

Práctica de Procesadores de Lenguaje

Número de grupo: 13

Fausto Martín
Gabriel Peñas Rodríguez
Bernardo Pericacho Sánchez
Luis Sánchez

1 Especificación del léxico del lenguaje

- Sep \equiv &
- PyComa \equiv ;
- Punto \equiv .
- DosPuntos \equiv :
- Id \equiv { Letra } ({ Letra } | { Dig })*
- boolean \equiv boolean
- true \equiv true
- false \equiv false
- character \equiv character
- char \equiv ' [^ '] '
- natural \equiv natural
- NNat \equiv Φ | ({ DigPos } { Dig })
- integer \equiv integer
- float \equiv float
- NFloat \equiv { Signo } ? { NNat } ({ Dec } ? { Exp } ?)
- Asig \equiv :=
- Menor \equiv <
- Mayor \equiv >
- MenorOIgual \equiv <=
- MayorOIgual \equiv >=
- Igual \equiv =
- Distinto \equiv !=
- Mas \equiv \+
- Menos \equiv \-
- Mult \equiv *
- Div \equiv /
- Mod \equiv %
- and \equiv and
- or \equiv or
- not \equiv not
- DespDer \equiv >>
- Desplzq \equiv <<
- Cnat \equiv (nat)
- Cint \equiv (int)
- Cchar \equiv (char)
- Cfloat \equiv (float)
- PAp \equiv \((

- PCie $\equiv \backslash$)
- Com $\equiv \# (\{ \text{Letra} \} | \{ \text{Dig} \})^*$
- Barrav $\equiv \backslash$
- in \equiv in
- out \equiv out
- if \equiv if
- then \equiv then
- else \equiv else
- while \equiv while
- do \equiv do
- for \equiv for
- to \equiv to
- tipo \equiv tipo
- array \equiv array
- CorcheteIzq $\equiv \backslash$ [
- CorcheteDer $\equiv \backslash$]
- of \equiv of
- LlaveIzq $\equiv \backslash$ {
- LlaveDer $\equiv \backslash$ }
- record \equiv record
- pointer \equiv pointer
- null \equiv null
- new \equiv new
- dispose \equiv dispose
- procedure \equiv procedure
- var \equiv var
- forward \equiv forward
- flecha $\equiv ->$

Auxiliares

- Letra $\equiv [a-z][A-Z]$
- Dig $\equiv [0-9]$
- Digpos $\equiv [1-9]$
- Dec $\equiv (\{ \text{Dig} \}^* \{ \text{Digpos} \})$
- Exp $\equiv (e \text{---} E) \{ \text{Signo} \}^? \{ \text{NNat} \}$
- Signo $\equiv (\backslash + | \backslash -)$

2 Especificación de la sintaxis del lenguaje

A continuación se muestra una tabla con los operadores que posee nuestro lenguaje así como sus características de aridad, asociatividad y prioridad. Cuanto mayor prioridad mayor número en la columna “prioridad”.

SIMBOLO	NOMBRE	ARIDAD	ASOCIATIVIDAD	PRIORIDAD
NOT,-	OP4A	1	Derechas	4
(char), (int), (float), (nat)	OP4NA	1	No asocian	4
<<, >>	OP3	2	Derechas	3
*, /,%	OP2	2	Izquierdas	2
and		2	Izquierdas	2
+, -	OP1	2	Izquierdas	1
or		2	Izquierdas	1
<, <=, >, >=, =, /=	OP0	2	No asocian	0

Expecificación sintáctica:

- $\text{Prog} \rightarrow \text{Decs} \ \& \ \text{Is}$
- $\text{Prog} \rightarrow \& \ \text{Is}$
- $\text{Decs} \rightarrow \text{Decs} \ ; \ \text{Dec}$
- $\text{Decs} \rightarrow \text{Dec}$
- $\text{Dec} \rightarrow \text{DecVar}$
- $\text{Dec} \rightarrow \text{DecTipo}$
- $\text{Dec} \rightarrow \text{DecProc}$
- $\text{DecVar} \rightarrow \text{Id} : \text{Tipo}$
- $\text{DecTipo} \rightarrow \text{tipo} \ \text{Id} = \text{Tipo}$
- $\text{Tipo} \rightarrow \text{boolean}$
- $\text{Tipo} \rightarrow \text{character}$
- $\text{Tipo} \rightarrow \text{natural}$

- $\text{Tipo} \rightarrow \text{integer}$
- $\text{Tipo} \rightarrow \text{float}$
- $\text{Tipo} \rightarrow \text{Id}$
- $\text{Tipo} \rightarrow \text{array } [\text{NNat}] \text{ of Tipo}$
- $\text{Tipo} \rightarrow \text{pointer Tipo}$
- $\text{Tipo} \rightarrow \text{record } \{\text{Campos}\}$
- $\text{Campos} \rightarrow \text{Campos}; \text{Campo}$
- $\text{Campos} \rightarrow \text{Campo}$
- $\text{Campo} \rightarrow \text{Tipo Id}$
- $\text{DecProc} \rightarrow \text{procedure Id(Params) Bloque}$
- $\text{Params} \rightarrow \text{LParams}$
- $\text{Params} \rightarrow \lambda$
- $\text{LParams} \rightarrow \text{LParam, Param}$
- $\text{LParams} \rightarrow \text{Param}$
- $\text{Param} \rightarrow \text{Modo Tipo Id}$
- $\text{Modo} \rightarrow \text{var}$
- $\text{Modo} \rightarrow \lambda$
- $\text{Bloque} \rightarrow \text{forward}$
- $\text{Bloque} \rightarrow \{\text{Decs \& Is}\}$
- $\text{Bloque} \rightarrow \{\& \text{Is}\}$
- $\text{Is} \rightarrow \text{Is}; \text{I}$
- $\text{Is} \rightarrow \text{I}$
- $\text{I} \rightarrow \text{IAsig}$
- $\text{I} \rightarrow \text{ILec}$
- $\text{I} \rightarrow \text{IEsc}$
- $\text{I} \rightarrow \text{If}$
- $\text{I} \rightarrow \text{IWhile}$
- $\text{I} \rightarrow \text{IFor}$
- $\text{I} \rightarrow \text{ICall}$
- $\text{I} \rightarrow \text{INew}$
- $\text{I} \rightarrow \text{IDelete}$
- $\text{IAsig} \rightarrow \text{Desig} := \text{Exp0}$
- $\text{Exp0} \rightarrow \text{Exp1 OP0 Exp1}$
- $\text{Exp0} \rightarrow \text{Exp1}$
- $\text{Exp1} \rightarrow \text{Exp1 OP1 Exp2}$
- $\text{Exp1} \rightarrow \text{Exp1 or Exp2}$
- $\text{Exp1} \rightarrow \text{Exp2}$
- $\text{Exp2} \rightarrow \text{Exp2 OP2 Exp3}$
- $\text{Exp2} \rightarrow \text{Exp2 and Exp3}$

- $\text{Exp2} \rightarrow \text{Exp3}$
- $\text{Exp3} \rightarrow \text{Exp4 OP3 Exp3}$
- $\text{Exp3} \rightarrow \text{Exp4}$
- $\text{Exp4} \rightarrow \text{OP4A Exp4}$
- $\text{Exp4} \rightarrow \text{OP4NA Exp5}$
- $\text{Exp4} \rightarrow \text{Exp5}$
- $\text{Exp5} \rightarrow (\text{Exp0})$
- $\text{Exp5} \rightarrow \text{NNat}$
- $\text{Exp5} \rightarrow \text{NFloat}$
- $\text{Exp5} \rightarrow \text{Signo NNat}$
- $\text{Exp5} \rightarrow \text{true}$
- $\text{Exp5} \rightarrow \text{false}$
- $\text{Exp5} \rightarrow \text{null}$
- $\text{Exp5} \rightarrow \text{char}$
- $\text{Exp5} \rightarrow \text{Desig}$
- $\text{Exp5} \rightarrow | \text{Exp0} |$
- $\text{Desig} \rightarrow \text{Id}$
- $\text{Desig} \rightarrow \rightarrow$
- $\text{Desig} \rightarrow \text{Desig} [\text{Exp0}]$
- $\text{Desig} \rightarrow \text{Desig.Id}$
- $\text{ILec} \rightarrow \text{in} (\text{id})$
- $\text{IEsc} \rightarrow \text{out} (\text{Exp0})$
- $\text{IIf} \rightarrow \text{if Exp0 then BloqueI PElse}$
- $\text{PElse} \rightarrow \text{else BloqueI}$
- $\text{PElse} \rightarrow \lambda$
- $\text{IWhile} \rightarrow \text{while Exp0 do BloqueI}$
- $\text{IFor} \rightarrow \text{for id = Exp0 to Exp0 do BloqueI}$
- $\text{BloqueI} \rightarrow \{ \text{Is} \}$
- $\text{BloqueI} \rightarrow \text{I}$
- $\text{ICall} \rightarrow \text{Id} (\text{Args})$
- $\text{Args} \rightarrow \text{LArgs}$
- $\text{Args} \rightarrow \lambda$
- $\text{LArgs} \rightarrow \text{LArgs, Exp0}$
- $\text{LArgs} \rightarrow \text{Exp0}$
- $\text{INew} \rightarrow \text{new Desig}$
- $\text{IDelete} \rightarrow \text{dispose Desig}$
- $\text{OP0} \rightarrow <$
- $\text{OP0} \rightarrow >$
- $\text{OP0} \rightarrow \leq$

- OP0 $\rightarrow >=$
- OP0 $\rightarrow /=$
- OP0 $\rightarrow =$
- OP1 $\rightarrow +$
- OP1 $\rightarrow -$
- OP2 $\rightarrow /$
- OP2 $\rightarrow *$
- OP2 $\rightarrow \%$
- OP3 $\rightarrow >>$
- OP3 $\rightarrow <<$
- OP4A $\rightarrow \text{not}$
- OP4A $\rightarrow -$
- OP4NA $\rightarrow (\text{float})$
- OP4NA $\rightarrow (\text{nat})$
- OP4NA $\rightarrow (\text{int})$
- OP4NA $\rightarrow (\text{char})$

3 Estructura y construcción de la tabla de símbolos

3.1 Estructura de la tabla de símbolos

Tabla de símbolos:

id	Tipo	dir	Clase
----	------	-----	-------

creaTs : Ts	crea una Ts vacía
existeId(Ts, id): boolean	devuelve true si el identificador id está en la tabla de símbolos Ts
Ts [id]	accede a los registros de un identificador id de la Tabla de Símbolos Ts
Ts [id] . _	accede a un registro de un identificador id de la tabla de símbolos Ts
ponId(Ts, id, props): ts	añade un nuevo identificador a ls Ts con su tipo y dirección, si ya estaba lo sustituye.
creaTs(<i>tsPadre</i> : Ts): Ts	Para crear una tabla de símbolos con una tabla padre. La búsqueda de un símbolo se llevará a cabo primeramente en la tabla, y, en caso de no estar allí, en la tabla padre.

