P’ → {TSG:= creaTS(), TSactual:=TSG, DespG:= 0} P {DestruyeTS (TSG)}

P → BP {}

P → FP {}

P → 𝜆 {}

B → let {ZonaDecl:= true} T id; {ZonaDecl:= false,

InsertaTipoTS (id.pos, T.tipo),

If (TSL = NULL) Then

{InsertaDespTS (id.pos, DespG) DespG := DespG + T.ancho}

Else

{InsertaDespTS (id.pos, DespL) DespL := DespL + T.ancho},

B.tipo := tipo\_ok,

B.tipoRet := vacío,

}

B → if (E) G {if (E.tipo != lógico) Then

{ B.tipo := tipo\_error

B.tipoRet : = vacio }

Else

{ B.tipo := G.tipo

B.tipoRet := G.tipoRet }

}

B → S {B.tipo := S.tipo, B.tipoRet := S.tipoRet}

T → int {T.tipo := entero, T.ancho := 1}

T → string {T.tipo := cadena, T.ancho := 64}

T → boolean {T.tipo := lógico, T.ancho := 1}

G → S {G.tipo := S.tipo, G.tipoRet := S.tipoRet}

G → { C } O { If (C.tipo = O.tipo = tipo\_ok) Then

{ G.tipo := tipo\_ok }

Else

{ G.tipo := tipo\_error }

If (C.tipoRet = O.tipoRet) Then

{ G.tipoRet := C.tipoRet }

Else If (O.tipoRet = vacío) Then

{ G.tipoRet := C.tipoRet }

Else If (C.tipoRet = vacío) Then

{ G.tipoRet := O.tipoRet }

Else

{ G.tipoRet := tipo\_error }

C → BC(1) {If (B.tipo = C1.tipo = tipo\_ok) Then

{ C.tipo := tipo\_ok }

Else

{ C.tipo := tipo\_error }

If (O.tipoRet = C1.tipoRet) Then

{ C.tipoRet := O.tipoRet }

Else If (C1.tipoRet = vacío) Then

{ G.tipoRet := O.tipoRet }

Else If (O.tipoRet = vacío) Then

{ G.tipoRet := C1.tipoRet }

Else

{ G.tipoRet := tipo\_error }

C → 𝜆 {C.tipo := tipo\_ok, C.tipoRet := vacio}

O → else { C } {O.tipo := C.tipo, O.tipoRet := C.tipoRet}

O → 𝜆 {O.tipo := tipo\_ok, O.tipoRet := vacio}

S → id W { if (W.tipo = function) {

If (W.param = buscarParamTS(id.pos))Then

{ S.tipo := tipo\_ok }

Else

{ S.tipo := tipo\_error }

Else if (buscaTipoTS(id.pos) != null) Then

{ InsertaTipoTS (id.pos, entero)

If (TSL = NULL) Then

{InsertaDespTS (id.pos, DespG) DespG := DespG + 1}

Else

{InsertaDespTS (id.pos, DespL) DespL := DespL + 1},

S.tipo := tipo\_ok,

}

Else if (W.tipo = buscaTipoTS(id.pos)) Then

{ S.tipo := tipo\_ok }

Else

{ S.tipo := tipo\_error }

}

S.tipoRet := vacío }

S → print (E); {S.tipo := if (E.tipo {entero, cadena})

Then tipo\_ok

Else tipo\_error,

S.tipoRet := vacío}

S → input (id); { if (BuscaTipoTS (id.pos) {entero, cadena})) Then

{ S.tipo := tipo\_ok }

Else if (buscaTipoTS(id.pos) = lógico) Then

{ S.tipo := tipo\_error }

Else

{ InsertaTipoTS (id.pos, entero),

If (TSL = NULL) Then

{InsertaDespTS (id.pos, DespG) DespG := DespG + 1}

Else

{InsertaDespTS (id.pos, DespL) DespL := DespL + 1},

S.tipo := tipo\_ok,

S.tipoRet := vacío}

S → return X; {S.tipo := if (X.tipo != tipo\_error)

Then tipo\_ok

Else tipo\_error,

S.tipoRet := X.tipo}

W → -=E; {W.tipo := E.tipo, W.param := vacío}

W → =E; {W.tipo := E.tipo, W.param := vacío}

W → (L); {W.tipo := function, W.param := L.tipo}

X → E {X.tipo := E.tipo}

X → 𝜆 {X.tipo := vacío}

L → EQ {If (Q.tipo = vacío)

