
Procesadores de Lenguajes

MEMORIA FINAL

Grupo 55

Daniel Tomás Sánchez
Aarón Cabero Blanco
Alejandro Cuadrón

Curso 2020/2021

Índice

1	Introducción	2
2	Diseño Analizador Léxico	3
2.1	Tokens	3
2.2	Gramática Regular	4
2.3	Autómata Finito Determinista	4
2.4	Acciones Semánticas	5
2.5	Errores	6
3	Diseño Analizador Sintáctico	7
3.1	Gramática	7
3.2	Tabla LR(1)	8
4	Diseño Analizador Semántico	11
5	Diseño Tabla de Símbolos	16
6	Anexo	17
6.1	Casos de prueba correctos	17
6.1.1	Prueba correcta 1	17
7	Referencias	19

1 | Introducción

El trabajo completo, tanto el léxico, como el sintáctico y el semántico, ha sido realizado con la herramienta o librería externa "SLY"[1].

Opciones de grupo:

- Sentencias: Sentencia repetitiva (**for**)
- Operadores especiales: Post-auto-decremento (**-- como sufijo**)
- Técnicas de Análisis Sintáctico: **Ascendente**
- Comentarios: Comentario de bloque (**/* */**)
- Cadenas: Con comillas dobles (**" "**)

2 | Diseño Analizador Léxico

2.1 Tokens

▪ Identificador	<ID, punteroTS>
▪ Constante entera	<CTEENTERA, valor>
▪ Cadena de caracteres	<CADENA, lexema>
▪ false	<CTELOGICA, 0>
▪ true	<CTELOGICA, 1>
▪ Palabra reservada Number	<NUMBER, ->
▪ Palabra reservada String	<STRING, ->
▪ Palabra reservada Boolean	<BOOLEAN, ->
▪ Palabra reservada Let	<LET, ->
▪ Palabra reservada Alert	<ALERT, ->
▪ Palabra reservada Input	<INPUT, ->
▪ Palabra reservada Function	<FUNCTION, ->
▪ Palabra reservada Return	<RETURN, ->
▪ Palabra reservada If	<IF, ->
▪ Palabra reservada For	<FOR, ->
▪ --	<OPESP, ->
▪ -	<OPARIT, ->
▪ =	<OPASIG, ->
▪ ==	<OPREL, ->
▪ &&	<OPLOG, ->
▪ (<ABPAREN, ->
▪)	<CEAPAREN, ->
▪ {	<ABLLAVE, ->
▪ }	<CELLAVE, ->
▪ ,	<COMA, ->
▪ ;	<PUNTOYCOMA, ->

2.2 Gramática Regular

Axioma = A

$A \rightarrow \text{del } A \mid d D \mid " S \mid / C \mid l I \mid -M \mid = E \mid \& N \mid (\mid) \mid \{ \mid \} \mid ; \mid ,$

$D \rightarrow d D \mid \lambda$

$S \rightarrow " \mid c S$

$C \rightarrow * C'$

$C' \rightarrow * C'' \mid c C'$

$C'' \rightarrow / A \mid c C'$

$I \rightarrow d I \mid l I \mid _ I \mid \lambda$

$M \rightarrow - \mid \lambda$

$E \rightarrow = \mid \lambda$

$N \rightarrow \&$

Siendo d un dígito, l una letra, c cualquier otro carácter y del un delimitador.

2.3 Autómata Finito Determinista

2.4 Acciones Semánticas

```
A: leer
B: number = int(d), leer
C: number = number * 10 + int(d), leer
D: if number > 32767
    pError("Número fuera de rango")
    else
        genToken(CTEENTERA, number);
E: string = "", contador = 0, leer
F: string = string + otroCS, contador++, leer
G: if contador > 64
    pError("Cadena demasiado larga")
    else
        genToken(CADENA, string)
    leer
H: string = 1, leer
I: string = string + 1/D/_ , leer
J: if palabrasReservadas.contains(string)
    if string == "number"
        genToken(NUMBER, -)
    elif string == "string"
        genToken(STRING, -)
    elif string == "boolean"
        genToken(BOOLEAN, -)
    elif string == "let"
        genToken(LET, -)
    elif string == "alert"
        genToken(ALERT, -)
    elif string == "input"
        genToken(INPUT, -)
    elif string == "return"
        genToken(RETURN, -)
    elif string == "if"
        genToken(IF, -)
    else
        genToken(FOR, -)
```

```

else    // palabrasReservadas.contains(string) = False
    puntero = TS.get(string)
    if zona_decl == True
        if puntero != None
            pError("Identificador ya declarado")
        else
            TS.update(string)
            puntero = TS.get(string)
            genToken(ID, puntero)
    else
        if puntero == None
            TS.update(string)
            puntero = TS.get(string)
            genToken(ID, puntero)
        else
            genToken(ID, puntero)

L: genToken(OPARIT, -)
M: genToken(OPESP, -), leer
N: genToken(OPASIG, -)
O: genToken(OPREL, -), leer
P: genToken(OPLOG, -), leer
Q: genToken(ABPAREN, -), leer
R: genToken(CEPAREN, -), leer
S: genToken(ABLLAVE, -), leer
T: genToken(CELLAVE, -), leer
U: genToken(COMA, -), leer
V: genToken(PUNTOYCOMA, -), leer
W: genToken(EOF, -), leer

