Procesadores de Lenguaje

Practica 2do Cuatrimestre

Alicia Pérez Jiménez
Gabriela Ruiz Escobar

03 de Junio de 2011

Contenido

Requisitos

Lenguaje:

El lenguaje a procesar contiene todas las características fijadas para la primera entrega, más las siguientes

- **Acciones de control**:
  - Acción *if*
    - Formato: Uno de los dos siguentes:
      - **if** *Casos* **else** As_d **fi;**
      - **if** *Casos* **fi;**

      (es decir, la parte *else* es opcional), donde *Casos* es una secuencia de uno o más *casos* separados por **elsif**
    - Cada caso es de la forma *Exp* **then** *As*, con *Exp* una expresión entera y *As* una secuencia de cero o más acciones.
    - La semántica operacional informal de **if** E_0 **then** As_0 **elsif** E_1 **then** As_1 **elsif** … **elsif** E_n **then** As_n **else** As_d **fi;** es:
      - Evaluar E_0
      - Si el resultado es distinto de 0, ejecutar As_0 y terminar
      - Evaluar E_1
      - Si el resultado es distinto de 0, ejecutar As_1 y terminar
      - …
      - Evaluar E_n
      - Si el resultado es distinto de 0, ejecutar As_n y terminar
      - Ejecutar As_d
    - La semántica operacional informal de **if** E_0 **then** As_0 **elsif** E_1 **then** As_1 **elsif** … **elsif** E_n **then** As_n **endif;**
      - Evaluar E_0
      - Si el resultado es distinto de 0, ejecutar As_0 y terminar
      - Evaluar E_1
      - Si el resultado es distinto de 0, ejecutar As_1 y terminar
      - …
      - Evaluar E_n
      - Si el resultado es distinto de 0, ejecutar As_n
  - Acción *while*.
    - Formato: **while** *Exp* **do** *As* **endwhile;**, con *Exp* una expresión entera y *As* una secuencia de cero o más acciones.
    - Semántica operacional informal:
      - [*comienzo*] Evaluar *Exp*
      - Si el resultado es distinto de 0
        - Ejecutar *As*
        - Volver a *comienzo*
      - Si el resultado es 0, no hacer nada

- **Tipos construidos**:
  - En las secciones de declaraciones se permitirá declarar tipos.
  - Cada declaración de tipo comenzará con la palabra reservada **tipo**. A continuación aparecerá una *descripción de tipo* seguida por un *identificador de tipo*, seguido por ;.

- Las descripciones de tipo pueden ser:
  - Los tipos básicos contemplados en la primera entrega: `int` y `real`.
  - Otro identificador de tipo.
  - La descripción de un tipo *array*: *DTipo* [*num*], con *num* un número natural, y *DTipo* una descripción de tipo (el *tipo base* del array; es decir, el tipo de los elementos).
  - La descripción de un tipo *registro*: **rec** C_0; …; C_n; **endrec**.
    - Cada C_i es una descripción de campo. El formato de dicha descripción de campo es *DTipo NombreCampo*.
    - Debe haber, por lo menos, una descripción de campo.
    - No se permiten nombres de campo duplicados.
  - La descripción de un tipo *puntero*: **pointer** *DTipo*.
    - *DTipo* es la descripción del tipo base del puntero (el tipo de los objetos apuntados).
    - Se introduce el literal **null**. Este valor es compatible con cualquier tipo puntero, y denota un puntero que no apunta a ningún objeto.
    - Los punteros pueden compararse mediante == y !=.
- Como regla general, cuando, al describir un tipo, se utiliza un identificador de tipo, dicho identificador debe haber sido previamente declarado. La excepción a esta regla es en la descripción del tipo base en un tipo *puntero*: en este caso, se permite referir, además, cualquier otro identificador de tipo de los declarados en la misma sección de declaraciones.
- Los tipos de las variables pueden ser descripciones de tipos arbitrarias.
- Para decidir si un objeto puede asignarse a otro objeto, se llevará a cabo una comprobación estructural de sus tipos de acuerdo con las reglas siguientes:
  - Un objeto de tipo `int` puede asignarse a otro objeto de tipo `int` o a uno de tipo `real`.
  - Un objeto de tipo `real` únicamente puede asignarse a otro objeto de tipo `real`.
  - Un objeto de tipo *array* puede asignarse únicamente a otro objeto de tipo *array*. Además, ambos objetos deben: (i) tener el mismo número de elementos, (ii) tener tipos base estructuralmente compatibles.
  - Un objeto de tipo *registro* puede asignarse únicamente a otro objeto de tipo *registro*. Además:
    - Ambos registros deben tener exactamente el mismo número de campos.
    - Sea *n* el número de campos en ambos registros. Para cada *i* ($1 \leq i \leq n$), los campos que aparecen declarados en las posiciones *i*-esimas en los tipos de ambos registros han de tener tipos estructuralmente compatibles (aunque pueden diferir en su nombre).
  - Un objeto de tipo *puntero* puede asignarse únicamente a otro objeto de tipo *puntero* siempre y cuando los tipos base sean estructuralmente compatibles.
  - Al considerar identificadores de tipo, se entenderá que dichos identificadores son equivalentes a sus descripciones de tipo asociadas.
  - Si, durante el proceso de comprobación de compatibilidad estructural, se plantea más de una vez la compatibilidad estructural del mismo par de tipos, ambos tipos deben considerarse estructuralmente compatibles.

- A fin de acceder a los diferentes elementos de un objeto de tipo estructurado, pueden utilizarse *designadores*. Estos designadores se ajustan a la siguiente estructura:
    - Una variable o parámetro formal (ver más adelante) es un designador. Su tipo será el tipo de la variable o parámetro.
    - *d*[*Exp*], con *d* un designador de tipo *array* y *Exp* una expresión entera. El tipo de *d*[*Exp*] es el tipo base del array.
    - *d*.*campo*, donde *d* es un designador de tipo *registro*, y *campo* es un campo de dicho registro. El tipo de *d*.*campo* es el tipo del campo.
    - *d*^, con *d* un designador de tipo *puntero*. El tipo que resulta será el tipo base del puntero.
- Los designadores son una generalización de las variables en la versión anterior del lenguaje: pueden jugar tanto el papel de expresiones básicas, como aparecer en la parte izquierda de una asignación.
- Se añaden dos nuevos tipos de acciones:
    - *Acción de reserva de memoria*. Comienza con la palabra reservada **alloc**, seguida de un designador de tipo *puntero*, seguido por ;. Su efecto es crear un nuevo objeto del tipo base del puntero, e inicializar el puntero con la dirección de dicho objeto.
    - *Acción de liberación de memoria*. Comienza con la palabra reservada **free**, seguida de un designador de tipo *puntero*, seguido por ;. Su efecto es liberar el espacio ocupado por el objeto apuntado por el puntero.

- **<u>Subprogramas</u>**:
    - En las secciones de declaraciones se permitirá declarar funciones.
    - Cada declaración de función constará de una *cabecera*, seguida de una sección de declaraciones (opcional), seguida de un cuerpo (una secuencia de cero o más acciones), seguida de **end** *nombreFuncion*, seguido de ;.
    - Formato de la cabecera: **fun** *nombreFuncion*(P_0, …, P_n) *TipoRet*
        - *TipoRet* es opcional. Si aparece, es una cláusula de la forma **returns** *DTipo*, con *DTipo* una descripción de tipo compatible con `int`, `real` o con un tipo puntero.
        - El *tipo de retorno* de la función es el indicado en *TipoRet* si dicha cláusula aparece, o `int` en otro caso.
        - Cada P_i es un *parámetro formal*, cuyo modo puede ser:
            - Por valor: *DTipo param*. *DTipo* es la descripción del tipo del parámetro, y *param* su nombre.
            - Por variable: *DTipo* & *param*.
        - La lista de parámetros es opcional. Aunque siempre deberán escribirse los dos paréntesis.
        - El nombre de la función debe coincidir con el nombre indicado en el *end* de la misma.
    - No puede haber parámetros duplicados. Así mismo, los nombres de los parámetros deben ser diferentes de los nombres de variables, tipos y funciones declarados en la sección de declaraciones de la función.
    - El nombre de la función no debe coincidir con el nombre de ninguno de los parámetros. Tampoco debe coincidir con ningún nombre de variable, tipo o función declarado en su sección de declaraciones.
    - El lenguaje adoptará un convenio de *ámbito léxico* para asociar el uso de los nombres con sus declaraciones. Cada ocurrencia de un identificador en la descripción de un tipo o en una instrucción deberá corresponderse con una

declaración de variable, tipo o función. Dicha declaración se buscará primeramente en la sección de declaraciones del bloque en el que está la ocurrencia. Si no aparece allí, se buscará en el bloque padre, … y así sucesivamente.

- Se introduce un nuevo tipo de acción: la acción **return**.
  - Formato: **return** *Exp* ;
  - Sólo puede aparecer en el cuerpo de una función.
  - El tipo de *Exp* debe ser compatible con el tipo de retorno de dicha función.
- Se introduce un nuevo tipo de expresión básica: *invocación de función*.
  - Formato: *nombreFunción* (E_0, …, E_n)
  - Cada parámetro real E_i es una expresión.
  - Una expresión se dice que tiene *modo var* si es, bien un designador, bien una expresión de la forma (*E*), siendo *E* una expresión en *modo var*.
  - La función invocada deberá existir, y el número y el tipo de los parámetros reales deberá coincidir con el número y el tipo de los correspondientes parámetros formales.
  - Al invocar una función sin parámetros deberán escribirse los paréntesis vacíos.
  - Así mismo, cuando el modo del parámetro es por variable, el modo del correspondiente parámetro real debe ser *var*, y el paso de parámetros se realizará *por variable*.
  - La ejecución de la función supone ejecutar el cuerpo de la misma, hasta que ocurre una de las dos siguientes situaciones:
    - Se ejecuta una acción **return** *E* ;. En este caso:
      - Se evalúa *E* en el ámbito de la función
      - El valor que resulta es el *valor devuelto* por la función.
    - Se ejecuta la última acción de la función, y ésta no es **return**. En este caso, el *valor devuelto* es:
      - 0 si el tipo de la función es `int` o `real`.
      - **null** si la función es de tipo puntero.
  - El valor que resulta de invocar la función es el *valor devuelto* tras la ejecución de la misma.
- Las funciones pueden invocarse recursivamente.

Ejemplo de programa en el lenguaje definido:

```
@ Programa de ejemplo
tipo rec
     int tope;
     real[100] valores;
   endrec tsecuencia;
tipo pointer tcelda tarbol;
tipo rec
     real valor;
     tarbol izq;
     tarbol der;
   endrec tcelda;

fun aniade(real valor, tarbol & arbol)
```

```
  fun inserta()
    alloc arbol;
    arbol^.valor = valor;
    arbol^.izq = null;
    arbol^.der = null;
  end inserta;

  if arbol == null then
     inserta();
  elsif arbol^.valor < valor then
     aniade(valor, arbol^.izq);
  elsif arbol^.valor > valor then
     aniade(valor, arbol^.der);
  endif;
end aniade;

fun libera(tarbol & arbol)
  fun liberacion(tarbol arbol)

    if arbol != null then
       liberacion(arbol^.izq);
       liberacion(arbol^.der);
       free arbol;
    endif;
  end liberacion;
  liberacion(arbol);
  arbol := null;
end libera;

fun recolecta(tarbol & arbol)
  float valor;

  libera(arbol);
  while in valor != 0 do
     aniade(valor, arbol);
  endwhile;
end recolecta;

fun aplana(tarbol arbol, tsecuencia & secuencia) returns real
  fun hazAplanado(tarbol arbol) returns real
    real sd;

    if arbol != null then
      sd = hazAplanado(arbol^.izq);
      secuencia.tope = secuencia.tope+1;
      secuencia.valores[secuencia.tope] = arbol^.valor;
      return sd + hazAplanado(arbol^.der);
    endif;
  end hazAplanado;

  secuencia.tope := -1;
  return hazAplanado(arbol);
```

```
end aplana;

tarbol arbol;
tsecuencia secuencia;

arbol := null;
recolecta(arbol);
out aplana(arbol,secuencia);
libera(arbol);
```

Implementación

Las normas de implementación son las mismas que para la primera entrega.

Forma de entrega

La práctica se entregará en un CD con la siguiente estructura de carpetas:

- Carpeta *memoria*. En dicha carpeta se dejará un archivo MS Word `memoria.doc`, conteniendo la presente plantilla convenientemente rellena.
- Carpeta *fuentes*. Los fuentes de los programas implementados
- Carpeta *pruebas.* Los casos de prueba.

El CD se entregará el día 28 de Mayo de 2010, coincidiendo con la prueba escrita. Deberá etiquetarse con el número de grupo, y los nombres y apellidos de los miembros del grupo que han realizado la práctica (aquéllos que hayan renunciado a participar en la práctica no deberán aparecer, ni en la carátula del CD, ni tampoco en la portada de la memoria).

1. Especificación del léxico del lenguaje

Especificación formal del léxico del lenguaje utilizando definiciones regulares.

Letras:

a ≡ a
b ≡ b
c ≡ c
…
z ≡ z
A ≡ A
B ≡ B
C ≡ C
…
Z ≡ Z

Palabras reservadas:

; ≡ ;
int ≡ {i}{n}{t}
real ≡ {r}{e}{a}{l}
. ≡ \.
< ≡ <
> ≡ >
<= ≡ < =
>= ≡ > =
= ≡ =
== ≡ ==
!= ≡ ! =
+ ≡ \+
- ≡ \-
* ≡ *
/ ≡ /
% ≡ %
&& ≡ &&
|| ≡ ||
! ≡ !
(≡ \(
) ≡ \)
(int) ≡ \({i}{n}{t} \)
(real) ≡ \({r}{e}{a}{l} \)
in ≡ {i}{n}
out ≡ {o}{u}{t}
@ ≡ @
_ ≡ _
\ ≡ \\
in ≡ {i}{n}

out ≡ {o}{u}{t}
if≡ {i}{f}
else ≡ {e}{l}{s}{e}
endif ≡ {e}{n}{d}{i}{f}
elsif ≡ {e}{l}{s}{i}{f}
else ≡ {e}{l}{s}{e}
then ≡ {t}{h}{e}{n}
while ≡ {w}{h}{i}{l}{e}
do ≡ {d}{o}
endwhile ≡ {e}{n}{d}{w}{h}{i}{l}{e}
tipo ≡ {t}{i}{p}{o}
rec ≡ {r}{e}{c}
endrec ≡ {e}{n}{d}{r}{e}{c}
pointer ≡ {p}{o}{i}{n}{t}{e}{r}
null ≡ {n}{u}{l}{l}
alloc ≡ {a}{l}{l}{o}{c}
free ≡ {f}{r}{e}{e}
end ≡ {e}{n}{d}
fun ≡ {f}{u}{n}
return ≡ {r}{e}{t}{u}{r}{n}
returns ≡ {r}{e}{t}{u}{r}{n}{s}
& ≡ &
Sentencias:
CaracterASCII: Cualquier carácter del código ASCII cuyo valor se encuentre en el intervalo [32 - 126] excluyendo '\\n'
letra ≡ [a..z | A..Z]
digito ≡ [0..9]
litInt ≡ (-? [1..9] {digito}*) | 0
litReal ≡ {litInt} (\. {digito}* [1..9] | (e | E) {litInt} | \. {digito}* [1..9] (e | E) {litInt})
id ≡ ({letra} | _) ({letra} | {digito} | _)*
type ≡ {int} | {real}
comentario ≡ @ { CaracterASCII }* \\ n

2. Especificación de la sintaxis del lenguaje

Especificación formal de los aspectos sintácticos del lenguaje utilizando una gramática incontextual. Dicha gramática debe representar de manera natural las prioridades y asociatividades de los operadores

2.1. Especificación formal de los aspectos sintácticos del lenguaje.

Utilizando una gramática incontextual. Dicha gramática debe representar de manera natural las prioridades y asociatividades de los operadores

Operadores relacionales

Operador	Aridad	Asociatividad	Prioridad
< (menor)	2	-	2
> (mayor)	2	-	2
<= (menor o igual)	2	-	2
>= (mayor o igual)	2	-	2
== (igual)	2	-	2
!= (distinto)	2	-	2

Operadores aritméticos

Operador	Aridad	Asociatividad	Prioridad
+ (suma)	2	Izquierda	3
- (resta)	2	Izquierda	3
* (multiplicación)	2	Izquierda	4
/ (división)	2	Izquierda	4
% (módulo)	2	Izquierda	4
- (cambio de signo)	1	Si	5

Operadores lógicos

Operador	Aridad	Asociatividad	Prioridad
|| (o lógica)	2	Izquierda	3
&& (y lógica)	2	Izquierda	4
! (negación)	1	Si	5

Operadores de asignación

Operador	Aridad	Asociatividad	Prioridad
= (asignación)	2	Derecha	1

Operadores de lectura/escritura

Operador	Aridad	Asociatividad	Prioridad
In (leer)	1	-	0
Out (escribir)	1	-	0

Operadores de conversión

Operador	Aridad	Asociatividad	Prioridad
(real)	1	-	5
(int)	1	-	5

2.2. Formalización de la sintaxis

Formalización de la sintaxis del lenguaje utilizando una gramática incontextual. Dicha gramática debe representar de manera natural las prioridades y la asociatividad de los operadores.

```
programa ≡ declaraciones acciones
programa ≡ acciones
declaraciones ≡ declaraciones declaracion
declaraciones ≡ declaracion
declaracion ≡ comentario
declaracion ≡ declaracionvar
declaracion ≡ declaraciontipo
declaracion ≡ declaracionfun
declaracionvar ≡ desctipo id ;
declaraciontipo ≡ tipo deftipo id ;
declaracionfun ≡ fun id ( listaparametros ) tiporeturn cuerpo end id ;
listaparametros ≡ parametros
listaParametros ≡ λ
parametros ≡ parametros , parametro
parametros ≡ parametro
parametro ≡ desctipo id
parametro ≡ desctipo & id
tiporeturn ≡ returns desctipo
tiporeturn ≡ λ
cuerpo ≡ declaraciones acciones
cuerpo ≡ acciones
desctipo ≡ id
desctipo ≡ int
desctipo ≡ real
deftipo ≡ desctipo [ litInt ]
deftipo ≡ rec campos endrec
deftipo ≡ pointer desctipo
campos ≡ campos campo
campos ≡ campo ;
campo ≡ desctipo id
acciones ≡ acciones accion
acciones ≡ accion
accion ≡ comentario
accion ≡ accionbasica
accion ≡ accionreturn
```

```
accion ≡ accionalternativa
accion ≡ accioniteracion
accion ≡ accionreserva
accion ≡ accionlibera
accion ≡ accioninvoca
mem ≡ id
mem ≡ mem[expresion2]
mem ≡ mem.id
mem ≡ mem^
accionbasica ≡ expresion ;
accionreturn ≡ return expresion2 ;
accionalternativa ≡ if expresion2 then bloque accionelse endif ;
accionelse ≡ else bloque
accionelse ≡ elsif expresion2 then bloque accionelse
accionelse ≡ λ
bloque ≡ acciones
bloque ≡ λ
accioniteracion ≡ while expresion2 do bloque endwhile ;
accionreserva ≡ alloc mem ;
accionlibera ≡ free men ;
accioninvoca ≡ id ( Aparams )
Aparams ≡ expresiones
Aparams ≡ λ
expresiones ≡ expresiones , expresion2
expresiones ≡ expresion2
expresion ≡ op0in mem
expresion ≡ op0out expresion1
expresion ≡ expresion1
expresion1 ≡ mem op1 expresion2
expresion1 ≡ expresion2
expresion2 ≡ expresion3 op2 expresion3
expresion2 ≡ expresion3
expresion3 ≡ expresion3 op3 expresion4
expresion3 ≡ expresion4
expresion4 ≡ expresion4 op4 expresion5
expresion4 ≡ expresion5
expresion5 ≡ op5asoc expresion5
expresion5 ≡ op5noasoc expresion6
expresion5 ≡ expresion6
expresion6 ≡ (expresion2)
expresion6 ≡ litInt
expresion6 ≡ litReal
expresion6 ≡ mem
expresion6 ≡ null
op0in ≡ in
op0out ≡ out
op1 ≡ =
op2 ≡ <
op2 ≡ >
op2 ≡ <=
op2 ≡ >=
op2 ≡ ==
op2 ≡ !=
op3 ≡ ||
op3 ≡ +
op3 ≡ -
```

```
op4 ≡ *
op4 ≡ /
op4 ≡ %
op4 ≡ &&
op5asoc ≡ -
op5asoc ≡ !
op5noasoc ≡ (int)
op5noasoc ≡ (real)
```

3. Estructura y construcción de la tabla de símbolos

Deberán contemplarse tanto los aspectos necesarios para comprobar las restricciones contextuales, como los necesarios para llevar a cabo la traducción

3.1. Estructura de la tabla de símbolos

Descripción de las operaciones de la tabla de símbolos, definiendo la cabecera de dichas operaciones, así como describiendo informalmente su cometido, incluyendo el propósito de cada uno de sus parámetros

Hemos planteado la tabla de símbolos con la siguiente estructura:
ID: Contendrá el lexema que identifique la entrada en la tabla de símbolos.

