

# MANUAL DE USO DEL PROCESADOR DE BOREAL CON ANÁLISIS SINTÁCTICO DESCENDENTE CON TABLAS

## Introducción

El presente manual pretende aportar los fundamentos para ayudar en la comprensión, modificación y personalización del Analizador del Procesador de lenguaje Boreal de cara a desarrollar la práctica de Traductores de Lenguajes.

En el Procesador de Boreal se ha utilizado la librería del gestor de Tablas de Símbolos: TS\_Gestor. Por tanto, deberá consultarse su correspondiente manual en la web para saber cómo utilizar las Tablas de Símbolos.

Tras una introducción general del Procesador, se comentan algunos detalles sobre las acciones semánticas del Analizador Semántico que pueden ser de utilidad para incorporar el Generador de Código Intermedio.

## Procesador de Boreal

El Procesador de Boreal incluye un **Analizador Léxico** completo para todo el lenguaje. Cada grupo de prácticas tendrá solamente algunas de las opciones del lenguaje, pero se recomienda no modificar el Analizador Léxico (aunque reconozca más elementos de los necesarios para el grupo de prácticas). Este Analizador Léxico funciona como un servicio que proporciona al Analizador Sintáctico el siguiente *token* disponible en el fichero de entrada, además de introducir en la Tabla de Símbolos correspondiente todos los identificadores que aparezcan en el programa fuente.

El Procesador también incluye un **Analizador Sintáctico Descendente por Tablas LL(1)** (si se desea utilizar otro tipo de analizador Sintáctico, en la web de Traductores de Lenguajes hay más disponibles). El Analizador Sintáctico es para el lenguaje Boreal completo. De nuevo, se recomienda no modificar el Analizador Sintáctico (aunque reconozca más declaraciones, sentencias y expresiones de las necesarias para el grupo de prácticas). El Anexo 1 contiene la gramática utilizada para construir el Analizador Sintáctico.

Integrado con el Analizador Sintáctico se dispone de un **Analizador Semántico** correspondiente al lenguaje completo. Igual que en los casos anteriores, se recomienda no eliminar nada, aunque no se corresponda con las opciones de la práctica asignada al grupo. La gramática del Analizador permite el uso de acciones semánticas intercaladas dentro de la Traducción Dirigida por la Sintaxis, tanto para el Analizador Semántico como para el Generador de Código Intermedio. El manejo de los atributos será mediante atributos de tipo sintetizado. El Anexo 2 muestra el Esquema de Traducción utilizado para desarrollar el Analizador Semántico.

Juntamente con los tres módulos de Análisis, se dispone de una **Tabla de Símbolos** completa, que permite almacenar toda la información necesaria sobre todos los identificadores de Boreal (además de las palabras reservadas). Los analizadores completarán estas Tablas de Símbolos (Global y Locales) con la información obtenida por el Analizador Léxico y por el Analizador Semántico. Por tanto, aunque la información es completa hasta ese punto, podría ser necesario incluir nueva información necesaria para la fase de síntesis del Compilador. Los detalles sobre este módulo se muestran en la página web de Traductores de Lenguajes.

Finalmente, se dispone de un sencillo **Gestor de Errores** que, en función del error enviado por el resto de los módulos, informarán al usuario de los posibles errores detectados durante el análisis del programa fuente.

Para la construcción del Compilador, deberá añadirse el **Generador de Código Intermedio** y el **Generador de Código Objeto**. El primero deberá integrarse junto al Analizador Semántico existente, para que funcione con el Analizador Léxico y el Analizador Sintáctico. Por ello, en los siguientes apartados de este manual, se darán algunas indicaciones sobre cómo incorporar al Analizador Semántico el Esquema de Traducción del

Generador de Código Intermedio. El Generador de Código se añadirá como un nuevo módulo que reciba los cuartetos del Generador de Código Intermedio para traducirlos directamente a ensamblador.

## Analizador Sintáctico

### Gramática Modificada

El Analizador Sintáctico utiliza una gramática LL(1) modificada, cargada desde un fichero HTML o CSV. El fichero contiene la tabla de análisis sintáctico LL donde para cada par (símbolo no terminal, *token* de entrada) se especifica la producción que debe aplicarse.

Esta gramática es equivalente a la gramática del analizador sintáctico, pero incluye las acciones semánticas como un símbolo en la producción. Esto, permite al analizador saber dónde se debe ejecutar cada acción semántica. De esta manera se puede ejecutar una acción semántica en cualquier punto de la producción. En la gramática modificada se usa el nombre del *token* generado por el analizador léxico (“MAS”, por ejemplo).

El nombre de cada acción semántica debe comenzar por “\_acc” y debe ir seguido del número de la producción a la que corresponde esa acción semántica y, opcionalmente, de un subrayado y un número secuencial que indica el orden de la acción semántica cuando hay varias en una regla. Por tanto, las acciones semánticas tienen la estructura “\_accXX” o “\_accXX\_YY”. Además, deberá añadirse una regla lambda por cada una de las acciones así definidas. Todas estas acciones semánticas deberán corresponder a los grupos de acciones semánticas establecidas por el Analizador Semántico y el Generador de Código Intermedio.

Una vez obtenida la gramática ampliada, se puede usar la herramienta SDGLL (proporcionada en la sección de herramientas de la web de Procesadores de Lenguajes) para generar la tabla LL y exportar el fichero HTML necesario para el Procesador.

## Analizador Semántico

### Estructura de la Clase ASem

#### Variables de la Clase

La clase que representa el Analizador Semántico contiene las siguientes variables de clase:

```
public class ASem {
    public static boolean tsGlobal;                // Indica si existe TS global
    private static boolean zonaDeclaracion;        // Indica si se está en zona de declaraciones
    private static Integer despGlobal, despLocal;  // Desplazamientos de Las variables
    private static Integer numEtiquetas;           // Número de etiquetas generadas
                                                    // Pila auxiliar para la gestión atributos semánticos
    static Stack<ItemPila> stackAux = ASint.stackAux;
    ...
}
```

- **tsGlobal**: Define el estado de la tabla de símbolos, almacenando *true* si la tabla global es la activa o *false* en caso contrario.
- **zonaDeclaracion**: Define el estado de la zona de declaración, almacenando *true* si se está en una declaración, o *false* en caso contrario.
- **despGlobal**, **despLocal**: Acumula los desplazamientos en la tabla de símbolos global y local, respectivamente.
- **numEtiquetas**: Contador de las etiquetas asignadas.
- **stackAux**: Estructura compartida con el analizador sintáctico que permite acceder a los símbolos ya procesados para obtener los atributos necesarios para realizar las acciones semánticas.

## Inicialización

Para la inicialización del Analizador Semántico se debe llamar a la función `iniciarASem()`. Esta función se encarga de la inicializar las variables globales y la declaración de los atributos de la Tabla de Símbolos mediante una llamada a `iniciarTS()`.

```
public static void iniciarASem () {
    tsGlobal= false;
    zonaDeclaracion= false;
    despGlobal= 0;
    despLocal= 0;
    numEtiq= 0;
    iniciarTS();
}

private static void iniciarTS () {
    if (debug) Procesador.gestorTS.activarDebug();
    Procesador.gestorTS.createAtributo("desplazamiento",
        DescripcionAtributo.DIR, TipoDatoAtributo.ENTERO);
    Procesador.gestorTS.createAtributo("etiqueta",
        DescripcionAtributo.ETIQUETA, TipoDatoAtributo.CADENA);
    Procesador.gestorTS.createAtributo("pasoParametros",
        DescripcionAtributo.MODO_PARAM, TipoDatoAtributo.LISTA);
    Procesador.gestorTS.createAtributo("tipoParametros",
        DescripcionAtributo.TIPO_PARAM, TipoDatoAtributo.LISTA);
    Procesador.gestorTS.createAtributo("tipoRetorno",
        DescripcionAtributo.TIPO_RET, TipoDatoAtributo.CADENA);
    Procesador.gestorTS.createAtributo("numParametro",
        DescripcionAtributo.NUM_PARAM, TipoDatoAtributo.ENTERO);
    Procesador.gestorTS.createAtributo("modoParametro",
        DescripcionAtributo.PARAM, TipoDatoAtributo.ENTERO);
}
```

## Diseño de una Acción Semántica

### Atributos

Una acción semántica de una regla se implementa mediante una función que usa los atributos obtenidos de la pila `StackAux`. Estos atributos son una instancia de la clase `Atributos`.