```

2.5 Errores

Error léxico (siempre se lanza cuando el analizador léxico encuentra un error).

1. Cadena con longitud mayor de 64 caracteres.
2. Número fuera de rango (mayor de 32767).
3. Identificador ya declarado.
4. Carácter ilegal.

Todo error va acompañado de la *línea* y *columna* en el que se ha encontrado dicho error.

3 | Diseño Analizador Sintáctico

3.1 Gramática

Axioma = B

No Terminales = { A B C D E F G H I J K L M N O P Q R S T U V W F1 F2 F3 }

Terminales = { && == - -- () = , ; id ent cad log let alert input return for if number
boolean string function }

Producciones = {

B → D

D → F D

D → G D

D → λ

G → if (E) S

G → S

S → H ;

H → id (I)

I → E J

I → λ

J → , E J

J → λ

S → K ;

K → id = E

S → alert (E) ;

S → input (id) ;

S → return L ;

L → E

L → λ

G → let M T id ;

M → λ

T → number

T → boolean

T → string

G → for (N ; E ; O) C

N → K

N → λ

O → K

O → -- id

O → λ

C → G C

C → λ

F → F1 F2 F3

F1 → function P Q id

P → λ

Q → T

Q → λ

F2 → (A)

A → T id AA

A → λ

AA → , T id AA

AA → λ

F3 → C

E → E && R

E → R

R → R == U

R → U

U → U - V

U → V

V → -- id

V → id

V → (E)

V → H

V → ent

V → cad

V → log

}

3.2 Tabla LR(1)

[illegible]

9

[illegible]

4 | Diseño Analizador Semántico

Acción semántica previa a empezar a funcionar el procesador:

```
{ TS_g = crear_TS()
  TS_actual = TS_g
  desp = 0
  zona_decl = false }
```

```
B → D {}
D → F D {}
D → G D {}
D → lambda {}
```

```
G → if ( E ) S { if E.tipo != "lógico"
                  then error(1) }

G → S {}
S → H ; {}
H → id ( I ) { if busca_tipo_TS(id.pos) != "función"
                then error(15)
                else if longitud(I.tipo) != busca_num_params_TS(id.pos)
                  then error(2)
                else if busca_tipo_TS(id.pos) != I.tipo
                  then error(3)
                else
                  H.tipo = busca_tipo_devuelto_TS(id.pos) }

I → E J { if longitud(J.tipo) == 0
          then I.tipo = E.tipo
          else
            I.tipo = E.tipo x J.tipo }

J → , E J1 { if longitud(J1.tipo) == 0
              then J.tipo = E.tipo
              else
                J.tipo = E.tipo x J1.tipo }

J → lambda {}
I → lambda { I.tipo = void }

S → K ; {}
K → id = E { if busca_tipo_TS(id.pos) != None
              añade_tipo_TS(id.pos, "entero")
              añade_desp_TS(id.pos, desp)
              desp+=2
              if busca_tipo_TS(id.pos) != E.tipo
                then error(10) }
```

```

S → alert ( E ) ; { if E.tipo != "cadena" && E.tipo != "entero"
                    then error(4) }
S → input ( id ) ; { if busca_tipo_TS(id.pos) != None
                    añade_tipo_TS(id.pos, "entero")
                    añade_desp_TS(id.pos, desp)
                    desp+=2
                    if busca_tipo_TS(id.pos) != "cadena"
                      && busca_tipo_TS(id.pos) != "entero"
                      then error(5) }
S → return L ; { if zona_function != true
                then error(7)}
                else if L.tipo != tipo_return
                then error(9) }
L → E { L.tipo = E.tipo }
L → lambda { L.tipo = void }