- CLASE: Se refiere al tipo de entrada. Ésta podrá ser:
    - tipo: Se refiere a un tipo de datos.
    - fun: Cuando se trata de una función.
    - var: Para los identificadores de variables o parametros por valor.
    - pvar: Para aquellos parametros pasados por referencia.
- DIR (para las variables): Dirección de inicio.
- TIPO: Contendrá una serie de expresiones de tipo que permitirá conocer las propiedades del identificador concreto. Se consideran los siguientes tipo de expresiones:
    - Entero: <t:int>
    - Real: <t:real>
    - Array: <t:array,nelems:...,tbase:...>
        - nelems: Numero de elementos del array.
        - tbase: La expresion de tipo para el tipo base del array
    - Registro: <t:registro,campos:...>
        - campos: Lista de elementos de la forma <id:...,tipo:...>
    - Puntero: <t:puntero,tbase:...>
        - tbase: La expresion de tipo para el tipo base del puntero
    - Funcion: <t:funcion,params:...,tbase:...>
        - params: Lista de elementos de la forma <modo:...,tipo:...>
        - modo: Distingue si es un parametro por valor o por referencia (valor || variable)
        - tipo: Expresión de tipo asociada al parametro.
        - tbase: La expresion de tipo para el objeto que devuelve la funcion.
    - Referencia: <t:referencia,id:..>
- NIVEL: Indica el nivel, de la pila de la tabla de simbolos, en el que se ha realizado una declaración.
- TAMAÑO: Tamaño de las expresiones de tipo.
    - <t:int,tam:1>
    - <t:real,tam:1>
    - <t:array,nelems:...,tbase:...,tam:...>
    - <t:registro,campos:...,tam:...>
    - <t:puntero,tbase:...,tam:1>
- DESPLAZAMIENTO: Desplazamiento para los campos de registros, que representa la posición de un campo con respecto a la dirección del registro:

<id:...,tipo:...,desp:...>

Para facilitar su tratamiento trataremos la tabla de símbolos como si de una tabla clave-valor se tratara, de forma que tendremos un identificador y unas propiedades asociadas a ese identificador. Para ello, hemos estructurado las propiedades de la siguiente forma:

<clase: ..., tipo: ..., nivel: ... >

Para el correcto funcionamiento de las funciones necesitamos una pila de tablas de símbolos, de forma que cuando compilamos el cuerpo relativo a una función, necesitamos su tabla de símbolos y la de niveles superiores, mientras que cuando acabamos la compilación del bloque desapilamos su tabla de símbolos porque ya no la necesitamos.

3.2. Construcción de la tabla de símbolos

Formalización de la construcción de la tabla de símbolos mediante una gramática de atributos

3.2.1 Funciones semánticas

Descripción, si procede, de las funciones semánticas adicionales utilizadas en la especificación. Para cada función debe indicarse explícitamente su cabecera, así como informalmente su cometido, incluyendo el propósito de cada uno de sus parámetros.

Esta sección puede dejarse vacía si no se van a usar funciones semánticas adicionales.

- creaTS(): TS
  - El resultado es una TS vacía.
- creaTS(tsPadre: TS): TS
  - El resultado es la creación de una tabla de símbolos vacía con una tabla padre tsPadre.
- eliminaTS(): TS
  - Desapila la tabla de simbolos de la cima, cuando se finaliza el ambito en el que ésta tabla era valida.
- añadeID(ts: TS, id: String, props: Props): TS
  - El resultado es la TS resultante de añadir id a ts, este metodo gestiona la asignación de dirección.
- existeID(ts: TS, id: String): Boolean
  - El resultado es true si id existe en ts, false en cualquier otro caso.
- dameTipo(ts:TS, id: String): Tipo
  - El resultado es el tipo la declaracion con lexema id
    - ts[id].tipo
- devuelveTipo(op: String, tipo1: Tipo, tipo2: Tipo)
  - El resultado es el tipo resultante de la operacion.

3.2.2 Atributos semánticos

Para cada categoría sintáctica relevante en este procesamiento deben enumerarse sus atributos semánticos, indicando si son heredados o sintetizados, y describiendo informalmente su propósito

Atributo	Categoría	Tipo	Significado
ts	Todas	Sintetizado	Contiene la tabla de símbolos
tsh		Heredado	
n		Sintetizado	Contiene el nivel de la pila de la TS en que se encuenta. Ambito
nh		Heredado	
campos	Registros	Sintetizado	Campos de un registro
camposh		Heredado	

desp	Parámetros	Sintetizado	Desplazamiento de un campo del registro con el origen del mismo.
desph		Heredado	
params		Sintetizado	Parametros de una funcion
paramsh		Heredado	
nparams		Sintetizado	Número de parámetros
nparamsh		Heredado	
modo	Variables y parametros	Sintetizado	Modo del parametro, valor o por variable
id	Memoria	Sintetizado	Identificador de la memoria
tipo	Expresiones de tipo	Sintetizado	Tipo de expresión
tam		Sintetizado	Tamaño de expresión.
op	Operadores	Sintetizado	Nombre de la operación
props	Propiedades	Sintentizado	
dir	Variables y memoria	Sintetizado	Almacena la siguiente dirección de memoria libre
dirh		Heredado	

3.2.3 Gramática de atributos

Gramática de atributos que formaliza la construcción de la tabla de símbolos

programa ≡ declaraciones acciones
declaraciones.tsh = creaTS()
declaraciones.nh = 0
declaraciones.dirh = 0
acciones.tsh = declaraciones.ts
acciones.nh = declaraciones.nh
programa.ts = acciones.ts
programa.n = acciones.n

programa ≡ acciones
acciones.tsh = creaTS()
acciones.nh = 0
programa.ts = acciones.ts
programa.ts = acciones.n

declaraciones ≡ declaraciones declaracion
$declaraciones_1$.tsh = $declaraciones_0$.tsh
$declaraciones_1$.nh = $declaraciones_0$.nh
$declaraciones_1$.dirh = $declaraciones_0$.dirh
declaracion.tsh = $declaraciones_1$.ts
declaracion.nh = $declaraciones_1$.n
declaracion.dirh = $declaraciones_1$.dir
$declaraciones_0$.ts = declaracion.ts
$declaraciones_0$.n = declaracion.n
$declaraciones_0$.dir = declaracion.dir

declaraciones ≡ declaracion
declaracion.tsh = declaraciones.tsh
declaracion.nh = declaraciones.nh
declaracion.dirh = declaraciones.dirh
declaraciones.ts = declaracion.ts
declaraciones.n = declaracion.n
declaraciones.dir = declaracion.dir

declaracion ≡ comentario
declaracion.ts = declaracion.tsh

declaracion.n = declaracion.nh
declaracion.dir = declaracion.dirh

declaracion ≡ declaracionvar

declaracionvar.tsh = declaracion.tsh
declaracionvar.nh = declaracion.nh
declaracionvar.dirh = declaracion.dirh
declaracion.ts = declaracionvar.ts
declaracion.n = declaracionvar.n
declaracion.dir = declaracionvar.dir

declaracion ≡ declaraciontipo

declaraciontipo.tsh = declaracion.tsh
declaraciontipo.nh = declaracion.nh
declaraciontipo.dirh = declaracion.dirh
declaracion.ts = declaraciontipo.ts
declaracion.n = declaraciontipo.n
declaracion.dir = declaraciontipo.dir

declaracion ≡ declaracionfun

declaracionfun.tsh = declaracion.tsh
declaracionfun.nh = declaracion.nh
declaracionfun.dirh = declaracion.dirh
declaracion.ts = declaracionfun.ts
declaracion.n = declaracionfun.n - 1
declaracion.dir = declaracionfun.dir

declaracionvar ≡ desctipo id ;

desctipo.tsh = declaracionvar.tsh
desctipo.nh = declaracionvar.nh
desctipo.dirh = declaracionvar.dirh
declaracionvar.ts = añadeID(desctipo.ts, id.lex, <clase:var, dir: desctipo.dirh, tipo: desctipo.tipo, nivel: desctipo.n>)
declaracionvar.n = desctipo.n
declaracionvar.dir = desctipo.dir + desctipo.tipo.tam

declaraciontipo ≡ tipo deftipo id ;

deftipo.tsh = declaraciontipo.tsh
deftipo.nh = declaraciontipo.nh
deftipo.dirh = declaraciontipo.dirh
declaraciontipo.ts = añadeID(deftipo.ts, id.lex, <clase:tipo, tipo: deftipo.tipo, nivel: deftipo.n>)
declaraciontipo.n = deftipo.n
declaraciontipo.dir = deftipo.dir

declaracionfun ≡ fun id (listaparametros) tiporeturn cuerpo end id ;

listaparametros.tsh = declaracionfun.tsh
listaparametros.nh = declaracionfun.nh + 1
listaparametros.dirh = declaracionfun.dirh
tiporeturn.tsh = listaparametros.ts
tiporeturn.nh = listaparametros.n
tiporeturn.dirh = listaparametros.dir
cuerpo.tsh = tiporeturn.ts
cuerpo.nh = tiporeturn.n
cuerpo.dirh = tiporeturn.dir
declaracionfun.ts = añadeID(cuerpo.ts, id.lex, <clase:fun, tipo: <t:funcion, listaparametros.params, tbase: tiporeturn.tipo>, nivel: cuerpo.n>)
declaracionfun.n = cuerpo.n

declaracionfun.dir = cuerpo.dir

listaparametros ≡ parametros

parametros.tsh = listaparametros.tsh
parametros.nh = listaparametros.nh
parametros.dirh = listaparametros.dirh
parametros.paramsh = λ
listaparametros.ts = parametros.ts
listaparametros.n = parametros.n
listaparametros.dir = parametros.dir
listaparametros.params = parametros.params

listaparametros ≡ λ

listaparametros.ts = listaparametros.tsh
listaparametros.n = listaparametros.nh
listaparametros.dir = listaparametros.dirh
listaparametros.params = λ

parametros ≡ parametros , parametro

$parametros_1$.tsh = $parametros_0$.tsh
$parametros_1$.nh = $parametros_0$.nh
$parametros_1$.dirh = $parametros_0$.dirh
$parametros_1$.paramsh = $parametros_0$.paramsh
parametro.tsh = $parametros_1$.ts
parametro.nh = $parametros_1$.n
parametro.dirh = $parametros_1$.dir
parametro.paramsh = $parametros_1$.params
$parametros_0$.ts = parametro.ts
$parametros_0$.n = parametro.n
$parametros_0$.dir = parametro.dir
$parametros_0$.params = parametro.params

parametros ≡ parametro

parametro.tsh = parametros.tsh
parametro.nh = parametros.nh
parametro.dirh = parámetros.dirh
parametro.paramsh = parametros.paramsh
parametros.ts = parametro.ts
parametros.n = parametro.n
parametros.dir = parametro.dir
parametros.params = parametro.params

parametro ≡ desctipo id

desctipo.tsh = parametro.tsh
desctipo.nh = parametro.nh
desctipo.dirh = parametro.dirh
parametro.ts = añadeID(desctipo.ts, id.lex, <clase:var, tipo:
desctipo.tipo, nivel: desctipo.n>)
parametro.n = desctipo.n
parametro.dir = desctipo.dir + 1
parametro.params = parametro.paramsh ++ <modo:valor,tipo:desctipo.tipo, dir:
desctipo.dir>

parametro ≡ desctipo & id

desctipo.tsh = parametro.tsh
desctipo.nh = parametro.nh
desctipo.dirh = parametro.dirh

parametro.ts = añadeID(desctipo.ts, id.lex, <clase:pvar, tipo: desctipo.tipo, nivel: desctipo.n>)
parametro.n = desctipo.n
parametro.dir = desctipo.dir + 1
parametro.params = parametro.paramsh ++ <modo:variable, tipo:destipo.tipo, dir: desctipo.dir>

tiporeturn ≡ returns desctipo

desctipo.tsh = tiporeturn.tsh
desctipo.nh = tiporeturn.nh
desctipo.dirh = tiporeturn.dirh
tiporeturn.ts = desctipo.ts
tiporeturn.n = desctipo.n
tiporeturn.dir = desctipo.dir
tiporeturn.tipo = desctipo.tipo

tiporeturn ≡ λ

tiporeturn.ts = tiporeturn.tsh
tiporeturn.n = tiporeturn.nh
tiporeturn.dir = tiporeturn.dirh
tiporeturn.tipo = <t:int,tam:1>

cuerpo ≡ declaraciones acciones

declaraciones.tsh = cuerpo.tsh
declaraciones.nh = cuerpo.nh
declaraciones.dirh = cuerpo.dirh
acciones.tsh = declaraciones.ts
acciones.nh = declaraciones.n
cuerpo.ts = acciones.ts
cuerpo.n = acciones.n
cuerpo.dir = declaraciones.dir

cuerpo ≡ acciones

acciones.tsh = cuerpo.tsh
acciones.nh = cuerpo.nh
acciones.dirh = cuerpo.dirh
cuerpo.ts = acciones.ts
cuerpo.ns = acciones.n
cuerpo.dir = acciones.dir

desctipo ≡ id

desctipo.ts = desctipo.tsh
desctipo.n = desctipo.nh
desctipo.dir = desctipo.dirh
desctipo.tipo = <t:referencia, id:id.lex, tam:desctipo.tsh[id.lex].tipo.tam>

desctipo ≡ int

desctipo.ts = desctipo.tsh
desctipo.n = desctipo.nh
desctipo.dir = desctipo.dirh
desctipo.tipo = <t:int,tam:1>

desctipo ≡ real

desctipo.ts = desctipo.tsh
desctipo.n = desctipo.nh
desctipo.dir = desctipo.dirh
desctipo.tipo = <t:real,tam:1>

deftipo ≡ desctipo [litInt]

desctipo.tsh = deftipo.tsh

desctipo.nh = deftipo.nh
desctipo.dirh = deftipo.dirh
deftipo.ts = desctipo.ts
deftipo.n = desctipo.n
deftipo.dir = desctipo.dir
deftipo.tipo = <t:array, nelems:litInt, tbase: desctipo.tipo, tam: litInt * desctipo.tipo.tam>

deftipo ≡ rec campos endrec

campos.tsh = deftipo.tsh
campos.nh = deftipo.nh
campos.dirh = deftipo.dirh
campos.camposh = λ
campos.desph = 0
deftipo.ts = campos.ts
deftipo.n = campos.n
deftipo.dir = campos.dir
deftipo.tipo = <t:registro, campos: campos.campos, tam: campos.tam>

deftipo ≡ pointer desctipo

desctipo.tsh = deftipo.tsh
desctipo.nh = deftipo.nh
desctipo.dirh = deftipo.dirh
deftipo.ts = desctipo.ts
deftipo.n = desctipo.n
deftipo.dir = desctipo.dir
deftipo.tipo = <t:puntero, tbase: desctipo.tipo, tam: 1>

campos ≡ campos campo

$campos_1$.tsh = $campos_0$.tsh
$campos_1$.nh = $campos_0$.nh
$campos_1$.dirh = $campos_0$.dirh
$campos_1$.camposh = $campos_0$.camposh
$campos_1$.tamh = $campos_0$.tamh
$campos_1$.desph = $campos_0$.desph
campo.tsh = $campos_1$.ts
campo.nh = $campos_1$.n
campo.dirh = $campos_1$.dir
campo.camposh = $campos_1$.campos
campo.desph = $campos_1$.desp
$campos_0$.ts = campo.ts
$campos_0$.n = campo.n
$campos_0$.dir = campo.dir
$campos_0$.campos = campo.campos
$campos_0$.desp = campo.desp

campos ≡ campo ,

campo.tsh = campos.tsh
campo.nh = campos.nh
campo.dirh = campos.dirh
campo.camposh = campos.camposh
campo.desph = campos.desph
campos.ts = campo.ts
campos.n = campo.n
campos.dir = campo.dir
campos.campos = campo.campos

campos.desp = campo.desp

campo ≡ desctipo id

desctipo.tsh = campo.tsh
desctipo.nh = campo.nh
desctipo.dirh = campo.dirh
campo.ts = desctipo.ts
campo.n = desctipo.n
campo.dir = desctipo.dir
campo.campos = campo.camposh ++ <id: id.lex, tipo: desctipo.tipo,
desp:desctipo.tipo.tam>
campo.desp = campo.desph + desctipo.tipo.tam

acciones ≡ acciones accion

$acciones_1$.tsh = $acciones_0$.tsh
$acciones_1$.nh = $acciones_0$.nh
accion.tsh = $acciones_1$.ts
accion.nh = $acciones_1$.n
$acciones_0$.ts = accion.ts
$acciones_0$.n = accion.n

acciones ≡ accion

accion.tsh = acciones.tsh
accion.nh = acciones.nh
acciones.ts = accion.ts
acciones.n = accion.n

accion ≡ comentario

accion.ts = accion.tsh
accion.n = accion.nh

accion ≡ accionbasica

accionbasica.tsh = accion.tsh
accionbasica.nh = accion.nh
accion.ts = accionbasica.ts
accion.n = accionbasica.n

accion ≡ accionreturn

accionreturn.tsh = accion.tsh
accionreturn.nh = accion.nh
accion.ts = accionreturn.ts
accion.n = accionreturn.n

accion ≡ accionalternativa

accionalternativa.tsh = accion.tsh
accionalternativa.nh = accion.nh
accion.ts = accionalternativa.ts
accion.n = accionalternativa.n

accion ≡ accioniteracion

accioniteracion.tsh = accion.tsh
accioniteracion.nh = accion.nh
accion.ts = accioniteracion.ts
accion.n = accioniteracion.n

accion ≡ accionreserva

accionreserva.tsh = accion.tsh
accionreserva.nh = accion.nh
accion.ts = accionreserva.ts
accion.n = accionreserva.n

accion ≡ accionlibera

accionlibera.tsh = accion.tsh
accionlibera.nh = accion.nh
accion.ts = accionlibera.ts
accion.n = accionlibera.n

accion ≡ accioninvoca

accioninvoca.tsh = accion.tsh
accioninvoca.nh = accion.nh
accion.ts = accioninvoca.ts
accion.n = accioninvoca.n

men ≡ id

mem.ts = mem.tsh
mem.n = mem.nh
mem.id = id.lex

men ≡ mem [expresion2]

mem_1.tsh = mem_0.tsh
mem_1.nh = mem_0.nh
expresion2.tsh = mem_1.ts
expresion2.nh = mem_1.n
mem_0.ts = expresion2.ts
mem_0.n = expresion2.n
mem_0.id = mem_1.id

men ≡ mem.id

mem_1.tsh = mem_0.tsh
mem_1.nh = mem_0.nh
mem_0.ts = mem_1.ts
mem_0.n = mem_1.n
mem_0.id = mem_1.id

men ≡ mem^

mem_1.tsh = mem_0.tsh
mem_1.nh = mem_0.nh
mem_0.ts = mem_1.ts
mem_0.n = mem_1.n
mem_0.id = mem_1.id

accionbasica ≡ expresion ;

expresion.tsh = accionbasica.tsh
expresion.nh = accionbasica.nh
accionbasica.ts = expresion.ts
accionbasica.n = expresion.n

accionreturn ≡ return expresion2 ;

expresion2.tsh = accionreturn.tsh
expresion2.nh = accionreturn.nh
accionreturn.ts = expresion2.ts
accionreturn.n = expresion2.n

accionalternativa ≡ if expresion2 then bloque accionelse endif ;

expresion2.tsh = accionalternativa.tsh
expresion2.nh = accionalternativa.nh
bloque.tsh = expresion2.ts
bloque.nh = expresion2.n
accionelse.tsh = bloque.ts
accionelse.nh = bloque.n
accionalternativa.ts = accionelse.ts
accionalternativa.n = accionelse.n

accionelse ≡ elsif expresion2 then bloque accionelse

expresion2.tsh = $accionelse_0$.tsh
expresion2.nh = $accionelse_0$.nh
bloque.tsh = expresion2.ts
bloque.nh = expresion2.n
$accionelse_1$.tsh = bloque.ts
$accionelse_1$.nh = bloque.n
$accionelse_0$.ts = $accionelse_1$.ts
$accionelse_0$.n = $accionelse_1$.n

accionelse ≡ else bloque

bloque.tsh = accionelse.tsh
bloque.nh = accionelse.nh
accionelse.ts = bloque.ts
accionelse.n = bloque.n

accionelse ≡λ

accionelse.ts = accionelse.tsh
accionelse.n = accionelse.nh

bloque ≡ acciones

acciones.tsh = bloque.tsh
acciones.nh = bloque.nh
bloque.ts = acciones.ts
bloque.n = acciones.n

bloque ≡λ

bloque.ts = bloque.tsh
bloque.n = bloque.nh

accioniteracion ≡ while expresion2 do bloque endwhile ;

expresion2.tsh = accioniteracion.tsh
expresion2.nh = accioniteracion.nh
bloque.tsh = expresion2.ts
bloque.nh = expresion2.n
accioniteracion.ts = bloque.ts
accioniteracion.n = bloque.n

accionreserva ≡ alloc mem ;

mem.tsh = accionreserva.tsh
mem.nh = accionreserva.nh
accionreserva.ts = mem.ts
accionreserva.n = mem.h

accionlibera ≡ free mem ;

mem.tsh = accionlibera.tsh
mem.nh = accionlibera.nh
accionlibera.ts = mem.ts
accionlibera.n = mem.h

accioninvoca ≡ id (Aparams) ;

Aparams.tsh = accioninvoca.tsh
Aparams.nh = accioninvoca.nh
Aparams.nparamsh = 0
Aparams.paramsh =
accioninvoca.ts = Aparams.ts
accioninvoca.n = Aparams.n

Aparams ≡ expresiones

expresiones.tsh = Aparams.tsh
expresiones.nh = Aparams.nh

expresiones.nparamsh = Aparams.nparamsh
expresiones.paramsh = Aparams.paramsh
Aparams.ts = expresiones.ts
Aparams.n = expresiones.n
Aparams.nparams = expresiones.nparams
Aparams.params = expresiones.params