La clase `Atributos` contiene las variables y métodos para manejar los atributos de cada símbolo gramatical, como posición (`pos`), tipo (`tipo`), cantidad de sentencias `exit` (`exit`), tamaño del tipo (`ancho`), lexema de una constante cadena (`lex`), etc.

El acceso y modificación de cada atributo se realiza mediante las correspondientes funciones `get` y `set` de la clase `Atributos`.

```
public class Atributos {
    private Integer pos; // Atributo posición en La Tabla de Símbolos
    private String tipo; // Atributo tipo del símbolo
    private Integer exit; // Atributo para contar el número de instrucciones exit
    private String ret; // Atributo para el tipo de retorno
    private Integer ancho; // Atributo para el tamaño de un tipo de datos
    private Integer longs; // Atributo para calcular la longitud de una lista
    private String referencia; // Atributo que indica si un parámetro es por referencia
    private String etiqueta; // Atributo para guardar una etiqueta
    private Integer val; // Atributo para guardar el valor numérico del símbolo
    private String lex; // Atributo para guardar el valor cadena del símbolo
    private Integer program_count; // Atributo para contar cuántos programas principales hay
    private Boolean asig; // Atributo para determinar si es asignación o llamada a funcion

    // Constructor que inicializa todas las variables de la clase
    public Atributos() {
        this.pos = null;
        this.tipo = null;
        this.exit = null;
    }
}
```

```

        this.ret = null;
        this.ancho = null;
        this.longs = null;
        this.referencia = null;
        this.etiqueta = null;
        this.val = null;
        this.lex = null;
        this.program_count = null;
        this.asig = null;
    }
    // Funciones get y set para las variables de la clase:
    public void setPos(Integer pos) { this.pos = pos; }
    public Integer getPos() {
        if (pos == null) return 0;
        return pos;
    }

    public void setTipo(String tipo) { this.tipo = tipo; }
    public String getTipo() {
        if (tipo == null) return "";
        return tipo;
    }

    public void setExit(Integer exit) { this.exit = exit; }
    public Integer getExit() {
        if (exit == null) return -1;
        return exit;
    }

    public void setRet(String ret) { this.ret = ret; }
    public String getRet() {
        if (ret == null) return "";
        return ret;
    }

    public void setAncho(Integer ancho) { this.ancho = ancho; }
    public Integer getAncho() {
        if (ancho == null) return -1;
        return ancho;
    }

    public void setLong(Integer longs) { this.longs = longs; }
    public Integer getLong() {
        if (longs == null) return -1;
        return longs;
    }

    public void setReferencia(String ref) { this.referencia = ref; }
    public String getReferencia() {
        if (referencia == null) return "";
        return referencia;
    }

    public void setEtiqueta(String et) { this.etiqueta = et; }
    public String getEtiqueta() {
        if (etiqueta == null) return "";
        return etiqueta;
    }

    public void setVal(Integer val) { this.val = val; }
    public Integer getVal() {
        if (val == null) return 32768; //Boreal solo admite valores hasta 32767
        return val;
    }

    public void setLex(String lex) { this.lex = lex; }
    public String getLex() {
        if (lex == null) return "";
        return lex;
    }
}

```

```

    public void setProgramCount(Integer i) { this.program_count = i; }
    public int getProgramCount() {
        if (program_count == null) return 0;
        return this.program_count;
    }

    public void setAsig(Boolean asig) { this.asig = asig; }
    public Boolean getAsig() {
        if (asig == null) return false;
        return asig;
    }
}

```

Si se desea añadir un nuevo atributo, se debe crear una nueva variable de clase del tipo deseado, inicializarla en el constructor y definir las respectivas funciones set y get de dicho nuevo atributo.

Por ejemplo, si fuera necesario crear un nuevo atributo (ficticio) para contabilizar la cantidad de bloques *begin-end*, se podría hacer de la siguiente manera:

```

class Atributos {
    ...
    private Integer begincont;                                // Nuevo atributo

    public Atributos() {
        ...
        this.begincont = null;                                // Inicialización del nuevo atributo en el constructor
    }
    ...
    // get y set para el nuevo atributo
    public void setBeginCont(Integer begincont) { this.begincont = begincont; }
    public Integer getBeginCont() { return this.begincont; }
}

```

## StackAux

StackAux es una pila de elementos *ItemPila*, compartida entre el analizador sintáctico y el analizador semántico. El analizado sintáctico añade los símbolos terminales y no terminales que se han procesado y el analizador semántico realiza las acciones semánticas con los atributos de estos. Una vez realizada la acción semántica correspondiente al final de una producción, los elimina de la pila.

Para acceder a los valores de los atributos en la pila, hay que tener en cuenta que los símbolos están en el orden inverso al que aparecen en la regla. Esto es porque se insertan en la pila a medida que son procesados por el analizador sintáctico. Por ejemplo, tras ejecutar la acción semántica 24, que corresponde a la producción *Bloque → begin C end*, los elementos se encuentran en *stackAux* en el siguiente orden:

- end
- C
- begin
- Bloque

## Acción Semántica

Por lo general, las acciones semánticas se desarrollan en tres etapas:

1. Obtención de los atributos de la pila de atributos:

```

// AA -> ; X id : T AA1
Atributos atrAA1 = peekAuxAtributos(0);                    // AA1
Atributos atrT = peekAuxAtributos(1);                      // T
Atributos atrID = peekAuxAtributos(3);                     // id
Atributos atrX = peekAuxAtributos(4);                      // X
Atributos atrAA = peekAuxAtributos(6);                     // AA

```

- La función `peekAuxAttributes(int nBuscar)` se usa para obtener la información de los símbolos de la pila `StackAux`. El parámetro `nBuscar` es el índice que indica la posición relativa desde la cima de la pila al elemento que se quiere consultar.
  - Si la regla 1 tiene la forma  $P \rightarrow D a K E$ , siendo 'a' un símbolo terminal y 'D', 'K' y 'E' símbolos no terminales, los atributos de 'P', 'D', 'K' y 'E' se almacenan en los índices 4, 3, 1 y 0, respectivamente, de la lista. Si 'a' tuviera también atributos, estarían almacenados en el índice 2.
  - Los símbolos terminales de ID, ENTERO y CADENA contienen atributos correspondientes a la posición en la tabla de símbolos, valor numérico y valor léxico respectivamente.
- 2. Desarrollo de la acción semántica. Se debe programar la acción semántica, que hará uso de los atributos obtenidos y podrá definir los valores de algún atributo.
- 3. En caso de que sea la última acción semántica de la producción, hay que eliminar los elementos correspondientes al consecuente de la pila. Para ello se usa el método `popAuxLast(int nBorrar)` se encarga de eliminar de la pila auxiliar los últimos elementos utilizados por una producción, una vez que la acción semántica final de dicha producción ha sido ejecutada. El parámetro `nBorrar` indica el número de elementos que se quieren eliminar de la pila, y debe coincidir con el número de elementos en el consecuente de la producción.
  - En caso de que sea una acción semántica intercalada en la producción, NO se debe eliminar ningún elemento.

