

# Memoria de la Práctica de Procesadores de Lenguajes

Diego José Abengózar Vilar, Alejandro García Castellanos,  
Ignacio Javier Encinas Ramos

Grupo 82

January 18, 2020

## Índice

<b>1</b>	<b>Diseño del Analizador Semántico</b>	<b>2</b>
1.1	Esquema de Traducción . . . . .	2
1.2	Implementación del EdT . . . . .	5
1.3	Gramática y Autómata Final del A. Sintáctico . . . . .	8
1.4	Errores . . . . .	10

# 1 Diseno del Analizador Semántico

## 1.1 Esquema de Traducción

0.  $P' \rightarrow \{TSG = creaTS(); DesplG = 0; TS\_actual = TSG;$   
     $P.func = false\} P \{liberaTS(TSG)\}$
1.  $P \rightarrow D \{ P_1.func = P.func \} P_1$
2.  $P \rightarrow F \{ P_1.func = P.func \} P_1$
3.  $P \rightarrow \{S.func = false\} S \{P_1 = P.func\} P_1$
4.  $D \rightarrow var \{zona\_decl = true\} T id ; \{InsertarTipoTS(id.posi, T.tipo)$   
    if( $TS\_actual=TS$ ) then  
        InsertarDespl( $id.posi, desplG$ )  
         $desplG = desplG + T.tamano$   
    else  
        InsertarDespl( $id.posi, desplL$ )  
         $desplL = desplL + T.tamano$   
     $zona\_decl=false\}$
5.  $T \rightarrow int \{T.tipo=entero; T.tamano = 1\}$
6.  $T \rightarrow string \{T.tipo=cadena; T.tamano = 64\}$
7.  $T \rightarrow boolean \{T.tipo=logico; T.tamano = 1\}$
8.  $F \rightarrow function \{zona\_decl =true\} T_1 id ( \{TSL = creaTS();desplL=0;$   
     $TS\_actual = TSL\} A )\{InsertaTipoTSG(id.posi,A.tipo \rightarrow T.tipo)\};$   
     $zona\_decl = false\} \{ \{ C.func = true\} C \} \{if (C.tipoRet \neq T.tipo)$   
        then error(2);  
         $TS\_actual = TSG; LiberarTS(TSL)\}$
9.  $T_1 \rightarrow \lambda \{T.tipo = tipo\_vacio\}$
10.  $T_1 \rightarrow T \{T_1.tipo = T.tipo\}$
11.  $A \rightarrow T id \{InsertarTipoTS(id.posi, T.tipo); InsertarDesplTS(id.posi, desplL);$   
     $desplL = desplL + T.tamano\} K \{A.tipo = if(K.tipo = tipo\_vacio)$   
        then  $T.tipo$   
    else  
         $T.tipo \times K.tipo\}$
12.  $A \rightarrow \lambda \{A.tipo = tipo\_vacio\}$
13.  $K \rightarrow \lambda \{K.tipo = tipo\_vacio\}$
14.  $K \rightarrow , T id \{InsertarTipoTS(id.posi, T.tipo); InsertarDesplTS(id.posi, desplL);$   
     $desplL = desplL + T.tamano\} K_1 \{K.tipo = if(K_1.tipo = tipo\_vacio)$   
        then  $T.tipo$   
    else  
         $T.tipo \times K.tipo\}$
15.  $C \rightarrow D \{C_1.func = C.func\} C_1 \{C.tipoRet = C_1.tipoRet\}$
16.  $C \rightarrow \{S.func = C.func\} S \{C_1.func = C.func\} C_1 \{C.tipoRet =$   
    if( $S.tipoRet = C_1.tipoRet$ ) then  
         $S.tipoRet$   
    else if( $S.tipoRet = tipo\_vacio$ ) then  
         $C_1.tipoRet$   
    else if( $C_1.tipoRet = tipo\_vacio$ ) then  
         $S.tipoRet$   
    else  
        error(2)}
17.  $C \rightarrow \lambda \{C.tipoRet = tipo\_vacio\}$
18.  $S \rightarrow id L E ; \{S.tipo = if(BuscaTipoTS(id.posi) = E.tipo)$   
    AND ( $E.tipo \neq tipo\_error$ )  
    then  $tipo\_ok$   
    else  
        error(3)  
     $S.tipoRet = tipo\_vacio\}$

```

19. S → id (M); {S.tipo = if (BuscaTipoTS(id.posi) = M.tipo → t)
                        then tipo_ok
                        else
                            error(4)
                        S.tipoRet = tipo_vacio}
20. S → print (E); {S.tipo = if (E.tipo = entero || E.tipo = cadena)
                        then tipo_ok
                        else
                            error(4)
                        S.tipoRet = tipo_vacio}
21. S → input(id); {S.tipo =
    if (BuscaTipoTS(id.posi) = entero || BuscaTipoTS(id.posi).tipo = cadena))
        then tipo_ok
    else
        error(4)
    S.tipoRet = tipo_vacio}
22. S → if (E) {S1.func = S.func} S1 {S.tipo =
    if (E.tipo = logico) then S1.tipo
    else
        error(5)
    S.tipoRet = S1.tipoRet}
23. S → return X; {S.tipo = if (S.func) then
    if (X.tipo != tipo.error) then tipo_ok
    else
        error(6)
    else
        error(1)
    S.tipoRet = X.tipoRet}
24. L → |= {}
25. L → = {}
26. M → EQ {M.tipo =
    if (E.tipo != tipo_error
    AND Q.tipo != tipo_error)
        then if (Q.tipo == tipo_vacio)
            then E.tipo
        else
            E.tipo x Q.tipo
    else
        error(7)}
27. M → λ {M.tipo = tipo_vacio}
28. Q → λ {Q.tipo = tipo_vacio}
29. Q → ,EQ1 {Q.tipo =
    if (E.tipo != tipo_error
    AND Q.tipo != tipo_error)
        then if (Q.tipo == tipo_vacio)
            then NuevaPila(E.tipo)
            else Q.tipo.push(E.tipo)
    else
        error(7)}
30. S1 → { {S2.func = S1.func} S2}
    {G.func=S1.func} G {S1.tipo =
        if (S2.tipo != tipo_error)
        if (G.tipo != tipo_error)
            then S2.tipo
        else error(9)
        else error(8);
    S1.tipoRet = if (S2.tipoRet == G.tipoRet OR G.tipoRet == tipo_vacio
    S2.tipoRet