Aparams ≡ λ

Aparams.ts = Aparams.tsh
Aparams.n = Aparams.nh
Aparams.nparams = Aparams.nparamsh
Aparams.params = Aparams.paramsh

expresiones ≡ expresiones, expresion2

$expresiones_1$.tsh = $expresiones_0$.tsh
$expresiones_1$.nh = $expresiones_0$.nh
$expresiones_1$.nparamsh = $expresiones_0$.nparamsh
$expresiones_1$.paramsh = $expresiones_0$.paramsh
expresion2.tsh = $expresiones_1$.ts
expresion2.nh = $expresiones_1$.n
expresion2.nparamsh = $expresiones_1$.nparams
expresion2.paramsh = $expresiones_1$.params
$expresiones_0$.ts = expresion2.ts
$expresiones_0$.n = expresion2.n
$expresiones_0$.nparams = expresion2.nparams
$expresiones_0$.params = expresion2.params

expresiones ≡ expresion2

expresion2.tsh = expresiones.tsh
expresion2.nh = expresiones.nh
expresion2.nparamsh = expresiones.nparamsh
expresion2.paramsh = expresiones.paramsh
expresiones.ts = expresion2.ts
expresiones.n = expresion2.n
expresiones.nparams = expresion2.nparams
expresiones.params = expresion2.params

expresion ≡ op0in id

op0in.tsh = expresion.tsh
op0in.nh = expresion.nh
expresion.ts = op0in.ts
expresion.n = op0in.n
expresion.tipo = dameTipo(op0in.ts,id.lex)

expresion ≡ op0out expresion1

op0out.tsh = expresion.tsh
op0out.nh = expresion.nh
expresion2.tsh = op0out.ts
expresion2.nh = op0out.n
expresion.ts = expresion2.ts
expresion.n = expresion2.n
expresion.tipo = expresion2.tipo

expresion ≡ expresion1

expresion1.tsh = expresion.tsh
expresion1.nh = expresion.nh
expresion.ts = expresion1.ts
expresion.n = expresion1.n

expresion.tipo = expresion1.tipo

expresion1 ≡ mem op1 expresion2

mem.tsh = expresion1.tsh
mem.nh = expresion1.nh
op1.tsh = mem.ts
op1.nh = mem.n
expresion2.tsh = op1.ts
expresion2.nh = op1.n
expresion1.ts = expresion2.ts
expresion1.n = expresion2.n
expresion1.tipo = dameTipo(expresion1.tsh,mem.lex)

expresion1 ≡ expresion2

expresion2.tsh = expresion1.tsh
expresion2.nh = expresion1.nh
expresion1.ts = expresion2.ts
expresion1.n = expresion2.n
expresion1.tipo = expresion2.tipo

expresion2 ≡ expresion3 op2 expresion3

$expresion3_0$.tsh = expresion2.tsh
$expresion3_0$.nh = expresion2.nh
op2.tsh = $expresion3_0$.ts
op2.nh = $expresion3_0$.n
$expresion3_1$.tsh = op2.ts
$expresion3_1$.nh = op2.n
expresion2.ts = $expresion3_1$.ts
expresion2.n = $expresion3_1$.n
expresion2.tipo = <t:int>
expresion2.nparams = expresion2.nparamsh + 1
expresion2.params = expresion2.paramsh ++ <modo:valor, tipo:<t:int,tam:1>>

expresion2 ≡ expresion3

expresion3.tsh = expresion2.tsh
expresion3.nh = expresion2.nh
expresion2.ts = expresion3.ts
expresion2.n = expresion3.n
expresion2.tipo = expresion3.tipo
expresion2.nparams = expresion2.nparamsh + 1
expresion2.params = expresion2.paramsh ++ <modo:expresion3.modo, tipo:
expresion3.tipo>

expresion3 ≡ expresion3 op3 expresion4

$expresion3_1$.tsh = $expresion3_0$.tsh
$expresion3_1$.nh = $expresion3_0$.nh
op3.tsh = $expresion3_1$.ts
op3.nh = $expresion3_1$.n
expresion4.tsh = op3.ts
expresion4.nh = op3.n
$expresion3_0$.ts = expresion4.ts
$expresion3_0$.n = expresion4.n
$expresion3_0$.tipo = devuelveTipo(op3.op, $expresion3_1$.tipo,expresion4.tipo)
$expresion3_0$.modo = valor

expresion3 ≡ expresion4

expresion4.tsh = expresion3.tsh
expresion4.nh = expresion3.nh

expresion3.ts = expresion4.ts
expresion3.n = expresion4.n
expresion3.tipo = expresion4.tipo
expresion3.modo = expresion4.modo

expresion4 ≡ expresion4 op4 expresion5

$expresion4_1$.tsh = $expresion4_0$.tsh
$expresion4_1$.nh = $expresion4_0$.nh
op4.tsh = $expresion4_1$.ts
op4.nh = $expresion4_1$.n
expresion5.tsh = op4.ts
expresion5.nh = op4.n
$expresion4_0$.ts = expresion5.ts
$expresion4_0$.n = expresion5.n
$expresion4_0$.tipo = devuelveTipo(op4.op, $expresion4_1$.tipo, expresion5.tipo)
$expresion4_0$.modo = valor

expresion4 ≡ expresion5

expresion5.tsh = expresion4.tsh
expresion5.nh = expresion4.nh
expresion4.ts = expresion5.ts
expresion4.n = expresion5.n
expresion4.tipo = expresion5.tipo
expresion4.modo = expresion5.modo

expresion5 ≡ op5asoc expresion5

op5asoc.tsh = $expresion5_0$.tsh
op5asoc.nh = $expresion5_0$.nh
$expresion5_1$.tsh = op5asoc.ts
$expresion5_1$.nh = op5asoc.n
$expresion5_0$.ts = $expresion5_1$.ts
$expresion5_0$.n = $expresion5_1$.n
$expresion5_0$.tipo = devuelveTipo(op5asoc.op, $expresion5_1$.tipo, $expresion5_1$.tipo)
$expresion5_0$.modo = valor

expresion5 ≡ op5noasoc expresion6

op5noasoc.tsh = expresion5.tsh
op5noasoc.nh = expresion5.nh
expresion6.tsh = op5noasoc.ts
expresion6.nh = op5noasoc.n
expresion5.ts = expresion6.ts
expresion5.n = expresion6.n
expresion5.tipo = devuelveTipo(op5noasoc.op, expresion6.tipo,ϕ)
expresion5.modo = valor

expresion5 ≡ expresion6

expresion6.tsh = expresion5.tsh
expresion6.nh = expresion5.nh
expresion5.ts = expresion6.ts
expresion5.n = expresion6.n
expresion5.tipo = expresion6.tipo
expresion5.modo = expresion6.modo

expresion6 ≡ (expresion2)

expresion2.tsh = expresion6.tsh
expresion2.nh = expresion6.nh
expresion6.ts = expresion2.ts
expresion6.n = expresion2.n

expresion6.tipo = expresion2.tipo
expresion6.modo = expresion2.modo

expresion6 ≡ litInt

expresion6.ts = expresion6.tsh
expresion6.n = expresion6.nh
expresion6.tipo = <t:int,tam:1>
expresion6.modo = valor

expresion6 ≡ litReal

expresion6.ts = expresion6.tsh
expresion6.n = expresion6.nh
expresion6.tipo = <t:real,tam:1>
expresion6.modo = valor

expresion6 ≡ mem

mem.tsh = expresion6.tsh
mem.nh = expresion6.nh
expresion6.ts = mem.ts
expresion6.n = mem.n
expresion6.tipo = dameTipo(expresion6.tsh,mem.id)
expresion6.modo = variable

expresion6 ≡ null

expresion6.tsh = expresion6.tsh
expresion6.nh = expresion6.nh
expresion6.ts = mem.ts
expresion6.n = mem.n
expresion6.tipo = <t:null,tam:1>
expresion6.modo = valor

op0in ≡ in

op0in.ts = op0in.tsh
op0in.n = op0in.nh
op0in.op = leer

op0out ≡ out

op0out.ts = op0out.tsh
op0out.n = op0out.nh
op0out.op = escribir

op1 ≡ =

op1.ts = op1.tsh
op1.n = op1.nh
op1.op = asignacion

op2 ≡ <

op2.ts = op2.tsh
op2.n = op2.nh
op2.op = menor

op2 ≡ >

op2.ts = op2.tsh
op2.n = op2.nh
op2.op = mayor

op2 ≡ <=

op2.ts = op2.tsh
op2.n = op2.nh
op2.op = menor_que

op2 ≡ >=

op2.ts = op2.tsh

op2.n = op2.nh
op2.op = mayor_que

op2 ≡ ==

op2.ts = op2.tsh
op2.n = op2.nh
op2.op = igual

op2 ≡ !=

op2.ts = op2.tsh
op2.n = op2.nh
op2.op = distinto

op3 ≡ ||

op3.ts = op3.tsh
op3.n = op3.nh
op3.op = o_logica

op3 ≡ +

op3.ts = op3.tsh
op3.n = op3.nh
op3.op = suma

op3 ≡ -

op3.ts = op3.tsh
op3.n = op3.nh
op3.op = resta

op4 ≡ *

op4.ts = op4.tsh
op4.n = op4.nh
op4.op = multiplicacion

op4 ≡ /

op4.ts = op4.tsh
op4.n = op4.nh
op4.op = division

op4 ≡ %

op4.ts = op4.tsh
op4.n = op4.nh
op4.op = modulo

op4 ≡ &&

op4.ts = op4.tsh
op4.n = op4.nh
op4.op = y_logica

op5asoc ≡ -

op5asoc.ts = op5asoc.tsh
op5asoc.n = op5asoc.nh
op5asoc.op = signo

op5asoc ≡ !

op5asoc.ts = op5asoc.tsh
op5asoc.n = op5asoc.nh
op5asoc.op = negacion

op5noasoc ≡ (int)

op5noasoc.ts = op5noasoc.tsh
op5noasoc.n = op5noasoc.nh
op5noasoc.op = cast_int

op5noasoc ≡ (real)

op5noasoc.ts = op5noasoc.tsh

op5noasoc.n = op5noasoc.nh
op5noasoc.op = cast_real

4. Especificación de las restricciones contextuales

4.1. Funciones semánticas

Descripción, si procede, de las funciones semánticas adicionales utilizadas en la especificación. Para cada función debe indicarse explícitamente su cabecera, así como informalmente su cometido, incluyendo el propósito de cada uno de sus parámetros.

Esta sección puede dejarse vacía si no se van a usar funciones semánticas adicionales.

- cast(ts,lex,tipo): Boolean
  - El resultado es true si la declaracion de id en la ts es del mismo tipo que el tipo pasado por parametro, y false en cualquier otro caso.
- existeCampo(campos,lex): Boolean
  - El resultado es true si existe algun campo en el listado de campos cuyo identificador sea igual a lex, y false en cualquier otro caso.
- compatibles(ts,lex,tipo): Boolean
  - El resultado es true si el tipo de la variable identificada por lex es compatible con el tipo dado, y false en cualquier otro caso.
- validoOperacion(ts,tipo1,op,tipo2): Boolean
  - El resultado es true si los tipos tipo1 y tipo2 son compatibles y validos para realizar la operación op, y false en cualquier otro caso.
- EliminaPendiente(List<lex>, lex):List<lex>
  - El resultado es la lista de tipos pendientes de declarar eliminando, en caso de existir previamente declarado lex.

4.2. Atributos semánticos

Para cada categoría sintáctica relevante en este procesamiento deben enumerarse sus atributos semánticos, indicando si son heredados o sintetizados, y describiendo informalmente su propósito.

Atributo	Tipo	Significado
error	Sintetizado	Contiene un booleano que indica si se ha producido algún error en algún punto.
errorh	Heredado	
pend	Sintetizado	Lista de tipos usados pendientes de declarar.
pendh	Heredado	
Tbloqueh	Sintetizado	Contiene el tipo que debe devolver una funciÓn.
tbloqueh	Heredado	

4.3. Gramática de atributos

Gramática de atributos que formaliza la comprobación de las restricciones contextuales.

programa ≡ declaraciones acciones

declaraciones.errorh = FALSE
declaraciones.pendh =
declaraciones.tbloqueh = int
acciones.errorh = declaraciones.error ∨ declaraciones.pend != Φ
acciones.tbloqueh = declaraciones.tbloque

programa.error = acciones.error

programa ≡ acciones

acciones.errorh = FALSE

acciones.tbloqueh = int

programa.error = acciones.error

declaraciones ≡ declaraciones declaracion

$declaraciones_1$.errorh = $declaraciones_0$.errorh

$declaraciones_1$.pendh = $declaraciones_0$.pendh

$declaraciones_1$.tbloqueh = $declaraciones_0$.tbloqueh

declaracion.errorh = $declaraciones_1$.error

declaracion.pendh = $declaraciones_1$.pendh

declaracion.tbloqueh = $declaraciones_1$.tbloque

$declaraciones_0$.error = declaracion.error

$declaraciones_0$.pend = declaracion.pend

$declaraciones_0$.tbloque = declaracion.tbloque

declaraciones ≡ declaracion

declaracion.errorh = declaraciones.errorh

declaracion.pendh = declaraciones.pendh

declaracion.tbloqueh = declaraciones.tbloqueh

declaraciones.error = declaracion.error

declaraciones.pend = declaracion.pend

declaraciones.tbloque = declaracion.tbloque

declaracion ≡ comentario

declaracion.error = declaracion.errorh

declaracion.pend = declaracion.pendh

declaracion ≡ declaracionvar

declaracionvar.errorh = declaracion.errorh

declaracionvar.pendh = declaracion.pendh

declaracion.error = declaracionvar.error

declaracion.pend = declaracionvar.pend

declaracion ≡ declaraciontipo

declaraciontipo.errorh = declaracion.errorh

declaraciontipo.pendh = declaracion.pendh

declaracion.error = declaraciontipo.error

declaracion.pend = declaraciontipo.pend

declaracion ≡ declaracionfun

declaracionfun.errorh = declaracion.errorh

declaracionfun.pendh = declaracion.pendh

declaracionfun.tbloqueh = declaracion.tbloqueh

declaracion.error = declaracionfun.error

declaracion.pend = declaracionfun.pend

declaracion.tbloque = declaracionfun.tbloque

declaracionvar ≡ desctipo id ;

desctipo.errorh = declaracionvar.errorh

desctipo.pendh = declaracionvar.pendh

declaracionvar.error = (existeID(desctipo.ts, id.lex)

desctipo.ts[id.lex].n == desctipo.n) V desctipo.error

declaracionvar.pend = desctipo.pend

declaraciontipo ≡ tipo deftipo id ;

deftipo.errorh = declaraciontipo.errorh

deftipo.pendh = declaraciontipo.pendh

declaraciontipo.error = (existeID(deftipo.ts, id.lex) ∧ deftipo.ts[id.lex].n == deftipo.n) ∨
deftipo.error
declaraciontipo.pend = deftipo.pend

declaracionfun ≡ fun id (listaparametros) tiporeturn cuerpo end id ;

listaparametros.errorh = declaracionfun.errorh
listaparametros.pendh = declaracionfun.pendh
listaparametros.tbloqueh = declaracionfun.tbloqueh
tiporeturn.erroh = listaparametros.error
tiporeturn.pendh = listaparametros.pend
tiporeturn.tbloqueh = listaparametros.tbloque
cuerpo.erroh = tiporeturn.error
cuerpo.pendh = tiporeturn.pend
cuerpo.tbloqueh = tiporeturn.tbloque
declaracionfun.error = (existeID(cuerpo.ts, id.lex) cuerpo.ts[id.lex].n == cuerpo.n) ∨
(tiporeturn.tipo.tipo {int, real, puntero}) ∨ cuerpo.error
declaracionfun.pend = cuerpo.pend
declaracionfun.tbloque = cuerpo.tbloque

listaparametros ≡ parametros

parametros.errorh = listaparametros.errorh
parametros.pendh = listaparametros.pendh
listaparametros.error = parametros.error
listaparametros.pend = parametros.pend
listaparametros.tbloque = listaparametros.tbloqueh

listaparametros ≡

listaparametros.error = listaparametros.errorh
listaparametros.pend = listaparametros.pendh
listaparametros.tbloque = listaparametros.tbloqueh

parametros ≡ parametros , parametro

$parametros_1$.errorh = $parametros_0$.errorh
$parametros_1$.pendh = $parametros_0$.pendh
parametro.erroh = $parametros_1$.error
parametro.pendh = $parametros_1$.pend
$parametros_0$.error = parametro.error
$parametros_0$.pend = parametro.pend

parametros ≡ parametro

parametro.errorh = parametros.errorh
parametro.pendh = parametros.pendh
parametros.error = parametro.error
parametros.pend = parametro.pend

parametro ≡ desctipo id

desctipo.errorh = parametro.errorh
desctipo.pendh = parametro.pendh
parametro.error = desctipo.error ∨ (existeID(desctipo.ts, id.lex) ∧
desctipo.ts[id.lex].n == desctipo.n)
parametro.pend = desctipo.pend

parametro ≡ desctipo & id

desctipo.errorh = parametro.errorh
desctipo.pendh = parametro.pendh
parametro.error = desctipo.error ∨ (existeID(desctipo.ts, id.lex) ∧
desctipo.ts[id.lex].n == desctipo.n)
parametro.pend = desctipo.pend

tiporeturn ≡ returns desctipo

desctipo.errorh = tiporeturn.errorh
desctipo.pendh = tiporeturn.pendh
tiporeturn.error = desctipo.error V desctipo.tipo ∉{int, real, puntero}
tiporeturn.pend = desctipo.pend
tiporeturn.tbloque = desctipo.tipo

tiporeturn ≡ λ

tiporeturn.error = tiporeturn.errorh
tiporeturn.pend = tiporeturn.pendh
tiporeturn.tbloque = int

cuerpo ≡ declaraciones acciones

declaraciones.errorh = cuerpo.errorh
declaraciones.pendh = cuerpo.pendh
declaraciones.tbloqueh = cuerpo.tbloqueh
acciones.errorh = declaraciones.error
acciones.tbloqueh = declaraciones.tbloque
cuerpo.error = acciones.error
cuerpo.pend = declaraciones.pend
cuerpo.tbloque = acciones.tbloque

cuerpo ≡ acciones

acciones.errorh = cuerpo.errorh
acciones.tbloqueh = cuerpo.tbloqueh
cuerpo.error = acciones.error
cuerpo.pend = cuerpo.pendh
cuerpo.tbloque = acciones.tbloque

desctipo ≡ id

desctipo.error = desctipo.errorh ∨
SI existeID(decstipo.tsh, id.lex) ENTONCES
desctipo.tsh[id.lex].clase != tipo
SI NO
TRUE
desctipo.pend = SI ¬exiteID(desctipo.tsh, id.lex) ENTONCES
desctipo.pendh ++ id.lex
SI NO
SI desctipo.tsh[id.lex].clase == tipo ENTONCES
eliminaPendiente(desctipo.pendh,id.lex)

desctipo ≡ int

desctipo.error = desctipo.erroh
desctipo.pend = desctipo.pendh

desctipo ≡ real

desctipo.error = desctipo.erroh
desctipo.pend = desctipo.pendh

deftipo ≡ desctipo [litInt]

desctipo.errorh = deftipo.erroh
desctipo.pendh = deftipo.pendh
deftipo.error = desctipo.error ∨ ¬cast(desctipo.ts, litInt,<t:int,tam:1>)
deftipo.pend = eliminaPendiente(desctipo.pendh, desctipo.lex)

deftipo ≡ rec campos endrec

campos.errorh = desctipo.erroh
campos.pendh = desctipo.pendh
desctipo.error = campos.error
desctipo.pend = campos.pend

deftipo ≡ pointer desctipo

desctipo.errorh = deftipo.erroh
desctipo.pendh = deftipo.pendh
deftipo.error = desctipo.error
deftipo.pend = SI ¬exiteID(desctipo.tsh, desctipo.lex) ENTONCES
desctipo.pendh ++ desctipo.lex
SI NO
eliminaPendiente(desctipo.pendh, desctipo.lex)

campos ≡ campos campo

$campos_1$.errorh = $campos_0$.erroh
$campos_1$.pendh = $campos_0$.pendh
campo.erroh = $campos_1$.error
campo.pendh = $campos_1$.pend
$campos_0$.error = campo.error
$campos_0$.pend = campo.pend

campos ≡ campo ;

campo.errorh = campos.erroh
campo.pendh = campos.pendh
campos.error = campo.error
campos.pend = campo.pend

campo ≡ desctipo id

desctipo.errorh = campo.erroh V ¬existeCampo(campo.camposh, id.lex)
desctipo.pendh = campo.pendh
campo.error = desctipo.error
campo.pend = elimina(campo.pendh, desctipo.lex)

acciones ≡ acciones accion

acciones1.errorh = acciones0.errorh
acciones1.tbloqueh = acciones0.tbloqueh
accion.errorh = acciones1.error
accion.tbloqueh = acciones1.tbloque
acciones0.error = accion.error
acciones0.tbloque = accion.tbloque

acciones ≡ accion

accion.errorh = acciones.errorh
accion.tbloqueh = acciones.tbloqueh
acciones.error = accion.error
acciones.tbloque = accion.tbloque

accion ≡ comentario

accion.error = accion.errorh

accion ≡ accionbasica

accionbasica.errorh = accion.errorh
accion.error = accionbasica.error

accion ≡ accionreturn

accionreturn.errorh = accion.errorh
accionreturn.tbloqueh = accion.tbloqueh
accion.error = accionreturn.error
accion.tbloque = accionreturn.tbloque

accion ≡ accionalternativa

accionalternativa.errorh = accion.errorh
accion.error = accionalternativa.error

accion ≡ accioniteracion

accioniteracion.errorh = accion.errorh

accion.error = accioniteracion.error

accion ≡ accionreserva

accionreserva.errorh = accion.errorh

accion.error = accionreserva.error

accion ≡ accionlibera

accionlibera.errorh = accion.errorh

accion.error = accionlibera.error

accion ≡ accioninvoca

accioninvoca.errorh = accion.errorh

accion.error = accioninvoca.error

mem ≡ id

mem.error = mem.errorh ∨ ¬existeID(mem.tsh, id.lex) ∨ mem.tsh[id.lex].clase != var

accion.error = accioninvoca.error

mem ≡ mem [expresion2]

mem_1.errorh = mem_0.errorh

expresion2.errorh = mem1.error

mem_0.error = expresion2.error ∨ mem_1.tipo != array ∨ expresion2.tipo!=
<t:int,tam:1>

mem ≡ mem.id

mem_1.errorh = mem_0.errorh

mem_0.error = mem_1.error ∨ mem_1.tipo.t != registro ∨
¬existeCampo(mem_1.camposh,id.lex)

mem ≡ mem^

mem_1.errorh = mem_0.errorh

mem_0.error = mem_1.error ∨ mem_1.tipo.t != puntero

accionbasica ≡ expresion ;

expresion.errorh = accionbasica.errorh

accionbasica.error = expresion.error

accionreturn ≡ return expresion2 ;

expresion2.errorh = accionreturn.errorh

accionreturn.error = expresion2.error ∨ expresion2.tipo != accionreturn.tbloqueh

accionalternativa ≡ if expresion2 then bloque accionelse endif ;

expresion2.errorh = accionalternativa.errorh

bloque.errorh = expresion2.error

accionelse.errorh = bloque.error

accionreturn.error = accionelse.error

accionelse ≡ elsif expresion2 then bloque accionelse

expresion2.errorh = $accionelse_0$.errorh

bloque.errorh = expresion2.error

$accionelse_1$.errorh = bloque.error

$accionelse_0$.error = $accionelse_1$.error

accionelse ≡ else bloque

bloque.errorh = accionelse.errorh

accionelse.error = bloque.error

accionelse ≡

accionelse.error = accionelse.errorh

bloque ≡ acciones

acciones.errorh = bloque.errorh

bloque.error = acciones.error

bloque ≡

bloque.error = bloque.errorh

accioniteracion ≡ while expresion2 do bloque endwhile ;

expresion2.errorh = accioniteracion.errorh

bloque.errorh = expresion2.error

accioniteracion.error = bloque.error

accionreserva ≡ alloc mem ;

mem.errorh = accionreserva.errorh mem.tipo <t:puntero>

accionreserva.error = mem.error

accionlibera ≡ free mem ;

mem.errorh = accionlibera.errorh

accionlibera.error = mem.error

accioninvoca ≡ id (Aparams) ;