props: <clase: clase, tipo: tipo,nivel:nivel,dir:dir> props registro propiedades
tipo:<t:tipo,tam:tamaño> → Tiene como mínimo esta información. tipo registro de tipo

3.2 Construcción de la tabla de símbolos

3.2.1 Funciones semánticas

3.2.2 Atributos semánticos

<u>Atributo</u>	<u>Tipo de Atributo</u>	<u>Descripción</u>
ts	sintetizado	Es la tabla de símbolos sintetizada
tsph	heredado	Tabla de símbolos heredada del nivel actual.
dir	sintetizado	Representa el contador de direcciones que llevamos para ir almacenando los identificadores en la tabla de símbolos
dirh	heredado	Contador de direcciones que heredan las declaraciones.
id	sintetizado	Representa el nombre del identificador(lexema).
tipo	sintetizado	Representa el tipo de cada categoría léxica
nh	heredado	Nivel de anidamiento actual
Tam	Sintetizado	Tamaño de la declaración
props	sintetizado	Registro de propiedades de la declaración
Campos	sintetizado	Lista con los campos de un registro
campo	sintetizado	Campo de un registro
Desph	Heredado	Desplazamiento de un campo de un registro
Params	Sintetizado	Lista de parámetros de un procedimiento
param	sintetizado	Parámetro de un procedimiento
modo	sintetizado	Indica si el parámetro es por variable o por valor
clase	sintetizado	Indica la clase de parámetro, pvar para paso por variable, var para paso por valor
fwd	sintetizado	Indica si la declaración de un procedimiento es forward o no

3.2.3 Gramática de atributos

- **Prog → Decs & Is**
Decs.tsph = creaTs()
Decs.nh = 0
Decs.dirh = 0
Is.tsh=Decs.ts
- **Prog → & Is**
Is.tsh= creaTs()
- **Decs → Decs ; Dec**
Decs₀.ts = ponId(Decs₁.ts, Dec.id, Dec.props)
Decs₀.dir = Decs₁.dir + Dec.tam
Dec.dirh = Decs₁.dirh
Decs₁.dirh=Decs₀.dirh
Decs₁.tsph = Decs₀.tsph
Dec.tsph = Decs₁.ts
Decs₁.nh = Decs₀.nh
Dec.nh = Decs₀.nh
- **Decs → Dec**
Decs.ts = ponId(Decs.tsph,Dec.id,Dec.props)
Decs.dir = Decs.dirh+Dec.tam
Dec.dirh = Decs.dirh
Dec.tsph = Decs.tsph
Dec.nh = Decs.nh
- **Dec → DecVar**
Dec.id = DecVar.id
Dec.props = <clase:var, tipo:DecVar.tipo, nivel:Dec.nh,dir,DecVar.dir>
DecVar.dir = Dec.dirh
Dec.tam = DecVar.tipo.tam
DecVar.tsph = Dec.tsph
- **Dec → DecTipo**
Dec.id = DecTipo.id
Dec.props = <clase:tipo, tipo:DecTipo.tipo, nivel:Dec.nh,dir:->
Dec.tam = 0
DecTipo.tsph = Dec.tsph
- **Dec → DecProc**
Dec.id = DecProc.id
Dec.props = <clase:DecProc.props.clase, tipo:DecProc.props.tipo,
nivel:Dec.nh,dir:->
DecProc.tsph = Dec.tsph
DecProc.nh = Dec.nh

DecProc.dirh=Dec.dirh

Dec.tam=0

- **DecVar → Id : Tipo**

DecVar.id = Id.lex

DecVar.tipo = Tipo.tipo

Tipo.tsph=DecVar.tsph

- **DecTipo → tipo Id = Tipo**

DecTipo.id = Id.lex

DecTipo.tipo = Tipo.tipo

Tipo.tsph=DecTipo.tsph

- **Tipo → boolean**

Tipo.tipo = <t:boolean,tam:1>

- **Tipo → character**

Tipo.tipo = <t:character,tam:1>

- **Tipo → natural**

Tipo.tipo = <t:natural,tam:1>

- **Tipo → integer**

Tipo.tipo = <t:integer,tam:1>

- **Tipo → float**

Tipo.tipo = <t:float,tam:1>

- **Tipo → Id**

Tipo.tipo = <t:ref, id:Id.lex, tam:Tipo.tsph[Id.lex].tipo.tam>

- **Tipo → array [NNat] of Tipo**

Tipo₀.tipo = <t:array, nElems: NNat.lex, tBase: Tipo₁.tipo, tam: NNat.lex * Tipo₁.tipo.tam>

- **Tipo → pointer Tipo**

Tipo₀.tipo = <t: punt, tBase: Tipo₁.tipo, tam: 1>

- **Tipo → record {Campos}**

Tipo.tipo = <t:reg, campos: Campos.campos, tam: Campos.tam>

- **Campos → Campos; Campo**

Campos₀.campos = Campos₁.campos || Campo.campo

Campos₀.tam = Campos₁.tam + Campo.tam

Campo.desph = Campos₁.tam

- **Campos → Campo**

Campos.campos = [Campo.campo]

Campos.tam = Campo.tam

Campo.desph = 0

- **Campo → Tipo Id**

Campo.campo = <id:Id.lex, tipo: Tipo.tipo, desp: Campo.desph>

Campo.tam = Tipo.tipo.tam

- **DecProc → procedure Id(Params) Bloque**

```
DecProc.id = Id.lex
DecProc.props = <clase:proc, tipo: <t:proc,params:Params.params>,nivel:DecProc.nh+1,dir:->
Params.tsph = creaTs(DecProc.tsph)
Params.nh = DecProc.nh + 1
Bloque.tsph = ponId(Params.ts,DecProc.Id,DecProc.props)
Bloque.nh = DecProc.nh + 1
Bloque.dirh=Params.dir
Params.dirh=DecProc.dirh
```

- **Params → LParams**

```
Params.params = LParams.params
LParams.tsph = Params.tsph
LParams.nh = Params.nh
Params.ts = Lparams.ts
Params.dir=LParams.dir
```

- **Params → λ**

```
Params.params = []
Params.ts = Params.tsph
Params.dir=0
```

- **LParams → LParams, Param**

```
LParams0.params = LParams1.params || [Param.param]
LParam0.ts = ponId(LParam1.ts,Param.id,Param.props)
LParams1.tsph = LParams0.tsph
LParams1.nh = LParams0.nh
Param.nh = Lparams0.nh
Lparams0.dir=LParams1.dir+Param.tam
Lparams1.dirh=LParams0.dirh
Param.dirh=Lparams1.dir
```

- **LParams → Param**

```
LParam.params = [Param.param]
LParam.ts = ponId(LParam.tsph,Param.id,Param.props)
Param.nh = Lparams.nh
Param.dirh=0
Lparams.dir=Param.tam
```

- **Param → Modo Tipo Id**

```
Param.id=Id.lex
Param.props=
  <clase:Modo.clase,tipo:Tipo.tipo,nivel:Param.nh,dir:Param.dirh>
Param.param= <modo:Modo.modo,tipo:Tipo.tipo>
Param.tam= si Modo.modo= variable entonces 1 si no Tipo.tipo.tam
```

- **Modo \rightarrow var**
 Modo.modos=variable
 Modo.clase=pvar
- **Modo $\rightarrow \lambda$**
 Modo.modos=valor
 Modo.clase=var
- **Bloque \rightarrow forward**
 Bloque.fwd=true
- **Bloque \rightarrow {Decs & Is}**
 Bloque.fwd=false
 Decs.tsph=Bloque.tsph
 Decs.dirh=Bloque.dirh
 Decs.nh=Bloque.nh
 Is.tsh=Decs.ts
 Bloque.ts=Decs.ts
- **Bloque \rightarrow {& Is}**
 Bloque.fwd=false
 Is.tsh=Bloque.tsph

4 Especificación de las restricciones contextuales

4.1 Funciones semánticas

<u>Función Semántica</u>	<u>Cometido de la función</u>	<u>Descripción de los parámetros</u>
errAsig(tipo1, tipo2)	Devuelve true si hay un error al asignar a un tipo 'tipo1' un tipo 'tipo2'	<i>Tipo1</i> : tipo del identificador al que queremos asignar. <i>Tipo2</i> : tipo de la expresión que queremos asignar.
tipoOPX.(tipo1, tipo2)	Devuelve el tipo de realizar la operación con operadores del nivel X	<i>Tipo1</i> : tipo del primer operando. <i>Tipo2</i> : tipo del segundo operando.
tipoOPX.(op ,tipo1, tipo2)	Devuelve el tipo de realizar la operación con operadores de nivel X que tiene como código de operacion 'op'	<i>Tipo1</i> : tipo del primer operando. <i>Tipo2</i> : tipo del segundo operando. <i>Op</i> : operación a realizar.
tipoOr(tipo1,tipo2)	Devuelve el tipo de realizar la operación or	<i>Tipo1</i> : tipo del primer operando. <i>Tipo2</i> : tipo del segundo operando.
tipoAnd(tipo1,tipo2)	Devuelve el tipo de realizar la operación and	<i>Tipo1</i> : tipo del primer operando. <i>Tipo2</i> : tipo del segundo operando.
tipoOP4A(op, tipo)	Devuelve el tipo de realizar la operación de nivel 4 con operadores que asocian con código de operación 'op'	<i>Tipo</i> : tipo del operando. <i>Op</i> : operación a realizar.
tipoOP4NA(op, tipo)	Devuelve el tipo de realizar la operación de nivel 4 con operadores que NO asocian con código de operación 'op'	<i>Tipo</i> : tipo del operando. <i>Op</i> : operación a realizar.
valorAbs(tipo)	Devuelve el tipo de realizar la operación Abs con un operado de tipo 'tipo'	<i>Tipo</i> : tipo del operando.
ReferenciaErronea(Tipo, Ts)	Devuelve true si existe el tipo en la tabla de simbolos y es un tipo puntero.	<i>Tipo</i> : tipo del operando. <i>Ts</i> : tabla de simbolos.
sigueRef(Tipo,Ts)	Devuelve el tipo de seguir una cadena de <i>alias</i> .	<i>Tipo</i> : tipo del operando. <i>Ts</i> : tabla de simbolos.
compatibles(tipo1, tipo2, Ts)	Devuelve true si los tipos a comparar son compatibles entre ellos y false en caso contrario	<i>Tipo1</i> : tipo del primer operando. <i>Tipo2</i> : tipo del segundo operando. <i>Ts</i> : tabla de simbolos.
esDuplicado(Campos, id)	Devuelve true cuando la lista de campos <i>campos</i> contenga el campo <i>id</i>	<i>Campos</i> : lista de campos. <i>Id</i> : id del campo a buscar.

```

fun compatibles(e1,e2,ts): boolean
  visitadas  $\leftarrow \emptyset$ 
  fun compatibles2(e1,e2)
    si <e1,e2>  $\in$  visitadas
      devuelve true
    si no
      visitadas  $\leftarrow$  visitadas  $\cup \{ \langle e1,e2 \rangle \}$ 
    si (e1.t = e2.t = natural)  $\vee$ 
      (e1.t = e2.t = boolean)  $\vee$ 
      (e1.t = e2.t = integer)  $\vee$ 
      (e1.t = e2.t = float)  $\vee$ 
      (e1.t = punt  $\wedge$  e2 = punt  $\wedge$  e2.tbase=null)  $\vee$ 
      (e1.t = e2.t = character)  $\vee$ 
      (e1.t = integer  $\wedge$  e2.t=natural)  $\vee$ 
      (e1.t = float  $\wedge$  e2.t = integer)  $\vee$ 
      (e1.t = float  $\wedge$  e2.t = natural)
      devuelve true
    si no si e1.t = ref devuelve
      compatibles2(ts[e1.id].tipo,e2)
    si no si e2.t = ref devuelve
      compatibles2(e1,ts[e2.id].tipo)
    si no si e1.t=e2.t=array  $\wedge$  e1.nelems=e2.nelems devuelve
      compatibles2(e1.tbase,e2.tbase)
    si no si e1.t = e2.t = reg  $\wedge$  |e1.campos| = |e2.campos|
      para i  $\leftarrow$  1 hasta |e1.campos| hacer
        si  $\neg$  compatibles2(e1.campos[i].tipo ,
          e2.campos[i].tipo)
          devolver false
      fsi
    fpara
      devolver true
    si no si e1.t = e2.t = punt
      devuelve compatible2(e1.tbase,e2.tbase)
    si no si e1.t=e2.t = proc  $\wedge$  |e1.params| = |e2.params|
      para i  $\leftarrow$  1 hasta |e1.params| hacer
        si  $\neg$  compatibles2(e1.params[i].tipo,e2.params[i].tipo)  $\vee$ 
          (e2.params[i].modo = var  $\wedge$  e1.params[i].modo  $\neq$  var)
          devolver falso
      fpara
        devolver cierto
    si no devuelve false
  ffun
  compatibles2(e1,e2)
ffun

fun referenciaErronea(tipo,ts):boolean
devuelve tipo.t=ref  $\wedge$   $\neg$  existeID(ts,tipo.id)
ffun