```

```

    else error(10)}
31. S1 → {S2.func=S1.func} S {S1.tipo=S.tipo; S1.tipoRet = S.tipoRet}
32. G → else{ {S2.func=G.func} S2}{G.tipo=S2.tipo;
               G.tipoRet = S2.tipoRet}
33. G → λ {G.tipo = tipo_vacio; G.tipoRet = tipo_vacio}
34. X → E {X.tipo = E.tipo}
35. X → λ {X.tipo = tipo_vacio}
36. E → E1 < U {E.tipo = if(E1.tipo = U.tipo = entero)
                then logico
                else
                    error(11)}
37. E → U {E.tipo = U.tipo}
38. U → U1 + R {U.tipo = if(U1.tipo == R.tipo == entero)
                then entero
                else
                    error(11)}
39. U → R {U.tipo = R.tipo}
40. R → !V {R.tipo = if(V.tipo == logico) then logico
            else error(11)}
41. R → V {R.tipo = V.tipo}
42. V → (E) {V.tipo = E.tipo}
43. V → id {V.tipo = BuscaTS(id.posi)}
44. V → id(M) {S.tipo = if(BuscaTipoTS(id.posi) == M.tipo → t)
                  then t
                  else
                      error(4)}
45. V → ent {V.tipo = entero}
46. V → cadena {V.tipo = cadena}
47. S2 → {S.func = S2.func} S {S'2.func = S2.func} S'2 {
    S2.tipo =
        if(S.tipo != tipo_error) then S2.tipo
        else
            error(12);
    S2.tipoRet = if (S.tipoRet =) tipo_vacio) then
        S'2.tipoRet
    else if (S'2.tipoRet =) tipo_vacio) then
        S.tipoRet
    else error(13) }
48. S2 → {S.func = S2.func} S {S2.tipo = S.tipo;
    S2.tipoRet = S.tipoRet}
49. P → λ {}