### Modificación de una Acción Semántica

Modificar una acción semántica de una de las reglas para añadir las acciones semánticas del Generador de Código Intermedio implica cambiar directamente la función de dicha regla correspondiente a la acción semántica que se quiere ampliar.

Por ejemplo, si se quisiera realizar una modificación (ficticia) sobre la regla 24 (`Bloque → begin C end`), se parte de la acción semántica actual:

```
Bloque → begin C end { Bloque.tipo:= C.tipo
                       Bloque.exit:= C.exit
                       Bloque.tipoRet:= C.tipoRet }
```

El código correspondiente a la acción semántica de la regla 24 es:

```
public static void acc24() {                                     // Acción semántica de La regla 24
    // Bloque → begin C end
    // Se obtienen los atributos que se van a usar/modificar:
    Atributos atrC = peekAuxAttributes(1);                       // C
    Atributos atrBloque = peekAuxAttributes(3);                   // Bloque
    // Implementación de las acciones semánticas:
    atrBloque.setTipo(atrC.getTipo());                           // Bloque.tipo:= C.tipo
    atrBloque.setExit(atrC.getExit());                           // Bloque.exit:= C.exit
    atrBloque.setRet(atrC.getRet());                             // Bloque.tipoRet:= C.tipoRet

    popAuxLast(3);                                               // Se eliminan los últimos 3 elementos de la pila
}
```

Ahora, si se supone que la nueva acción semántica que se quiere realizar para dicha regla es:

```
Bloque → begin C end { Bloque.tipo:= C.tipo
                       Bloque.exit:= C.exit
                       Bloque.tipoRet:= C.tipoRet
                       Bloque.BeginCont:= C.BeginCont + 1 }
```

El nuevo código de la acción semántica de la regla 24 sería:

```

public static void acc24() {                                     // Acción semántica de la regla 24
    // Bloque → begin C end
    // Se obtienen Los atributos que se van a usar/modificar:
    Atributos atrC = peekAuxAtributos(1);                      // C
    Atributos atrBloque = peekAuxAtributos(3);                 // Bloque
    // Implementación de las cciones semánticas:
    atrBloque.setTipo(atrC.getTipo());                          // Bloque.tipo:= C.tipo
    atrBloque.setExit(atrC.getExit());                          // Bloque.exit:= C.exit
    atrBloque.setRet(atrC.getRet());                            // Bloque.tipoRet:= C.tipoRet
    atrBloque.setBeginCont(atrC.getBeginCont() + 1)           // Bloque.BeginCont:= C.BeginCont + 1
    popAuxLast(3);                                             // Se eliminan Los últimos 3 elementos de la pila
}

```

## Creación de una Nueva Acción Semántica

Si se desea crear una nueva Acción Semántica en un punto en el que no existe ninguna, a mitad de una producción, primero se debe actualizar la gramática modificada. La gramática se encuentra en el Anexo 3 (y en la carpeta *resources/gramatica\_modificada.ll1*). Todas las acciones semánticas que se añadan deben tener la forma “\_accXX” o “\_accXX\_YY”, siendo XX el número de la producción en la que se encuentran e YY el orden de la acción dentro de la producción. Para añadir una acción semántica en el fichero de la gramática ampliada hay que:

- Modificar la producción donde se quiera añadir la acción semántica para incluir \_accXX o \_accXX\_YY en el punto deseado
- Añadir la producción \_accXX → lambda o \_accXX\_YY → lambda al final de la lista de producciones
- Añadir el nombre de la acción semántica al conjunto de NoTerminales.

Por ejemplo, si se quiere incluir una nueva acción semántica (\_acc33\_0) intercalada en la regla 33, B → loop C \_acc33\_0 end PYC \_acc33, se deberá modificar el fichero de la gramática:

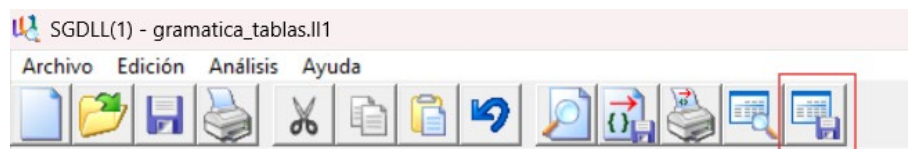
```

NoTerminales = {... _acc33_0 ...}                             // Añadir La acción a NoTerminales
...
B -> loop C _acc33_0 end PYC _acc33                             // Modificar La regla
...
_acc33_0 -> lambda                                             // Añadir La regla: acción -> λ

```

El siguiente paso consiste en obtener el fichero HTML correspondiente a esta gramática modificada. Para ello se usa la herramienta Sistema Comprobador de Gramáticas LL(1) (SDGLL1) que está disponible en la página web de Procesadores de Lenguajes, en el apartado de herramientas.

Se carga en el sistema la nueva gramática modificada y se pulsa sobre “guardar tabla sintáctica”. El fichero obtenido se tendrá que colocar en la carpeta *resources*, sustituyendo al fichero *tabla\_acc.htm*.



Después, se debe programar la nueva acción semántica en *ASem.java* con el mismo nombre que la acción semántica, pero sin el carácter de subrayado del principio.

```

public static void acc33_0() {
    ...
}

```

## Tipos de Datos

El Analizador Semántico maneja una serie de tipos asociados a los distintos símbolos gramaticales. Para ello, se utilizan cadenas con los nombres de los tipos escritos en minúsculas de la siguiente manera: “cadena”, “entero”, “lógico”, “función”, “procedimiento”, “tipo\_ok” y “tipo\_error”; y “vacío” para el tipo vacío.



## Gestor de Errores

El Gestor de Errores permite informar al usuario los errores detectados durante la compilación. Para el análisis sintáctico y semántico se han usado los errores flexibles. Se emplea la función `writeError` de la clase `GestorError` para imprimir un mensaje de error puntual. Tienen dos argumentos, el primero recoge el tipo de error y el segundo una pequeña descripción.

```
GestorError.writeError("Sintáctico", "Exit fuera de bucle detectado en Procedure");
```

Este código imprimirá: Error Sintáctico en la línea [1]: Exit fuera de bucle detectado en Procedure

También se pueden usar los errores predefinidos. Se crea una instancia en la clase `Accion`, se añade un valor en el enum `Acciones` y una entrada en un nuevo case con el mensaje de error en la función `printError()` de la clase `GestorError`. La llamada se realiza usando la función `setError`, a la que se le puede enviar información adicional como segundo parámetro. Esta forma es la recomendada si el mensaje de error se puede producir en distintas situaciones.

```
GestorError.setError(Acciones.eSem5_case_otherwise_error, "");
```