```

[illegible]

- **Función tipoOP1(toper1, toper2)**

[illegible]

- **Función tipoOr(op, toper1, toper2)**

[illegible]

- **Función tipoOP2(op,toper1, toper2)**

Si (op = *) \vee (op = /) entonces

[illegible]

Si (op = %) entonces

[illegible]

- **Función tipoAnd(toper1, toper2)**

[illegible]

- **Función tipoOP3(toper1, toper2)**

[illegible]

- **Función tipoOP4A(op, toper)**

Si (op = not) entonces

<u>toper</u>	<u>Resul</u>
natural	Err
Integer	Err
Float	Err
character	Err
Boolean	Boolean
array	err
reg	err
punt	err
proc	err
Err	Err

Si (op = --) entonces

<u>Toper</u>	<u>Resul</u>
natural	Integer
integer	Integer
float	Float
character	Err
boolean	Err
array	err
reg	err
punt	err
proc	err
Err	Err

- **Función tipoOP4NA(op, toper)**

Si (op = cfloat) entonces

<u>toper</u>	<u>Resul</u>
natural	Float
integer	Float
float	Float
character	Float
Boolean	Err
array	err
reg	err
punt	err
proc	err
Err	Err

Si (op = cint) entonces

<u>toper</u>	<u>Resul</u>
natural	Integer
integer	Integer
float	Integer
character	Integer
boolean	Err
array	err
reg	err
punt	err
proc	err
Err	Err

Si (op = cnat) entonces

<u>toper</u>	<u>Resul</u>
natural	Natural
integer	Err
float	Err
character	Natural
boolean	Err
array	err
reg	err
punt	err
proc	err
Err	Err

Si (op = cchar) entonces

<u>toper</u>	<u>Resul</u>
natural	Character
integer	Err
float	Err
character	Character
boolean	Err
array	err
reg	err
punt	err
proc	err
Err	Err

- **Función Abs(toper)**

<u>toper</u>	<u>Resul</u>
natural	Natural
integer	Natural
float	Float
character	Err
boolean	Err
array	err
reg	err
punt	err
proc	err
Err	Err

4.2 Atributos semánticos

<u>Atributo</u>	<u>Tipo de Atributo</u>	<u>Descripción</u>
err	sintetizado	Representa si hay errores o no.
tsh	heredado	Representa la tabla de símbolos heredada.
tipo	sintetizado	Representa el tipo final de la expresión de nivel X.
Op	Sintetizado	Representa la operación que realizada el OpX
pend	Sintetizado	Conjunto con los tipos pendientes de definir
procpend	sintetizado	Conjunto con los procedimientos pendientes de definir
modo	sintetizado	Indica si la expresion se para por variable o por parámetro

4.3 Gramática de atributos

Restricciones contextuales

- **Prog → Decs & Is**

$$\text{Prog.err} = \text{Decs.err} \vee \text{Is.err} \vee \text{Decs.pend} \diamond \{\} \vee \text{Decs.procpend} \diamond \{\}$$

$$\text{Is.tsh} = \text{Decs.ts}$$
- **Prog → & Is**

$$\text{Prog.err} = \text{Is.err}$$
- **Decs → Decs ; Dec**

$$\text{Decs}_0.\text{err} = \text{Decs}_1.\text{err} \vee \text{Dec.err} \vee (\text{existeId}(\text{Decs}_1.\text{ts}, \text{Dec.Id}) \wedge \text{Decs}_1.\text{ts}[\text{Dec.id}].\text{nivel} = \text{Decs}_0.\text{nh} \wedge \text{Dec.props.clase} \diamond \text{proc}) \vee$$

$(\text{existeId}(\text{Dec}_1.\text{ts}, \text{Dec}.\text{Id}) \wedge \text{Dec}_1.\text{ts}[\text{Dec}.\text{id}].\text{nivel} = \text{Dec}_0.\text{nh} \wedge$
 $(\neg \text{pendientes}(\text{Dec}.\text{id}, \text{Dec}_1.\text{procpend}) \vee (\text{Dec}.\text{fwd}))$
 $\text{Dec}_0.\text{pend} = \text{Dec}_1.\text{pend} \cup \text{Dec}.\text{pend} - (\text{Si } \text{Dec}.\text{props}.\text{clase} = \text{tipo} \text{ entonces}$
 $\{\text{Dec}.\text{id}\} \text{ sino } \{\})$
 $\text{Dec}_0.\text{procpend} = \text{Dec}_1.\text{procpend} \cup \text{Dec}.\text{procpend} - (\text{si } \text{Dec}.\text{fwd} = \text{false} \wedge$
 $\text{Dec}.\text{props}.\text{clase} = \text{proc} \text{ entonces } \{\text{Dec}.\text{id}\} \text{ sino } \{\})$
 $\text{Dec}_1.\text{tsh} = \text{Dec}_0.\text{tsh}$
 $\text{Dec}.\text{tsh} = \text{Dec}_1.\text{ts}$

- Decs \rightarrow Dec**
 $\text{Decs}.\text{err} = \text{Dec}.\text{err} \vee$
 $(\text{existeId}(\text{Decs}.\text{tsh}, \text{Dec}.\text{Id}) \wedge \text{Decs}.\text{tsh}[\text{Dec}.\text{id}].\text{nivel} = \text{Decs}.\text{nh} \wedge$
 $\text{Dec}.\text{props}.\text{clase} \triangleleft \text{proc})$
 $\text{Decs}.\text{pend} = \text{Dec}.\text{pend} - (\text{Si } \text{Dec}.\text{props}.\text{clase} = \text{tipo} \text{ entonces } \{\text{Dec}.\text{id}\} \text{ sino } \{\})$
 $\text{Decs}.\text{procpend} = \text{Dec}.\text{procpend} - (\text{si } \text{Dec}.\text{fwd} = \text{false} \wedge \text{Dec}.\text{props}.\text{clase} = \text{proc}$
 $\text{entonces } \{\text{Dec}.\text{id}\} \text{ sino } \{\})$
 $\text{Dec}.\text{tsh} = \text{Decs}.\text{tsh}$
- Dec \rightarrow DecVar**
 $\text{Dec}.\text{err} = \text{DecVar}.\text{err}$
 $\text{Dec}.\text{pend} = \text{DecVar}.\text{pend}$
 $\text{DecVar}.\text{tsh} = \text{Dec}.\text{tsh}$
 $\text{Dec}.\text{procpend} = \{\}$
 $\text{Dec}.\text{fwd} = \text{false}$
- Dec \rightarrow DecTipo**
 $\text{Dec}.\text{err} = \text{DecTipo}.\text{err}$
 $\text{Dec}.\text{pend} = \text{DecTipo}.\text{pend}$
 $\text{DecTipo}.\text{tsh} = \text{Dec}.\text{tsh}$
 $\text{Dec}.\text{procpend} = \{\}$
 $\text{Dec}.\text{fwd} = \text{false}$
- Dec \rightarrow DecProc**
 $\text{Dec}.\text{err} = \text{DecProc}.\text{err}$
 $\text{DecProc}.\text{tsh} = \text{Dec}.\text{tsh}$
 $\text{Dec}.\text{procpend} = \text{DecProc}.\text{procpend}$
 $\text{Dec}.\text{fwd} = \text{DecProc}.\text{fwd}$
- DecVar \rightarrow Id : Tipo**
 $\text{DecVar}.\text{pend} = \text{Tipo}.\text{pend}$
 $\text{DecVar}.\text{err} = \text{Tipo}.\text{err} \vee \text{existeId}(\text{DecVar}.\text{tsh}, \text{Id}.\text{lex}) \vee$
 $\text{referenciaErronea}(\text{Tipo}.\text{tipo}, \text{DecVar}.\text{tsh})$
 $\text{Tipo}.\text{tsh} = \text{DecVar}.\text{tsh}$
- DecTipo \rightarrow tipo Id = Tipo**
 $\text{DecTipo}.\text{pend} = \text{Tipo}.\text{pend}$
 $\text{DecTipo}.\text{err} = \text{Tipo}.\text{err} \vee \text{existeId}(\text{DecTipo}.\text{tsh}, \text{Id}.\text{lex}) \vee$
 $\text{referenciaErronea}(\text{Tipo}.\text{tipo}, \text{DecTipo}.\text{tsh})$