```

## 1.2 Implementación del EdT

0.  $P' \rightarrow MM_1 P \{ liberaTS(TSG) \}$
1.  $P \rightarrow D P_1 \{ Aux[ntope].tipoRet = Aux[tope].tipoRet \}$
2.  $P \rightarrow F P_1 \{ Aux[ntope].tipoRet = Aux[tope].tipoRet \}$
3.  $P \rightarrow S P_1 \{ Aux[ntope].tipoRet = \text{if}(Aux[tope-1].tipoRet = tipo\_vacio) \text{ then } Aux[tope].tipoRet \text{ else } error(1) \}$
4.  $D \rightarrow \text{var } MM_2 T \text{ id } MM_3 ; \{ InsertarTipoTS(Aux[tope-2].posi, Aux[tope-3].tipo) \text{ if}(TS\_actual = TSG) \text{ then } InsertarDespl(Aux[tope-1].posi, desplG) \text{ desplG} = desplG + Aux[tope-3].tamano \text{ else } InsertarDespl(Aux[tope-2].posi, desplL) \text{ desplL} = desplL + Aux[tope-3] \}$
5.  $T \rightarrow \text{int} \{ Aux[ntope].tipo = entero; Aux[ntope].tamano = 1 \}$
6.  $T \rightarrow \text{string} \{ Aux[ntope].tipo = cadena; Aux[ntope].tamano = 64 \}$
7.  $T \rightarrow \text{boolean} \{ Aux[ntope].tipo = logico; Aux[ntope].tamano = 1 \}$
8.  $F \rightarrow \text{function } MM_3 T_1 \text{ id } MM_4 (A) MM_5 \{ C \} \{ \text{if}(Aux[tope-1].tipoRet \neq Aux[tope-9].tipo) \text{ then } error(2); TS\_actual = TSG; LiberarTS(TSL) \}$
9.  $T_1 \rightarrow \lambda \{ Aux[ntope].tipo = tipo\_vacio \}$
10.  $T_1 \rightarrow T \{ Aux[ntope].tipo = Aux[tope].tipo \}$
11.  $A \rightarrow T \text{ id } MM_6 K \{ Aux[ntope].tipo = \text{if}(Aux[tope].tipo = tipo\_vacio) \text{ then } Aux[tope-3].tipo \text{ else } Aux[tope].tipo.push(Aux[tope-3].tipo) \}$
12.  $A \rightarrow \lambda \{ Aux[ntope].tipo = tipo\_vacio \}$
13.  $K \rightarrow \lambda \{ Aux[ntope].tipo = tipo\_vacio \}$
14.  $K \rightarrow , T \text{ id } MM_7 K_1 \{ Aux[ntope].tipo = \text{if}(Aux[tope].tipo = tipo\_vacio) \text{ then } NuevaPila(Aux[tope-2].tipo) \text{ else } Aux[tope].tipo.push(Aux[tope-4].tipo) \}$
15.  $C \rightarrow D C_1 \{ Aux[ntope].tipoRet = Aux[tope].tipoRet \}$
16.  $C \rightarrow S C_1 \{ Aux[ntope].tipoRet = \text{if}(Aux[tope-1].tipoRet = Aux[tope].tipoRet) \text{ then } Aux[tope-1].tipoRet \text{ else if}(Aux[tope-1].tipoRet = tipo\_vacio) \text{ then } Aux[tope-1].tipoRet \text{ else if}(Aux[tope].tipoRet = tipo\_vacio) \text{ then } Aux[tope-1].tipoRet \text{ else } error(2) \}$
17.  $C \rightarrow \lambda \{ Aux[ntope].tipoRet = tipo\_vacio \}$
18.  $S \rightarrow \text{id } L E ; \{ Aux[ntope].tipo = \text{if}(BuscaTipoTS(Aux[tope-3].posi) = (Aux[tope-1].tipo) \text{ AND } (Aux[tope-1].tipo \neq tipo\_error)) \text{ then } tipo\_ok \text{ else } error(3) \}$   
 $Aux[ntope].tipoRet = tipo\_vacio \}$
19.  $S \rightarrow \text{id } (M) ; \{ Aux[ntope].tipo = \text{if}(BuscaTipoTS(Aux[tope-4].posi) = ParFunc(Aux[tope-2].tipo, t) \text{ then } tipo\_ok \text{ else } error(4) \}$   
 $Aux[ntope].tipoRet = tipo\_vacio \}$
20.  $S \rightarrow \text{print } (E) ; \{ Aux[ntope].tipo = \text{if}(Aux[tope-2].tipo = entero \mid Aux[tope-2].tipo = cadena) \}$

```

        then tipo_ok
        else error(4)
        Aux[ntope].tipoRet = tipo_vacio}
21. S → input(id); {Aux[ntope].tipo =
    if (BuscaTipoTS(Aux[tope-2].posi = entero
        || Aux[tope-2].tipo = cadena)) then tipo_ok
    else error(4)
    Aux[ntope].tipoRet = tipo_vacio}
22. S → if(E) S1 {Aux[ntope].tipo =
    if (Aux[tope-2].tipo = logico) then Aux[tope].tipo
    else error(5)
    Aux[ntope].tipoRet = tipo_vacio}
23. S → return X ; {Aux[ntope].tipo =
    if (Aux[tope-1].tipo != tipo.error) then tipo_ok
    else error(6)}
24. L → |= {}
25. L → = {}
26. M → EQ {Aux[ntope].tipo =
    if (Aux[tope-1].tipo != tipo_error
    AND Aux[tope].tipo != tipo_error)
    then if (Aux[tope].tipo = tipo_vacio)
        then Aux[tope-1].tipo
    else
        Aux[tope].tipo.push(Aux[tope-1].tipo)
    else
        error(7)}
27. M → λ {Aux[ntope].tipo = tipo_vacio}
28. Q → λ {Aux[ntope].tipo = tipo_vacio}
29. Q → , E Q-1 {if (Aux[tope-1].tipo != tipo_error
    AND Aux[tope].tipo != tipo_error)
    then if (Aux[tope].tipo = tipo_vacio)
        then NuevaPila(Aux[tope-1].tipo)
        else Aux[tope].tipo.push(Aux[tope-1].tipo)
    else
        error(7)}
30. S1 → { S2 } G { Aux[ntope].tipo =
    if (Aux[tope-2].tipo != tipo_error)
    if (Aux[tope].tipo != tipo_error)
        then Aux[tope-2].tipo
    else error(9)
    else error(8);
    Aux[ntope].tipoRet =
    if (Aux[tope-2].tipoRet = Aux[tope].tipoRet
    OR Aux[tope].tipoRet = tipo_vacio)
        Aux[tope-2].tipoRet
    else
        error(10)}
31. S1 → S {Aux[ntope].tipo = Aux[tope].tipo;
    Aux[ntope].tipoRet = Aux[tope].tipoRet}
32. G → else {S2} {Aux[ntope].tipo = Aux[tope-1].tipo;
    Aux[ntope].tipoRet = Aux[tope-1].tipoRet}
33. G → λ {Aux[ntope].tipo = tipo_vacio;
    Aux[ntope].tipoRet = tipo_vacio}
34. X → E {Aux[ntope].tipo = Aux[tope].tipo}
35. X → λ {Aux[ntope].tipo = tipo_vacio}
36. E → E1 < U {Aux[ntope].tipo =
    if (Aux[tope-2].tipo = Aux[tope].tipo = entero)
        then logico