Aparams.errorh = accioninvoca.errorh ∨ ¬existeID(accioninvoca.tsh, id.lex) ∨
accioninvoca.tsh[id.lex].tipo.t != <t:funcion>

accioninvoca.error = Aparams.error

Aparams ≡ expresiones

expresiones.errorh = Aparams.errorh

Aparams.error = expresiones.error

Aparams ≡

Aparams.error = Aparams.errorh

expresiones ≡ expresiones , expresion2

$expresiones_1$.errorh = $expresiones_0$.errorh

expresion2.errorh = $expresiones_1$.error

$expresiones_0$.error = expresion2.error

expresiones ≡ expresion2

expresion2.errorh = expresiones.errorh

expresiones.error = expresion2.error

expresion ≡ op0in id

op0in.errorh = expresion.errorh

expresion.error = op0in.error V ¬existeID(op0in.ts, id.lex)

expresion ≡ op0out expresion1

op0out.errorh = expresion.errorh

expresion1.errorh = op0out.error

expresion.error = expresion1.error

expresion ≡ expresion1

expresion1.errorh = expresion.errorh

expresion.error = expresion1.error

expresion1 ≡ mem op1 expresion2

mem.errorh = expresion1.errorh

op1.errorh = mem.error

expresion2.errorh = op1.error

expresion1.error = expresion2.error V
¬compatibles(expresion2.ts,mem.id,expresion2.tipo)

expresion1 ≡ expresion2

expresion2.errorh = expresion1.errorh

expresion1.error = expresion2.error

expresion2 ≡ expresion3 op2 expresion3

$expresion3_0$.errorh = expresion2.errorh

op2.errorh = $expresion3_0$.error

$expresion3_1$.errorh = op2.error

expresion2.error = $expresion3_1$.error ∨ ¬validoOperacion($expresion3_1$ .ts,
$expresion3_0$ .tipo,op2.op,$expresion3_1$.tipo)

expresion2 ≡ expresion3

expresion3.errorh = expresion2.errorh
expresion2.error = expresion3.error

expresion3 ≡ expresion3 op3 expresion4

$expresion3_1$.errorh = $expresion3_0$.errorh
op3.errorh = $expresion3_1$.error
expresion4.errorh = op3.error
$expresion3_0$.error = expresion4.error ∨ ¬validoOperacion(expresion4.ts, $expresion3_1$.tipo, op3.op, expresion4.tipo)

expresion3 ≡ expresion4

expresion4.errorh = expresion3.errorh
expresion3.error = expresion4.error

expresion4 ≡ expresion4 op4 expresion5

$expresion4_1$.errorh = $expresion4_0$.errorh
op4.errorh = $expresion4_1$.error
expresion5.errorh = op4.error
$expresion4_0$.error = expresion5.error ∨¬ validoOperacion(expresion5.ts, $expresion4_1$.tipo, op4.op, expresion5.tipo)

expresion4 ≡ expresion5

expresion5.errorh = expresion4.errorh
expresion4.error = expresion5.error

expresion5 ≡ op5asoc expresion5

op5asoc.errorh = $expresion5_0$.errorh
$expresion5_1$.errorh = op5asoc.error
$expresion5_0$.error = $expresion5_1$.error ∨ ¬validoOperacion($expresion5_1$ .ts, $expresion5_1$.tipo, op5asoc.op, NULL)

expresion5 ≡ op5noasoc expresion6

op5noasoc.errorh = expresion5.errorh
expresion6.errorh = op5noasoc.error
expresion5.error = expresion6.error ∨ ¬validoOperacion(expresion6.ts, expresion6.tipo, op5noasoc.op, NULL)

expresion5 ≡ expresion6

expresion6.errorh = expresion5.errorh
expresion5.error = expresion6.error

expresion6 ≡ (expresion2)

expresion2.errorh = expresion6.errorh
expresion6.error = expresion2.error

expresion6 ≡ litInt

expresion6.error = expresion6.errorh ∨ ¬cast(expresion6.tsh,litInt.lex,<t:int>)

expresion6 ≡ litReal

expresion6.error = expresion6.errorh ∨ ¬cast(expresion6.tsh,litReal.lex,<t:real>)

expresion6 ≡ null

expresion6.error = expresion6.errorh

expresion6 ≡ mem

mem.errorh = expresion6.errorh
expresion6.error = mem.error ∨ ¬ existeID(mem.ts,mem.id)

op0in ≡ in

op0in.error = op0in.errorh

op0out ≡ out

op0out.error = op0out.errorh

op1 ≡ =

op1.error = op1.errorh

op2 ≡ <

op2.error = op2.errorh

op2 ≡ >

op2.error = op2.errorh

op2 ≡ <=

op2.error = op2.errorh

op2 ≡ >=

op2.error = op2.errorh

op2 ≡ ==

op2.error = op2.errorh

op2 ≡ !=

op2.error = op2.errorh

op3 ≡ ||

op3.error = op3.errorh

op3 ≡ +

op3.error = op3.errorh

op3 ≡ -

op3.error = op3.errorh

op4 ≡ *

op4.error = op4.errorh

op4 ≡ /

op4.error = op4.errorh

op4 ≡ %

op4.error = op4.errorh

op4 ≡ &&

op4.error = op4.errorh

op5asoc ≡ -

op5asoc.error = op5asoc.errorh

op5asoc ≡ !

op5asoc.error = op5asoc.errorh

op5noasoc ≡ (int)

op5noasoc.error = op5noasoc.errorh

op5noasoc ≡ (real)

op5noasoc.error = op5noasoc.errorh

5. Especificación de la traducción

5.1. Lenguaje objeto

5.1.1. Arquitectura de la máquina P

Explicar cómo es la arquitectura de la máquina P que se va a emplear en esta práctica.

- **Mem:** Memoria principal con celdas direccionables con datos. Cada celda de la memoria es capaz de guardar cualquier tipo de datos.
- **Prog:** Memoria de programa con celdas direccionables con instrucciones. Cada celda es capaz de guardar cualquier tipo de instrucción, independientemente del tamaño de la misma.
- **CProg:** Contador de programa con un registro para la dirección de la instrucción que está actualmente en ejecución.
- **Pila:** Pila de datos con celdas direccionables con datos. Cada celda de la pila es capaz de guardar cualquier tipo de dato, independientemente del tamaño del mismo.
- **CPila:** Cima de la pila de datos con un registro para la dirección del datos situado actualmente en la cima de la pila.
- **P:** Registro con un bit de parada que detiene la ejecución.

Se podrá utilizar como tipos de datos:

- **Integer**
- **Real**

Comportamiento interno:

- mem[direccion] : Dato de una celda de memoria principal localizado a traves de una direccion.
- prog[direccion] : Instruccion de una celda de memoria del programa localizada a traves de una direccion.
- Registro ← valor : Escritura de un valor en un registro.

5.1.2. Instrucciones en el lenguaje objeto

Enumeración de todo el repertorio de instrucciones del lenguaje objeto de la máquina a pila (máquina P) que se van a utilizar, así como descripción informal de su cometido.

- **LEER:** Leer una entrada y almacenar su valor en la cima de la pila.
- **ESCRIBIR:** Escribir por pantalla el valor de la cima de la pila.
- **MENOR:** Compara el valor del contenido en la cima y la subcima de la pila y guarda el resultado en la misma, considerando el valor 1 como verdadero y el valor 0 como falso.
- **MAYOR:** Compara el valor del contenido en la cima y la subcima de la pila y guarda el resultado en la misma, considerando el valor 1 como verdadero y el valor 0 como falso.
- **MENOR_IGUAL:** Compara el valor del contenido en la cima y la subcima de la pila y guarda el resultado en la misma, considerando el valor 1 como verdadero y el valor 0 como falso.
- **MAYOR_IGUAL:** Compara el valor del contenido en la cima y la subcima de la pila y guarda el resultado en la misma, considerando el valor 1 como verdadero y el valor 0 como falso.

- **IGUAL:** Compara el valor del contenido en la cima y la subcima de la pila y guarda el resultado en la misma, considerando el valor 1 como verdadero y el valor 0 como falso.
- **DISTINTO**: Compara el valor del contenido en la cima y la subcima de la pila y guarda el resultado en la misma, considerando el valor 1 como verdadero y el valor 0 como falso.
- **O_LOGICA:** Realiza la operación o-lógica con los valores contenidos en la cima y la subcima de la pila y guarda el resultado en la misma. Los valores solo pueden ser enteros.
- **SUMA:** Realiza la operación de suma con los valores numéricos contenidos en la cima y la subcima de la pila y guarda el resultado en la misma.
- **RESTA:** Realiza la operación de resta con los valores contenidos en la cima y la subcima de la pila y guarda el resultado en la misma.
- **MULTIPLICACION:** Realiza la operación de multiplicación con los valores contenidos en la cima y la subcima de la pila y guarda el resultado en la misma.
- **DIVISION:** Realiza la operación de división con los valores contenidos en la cima y la subcima de la pila y guarda el resultado en la misma. Con la excepción de que la división actuara como división real siempre que alguno de los dos valores sea de tipo real y como división entera cuando ambos operandos sean enteros.
- **MODULO:** Realiza la operación de modulo con los valores contenidos en la cima y la subcima de la pila y guarda el resultado en la misma. Con la excepcion de que el valor de la cima y la subcima deben ser enteros.
- **Y_LOGICA:** Realiza la operación y-lógica con los valores contenidos en la cima y la subcima de la pila y guarda el resultado en la misma. Los valores solo pueden ser enteros.
- **SIGNO:** Cambia el signo del valor contenido en la cima de la pila y guarda el resultado en la misma.
- **NEGACION:** Realiza la operación negación-lógica con los valores contenidos en la cima y la subcima de la pila y guarda el resultado en la misma. Los valores solo pueden ser enteros.
- **CAST_INT:** Realiza la conversión del valor de la cima de la pila a un valor entero y lo guarda en la misma.
- **CAST_REAL:** Realiza la conversión del valor de la cima de la pila a un valor real y lo guarda en la misma.
- **APILA:** Almacena en la cima de la pila el valor contenido en el argumento de la instrucción.
- **DESAPILA:** Retira de la pila, el valor contenido en la cima de la misma.
- **APILA_DIR:** Almacena en la cima de la pila el valor contenido en la dirección de memoria identificada mediante el contenido del argumento de la instrucción.
- **DESAPILA_DIR:** Guarda en la posición de memoria indentificada mediante el contenido del argumento de la instrucción, el valor contenido en la cima de la pila.
- **APILA_IND:** Almacena en la cima de la pila el valor contenido en la direccion de memoria identificada mediante el valor obtenido de la cima de la pila con anterioridad al nuevo almacenamiento.
- **DESAPILA_IND:** Guarda el valor contenido en la cima de la pila, en la posicion de memoria indicada por el valor contenido en la subcima.
- **IR_A:** Realiza un salto a la direccion contenida en el argumento de la instrucción.
- **IR_F:** Reaiza un salto a la direccion contenida en el argumento de la instrucción, si el valor contenido en la cima de la pila es FALSE.
- **IR_V:** Reaiza un salto a la direccion contenida en el argumento de la instrucción, si el valor contenido en la cima de la pila es TRUE.
- **IR_IND:** Realiza un salto a la direccion contenida en la cima de la pila.

- **MUEVE:** Mueve el valor contenido del numero de celdas consecutivas, indicado por el valor del argumento de la instrucción, de la direccion indicada por el valor contenido en la cima de la pila (que actua como origen) a la direccion indicada por el valor contenido en la subcima de la pila (que actua como destino).
- **COPIA:** Duplica, en la pila, el valor de la cima.
- **NEW:** Reserva el espacio, indicado en el argumento de la instrucción, en la memoria y guarda la dirección de este espacio en la cima de la pila.
- **DEL:** Libera el espacio, indicado en el argumento de la instrucción, en la memoria a partir de la direccion obtenida del valor de la cima de la pila.
- **STOP:** Detiene la ejecución del programa.

5.2. Funciones semánticas

Descripción, si procede, de las funciones semánticas adicionales utilizadas en la especificación. Para cada función debe indicarse explícitamente su cabecera, así como informalmente su cometido, incluyendo el propósito de cada uno de sus parámetros.

Esta sección puede dejarse vacía si no se van a usar funciones semánticas adicionales.

- **ConversionAsignacion** (tipo): op
  - Determina que conversion debe ejecutar para realizar una correcta asignación. Si el tipo es <t:int> devuelve cast_int, en cualquier otro caso devuelve cast_real.
- fun **inicio** (numNiveles, tamDatos)
  apila (numNiveles + 2)
  desapila_dir (1)
  apila (1+numNiveles+tamDatos)
  desapila_dir(0)
- cons longInicio = 4
- fun **apila-ret** (ret)
  apila_dir (0)
  apila (1)
  suma
  apila (ret)
  desapila_ind
- cons **longApilaRet** = 5
- fun **prologo** (nivel, tamLocales)
  apila_dir (0)
  apila (2)
  suma
  apila_dir (1+nivel)
  desapila_ind
  apila_dir (0)
  apila (3)
  suma
  desapila_dir (1+nivel)
  apila_dir (0)
  apila (tamLocales+2)
  suma
  desapila_dir (0)
- cons **longPrologo** = 13
- fun **epilogo** (nivel)

apila_dir (1+nivel)
apila (2)
resta
apila_ind
apila_dir (1+nivel)
apila (3)
resta
copia
desapila_dir (0)
apila (2)
suma
apila_ind
desapila_dir (1+nivel)

- cons **longEpilogo** = 13
- fun **accesoVar** (infoID)

apila_dir (1+infoID.nivel)
apila (infoID.dir)
suma
SI infoID.clase == pvar ENTONCES
apila_ind
SI NO
λ

- fun **longAccesoVar** (infoID)

SI infoID.clase == pvar ENTONCES
4
SI NO
3

- cons **inicio-paso** =

apila_dir (0)
apila (3)
suma

- cons **longInicioPaso** = 3
- cons **fin-paso** = desapila
- cons **longFinPaso** = 1
- fun **direccionParFormal** (pformal)

apila (pformal.dir)
suma

- cons **longDireccionParFormal** = 2
- fun **pasoParametro** (modoReal, pformal)

SI pformal.modo == val modoReal == var ENTONCES
mueve (pformal.tipo.tam)
SI NO
desapila_ind

- cons **longPasoParametro** = 1

5.3. Atributos semánticos

Para cada categoría sintáctica relevante en este procesamiento deben enumerarse sus atributos semánticos, indicando si son heredados o sintetizados, y describiendo informalmente su propósito.

Atributo	Tipo	Significado
cod	Sintetizado	Atributo que almacena el código de instrucciones de la maquina P
codh	Heredado	
etq	Sintetizado	Atributo que indica la dirección en el código de la siguiente instrucción
etqh	Heredado	
propsop	Sintetizado	Indica el inicio de cada funcion

5.4. Gramática de atributos

Gramática de atributos que formaliza la traducción

programa ≡ declaraciones acciones
declaraciones.codh = inicio(declaraciones.n, declaraciones.dir) ++
ir_a(declaraciones.etq)
declaraciones.etqh = **longInicio** + 1
acciones.codh = declaraciones.cod
acciones.etqh = declaraciones.etq
programa.cod = acciones.cod ++ stop
programa ≡ acciones
acciones.codh = Ø
acciones.etqh = 0 + 1
programa.cod = acciones.cod ++ stop
declaraciones ≡ declaraciones declaracion
$declaraciones_1$.etqh = $declaraciones_0$.etqh
$declaraciones_1$.codh = $declaraciones_0$.codh
declaracion.etqh = $declaraciones_1$.etq
declaracion.codh = $declaraciones_1$.cod
$declaraciones_0$.cod = declaracion.cod
$declaraciones_0$.etq = declaracion.etq
declaraciones ≡ declaracion
declaracion.codh = declaraciones.codh
declaracion.etqh = declaraciones.etqh
declaraciones.cod = declaracion.cod
declaraciones.etq = declaracion.etq
declaracion ≡ comentario
declaracion.cod = declaracion.codh
declaracion.etq = declaracion.etqh
declaracion ≡ declaracionvar
declaracionvar.codh = declaracion.codh
declaracionvar.etqh = declaracion.etqh
declaracion.cod = declaracionvar.cod
declaracion.etq = declaracionvar.etq
declaracion ≡ declaraciontipo
declaraciontipo.codh = declaracion.codh
declaraciontipo.etqh = declaracion.etqh
declaracion.cod = declaraciontipo.cod
declaracion.etq = declaraciontipo.etq

declaracion ≡ declaracionfun

declaracionfun.codh = declaracion.codh
declaracionfun.etqh = declaracion.etqh
declaracion.cod = declaracionfun.cod
declaracion.etq = declaracionfun.etq
declaracion.propsop = declaracionfun.propsop

declaracionvar ≡ desctipo id ;

desctipo.codh = declaracionvar.codh
desctipo.etqh = declaracionvar.etqh
declaracionvar.cod = desctipo.cod
declaracionvar.etq = desctipo.etq

declaraciontipo ≡ tipo deftipo id ;

deftipo.codh = declaraciontipo.codh
deftipo.etqh = declaraciontipo.etqh
deftipo.cod = declaraciontipo.cod
deftipo.etq = declaraciontipo.etq

declaracionfun ≡ fun id (listaparametros) tiporeturn cuerpo end id ;

listaparametros.codh = declaracionfun.codh
listaparametros.etqh = declaracionfun.etqh
tiporeturn.codh = listaparametros.cod
tiporeturn.etqh = listaparametros.etq
cuerpo.codh = tiporeturn.cod
cuerpo.etqh = tiporeturn.etq
declaracionfun.etq = cuerpo.etq
declaracionfun.cod = cuerpo.cod
declaracionfun.propsop = <inicio:Cuerpo.inicio>

listaparametros ≡ parametros

parametros.codh = listaparametros.codh
parametros.etqh = listaparametros.etqh
listaparametros.cod = parametros.cod
listaparametros.etq = parametros.etq

listaParametros ≡ λ

listaparametros.cod = listaparametros.codh
listaparametros.etq = listaparametros.etqh

parametros ≡ parametros , parametro

$parametros_1$.codh = $parametros_0$.codh
$parametros_1$.etqh = $parametros_0$.etqh
parametro.codh = $parametros_1$.cod
parametro.etqh = $parametros_1$.etq
$parametros_0$.cod = parametro.cod
$parametros_0$.etq = parametro.etq

parametros ≡ parametro

parametro.codh = parametros.codh
parametro.etqh = parametros.etqh
parametros.cod = parametro.cod
parametros.etq = parametro.etq

parametro ≡ desctipo id

desctipo.codh = parametro.codh
desctipo.etqh = parametro.etqh
parametro.cod = desctipo.cod
parametro.etq = desctipo.etq

parametro ≡ desctipo & id

desctipo.codh = parametro.codh
desctipo.etqh = parametro.etqh
parametro.cod = desctipo.cod
parametro.etq = desctipo.etq

tiporeturn ≡ returns desctipo

desctipo.codh = tiporeturn.codh
desctipo.etqh = tiporeturn.etqh
tiporeturn.cod = desctipo.cod
tiporeturn.etq = desctipo.etq

tiporeturn ≡ λ

tiporeturn.cod = tiporeturn.codh
tiporeturn.etq = tiporeturn.etqh

cuerpo ≡ declaraciones acciones

declaraciones.codh = cuerpo.codh
declaraciones.etqh = cuerpo.etqh
cuerpo.inicio = declaraciones.etqh
acciones.etqh = declaraciones.etq + longPrologo
acciones.codh = declaraciones.cod ++ prologo(cuerpo.nh, declaraciones.dir)
cuerpo.etq = acciones.etq + longEpilogo + 1
cuerpo.cod = acciones.cod ++ epilogo(cuerpo.nh) ++ ir-ind

cuerpo ≡ acciones

acciones.codh = cuerpo.codh
acciones.etqh = cuerpo.etqh
cuerpo.cod = acciones.cod ++ epilogo(cuerpo.nh) ++ ir-ind
cuerpo.inicio = cuerpo.etqh
cuerpo.etq = acciones.etq + longEpilogo + 1

desctipo ≡ id

desctipo.cod = desctipo.codh
desctipo.etq = desctipo.etqh

desctipo ≡ int

desctipo.cod = desctipo.codh
desctipo.etq = desctipo.etqh

desctipo ≡ real

desctipo.cod = desctipo.codh
desctipo.etq = desctipo.etqh

deftipo ≡ desctipo [litInt]

desctipo.codh = deftipo.codh
desctipo.etqh = deftipo.etqh
deftipo.cod = desctipo.cod
deftipo.etq = desctipo.etq

deftipo ≡ rec campos endrec

campos.etqh = deftipo.etqh
campos.codh = deftipo.codh
deftipo.cod = campos.cod
deftipo.etq = campos.etq

deftipo ≡ pointer desctipo

desctipo.codh = deftipo.codh
desctipo.etqh = deftipo.etqh
deftipo.cod = desctipo.cod
deftipo.etq = desctipo.etq