## Anexo 1: Gramática del Analizador Sintáctico

```
//// Conjunto de símbolos terminales
Terminales = { program ; id procedure function var boolean integer string ( : ) if then begin end else
while do repeat until loop for := to case of entero otherwise write writeln read return
exit when , or xor and = < > >= < <= + - * / mod ** not cadena true false in max min }

//// Conjunto de símbolos no terminales
NoTerminales = { P R PP PR PF D DD T A AA C B Bloque N O S LL L Q V W Y E F G H I J K Z X
Eprima Fprima Gprima Hprima Iprima Jprima Zprima Bcola Scola BlqElse }
//// Axioma
Axioma = P

//// Lista de producciones
Producciones = {
P -> D R
R -> PP R
R -> PR R
R -> PF R
R -> lambda
PP -> program id ; D Bloque ;
PR -> procedure id A ; D Bloque ;
PF -> function id A : T ; D Bloque ;
D -> var id : T ; DD
D -> lambda
DD -> id : T ; DD
DD -> lambda
T -> boolean
T -> integer
T -> string
A -> ( X id : T AA )
A -> lambda
AA -> ; X id : T AA
AA -> lambda
X -> var
X -> lambda
C -> B C
C -> lambda
Bloque -> begin C end
B -> S
B -> if E then Bcola
Bcola -> S
Bcola -> Bloque ; BlqElse
BlqElse -> else Bloque ;
BlqElse -> lambda
```



```

B -> while E do Bloque ;
B -> repeat C until E ;
B -> loop C end ;
B -> for id := E to E do Bloque ;
B -> case E of N O end ;
N -> entero : Bloque ; N
N -> lambda
O -> otherwise : Bloque ;
O -> lambda
S -> write LL ;
S -> writeln LL ;
S -> read ( V ) ;
S -> id Scola
Scola -> := E ;
Scola -> LL ;
S -> return Y ;
S -> exit when E ;
LL -> ( L )
LL -> lambda
L -> E Q
Q -> , E Q
Q -> lambda
V -> id W
W -> , id W
W -> lambda
Y -> E
Y -> lambda
E -> F Eprima
Eprima -> or F Eprima
Eprima -> xor F Eprima
Eprima -> lambda
F -> G Fprima
Fprima -> and G Fprima
Fprima -> lambda
G -> H Gprima
Gprima -> = H Gprima
Gprima -> <> H Gprima
Gprima -> > H Gprima
Gprima -> >= H Gprima
Gprima -> < H Gprima
Gprima -> <= H Gprima
Gprima -> lambda
H -> I Hprima
Hprima -> + I Hprima
Hprima -> - I Hprima
Hprima -> lambda
I -> J Iprima
Iprima -> * J Iprima
Iprima -> / J Iprima
Iprima -> mod J Iprima
Iprima -> lambda
J -> K Jprima
Jprima -> ** K Jprima
Jprima -> lambda
K -> not K
K -> + K
K -> - K
K -> Z
Z -> entero Zprima
Z -> cadena
Z -> true
Z -> false
Z -> id LL Zprima
Z -> ( E ) Zprima
Z -> max ( L )
Z -> min ( L )
Zprima -> in ( L )
Zprima -> lambda
}

```

## Anexo 2: Esquema de Traducción del Analizador Semántico

```

1.  P → {   TSG:= CreadTS ()
            TSActual:= TSG
            desp_global:=0
            zona_decl:= true
        }

    D R
        {   If (R.program ≠ 1) Then Error ("Debe haber 1 y solo 1 Program")
            DestruirTS (TSG)
        }

2.  R → PP R1 {R.program:= 1 + R1.program}
3.  R → PR R1 {R.program:= R1.program}
4.  R → PF R1 {R.program:= R1.program}
5.  R → λ      {R.program:= 0}

6.  PP → program id ; {   InsetarTipoTS (id.pos, vacío→vacío)
                            InsertarEtiquTS (id.pos, "main")
                            TSL:= CreadTS ()
                            TSActual:= TSL
                            despl_local:= 0
                        }
    D
    Bloque ; {   zona_decl:= false }
                {   if (Bloque.tipo = tipo_error)
                    then Error ("Error detectado en el cuerpo del Programa principal")
                    if(Bloque.tipoRet ≠ tipo_ok AND Bloque.tipoRet ≠ vacío)
                        then Error ("Programa Principal con instrucción de retorno no vacío")
                    if(Bloque.exit > 0)
                        then Error ("Exit fuera de bucle detectado en Programa Principal")
                    destruirTS (TSL)
                    TSActual:= TSG
                    zona_decl:= true
                }

7.  PR → procedure id {   TSL:= CreadTS ()
                            TSActual:= TSL
                            despl_local:= 0
                        }
    A ; {   InsetarTipoTS (id.pos, A.tipo→vacío)
            InsertarModoTS (id.pos, A.referencia)
            // A.referencia es un producto cartesiano de lógicos
            InsertarEtiquetaTS (id.pos, nuevaEt ())
        }
    D
    Bloque ; {   zona_decl:= false }
                {   if (Bloque.tipo = tipo_error)
                    then Error ("Error detectado en el cuerpo del Procedure")
                    if (Bloque.tipoRet ≠ tipo_ok AND Bloque.tipoRet ≠ vacío)
                        then Error ("Procedure con instrucción de retorno no vacío")
                    if (Bloque.exit > 0)
                        then Error ("Exit no puede situarse fuera del bucle en el Procedure")
                    DestruirTS (TSL)
                    TSActual:= TSG
                    zona_decl:= true
                }
    
```

```

8.  PF → function id  {  TSL:= CrearTS ()
                        TSGlobal:= TSL
                        Despl_local:= 0
                        }
      A:T;            {  InsertarTipoTS (id.pos, A.tipo→T.tipo)
                        InsertarModoTS (id.pos, A.referencia)
                        // A.referencia es un producto cartesiano de lógicos
                        InsertarEtiquetaTS (id.pos, nuevaEt ())
                        }
      D
      Bloque;         {  zona_decl:= false }
                        {  if (Bloque.tipo = tipo_error)
                        then Error ("Error detectado en el cuerpo de la función")
                        if (Bloque.tipoRet ≠ tipo_ok AND Bloque.tipoRet ≠ T.tipo)
                        then Error ("Función con retorno inválido")
                        if (Bloque.exit > 0)
                        then Error ("Exit no puede situarse fuera del bucle en la función")
                        DestruirTS (TSL)
                        TSGlobal:= TSG
                        zona_decl:= true
                        }
}

9.  D → var id : T; {  InsertarTipoTS (id.pos, T.tipo)
                        if (TSGlobal = TSG) then
                        {
                            InsertarDespTS (id.pos, despl_global)
                            despl_global:= despl_global + T.ancho
                        }
                        else
                        {
                            InsertarDespTS (id.pos, despl_local)
                            despl_local:= despl_local + T.ancho
                        }
                        }
}
      DD

10. D → λ           { }

11. DD → id : T;   {  InsertarTipoTS (id.pos, T.tipo)
                        if (TSGlobal = TSG) then
                        {
                            InsertarDespTS (id.pos, despl_global)
                            despl_global:= despl_global + T.ancho
                        }
                        else
                        {
                            InsertarDespTS (id.pos, despl_local)
                            despl_local:= despl_local + T.ancho
                        }
                        }
}
      DD

12. DD → λ         { }

13. T → boolean    {T.tipo:= lógico; T.ancho:= 1}
14. T → integer     {T.tipo:= entero; T.ancho:= 1}
15. T → string      {T.tipo:= cadena; T.ancho:= 64}

```

```

16. A → (X id : T {   InsertarTipoTS (id.pos, T.tipo)
                        InsertarDespTS (id.pos, despl_local)
                        if (X.referencia) then
                        {
                            InsertaAtributoPasoParametrosTS (id.pos, "referencia")
                            despl_local:= despl_local + 1
                        }
                        else
                        {
                            InsertaAtributoPasoParametrosTS (id.pos, "valor")
                            despl_local:= despl_local + T.ancho
                        }
                    }
AA) {
    if (AA.tipo ≠ vacío) then
    {
        A.tipo:= T.tipo x AA.tipo
        A.referencia:= X.referencia x AA.referencia // producto cartesiano de lógicos
    }
    else
    {
        A.tipo:= T.tipo
        A.referencia:= X.referencia
    }
    A.long:= 1 + AA.long
}

17. A → λ {   A.tipo:= vacío
                A.long:= 0
                A.referencia:= ∅
            }

18. AA → ; X id : T {   InsertarTipoTS (id.pos, T.tipo)
                        InsertarDespTS (id.pos, despl_local)
                        if (X.referencia) then
                        {
                            InsertaAtributoPasoParametrosTS (id.pos, "referencia")
                            despl_local:= despl_local + 1
                        }
                        else
                        {
                            InsertaAtributoPasoParametrosTS (id.pos, "valor")
                            despl_local:= despl_local + T.ancho
                        }
                    }
AA1 {
    if (AA1.tipo ≠ vacío) then
    {
        AA.tipo:= T.tipo x AA1.tipo
        AA.referencia := X.referencia x AA1.referencia
                                                // producto cartesiano de lógicos
    }
    else
    {
        AA.tipo:= T.tipo
        AA.referencia:= X.referencia
    }
    AA.long:= 1 + AA1.long
}

19. AA → λ {   AA.tipo:= vacío
                AA.long:= 0
                AA.referencia:= ∅
            }

20. X → var {X.referencia = true}
21. X → λ  {X.referencia = false}