Tipo.tsh=DecTipo.tsh

- **Tipo → boolean**

Tipo.err=false

Tipo.pend={}

- **Tipo → character**

Tipo.err=false

Tipo.pend={}

- **Tipo → natural**

Tipo.err=false

Tipo.pend={}

- **Tipo → integer**

Tipo.err=false

Tipo.pend={}

- **Tipo → float**

Tipo.err=false

Tipo.pend={}

- **Tipo → Id**

Tipo.err=si existeId(Tipo.tsh,Id.lex) entonces Tipo.tsh[id.lex].clase<>tipo
si no false

Tipo.pend= si ¬existeId(Tipo.tsh,Id.lex) entonces {id.lex} si no {}

- **Tipo → array [NNat] of Tipo**

Tipo₀.pend=Tipo₁.pend

Tipo₀.err= Tipo₁.err v referenciaErronea(Tipo₁.tipo,Tipo₀.tsh)

- **Tipo → pointer Tipo**

Tipo₀.pend=Tipo₁.pend

Tipo₀.err=Tipo₁.err

- **Tipo → record {Campos}**

Tipo.err=Campos.err

Tipo.pend=Campos.pend

Campos.tsh=Tipo.tsh

- **Campos → Campos; Campo**

Campos₀.err= Campos₁.err v Campo.err v
esDuplicado(Campos₁.campos,Campo.id)

Campos₀.pend=Campos₁.pend U Campo.pend

Campos₁.tsh=Campos₀.tsh

Campo.tsh=Campos₀.tsh

- **Campos → Campo**

Campos.err=Campo.err

Campos.pend=Campo.pend

Campo.tsh=Campos.tsh

- **Campo → Tipo Id**
 Campo.err=Tipo.err v referenciaErronea(Tipo.tipo,Campo.tsh)
 Campo.pend=Tipo.pend
- **DecProc → procedure Id(Params) Bloque**
 DecProc.err=Params.err v Bloque.err v (existeId(Params.ts,id.lex) ^
 Params.ts[id.lex].nivel=DecProc.nh+1)
 DecProc.procpend=Bloque.procpend U si Bloque.fwd=true entonces {id.lex}
 si no {}
 DecProc.fwd=Bloque.fwd
- **Params → Lparams**
 Params.err=LParams.err
- **Params → λ**
 Params.err=false
- **LParams → LParam, Param**
 Lparams₀.err=LParams₁.err v Param.err v (existeId(LParams₁.ts,Param.id)
 ^Lparams₁.ts[Param.id].nivel=LParams₀.nh)
- **LParams → Param**
 Lparams.err=Param.err
- **Param → Modo Tipo Id**
 Param.err=Tipo.err
- **Modo → var**
- **Modo → λ**
- **Bloque → forward**
 Bloque.err=false
- **Bloque → {Decs & Is}**
 Bloque.err=Decs.err v Is.err
 Is.tsh=Decs.ts
 Bloque.procpend=Decs.procpend
- **Bloque → {& Is}**
 Bloque.err=Is.err
 Is.tsh=Bloque.tsph
 Bloque.procpend={}
- **Is → Is ; I**
 Is₁.tsh=Is₀.tsh
 I.tsh=Is₀.tsh
 Is₀.err=Is₁.err v I.err
- **Is → I**
 I.tsh=Is.tsh
 Is.err=I.err

- **$I \rightarrow IAsig$**
 $IAsig.tsh = I.tsh$
 $I.err = IAsig.err$
- **$I \rightarrow Ilec$**
 $Ilec.tsh = I.tsh$
 $I.err = Ilec.err$
- **$I \rightarrow Iesc$**
 $Iesc.tsh = I.tsh$
 $I.err = Iesc.err$
- **$I \rightarrow Iif$**
 $Iif.tsh = I.tsh$
 $I.err = Iif.err$
- **$I \rightarrow Iwhile$**
 $Iwhile.tsh = I.tsh$
 $I.err = Iwhile.err$
- **$I \rightarrow Ifor$**
 $Ifor.tsh = I.tsh$
 $I.err = Ifor.err$
- **$I \rightarrow Icall$**
 $Icall.tsh = I.tsh$
 $I.err = Icall.err$
- **$I \rightarrow INew$**
 $Inew.tsh = I.tsh$
 $I.err = Inew.err$
- **$I \rightarrow IDelete$**
 $IDelete.tsh = I.tsh$
 $I.err = IDelete.err$
- **$IAsig \rightarrow Desig := Exp0$**
 $Exp0.tsh = IAsig.tsh$
 $IAsig.err = \neg compatibles(Desig.tipo, Exp0.tipo, IAsig.tsh)$
- **$Exp0 \rightarrow Exp1 \text{ OP0 } Exp1$**
 $Exp1_0.tsh = Exp0.tsh$
 $Exp1_1.tsh = Exp0.tsh$
 $Exp0.tipo = tipoOP0(OP0.op, Exp1_0.tipo, Exp1_1.tipo)$
 $Exp0.mod0 = val$
- **$Exp0 \rightarrow Exp1$**
 $Exp1.tsh = Exp0.tsh$
 $Exp0.tipo = Exp1.tipo$
 $Exp0.mod0 = Exp1.mod0$

- **Exp1 → Exp1 OP1 Exp2**
 $\text{Exp1}_1.\text{tsh} = \text{Exp1}_0.\text{tsh}$
 $\text{Exp2.tsh} = \text{Exp1}_0.\text{tsh}$
 $\text{Exp1}_0.\text{tipo} = \text{tipoOP1}(\text{Exp1}_1.\text{tipo}, \text{Exp2.tipo})$
 $\text{Exp1.mod} = \text{val}$
- **Exp1 → Exp1 or Exp2**
 $\text{Exp1}_1.\text{tsh} = \text{Exp1}_0.\text{tsh}$
 $\text{Exp2.tsh} = \text{Exp1}_0.\text{tsh}$
 $\text{Exp1}_0.\text{tipo} = \text{tipoOr}(\text{Exp1}_1.\text{tipo}, \text{Exp2.tipo})$
 $\text{Exp1.mod} = \text{val}$
- **Exp1 → Exp2**
 $\text{Exp2.tsh} = \text{Exp1.tsh}$
 $\text{Exp1.tipo} = \text{Exp2.tipo}$
 $\text{Exp1.mod} = \text{Exp2.mod}$
- **Exp2 → Exp2 OP2 Exp3**
 $\text{Exp2}_1.\text{tsh} = \text{Exp2}_0.\text{tsh}$
 $\text{Exp3.tsh} = \text{Exp2}_0.\text{tsh}$
 $\text{Exp2}_0.\text{tipo} = \text{tipoOP2}(\text{OP2.op}, \text{Exp2}_1.\text{tipo}, \text{Exp3.tipo})$
 $\text{Exp2.mod} = \text{val}$
- **Exp2 → Exp2 and Exp3**
 $\text{Exp2}_1.\text{tsh} = \text{Exp2}_0.\text{tsh}$
 $\text{Exp3.tsh} = \text{Exp2}_0.\text{tsh}$
 $\text{Exp2}_0.\text{tipo} = \text{tipoAnd}(\text{Exp2}_1.\text{tipo}, \text{Exp3.tipo})$
 $\text{Exp2.mod} = \text{val}$
- **Exp2 → Exp3**
 $\text{Exp3.tsh} = \text{Exp2.tsh}$
 $\text{Exp2.tipo} = \text{Exp3.tipo}$
 $\text{Exp2.mod} = \text{Exp3.mod}$
- **Exp3 → Exp4 OP3 Exp3**
 $\text{Exp3}_1.\text{tsh} = \text{Exp3}_0.\text{tsh}$
 $\text{Exp4.tsh} = \text{Exp3}_0.\text{tsh}$
 $\text{Exp3}_0.\text{tipo} = \text{tipoOP3}(\text{Exp4.tipo}, \text{Exp3}_1.\text{tipo})$
 $\text{Exp3.mod} = \text{val}$
- **Exp3 → Exp4**
 $\text{Exp4.tsh} = \text{Exp3.tsh}$
 $\text{Exp3.tipo} = \text{Exp4.tipo}$
 $\text{Exp3.mod} = \text{Exp4.mod}$
- **Exp4 → OP4A Exp4**
 $\text{Exp4}_1.\text{tsh} = \text{Exp4}_0.\text{tsh}$
 $\text{Exp4}_0.\text{tipo} = \text{tipoOP4A}(\text{OP4A.op}, \text{Exp4}_1.\text{tipo})$
 $\text{Exp4.mod} = \text{val}$

- **Exp4 → OP4NA Exp5**
Exp4.tsh = Exp5.tsh
Exp4.tipo = tipoOP4NA(OP4NA.op, Exp5.tipo)
Exp4.modos=vals
- **Exp4 → Exp5**
Exp5.tsh = Exp4.tsh
Exp4.tipo = Exp5.tipo
Exp4.modos=Exp5.modos
- **Exp5 → (Exp0)**
Exp5.modos=Exp0.modos
Exp0.tsh=Exp5.tsh
Exp5.tipo = Exp0.tipo
- **Exp5 → NNat**
Exp5.tipo=<t:natural>
Exp5.modos=vals
- **Exp5 → NFloat**
Exp5.tipo=<t:float>
Exp5.modos=vals
- **Exp5 → Signo Nnat**
Exp5.tipo=<t:integer>
Exp5.modos=vals
- **Exp5 → true**
Exp5.tipo=<t:boolean>
Exp5.modos=vals
- **Exp5 → false**
Exp5.tipo=<t:boolean>
Exp5.modos=vals
- **Exp5 → null**
Exp5.tipo=<t:punt,tbase=null,tam:0>
Exp5.modos=vals
- **Exp5 → char**
Exp5.tipo=<t:character>
Exp5.modos=vals
- **Exp5 → Desig**
Exp5.tipo=Desig.tipo
Exp5.modos=vars
Desig.tsh=Exp5.tsh

- **Exp5 → | Exp0 |**
Exp5.tipo=valorAbs(Exp0.tipo)
Exp0.tsh=Exp5.tsh
Exp5.modos=val
- **Desig → Id**
Desig.tipo= si existeId(Desig.tsh,id.lex) entonces
 si Desig.tsh[id.lex].clase=pvar entonces
 sigueRef(Desig.tsh[id.lex].tipo,Desig.tsh)
 si no <t:error>
 si no <t:error>
- **Desig → Desig →**
Desig0.tipo= si Desig1.tipo.t = punt entonces
 sigueRef(Desig1.tipo.tbases,Desig0.tsh)
 si no <t:error>
- **Desig → Desig [Exp0]**
Desig0.tipo= si Desig1.tipo.t = array ^ (Exp0.tipo.t=natural v
 Exp0.tipo.t=integer)entonces
 sigueRef(Desig1.tipo.tbases,Desig0.tsh)
 si no <t:error>
- **Desig → Desig.Id**
Desig0.tipo= si Desig1.tipo.t = reg entonces
 si esDuplicado(Desig1.tipo.campos,Id.lex) entonces
 sigueRef(Desig1.tipo.campos[id.lex].tipo,Desig0.tsh)
 si no <t:error>
 si no <t:error>
- **ILec → in (id)**
ILec.err = si ¬existeId(Ilec.tsh, id.lex) entonces true si no false)
- **IEsc → out (Exp0)**
Exp0.tsh=IEsc.tsh
IEsc.err = si Exp0.tipo=err entonces true si no false
- **IIf → if Exp0 then BloqueI Pelse**
Iif.err=BloqueI.err v Pelse.err v Exp0.tipo.t<>boolean
Exp0.tsh=Iif.tsh
Pelse.tsh=Iif.tsh
- **PElse → else BloqueI**
Pelse.err=BloqueI.err
BloqueI.tsh=PElse.tsh
- **PElse → λ**
Pelse.err=false

