupa las opciones para el label de cada bloque, podría ser un case (case 8:) o el defecto (default:).

## Sentencia WHILE

```
679
680 while_statement :
681     WHILE LPAREN expression RPAREN statement {
682         $$ = new AST("WHILE", null, @1.first_line, @1.first_column, $3, $5);
683     }
684     ;
685
```

La sentencia while cuenta de una estructura muy simple,, la condición y luego la lista de sentencias a ejecutar si la condición fue cumplida.

## Sentencias de Escape

```
887
888 break_statement :
889     BREAK SEMICOLON {
890         $$ = new AST("BREAK", null, @1.first_line, @1.first_column);
891     }
892     ;
893
894 continue_statement :
895     CONTINUE SEMICOLON {
896         $$ = new AST("CONTINUE", null, @1.first_line, @1.first_column);
897     }
898     ;
899
900 return_statement :
901     RETURN SEMICOLON {
902         $$ = new AST("RETURN", null, @1.first_line, @1.first_column);
903     }
904     | RETURN expression SEMICOLON {
905         $$ = new AST("RETURN", null, @1.first_line, @1.first_column,$2);
906     }
907     ;
908
909
```

Sentencias de escape, sólo se sube el nodo que se desea; en el caso de return si existe una expresión asociada se añade y se sube como tal.

## Lista de parámetros

```
945 argument_list :
946     expression {
947         $$ = new AST("EXPRESSION_LIST", null, @1.first_line, @1.first_column, $1);
948     }
949     | DOLLAR IDENTIFIER {
950         $$ = new AST("EXPRESSION_LIST", null, @1.first_line, @1.first_column,
951             new AST("DOLLAR", null, @1.first_line, @1.first_column,
952                 new AST("IDENTIFIER", $2, @2.first_line, @2.first_column)
953             )
954         );
955     }
956     | argument_list COMMA expression {
957         $$ = $1;
958         $$.addChild($3);
959     }
960     | argument_list COMMA DOLLAR IDENTIFIER {
961         $$ = $1;
962         $$.addChild(new AST("DOLLAR", null, @3.first_line, @3.first_column,
963             new AST("IDENTIFIER", $4, @4.first_line, @4.first_column)
964             )
965         );
966     }
967 ;
968
```

Inicio de expresiones, `argument_list` es para la lista de parámetros que puede tener una función o epsilon en caso que no tenga ninguno.

## Arreglos

```
969 array_creation_expression :
970     NEW primitive_type dim_exprs {
971         $$ = new AST("NEW_ARRAY", null, @1.first_line, @1.first_column, $2, $3);
972     }
973     | NEW name dim_exprs {
974         $$ = new AST("NEW_ARRAY", null, @1.first_line, @1.first_column, $2, $3);
975     }
976     | array_initializer {
977         $$ = $1;
978     };
979
980 dim_exprs : dim_expr {
981     };
982     | dim_exprs dim_expr {
983         $$ = new AST("ARRAY_DIMS", null, @1.first_line, @1.first_column,
984             $1, $2
985         );
986     }
987     | dim_exprs dim_expr {
988         $$ = $1;
989         $$.addChild($2);
990     };
991
992 dim_expr : LBRACK expression RBRACK {
993     };
994     |
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3290
3291
3292
3293
3294
3295
3296
3297
3298
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3310
3311
3312
3313
3314
3315
33
```

## Invocación de Métodos

```

1014
1015 method_invocation :
1016     name LPAREN argument_list RPAREN {
1017         let type = native_functions.find(item => item == $1.getValue());
1018         $$ = new AST((type === undefined?"FUNCTION_CALL":"NATIVE_FUNCTION_CALL"),null,@1.first_line,@1.first_line,@1.first_line);
1019     }
1020     | name LPAREN RPAREN {
1021         let type = native_functions.find(it=>it==-$1.getValue());
1022         $$ = new AST(((type === undefined?"FUNCTION_CALL":"NATIVE_FUNCTION_CALL"),null,@1.first_line,@1.first_line,@1.first_line);
1023     }
1024     |
1025     primary DOT IDENTIFIER LPAREN RPAREN {
1026         $$ = new AST("FUNCTION_CALL",null,@3.first_line,@3.first_column,
1027             new AST("DOT", null, @2.first_line, @2.first_column, $1, new AST("IDENTIFIER",$3,@3.first_line,@3.first_column));
1028     }
1029     | primary DOT IDENTIFIER LPAREN argument_list RPAREN {
1030         $$ = new AST("FUNCTION_CALL",null,@3.first_line,@3.first_column,
1031             new AST("DOT", null, @2.first_line, @2.first_column,
1032                 $1,
1033                 new AST("IDENTIFIER",$3,@3.first_line,@3.first_column)
1034             ),
1035             $$
1036         );
1037     }
1038 ;

```

Esta producción encapsula todas las formas de invocación de los métodos. Lo que varía entre estas es el número de parámetros con las que cuenta el método. Como hijos tiene el nombre del método y la lista de parámetros.

