Analizador Semántico

Aumentada

Z → P

```
1. If(P.tipo=tipoError) error()
2. Z.tipo = P.tipo
```

Inicio

P → BP

```
1. P.tipo = (B.tipo == tipoOk && P.tipo==tipoOk)? tipoOk :
   tipoError
```

P→ FP

```
1. P.tipo = (F.tipo == tipoOk && P.tipo==tipoOk)? tipoOk :
   tipoError
```

P→ λ

```
1. P.tipo = tipoOk
```

Funciones

F → I J G

```
1. if(G.hasReturn && G.returnType == I.Tipo)
2.   F.tipo = tipoOk
3. Else{
4.   F.tipo = tipoError
5.   Error()
6. }
```

I → function H id

```
1. if(!estaTS(id))
2.   I.tipo = H.tipo
3. else{
4.   I.tipo = tipoError
5.   error()
6. }
```

J → (A)

```
1. insertarDatosFuncionTS(idUltimaFuncion,A.numeros,A.tipos)
2. # Se almacena el ultimo id de la funcion usado(Permitiendo
   recursividad dentro de la funcion)
```

G → { C }

```
1. G.tipo = C.tipo
2. G.returnType = C.returnType
3. G.hasReturn = C.hasReturn
```

H → T

```
1. H.tipo = T.tipo
```

H → λ

```
1. H.tipo= EMPTY
```

A → T id K

```
1. A.numeros=1+k.numeros
2. A.tipos=[T] U K.tipos
3. insertarTSActual(id,T.tipo)
```

A → λ

```
1. A.numeros = 0
2. A.tipos=[]
```

K → , T id K

```
1. K.numeros=1+K1.numeros
2. K.tipos = [T.tipo] U K.tipos
3. insertarTSActual(id,T.tipo)
```

K → λ

```
1. K.numeros = 0
2. K.tipo = []
```

C → BC

```
1. C.tipo = (B.tipo = C1.tipo = tipoOk)? tipoOk : (tipoError &&
   Error() )
2.
3. if(B.hasReturn){
4.   c.returnType = B.returnType
5. }else if(C1.hasReturn){
6.   C.returnType = C1.returnType
7. }
```

C → λ

```
1. C.tipo = tipoOk
```

Sentencias:

Simples

S → id = E ;

```
1. If(comprobarTS(id))
2.   S.tipo = (getTipoTS(id)== E.tipo)? tipoOk : tipoError
3. Else{
4.   insertarTSGlobal(id,Entero)
5.   S.tipo = (E.tipo== E.tipo)? tipoOk : tipoError
6. }
```

S → id (L) ;

```
1. If(estaTS(id))
2.   S.tipo = comprobarArgumentosYNum(L.tipos, L.numeros)? tipoOk :
   tipoError
3. Else
4.   Error("funcion no declarada")
```

S → alert (E) ;

```
1. S.tipo = (E.tipo == NUMBER || E.tipo=String)? tipoOK : tipoError
```

S → input (id) ;

```
1. If(estaTS(id))
2.          S.tipo = (getTipoTS(id) == NUMBER ||
   getTipoTS(id)=String)? tipoOK : tipoError
3. Else
4.   insertarTSGlobar(id,ENTERO)
5.   S.tipo = (getTipoTS(id) == NUMBER || getTipoTS(id)=String)?
   tipoOK : tipoError
```

S → return X ;

```
1. S.returnType = X.tipo
2. S.hasReturn=true
```

S → id -= E ;

```
1. if(comprobarTS)
2.   S.tipo = (getTipoTS(id) = E.tipo = NUMBER)? tipoOk: tipoError
3. Else{
4.   insertarTSGlobal(id,ENTERO)
5.   S.tipo = (E.tipo = NUMBER)? tipoOk: tipoError
6. }
```

Compuestas

B → if (E) S

```
1. B.atrib = (E.tipo == BOOLEAN && S.tipo=tipoOk)? S.atrib : (
   error()&& tipoError)
```

