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

1 Diseno del Analizador Semántico

1.1 Esquema de Traducción

```
0. P' → {TSG = creaTS(); DesplG = 0; TS_actual = TSG;
    P.func = false} P {liberaTS(TSG)}
1. P → D { P1 .func = P.func} P1
2. P → F { P1 .func = P.func} P1
3. P → {S.func = false} S {P1 = P.func} P1
4. D → var {zona_decl = true} T id ;
    {InsertarTipoTS(id.posi, T.tipo)
        if(TS_actual==TSG) then
            InsertarDespl(id.posi, desplG)
            desplG = desplG + T.tamano
        else
            InsertarDespl(id.posi, desplL)
            desplL = desplL + T.tamano
        zona_decl=false
    }
5. T → int {T.tipo=entero; T.tamano = 1}
6. T → string {T.tipo=cadena; T.tamano = 64}
7. T → boolean {T.tipo=logico; T.tamano = 1}
8. F → function {zona_decl =true} T1 id({TSL = creaTS();
    desplL=0; TS_actual = TSL} A){InsertaTipoTSG}(id.posi,
    A.tipo → T.tipo)); zona_decl = false}
    { { C.func = true} C {if (C.tipoRet != T.tipo)
    then error(1); TS_actual = TSG; LiberarTS(TSL) }}
9. T1 →λ {T.tipo = tipo_vacio}
10. T1 →T{T1.tipo = T.tipo }
11. A → 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 x K.tipo
     }
12. A → λ {A.tipo = tipo_vacio}
13. K → λ {K.tipo = tipo_vacio}
14. K → , T id {InsertarTipoTS(id.posi, T.tipo);
     InsertarDesplTS(id.posi, desplL);
     desplL = desplL + T.tamano} K1
     {K.tipo = if(K1.tipo == tipo_vacio) then
                    T.tipo
                else
                    T.tipo x K.tipo
      }
15. C → D {C1.func = C.func}
         C1 {C.tipoRet = C1.tipoRet}
16. C → {S.func = C.func} S {C1.func = C.func}C1
        {C.tipoRet =
        if(S.tipoRet == C1.tipoRet) then
            S.tipoRet
        else if(S.tipoRet == tipo_vacio) then
            C1.tipoRet
        else if(C1.tipoRet == tipo_vacio) then
            S.tipoRet
```

```
        else
            error(2)
    }
17. C → λ{C.tipoRet = tipo_vacio}
18. S→ id L E ; {S.tipo =
    if(BuscaTipoTS(id.posi) == E.tipo)
    AND (E.tipo != tipo_error))then
        tipo_ok
    else
        error(3)}
19. S→ id (M); {S.tipo =
    if(BuscaTipoTS(id.posi) == M.tipo → t)
        then tipo_ok
    else
        error(4)}
20. S → print (E); {S.tipo =
    if(E.tipo == entero || E.tipo == cadena)
        then tipo_ok
    else
        error(5)
        }
21. S → input(id); {S.tipo =
    if(BuscaTipoTS(id.posi) == entero
        || BuscaTipoTS(id.posi).tipo == cadena)) then tipo_ok
    else
        error(6)
    }
22. S → if(E) {S1.func = S.func} S1 {S.tipo =
    if(E.tipo == logico) then S1.tipo
    else
        error(7)
    }
23. S → return X; {S.tipo =
    if(S.func) then
        if(X.tipo != tipo.error) then tipo_ok
        else
            error(8)
    else
        error(9)
    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(10)
   }
27. M → λ {M.tipo = tipo_vacio}
28. Q → λ {Q.tipo = tipo_vacio}
29. Q → ,EQ1 {Q.tipo=
                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(11)
                }
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(13)
                           else error(12)
     }
31. S1 → { {S2.func=s1.func} S {Aux[ntope].tipo=Aux[tope].tipo}
32. G → else{ {S2.func=G.func} S2}{G.tipo=S2.tipo}
33. G → λ {G.tipo = 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(14)
                  }
37. E → U {E.tipo = U.tipo}
38. U → U1 + R {U.tipo = if(U1.tipo = R.tipo = entero)
                               then entero
                               else
                                 error(15)
                  }
39. U→ R {U.tipo = R.tipo}
40. R → !V {R.tipo = if(V.tipo = logico) then logico