```

```

22. C → B C1 {   C.tipo:= if (B.tipo = tipo_ok)
                    then C1.tipo
                    else tipo_error
    C.exit:= B.exit + C1.exit
    C.tipoRet:= if (B.tipoRet = C1.tipoRet)
                then B.tipoRet
                else if (B.tipoRet = tipo_ok)
                    then C1.tipoRet
                else if (C1.tipoRet = tipo_ok)
                    then B.tipoRet
                else tipo_error
                }
23. C → λ      {   C.tipo:= tipo_ok
                    C.exit:= 0
                    C.tipoRet:= tipo_ok
                }

24. Bloque → begin C end {   Bloque.tipo:= C.tipo
                              Bloque.exit:= C.exit
                              Bloque.tipoRet:= C.tipoRet
                              }

25. B → S      {   B.tipo:= S.tipo
                    B.exit:= S.exit
                    B.tipoRet:= S.tipoRet
                }

26. B → if E then Bcola {   B.tipo:= if (E.tipo = lógico)
                              then Bcola.tipo
                              else tipo_error
                              B.exit:= Bcola.exit
                              B.tipoRet:= Bcola.tipoRet
                              }
27. Bcola → S    {   Bcola.tipo:= S.tipo
                    Bcola.exit:= S.exit
                    Bcola.tipoRet:= S.tipoRet
                }
28. Bcola → Bloque ; BlqElse
    {   Bcola.tipo:= if(Bloque.tipo = tipo_ok)
                then BlqElse.tipo
                else tipo_error
        Bcola.exit:= Bloque.exit + BlqElse.exit
        Bcola.tipoRet:= if (Bloque.tipoRet = BlqElse.tipoRet)
                        then Bloque.tipoRet
                        else if (Bloque.tipoRet = tipo_ok)
                            then BlqElse.tipoRet
                        else if (BlqElse.tipoRet = tipo_ok)
                            then Bloque.tipoRet
                        else tipo_error
    }
29. BlqElse → else Bloque ; {   BlqElse.tipo:= Bloque.tipo
                              BlqElse.exit:= Bloque.exit
                              BlqElse.tipoRet:= Bloque.tipoRet   }
30. BlqElse → λ    {   BlqElse.tipo:= tipo_ok
                    BlqElse.exit:= 0
                    BlqElse.tipoRet:= tipo_ok }
31. B → while E do Bloque ;
    {   B.tipo:= if (E.tipo = lógico)
                then Bloque.tipo
                else tipo_error
        B.exit:= Bloque.exit
        B.tipoRet:= Bloque.tipoRet
    }

```

```

32. B → repeat C until E ;
    {   B.tipo:= if (E.tipo = lógico)
        then C.tipo
        else tipo_error
        B.exit:= C.exit
        B.tipoRet:= C.tipoRet
    }

33. B → loop C end ; {   if (C.exit ≠ 1)
    then Error (“dentro de loop debe haber 1 y solo 1 exit”)
    B.tipo:= C.tipo
    B.exit:= 0
    B.tipoRet:= C.tipoRet
}

34. B → for id := E1 to E2 do Bloque ;
    {   B.tipo:= if (E1.tipo=entero AND E2.tipo=entero AND BuscarTipoTS(id.pos) = entero)
        then Bloque.tipo
        else tipo_error
    B.exit:= Bloque.exit
    B.tipoRet:= Bloque.tipoRet
}

35. B → case E of N O end;
    {   B.tipo:= if (E.tipo = entero AND N.tipo = O.tipo = tipo_ok)
        then tipo_ok
        else tipo_error
    B.exit:= N.exit + O.exit
    B.tipoRet:= if (N.tipoRet = O.tipoRet)
        then N.tipoRet
        else if (N.tipoRet = tipo_ok)
            then O.tipoRet
        else if (O.tipoRet = tipo_ok)
            then N.tipoRet
        else tipo_error
    }

36. N → entero : Bloque ; N1
    {   N.tipo:= if (N1.tipo = tipo_ok)
        then Bloque.tipo
        else tipo_error
    N.exit:= Bloque.exit + N1.exit
    N.tipoRet:= if (Bloque.tipoRet = N1.tipoRet)
        then Bloque.tipoRet
        else if (Bloque.tipoRet = tipo_ok)
            then N1.tipoRet
        else if (N1.tipoRet = tipo_ok)
            then Bloque.tipoRet
        else tipo_error
    }

37. N → λ
    {   N.tipo:= tipo_ok
    N.exit:= 0
    N.tipoRet:= tipo_ok }

38. O → otherwise : Bloque ; {   O.tipo:= Bloque.tipo
    O.exit:= Bloque.exit
    O.tipoRet:= Bloque.tipoRet }

39. O → λ {   O.tipo:= tipo_ok
    O.exit:= 0
    O.tipoRet:= tipo_ok}

```

```

40. S → write LL; {   S.tipo:= tipo_ok
                      if (LL.tipo ≠ vacío) then
                      {
                        for i:= 1 to LL.long
                          if (LL.tipo[i] ≠ entero AND LL.tipo[i] ≠ cadena)
                            then S.tipo:= tipo_error
                        }
                      S.exit:= 0
                      S.tipoRet:= tipo_ok
                    }

41. S → writeln LL; { S.tipo:= tipo_ok
                      if (LL.tipo ≠ vacío) then
                      {
                        for i:= 1 to LL.long
                          if (LL.tipo[i] ≠ entero AND LL.tipo[i] ≠ cadena)
                            then S.tipo:= tipo_error
                        }
                      S.exit:= 0
                      S.tipoRet:= tipo_ok
                    }

42. S → read (V); {   S.tipo:= tipo_ok
                      for i:= 1 to V.long
                        if (V.tipo[i] ≠ entero AND V.tipo[i] ≠ cadena)
                          then S.tipo:= tipo_error
                      S.exit:= 0
                      S.tipoRet:= tipo_ok
                    }

43. S → id Scola {   id.tipo:= BuscaTipoTS (id.pos)
                      id.et:= BuscaEtiqTS (id.pos)
                      if (id.tipo = Scola.tipo AND Scola.asig) then
                      {
                        S.tipo:= tipo_ok
                      }
                      else if (id.tipo = Scola.tipo→vacío AND NOT Scola.asig AND id.et ≠ "main") then
                      {
                        S.tipo:= tipo_ok
                      }
                      else S.tipo:= tipo_error // Asignación con tipos distintos, Llamada a identificador que
                                                // no es procedimiento o identificador es program
                      S.exit:= 0
                      S.tipoRet:= tipo_ok
                    }

44. Scola → := E; {   Scola.tipo := E.tipo
                      Scola.long := 0
                      Scola.asig := true   }

45. Scola → LL; {    Scola.tipo := LL.tipo
                      Scola.long := LL.long
                      Scola.asig := false  }

46. S → return Y; {   S.tipo:= if (Y.tipo ≠ tipo _error)
                        then tipo_ok
                        else tipo_error
                      S.exit:= 0
                      S.tipoRet:= Y.tipo
                      // tipoRet puede ser vacío (para procedimientos), un tipo (para funciones) o tipo_ok (cuando no hay return)
                    }

47. S → exit when E; {   S.tipo:= if (E.tipo = lógico)
                        then tipo_ok
                        else tipo_error
                      S.exit:= 1
                      S.tipoRet:= tipo_ok
                    }

48. LL → (L) {   LL.tipo:= L.tipo
                  LL.long:= L.long }