## Expresiones

```

1004
1005 unary_expression :
1006     preincrement_expression {
1007         $$ = $1;
1008     }
1009     | predecrement_expression {
1010         $$ = $1;
1011     }
1012     | PLUS unary_expression {
1013         $$ = new AST("PLUS", null, @1.first_line, @1.first_column, $2);
1014     }
1015     | MINUS unary_expression {
1016         $$ = new AST("MINUS", null, @1.first_line, @1.first_column, $2);
1017     }
1018     | unary_expression_not_plus_minus {
1019         $$ = $1;
1020     };
1021
1022 postfix_expression :
1023     primary {
1024         $$ = $1;
1025     }
1026     | name {
1027         $$ = $1;
1028     }
1029     | postincrement_expression {
1030         $$ = $1;
1031     }
1032     | postdecrement_expression {
1033         $$ = $1;
1034     };

```

A partir de los literales (producciones anteriores) acá se permite la negación tanto aritmética como lógica. La división en producciones es para la precedencia.

```

1142 power_expression :
1143     unary_expression {
1144         $$ = $1;
1145     }
1146 | power_expression POT unary_expression {
1147     /* FIX THE RECURSION */
1148     $$ = new AST("POT", null, @2.first_line,@2.first_column);
1149
1150     //we check for lhs (if lhs type == ^) we must rotate
1151     if($1.getType() == tree_types.types.POW && $1.grouped){
1152         let tmp = $1.deleteAt(1)[0];
1153         $$.addChild(tmp);
1154         $$.addChild($3);
1155         $$.addChild($5);
1156         $$ = $1;
1157     }
1158     }else{
1159         $$.addChild($1,$3);
1160     }
1161 };

```

Esta es la única lista de operaciones binarias de las que tenemos que hacer hincapié, ya que la asociatividad debe ser por la derecha. por lo que deben de intercambiarse los nodos como se muestra en la producción.

A partir de acá las operaciones binarias son las mismas y sólo se dividen en distintas producciones para que se comporte como se desea. En todos los casos son listas y el escape es la salida a otra lista. Si es una operación binaria se devuelve el operador con sus 2 hijos.

```
1162 multiplicative_expression :  
1163     power_expression {  
1164         $$ = $1;  
1165     }  
1166 | multiplicative_expression MULT power_expression {  
1167     $$ = new AST("MULT", null, @2.first_line,@2.first_column, $1,$3);  
1168 }  
1169 | multiplicative_expression DIV power_expression {  
1170     $$ = new AST("DIV", null, @2.first_line,@2.first_column, $1,$3);  
1171 }  
1172 | multiplicative_expression MOD power_expression {  
1173     $$ = new AST("MOD", null, @2.first_line,@2.first_column, $1,$3);  
1174 };  
1175
```

```
1176 additive_expression :  
1177     multiplicative_expression {  
1178         $$ = $1;  
1179     }  
1180 | additive_expression PLUS multiplicative_expression {  
1181     $$ = new AST("PLUS", null, @2.first_line,@2.first_column, $1,$3);  
1182 }  
1183 | additive_expression MINUS multiplicative_expression {  
1184     $$ = new AST("MINUS", null, @2.first_line,@2.first_column, $1,$3);  
1185 };  
1186
```

```
1187 relational_expression :  
1188     additive_expression {  
1189         $$ = $1;  
1190     }  
1191 | relational_expression LT additive_expression {  
1192     $$ = new AST("LT", null, @2.first_line,@2.first_column, $1,$3);  
1193 }  
1194 | relational_expression GT additive_expression {  
1195     $$ = new AST("GT", null, @2.first_line,@2.first_column, $1,$3);  
1196 }  
1197 | relational_expression LTEQ additive_expression {  
1198     $$ = new AST("LTEQ", null, @2.first_line,@2.first_column, $1,$3);  
1199 }  
1200 | relational_expression GTEQ additive_expression {  
1201     $$ = new AST("GTEQ", null, @2.first_line,@2.first_column, $1,$3);  
1202 };  
1203
```

```
1204 equality_expression :  
1205     relational_expression {  
1206         $$ = $1;  
1207     }  
1208 | equality_expression EQEQ relational_expression {  
1209     $$ = new AST("EQEQ", null, @2.first_line,@2.first_column, $1,$3);  
1210 }  
1211 | equality_expression REEQ relational_expression {  
1212     $$ = new AST("REEQ", null, @2.first_line,@2.first_column, $1,$3);  
1213 }  
1214 | equality_expression NOTEQ relational_expression {  
1215     $$ = new AST("NOTEQ", null, @2.first_line,@2.first_column, $1,$3);  
1216 };  
1217
```

```
1227 conditional_and_expression :  
1228     exclusive_or_expression {  
1229         $$ = $1;  
1230     }  
1231 | conditional_and_expression ANDAND exclusive_or_expression {  
1232     $$ = new AST("ANDAND", null, @2.first_line,@2.first_column, $1,$3);  
1233 };  
1234  
1235 conditional_or_expression :  
1236     conditional_and_expression {  
1237         $$ = $1;  
1238     }  
1239 | conditional_or_expression OROR conditional_and_expression {  
1240     $$ = new AST("OROR", null, @2.first_line,@2.first_column, $1,$3);  
1241 };  
1242  
1243 conditional_expression :  
1244     conditional_or_expression {  
1245         $$ = $1;  
1246     }  
1247 | conditional_or_expression QUESTION expression COLON conditional_expression {  
1248     $$ = new AST("QUESTION", null, @2.first_line,@2.first_column, $1,$3,$5);  
1249 };  
1250
```

```
1243 conditional_expression :  
1244     conditional_or_expression {  
1245         $$ = $1;  
1246     }  
1247 | conditional_or_expression QUESTION expression COLON conditional_expression {  
1248     $$ = new AST("QUESTION", null, @2.first_line,@2.first_column, $1,$3,$5);  
1249 };  
1250
```



---

A este nivel (el más alto) se realiza el operador ternario. Al igual que en los anteriores se usa QUESTION de raíz e hijo uno es condición, hijo dos es primera expresión, hijo tres es segunda expresión. Semánticamente se revisa que las dos expresiones sean del mismo tipo.