- **IWhile \rightarrow while Exp0 do Bloquel**
 $\text{Exp0.tsh} = \text{Iwhile.tsh}$
 $\text{Iwhile.err} = \text{Bloquel.err} \vee \text{Exp0.tipo.t} \not\hookrightarrow \text{boolean}$
- **IFor \rightarrow for id = Exp0 to Exp0 do Bloquel**
 $\text{Ifor.err} = \text{Bloquel.err} \vee \neg((\text{Exp0}_0.\text{tipo.t} = \text{natural} \wedge \text{Exp0}_1.\text{tipo.t} = \text{natural}) \vee$
 $(\text{Exp0}_0.\text{tipo.t} = \text{integer} \wedge \text{Exp0}_1.\text{tipo.t} = \text{integer})) \vee$
 $\neg \text{existeId}(\text{Ifor.tsh}, \text{id.lex}) \vee$
 $\neg \text{compatibles}(\text{Ifor.tsh}[\text{id.lex}].\text{tipo.t}, \text{Exp0}_0.\text{tipo}, \text{Ifor.tsh})$
 $\text{Exp0}_0.\text{tsh} = \text{Ifor.tsh}$
 $\text{Exp0}_1.\text{tsh} = \text{Ifor.tsh}$
 $\text{Bloquel.tsh} = \text{Ifor.tsh}$
- **Bloquel \rightarrow {Is}**
 $\text{Bloquel.err} = \text{Is.err}$
 $\text{Is.tsh} = \text{Bloquel.tsh}$
- **Bloquel \rightarrow I**
 $\text{Bloquel.err} = \text{I.err}$
 $\text{I.tsh} = \text{Bloquel.tsh}$
- **ICall \rightarrow Id (Args)**
 $\text{Icall.err} = \neg \text{existeId}(\text{Icall.tsh}, \text{Id.lex}) \vee \text{Icall.tsh}[\text{id.lex}].\text{clase} \not\hookrightarrow \text{proc} \vee \text{Args.err}$
 $\text{Args.tsh} = \text{Icall.tsh}$
 $\text{Args.paramsh} = \text{Icall.tsh}[\text{id.lex}].\text{tipo.params}$
- **Args \rightarrow LArgs**
 $\text{Args.err} = \text{LArgs.err} \vee |\text{Args.paramsh}| \not\hookrightarrow \text{LArgs.nparams}$
 $\text{LArgs.tsh} = \text{Args.tsh}$
 $\text{LArgs.paramsh} = \text{Args.paramsh}$
- **Args $\rightarrow \lambda$**
 $\text{Args.err} = |\text{Args.paramsh}| > 0$
- **LArgs \rightarrow LArgs, Exp0**
 $\text{Largs}_0.\text{err} = \text{LArgs}_1.\text{err} \vee \text{Exp0.tipo.t} = \text{error} \vee$
 $(\text{LArgs}_0.\text{nparams} > \text{LArgs}_0.\text{paramsh}) \vee$
 $(\text{LArgs}_0.\text{paramsh}[\text{LArgs}_0.\text{nparams}].\text{modo} = \text{var} \wedge \text{Exp0.modo} = \text{val}) \vee$
 $\neg \text{compatibles}(\text{LArgs}_0.\text{paramsh}[\text{LArgs}_0.\text{nparams}].\text{tipo}, \text{Exp0.tipo}, \text{LArgs}_0.\text{tsh})$
 $\text{Largs}_0.\text{nparams} = \text{LArgs}_1.\text{nparams} + 1$
 $\text{Largs}_1.\text{tsh} = \text{LArgs}_0.\text{tsh}$
 $\text{Exp0.tsh} = \text{LArgs}_0.\text{tsh}$
 $\text{Largs}_1.\text{paramsh} = \text{LArgs}_0.\text{paramsh}$
- **LArgs \rightarrow Exp0**
 $\text{Largs.err} = |\text{LArgs.paramsh}| = 0 \vee (\text{Largs.paramsh}[1].\text{modo} = \text{var} \wedge$
 $\text{Exp0.modo} = \text{val}) \vee$
 $\neg \text{compatibles}(\text{LArgs.paramsh}[1].\text{tipo}, \text{Exp0.tipo}, \text{LArgs.tsh})$
 $\text{Largs.nparams} = 1$

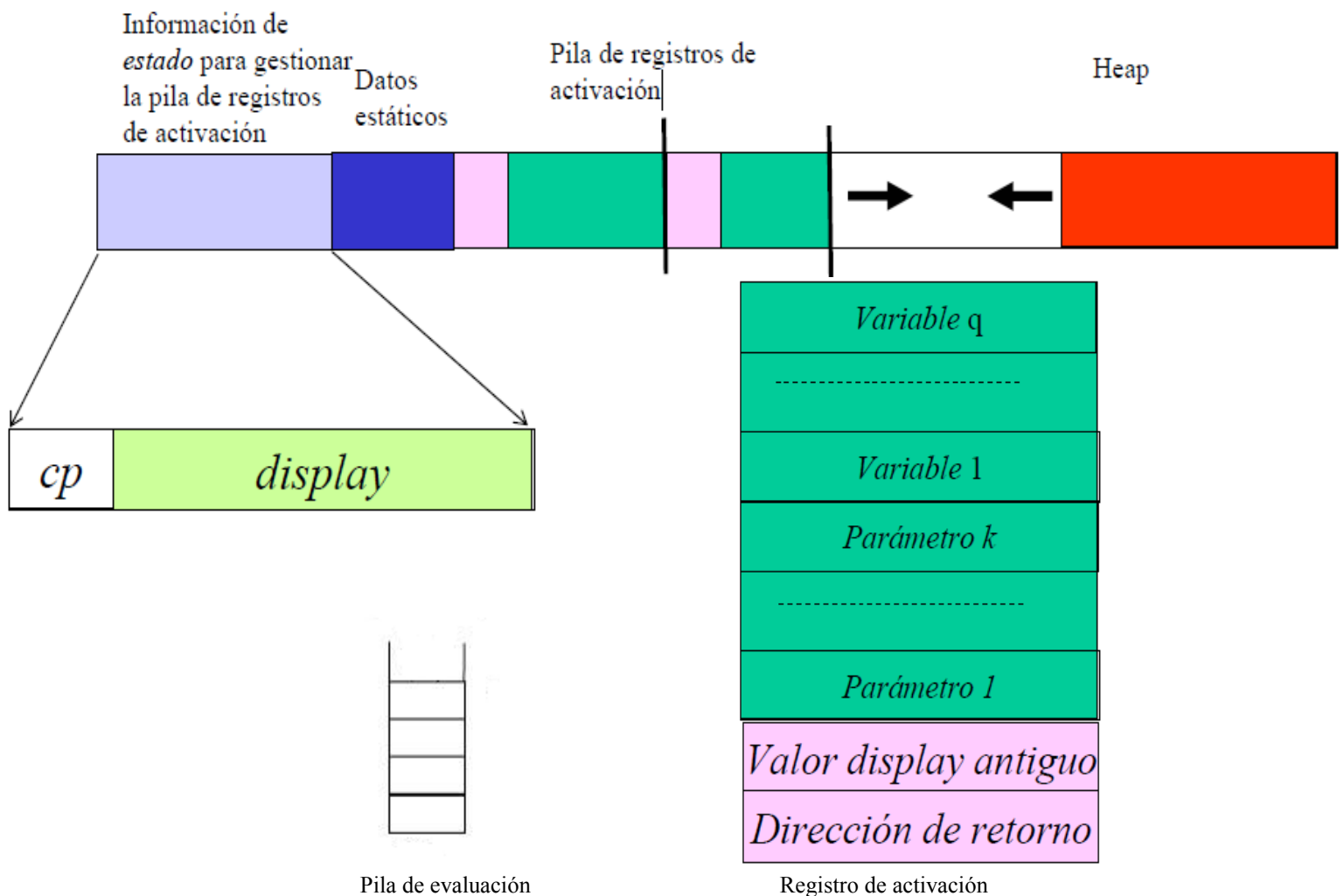
- **INew → new Desig**
INew.err=Desig.tipo.t <>punt
Desig.tsh=INew.tsh
- **IDelete → dispose Desig**
IDelete.err=Desig.tipo.t <>punt
Desig.tsh=IDelete.tsh
- **OP0 → <**
Op0.op=<
- **OP0 → >**
Op0.op=>
- **OP0 → <=**
Op0.op=<=
- **OP0 → >=**
Op0.op=>=
- **OP0 → /=**
Op0.op/=
- **OP0 → =**
Op0.op=
- **OP1 → mas**
Op1.op= +
- **OP1 → menos**
Op1.op= -
- **OP2 → /**
Op2.op= /
- **OP2 → ***
Op2.op= *
- **OP2 → %**
Op2.op= %
- **OP3 → >>**
Op3.op= >>
- **OP3 → <<**
Op3.op= <<
- **OP4A → not**
Op4A.op= not
- **OP4A → menos**
Op4A.op= -
- **OP4NA → (float)**
Op4NA.op=cfloat
- **OP4NA → (nat)**
Op4NA.op=cnat

- **OP4NA \rightarrow (int)**
Op4NA.op=cint
- **OP4NA \rightarrow (char)**
Op4NA.op=cchar

5 Especificación de la traducción

5.1 Lenguaje Objeto

5.1.1 Arquitectura de la Máquina P



La máquina P se compondrá de una pila de evaluación cuyas celdas serán abstractas ya que podrán contener datos de tipos diferentes. Por otra parte también tendrá almacenada información de control (CP + Displays) para proporcionar una estructura que facilite el acceso a los datos globales, así como de los datos estáticos (globales) una serie de registros de activación para gestionar los procedimientos así como el heap para la memoria dinámica. Por último existirá una lista para gestionar los huecos libres de la memoria dinámica (heap).