campos ≡ campos campo

$campos_1$.codh = $campos_0$.codh

$campos_1$.etqh = $campos_0$.etqh
campo.codh = $campos_1$.cod
campo.etq = $campos_1$.etq

campos ≡ campo ,

campo.codh = campos.codh
campo.etqh = campos.etqh
campos.cod = campo.cod
campos.etq = campo.etq

campo ≡ desctipo id

desctipo.codh = campo.codh
desctipo.etqh = campo.etqh
campo.cod = desctipo.cod
campo.etq = desctipo.etq

acciones ≡ acciones accion

$acciones_1$.etqh = $acciones_0$.etqh
$acciones_1$.codh = $acciones_0$.codh
accion.etqh = $acciones_1$.etq
accion.codh = $acciones_1$.cod
$acciones_0$.etq = accion.etq
$acciones_0$.cod = accion.cod

acciones ≡ accion

accion.etqh = acciones.etqh
accion.codh = acciones.codh
acciones.etq = accion.etq
acciones.cod = accion.cod

accion ≡ comentario

accion.cod = accion.codh
accion.etq = accion.etqh

accion ≡ accionbasica

accion.cod = accionbasica.cod
accion basica.etqh = accion.etqh
accion.etq = accionbasica.etq

accion ≡ accionreturn

accionteturn.codh = accion.codh
accionreturn.etqh = accion.etqh
accion.cod = accionreturn.cod
accion.etq = accionreturn.etq

accion ≡ accionalternativa

accionalternativa.codh = accion.codh
accionalternativa.etqh = accion.etqh
accion.cod = accionalternativa.cod
accion.etq = accionalternativa.etq

accion ≡ accioniteracion

accioniteracion.codh = accion.codh
accioniteracion.etqh = accion.etqh
accion.cod = accioniteracion.cod
accion.etq = accioniteracion.etq

accion ≡ accionreserva

accionreserva.codh = accion.codh
accionreserva.etqh = accion.etqh
accion.cod = accionreserva.cod
accion.etq = accionreserva.etq

accion ≡ accionlibera

accionlibera.codh = accion.codh
accionlibera.etqh = accion.etqh
accion.cod = accionlibera.cod
accion.etq = accionlibera.etq

accion ≡ accioninvoca

accioninvoca.codh = accion.codh
accioninvoca.etqh = accion.etqh
accion.cod = accioninvoca.cod
accion.etq = accioninvoca.etq

mem ≡ id

mem.cod = mem.codh ++ apila(mem.tsh[id.lex].dir)
mem.etq = mem.etqh + 1

mem ≡ mem [expresion2]

mem_1.codh = mem_0.codh
mem_1.etqh = mem_0.etqh
expresion2.codh = mem_1.cod
expresion2.etqh = mem_1.etq
mem_0.cod = expresion2.cod ++ apila (mem_1.tipo.tbase.tam) ++ multiplica ++ suma
mem_0.etq = exresion2.etq + 3

mem ≡ mem.id

mem_1.codh = mem0.codh
mem_1.etqh = mem_0.etqh
mem_0.cod = mem_1.cod ++ apila(mem_1.tipo.campos[id.lex].desp) ++ suma
mem_0.etq = mem_1.etq + 2

mem ≡ mem^

mem_1.codh = mem_0.codh
mem_1.etqh = mem_0.etqh
mem_0.cod = mem_1.cod ++ apila-ind
mem_0.etq = mem_1.etq + 1

accionbasica ≡ expresion ;

expresion.codh = accionbasica.codh
expresion.etqh = accionbasica.etqh
accionbasica.cod = expresion.cod
accionbasica.etq = expresion.etq

accionreturn ≡ returns expresion2 ;

expresion2.codh = accionreturn.codh
expresion2.etqh = accionreturn.etqh
accionreturn.etq = expresion2.etq
accionreturn.cod = expresion2.cod

accionalternativa ≡ if expresion2 then bloque accionelse endif ;

expresion2.etqh = accionalternativa.etqh
expresion2.codh = accionalternativa.codh
bloque.etqh = expresion2.etq + 1
bloque.codh = expresion2.cod ++ ir-f(bloque.etq + 1)
accionelse.etqh = bloque.etq + 1
accionelse.codh = bloque.cod ++ ir-a(accionelse.etq)
accionalternativa.etq = accionelse.etq
accionalternativa.cod = accionelse.cod

accionelse ≡ elsif expresion2 then bloque accionelse

expresion2.etqh = $accionelse_0$.etqh
expresion2.codh = $accionelse_0$.codh

bloque.etqh = expresion2.etq + 1
bloque.codh = expresion2.cod ++ ir-f(bloque.etq + 1)
$accionelse_1$.etqh = bloque.etq + 1
$accionelse_1$.codh = bloque.cod ++ ir-a($accionelse_1$.etq)
$accionelse_0$.etq = $accionelse_1$.etq
$accionelse_0$.cod = $accionelse_1$.cod

accionelse ≡ else bloque

bloque.etqh = accionelse.etqh
bloque.codh = accionelse.codh
accionelse.etq = bloque.etq
accionelse.cod = bloque.cod

accionelse ≡ λ

accionelse.cod = accionelse.codh
accionelse.etq = accionelse.etqh

bloque ≡ acciones

acciones.codh = bloque.codh
acciones.etqh = bloque.etqh
bloque.etq = acciones.etq
bloque.cod = acciones.cod

bloque ≡ λ

bloque.cod = bloque.codh
bloque.etq = bloque.etqh

accioniteracion ≡ while expresion2.cod do bloque endwhile ;

expresion2.etqh = accioniteracion.etqh
expresion2.codh = accioniteracion.codh
bloque.etqh = expresion2.etq + 1
bloque.codh = expresion2.cod ++ ir-f(bloque.etq + 1)
accioniteracion.etq = bloque.etq + 1
accioniteracion.cod = bloque.cod ++ ir-a(accioniteracion.etqh)

accionreserva ≡ alloc mem

mem.etqh = accionreserva.etqh
mem.codh = accionreserva.codh
accionreserva.etq = mem.etq + 2
accionreserva.cod = mem.cod ++ new(
SI mem.tipo.t == Referencia ENTONCES
accionreserva.tsh[mem.tipo.tbase.id].tam
SI NO
1
) ++ desapila-ind

accionlibera ≡ free mem

mem.etqh = accionlibera.etqh
mem.codh = accionlibera.codh
accionlibera.etq = mem.etq + 1
accionlibera.cod = mem.cod ++ delete(
SI mem.tipo.t == Referencia ENTONCES
accionreserva.tsh[mem.tipo.tbase.id].tam
SI NO
1
)

accioninvoca ≡ id (Aparams)

Aparams.etqh = accioninvoca.etqh + longApilaRet
Aparams.codh = accioninvoca.codh ++ apila_ret(parchea(accioninvoca.etq))

accioninvoca.etq = expresiones.etq + 1
accioninvoca.cod = Aparams.cod ++ ir-a(accioninvoca.tsh[id.lex].inicio)

Aparams ≡ expresiones

expresiones.etqh = Aparams.etqh + longInicioPaso
expresiones.codh = Aparams.codh ++ inicio-paso
Aparams.etq = expresiones.etq + longFinPaso
Aparams.cod = expresiones.cod ++ fin-paso

Aparams ≡ λ

Aparams.cod = Aparams.codh
Aparams.etq = Aparams.etqh

expresiones ≡ expresiones , expresion2

$expresiones_1$.etqh = $expresiones_0$.etqh
$expresiones_1$.codh = $expresiones_0$.codh
expresion2.etqh = $expresiones_1$.etq + 1
expresion2.codh = $expresiones_1$.cod ++ copia
$expresiones_0$.etq = expresion2.etq +
longDireccionParFormal($expresiones_0$.fparams[$expresiones_0$.nparams])
+ longPasoParametro(expresion2.modo,
$expresiones_0$.fparams[$expresiones_0$.nparams])
$expresiones_0$.cod = expresion2.cod ++
direccionParFormal($expresiones_0$.fparams[$expresiones_0$.nparams]) ++
pasoParametro(expresion2.modo,$expresiones_0$.fparams[$expresiones_0$.nparams
])

expresiones ≡ expresion2

expresion2.etqh = expresiones.etqh + 1
expresion2.codh = expresiones.codh ++ copia
expresiones.etq = expresion2.etq + longPasoParametro(expresion2.modo,
expresiones.fparams[1])
expresiones.cod = expresion2.cod ++ pasoParametro(expresion2.modo,
expresiones.fparams[1])

expresion ≡ op0in mem

op0in.etqh = expresion.etqh
op0in.codh = expresion.codh
mem.etqh = op0in.etq
mem.codh = op0in.cod
expresion.etq = mem.etq + 1
expresion.cod = mem.cod ++ op0in.op

expresion ≡ op0out expresion1

op0out.etqh = expresion.etqh
op0out.codh = expresion.codh
expresion2.etqh = op0out.etq
expresion2.codh = op0out.cod
expresion.etq = expresion2.etq + 1
expresion.cod = expresion2.cod ++ op0out.op

expresion ≡ expresion1

expresion1.etqh = expresion.etqh
expresion1.codh = expresion.codh
expresion.etq = expresion1.etq
expresion.cod = expresion1.cod

expresion1 ≡ mem op1 expresion2

mem.etqh = expresion1.etqh
mem.codh = expresion1.codh

expresion2.etqh = mem.etq
expresion2.codh = mem.cod
expresion1.etq = expresion2.etq + 1
expresion1.parh = false
expresion1.cod = expresion2.cod ++ mueve(mem.tipo.tam)

expresion1 ≡ expresion2

expresion2.etqh = expresion1.etqh
expresion2.codh = expresion1.codh
expresion1.etq = expresion2.etq
expresion1.cod = expresion2.cod

expresion2 ≡ expresion3 op2 expresion3

$expresion3_0$.etqh = expresion2.etqh
$expresion3_0$.codh = expresion2.codh
$expresion3_1$.etqh = $expresion3_0$.etq
$expresion3_1$.codh = $expresion3_0$.cod
expresion2.etq = $expresion3_1$.etq + 1
expresion2.cod = $expresion3_1$.cod ++ op2.op

expresion2 ≡ expresion3

expresion3.etqh = expresion2.etqh
expresion3.codh = expresion2.codh
expresion2.etq = expresion3.etq
expresion2.cod = expresion3.cod

expresion3 ≡ expresion3 op3 expresion4

$expresion3_1$.etqh = $expresion3_0$.etqh
$expresion3_1$.codh = $expresion3_0$.codh
op3.etqh = $expresion3_1$.etq
op3.codh = $expresion3_1$.cod
expresion4.etqh = op3.etq
expresion4.codh = op3.cod
$expresion3_0$.etq = expresion4.etq +
SI op3.op == || ENTONCES
3
SI NO
1
$expresion3_0$.cod = expresion4.cod ++
SI op3.op == || ENTONCES
copia ++ ir-v(expresion4.etq) ++ desapila
SI NO
op3.op

expresion3 ≡ expresion4

expresion4.etqh = expresion3.etqh
expresion4.codh = expresion3.codh
expresion3.etq = expresion4.etq
expresion3.cod = expresion4.cod

expresion4 ≡ expresion4 op4 expresion5

$expresion4_1$.etqh = $expresion4_0$.etqh
$expresion4_1$.codh = $expresion4_0$.codh
op4.etqh = $expresion4_1$.etq
op4.codh = $expresion4_1$.codh
expresion5.etqh = op4.etq + op4.op == && ENTONCES 1
expresion5.codh = op4.cod ++ SI op4.op == && ENTONCES ir-f(expresion5.etq + 1)
$expresion4_0$.etq = expresion5.etq +

SI op4.op == && ENTONCES 2
SI NO 1
expresion4$_0$.cod = expresion5.cod ++
SI op4.op == && ENTONCES
ir-a(expresion5.etq + 2) ++ apila(0)
SI NO
op4.op

expresion4 ≡ expresion5

expresion5.etqh = expresion4.etqh
expresion5.codh = expresion4.codh
expresion4.etq = expresion5.etq
expresion4.cod = expresion5.cod

expresion5 ≡ op5asoc expresion5

op5asoc.etqh = expresion5$_0$.etqh
op5asoc.codh = expresion5$_0$.codh
expresion5$_1$.codh = op5asoc.codh
expresion5$_1$.etqh = op5asoc.etq
expresion5$_0$.etq = expresion5$_1$.etq + 1
expresion5$_0$.cod = expresion5$_1$.cod ++ op5asoc.op

expresion5 ≡ op5noasoc expresion6

op5noasoc.etqh = expresion5.etqh
op5noasoc.codh = expresion5.codh
expresion6.etqh = op5noasoc.etqh
expresion6.codh = op5noasoc.codh
expresion5.etq = expresion6.etq + 1
expresion5.cod = expresion6.cod ++ op5noasoc.op

expresion5 ≡ expresion6

expresion6.etqh = expresion5.etqh
expresion6.codh = expresion5.codh
expresion5.etq = expresion6.etq
expresion5.cod = expresion6.cod

expresion6 ≡ (expresion2)

expresion2.etqh = expresion6.etqh
expresion2.codh = expresion6.codh
expresion6.etq = expresion2.etq
expresion6.cod = expresion2.cod

expresion6 ≡ litInt

expresion6.etq = expresion6.etqh + 1
expresion6.cod = expresion6.codh ++ apila(litInt.lex)

expresion6 ≡ litReal

expresion6.etq = expresion6.etqh + 1
expresion6.cod = expresion6.codh ++ apila(litReal.lex)

expresion6 ≡ null

expresion6.etq = expresion6.etqh + 1
expresion6.cod = expresion6.codh ++ apila(MIN_ENTERO)

expresion6 ≡ mem

expresion6.etq = expresion6.etqh + 1
expresion6.cod = expresion6.codh ++ apila_dir(expresion6.tsh[mem.id].dir)

op1 ≡ =

op1.cod = op1.codh
op1.etq = op1.etqh

op2 ≡ <

op2.cod = op2.codh
op2.etq = op2.etqh

op2 ≡ >

op2.cod = op2.codh
op2.etq = op2.etqh

op2 ≡ <=

op2.cod = op2.codh
op2.etq = op2.etqh

op2 ≡ >=

op2.cod = op2.codh
op2.etq = op2.etqh

op2 ≡ ==

op2.cod = op2.codh
op2.etq = op2.etqh

op2 ≡ !=

op2.cod = op2.codh
op2.etq = op2.etqh

op3 ≡ ||

op3.cod = op3.codh
op3.etq = op3.etqh

op3 ≡ +

op3.cod = op3.codh
op3.etq = op3.etqh

op3 ≡ -

op3.cod = op3.codh
op3.etq = op3.etqh

op4 ≡ *

op4.cod = op4.codh
op4.etq = op4.etqh

op4 ≡ /

op4.cod = op4.codh
op4.etq = op4.etqh

op4 ≡ %

op4.cod = op4.codh
op4.etq = op4.etqh

op4 ≡ &&

op4.cod = op4.codh
op4.etq = op4.etqh

op5asoc ≡ -

op5asoc.cod = op5asoc.codh
op5asoc.etq = op5asoc.etqh

op5asoc ≡ !

op5asoc.cod = op5asoc.codh
op5asoc.etq = op5asoc.etqh

op5noasoc ≡ (int)

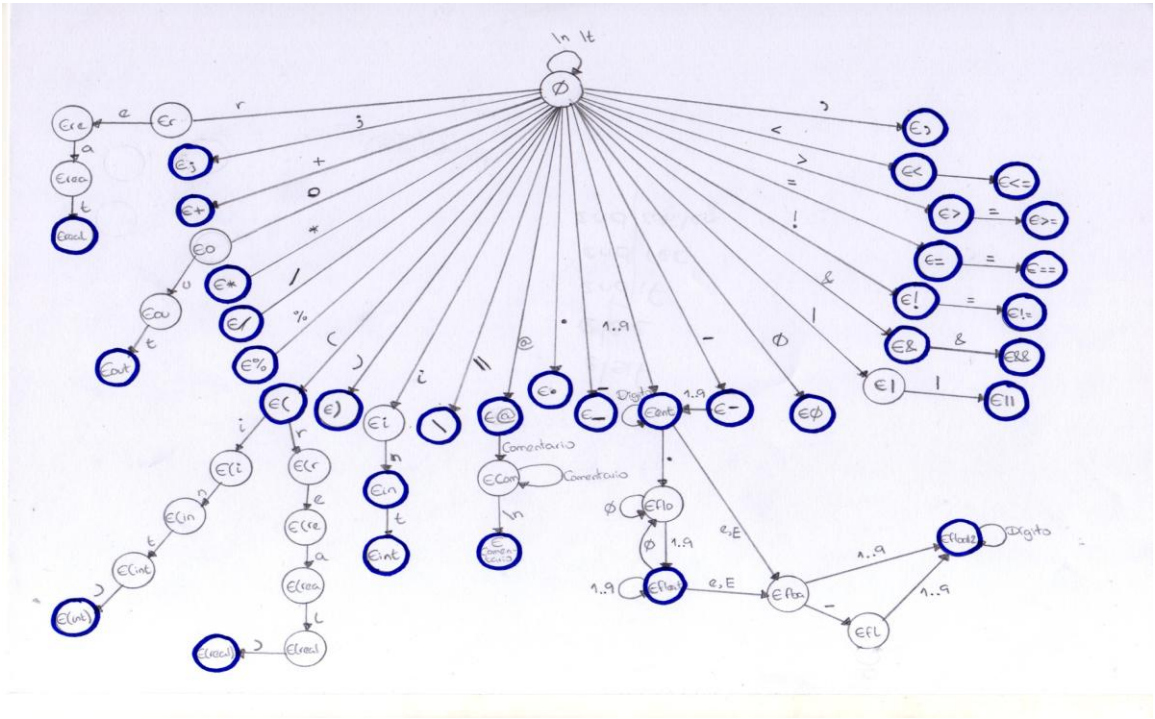
op5noasoc.cod = op5noasoc.codh
op5noasoc.etq = op5noasoc.etqh

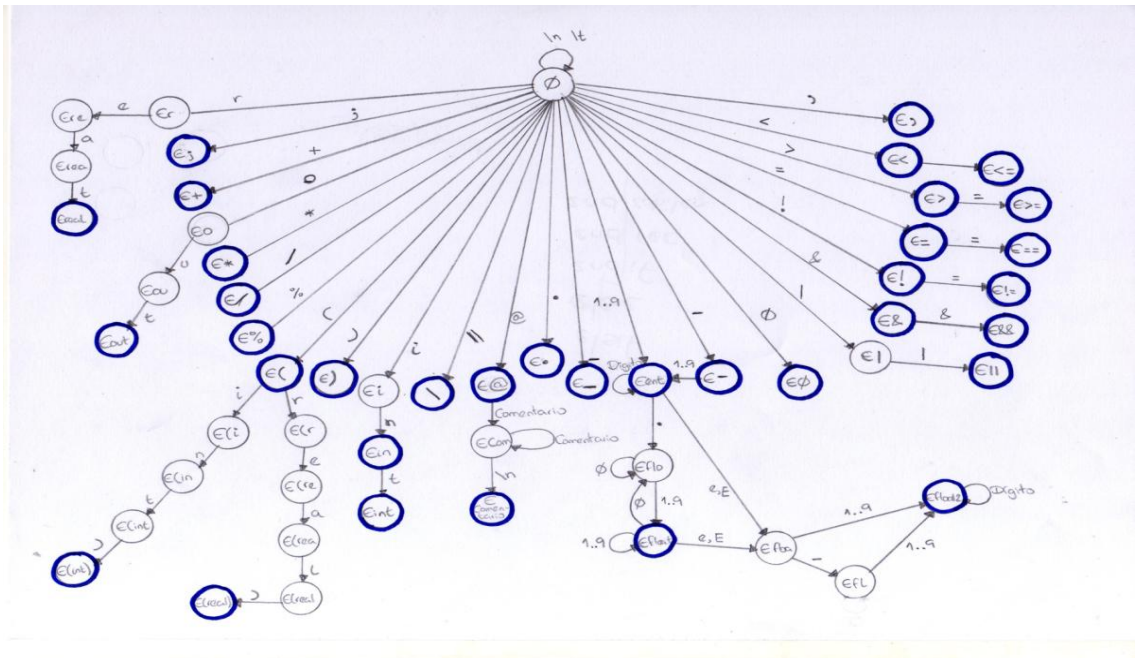
op5noasoc ≡ (real)

op5noasoc.cod = op5noasoc.codh
op5noasoc.etq = op5noasoc.etqh

6. Diseño del Analizador Léxico

Diagrama de transición que caracterice el diseño del analizador léxico. La implementación del analizador léxico debe estar guiada por este diseño





Se adjuntan ambas imagenes para su correcta visualización

7. Acondicionamiento de las gramáticas de atributos

Transformaciones realizadas sobre las gramáticas de atributos para permitir la traducción.

Únicamente deben incluirse la transformación de las producciones que se ven afectadas. Si alguna de las gramáticas no necesitan acondicionamiento, dejar el correspondiente subapartado en blanco.