```



```

49. LL → λ { LL.tipo:= vacío
               LL.long:= 0 }

50. L → E Q { L.tipo:= if (Q.tipo = vacío)
               {
                 L.tipo:= E.tipo
               }
               else
               {
                 L.tipo:= E.tipo x Q.tipo
               }
               L.long:= 1 + Q.long
             }

51. Q → , E Q1 { if (Q1.tipo = vacío) then
                   {
                     Q.tipo:= E.tipo
                   }
                   else
                   {
                     Q.tipo:= E.tipo x Q1.tipo
                   }
                   Q.long:= 1 + Q1.long
                 }

52. Q → λ { Q.tipo:= vacío
              Q.long:= 0 }

53. V → id W { if (W.tipo = vacío)
                {
                  V.tipo:= buscaTipoTS (id.pos)
                }
                else
                {
                  V.tipo:= buscaTipoTS (id.pos) x W.tipo
                }
                V.long:= 1 + W.long
              }

54. W → , id W1 { if (W1.tipo = vacío) then
                   {
                     W.tipo:= buscaTipoTS (id.pos)
                   }
                   else
                   {
                     W.tipo:= buscaTipoTS (id.pos) x W1.tipo
                   }
                   W.long:= 1 + W1.long
                 }

55. W → λ { W.tipo:= vacío
              W.long:= 0 }

56. Y → E { Y.tipo:= E.tipo }
57. Y → λ { Y.tipo:= vacío }

58. E → F Eprima { if (Eprima.tipo = vacío)
                     then
                     {
                       E.tipo:= F.tipo
                     }
                     else if (F.tipo = Eprima.tipo = lógico)
                     then
                     {
                       E.tipo:= lógico
                     }
                     else F.tipo:= tipo_error
                   }

```

```

59. Eprima  $\rightarrow$  or F Eprima1 {   if (Eprima1.tipo = vacío)
                                then if(F.tipo = lógico)
                                    then
                                        {
                                            Eprima.tipo:= lógico
                                        }
                                    else Eprima.tipo:= tipo_error
                                else if (F.tipo = Eprima1.tipo = lógico)
                                    then
                                        {
                                            E.prima.tipo:= lógico
                                        }
                                    else Eprima.tipo:= tipo_error
                                }
60. Eprima  $\rightarrow$  xor F Eprima1 {   if (Eprima1.tipo = vacío)
                                then if(F.tipo = lógico)
                                    then
                                        {
                                            Eprima.tipo:= lógico
                                        }
                                    else Eprima.tipo:= tipo_error
                                else if (F.tipo = Eprima1.tipo = lógico)
                                    then
                                        {
                                            E.prima.tipo:= lógico
                                        }
                                    else Eprima.tipo:= tipo_error
                                }
61. Eprima  $\rightarrow$   $\lambda$  {   Eprima.tipo:= vacío }
62. F  $\rightarrow$  G Fprima {   if (Fprima.tipo = vacío)
                        then
                            {
                                F.tipo:= G.tipo
                            }
                        else if (G.tipo = Fprima.tipo = lógico)
                            then
                                {
                                    F.tipo:= lógico
                                }
                        else F.tipo:= tipo_error
                    }
63. Fprima  $\rightarrow$  and G Fprima1 {   if (Fprima1.tipo = vacío)
                                then if (G.tipo = lógico)
                                    then
                                        {
                                            Fprima.tipo:= lógico
                                        }
                                    else Fprima.tipo:= tipo_error
                                else if (G.tipo = Fprima1.tipo = lógico)
                                    then
                                        {
                                            Fprima.tipo:= lógico
                                        }
                                    else Fprima.tipo:= tipo_error
                                }
64. Fprima  $\rightarrow$   $\lambda$  {   Fprima.tipo:= vacío }
65. G  $\rightarrow$  H Gprima {   if (Gprima.tipo = vacío)
                        then
                            {
                                G.tipo:= H.tipo
                            }
                        else if (H.tipo = entero AND Gprima.tipo = lógico)
                            then G.tipo:= lógico
                            else G.tipo:= tipo_error
                    }

```

```

66. Gprima  $\rightarrow$  = H Gprima1 { if (Gprima1.tipo = vacío)
                                then if (H.tipo = entero)
                                    then
                                        {
                                            Gprima.tipo:= lógico
                                        }
                                    else Gprima.tipo:= tipo_error
                                else Gprima.tipo:= tipo_error
                                }
67. Gprima  $\rightarrow$  <> H Gprima1 { if (Gprima1.tipo = vacío)
                                then if (H.tipo = entero)
                                    then
                                        {
                                            Gprima.tipo:= lógico
                                        }
                                    else Gprima.tipo:= tipo_error
                                else Gprima.tipo:= tipo_error
                                }
68. Gprima  $\rightarrow$  > H Gprima1 { if (Gprima1.tipo = vacío)
                                then if (H.tipo = entero)
                                    then
                                        {
                                            Gprima.tipo:= lógico
                                        }
                                    else Gprima.tipo:= tipo_error
                                else Gprima.tipo:= tipo_error
                                }
69. Gprima  $\rightarrow$  >= H Gprima1 { if (Gprima1.tipo = vacío)
                                then if (H.tipo = entero)
                                    then
                                        {
                                            Gprima.tipo:= lógico
                                        }
                                    else Gprima.tipo:= tipo_error
                                else Gprima.tipo:= tipo_error
                                }
70. Gprima  $\rightarrow$  < H Gprima1 { if (Gprima1.tipo = vacío)
                                then if (H.tipo = entero)
                                    then
                                        {
                                            Gprima.tipo:= lógico
                                        }
                                    else Gprima.tipo:= tipo_error
                                else Gprima.tipo:= tipo_error
                                }
71. Gprima  $\rightarrow$  <= H Gprima1 { if (Gprima1.tipo = vacío)
                                then if (H.tipo = entero)
                                    then
                                        {
                                            Gprima.tipo:= lógico
                                        }
                                    else Gprima.tipo:= tipo_error
                                else Gprima.tipo:= tipo_error
                                }
72. Gprima  $\rightarrow$   $\lambda$  {Gprima.tipo:= vacío}

```

```

73. H → I Hprima      {   if (Hprima.tipo = vacío)
                           then
                           {
                               H.tipo:= I.tipo
                           }
                           else if (I.tipo = Hprima.tipo = cadena)
                           then
                           {
                               H.tipo:= cadena
                           }
                           else if (I.tipo = entero AND Hprima.tipo = entero)
                           then
                           {
                               H.tipo:= entero
                           }
                           else H.tipo:= tipo_error
                           }

74. Hprima → + I Hprima1 {   if (Hprima1.tipo = vacío)
                               then if (I.tipo ∈ {entero, cadena})
                                   then
                                   {
                                       Hprima.tipo:= I.tipo
                                   }
                                   else Hprima.tipo:= tipo_error
                               else if (I.tipo = Hprima1.tipo ∈ {entero, cadena})
                               then
                               {
                                   Hprima.tipo:= I.tipo
                               }
                               else Hprima.tipo:= tipo_error
                               }

75. Hprima → - I Hprima1 {   if (Hprima1.tipo = vacío)
                               then if (I.tipo = entero)
                                   then
                                   {
                                       Hprima.tipo:= entero
                                   }
                                   else Hprima.tipo:= tipo_error
                               else if (I.tipo = Hprima1.tipo = entero)
                               then
                               {
                                   Hprima.tipo:= entero
                               }
                               else Hprima.tipo:= tipo_error
                               }

76. Hprima → λ      {Hprima.tipo:= vacío}

77. I → J Iprima    {   if (Iprima.tipo = vacío)
                           {
                               I.tipo:= J.tipo
                           }
                           else if (J.tipo = Iprima.tipo = entero)
                           then
                           {
                               I.tipo:= entero
                           }
                           else J.tipo:= tipo_error
                           }