5.1.2 Instrucciones del Lenguaje Objeto

- **apila_dir(pos)** \equiv lleva a la pila el contenido de la memoria de programas contenida en la posición *pos*.
- **apila(num)** \equiv almacena en la pila un número.
- **desapila** \equiv elimina de la pila la cima
- **desapila_dir(pos)** \equiv transfiere el contenido de la cima de la pila a la dirección de la memoria de programas indicada en la posición *pos*.
- **copia** \equiv copia el valor de la cima de la pila.
- **flip** \equiv intercambia la cima con la subcima de la pila de evaluación.
- **new(tam)** \equiv reserva espacio en el *heap* para *tam* celdas consecutivas y apila en la cima de la pila la dirección de comienzo.
- **dispose(tam)** \equiv desapila una dirección de comienzo *d* de la cima de la pila, y libera en el *heap tam* celdas consecutivas a partir de *d*.
- **ir_a(dir)** \equiv salto incondicional a la dirección especificada en *dir*.
- **ir_f(dir)** \equiv salta si en la cima de la pila hay false a la dirección especificada por *dir*.
- **ir_v(dir)** \equiv salta si en la cima de la pila hay true a la dirección especificada por *dir*.
- **ir_ind** \equiv salta a la dirección indicada en la cima de la pila de evaluación consumiendo dicha cima.
- **apila_ind** \equiv interpreta el valor en la cima de la pila como un número de celda en la memoria y sustituye dicho valor por el almacenado en la celda.
- **desapila_ind** \equiv desapila el valor de la cima y la subcima, interpreta la subcima como un número de celda en la memoria y almacena el contenido de la cima en dicha celda.
- **mueve(tam)** \equiv instrucción encuentra en la cima la dirección origen *o* y en la subcima la dirección destino *d*, y realiza el movimiento de *s* celdas desde *o* a *s*.

IN	Lee un elemento de la consola y lo apila.
OUT	Desapila un elemento de la pila y los muestra por consola.
GREATER	Desapila dos elementos de la pila y compara si la subcima es mayor que la cima.
GREATEREQ	Desapila dos elementos de la pila y compara si la subcima es mayor o igual que la cima.
LESS	Desapila dos elementos de la pila y compara si la subcima es menor que la cima.
LESSEQ	Desapila dos elementos de la pila y compara si la subcima es menor o igual que la cima.
EQUAL	Desapila dos elementos de la pila y compara si la subcima es igual que la cima.
NEQUAL	Desapila dos elementos de la pila y compara si la subcima es distinto que la cima.
ADD	Desapila dos elementos de la pila y los suma.
SUB	Desapila dos elementos de la pila y los resta.
OR	Desapila dos elementos de la pila y hace la or lógica de ellos.

MULT	Desapila dos elementos de la pila y los multiplica.
MOD	Desapila un elemento de la pila y calcula el modulo.
AND	Desapila dos elementos de la pila y calcula el 'y' lógico de ellos.
CFLOAT	Desapila un elemento de la pila y lo convierte a float.
CINT	Desapila un elemento de la pila y lo convierte a entero.
CNAT	Desapila un elemento de la pila y lo convierte a natural.
CCHAR	Desapila un elemento de la pila y lo convierte a character.
DESPDER	Desapila un elemento de la pila y desplaza una posición a la derecha.
DESPIZQ	Desapila un elemento de la pila y lo desplaza un posición a la izquierda.
DIV	Desapila dos elementos de la pila y divide la subcima entre la cima.
NOT	Desapila un elemento de la pila y calcula su negación lógica.
INV	Desapila un elemento de la pila y calcula su inverso.
ABS	Desapila un elemento de la pila y calcula su valor absoluto.

5.2 Funciones Semánticas

```

fun inicio(numNiveles,tamDatos)
    apila(numNiveles+1)
    desapila-dir(1)
    apila(1+numNiveles+tamDatos)
    desapila-dir(0)

```

ffun

```

cons longInicio = 4

```

```

fun max(nivel1,nivel2):integer
    si nivel1>=nivel2 entonces
        devuelve nivel1
    si no
        devuelve nivel2

```

ffun

```

fun prologo(nivel,tamDatosLocales)
    devuelve 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(tamDatosLocales+2) || suma || desapila-dir(0)
ffun

cons longPrologo = 13

fun epilogo(nivel)
    devuelve 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)
ffun

cons longEpilogo = 13

fun compatibleTipoBasico(tipo,ts):boolean
    devuelve compatible(tipo,<t:bool>,ts) v compatible(tipo,<t:integer>,ts) v
    compatible(tipo,<t:natural>,ts) v compatible(tipo,<t:float>,ts) v
    compatible(tipo,<t:character>,ts)
ffun

fun parchea(cod,listav,listaf,etqv,etqf):cod
    para i en listav hacer
        cod[i]:=ir_v(etqv)
    fin_para

    para i en listaf hacer
        cod[i]:=if_f(etqf)
    fin_para
    devuelve cod
ffun

fun accesoVar(idProps):cod
    devuelve apila-dir(1+idProps.nivel) || apila(idProps.dir) || suma ||
    ( si idProps.clase = pvar entonces apila-ind
    si no  $\lambda$  )
ffun

fun longAccesoVar(idProps):integer
    devuelve si idProps.clase = pvar entonces 4
    si no 3
ffun

```

```

fun apilaDirRetorno(ret)
    evuelve apila-dir(0) || apila(1) || suma || apila(ret) ||desapila-ind
ffun

cons longApilaDirRetorno = 5

cons inicio_paso_param = apila-dir(0) || apila(3) || suma

cons fin_paso_param = desapila

cons longInicioPasoParam = 3

cons longFinPasoParam = 1

fun pasoParametro(modoReal,pformal)
    devuelve apila(pformal.dir) || suma || flip || (si pformal.modo = valor ^
    modoReal = var mueve(pformal.tipo.tam) // copia del valor
    si no desapila-ind
ffun

fun longPasoParametro(modoReal,pformal)
    devuelve 4
ffun

```

5.3 Atributos Semánticos

<u>Atributo</u>	<u>Tipo de Atributo</u>	<u>Descripción</u>
cod	sintetizado	Es el código generado hasta el momento realizando un recorrido de izquierda a derecha preferente en profundidad.
etq	sintetizado	Contador de instrucciones del programa
etqh	heredado	Contador de instrucciones del programa que heredan los no terminales
n	sintetizado	Número de niveles de anidamiento
ini	sintetizado	Dirección de comienzo de las instrucciones del procedimiento
parh	heredado	determina si el contexto de ocurrencia de un designador es como parámetro real o no
listav	sintetizado	Instrucciones ir-v pendientes de parchear
listaf	sintetizado	Instrucciones ir-f pendientes de parchear

5.4 Gramática de Atributos

En rojo direcciones a parchear : Esto se solucionará en el esquema de traducción orientado al traductor.

- **Prog \rightarrow Decs & Is**
Prog.cod=inicio(Decs.n,Decs.dir) || ir_a(Decs.etq) || Decs.cod || Is.cod
Decs.etqh=longInicio+1
Is.etqh=Decs.etq
- **Prog \rightarrow & Is**
Prog.cod=inicio(1,0)||Is.cod
Is.etqh=longInicio
- **Decs \rightarrow Decs ; Dec**
Decs₁.etqh = Decs₀.etqh
Dec.etqh = Decs₁.etq
Decs₀.etq = Dec.etq
Decs₀.cod = Decs₁.cod || Dec.cod
Decs₀.n = max(Decs₁.n,Dec.n)
- **Decs \rightarrow Dec**
Dec.etqh = Decs.etqh
Decs.etq = Dec.etq
Decs.cod = Dec.cod
Decs.n=Dec.n
- **Dec \rightarrow DecVar**
Dec.cod = λ
Dec.etq = Dec.etqh
Dec.n=Dec.nh
- **Dec \rightarrow DecTipo**
Dec.cod = λ
Dec.etq = Dec.etqh
Dec.n=Dec.nh
- **Dec \rightarrow DecProc**
DecProc.etqh=Dec.etqh
Dec.etq=DecProc.etq
Dec.cod=DecProc.cod
Dec.n=DecProc.n
Dec.props=DecProc.props \otimes DecProc.ini
- **DecProc \rightarrow procedure Id(Params) Bloque**
DecProc.cod=Bloque.cod
Bloque.etqh=DecProc.etqh
DecProc.etq=Bloque.etq
DecProc.ini=<inicio:Bloque.inicio>
Bloque.tsph = ponId(Params.ts,DecProc.Id,DecProc.props \otimes DecProc.ini)

DecProc.n=Bloque.n

- **Bloque → forwad**

Bloque.etq=Bloque.etqh

Bloque.inicio= -

Bloque.n=Bloque.nh

- **Bloque → {Decs & Is}**

Decs.etqh=Bloque.etqh

Bloque.inicio=Decs.etq

Is.etqh=Decs.etq+ longPrologo

Bloque.etq=Is.etq+longEpilogo+1

Bloque.cod=Decs.cod || prologo(Bloque.nh,Decs.dir) || Is.cod ||
epilogo(Bloque.nh) || ir_ind

Bloque.n=Decs.n

- **Bloque → {& Is}**

Is.etqh = Bloque.etqh + longPrologo

Bloque.inicio=Bloque.etqh

Bloque.etq= Is.etq + longEpilogo + 1

Bloque.cod=prologo(Bloque.nh,Bloque.dirh) || Is.cod || epilogo(Bloque.nh) ||
ir_ind

Bloque.n=Bloque.nh

- **Is → Is ; I**

Is₀.cod = Is₁.cod || I.cod

Is₁.etqh = Is₀.etqh

I.etqh = Is₁.etq

Is₀.etq = I.etq

- **Is → I**

Is.cod = I.cod

Is.etq = I.etq

I.etqh = Is.etqh

- **I → IAsig**

I.cod = IAsig.cod

I.etq = IAsig.etq

IAsig.etqh = I.etqh

- **I → ILec**

I.cod = ILec.cod

I.etq = ILec.etq

ILec.etqh = I.etqh

- **I → IEsc**

I.cod = IEsc.cod

I.etq = IEsc.etq

IEsc.etqh = I.etqh

- **I → IIf**
I.cod = IIf.cod
I.etq = IIf.etq
IIf.etqh = I.etqh
- **I → IWhile**
I.cod = IWhile.cod
I.etq = IWhile.etq
IWhile.etqh = I.etqh
- **I → IFor**
I.cod = IFor.cod
I.etq = IFor.etq
IFor.etqh = I.etqh
- **I → ICall**
Icall.etqh=I.etqh
I.etq=Icall.etq
I.cod=Icall.cod
- **I → INew**
I.cod = INew.cod
I.etq = INew.etq
INew.etqh = I.etqh
- **I → IDelete**
I.cod = IDelete.cod
I.etq = IDelete.etq
IDelete.etqh = I.etqh
- **IAsig → Desig := Exp0**
Exp0.parh=false
IAsig.cod= si compatibleTipoBasico(Desig.tipo,IAsig.tsh) entonces
Desig.cod || Exp0.cod || desapila_ind
si no Desig.cod || Exp0.cod || mueve (Desig.tipo.tam)
Desig.etqh=IAsig.etqh
Exp0.etqh=Desig.etq
IAsig.etq=Exp0.etq+1
- **Exp0 → Exp1 OP0 Exp1**
Exp0.cod=parchea(Exp1₀.cod,Exp1₀.listav,Exp1₀.listaf,Exp1₀.etq,Exp1₀.etq) ||
parchea(Exp1₁.cod,Exp1₁.listav,Exp1₁.listaf,Exp1₁.etq,Exp1₁.etq)
|| OP0.cod
Exp1₀.parh=false
Exp1₁.parh=false
Exp0.listav=[]
Exp0.listaf=[]
Exp1₀.etqh=Exp0.etqh
Exp1₁.etqh=Exp1₀.etq
Exp0.etq=Exp1₁.etq+1