```

```

G → let M T id ; { añadir_tipo_TS(id.pos, T.tipo)
                  añadir_desp_TS(id.pos, desp)
                  desp += T.ancho
                  zona_declaración = false }
M → lambda { zona_declaracion = true }
T → number { T.tipo = "entero"
            T.ancho = 2} // size_of("entero")
T → boolean { T.tipo = "lógico"
             T.ancho = 2} // size_of("lógico")
T → string { T.tipo = "cadena"
            T.ancho = 128} // size_of("cadena")

```

```

G → for ( N ; E ; O ) { C } { if E.tipo != "lógico"
                              then error(6) }
N → K {}
N → lambda {}
O → K {}
O → -- id { if busca_tipo_TS(id.pos) != None
            añade_tipo_TS(id.pos, "entero")
            añade_desp_TS(id.pos, desp)
            desp+=2
            if busca_tipo_TS(id.pos) != "entero"
              then error(11) }
O → lambda {}
C → G C {}
C → lambda {}

```

```

F → F1 F2 F3 { destruye_TS (TS_l)
                zona_function = false
                TS_actual = TS_g
                desp = desp_g
                tipo_return = NULL }
F1 → function P Q id { TS_l = crear_TS()
                      TS_actual = TS_l
                      desp_g = desp
                      desp = 0
                      pos_id_fun = id.pos
                      zona_function = true
                      tipo_return = Q.tipo
                      añadir_tipo_devuelto_TS(id.pos, tipo_return)
                      añadir_tipo_TS(id.pos, "función")
                      inserta_et_TS (id.pos, nueva_et ()) }
P → lambda {}
Q → T { zona_declaracion = true
        Q.tipo = T.tipo }
Q → lambda { zona_declaracion = true
             Q.tipo = void }

F2 → ( A ) { añadir_param_TS(pos_id_fun, A.tipo, longitud(A.tipo))
           zona_declaración = false }
A → T id AA { añadir_tipo_TS(id.pos, T.tipo)
             añadir_desp_TS(id.pos, desp)
             desp += T.ancho
             if longitud(AA.tipo) == 0
               then A.tipo = T.tipo
             else
               A.tipo = T.tipo x AA.tipo }
A → lambda { A.tipo = void }
AA → , T id AA1 { añadir_tipo_TS(id.pos, T.tipo)
                 añadir_desp_TS(id.pos, desp)
                 desp += T.ancho
                 if longitud(AA1.tipo) == 0
                   then AA.tipo = T.tipo
                 else
                   AA.tipo = T.tipo x AA1.tipo }
AA → lambda {}

F3 → { C } {}

E → E1 && R { if E1.tipo != "lógico" || R.tipo != "lógico"
              then error(12)
              else
                E.tipo = "lógico" }
E → R {E.tipo = R.tipo }

```

```

R → R1 == U { if R1.tipo != "entero" || U.tipo != "entero"
               then error(13)
               else
                 E.tipo = "lógico" }
R → U { R.tipo = U.tipo }
U → U1 - V { if U1.tipo != "entero" || V.tipo != "entero"
              then error(14)
              else
                U.tipo = "entero" }
U → V { U.tipo = V.tipo }
V → -- id { if busca_tipo_TS(id.pos) != None
             añade_tipo_TS(id.pos, "entero")
             añade_desp_TS(id.pos, desp)
             desp+=2
             if busca_tipo_TS(id.pos) != "entero"
             then error(15)
             else
               V.tipo = "entero" }
V → id { if busca_tipo_TS(id.pos) != None
          añade_tipo_TS(id.pos, "entero")
          añade_desp_TS(id.pos, desp)
          desp+=2
          V.tipo = busca_tipo_TS(id.pos) }
V → ( E ) { V.tipo = E.tipo }
V → H { V.tipo = H.tipo }
V → ent { V.tipo = "entero" }
V → cad { V.tipo = "cadena" }
V → log { V.tipo = "lógico" }

error_code_dict = {
  1: "La condición debe ser un lógico",
  2: "El número de parámetros introducidos no son los esperados, \
      deberían ser {busca_num_params_TS(id.pos)}",
  3: "El tipo de los parámetros no es el esperado, \
      se esperaban {busca_tipo_params_TS(id.pos)}",
  4: "La expresión introducida no es una cadena o un entero",
  5: "La variable introducida no es de tipo cadena o entero",
  6: "La condición debe ser un lógico",
  7: "No puede haber una sentencia return fuera de una función",
  8: "No se permite la definición de funciones anidadas",
  9: "El tipo de retorno no corresponde con el tipo de retorno de la función, \
      se esperaba {tipo_return}",
  10: "El tipo de la variable a asignar no corresponde con el tipo asignado",
  11: "El operador especial "--" solo trabaja con tipos de datos enteros",
  12: "El operador lógico "&&" solo trabaja con tipos de datos lógicos",
  13: "El operador de relación "==" solo trabaja con tipos de datos enteros",
  14: "El operador aritmético "-" solo trabaja con tipos de datos enteros",
  15: "La variable no se puede invocar como una función, con argumentos"
}