7.1. Acondicionamiento de la Gramática para la Construcción de la tabla de símbolos

7.1.1. Acondicionamiento de: declaraciones ≡ declaraciones declaracion
declaraciones ≡ declaracion

declaraciones ≡ declaracion declaracionesRE
declaracion.tsh = declaraciones.tsh
declaracion.nh = declaraciones.nh
declaracion.dirh = declaraciones.dirh
declaracionesRE.tsh = declaracion.ts
declaracionesRE.nh = declaracion.n
declaracionesRE.dirh = declaracion.dir
declaraciones.ts = declaracionesRE.ts
declaraciones.n = declaracionesRE.n
declaraciones.dir = declaracionesRE.dir
declaracionesRE ≡ declaracion declaracionesRE
declaracion.tsh = $declaracionesRE_0$.tsh
declaracion.nh = $declaracionesRE_0$.nh
declaracion.dirh = $declaracionesRE_0$.dirh
$declaracionesRE_1$.tsh = declaracion.ts
$declaracionesRE_1$.nh = declaracion.n
$declaracionesRE_1$.dirh = declaracion.dir
$declaracionesRE_0$.ts = $declaracionesRE_1$.ts
$declaracionesRE_0$.n = $declaracionesRE_1$.n
$declaracionesRE_0$.dir = $declaracionesRE_1$.dir
declaracionesRE ≡ λ
declaracionesRE.ts = declaracionesRE.tsh
declaracionesRE.n = declaracionesRE.nh
declaracionesRE.dir = declaracionesRE.dirh

7.1.2. Acondicionamiento de: parametros ≡ parametros , parámetro
parametros ≡ parámetro

parametros ≡ parametro parametrosRE
parametro.tsh = parametros.tsh
parametro.nh = parametros.nh
parametro.dirh = parametros.dirh
parametro.paramsh = parametros.paramsh
parametrosRE.tsh = parametro.ts

parametrosRE.nh = parametro.n
parametrosRE.dirh = parametro.dir
parametrosRE.paramsh = parametro.params
parametros.ts = parametrosRE.ts
parametros.n = parametrosRE.n
parametros.dir = parametrosRE.dir
parametros.params = parametrosRE.params

parametrosRE ≡ , parametro parametrosRE

parametro.tsh = $parametrosRE_0$.tsh
parametro.nh = $parametrosRE_0$.nh
parametro.dirh = $parametrosRE_0$.dirh
parametro.paramsh = $parametrosRE_0$.paramsh
$parametrosRE_1$.tsh = parametro.ts
$parametrosRE_1$.nh = parametro.n
$parametrosRE_1$.dirh = parametro.dir
$parametrosRE_1$.paramsh = parametro.params
$parametrosRE_0$.ts = $parametrosRE_1$.ts
$parametrosRE_0$.n = $parametrosRE_1$.n
$parametrosRE_0$.dir = $parametrosRE_1$.dir
$parametrosRE_0$.params = $parametrosRE_1$.params

parametrosRE ≡ λ

parametros.ts = parametrosRE.tsh
parametros.n = parametrosRE.nh
parametros.dir = parametrosRE.dirh
parametros.params = parametrosRE.paramsh

7.1.3. Acondicionamiento de: campos ≡ campos campo
campos ≡ campo ,

campos ≡ campo camposRE

campo.tsh = campos.tsh
campo.nh = campos.nh
campo.dirh = campos.dirh
campo.camposh = campos.camposh
campo.tamh = campos.tamh
campo.desph = campos.desph
camposRE.tsh = campo.ts
camposRE.nh = campo.n
camposRE.dirh = campo.dir
camposRE.camposh = campo.campos
camposRE.tamh = campo.tam
camposRE.desph = camposRE.desp
campos.ts = camposRE.ts
campos.n = camposRE.n
campos.dir = camposRE.dir
campos.campos = camposRE.campos
campos.tam = camposRE.tam
campos.desp = camposRE.desp

camposRE ≡ , campo camposRE

campo.tsh = $camposRE_0$.tsh

campo.nh = $camposRE_0$.nh
campo.dirh = $camposRE_0$.dirh
campo.camposh = $camposRE_0$.camposh
campo.tamh = $camposRE_0$.tamh
campo.desph = $camposRE_0$.desph
$camposRE_1$.tsh = campo.ts
$camposRE_1$.nh = campo.n
$camposRE_1$.dirh = campo.dir
$camposRE_1$.camposh = campo.campos
$camposRE_1$.tamh = campo.tam
$camposRE_1$.desph = campo.desp
$camposRE_0$.ts = $camposRE_1$.ts
$camposRE_0$.n = $camposRE_1$.n
$camposRE_0$.dir = $camposRE_1$.dir
$camposRE_0$.campos = $camposRE_1$.campos
$camposRE_0$.tam = $camposRE_1$.tam
$camposRE_0$.desph = $camposRE_1$.desp

camposRE ≡ λ

camposRE.ts = camposRE.tsh
camposRE.n = camposRE.nh
camposRE.dir = camposRE.dirh
camposRE.campos = camposRE.camposh
camposRE.tam = camposRE.tamh
camposRE.desph = camposRE.desph

7.1.4. Acondicionamiento de: acciones ≡ acciones accion
acciones ≡ accion

acciones ≡ accion accionesRE

accion.tsh = acciones.tsh
accion.nh = acciones.nh
accionesRE.tsh = accion.ts
accionesRE.nh = accion.n
acciones.ts = accionesRE.ts
acciones.n = accionesRE.n

accionesRE ≡ accion accionesRE

accion.tsh = $accionesRE_0$.tsh
accion.nh = $accionesRE_0$.nh
$accionesRE_1$.tsh = accion.ts
$accionesRE_1$.nh = accion.n
$accionesRE_1$.errorh = accion.error
$accionesRE_0$.ts = $accionesRE_1$.ts
$accionesRE_0$.n = $accionesRE_1$.n

accionesRE ≡ λ

accionesRE.ts = accionesRE.tsh
accionesRE.n = accionesRE.nh

7.1.5. Acondicionamiento de: mem ≡ id
mem ≡ mem [expresion2]
mem ≡ mem.id
mem ≡ mem^

mem ≡ id memRE

memRE.tsh = mem.ts
memRE.nh = mem.nh
mem.ts = memRE.ts
mem.n = memRE.n
mem.id = id.lex

memRE ≡ [expresion2] memRE

expresion2.tsh = $memRE_0$.tsh
expresion2.nh = $memRE_0$.nh
$memRE_1$.tsh = expresion2.ts
$memRE_1$.nh = expresion2.n
$memRE_0$.ts = $memRE_1$.ts
$memRE_0$.n = $memRE_1$.n

memRE ≡ . id memRE

$memRE_1$.tsh = $memRE_0$.tsh
$memRE_1$.nh = $memRE_0$.nh
$memRE_0$.ts = $memRE_1$.ts
$memRE_0$.n = $memRE_1$.n

memRE ≡ ^ memRE

$memRE_1$.tsh = $memRE_0$.tsh
$memRE_1$.nh = $memRE_0$.nh
$memRE_0$.ts = $memRE_1$.ts
$memRE_0$.n = $memRE_1$.n

memRE ≡

memRE.ts = memRE.tsh
memRE.n = memRE.nh

7.1.6. Acondicionamiento de expresiones ≡ expresiones , expresion2 expresiones ≡ expresion2

expresiones ≡ expresion2 expresionesRE

expresion2.tsh = expresiones.tsh
expresion2.nh = expresiones.nh
expresion2.nparamsh = expresiones.nparamsh
expresion2.paramsh = expresiones.paramsh
expresionesRE.tsh = expresion2.ts
expresionesRE.nh = expresion2.n
expresionesRE.nparamsh = expresion2.nparams
expresionesRE.paramsh = expresion2.params
expresiones.ts = expresionesRE.ts
expresiones.n = expresionesRE.n
expresiones.nparams = expresionesRE.nparams
expresiones.params = expresionesRE.params

expresionesRE ≡ , expresion2 expresionesRE

expresion2.tsh = $expresionesRE_0$.tsh
expresion2.nh = $expresionesRE_0$.nh
expresion2.nparamsh = $expresionesRE_0$.nparamsh
expresion2.paramsh = $expresionesRE_0$.paramsh
$expresionesRE_1$.tsh = expresion2.ts
$expresionesRE_1$.nh = expresion2.n
$expresionesRE_1$.nparamsh = expresion2.nparams
$expresionesRE_1$.paramsh = expresion2.params

expresionesRE$_0$.ts = expresionesRE$_1$.ts
expresionesRE$_0$.n = expresionesRE$_1$.n
expresionesRE$_0$.nparams = expresionesRE$_1$.nparams
expresionesRE$_0$.params = expresionesRE$_1$.params

expresionesRE ≡ λ

expresionesRE.ts = expresionesRE.tsh
expresionesRE.n = expresionesRE.nh
expresionesRE.nparams = expresionesRE.nparamsh
expresionesRE.params = expresionesRE.paramsh

7.1.7. Acondicionamiento de: expresion2 ≡ expresion3 op2 expresion3 / expresion2 ≡ expresion3

expresion2 ≡ expresion3 expresion2RE

expresion3.tsh = expresion2.tsh
expresion3.nh = expresion2.nh
expresion3.nparamsh = expresion2.nparamsh
expresion3.paramsh = expresion2.paramsh
expresion2RE.tsh = expresion3.ts
expresion2RE.nh = expresion3.n
expresion2RE.nparamsh = expresion3.nparams + 1
expresion2RE.paramsh = expresion2.params ++ <modo:expresion3.modo, tipo: expresion3.tipo>
expresion2RE.tipo = expresion3.tipo
expresion2RE.modoh = expresion3.modo
expresion2.ts = expresion2RE.ts
expresion2.n = expresion2RE.n
expresion2.nparams = expresion2RE.nparams
expresion2.params = expresion2RE.params
expresion2.tipo = expresion2RE.tipo
expresion2.modo = expresion2RE.modo

expresion2RE ≡ op2 expresion3 expresion2RE

op2.tsh = expresion2RE$_0$.tsh
op2.nh = expresion2RE$_0$.nh
expresion3.tsh = op2.ts
expresion3.nh = op2.n
expresion2RE$_1$.tsh = expresion3.ts
expresion2RE$_1$.n = expresion3.n
expresion2RE$_0$.ts = expresion2RE$_1$.ts
expresion2RE$_0$.n = expresion2RE$_1$.n
expresion2RE$_0$.tipo = expresion2RE$_1$.tipo
expresion2RE$_0$.modo = valor

expresion2RE ≡ λ

expresion2RE.ts = expresion2RE.tsh
expresion2RE.n = expresion2RE.nh
expresion2RE.tipo = expresion2RE.tipoh
expresion2RE.modo = expresion2RE.modoh

7.1.8. Acondicionamiento de: expresion3 ≡ expresion3 op3 expresion4 / expresion3 ≡ expresion4

expresion3 ≡ expresion4 expresion3RE

expresion4.tsh = expresion3.tsh

expresion4.nh = expresion3.nh
expresion3RE.tsh = expresion4.ts
expresion3RE.nh = expresion4.n
expresion3RE.tipoh = expresion4.tipo
expresion3RE.modo = expresion4.modo
expresion3.ts = expresion3RE.ts
expresion3.n = expresion3RE.n
expresion3.tipo = expresion3RE.tipo
expresion3.modo = expresion3RE.modo

expresion3RE ≡ op3 expresion4 expresion3RE

op3.tsh = expresion3RE$_0$.tsh
op3.nh = expresion3RE$_0$.nh
expresion4.tsh = op3.ts
expresion4.nh = op3.n
expresion3RE$_1$.tsh = expresion4.ts
expresion3RE$_1$.nh = expresion4.n
expresion3RE$_1$.tipoh = expresion4.tipo
expresion3RE$_0$.tipo = devuelveTipo(op3.op, expresion3RE$_0$.tipoh, expresion3RE$_1$.tipo)
expresion3RE$_0$.modo = valor

expresion3RE ≡ λ

expresion3RE.ts = expresion3RE.tsh
expresion3RE.n = expresion3RE.nh
expresion3RE.tipo = expresion3RE.tipoh

7.1.9. Acondicionamiento de: expresion4 ≡ expresion4 op4 expresion5 expresion4 ≡ expresion5

expresion4 ≡ expresion5 expresion4RE

expresion5.tsh = expresion4.tsh
expresion5.nh = expresion4.nh
expresion4RE.tsh = expresion5.ts
expresion4RE.nh = expresion5.n
expresion4RE.tipoh = expresion5.tipo
expresion4RE.modo = expresion5.modo
expresion4.ts = expresion4RE.ts
expresion4.n = expresion4RE.n
expresion4.tipo = expresion4RE.tipo
expresion4.modo = expresion4RE.modo

expresion4RE ≡ op4 expresion5 expresion4RE

op4.tsh = expresion4RE$_0$.tsh
op4.nh = expresion4RE$_0$.nh
expresion5.tsh = op4.ts
expresion5.nh = op4.n
expresion4RE$_1$.tsh = expresion5.ts
expresion4RE$_1$.nh = expresion5.n
expresion4RE$_1$.tipoh = expresion5.tipo
expresion4RE$_0$.ts = expresion4RE1.ts
expresion4RE$_0$.n = expresion4RE1.n
expresion4RE$_0$.tipo = devuelveTipo(op4.op, expresion4RE$_0$.tipoh, expresion4RE$_1$.tipo)
expresion4RE$_0$.modo = valor

expresion4RE ≡ λ

expresion4RE.ts = expresion4RE.tsh
expresion4RE.n = expresion4RE.nh
expresion4RE.tipo = expresion4RE.tipoh
expresion4RE.modo = expresion4RE.modoh

7.2. Acondicionamiento de la Gramática para la Comprobación de las Restricciones Contextuales

7.2.1. Acondicionamiento de: declaraciones ≡ declaraciones declaracion
declaraciones ≡ declaracion

declaraciones ≡ declaracion declaracionesRE

declaracion.errorh = declaraciones.errorh
declaracion.pendh = declaraciones.pendh
declaracion.tbloqueh = declaraciones.tbloqueh
declaracionesRE.errorh = declaracion.error
declaracionesRE.pendh = declaracion.pend
declaracionesRE.tbloqueh = declaracion.tbloque
declaraciones.error = declaracionesRE.error
declaraciones.pend = declaracionesRE.pend
declaraciones.tbloque = declaracionesRE.tbloque

declaracionesRE ≡ declaracion declaracionesRE

declaracion.errorh = declaracionesRE$_0$.errorh
declaracion.pendh = declaracionesRE$_0$.pendh
declaracion.tbloqueh = declaracionesRE$_0$.tbloqueh
declaracionesRE$_1$.errorh = declaracion.error
declaracionesRE$_1$.pendh = declaracion.pend
declaracionesRE$_1$.tbloqueh = declaracion.tbloque
declaracionesRE$_0$.error = declaracionesRE$_1$.error
declaracionesRE$_0$.pend = declaracionesRE$_1$.pend
declaracionesRE$_0$.tbloque = declaracionesRE$_1$.tbloque

declaracionesRE ≡ λ

declaracionesRE.error = declaracionesRE.errorh
declaracionesRE.pend = declaracionesRE.pendh
declaracionesRE.tbloque = declaracionesRE.tbloqueh

7.2.2. Acondicionamiento de: parametros ≡ parametros , parámetro
parametros ≡ parámetro

parametros ≡ parametro parametrosRE

parametro.errorh = parametros.errorh
parametro.pendh = parametros.pendh
parametrosRE.errorh = parametro.error
parametrosRE.pendh = parametro.pend
parametros.error = parametrosRE.error
parametros.pend = parametrosRE.pend

parametrosRE ≡ , parametro parametrosRE

parametro.errorh = parametrosRE$_0$.errorh
parametro.pendh = parametrosRE$_0$.pendh

parametrosRE$_1$.errorh = parametro.error
parametrosRE$_1$.pendh = parametro.pend
parametrosRE$_0$.error = parametrosRE$_1$.error
parametrosRE$_0$.pend = parametrosRE$_1$.pend

parametrosRE ≡ λ

parametrosRE.error = parametrosRE.errorh
parametrosRE.pend = parametrosRE.pendh

7.2.3. Acondicionamiento de: campos ≡ campos campo / campos ≡ campo ,

campos ≡ campo camposRE

campo.errorh = campos.errorh
campo.pendh = campos.pendh
camposRE.errorh = campo.error
camposRE.pendh = campo.pend
campos.error = camposRE.error
campos.pend = camposRE.pend

camposRE ≡ , campo camposRE

campo.errorh = camposRE$_0$.errorh
campo.pendh = camposRE$_0$.pendh
camposRE$_1$.errorh = campo.error
camposRE$_1$.pendh = campo.pend
camposRE$_0$.error = camposRE$_1$.error
camposRE$_0$.pend = camposRE$_1$.pend

camposRE ≡ λ

camposRE.error = camposRE.errorh
camposRE.pend = camposRE.pendh

7.2.4. Acondicionamiento de: acciones ≡ acciones accion / acciones ≡ accion

acciones ≡ accion accionesRE

accion.errorh = acciones.errorh
accion.tbloqueh = acciones.tbloqueh
accionesRE.errorh = accion.error
accionesRE.tbloqueh = accion.tbloque
acciones.error = accionesRE.error
acciones.tbloque = accionesRE.tbloque

accionesRE ≡ accion accionesRE

accion.errorh = accionesRE$_0$.errorh
accion.tbloqueh = accionesRE$_0$.tbloqueh
accionesRE$_1$.errorh = accion.error
accionesRE$_1$.tbloqueh = accion.tbloque
accionesRE$_0$.error = accionesRE$_1$.error
accionesRE$_0$.tbloque = accionesRE$_1$.tbloque

accionesRE ≡ λ

accionesRE.error = accionesRE.errorh
accionesRE.tbloque = accionesRE.tbloqueh

7.2.5. Acondicionamiento de: mem ≡ id
mem ≡ mem [expresion2]
mem ≡ mem.id
mem ≡ mem^

mem ≡ id memRE

memRE.errorh = mem.errorh ∨ ¬ existeID(mem.tsh, id.lex) ∨ mem.tsh[id.lex].clase != var
mem.error = memRE.error

memRE ≡ [expresion2] memRE

expresion2.errorh = $memRE_0$.erroh ∨ $memRE_1$.tipo != array ∨ expresion2.tipo ! = <t:int,tam:1>
$memRE_1$.errorh = expresion2.error
$memRE_0$.error = $memRE_1$.error

memRE ≡ . id memRE

$memRE_1$.errorh = $memRE_0$.errorh ∨ $memRE_1$.tipo ! = registro ∨ ¬ existeCampo($memRE_1$.camposh,id.lex)
$memRE_0$.error = $memRE_1$.error

memRE ≡ ^ memRE

$memRE_1$.errorh = $memRE_0$.errorh ∨ $memRE_1$.tipo! =
$memRE_0$.error = $memRE_1$.error

memRE ≡ λ

memRE.error = memRE.errorh

7.2.6. Acondicionamiento de expresiones ≡ expresiones , expresion2
expresiones ≡ expresion2

expresiones ≡ expresion2 expresionesRE

expresion2.errorh = expresiones.errorh
expresionesRE.errorh = expresion2.error
expresiones.error = expresionesRE.error

expresionesRE ≡ , expresion2 expresionesRE

expresion2.errorh = $expresionesRE_0$.errorh
$expresionesRE_1$.errorh = expresion2.error
$expresionesRE_0$.error = $expresionesRE_1$.error

expresionesRE ≡ λ

expresionesRE.error = expresionesRE.errorh

7.2.7. Acondicionamiento de: expresion2 ≡ expresion3 op2 expresion3
expresion2 ≡ expresion3

expresion2 ≡ expresion3 expresion2RE

expresion3.errorh = expresion2.errorh
expresion2RE.errorh = expresion3.error
expresion2.error = expresion2RE.error

expresion2RE ≡ op2 expresion3 expresion2RE

op2.errorh = $expresion2RE_0$.errorh
expresion3.errorh = op2.error
$expresion2RE_1$.error = expresion3.error ∨ ¬ validoOperacion(expresion3.ts,
$expresion2RE_0$.tipoh,op2.op,expresion3.tipo)
$expresion2RE_0$.error = $expresion2RE_1$.error

expresion2RE ≡ λ

expresion2RE.error = expresion2RE.errorh

7.2.8. Acondicionamiento de: expresion3 ≡ expresion3 op3 expresion4
expresion3 ≡ expresion4

expresion3 ≡ expresion4 expresion3RE

expresion4.errorh = expresion3.errorh
expresion3RE.errorh = expresion4.error
expresion3.error = expresion3RE.error

expresion3RE ≡ op3 expresion4 expresion3RE

op3.errorh = $expresion3RE_0$.errorh
expresion4.errorh = op3.error
$expresion3RE_1$.errorh = expresion4.error ∨ ¬ validoOperacion
(expresion4.ts,$expresion3RE_0$.tipo, op3.op, expresion4.tipo)
$expresion3RE_0$.error = $expresion3RE_1$.error

expresion3RE ≡ λ

expresion3RE.error = expresion3RE.errorh

7.2.9. Acondicionamiento de: expresion4 ≡ expresion4 op4 expresion5
expresion4 ≡ expresion5

expresion4 ≡ expresion5 expresion4RE

expresion5.errorh = expresion4.errorh
expresion4RE.errorh = expresion5.error
expresion4.error = expresion4RE.error

expresion4RE ≡ op4 expresion5 expresion4RE

op4.errorh = $expresion4RE_0$.errorh
expresion5.errorh = op4.error
$expresion4RE_1$.errorh = expresion5.error ∨ ¬ validoOperacion(expresion5.ts,
$expresion4RE_0$.tipo, op4.op, expresion5.tipo)
$expresion4RE_0$.error = $expresion4RE_1$.error

expresion4RE ≡ λ

expresion4RE.error = expresion4RE.errorh

7.3. Acondicionamiento de la Gramática para la Traducción

7.3.1. Acondicionamiento de: declaraciones ≡ declaraciones declaracion
declaraciones ≡ declaracion

declaraciones ≡ declaracion declaracionesRE

declaracion.etqh = declaraciones.etqh
declaracion.codh = declaraciones.codh
declaracionesRE.codh = declaracion.cod
declaracionesRE.etqh = declaracion.etq
declaraciones.etq = declaracionesRE.etq
declaraciones.cod = declaracionesRE.cod

declaracionesRE ≡ declaracion declaracionesRE

declaracion.codh = $declaracionesRE_0$.codh
declaracion.etqh = $declaracionesRE_0$.etqh
$declaracionesRE_1$.codh = declaracion.cod
$declaracionesRE_1$.etqh = declaracion.etq

declaracionesRE$_0$.cod = declaracionesRE$_1$.cod
declaracionesRE$_0$.etq = declaracionesRE$_1$.etq

declaracionesRE ≡ λ

declaracionesRE.etq = declaracionesRE.etqh
declaracionesRE.cod = declaracionesRE.codh

