Memoria Práctica PDL: Analizador Sintáctico

Andrés Ollero Morales, Gabriel de Oliveira Trindade, Víctor Alejandro Sanz Ararat

Grupo 17

Índice

1. Diseño del Analizador Sintáctico

1.1. Gramática

Para ello hemos tomado la gramática que venía dada en las diapositivas de la explicación de la práctica modificándola a los terminales que escogimos al inicio. Eliminamos la recursividad por la izquierda y la factorizamos para cumplir la condición LL(1).

```
Axioma = P

NoTerminales = { P S SS E R RR U UU V VV L Q X B T A K C F H O D }

Terminales = { ! == + id ( ) constEnt cadena %= print input return , if break
switch case int boolean string let function ; : = { } default }

Producciones = {
     E -> ! R
     E -> R
     R -> U RR
     RR -> + R
     RR -> lambda
     U -> V UU
     UU -> lambda
     UU -> == U
     V -> id VV
     V -> ( E )
     V -> constEnt
     V -> cadena
     VV -> ( L )
     VV -> lambda
     S -> id SS
     SS -> %= E ;
     SS -> = E ;
     SS -> ( L ) ;
     S -> print R ;
     S -> input id ;
     S -> return X ;
     L -> E Q
     L -> lambda
     Q -> , E Q
     Q -> lambda
     X -> E
     X -> lambda
     B -> switch ( E ) { O }
     B -> if ( E ) S
     O -> case E : P D O
     O -> default : P D O
     O -> lambda
     D -> break ;
     D -> lambda
```

```
     B -> let id T ;
     T -> int
     T -> boolean
     T -> string
     B -> S
     F -> function id H ( A ) { C }
     H -> T
     H -> lambda
     A -> T id K
     A -> lambda
     K -> , T id K
     K -> lambda
     C -> B C
     C -> lambda
     P -> B P
     P -> F P
     P -> lambda
}
```

1.2. First y Follow

```
First(E)={!, id, (, ent, cad} <- First(R)
First(R)={id, (, ent, cad} <- First(U)
First(RR)={+, lambda}
First(U)={id, (, ent, cad} <- First(V)
First(UU)={==, lambda}
First(V)={id, (, ent, cad}
First(VV)={(, lambda}
First(S)={id, print, input, return}
First(SS)={%, =, (}
First(L)={!, id, (, ent, cad, lambda} <- First(E)
First(Q)={‘,’ , lambda}
First(X)={!, id, (, ent, cad, lambda} <- First(E)
First(B)={switch, let, if, id, print, input, return} <- First(S)
First(O)={case, default, lambda}
First(D)= { break, lambda}
First(T)={int, boolean, string}
First(F)={function}
First(H)={function, lambda} <- First(T)
First(A)={function, lambda} <- First(T)
First(K)={‘,’ , lambda}
First(C)={switch, let, if, id, print, input, return, lambda} <- First(B)
First(P)={function, switch, let, if, id, print, input, return, lambda} <- First(B, F)

Follow(E)={), ;, ‘,’, :} <- First(Q) && Follow(L,  X)
Follow(R)={), ;, ‘,’, :} <- Follow(E, RR)
Follow(RR)={), ;, ‘,’, :} <- Follow(R)
Follow(U)={+, ), ;, ‘,’, :} <- First(RR), Follow(UU)
Follow(UU)={+, ), ;, ‘,’, :} <- Follow(U)
```

```
Follow(V)={==}  <- First(UU)
Follow(VV)={==} <- Follow(V)
Follow(S)={function, switch, let, if, id, print, input, return} <- Follow(B)
Follow(SS)={function, switch, let, if, id, print, input, return} <- Follow(S)
Follow(L)={ ) }
Follow(Q)={ ) } <- Follow(L)
Follow(X)={ ; }
Follow(B)={function, switch, let, if, id, print, input, return} <- First(C, P)
Follow(O)={ } }
Follow(D)={case, default} <- First(O)
Follow(T)={;, id, (} <- Follow(H)
Follow(F)={break, $} <- Follow(P)
Follow(H)={ ( }
Follow(A)={ ) }
Follow(K)={ ) } <- Follow(A)
Follow(C)={ } }
Follow(P)={break,  $} <- First(D)
```

1.3. Demostración LL1

Para las 50 reglas de producción, hallamos que para cada No Terminal que tenga más de una producción en la gramática:
1. No exista ningún terminal se deriven de los No Terminales y sea la primera aparición (la intersección de sus firsts sea vacía).
2. Si un No Terminal se puede derivar a Lambda, el terminal producido por la regla no puede estar contenido en su follow.

