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

P → BP {}

P → FP {}

P → 𝜆 {}

B → let T id; {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 {B.tipo := if (E.tipo != lógico)

Then tipo\_error

Else 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} ¿Este tamaño tiene sentido? (Documentacion PDL)

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

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

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

Then tipo\_ok

Else tipo\_error,

G.tipoRet := If (C.tipoRet = O.tipoRet)

Then C.tipoRet

Else If (O.tipoRet = vacío)

Then C.tipoRet

Else If (C.tipoRet = vacío)

Then O.tipoRet

Else tipo\_error}

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

Then tipo\_ok

Else tipo\_error

C.tipoRet := If (B.tipoRet = C1.tipoRet)

Then B.tipoRet

Else If (C1.tipoRet = vacío)

Then B.tipoRet

Else If (B.tipoRet = vacío)

Then C1.tipoRet

Else 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){

InsertaTipoTS (id.pos, W.tipo), If (TSL = NULL)

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

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

S.tipo := tipo\_ok

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

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 {InsertaTipoTS (id.pos, entero), If (TSL = NULL)

Then {InsertaDespTS (id.pos, DespG) DespG := DespG + 2}

Else {InsertaDespTS (id.pos, DespL) DespL := DespL + 2},

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 {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)}

(A) { If (A.tipo = tipo\_error)

Then error,

InsertaTipoParam (id.pos, A.param)

{ C } ¿Falta algo? Pablo se ha rayado

{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 {If (InsertaTipoTS (id.pos, T.tipo) = true)

Then InsertaDespTS (id.pos, DespL)

DespL := DespL + T.ancho

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

A.tipo := K.tipo

Else tipo\_error}

A → 𝜆 {A.tipo := tipo\_ok}

K → , T id K {If (InsertaTipoTS (id.pos, T.tipo) = true)

Then InsertaDespTS (id.pos, DespL)

DespL := DespL + T.ancho

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

K.tipo := K1.tipo

Else K.tipo := tipo\_error}

K → 𝜆 {K.tipo := tipo\_ok}

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’ {E’.tipo := If (R.tipo = lógico and E’.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’ {R’.tipo := If (U.tipo != tipo\_error and R’.tipo != tipo\_error) Then lógico Else tipo\_error}

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

U’ → -VU’ {U’.tipo := If (V.tipo != tipo\_error and U’.tipo != tipo\_error) 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 := tipo\_ok

Else V.tipo := tipo\_error

Else if (buscaTipoTS(id.pos) = NULL){

InsertaTipoTS (id.pos, D.tipo), If (TSL = NULL)

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

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

V.tipo := tipo\_ok

Else if (D.tipo = buscaTipoTS(id.pos)){

V.tipo := tipo\_ok

Else V.tipo := tipo\_error

}

V.ancho := 0}

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}