Al no haber anidamiento no hacen falta una pila de tablas de símbolos.

Nos servirá tener:

- Una TSG (General) que se crea al principio y siempre esta

disponible.

- Una TSL (Local) que se crea al comenzar una función y se

destruye al terminar.

El Analizador Semántico se encarga de comprobar el flujo del control

(por ejemplo, si el return aparece dentro de una función con el tipo adecuado,

o el break en un case, o el continue en una sentencia iterativa...) además de la comprobación de tipos.

Terminales = { var id ; if ( ) while } { switch case break default int bool string return print

prompt function , || && == != < > -= ! = + - \* / % CAD : cteent ctebool }

NoTerminales = { P B T S \_S F H L Q A K X C E \_E R \_R U \_U V \_V W \_W Z \_Z G \_G D }

Axioma = P

Producciones = {

P -> B P {Nada}

P -> F P {Nada}

P -> lambda {Nada}

B -> var T id ; { insertaTipoTS(id.lex, T.Tipo); insertaDespTS(id.lex, T.desp)}

B -> if ( E ) S

B -> while ( E ) { C }

B -> switch ( E ) { D }

B -> S

T -> int {T.Tipo = int}

T -> bool {T.Tipo = bool}

T -> string {T.Tipo = string}

S -> id \_S

S -> return X ;

S -> print ( E ) ;

S -> prompt ( id ) ;

S -> break ;

\_S -> = E ;

\_S -> -= E ;

\_S -> ( L ) ;

F -> function H id ( A ) { C }

H -> T

H -> lambda

L -> E Q

L -> lambda

Q -> , E Q

Q -> lambda

A -> T id {} K

A -> lambda

K -> , T id {} K

K -> lambda

X -> E

X -> lambda

C -> B C

C -> lambda

E -> R \_E

\_E -> || R \_E

\_E -> lambda

R -> U \_R

\_R -> && U \_R

\_R -> lambda

U -> V \_U

\_U -> == V

\_U -> != V

\_U -> lambda

V -> W \_V

\_V -> > W \_V

\_V -> < W \_V

\_V -> lambda

W -> Z \_W

\_W -> + Z \_W

\_W -> - Z \_W {\_W.Tipo = if( (G.Tipo==int) AND (\_Z1.Tipo==int OR vacio) ) then int else

tipo\_error }

\_W -> lambda {\_W.Tipo = tipo\_vacio } Sintetizado

\_W puede ser: int, vacío, tipo\_error

Z -> G \_Z {Z.Tipo = if(\_Z.Tipo== tipo\_vacio) then G.Tipo else

if(\_Z.Tipo== int AND G.Tipo == int) then int else

tipo\_error }

\_Z -> \* G \_Z {\_Z.Tipo = if( (G.Tipo==int) AND (\_Z1.Tipo==int OR vacio) ) then int else

tipo\_error }

\_Z -> / G \_Z {\_Z.Tipo = if( (G.Tipo==int) AND (\_Z1.Tipo==int OR vacio) ) then int else

tipo\_error }

\_Z -> % G \_Z1 {\_Z.Tipo = if( (G.Tipo==int) AND (\_Z1.Tipo==int OR vacio) ) then int else

tipo\_error }

\_Z -> lambda {\_Z.Tipo = tipo\_vacio } Sintetizado

Z puede ser: int, string, bool, vacio, tipo\_error

\_Z puede ser: int, vacio, tipo\_error

G -> ! G1 {G.Tipo = if(G1.Tipo==bool) then bool else tipo\_error} Sintetizado

G -> id \_G {G.Tipo = if(\_G.Tipo== tipo\_vacio) then buscaTipoTS(id.pos) else

If (buscaTipoTS(id.pos)==\_G.Tipo->t then t else tipo\_error) }

G -> ( E ) {G.Tipo = E.Tipo} Sintetizado

G -> cteent {G.Tipo = int} Sintetizado

G -> CAD {G.Tipo = string} Sintetizado

G -> ctebool {G.Tipo = bool} Sintetizado

\_G -> ( L ) {\_G.Tipo = L.Tipo} tipo\_vacio Sintetizado

\_G -> lambda {\_G.Tipo = tipo\_vacio } Sintetizado

G puede ser: int, string, bool, vacio, tipo\_error

\_G puede ser: vacio, arg1 x arg2 x … x argN

D -> case cteent : C D

D -> default : C

}