```

```

        else error(11)}
37. E → U {Aux[ntope].tipo = Aux[tope].tipo}
38. U → U1 + R {Aux[ntope].tipo =
        if(Aux[tope-2].tipo = Aux[tope-2].tipo = entero)
        then entero
        else error(11)}
39. U → R {Aux[ntope].tipo = Aux[tope].tipo}
40. R → !V {Aux[ntope].tipo = if(Aux[tope].tipo = logico) then
        logico
        else error(11)}
41. R → V {Aux[ntope].tipo = Aux[tope].tipo}
42. V → (E) {Aux[ntope].tipo = Aux[tope-1].tipo}
43. V → id {Aux[ntope].tipo = BuscaTipoTS(Aux[tope].posi)}
44. V → id(M) {Aux[ntope].tipo =
        if(BuscaTipoTS(Aux[tope-3].posi) = ParFunc(Aux[tope-1].tipo, t)) then t
        else error(17)}
45. V → ent {Aux[ntope].tipo = entero}
46. V → cadena {Aux[ntope].tipo = cadena}
47. S2 → S S'2 {Aux[ntope] = if(Aux[tope-1].tipo != tipo_error) then
        Aux[tope].tipo
        else
            error(12);
        Aux[ntope] = if(Aux[tope-1].tipoRet = tipo_vacio) then
            Aux[tope].tipoRet
            else if(Aux[tope].tipoRet = tipo_vacio) then
            Aux[tope-1].tipoRet
            else error(13);}
48. S2 → S {Aux[ntope].tipo = Aux[tope].tipo;
        Aux[ntope].tipoRet = Aux[tope].tipoRet}
49. P → λ {Aux[ntope].tipoRet = tipo_vacio}
50. MM1 → λ {TSG = creaTS();
        desplG = 0;
        TSA = TSG}
51. MM2 → λ {zona_decl = true}
52. MM3 → λ {zona_decl = true}
53. MM4 → λ {TSL = creaTS();
        desplL = 0;
        TSA = TSL}
54. MM5 → λ {InsertarTipoTS(Aux[tope-4].posi, Aux[tope-1].tipo) ;
        InsertarNArgsTS(Aux[tope-4].posi, Aux[tope-1].NArgs) ;
        for i in NArgs :
            InsertarTipoArgsTS(Aux[tope-4].posi, Aux[tope-1].tipoLista(i));
            InsertarEtiquetaTS(Aux[tope-4].posi, Aux[tope-4].lexema + Aux[tope-4].posi);
            InsertarTipoDevueltoTS(Aux[tope-4].posi, Aux[tope-5].tipo)}
55. MM6 → λ {InsertarTipoTS(Aux[tope].posi, Aux[tope-1].tipo) ;
        InsertarDesplTS(Aux[tope].posi, desplL);
        desplL = desplL + Aux[tope-1].tamano}
56. MM7 → λ {InsertarTipoTS(Aux[tope].posi, Aux[tope-1].tipo) ;
        InsertarDesplTS(Aux[tope].posi, desplL);
        desplL = desplL + Aux[tope-1].tamano}
57. MM8 → λ {zona_decl = false}