Los no terminales que contienen dos o más producciones son: E, RR, UU, V, VV, S, SS, L, Q, X, B, O, D, T, H, A, K, C, P.

E:
E ->! R
E ->R First(!R) $\cap First(R) = \{!\} \cap \{id, (, ent, cad\} = \emptyset$

RR:
RR ->+ R
RR ->λ First(+) $\cap Follow(RR) = \{+\} \cap \{), ;, ', :\} = \emptyset$

UU:
UU ->== U
UU ->λ First(== U) $\cap Follow(UU) = \{==\} \cap \{+,), ;, ', :\} = \emptyset$

V:
V ->id VV
V ->(E)
V ->constEnt
V ->cadena First(id VV) $\cap First((E)) \cap First(constEnt) \cap First(cadena) = \emptyset$

VV:

VV ->(L)
VV ->λ First((L)) $\cap Follow(VV) = \{(\} \cap \{+\} = \emptyset$

S:

S ->id SS
S ->print R ;
S ->input id ;
S ->return X ;

First(id SS) $\cap First(printR) \cap First(inputid;) \cap First(returnX;) = \{id\} \cap \{print\} \cap \{input\} \cap \{return\} = \emptyset$

L:

L ->E Q
L ->λ First(E) $\cap Follow(L) = \{!, id, (, constEnt, cadena\} \cap \{)\} = \emptyset$

Q:

Q ->, E Q
Q ->λ First(, E Q) $\cap Follow(Q) = \{, \} \cap \{)\} = \emptyset$

X:

X ->E
E ->λ First(E) $\cap Follow(X) = \{!, id, (, constEnt, cadena\} \cap \{;\} = \emptyset$

B:

B ->switch (E) $\{O\}$
$B- > if(E)S$
$B- > letidT;$
$B- > S$ $First(switch(E)\{O\}) \cap First(if(E)S) \cap First(letidT) \cap First(S) = \emptyset$

O:

O ->case E : P D O
O ->default: P D O
O ->λ

First(case E : P D O) $\cap First(default : PDO) = \{case\} \cap \{default\} = \emptyset$
$First(caseE : PDO) \cap Follow(O) = \{case\} \cap \{\}\} = \emptyset$
$First(defaultE : PDO) \cap Follow(O) = \{default\} \cap \{\}\} = \emptyset$

D:

D ->break ;
D ->λ First(break ;) $\cap Follow(D) = \{break\} \cap \{case, default\} = \emptyset$

T:

T ->int
T ->boolean
T ->string First(int) $\cap First(boolean) \cap First(string) = \emptyset$

H:

H ->T

H ->λ First(T) $\cap Follow(H) = \{int, boolean, string\} \cap \{(\} = \emptyset$

A:

A ->T id K

A ->λ First(T id K) $\cap Follow(A) = \{int, boolean, string\} \cap \{(\} = \emptyset$

K:

K ->, T id K

K ->λ First(, T id K) $\cap Follow(K) = \{, \} \cap \{)\} = \emptyset$

C:

C ->B C

C ->λ

First(B C) $\cap Follow(C) = \{switch, let, if, id, print, input, return\} \cap \{\}\} = \emptyset$

P:

P ->B P

P ->F P

P ->λ

First(B P) $\cap First(FP) = \{switch, let, if, id, print, input, return\} \cap \{function\} = \emptyset$
$First(BP) \cap Follow(P) = \{switch, let, if, id, print, input, return\} \cap \{break, \$\} = \emptyset$
$First(FP) \cap Follow(P) = \{function\} \cap \{break, \$\} = \emptyset$

1.4. Tabla de Análisis

Utilizamos la herramienta Sistema Generador de Gramáticas LL1, donde introdujimos la gramática factorizada, sin recursividad por la izquierda, y nos produjo la siguiente tabla, que le indica al Analizador Sintáctico que reglas aplicar según los tokens que reciba.