B → let T id ;

```
1. If(!estaTS)
2.   insertarTSActual(getTipoTS(id),T.tipo)
3.   B.tipo = tipoOk
4. Else
5.   B.tipo = tipoError
```

B → S

```
1. B.atributos = S.atributos
2. # .atributos denota todos los posibles atributos que tenga S
```

B → do { C } while (E);

```
1. B.atrib = (E.tipo == BOOLEAN && C.tipo=tipoOk)? C.atrib :
   (error() && B.tipo = tipoError)
```

T → number

```
1. T.tipo = NUMBER
```

T → boolean

```
1. T.tipo = BOOLEAN
```

T → String

```
1. T.tipo = STRING
```

Expresiones

E → E || R

```
1. E.tipo = (E1.tipo = R.tipo = BOOLEAN)? BOOLEAN: error() &&
   E.tipo = tipoError
```

E → R

```
1. E.tipo = R.tipo
```

R → R && U

```
1. R.tipo = (R1.tipo = U.tipo = BOOLEAN)? BOOLEAN: error() &&
   R.tipo = tipoError
```

R → U

```
1. R .tipo = U.tipo
```

U → U == V

```
1. U.tipo = (U1.tipo == V.tipo)? BOOLEAN: error() && U.tipo =
   tipoError
```

U → U != V

```
1. U.tipo = (U1.tipo == V.tipo)? BOOLEAN: error() && U.tipo =
   tipoError
```

U → V

```
1. U.tipo = V.tipo
```

V → V + W

```
1. V.tipo = (V1.tipo == NUMBER && W.tipo == NUMBER)? NUMBER :
   error() && V.tipo = tipoError
```

V → V - W

```
1. V.tipo = (V1.tipo == NUMBER && W.tipo == NUMBER)? NUMBER :
   error() && V.tipo = tipoError
```

V → W

```
1. V.tipo = W.tipo
```

W → id

```
1. if(!estaEnTS(id.valor)) {
2.          insertarTSGlobal(id,entero)
3. }
4. W.tipo = getTipoTS(id)
```

W → (E)

```
1. W.tipo = E.tipo
```

W → id (L)

```
1. if(getReturnTypeTS(id) != Empty &&
   comprobarArgumentosYNum(id,L.tipos,L.numeros)){
2.   W.tipo = getReturnTypeTS(id)
3. }else{
4.   error()
5.   W.tipo = tipoError
6. }
```

W → entero

```
1. W.tipo = NUMBER
```

W → cadena

```
1. W.tipo = STRING
```

Argumentos de la función

L → E Q

```
1. L.numeros = Q.numeros+1
2. L.tipos= [E.tipo] U Q.tipos
```

L → λ

```
1. L.numeros = 0
2. L.tipos=[]
```

Q → , E Q

```
1. Q.numeros = Q1.numeros+1
2. Q.tipos = [E.tipo] U Q.tipos
```

Q → λ

```
1. Q.numeros = 0
2. Q.tipos=[]
```

Return

X → E

```
1. X.tipo = E.tipo
```

X → λ

```
1. X.tipo = EMPTY
```

0. Z → P
1. P → BP
2. P→ FP
3. P→ λ
4. F → I J G
5. I → function H id
6. J → (A)
7. G → { C }
8. H → T
9. H → λ
10. A → T id K
11. A → λ
12. K → , T id K
13. K → λ
14. C → BC
15. C → λ
16. S → id = E ;
17. S → id (L) ;
18. S → alert (E) ;
19. S → input (id) ;
20. S → return X ;
21. S → id -= E ;
22. B → if (E) S
23. B → let T id ;
24. B → S
25. B → do { C } while (E);
26. T → number
27. T → boolean
28. T → String
29. E → E || R
30. E → R
31. R → R && U
32. R → U
33. U → U == V
34. U → U != V
35. U → V
36. V → V + W
37. V → V – W
38. V → W
39. W → id
40. W → (E)
41. W → id (L)
42. W → entero
43. W → cadena
44. L → E Q
45. L → λ
46. Q → , E Q
47. Q → λ
48. X → E
49. X → λ