                       else error(16)
            }
41. R → V {R.tipo = V.tipo}
42. V → (E) {V.tipo = AE.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(17)
               }
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(18)
            }
48. S2 →{S.func = S2.func} S {S2.tipo = S.tipo}
49. P → λ {}
```

1.2 Implementación del EdT

```
0. P' → MM1 P {liberaTS(TSG)}
1. P → D P1 {Aux[ntope].tipoRet = Aux[tope].tipoRet}
2. P → F P1 {Aux[ntope].tipoRet = Aux[tope].tipoRet}
3. P → S P1 {Aux[ntope].tipoRet = if(Aux[tope-1].tipoRet = tipo_vacio)then
                                        Aux[tope].tipoRet
                                    else error(1)}
4. D → var MM2 T id MM8 ; {InsertarTipoTS(Aux[tope-2].posi, Aux[tope-3].tipo)
                           if(TS_actual = TSG) then
                               InsertarDespl(Aux[tope-1].posi, desplG)
                               desplG = desplG + Aux[tope-3].tamano
                           else
                               InsertarDespl(Aux[tope-2].posi, desplL)
                               desplL = desplL + Aux[tope-3]}
5. T → int {Aux[ntope].tipo=entero; Aux[ntope].tamano = 1}
6. T → string {Aux[ntope].tipo=cadena; Aux[ntope].tamano = 64}
7. T → boolean {Aux[ntope].tipo=logico; Aux[ntope].tamano = 1}
8. F → function MM3 T1 id MM4 ( A ) MM5 { C }
       {if (Aux[tope-1].tipoRet != Aux[tope-9].tipo)then error(2);
       TS_actual = TSG; LiberarTS(TSL)}
9. T1 → λ {Aux[ntope].tipo = tipo_vacio}
10. T1 → T {Aux[ntope].tipo = Aux[tope].tipo}
11. A → T id MM6 K {Aux[ntope].tipo = if(Aux[tope].tipo = tipo_vacio)
                                        then Aux[tope-3].tipo}
                                    else
                                        Aux[tope].tipo.push(Aux[tope-3].tipo)}
12. A → λ {Aux[ntope].tipo = tipo_vacio}
13. K → λ {Aux[ntope].tipo = tipo_vacio}
14. K → , T id MM7 K1 {Aux[ntope].tipo = if(Aux[tope].tipo = tipo_vacio)
                                        then NuevaPila(Aux[tope-2].tipo)
                                    else
                                        Aux[tope].tipo.push(Aux[tope-4].tipo)}
15. C → D C1 {Aux[ntope].tipoRet = Aux[tope].tipoRet}
16. C → S C1 {Aux[ntope].tipoRet=
                if(Aux[tope-1].tipoRet = Aux[tope].tipoRet) then
                    Aux[tope-1].tipoRet
                else if(Aux[tope-1].tipoRet = tipo_vacio) then
                    Aux[tope-1].tipoRet
                else if(Aux[tope].tipoRet = tipo_vacio) then
                    Aux[tope-1].tipoRet}
                else
                    error(2)}
17. C → λ {Aux[ntope].tipoRet = tipo_vacio}
18. S→ id L E ; {Aux[ntope].tipo =
     if(BuscaTipoTS(Aux[tope-3].posi)=(Aux[tope-1].tipo)
            AND (Aux[tope-1].tipo != tipo_error))then
        tipo_ok
     else
        error(3)
     Aux[ntope].tipoRet = tipo_vacio}
19. S→ id (M) ; {Aux[ntope].tipo =
     if(BuscaTipoTS(Aux[tope-4].posi) = ParFunc(Aux[tope-2].tipo, t)
        then tipo_ok
     else error(4)
     Aux[ntope].tipoRet = tipo_vacio}
20. S → print (E) ; {Aux[ntope].tipo =
     if(Aux[tope-2].tipo = entero || 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();
            deplG = 0;
            TSA = TSG}
51. MM2 →λ {zona_decl = true}
52. MM3 →λ {zona_decl = true}
53. MM4 →λ {TSL = creaTS();
            deplL = 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, deplL);
     deplL = deplL + Aux[tope-1].tamano}
56. MM7 →λ {InsertarTipoTS(Aux[tope].posi, Aux[tope-1].tipo) ;
     InsertarDesplTS(Aux[tope].posi, deplL);
     deplL = deplL + 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.