- **Exp0 → Exp1**
Exp0.cod=Exp1.cod
Exp1.parh=Exp0.parh
Exp0.listav=Exp1.listav
Exp0.listaf=Exp1.listaf
Exp1.etqh=Exp0.etqh
Exp0.etq=Exp1.etq
- **Exp1 → Exp1 OP1 Exp2**
Exp1₀.cod=parchea(Exp1₁.cod,Exp1₁.listav,Exp1₁.listaf,Exp1₁.etq,Exp1₁.etq) ||
parchea(Exp2.cod,Exp2.listav,Exp2.listaf,Exp2.etq,Exp2.etq) ||
OP1.cod
Exp1₁.parh=false
Exp2.parh=false
Exp1₀.listav=[]
Exp1₀.listaf=[]
Exp1₁.etqh=Exp1₀.etqh
Exp2.etqh=Exp1.etq
Exp1₀.etq=Exp2.etq+1
- **Exp1 → Exp1 or Exp2**
Exp1₀.cod=parchea(Exp1₁.cod,[],Exp1₁.listaf?,Exp1₁.etq+2) || copia || ir-v(?) ||
desapila ||parchea(Exp2.cod,[],Exp2.listaf?,Exp2.etq))
Exp1₀.listav = Exp1₁.listav || Exp2.listav || [Exp1₁.etq + 1]
Exp1₀.listaf = []
Exp1₁.parh=false
Exp2.parh=false
Exp1₁.etqh=Exp1₀.etqh
Exp2.etqh=Exp1₁.etq+3
Exp1₀.etq=Exp2.etq
- **Exp1 → Exp2**
Exp1.cod=Exp2.cod
Exp2.parh=Exp1.parh
Exp1.listav=Exp2.listav
Exp1.listaf=Exp2.listaf
Exp2.etqh=Exp1.etqh
Exp1.etq=Exp2.etq
- **Exp2 → Exp2 OP2 Exp3**
Exp2₀.cod=parchea(Exp2₁.cod,Exp2₁.listav,Exp2₁.listaf,Exp2₁.etq,Exp2₁.etq) ||
parchea(Exp3.cod,Exp3.listav,Exp3.listaf,Exp3.etq,Exp3.etq) ||
OP2.cod
Exp2₁.parh=false
Exp3.parh=false
Exp2₀.listav=[]


```

Exp20.listaf=[]
Exp21.etqh=Exp20.etqh
Exp3.etqh=Exp21.etq
Exp20.etq=Exp3.etq

```

- **Exp2 → Exp2 and Exp3**

```

Exp20.cod = parchea(Exp21.cod,Exp21.listav,[],Exp21.etq+2,?) || copia ||
  ir-f(?) || desapila || parchea(Exp3.cod,Exp3.listav,[],Exp3.etq,?)
Exp20.listf = Exp21.listaf || Exp3.listaf || [Exp21.etq + 1]
Exp20.listav = []
Exp21.parh=false
Exp3.parh=false
Exp21.etqh=Exp20.etqh
Exp3.etqh=Exp21.etq + 3
Exp20.etq=Exp3.etq

```

- **Exp2 → Exp3**

```

Exp2.cod=Exp3.cod
Exp3.parh=Exp2.parh
Exp2.listav=Exp3.listav
Exp2.listaf=Exp3.listaf
Exp3.etqh=Exp2.etqh
Exp2.etq=Exp3.etq

```

- **Exp3 → Exp4 OP3 Exp3**

```

Exp30.cod=parchea(Exp4.cod,Exp4.listav,Exp4.listaf,Exp4.etq,Exp4.etq) ||
  parchea(Exp31.cod,Exp31.listav,Exp31.listaf,Exp31.etq,Exp31.etq) ||
  OP3.cod

```

```

Exp4.parh=false
Exp31.parh=false
Exp30.listav=[]
Exp30.listaf=[]
Exp4.etqh=Exp30.etqh
Exp31.etqh=Exp4.etq
Exp30.etq=Exp31.etq

```

- **Exp3 → Exp4**

```

Exp3.cod=Exp4.cod
Exp4.parh=Exp3.parh
Exp3.listav=Exp4.listav
Exp3.listaf=Exp4.listaf
Exp4.etqh=Exp4.etqh
Exp3.etq=Exp4.etq

```

- **Exp4 → OP4A Exp4**
Exp4₀.cod=parchea(Exp4₁.cod,Exp4₁.listav,Exp4₁.listaf,Exp4₁.etq,Exp4₁.etq)
|| OP4A.cod
Exp4₁.parh=false
Exp4₀.listav=[]
Exp4₀.listaf=[]
Exp4₁.etqh=Exp4₀.etqh
Exp4₀.etq=Exp4₁.etq+1
- **Exp4 → OP4NA Exp5**
Exp4.cod=parchea(Exp5.cod,Exp5.listav,Exp5.listaf,Exp5.etq,Exp5.etq) ||
OP4NA.cod
Exp5.parh=false
Exp4.listav=[]
Exp4.listaf=[]
Exp5.etqh=Exp4.etqh
Exp4.etq=Exp5.etq+1
- **Exp4 → Exp5**
Exp4.cod=Exp5.cod
Exp5.parh=Exp4.parh
Exp4.listav=Exp5.listav
Exp4.listaf=Exp5.listaf
Exp5.etqh=Exp4.etqh
Exp4.etq=Exp5.etq
- **Exp5 → (Exp0)**
Exp5.cod=Exp0.cod
Exp0.parh=Exp5.parh
Exp5.listav=Exp0.listav
Exp5.listaf=Exp0.listaf
Exp0.etqh=Exp5.etqh
Exp5.etq=Exp0.etq
- **Exp5 → NNat**
Exp5.cod=apila(NNat.lex)
Exp5.listav=[]
Exp5.listaf=[]
Exp5.etq=Exp5.etqh+1
- **Exp5 → NFloat**
Exp5.cod=apila(NFloat.lex)
Exp5.listav=[]
Exp5.listaf=[]
Exp5.etq=Exp5.etqh+1

- **Exp5 → Signo Nnat**
 Exp5.cod=si Signo.lex=- entonces apila(NNat.lex) || inv
 si no apila(NNat.lex)
 Exp5.listav=[]
 Exp5.listaf=[]
 Exp5.etq= si Signo.lex=- entonces Exp5.etqh+2
 si no Exp5.etqh + 1
- **Exp5 → true**
 Exp5.cod=apila(true)
 Exp5.listav=[]
 Exp5.listaf=[]
 Exp5.etq=Exp5.etqh+1
- **Exp5 → false**
 Exp5.cod=apila(false)
 Exp5.listav=[]
 Exp5.listaf=[]
 Exp5.etq=Exp5.etqh+1
- **Exp5 → null**
 Exp5.cod=apila(null)
 Exp5.listav=[]
 Exp5.listaf=[]
 Exp5.etq=Exp5.etqh+1
- **Exp5 → char**
 Exp5.cod=apila(char.lex)
 Exp5.listav=[]
 Exp5.listaf=[]
 Exp5.etq=Exp5.etqh+1
- **Exp5 → Desig**
 Exp5.cod= si compatibleTipoBasico(Desig.tipo,Desig.tsh) ^ ¬Exp5.parh
 entonces
 Desig.cod || apila_ind
 si no Desig.cod
 Desig.etqh=Exp5.etqh
 Exp5.etq= si compatibleTipoBasico(Desig.tipo,Desig.tsh) ^ ¬Exp5.parh
 entonces
 Desig.etq + 1
 si no Desig.etq
 Exp5.listav=[]
 Exp5.listaf=[]