```

De forma que hemos tenido que transformar el EdT para que en vez de usar atributos heredados, lo cual complica bastante la implementación, hemos transformado el atributo heredado `func` al atributo sintetizado `tipoRet`.

También hemos tenido que modificar la gramática del sintáctico añadiendo los marcadores  $MM_i$  y sus correspondientes reglas lambda para poder implementar las acciones con efectos laterales.

Ej:  
 $D \rightarrow \text{var } \{zdecl:=true\} T \text{ id}; \{otras \text{ acciones}\}$   
 Lo transformamos en:  
 $D \rightarrow \text{var } MM \text{ T id}; \{otras \text{ acciones}\}$   
 $MM \rightarrow \text{lambda } \{zdecl:=true\}$

### 1.3 Gramática y Autómata Final del A. Sintáctico

*Terminales* = { ; { } *id ent cadena* ( ) + < ! = | = *var int*  
*boolean string print input , return function if else* }  
*NoTerminales* = { *P1 P D T F T1 A K C S L M Q S1 G X E U R V*  
*S2 M1 M2 M3 M4 M5 M6 M7 M8* }  
*Axioma* = *P1*  
*Producciones* = {  
 $P1 \rightarrow M1 P$   
 $P \rightarrow D P$   
 $P \rightarrow F P$   
 $P \rightarrow S P$   
 $D \rightarrow \text{var } M2 T \text{ id } M8 ;$   
 $T \rightarrow \text{int}$   
 $T \rightarrow \text{string}$   
 $T \rightarrow \text{boolean}$   
 $F \rightarrow \text{function } M3 T1 \text{ id } M4 ( A ) M5 \{ C \}$   
 $T1 \rightarrow \text{lambda}$   
 $T1 \rightarrow T$   
 $A \rightarrow T \text{ id } M6 K$   
 $A \rightarrow \text{lambda}$   
 $K \rightarrow \text{lambda}$   
 $K \rightarrow , T \text{ id } M7 K$   
 $C \rightarrow D C$   
 $C \rightarrow S C$   
 $C \rightarrow \text{lambda}$   
 $S \rightarrow \text{id } L E ;$   
 $S \rightarrow \text{id } ( M ) ;$   
 $S \rightarrow \text{print } ( E ) ;$   
 $S \rightarrow \text{input } ( \text{id} ) ;$   
 $S \rightarrow \text{if } ( E ) S1$   
 $S \rightarrow \text{return } X ;$   
 $L \rightarrow |=$   
 $L \rightarrow =$   
 $M \rightarrow E Q$   
 $M \rightarrow \text{lambda}$   
 $Q \rightarrow \text{lambda}$   
 $Q \rightarrow , E Q$