```

```
function error(error_code):  
    res = ""  
    if error_code <= 6:  
        res = "ErrorDeAtributo: "  
    elif error_code >= 9:  
        res = "ErrorDeTipado: "  
    else  
        res = "NoImplementado: "  
    res = res + error_code_dict.get(error_code, default="Código no válido")  
    print(res)
```


5 | Diseño Tabla de Símbolos

La TS se compone de una lista de tablas, una de ellas es global y se crea al empezar, mientras que el resto son locales y se van creando a medida que avanza la compilación. Están ordenadas en orden de creación. Cada tabla tiene un flag que indica si la tabla existe (esta activa) o se ha eliminado, pero realmente no las eliminamos para posteriormente imprimirlas.

A su vez cada tabla contiene una lista de diccionarios, cada diccionario simboliza una entrada en la tabla de símbolos. Un diccionario es un hashmap que tiene como claves la palabra "lexema" y los tipos de atributos que le corresponda (por ejemplo "Tipo" ó "Despl"), como valores tiene el valor del lexema y de sus atributos en si mismos (por ejemplo "main" ó "entero").

6 | Anexo

6.1 Casos de prueba correctos

6.1.1 Prueba correcta 1

Programa introducido:

```
let string texto;  
let string textoAux;  
textoAux = texto;  
alert  
    (textoAux);
```

Tokens:

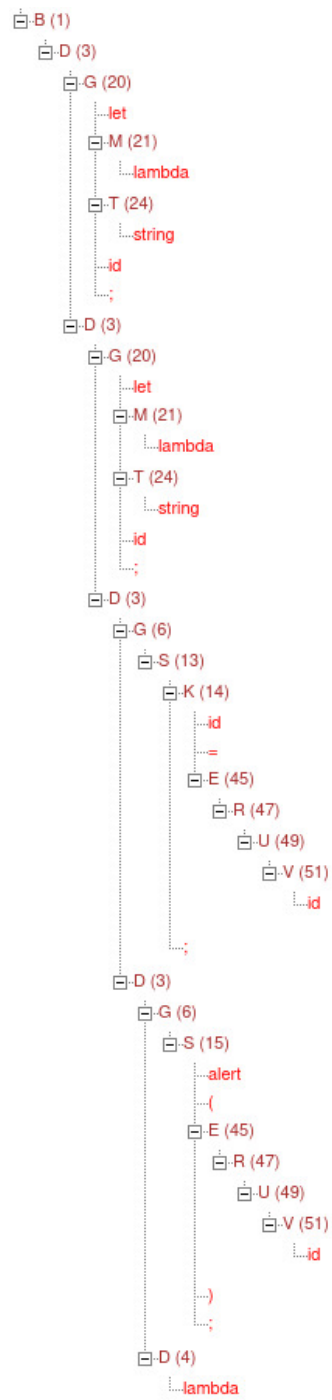
```
<LET , >  
<STRING , >  
<ID , 0>  
<PUNTOYCOMA , >  
<LET , >  
<STRING , >  
<ID , 1>  
<PUNTOYCOMA , >  
<ID , 1>  
<OPASIG , >  
<ID , 0>  
<PUNTOYCOMA , >  
<ALERT , >  
<ABPAREN , >  
<ID , 1>  
<CEPAREN , >  
<PUNTOYCOMA , >
```

Tabla de símbolos:

CONTENIDO DE LA TABLA 0 :

```
*      LEXEMA : 'texto'  
      ATRIBUTOS :  
      + Tipo : 'cadena'  
      + Despl : '0'  
-----  
*      LEXEMA : 'textoAux'  
      ATRIBUTOS :  
      + Tipo : 'cadena'  
      + Despl : '64'  
-----
```

VAST:



7 | Referencias

1. *Documentación librería SLY*
<https://sly.readthedocs.io/en/latest/>
2. *Generador de tabla LR(1)*
<http://jsmachines.sourceforge.net/machines/lr1.html>