```

```

78. Iprima → * J Iprima1    {   if (Iprima1.tipo = vacío)
                                then if (J.tipo = entero)
                                    then
                                        {
                                            Iprima.tipo:= entero
                                        }
                                    else Iprima.tipo:= tipo_error
                                else if (J.tipo = Iprima1.tipo = entero)
                                    then
                                        {
                                            Iprima.tipo:= entero
                                        }
                                    else Iprima.tipo:= tipo_error
                                }
79. Iprima → / J Iprima1    {   if (Iprima1.tipo = vacío)
                                then if (J.tipo = entero)
                                    then
                                        {
                                            Iprima.tipo:= entero
                                        }
                                    else Iprima.tipo:= tipo_error
                                else if (J.tipo = Iprima1.tipo = entero)
                                    then
                                        {
                                            Iprima.tipo:= entero
                                        }
                                    else Iprima.tipo:= tipo_error
                                }
80. Iprima → mod J Iprima1  {   if (Iprima1.tipo = vacío)
                                then if (J.tipo = entero)
                                    then
                                        {
                                            Iprima.tipo:= entero
                                        }
                                    else Iprima.tipo:= tipo_error
                                else if (J.tipo = Iprima1.tipo = entero)
                                    then
                                        {
                                            Iprima.tipo:= entero
                                        }
                                    else Iprima.tipo:= tipo_error
                                }
81. Iprima → λ {Iprima.tipo:= vacío}
82. J → K Jprima    {   if (Jprima.tipo = vacío)
                        then
                            {
                                J.tipo:= K.tipo
                            }
                        else if (K.tipo = Jprima.tipo = entero)
                            then
                                {
                                    J.tipo:= entero
                                }
                        else J.tipo:= tipo_error
                    }

```

```

83. Jprima → ** K Jprima1  {   if (Jprima1.tipo = vacío)
                                then if (K.tipo = entero)
                                    then
                                        {
                                            Jprima.tipo:= entero
                                        }
                                    else Jprima.tipo:= tipo_error
                                else if (K.tipo = Jprima1.tipo = entero)
                                    then
                                        {
                                            Jprima.tipo:= entero
                                        }
                                    else Jprima.tipo:= tipo_error
                                }

84. Jprima → λ{Jprima.tipo:= vacío}

85. K → not K1 {   if (K1.tipo = lógico)
                    then
                        {
                            K.tipo:= lógico
                        }
                    else K.tipo:= tipo_error
                }

86. K → + K1  {   if (K1.tipo = entero)
                    then
                        {
                            K.tipo:= entero
                        }
                    else K.tipo:= tipo_error
                }

87. K → - K1  {   if (K1.tipo = entero)
                    then
                        {
                            K.tipo:= entero
                        }
                    else K.tipo:= tipo_error
                }

88. K → Z      {K.tipo:= Z.tipo}

89. Z → entero Zprima {   if (Zprima.tipo = vacío)
                           then
                               {
                                   Z.tipo:= entero
                               }
                           else if (Zprima.tipo = entero)
                               then
                                   {
                                       Z.tipo:= lógico
                                   }
                           else Z.tipo:= tipo_error
                           }

90. Z → cadena    {Z.tipo:= cadena}
91. Z → true      {Z.tipo:= lógico}
92. Z → false     {Z.tipo:= lógico}

```

```

93. Z → id LL Zprima    {   id.tipo:= BuscaTipoTS (id.pos)
                           if (id.tipo = LL.tipo→t)
                           then if (t ≠ vacío)    // id es función
                                then if (Zprima.tipo = vacío)
                                     then
                                     {
                                         Z.tipo:= t
                                     }
                                else if (Zprima.tipo = t = entero)
                                then
                                {
                                    Z.tipo:= lógico
                                }
                                else Z.tipo:= tipo_error
                           else Z.tipo:= tipo_error // la función tendría que devolver un valor
                           else if (LL.tipo = vacío) // id es variable
                           then if (Zprima.tipo = vacío)
                                then
                                {
                                    Z.tipo:= id.tipo
                                }
                                else if (Zprima.tipo = id.tipo = entero)
                                then
                                {
                                    Z.tipo:= lógico
                                }
                                else Z.tipo:= tipo_error
                           else Z.tipo:= tipo_error // variable con parámetros
                           }

94. Z → ( E ) Zprima {   if (Zprima.tipo = vacío)
                           then
                           {
                               Z.tipo:= E.tipo
                           }
                           else if (Zprima.tipo = E.tipo = entero)
                           then
                           {
                               Z.tipo:= lógico
                           }
                           else Z.tipo:= tipo_error
                           }

95. Z → max ( L )    {   Z.tipo:= entero
                           for i:= 1 to L.long
                               if (L.tipo[i] ≠ entero)
                               then Z.tipo:= tipo_error
                           }

96. Z → min ( L )    {   Z.tipo:= entero
                           for i:= 1 to L.long
                               if (L.tipo[i] ≠ entero)
                               then Z.tipo:= tipo_error
                           }

97. Zprima → in (L) {   Zprima.tipo:= entero
                           for i:= 1 to L.long
                               if (L.tipo[i] ≠ entero)
                               then Zprima.tipo:= tipo_error
                           }

98. Zprima → λ      {Zprima.tipo:= vacío}

```

## Anexo 3: Gramática Modificada para Generar la Tabla LL

//// Conjunto de símbolos terminales:

Terminales = { program ; id procedure function var boolean integer string ( : ) if then begin end else while do repeat until loop for := to case of entero otherwise write writeln read return exit when , or xor and = <> > >= < <= + - \* / mod \*\* not cadena true false in max min }