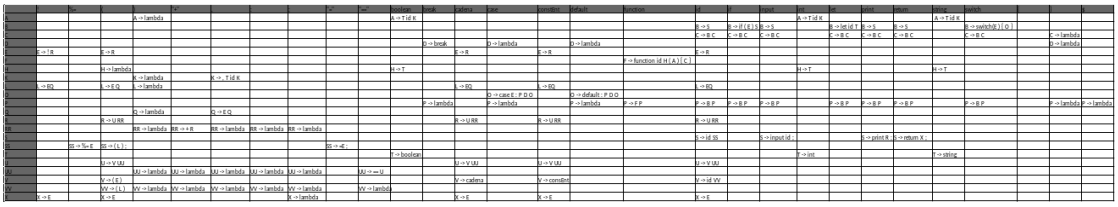


Figura 1: Tabla de Análisis de LL1 (Click para una hoja de cálculo con mayor resolución).

A. Casos de Prueba

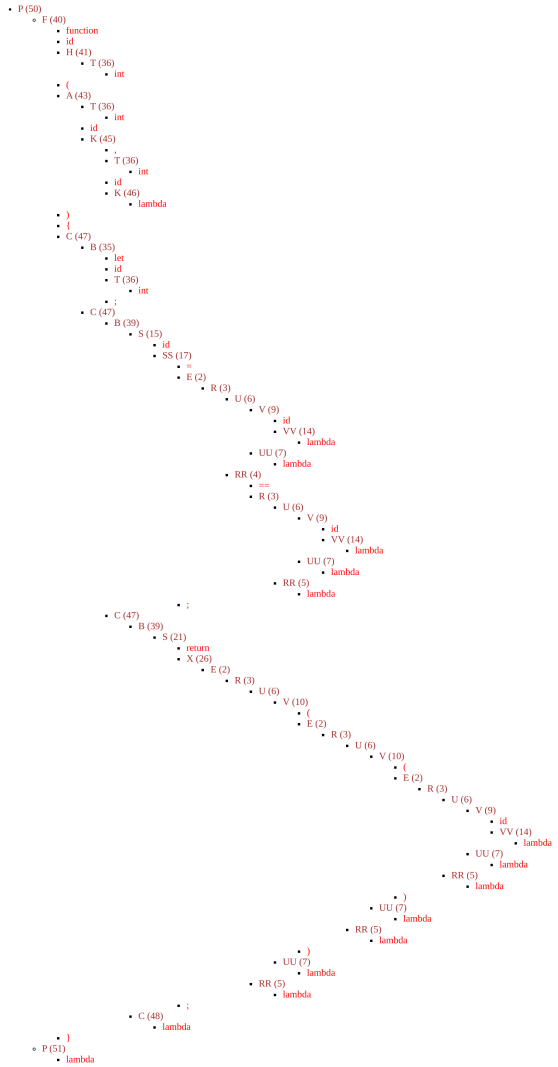
A.1. Prueba Funcional 1

Código fuente:

```
Texto fuente:
function operacion int(int num2,int num1)
{
  let  res int;
  res=num1+num2;
  return ((res));
}
```

Fichero de Parse y Árbol Sintáctico:

```
Descendente 50 40 41 36 43 36 45 36 46 47 35 36 47 39 15 17 2 3 6 9 14 7 4 3 6 9
     14 7 5 47 39 21 26 2 3 6 10 2 3 6 10 2 3 6 9 14 7 5 7 5 7 5 48 51
```



A.2. Prueba Funcional 2

Código fuente:

```
let a    int   ;
let b     int  ;
let number int ;
print  "Introduce el primer operando" ;
input a;
```

Fichero de Parse y Árbol sintáctico:

```
Descendente 49 35 36 49 35 36 49 35 36 49 39 19 3 6 12 7 5 49 39 20 51
```

- P (49)
  - B (35)
    - let
    - id
    - T (36)
      - int
    - ;
  - P (49)
    - B (35)
      - let
      - id
      - T (36)
        - int
      - ;
    - P (49)
      - B (35)
        - let
        - id
        - T (36)
          - int
        - ;
      - P (49)
        - B (39)
          - S (19)
            - print
            - R (3)
              - U (6)
                - V (12)
                  - cad
                - UU (7)
                  - lambda
              - RR (5)
                - lambda
            - ;
        - P (49)
          - B (39)
            - S (20)
              - input
              - id
              - ;
          - P (51)
            - lambda