Then L.tipo := E.tipo

Else L.tipo := E.tipo x Q.tipo}

L → 𝜆 {L.tipo := vacío}

Q → , EQ(1)  {If (Q1.tipo = vacío)

Then Q.tipo := E.tipo

Else Q.tipo := E.tipo x Q1.tipo}

Q → 𝜆 {Q.tipo := vacío}

F → function id {TSL := CreaTS (), TSactual := TSL, DespL := 0, InsertaEtTS (id.pos, nuevaEt())}

H {InsertaTipoRet (id.pos, H.tipo), ZonaDecl : = True}

1. { ZonaDecl : = False, InsertaTipoParam (id.pos, A.param) }

{ C }

{ If (C.tipoRet != H.tipo) Then Error

If (C.tipo = tipo\_error) Then Error

DestruyeTS (TSL)

TSactual := TSG }

H → T {H.tipo := T.tipo}

H → 𝜆 {H.tipo := vacío}

A → T id K {InsertaTipoTS (id.pos, T.tipo)

InsertaDespTS (id.pos, DespL)

DespL := DespL + T.ancho

If (k.param = vacio) Then

{ A.param := T.tipo }

Else

{ A.param := T.tipo x K.param }

A.tipo := K.tipo

}

A → 𝜆 {A.tipo := tipo\_ok, A.param := vacio}

K → , T id K(1) { InsertaTipoTS (id.pos, T.tipo)

InsertaDespTS (id.pos, DespL)

DespL := DespL + T.ancho

If (k1.param = vacio) Then

{ K.param := T.tipo }

Else

{ K.param := T.tipo x K1.param }

K.tipo := K1.tipo

}

K → 𝜆 {K.tipo := tipo\_ok, K.param := vacio}

E → RE’ { E.tipo := If (R.tipo = E’.tipo = lógico)

Then lógico

Else If (E’.tipo = vacío)

Then R.tipo

Else tipo\_error}

E’ → &&RE’(1) {E’.tipo := If (R.tipo = lógico and E1’.tipo != tipo\_error) Then lógico Else tipo\_error}

E’ → 𝜆 {E’.tipo := vacío}

R → UR’ {If (R’.tipo = lógico)

Then R.tipo := lógico

Else If (R’.tipo = vacío)

Then R.tipo := U.tipo

Else tipo\_error}

R’ → <UR’(1) {R’.tipo := If (U.tipo != tipo\_error and R1’.tipo != tipo\_error) Then lógico Else tipo\_error}

R’ → >UR’(1) {R’.tipo := If (U.tipo != tipo\_error and R1’.tipo != tipo\_error) Then lógico Else tipo\_error}

R’ → 𝜆 {R’.tipo := vacío}

U → VU’ {If (U’.tipo = entero)

Then U.tipo := entero

Else If (U’.tipo = vacío)

Then U.tipo := V.tipo

Else tipo\_error}

U’ → +VU’(1) {U’.tipo := If (V.tipo = entero and U1’.tipo = entero || U1’.tipo = vacio) Then entero Else tipo\_error}

U’ → -VU’(1) {U’.tipo := If (V.tipo = entero and U1’.tipo = entero || U1’.tipo = vacio) Then entero Else tipo\_error}

U’ → 𝜆 {U’.tipo := vacío}

V → id D { if (D.tipo = function) {

If (D.param = buscarParamTS(id.pos)) Then

{ V.tipo := buscaTipoRet(id.pos)

V.ancho := anchoTipo(V.tipo) }

Else

{ V.tipo := tipo\_error

V.ancho := 0 }

Else if (buscaTipoTS != null) Then

{ InsertaTipoTS (id.pos, entero),

If (TSL = NULL) Then

{InsertaDespTS (id.pos, DespG) DespG := DespG + 1}

Else

{InsertaDespTS (id.pos, DespL) DespL := DespL + 1},

V.tipo := entero

V.ancho : = 1}

Else

{ V.tipo := buscaTipoTS(id.pos)

V.ancho := anchoTipo(V.tipo) }

}

V → (E) {V.tipo := E.tipo, V.ancho := 0}

V → entero {V.tipo := entero, V.ancho := 1}

V → cadena {V.tipo := cadena, V.ancho := 64}

D → (L) {D.tipo := function, D.param := L.tipo}

D → 𝜆 {D.tipo := vacío, D.param := vacío}