//// Conjunto de símbolos no terminales y acciones semánticas:





```

NoTerminales = { P R PP PR PF D DD T A AA C B Bloque N O S LL L Q V W Y E F G H I J K Z X
Eprima Fprima Gprima Hprima Iprima Jprima Zprima Bcola Scola BlqElse
_acc1_1 _acc1_2 _acc2 _acc3 _acc4 _acc5 _acc6_1 _acc6_2 _acc6_3 _acc7_1 _acc7_2 _acc7_3 _acc7_4 _acc8_1
_acc8_2 _acc8_3 _acc8_4 _acc9_1 _acc9_2 _acc10 _acc11_1 _acc11_2 _acc12 _acc13 _acc14 _acc15 _acc16_1
_acc16_2 _acc17 _acc18_1 _acc18_2 _acc19 _acc20 _acc21 _acc22 _acc23 _acc24 _acc25 _acc26 _acc27 _acc28
_acc29 _acc30 _acc31 _acc32 _acc33 _acc34 _acc35 _acc36 _acc37 _acc38 _acc39 _acc40 _acc41 _acc42 _acc43
_acc44 _acc45 _acc46 _acc47 _acc48 _acc49 _acc50 _acc51 _acc52 _acc53 _acc54 _acc55 _acc56 _acc57 _acc58
_acc59 _acc60 _acc61 _acc62 _acc63 _acc64 _acc65 _acc66 _acc67 _acc68 _acc69 _acc70 _acc71 _acc72 _acc73
_acc74 _acc75 _acc76 _acc77 _acc78 _acc79 _acc80 _acc81 _acc82 _acc83 _acc84 _acc85 _acc86 _acc87 _acc88
_acc89 _acc90 _acc91 _acc92 _acc93 _acc94 _acc95 _acc96 _acc97 _acc98 }

//// Axioma
Axioma = P

//// Lista de reglas:
Producciones = {
P -> _acc1_1 D R _acc1_2

R -> PP R _acc2
R -> PR R _acc3
R -> PF R _acc4
R -> _acc5

PP -> program id _acc6_1 PYC D _acc6_2 Bloque PYC _acc6_3

PR -> procedure id _acc7_1 A PYC _acc7_2 D _acc7_3 Bloque PYC _acc7_4

PF -> function id _acc8_1 A DOSPUNTOS T PYC _acc8_2 D _acc8_3 Bloque PYC _acc8_4

D -> var id DOSPUNTOS T PYC _acc9_1 DD _acc9_2
D -> _acc10

DD -> id DOSPUNTOS T PYC _acc11_1 DD _acc11_2
DD -> _acc12

T -> boolean _acc13
T -> integer _acc14
T -> string _acc15

A -> PARENT_ABRIR X id DOSPUNTOS T _acc16_1 AA PARENT_CERRAR _acc16_2
A -> _acc17

AA -> PYC X id DOSPUNTOS T _acc18_1 AA _acc18_2
AA -> _acc19

X -> var _acc20
X -> _acc21

C -> B C _acc22
C -> _acc23

Bloque -> begin C end _acc24

B -> S _acc25
B -> if E then Bcola _acc26

Bcola -> S _acc27
Bcola -> Bloque PYC BlqElse _acc28

BlqElse -> else Bloque PYC _acc29
BlqElse -> _acc30

B -> while E do Bloque PYC _acc31
B -> repeat C until E PYC _acc32
B -> loop C end PYC _acc33
B -> for id ASIGNACION E to E do Bloque PYC _acc34
B -> case E of N O end PYC _acc35

N -> entero DOSPUNTOS Bloque PYC N _acc36
N -> _acc37

```

```

O -> otherwise DOSPUNTOS Bloque PYC _acc38
O -> _acc39

S -> write LL PYC _acc40
S -> writeln LL PYC _acc41
S -> read PARENT_ABRIR V PARENT_CERRAR PYC _acc42
S -> id Scola _acc43

Scola -> ASIGNACION E PYC _acc44
Scola -> LL PYC _acc45

S -> return Y PYC _acc46
S -> exit when E PYC _acc47

LL -> PARENT_ABRIR L PARENT_CERRAR _acc48
LL -> _acc49

L -> E Q _acc50

Q -> COMA E Q _acc51
Q -> _acc52

V -> id W _acc53

W -> COMA id W _acc54
W -> _acc55

Y -> E _acc56
Y -> _acc57

E -> F Eprima _acc58

Eprima -> or F Eprima _acc59
Eprima -> xor F Eprima _acc60
Eprima -> _acc61

F -> G Fprima _acc62

Fprima -> and G Fprima _acc63
Fprima -> _acc64

G -> H Gprima _acc65

Gprima -> IGUAL H Gprima _acc66
Gprima -> DISTINTO H Gprima _acc67
Gprima -> MAYOR H Gprima _acc68
Gprima -> MAYOR_IGUAL H Gprima _acc69
Gprima -> MENOR H Gprima _acc70
Gprima -> MENOR_IGUAL H Gprima _acc71
Gprima -> _acc72

H -> I Hprima _acc73

Hprima -> MAS I Hprima _acc74
Hprima -> MENOS I Hprima _acc75
Hprima -> _acc76

I -> J Iprima _acc77

Iprima -> PRODUCTO J Iprima _acc78
Iprima -> DIVISION J Iprima _acc79
Iprima -> mod J Iprima _acc80
Iprima -> _acc81

J -> K Jprima _acc82

Jprima -> POTENCIA K Jprima _acc83
Jprima -> _acc84

```

```

K -> not K _acc85
K -> MAS K _acc86
K -> MENOS K _acc87
K -> Z _acc88

Z -> entero Zprima _acc89
Z -> cadena _acc90
Z -> true _acc91
Z -> false _acc92
Z -> id LL Zprima _acc93
Z -> PARENT_ABRIR E PARENT_CERRAR Zprima _acc94
Z -> max PARENT_ABRIR L PARENT_CERRAR _acc95
Z -> min PARENT_ABRIR L PARENT_CERRAR _acc96

Zprima -> in PARENT_ABRIR L PARENT_CERRAR _acc97
Zprima -> _acc98

```

```

_acc1_1 -> lambda
_acc1_2 -> lambda
_acc2 -> lambda
_acc3 -> lambda
_acc4 -> lambda
_acc5 -> lambda
_acc6_1 -> lambda
_acc6_2 -> lambda
_acc6_3 -> lambda
_acc7_1 -> lambda
_acc7_2 -> lambda
_acc7_3 -> lambda
_acc7_4 -> lambda
_acc8_1 -> lambda
_acc8_2 -> lambda
_acc8_3 -> lambda
_acc8_4 -> lambda
_acc9_1 -> lambda
_acc9_2 -> lambda
_acc10 -> lambda
_acc11_1 -> lambda
_acc11_2 -> lambda
_acc12 -> lambda
_acc13 -> lambda
_acc14 -> lambda
_acc15 -> lambda
_acc16_1 -> lambda
_acc16_2 -> lambda
_acc17 -> lambda
_acc18_1 -> lambda
_acc18_2 -> lambda
_acc19 -> lambda
_acc20 -> lambda
_acc21 -> lambda
_acc22 -> lambda
_acc23 -> lambda
_acc24 -> lambda
_acc25 -> lambda
_acc26 -> lambda
_acc27 -> lambda
_acc28 -> lambda
_acc29 -> lambda
_acc30 -> lambda
_acc31 -> lambda
_acc32 -> lambda
_acc33 -> lambda
_acc34 -> lambda
_acc35 -> lambda
_acc36 -> lambda
_acc37 -> lambda
_acc38 -> lambda
_acc39 -> lambda

```

```
_acc40 -> lambda
_acc41 -> lambda
_acc42 -> lambda
_acc43 -> lambda
_acc44 -> lambda
_acc45 -> lambda
_acc46 -> lambda
_acc47 -> lambda
_acc48 -> lambda
_acc49 -> lambda
_acc50 -> lambda
_acc51 -> lambda
_acc52 -> lambda
_acc53 -> lambda
_acc54 -> lambda
_acc55 -> lambda
_acc56 -> lambda
_acc57 -> lambda
_acc58 -> lambda
_acc59 -> lambda
_acc60 -> lambda
_acc61 -> lambda
_acc62 -> lambda
_acc63 -> lambda
_acc64 -> lambda
_acc65 -> lambda
_acc66 -> lambda
_acc67 -> lambda
_acc68 -> lambda
_acc69 -> lambda
_acc70 -> lambda
_acc71 -> lambda
_acc72 -> lambda
_acc73 -> lambda
_acc74 -> lambda
_acc75 -> lambda
_acc76 -> lambda
_acc77 -> lambda
_acc78 -> lambda
_acc79 -> lambda
_acc80 -> lambda
_acc81 -> lambda
_acc82 -> lambda
_acc83 -> lambda
_acc84 -> lambda
_acc85 -> lambda
_acc86 -> lambda
_acc87 -> lambda
_acc88 -> lambda
_acc89 -> lambda
_acc90 -> lambda
_acc91 -> lambda
_acc92 -> lambda
_acc93 -> lambda
_acc94 -> lambda
_acc95 -> lambda
_acc96 -> lambda
_acc97 -> lambda
_acc98 -> lambda
}
```