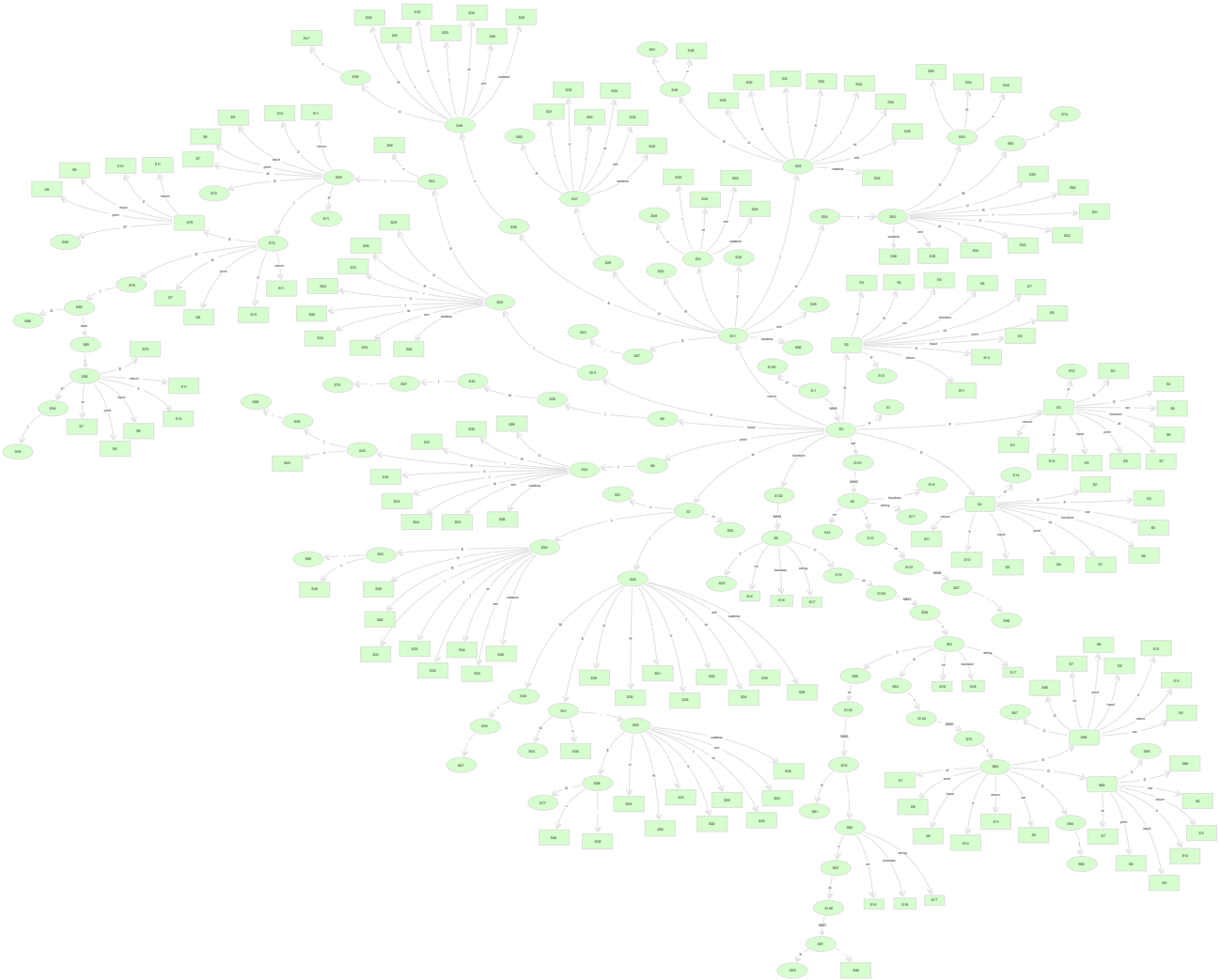
```
Ej:
D → var {zdecl:=true} T id; {otras acciones}
Lo transformamos en:
D → var MM T id; {otras acciones}
MM → lambda {zdecl:=true}
```

1.3 Gramática y Autómata Final del A. Sintáctico

Terminales = { ; { } *id ent cadena* ( ) + < ! = |= *var int boolean string print input* , *return function if else* }
NoTerminales = { *P D T F T1 A K C S L M Q S1 G X E U R V S2* }
Axioma = *P*
Producciones = {
P → *D P*
P → *F P*
P → *S P*
D → *var T id* ;
T → *int*
T → *string*
T → *boolean*
F → *function T1 id* (*A*) { *C* }
T1 → *lambda*
T1 → *T*
A → *T id K*
A → *lambda*
K → *lambda*
K → , *T id K*
C → *D C*
C → *S C*
C → *lambda*
S → *id L E* ;
S → *id* (*M*) ;
S → *print* (*E*) ;
S → *input* (*id*) ;
S → *if* (*E*) *S1*
S → *return X* ;
L → |=
L → =
M → *E Q*
M → *lambda*
Q → *lambda*
Q → , *E Q*
S1 → { *S2* } *G*
S1 → *S*

$G \rightarrow \ else \ \{ \ S2 \ \}$
$G \rightarrow \ lambda$
$X \rightarrow \ E$
$X \rightarrow \ lambda$
$E \rightarrow \ E < U$
$E \rightarrow \ U$
$U \rightarrow \ U + R$
$U \rightarrow \ R$
$R \rightarrow \ ! \ V$
$R \rightarrow \ V$
$V \rightarrow \ (\ E \ )$
$V \rightarrow \ id$
$V \rightarrow \ id \ (\ M \ )$
$V \rightarrow \ ent$
$V \rightarrow \ cadena$
$S2 \rightarrow \ S \ S2$
$S2 \rightarrow \ S$
$P \rightarrow \ lambda$
}

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



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