7.3.2. Acondicionamiento de: parametros ≡ parametros , parametro
parametros ≡ parámetro

parametros ≡ parametro parametrosRE

parametro.codh = parametros.codh
parametro.etqh = parametros.etqh
parametrosRE.codh = parametro.cod
parametrosRE.etqh = parametro.etq
parametros.cod = parametrosRE.cod
parametros.etq = parametrosRE.etq

parametrosRE ≡ , parametro parametrosRE

parametro.codh = parametrosRE$_0$.codh
parametro.etqh = parametrosRE$_0$.etqh
parametrosRE$_1$.codh = parametro.cod
parametrosRE$_1$.etqh = parametro.etq
parametrosRE$_0$.etq = parametrosRE$_1$.etq
parametrosRE$_0$.cod = parametrosRE$_1$.cod

parametrosRE ≡ λ

parametrosRE.cod = parametrosRE.codh
parametrosRE.etq = parametrosRE.etqh

7.3.3. Acondicionamiento de: campos ≡ campos campo
campos ≡ campo ,

campos ≡ campo camposRE

campo.codh = campos.codh
campo.etqh = campos.etqh
camposRE.codh = campo.cod
camposRE.etqh = campo.etq
campos.cod = camposRE.cod
campos.etq = camposRE.etq

camposRE ≡ , campo camposRE

campo.codh = camposRE$_0$.codh
campo.etqh = camposRE$_0$.etqh
camposRE$_1$.codh = campo.cod
camposRE$_1$.etqh = campo.etq
camposRE$_0$.cod = camposRE$_1$.cod
camposRE$_0$.etq = camposRE$_1$.etq

camposRE ≡ λ

camposRE.cod = camposRE.codh
camposRE.etq = camposRE.etqh

7.3.4. Acondicionamiento de: acciones ≡ acciones accion
acciones ≡ accion

acciones ≡ accion accionesRE

accion.cod = acciones.codh
accion.etqh = acciones.etqh
accionesRE.codh = accion.cod
accionesRE.etqh = accion.etq
acciones.etq = accionesRE.etq
acciones.cod = accionesRE.cod

accionesRE ≡ accion accionesRE

accion.codh = accionesRE$_0$.codh
accion.etqh = accionesRE$_0$.etqh
accionesRE$_1$.codh = accion.cod
accionesRE$_1$.etqh = accion.etq
accionesRE$_0$.etq = accionesRE$_1$.etq
accionesRE$_0$.cod = accionesRE$_1$.cod

accionesRE ≡ λ

accionesRE.etq = accionesRE.etqh
accionesRE.cod = accionesRE.codh

7.3.5. Acondicionamiento de: mem ≡ id
mem ≡ mem [expresion2]
mem ≡ mem.id
mem ≡ mem^

mem ≡ id memRE

memRE.codh = mem.codh ++ accesoVar(mem.tsh[id.lex])
memRE.etqh = mem.etqh + longAccesoVar(mem.tsh[id.lex])
mem.etq = memRE.etq
mem.cod = memRE.cod

memRE ≡ [expresion2] memRE

expresion2.codh = memRE$_0$.codh
expresion2.etqh = memRE$_0$.etqh
memRE$_1$.codh = expresion2.cod ++ apila(memRE$_0$.tipo.tbase.tam) ++ multiplicar ++ suma
memRE$_1$.etqh = expresion2.etq + 3
memRE$_0$.cod = memRE$_1$.cod
memRE$_0$.etq = memRE$_1$.etq

memRE ≡ . id memRE

memRE$_1$.codh = menRE$_0$.codh
memRE$_1$.etqh = memRE$_0$.etqh
memRE$_0$.cod = memRE$_1$.cod ++ apila(memRE$_0$.tipo.campos[id.lex].desp) ++ suma
memRE$_0$.etq = memRE$_1$.etq + 2

memRE ≡ ^ memRE

memRE$_1$.codh = memRE$_0$.codh
memRE$_1$.etqh = memRE$_0$.etqh
memRE$_0$.cod = memRE$_1$.cod ++ apila-ind
memRE$_0$.etq = memRE$_1$.etq + 1

memRE ≡ λ

memRE.cod = memRE.codh
memRE.etq = memRE.etqh

7.3.6. Acondicionamiento de expresiones ≡ expresiones , expresion2
expresiones ≡ expresion2

expresiones ≡ expresion2 expresionesRE

expresion2.codh = expresiones.codh ++ copia
expresion2.etqh = expresiones.etqh + 1
expresionesRE.codh = expresion2.cod ++ pasoParametro(expresion2.modo,
expresiones.fparams[1])
expresionesRE.etqh = expresion2.etq + longPasoParametro
expresiones.etq = expresionesRE.etq
expresiones.cod = expresionesRE.cod

expresionesRE ≡ , expresion2 expresionesRE

expresion2.codh = $expresionesRE_0$.codh ++ copia
expresion2.etqh = $expresionesRE_0$.etqh + 1
$expresionesRE_1$.codh = expresion2.cod ++
direccionParFormal($expresionesRE_0$.fparams[expresionesRE0.nparams]) ++
pasoParametro(expresion2.modo,
$expresionesRE_0$.fparams[expresionesRE0.nparams])
$expresionesRE_1$.etqh = expresion2.etq ++
longDireccionParFormal($expresionesRE_0$.fparams[expresionesRE0.nparams]) +
longPasoParametro(expresion2.modo,
$expresionesRE_0$.fparams[$expresionesRE_0$.nparams])
$expresionesRE_0$.etq = $expresionesRE_1$.etq
$expresionesRE_0$.cod = $expresionesRE_1$.cod

expresionesRE ≡ λ

expresionesRE.cod = expresionesRE.codh
expresionesRE.etq = expresionesRE.etqh

7.3.7. Acondicionamiento de: expresion2 ≡ expresion3 op2 expresion3
expresion2 ≡ expresion3

expresion2 ≡ expresion3 expresion2RE

expresion3.codh = exprsion2.codh
expresion3.etqh = expresion2.etqh
expresion2RE.codh = expresion3.cod
expresion2RE.etqh = expresion3.etq
expresion2.cod = expresion2RE.cod
expresion2.etq = expresion2RE.etq

expresion2RE op2 expresion3 expresion2RE

op2.codh = $expresion2RE_0$.codh
op2.etqh = $expresion2RE_0$.etqh
expresion3.codh = op2.cod
expresion3.etqh = op2.etq
$expresion2RE_1$.codh = expresion3.cod ++ op2.op
$expresion2RE_1$.etqh = expresion3.etq + 1
$expresion2RE_0$.cod = $expresion2RE_1$.cod
$expresion2RE_0$.etq = $expresion2RE_1$.etq

expresion2RE

expresion2RE.cod = expresion2RE.codh
expresion2RE.etq = expresion2RE.etqh

7.3.8. Acondicionamiento de: expresion3 ≡ expresion3 op3 expresion4
expresion3 ≡ expresion4

expresion3 ≡ expresion4 expresion3RE
expresion4.codh = expresion3.codh
expresion4.etqh = expresion3.etqh
expresion3RE.codh = expresion4.cod
expresion3RE.etqh = expresion4.etq
expresion3.etq = expresion3RE.etq
expresion3.cod = expresion3RE.cod
expresion3RE ≡ op3 expresion4 expresion3RE
op3.codh = expresion3RE$_0$.codh
op3.etqh = expresion3RE$_0$.etqh
expresion4.codh = op3.cod ++
SI op3.op == || ENTONCES
copia ++ ir-v(expresion4.etq) ++ desapila
expresion4.etqh = op3.etq ++
SI op3.op == || ENTONCES 3
expresion3RE$_1$.codh = expresion4.cod ++
SI op3.op != || ENTONCES op3.op
expresion3RE$_1$.etqh = expresion4.etq +
SI op3.op != || ENTONCES 1
expresion3RE$_0$.etq = expresion3RE$_1$.etq
expresion3RE$_0$.cod = expresion3RE$_1$.cod
expresion3RE ≡ λ
expresion3RE.etq = expresion3RE.etqh
expresion3RE.cod = expresion3RE.codh

7.3.9. Acondicionamiento de: expresion4 ≡ expresion4 op4 expresion5
expresion4 ≡ expresion5

expresion4 ≡ expresion5 expresion4RE
expresion5.codh = expresion4.codh
expresion5.etqh = expresion4.etq
expresion4RE.codh = expresion5.cod
expresion4RE.etqh = expresion5.etq
expresion4.cod = expresion4RE.cod
expresion4.etq = expresion4RE.etq
expresion4RE ≡ op4 expresion5 expresion4RE
op4.codh = expresion4RE$_0$.codh
op4.etqh = expresion4RE$_0$.etqh
expresion5.etqh = op4.etq +
SI op4.op == && ENTONCES
1
SI NO
0
expresion5.codh = op4.cod ++
SI op4.op == && ENTONCES
ir-f(expresion5.etq + 1)
SI NO

expresion5.cod ++ op4.op

expresion4RE$_1$.etqh = expresion5.etq +

SI op4.op == && ENTONCES

2

SI NO

1

expresion4RE$_1$.codh = expresion5.cod ++

SI op4.op == && ENTONCES

ir-a(expresion5.etq + 2) ++ apila(0)

SI NO

op4.op

expresion4RE$_0$.etq = expresion4RE$_1$.etq

expresion4RE$_0$.cod = expresion4RE$_1$.cod

expresion4RE ≡ λ

expresion4RE.etq = expresion4RE.etqh

expresion4RE.cod = expresion4RE.codh

9. Esquema de traducción orientado al traductor

En el caso de traductores predictivo recursivos, esquema de traducción en el que se haga explícito los parámetros utilizados para representar los atributos, así como en los que se muestre la implementación de las ecuaciones semánticas como asignaciones a dichos parámetros. En el caso de traductores descendente tabulares o traductores ascendentes, se deberá usar pseudovariables apropiadas para referir a los atributos.

NOTA: en esta segunda entrega NO es necesario incluir el esquema de traducción orientado a la gramática de atributos

9.1. Variables globales

Descripción y propósito de las variables globales usadas.

Si no se usan variables globales, dejar este apartado en blanco.

- **Ts:** Tabla de simbolos.
- **Error:** Indica si se ha producido algún error en algún punto del programa.
- **Dir:** Siguiente dirección de memoria disponible.
- **Cod:** Contiene el codigo del programa.
- **Etq:** Direccion de la instruccion en el interprete
- **Pend:** Lista de tipos pendientes de ser declarados antes del fin de las declaraciones

9.2. Nuevas operaciones y transformación de ecuaciones semánticas

En caso de introducir nuevas operaciones (por ejemplo: el procedimiento “emite”), deben describirse aquí.

Debe describirse, así mismo, qué ecuaciones semánticas se ven transformadas y cómo (por ejemplo: la generación de código se implementa emitiendo la última instrucción).

Para evitar la complejidad de utilizar un gran numero de funciones en el codigo del compilador, los metodos de reconocimiento de tokens han sido sustituidos por el metodo ***reconoce(in tokenAReconocer)*** que devuelve el lexeme del token que debe reconocer, por ejemplo:

- reconoce(tkid):lex

A excepción del caso de asignacion(), este es sustituído por el metodo ***reconoceAsignacion().*** En lugar de reconocer el token esperado, éste metodo se cerciora de la futura existencia de un operador de asignacion como siguiente token, permitiendo diferentes caminos en caso de no existir dicho operador.

9.3. Esquema de traducción

programa() ≡
{
ts = creaTS()
n = 0
dir = 0
error = FALSE
pend = vacio
tbloque = int
etq = **longInicio** + 1
}
declaraciones()
{
error = error V pend !=
cod = inicio(n,dir) ++ ir_a(etq)
}
acciones()
{
cod = cod ++ stop
}
programa() ≡
{
ts = creaTS()
n = 0
error = FALSE
tbloque = int
etq = 0
cod = inicio(n,dir) ++ ir_a(etq)
}
acciones()
{
cod = cod ++ stop
}
declaraciones() ≡
{
}
declaracion()
{
}
declaracionesRE()
{
}
declaracionesRE() ≡
{
}
declaracion()
{
}

```
declaracionesRE()
{
}
declaracionesRE() ≡
{
}
vacio()
{
}
declaracion() ≡
{
}
comentario()
{
}
declaracion() ≡
{
}
declaracionvar()
{
}
declaracion() ≡
{
}
declaraciontipo()
{
}
declaracion() ≡
{
}
declaracionfun()
{
        n = n -1
}
declaracionvar() ≡
{
}
desctipo(out: tipo)
{
}
id(out: lex)
{
}
puntoYComa()
{
        ts = añadeID(ts, lex, <clase:var, dir: dir, tipo: tipo, nivel: n>)
        dir = dir + tipo.tam
        error = (existeID(ts, lex) ∨ error
}
declaraciontipo() ≡
{
}
```

```
tipo()
{
}
deftipo(out: tipo)
{
}
id(out: lex)
{
}
puntoYComa()
{
        ts = añadeID(ts, lex, <clase:tipo, tipo: tipo, nivel: n>)
        error = (existeID(ts, lex) AND  ts[lex].n == n) ∨ error
        pend = eliminaPendiente(pend, lex)
}
declaracionfun() ≡
{
}
fun()
{
}
id(out: lex)
{
}
parentesisAbierto()
{
        n = n + 1
}
listaparametros(out: params)
{
}
parentesisCerrado()
{
}
tiporeturn(out: tipo)
{
}
cuerpo(out: inicio)
{
}
end()
{
}
id(out: lex)
{
}
puntoYComa()
{
        ts = añadeID(ts, lex, <clase:fun, tipo: <t:funcion, params, tbase:          tipo>, nivel: n>)
        error = (existeID(ts, lex) ∨ (tipo NOT IN {int, real, puntero}) ∨ error
        propsop = <inicio:inicio>
}
```

```
listaparametros(out: params) ≡
{
params =
}
parametros(in: params, out params2)
{
   params = params2
}
listaparametros(out: params) ≡
{
}
vacio()
{
        params = λ
        nparams = 0
}
parametros(in: params, out: params2) ≡
{
}
parametro(in: params, out: params3)
{
}
parametrosRE(in: params3, out: params4)
{
        params2 = params4
}
parametrosRE(in: params, out: params2) ≡
{
}
coma()
{
}
parametro(in: params, out: params3)
{
}
parametrosRE(in: params3, out: params4)
{
        params2 = params4
}
parametrosRE(in: params, out: params2) ≡
{
}
vacio()
{
        params2 = params
}
parametro(in: params, out: params2) ≡
{
}
desctipo(out: tipo)
{
}
```

```
id(out: lex)
{
        ts = añadeID(ts, lex, <clase:var, tipo: tipo, nivel: n>)
        params2 = params ++ <modo:valor,tipo:tipo, dir: dir>
        error = error  ts[lex].n == n)
        dir = dir + 1
}
parametro(in: params, out: params2) ≡
{
}
desctipo(out: tipo)
{
}
amspersand()
{
}
id(out: lex)
{
        ts = añadeID(ts, lex, <clase: pvar, tipo: tipo, nivel: n>)
        params2 = params ++ <modo:variable, tipo: tipo, dir: dir>
        error = error  ts[lex].n == n)
         dir = dir + tipo.tam
}
tiporeturn(out: tbloque) ≡
{
}
returns()
{
}
desctipo(out: tipo)
{
        error = error V tipo NOT IN {int,real,puntero}
        tbloque = tipo.id
}
tiporeturn(out: tbloque) ≡
{
}
vacio()
{
        tipo = <t:int,tam:1>
        tbloque = int
}
cuerpo(out: inicio) ≡
{
}
declaraciones()
{
        cod = cod ++ prologo(n, dir)
        etq = etq + longPrologo
}
acciones()
{
```

```
            error = error
             inicio = etq
             cod = cod ++ epilogo(n) ++ ir-ind
            etq = etq + longEpilogo + 1
}
cuerpo(out: inicio) ≡
{
            cod = cod ++ prologo(n, dir)
             etq = etq + longPrologo
}
acciones()
{
            inicio = etq
            cod = cod ++ epilogo(n) ++ ir-ind
            etq = etq + longEpilogo + 1
}

desctipo(out: tipo) ≡
{
}
id(out: lex)
{
            tipo = <t:referencia, id:lex, tam: ts[lex].tipo.tam>
             error = error ∨
                        SI existeID(ts, lex) ENTONCES
                                    ts[lex].clase !=tipo
                        SI NO
                                     TRUE
            pend = SI ¬exiteID(ts,lex)  ENTONCES
                                    pend ++ lex
                         SI NO
                                    SI ts[lex].clase == tipo ENTONCES
                                                eliminaPendiente(desctipo.pendh,id.lex)
}

desctipo(out: tipo) ≡
{
}
int()
{
            tipo = <t:int,tam:1>
}
desctipo(out: tipo) ≡
{
}
real()
{
            tipo = <t:real,tam:1>
}
deftipo(out: tipo1) ≡
{
}
```

```
desctipo(out: tipo2)
{
}
corcheteAbierto()
{
}
litInt()
{
}
corcheteCerrado()
{
        tipo1 = <t:array, nelems:litInt, tbase:tipo2, tam: litInt * tipo2.tam>
        error = error
        pend = eliminaPendiente(pend, lex)
}
deftipo(out: tipo) ≡
{
}
rec()
{
        campos = λ
        desp = 0
}
campos(out: campos)
{
}
endrec()
{
        tipo = <t:registro, campos: campos, tam: campos.tam>
}

deftipo(out: tipo) ≡
{
}
pointer()
{
}
desctipo(out: tipo2, id)
{
        tipo = <t:puntero, tbase: tipo2, tam: 1>
        error = error Ú referenciaErronea
        pend = SI
                        pend = pend ++ id
                SI NO
                        eliminaPendiente(pend, id)
}
campos(out: campos) ≡
{
}
campo(in: campos, out: campos2)
{
}
```

```
camposRE(in: campos2, out: campos3)
{
        campos = campos3
}
camposRE(in: campos, out: campos2) ≡
{
}
coma()
{
}
campo(in: campos ,out: campos3)
{
}
camposRE(in: campos3,out: campos4)
{
        campos2 = campos4
}
camposRE(in: campos, out: campos2) ≡
{
}
vacio()
{
        campos2 = campos
}
campo(in: campos, out: campos2) ≡
{
}
desctipo (out: tipo)
{
}
id(out: lex)
{
        campos = campos ++ <id: lex, tipo: tipo, desp: tipo.tam>
        desp = desp + tipo.tam
        error = error V existeCampo(campos, lex)
        pend = elimina(pend, lex)
}
acciones() ≡
{
}
accion()
{
}
accionesRE()
{
}
accionesRE() ≡
{
}
accion()
{
}
```