$S1 \rightarrow \{ S2 \} G$   
 $S1 \rightarrow S$   
 $G \rightarrow else \{ S2 \}$   
 $G \rightarrow lambda$   
 $X \rightarrow E$   
 $X \rightarrow lambda$   
 $E \rightarrow E < U$   
 $E \rightarrow U$   
 $U \rightarrow U + R$   
 $U \rightarrow R$   
 $R \rightarrow ! V$   
 $R \rightarrow V$   
 $V \rightarrow ( E )$   
 $V \rightarrow id$   
 $V \rightarrow id ( M )$   
 $V \rightarrow ent$   
 $V \rightarrow cadena$   
 $S2 \rightarrow S S2$   
 $S2 \rightarrow S$   
 $P \rightarrow lambda$   
 $M1 \rightarrow lambda$   
 $M2 \rightarrow lambda$   
 $M3 \rightarrow lambda$   
 $M4 \rightarrow lambda$   
 $M5 \rightarrow lambda$   
 $M6 \rightarrow lambda$   
 $M7 \rightarrow lambda$   
 $M8 \rightarrow lambda$   
 $\}$

Dando lugar a modificaciones en el Autómata (y en consecuencia en la Tabla de Decisión):

Donde :

$S_{-1} = \{ P2 \rightarrow \bullet P1, P1 \rightarrow \bullet MM1 P, P \rightarrow \bullet SP, MM1 \rightarrow \bullet \}$   
 $S_1 = \{ P1 \rightarrow MM1 P \bullet \}$  [Ahora no se ACEPTA, solo se REDUCE]  
 $S_{100} = \{ P2 \rightarrow P1 \bullet \}$  [Ahora este es el estado que ACEPTA]  
 $S_{101} = \{ D \rightarrow var \bullet MM2 T id MM8 ;, MM2 \rightarrow \bullet \}$   
 $S_{102} = \{ F \rightarrow function \bullet MM3 T1 id MM4 ( A ) MM5 \{ C \}, MM3 \rightarrow \bullet \}$   
 $S_{103} = \{ F \rightarrow function MM3 T1 id \bullet MM4 ( A ) MM5 \{ C \}, MM4 \rightarrow \bullet \}$   
 $S_{104} = \{ D \rightarrow F \rightarrow function \bullet MM3 T1 id MM4 ( A ) \bullet MM5 \{ C \}, MM5 \rightarrow \bullet \}$   
 $S_{105} = \{ A \rightarrow T id \bullet MM6 K, MM6 \rightarrow \bullet \}$   
 $S_{106} = \{ D \rightarrow , T id \bullet MM7 K, MM7 \rightarrow \bullet \}$   
 $S_{107} = \{ D \rightarrow var MM2 T id \bullet MM8 ;, MM8 \rightarrow \bullet \}$

## 1.4 Errores

Error 1: "RETURN fuera de funcion."

Error 2: "El tipo devuelto no coincide con el declarado en la funcion."

Error 3: "Asignacion incorrecta.";

Error 4: "Incoherencia entre parametros formales y actuales en la llamada a funcion.";

Error 5: "La condicion del IF no es de tipo logico.";

Error 6: "Error en la sentencia del RETURN.";

Error 7: "Error al definir los parametros de llamada de una funcion.";

Error 8: "Error en el cuerpo del IF.";

Error 9: "Error en el cuerpo del ELSE.";

Error 10: "No concuerdan los RETURN de las sentencias IF-ELSE.";

Error 11: "Tipos incompatibles entre operandos y operadores.";

Error 12: "Error en el cuerpo del IF-ELSE.";

Error 13: "Error en el RETURN en el cuerpo del IF-ELSE.";