A.3. Prueba Funcional 3

Código fuente:

```
  switch (int1){
  case 0: logico1 = int1 == int2;break;
  case 8888: print(0);
  case 3333: logico2="";
  }
```

Fichero de Parse y Árbol sintáctico:

```
Descendente 49 28 2 3 6 9 14 7 5 30 2 3 6 11 7 5 49 39 15 17 2 3 6 9 14 8 6 9 14 7
     5 51 33 30 2 3 6 11 7 5 49 39 19 3 6 10 2 3 6 11 7 5 7 5 51 34 30 2 3 6 11 7
    5 49 39 15 17 2 3 6 12 7 5 51 34 32 51
```

- P (49)
  - B (28)
    - switch
    - (
    - E (2)
      - R (3)
        - U (6)
          - V (9)
            - id
            - VV (14)
              - lambda
          - UU (7)
            - lambda
        - RR (5)
          - lambda
    -)
    - {
    - O (30)
      - case
      - E (2)
        - R (3)
          - U (6)
            - V (11)
              - ent
            - UU (7)
              - lambda
          - RR (5)
            - lambda
      - :
      - P (49)
        - B (39)
          - S (15)
            - id
            - SS (17)
              - =
              - E (2)
                - R (3)
                  - U (6)
                    - V (9)
                      - id
                      - VV (14)
                        - lambda
                    - UU (8)
                      - ==
                      - U (6)
                        - V (9)
                          - id
                          - VV (14)
                            - lambda
                        - UU (7)
                          - lambda
                  - RR (5)
                    - lambda
              - ;
        - P (51)
          - lambda
      - D (33)
        - break
        - ;
      - O (30)
        - case
        - E (2)
          - R (3)
            - U (6)
              - V (11)
                - ent
              - UU (7)
                - lambda
            - RR (5)
              - lambda
        - :
        - P (49)
          - B (39)
            - S (19)
              - print
              - R (3)
                - U (6)
                  - V (10)
                    - (
                    - E (2)
                      - R (3)
                        - U (6)
                          - V (11)
                            - ent
                          - UU (7)
                            - lambda
                        - RR (5)
                          - lambda
                    -)
                  - UU (7)
                    - lambda
              - RR (5)
                - lambda
              - ;
          - P (51)
            - lambda
        - D (34)
          - lambda
        - O (30)
          - case
          - E (2)
            - R (3)
              - U (6)
                - V (11)
                  - ent
                - UU (7)
                  - lambda
              - RR (5)
                - lambda
          - :
          - P (49)
            - B (39)
              - S (15)
                - id
                - SS (17)
                  - =
                  - E (2)
                    - R (3)
                      - U (6)
                        - V (12)
                          - cad
                        - UU (7)
                          - lambda
                      - RR (5)
                        - lambda
                  - ;
            - P (51)
              - lambda
          - D (34)
            - lambda
          - O (32)
            - lambda
    - }
  - P (51)
    - lambda

A.4. Prueba No Funcional 1

Código fuente:

```
function operacion int(int num2,int num1)
{
  let  res int;
  res=let num1 bool+num2;
  return ((res));
}
```

En este caso de prueba vemos que al no poderse declarar una variable dentro de una asignación, a la hora de llegar al apartado de expresiones de la gramática, no existe una regla que permita ir a let, se para la ejecución y imprime por pantalla:

Error Sintáctico: No existe regla para M[E, <palabraReservada, let>]

A.5. Prueba No Funcional 2

Código fuente:

```
function operacion2 int(int num1)
{
  let  res int;
  res=num2+num2;
  return ((res));
```

En este caso de prueba vemos que al no cerrar una llave al acabar el programa, la pila no se vacia al final del programa y finaliza la ejecución.

Error Sintáctico: No existe regla para M[C, <$, >]

A.6. Prueba No Funcional 3

Código fuente:

```
function doble int(int num1)
{
  let  res int;
  res=num1+num1;
  return (res);
}
function int int(int num2)
{
    let int2 int;
    return num2;
}
```

En este caso de prueba vemos que al poner una palabra reservada como nombre de una función envia un error, ya que el token que se esperaba era un id, sin embargo, al recibir el token de palabra reservada, no coincide con el de la cima de la pila y acaba la ejecución.

Error Sintáctico: El terminal de la cima de la pila ïd"no coincide con el token <palabra-Reservada, int>enviado por el AnLex