accionesRE()
{
}
accionesRE() ≡
{
}
vacio()
{
}
accion() ≡
{
}
comentario()
{
}

accion() ≡
{
}
accionbasica()
{
}

accion() ≡
{
}
accionreturn()
{
}

accion() ≡
{
}
accionalternativa()
{
}

accion() ≡
{
}
accioniteracion()
{
}

accion() ≡
{
}
accionreserva()
{
}

accion() ≡

```
{
}
accionlibera()
{
}

accion() ≡
{
}
accioninvoca()
{
}

mem(out: id,tipo) ≡
{
}
id(out: lex)
{
        error = error ∨ ¬existeID(ts, lex)
        cod = cod ++ accesoVar(ts[lex])
         etq = etq + longAccesoVar(ts[lex])
}
memRE(in: id2, out:tipo2)
{
        id = id2
        tipo = tipo2
}
memRE(in: id, out: tipo) ≡
{
}
corcheteAbierto()
{
}
expresion2(out: tipo1)
{
}
corcheteCerrado()
{
        cod = cod ++ apila(ts[id].tbase.tam) ++ multiplicar ++ suma
         etq = etq + 3
        error = error ∨ ts[id].tipo != array
}
memRE(in: id, out: tipo2)
{
        tipo = ts[id].tipo.tbase
}
memRE(in id, out: tipo) ≡
{
}
punto()
{
}
```

```
id(out: lex)
{
        error = error ∨ ts[id].tipo  !=  registro
        cod = cod ++ apila(ts[id].tipo.campos[lex].desp) ++ suma
         etq = etq + 2
}
memRE(in: id, out: tipo2)
{
        tipo = tipo2
}
memRE(in: id, out: tipo) ≡
{
}
flecha()
{
        error = error ∨ memRE1.tipo  !=
}
memRE(in: id, out: tipo2)
{
        cod = cod ++ apila-ind
         etq = etq + 1
         tipo = tipo2
}
memRE(in: id, out: tipo) ≡
{
}
vacio()
{
}
accionbasica() ≡
{
}
expresion()
{
}
puntoYComa()
{
}
accionreturn(in: tbloque) ≡
{
}
return()
{
}
expresion2(out: tipo)
{
}
puntoYComa()
{
         error = error   tipo != tbloque
}
accionalternativa() ≡
```

```
{
}
if()
{
}
expresion2(out: tipo)
{
}
then()
{
        bloque.errorh = expresion2.error  tipo != <t:int>
        cod = cod ++ ir-f(bloqueEtq + 1)
        etq = etq + 1
}
bloque()
{
        cod = cod ++ parchea(ir-a(accionelseEtq + 1))
        etq = etq + 1
}
accionelse()
{
}
endif()
{
}
puntoYComa()
{
}
accionelse() ≡
{
}
elsif()
{
}
expresion2(out: tipo)
{
}
then()
{
        cod = cod ++ parchea(ir-f(bloqueEtq + 1))
        etq = etq + 1
}
bloque()
{
        cod = cod ++ parchea(ir-a(accionelse1Etq + 1))
        etq = etq + 1
}
accionelse()
{
}

accionelse() ≡
```

```
{
}
else()
{
}
bloque()
{
}

accionelse() ≡
{
}
vacio()
{
}
bloque() ≡
{
}
acciones()
{
}
bloque() ≡
{
}
vacio()
{
}
accioniteracion() ≡
{
        etq = etq
}
while()
{
}
expresion2(out: tipo)
{
}
do()
{
        cod = cod ++ parchea(ir-f(bloqueEtq + 1))
        etq = etq + 1
}
bloque()
{
}
endwhile()
{
}
puntoYComa()
{
        cod = cod ++ ir-a(etqIteracion)
         etq = etq + 1
```

```
}
accionreserva() ≡
{
}
alloc()
{
}
mem(out: tipo)
{
}
puntoYComa()
{
        cod = cod ++ new(1) ++ desapila-ind
        etq = etq + 2
}
accionlibera() ≡
{
}
free()
{
}
mem(out: tipo)
{
}
puntoYComa()
{
        cod = cod ++ delete(1 )
        etq = etq + 1
}
accioninvoca() ≡
{
}
id(out: lex)
{
}
parentesisAbierto()
{
        nparams = 0
        params = λ
        error = error ∨ ¬ existeID(ts, lex)
}
Aparams(in: params, out: params2)
{
}
parentesisCerrado()
{
}
puntoYComa()
{
        error = error ∨  ts[id].nparams != nparams2
        cod = cod ++ apila_ret(etq) ++ ir-a(tsh[lex].inicio)
        etq = etq + longApilaRet + 1
```

```
}
Aparams(out: params, nparams) ≡
{
        nparams = 0
        params = λ
        cod = codh ++ inicio-paso
        etq = etq + longInicioPaso
}
expresiones(in: params, nparams, out: params1, nparams2)
{
        cod = cod ++ fin-paso
        etq = etq + longFinPaso
        params = params1
        nparams = nparams1
}
Aparams(out: params, nparams) ≡
{
}
vacio()
{
}
expresiones(in: params, nparams, out: params1, nparams2) ≡
{
        cod = cod ++ copia
        etq = etq + 1
}
expresion2(in: params, nparams, out: modo)
{
        cod = cod ++ pasoParametro(modo,params[1])
        etq = etq + longPasoParametro
}
expresionesRE(in: params, out: params2)
{
        params1 = params2
}
expresionesRE(in: params, nparams, out: params2, nparams2, modo) ≡
{
}
coma()
{
        cod = cod ++ copia
        etq = etq + 1
}
expresion2(out: modo)
{
        cod = cod ++ direccionParFormal(params[nparams]) ++ pasoParametro(modo,
                params[nparams])
        etq = etq + longDireccionParFormal(params[nparams]) +
                longPasoParametro(modo,params[nparams])
}
expresionesRE(in: params, nparams, out: params3, nparams3)
{
```

```
        params2 = params3
        nparams2 = nparams3
}
expresionesRE(in: params, nparams, out: params2, nparams2) ≡
{
}
vacio()
{
}
expresion(out: tipo) ≡
{
}
op0in (out: op)
{
}
id(out: lex)
{
        tipo = ts[lex].tipo
        error = error ∨ ¬existeID(ts,lex)
        cod = cod ++ op
         etq = etq + 1
}
expresion(out: tipo) ≡
{
}
op0out(out: op)
{
}
expresion1()
{
        etq = etq + 1
        cod = cod ++ escritura
}
expresion(out: tipo) ≡
{
}
expresion1(out: tipo1)
{
        tipo = tipo1
}
expresion1(out: tipo) ≡
{
}
mem(out: id)
{
}
op1(out: op)
{
}
expresion2(out: tipo2)
{
        tipo = ts[id].tipo
```

```
        parh = false
        cod = cod ++ mueve(ts[id].tipo.tam)
        etq = etq + 1
}
expresion1(out: tipo) ≡
{
}
expresion2(out: tipo1)
{
tipo = tipo1
}
expresion2(in: params, nparams, out: tipo, nparams1, params1, modo1) ≡
{
}
expresion3(out: tipo2, modo)
{
        nparams = nparams + 1
        params = params ++ <modo: modo,tipo: tipo2>
        modo1 = modo
}
expresion2RE(in: params1, nparams1, out: tipo2, nparams2, params2, modo2)
{
}
expresion2RE(in: params, nparams, tipo1, out: tipo, nparams2, params2, modo)
{
}
op2(out: op)
{
}
expresion3(out: tipo2, modo)
{
        cod = cod ++ op
        etq = etq + 1
}
expresion2RE()
{
        modo = valor
}
expresion2RE()≡
{
}
vacio()
{
}
expresion3() ≡
{
}
expresion4()
{
        etq = etq
}
expresion3RE()
```

```
{
}
expresion3RE(in: tipo, out: tipo1, modo) ≡
{
}
op3 (out: op)
{
        cod = cod ++ SI op == || ENTONCES
                copia ++ ir-v(etqAux) ++ desapila
         etq = etq + SI op == || ENTONCES 3
}
expresion4(out: tipo2)
{
        cod = cod ++ SI op3.op != || ENTONCES op
        etq = etq + SI op3.op != || ENTONCES 1
}
expresion3RE(in: tipo2, out: tipo3)
{
        tipo1 = devuelveTipo(op, tipo, tipo3)
         modo = valor
}
expresion3RE() ≡
{
}
vacio()
{
}
expresion4() ≡
{
}
expresion5()
{
        etqAux = etq
}
expresion4RE()
{
}
expresion4RE(in: tipo) ≡
{
}
op4 (out: op)
{
        etq = etq + SI op == && ENTONCES 1
        cod = cod ++ SI op == && ENTONCES ir-f(etqAux + 1)
}
expresion5 (out: tipo2)
{
         etqAux2 = etq
         etq = etq + SI op == && ENTONCES
                        2
                SI NO
                        1
```

```
cod = cod ++ SI op == && ENTONCES
                        ir-a(etqAux2 + 2) ++ apila(0)
        SI NO
                        op
}
expresion4RE()
{
        tipo = devuelveTipo(op, tipo, tipo2)
        modo = valor
}
expresion4RE()≡
{
}
vacio()
{
}
expresion5(out: tipo, modo) ≡
{
}
op5asoc (out: op)
{
}
expresion5(out: tipo1, modo1)
{
        tipo = devuelveTipo(op, tipo1, tipo1)
        modo = valor
        error = error ∨ ¬validoOperacion(ts, tipo1, op, NULL)
        etq = etq + 1
        cod = cod ++ op
}
expresion5(out: tipo, modo) ≡
{
}
op5noasoc(out: op)
{
}
expresion6(out: tipo2, modo2)
{
        tipo = devuelveTipo(op, null, null)
        modo = valor
        etq = etq + 1
        cod = cod ++ op
}
expresion5(out: tipo, modo) ≡
{
}
expresion6(out: tipo2, modo2)
{
        tipo = tipo2
         modo = modo2
}
expresion6(out: tipo, modo) ≡
```

```
{
}
parentesisAbierto()
{
}
expresion2(out: tipo2, modo2)
{
        tipo = tipo2
        modo = modo2
}
parentesisCerrado()
{
}
expresion6(out: tipo, modo) ≡
{
}
litInt(out: lex)
{
        tipo = <t:int,tam:1>
        modo = valor
        error = error ∨ ¬cast(ts, lex, <t:int>)
        cod = cod ++ apila(lex)
        etq = etq + 1
}
expresion6(out: tipo, modo) ≡
{
}
litReal(out: lex)
{
        tipo = <t:real,tam:1>
        modo = valor
        error = errorh ¬cast(ts, lex, <t:real>)
        cod = cod ++ apila(lex)
        etq = etq + 1
}
expresion6(out: tipo, modo) ≡
{
}
mem(out: id)
{
        tipo = dameTipo(expresion6.tsh,mem.id)
        modo = variable
        error = error ∨ ¬existeID(ts, id)
        cod = cod ++ apila_dir(ts.[id].dir)
        etq = etq + 1
}
op0in(out: op) ≡
{
}
in()
{
        op = leer
```

```
}
op0out() ≡
{
}
out()
{
        op = escribir
}
op1(out: op) ≡
{
}
asignacion()
{
        op = asignacion
}
op2(out: op)  ≡
{
}
menor()
{
        op = menor
}
op2(out: op)  ≡
{
}
mayor()
{
        op = mayor
}
op2(out: op)  ≡
{
}
menorIgual()
{
        op = menor_que
}
op2(out: op)  ≡
{
}
mayorIgual()
{
        op = mayor_que
}
op2(out: op)  ≡
{
}
igual()
{
        op = igual
}
op2(out: op)  ≡
{
```

```
}
distinto()
{
        op = distinto
}
op3(out: op) ≡
{
}
oLogica()
{
        op = o_logica
}
op3(out: op) ≡
{
}
suma()
{
        op = suma
}
op3(out: op) ≡
{
}
resta()
{
        op = resta
}
op4(out: op) ≡
{
}
multiplicacion()
{
        op = multiplicacion
}
op4(out: op) ≡
{
}
division()
{
        op = division
}
op4(out: op) ≡
{
}
modulo()
{
        op = modulo
}
op4(out: op) ≡
{
}
yLogica()
{
```

```
        op = y_logica
}
op5asoc(out: op) ≡
{
}
signo()
{
        op = signo
}
op5asoc(out: op) ≡
{
}
negacion()
{
        op = negacion
}
op5noasoc(out: op) ≡
{
}
parentesisAbierto()
{
}
int()
{
}
parentesisCerrado()
{
        op = cast_int
}
op5noasoc(out: op) ≡
{
}
parentesisAbierto()
{
}
real()
{
}
parentesisCerrado()
{
        op = cast_real
}
```

10. Formato de representación del código P

Deberá describirse el formato de archivo aceptado por el intérprete de la máquina P, llamado código a pila (código P). Se valorará la eficiencia de dicho formato (por ejemplo: uso de *bytecode* binario en lugar de texto).

El interprete cargará un archivo codificado en binario que se traducirá en un vector de instrucciones de maquina P, este tipo de instrucción constará de una operación de maquina P y en caso de requerirlo, un argumento. Este vector es generado por la compilación del código del programa, recibido por el compilador.

La máquina P se constituye por:

- **Pila**, infinita, capaz de almacenar los diferentes tipos de datos que utiliza el intérprete.
- **Memoria del programa**, capaz de almacenar cualquier tipo de instrucción generada por el compilador, sin valorar el tamaño.
- **Memoria**, que contiene el heap o cabecera dinámica, capaz de almacenar los diferentes tipos de datos que utiliza el intérprete.
- **Contador de programa**, cuya función es indicar la posición en la memoria del programa de la instrucción que se ejecuta.

Instrucciones de la maquina P:

Operación	Identificador de la operacion en ByteCode	Argumento	Descripción
PARAR	0		Detiene la ejecución
APILAR	1	Dato	Almacena en la cima de la pila el valor contenido en el argumento de la instrucción.
DESAPILAR	3	---	Desapila la cima
APILAR_DIR	2	Dirección	Almacena en la cima de la pila el valor contenido en la direccion de memoria indicada en el argumento de la instrucción
DESAPILAR_DIR	4	Dirección	Almacena en la direccionde memoria, indicada en el argumento de la instrucción, el valor de la cima de la pila

MENOR	5	---	res = valor1 < valor2
MAYOR	6	---	res = valor1 > valor2
MENOR_IGUAL	7	---	res = valor1 <= valor2
MAYOR_IGUAL	8	---	res = valor1 >= valor2
IGUAL	9	---	res = valor1 == valor2
DISTINTO	10	---	res = valor1 ≠ valor2
SUMA	11	---	res = valor1 + valor2
RESTA	12	---	res = valor1 - valor2
MULTIPLICA	13	---	res = valor1 * valor2
DIVISON	14	---	res = valor1 / valor2
MODULO	15	---	res = valor1 % valor2
Y_LOGICA	16	---	res = valor1 && valor2
O_LOGICA	17	---	res = valor1 || valor2
NEGACION	18	---	res = ! valor1
SIGNO	19	---	res = - valor1
CAST_INT	22	---	Convierte el valor1 en un valor de tipo entero
CAST_REAL	25	---	Convierte el valor1 en un valor de tipo real
ESCRIBIR	27	---	Escribe por pantalla en valor de la cima de la pila
LEER	30	---	Lee un valor y lo almacena en la cima de la pila
LIMPIAR	31	---	Elimina el contenido de la pila
IR_F	32	Direccion	Modifica el contador de programa a la direccion especificada en caso de que el dato obtenido de la cima de la pila sea falso (0)
IR_A	33	Direccion	Modifica el

			contador de programa a la direccion especificada
IR_V	34	Direccion	Modifica el contador de programa a la direccion especificada en caso de que el dato obtenido de la cima de la pila sea verdadero (1)
IR_IND	35	---	El contador de programa pasa a tener el valor contenido en la cima de la pila
MUEVE	36	Tamaño	Mueve el contenido de las celdas consecutivas que conforman un tamaño tam, cuya dirección de origen es origen, a la direccion destino, destino
COPIA	37	---	Duplica el valor contenido en la cima de la pila.
NEW	38	Tamaño	Reserva una posición de tamaño tam en el heap o cabecera, utilizando como direccion de inicio valor1
DELETE	39	Tamaño	Elimina una posición de tamaño tam en el heap o cabecera, que tiene como direccion el valor contenido en la cima de la pila
APILAR_IND	40	---	Almacena en la cima de la pila, el valor contenido en la direccion de memoria identificada por el valor de la cima previamente desapilado

DESAPILAR_IND	41	---	Guarda el valor contenido en la cima de la pila en la direccion de memoria identificada por el valor de la subcima de la pila, ambos valores previamente desapilados

11 Notas sobre la Implementación

Descripción de la implementación realizada.

11.1. Descripción de archivos

Enumeración de los archivos con el código fuente de la implementación, y descripción de lo que contiene cada archivo.

El código en sí deberá estar adecuadamente comentado, pero NO se incluirá en la memoria de la práctica. Aquí sólo debe describirse someramente qué contiene cada archivo.

Detallamos a continuacion, las clases que se corresponden con el compilador e interprete, obviando la interfaz grafica generada.

Package compilador

Package compilador.analizadorLexico

- **AnalizadorLexico:** Implementación del automata.
- **DatosToken:** Datos que representan cada token.
- **PalabrasReservadas:** Tabla que contiene el conjunto de palabras reservadas del lenguaje y el token al que corresponden en la traducción.

Package compilador.analizadorSintactico

- **Acciones:** Implementacion de las acciones del analizador sintactico.
- **AnalizadorSintactico:** Implementación del analizador sintáctico, que se delega tareas en el resto de clases de este paquete.
- **Declaraciones:** Implementacion de las declaraciones del analizador sintactico.
- **Expresiones:** Implementacion de las expresiones del analizador sintactico.
- **Params:** Implementacion de los parametros del analizador sintactico.
- **SintacticoException:** Excepciones relativas al analizador sintactico.
- **Tipos:** Implementacion de los tipos del analizador sintactico.

Package compilador.tablaSimbolos

- **Detalles:** Contiene las propiedades de cada elemento de la table de simbolos. Identificador (id), dirección (dir), tipo, clase, inicio y nivel.
- **GestorTS:** Útil para controlador la existencia de una única instancia de la pila table de simbolos.
- **TS:** Tabla de simbolos compuesta por un identificador y los detalles respectivos al mismo.

Package compilador.tablaSimbolos.tipos

- **Campo:** Descripción de un campo.
- **Tipo:** Clase abstracta de tipos.
- **TipoArray:** Descripcion de un tipo array.
- **TipoEntero:** Descripcion de un tipo entero.
- **TipoFuncion:** Descripcion de un tipo funcion.
- **TipoNull:** Descripcion de un tipo null.
- **TipoPuntero:** Descripcion de un tipo puntero.
- **TipoReal:** Descripcion de un tipo real.
- **TipoRegistro:** Descripcion de un tipo registro.

Package interprete

- **EscritorPila.**
- **InstruccionInterprete:** Especificación de la estructura de las instrucciones del interprete.
- **Interprete:** Simulador de la ejecución del código generado por el compilador.
- **InterpreteException:** Excepciones relativas al interprete.
- **LectorPila.**

Package interprete.datoPila

- **datoPila:** Descripción de los tipos de datos (enteros y reales) que utilize el interprete.

Package interprete.instrucciones

Contiene una clase para cada instrucción del interprete, cuya estructura es la implementación de cada metodo de clase interprete.InstruccionInterprete, especificando las acciones que debe realizar.

Package interprete.memoria

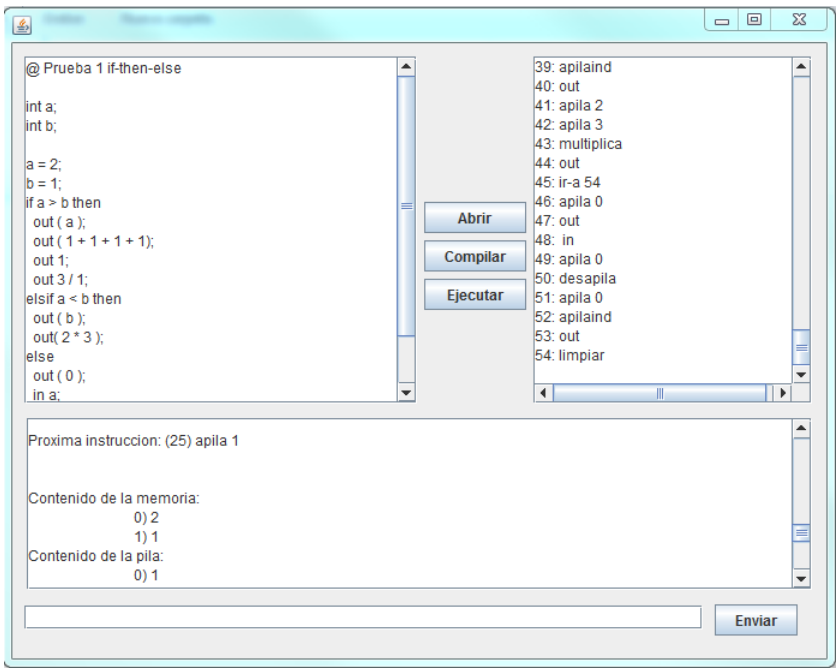
- **Hueco:** Descripción de la memoria dinamica.
- **HuecoDireccionComparator:** Comparador de huecos de memoria dinamica.
- **Memoria:** Memoria del interprete.

11.2. Otras notas

Otras notas sobre la implementación que se consideren pertinentes (por ejemplo: diagramas de clase UML describiendo la arquitectura del sistema, prototipos previos realizados, etc.).

Ejecución

Ademas de la ejecución especificada en el enunciado de la practica, se añade un jar ejecutable con la interfaz grafica.



Procedimiento:

- Abrir: Localización del fichero que contiene el programa que se desea compilar.
- Compilar: Realiza la compilación del código elegido previamente, permitiendo la visualización del resultado en el lado izquierdo (tokens e instrucciones) y en la parte inferior de mayor tamaño (memoria, pila y posibles errores).
- Ejecutar: Realiza la compilación y ejecución del código elegido previamente; mostrando el resultado de la compilacion (tokens e instrucciones) en el recuadro izquierdo y el resultado de la ejecución (memoria, pila, posibles errores y resultado del programa) en el recuadro inferior de mayor tamaño. Se incluye la posibilidad de la ejecución del programa en modo depuración, mostrando el contenido de la memoria y de la pila en cada ejecución.
- Enviar: Toda entrada que deba realizar el usuario debera ser introducida mediante el recuadro inferior y haciendo click en el boton Enviar.

Resultado de test de ejemplos

Test	Código válido	Código modificado	Resultado
Prueba1.txt	NO		NOK

Prueba1ok.txt	SI	`@ Prueba 1 if-then-else`
		
		`int a;`
		`int b;`
		
		`a = 1;`
		`b = 2;`
		`if a > b then`
		`  out ( a );`
		`elsif a < b then`
		`  out ( b );`
		`else`
		`  out ( 0 );`
		`endif;`
			OK

Prueba1ok1.txt	SI	`@ Prueba 1 if-then-else`
		
		`int a;`
		`int b;`
		
		`a = 2;`
		`b = 1;`
		`if a > b then`
		`  out ( a );`
		`  out ( 1 + 1 + 1 + 1);`
		`  out 1;`
		`  out 3 / 1;`
		`elsif a < b then`
		`  out ( b );`
		`  out( 2 * 3 );`
		`else`
		`  out ( 0 );`
		`  in a;`
		`  out a;`
		`endif;`
			OK

Prueba2.txt	SI		OK
Prueba3.txt	NO		NOK

Prueba3ok.txt	SI	`@ Prueba 3 reales`
		
		`real r ;`
		`int i;`
		
		`r = 1.5 ;`
		`i = 20;`
		`if r < 2.8 then`
		`  r = r + 10.0 *`
		`(real)i ;`
		`endif;`
		`i = (int) r;`
		`out ( i );`
			OK

Prueba4.txt	NO		NOK

Prueba4ok.txt	SI	`@ Prueba 4 arit. booleana`
		
		`int a;`
		
		`a = ( 1 > 0 ) && ( 2 < 3 ) || ( 5 > 7 ) ;`
		`out ( a );`
			OK

Prueba5.txt	NO		NOK

Prueba5ok.txt	SI	`@ Prueba 5 tipos (I)`
		
		`int a ;`
		`real b ;`
		`h tipoPend;`
		`tipo real[10] c ;`
		`tipo pointer real h;`
		
		`a = 1 ;`
		`out ( a ) ;`
			OK

Prueba6.txt	NO		NOK

Prueba6ok.txt	SI	`@ Prueba 6 tipos (II)`
		
		`tipo real[10] Tarray10;`
		`tipo rec int x; int y; endrec Tpunto;`
		`tipo pointer real Tpuntreal;`
		
		`Tarray10 var1;`
		`Tpunto var2;`
		`Tpuntreal var3;`
		
		`var1[0]=13;`
		`var2.x = 4;`
		`var3 = null;`
		
		`out(var1[0]);`
		`out(var2.x);`
			OK

Prueba7.txt	NO		NOK
Prueba7ok.txt	SI	`@ Prueba 7 arrays`	OK

		tipo real[10]
		Tarray10;
		
		Tarray10 a1;
		
		tipo real[10]
		Tarray20;
		
		Tarray20 a2;
		
		a1[2+3]=(real)17;
		a2 = a1;
		
		out(a2[5]);	

Prueba8.txt	NO		NOK
Prueba8ok.txt	SI	@ Prueba 8: punteros tipo pointer real TPuntReal; TPuntReal p; p = null ; alloc p ; p^ = 1.5 ; out(p^); free p ;	OK
Prueba9.txt			

Prueba9Declaracion.txt	SI	@ Prueba 9:
		funciones (I)
		fun suma (real b ,
		real c) returns real
		real a;
		a = b+c;
		return a+1.0;
		end suma;
		real x;
		real y;
		real z;
		x=3.0;
		y=4.0;
		out(y);
			OK

Prueba10.txt			
Prueba11.txt			
Prueba12.txt			