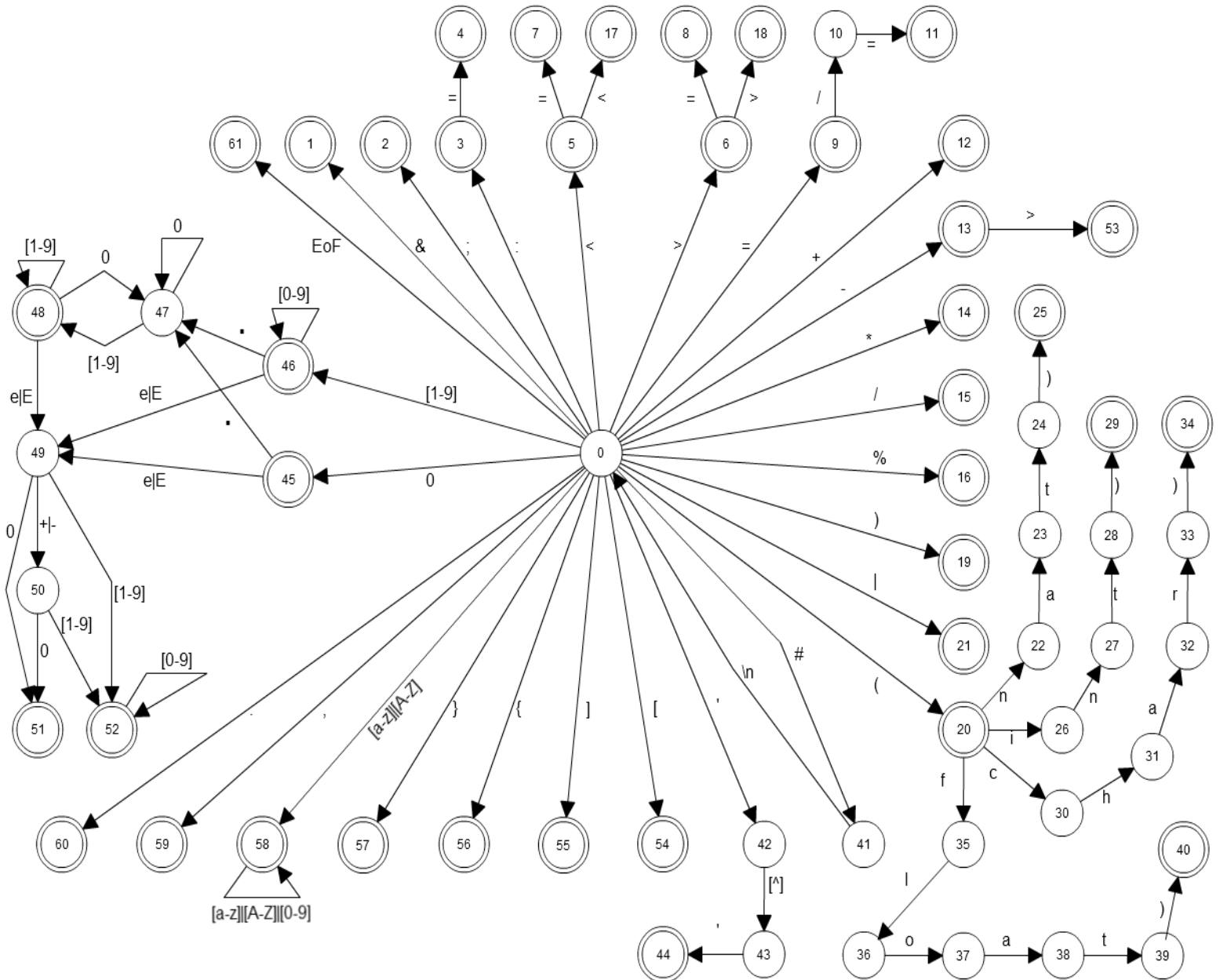
- **Exp5 → | Exp0 |**
Exp5.cod = Exp0.cod || abs
Exp0.parh=Exp5.parh
Exp5.listav=Exp0.listav
Exp5.listaf=Exp0.listaf
Exp0.etqh=Exp5.etqh
Exp5.etq=Exp0.etq + 1
- **Desig → Id**
Desig.cod=accesoVar(Desig.tsh[id.lex])
Desig.etq=Desig.etqh + longAccesoVar(Desig.tsh[id.lex])
- **Desig → Desig →**
Desig₀.cod=Desig₁.cod || apila_ind
Desig₁.etqh=Desig₀.etqh
Desig₀.etq=Desig₁.etq+1
- **Desig → Desig [Exp0]**
Desig₀.cod=Desig₁.cod || Exp0.cod || apila(Desig₁.tipo.tbases.tam) || multiplica
|| suma
Desig₁.etqh=Desig₀.etqh
Exp0.etqh=Desig₁.etq
Desig₀.etq=Exp0.etq + 3
Exp0.parh=false
- **Desig → Desig.Id**
Desig₀.cod=Desig₁.cod || apila(Desig₁.tipo.campos[id.lex].desp)|| suma
Desig₁.etqh=Desig₀.etqh
Desig₀.etq=Desig₁.etq+2
- **ILec → in (id)**
ILec.cod = read || apila_dir(ILec.tsh[id.lex].dir)
ILec.etq=ILec.etqh+2
- **IEsc → out (Exp0)**
IEsc.cod = Exp0.cod || write
Exp0.etqh=IEsc.etqh
IEsc.etq=Exp0.etq+1
Exp0.parh=false
- **IIf → if Exp0 then BloqueI PElse**
IIf.cod=parchea(Exp0.cod,Exp0.listav,Exp0.listaf,Exp0.etq,Exp0.etq) ||
ir_f(PElse.etqi) || BloqueI.cod || PElse.cod
Exp0.parh=false
Exp0.etqh=IIf.etqh
BloqueI.etqh=Exp0.etq+1
PElse.etqh=BloqueI.etq
IIf.etq=PElse.etq

- **PElse \rightarrow else BloqueI**
PElse.cod=ir_a(BloqueI.etq) || BloqueI.cod
BloqueI.etqh=PElse.etqh+1
PElse.etqi=PElse.etqh+1
PElse.etq=BloqueI.etq
- **PElse $\rightarrow \lambda$**
PElse.cod= λ
PElse.etq=PElse.etqh
PElse.etqi=PElse.etqh
- **IWhile \rightarrow while Exp0 do BloqueI**
Iwhile.cod=parchea(Exp0.cod,Exp0,listav,Exp0.listaf,Exp0.etq,Exp0.etq) ||
ir_f(BloqueI.etq+1)||BloqueI.cod || ir_a(IWhile.etqh)
Exp0.parh=false
Exp0.etqh=IWhile.etqh
BloqueI.etqh=Exp0.etq+1
Iwhile.etq=BloqueI.etq+1
- **IFor \rightarrow for id = Exp0 to Exp0 do BloqueI**
IFor.cod=parchea(Exp0₀.cod,Exp0₀,listav,Exp0₀.listaf,Exp0₀.etq,Exp0₀.etq)||
copia || desapila_dir(IFor.tsh[id.lex].dir)
parchea(Exp0₁.cod,Exp0₁,listav,Exp0₁.listaf,Exp0₁.etq,Exp0₁.etq) ||
menorigual || ir_f(IFor.etq) || BloqueI.cod ||
apila_dir(IFor.tsh[id.lex].dir) || apila(1) || suma || ir_a(Exp0₀.etq)
Exp0₀.parh=false
Exp0₁.parh=false
Exp0₀.etqh=IFor.etqh
Exp0₁.etqh=Exp0₀.etq + 2
BloqueI.etqh=Exp0₁.etq + 2
IFor.etq=BloqueI.etq + 4
- **BloqueI \rightarrow {Is}**
BloqueI.cod = Is.cod
Is.etqh=BloqueI.etqh
BloqueI.etq=Is.etq
- **BloqueI \rightarrow I**
BloqueI.cod = I.cod
I.etqh=BloqueI.etqh
BloqueI.etq=I.etq
- **ICall \rightarrow Id (Args)**
Args.etqh=ICall.etqh + longApilaDirRetorno
ICall.etq=Args.etq + 1
ICall.cod=apilaDirRetorno(**ICall.etq**) || Args.cod || ir-a (ICall.tsh[id.lex].inicio)

- **Args → LArgs**
Args.cod= inicio_paso_param || LArgs.cod || fin_paso_param
LArgs.etqh=Args.etqh + longInicioPasoParam
Args.etq=LArgs.etq + longFinPasoParam
- **Args → λ**
Args.cod= λ
Args.etq=Args.etqh
- **LArgs → LArgs, Exp0**
LArgs₀.cod = LArgs₁.cod || copia || Exp0.cod || flip ||
pasoParametro(Exp0.mod0,LArgs₀.params[LArgs₀.nparams])
LArgs₁.etqh = LArgs₀.etqh
Exp0.etqh = LArgs₁.etq+1
LArgs₀.etq = Exp.etq + 1 +
longPasoParametro(Exp0.mod0,LArgs₀.params[LArgs₀.nparams])
Exp0.parh= LArgs₀.paramsh[LArgs₀.nparams].modo=var
- **LArgs → Exp0**
LArgs.cod = copia || Exp0.cod || flip ||
pasoParametro(Exp0.mod0,LArgs.params[1])
Exp0.etqh = LArgs.etq+1
LArgs.etq = Exp0.etq + 1 +
longPasoParametro(Exp0.mod0,LArgs.params[1])
Exp0.parh= LArgs.paramsh[1].modo=var
- **INew → new Desig**
INew.cod = Desig.cod || new(si Desig.tipo.t = ref entonces
INew.tsh[Desig.tipo.id].tam
sino Desig.tipo.tbases.tam) || desapila-ind
Desig.etqh = INew.etqh
INew.etq = Desig.etq + 2
- **IDelete → dispose Desig**
IDelete.cod = Desig.cod || dispose(si Desig.tipo.t = ref entonces
IDelete.tsh[Desig.tipo.id].tam
si no Desig.tipo.tbases.tam)
Desig.etqh = IDelete.etqh
IDelete.etq = Desig.etq + 1
- **OP0 → <**
Op0.cod=menor
- **OP0 → >**
Op0.cod=mayor
- **OP0 → <=**
Op0.cod=menorigual
- **OP0 → >=**
Op0.cod=mayorigual
- **OP0 → /=**
Op0.cod=distinto

- **OP0 → =**
Op0.cod=igual
- **OP1 → +**
Op1.cod=suma
- **OP1 → -**
Op1.cod=resta
- **OP2 → /**
Op2.cod=div
- **OP2 → ***
Op2.cod=mul
- **OP2 → %**
Op2.cod=mod
- **OP3 → >>**
Op3.cod=despDer
- **OP3 → <<**
Op3.cod=despIzq
- **OP4A → not**
Op4A.cod=not
- **OP4A → -**
Op4A.cod=inv
- **OP4NA → (float)**
Op4NA.cod=cfloat
- **OP4NA → (nat)**
Op4NA.cod=cnat
- **OP4NA → (int)**
Op4NA.cod=cint
- **OP4NA → (char)**
Op4NA.cod=cchar

6 Diseño del Analizador Léxico



7 Acondicionamiento de las Gramáticas de Atributos

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

- **Decs \rightarrow Dec RDecs**
RDecs.tsph= ponId(Decs.tsph,Dec.id,Dec.props)
Dec.dirh=Decs.dirh
RDecs.dirh=Dec.dirh+Dec.tam
Dec.tsph=Decs.tsph
Decs.ts=RDecs.ts
Decs.dir=RDecs.dir
Dec.nh=Decs.nh
RDecs.nh=Decs.nh
- **RDecs \rightarrow ; Dec RDecs**
RDecs1.tsph= ponId(RDecs0.tsph,Dec.id,Dec.props)
Dec.dirh=RDecs0.dirh
RDecs1.dirh=RDecs0.dirh+Dec.tam
Dec.tsph=RDecs0.tsph
RDecs0.ts=RDecs1.ts
RDecs0.dir=RDecs1.dir
Dec.nh=RDecs0.nh
RDecs1.nh=RDecs0.nh
- **RDecs $\rightarrow \lambda$**
RDecs.dir=RDecs.dirh
RDecs.ts=RDecs.tsph
- **Campos \rightarrow Campo RCampos**
Campo.desph=0
RCampos.desph=Campo.tam
RCampos.camposh=[Campo.campo]
RCampos.tamh=Campo.tam
Campos.tam=RCampos.tam
Campos.campos=RCampos.campos
- **RCampos \rightarrow ; Campo RCampos**
Campo.desph=RCampos0.desph
RCampos.desph=RCampos0.desph+Campo.tam
RCampos1.camposh=RCampos0.camposh || [Campo.campo]
RCampos1.tamh=RCampos0.tamh+Campo.tam
RCampos0.tam=RCampos1.tam

RCampos₀.campos=RCampos₁.campos

- **RCampos** → λ

RCampos.tam=RCampos.tamh

RCampos.campos=RCampos.camposh

- **LParams** → Param **RLParams**

RLParams.paramsh=[Param.param]

Param.dirh=0

Param.nh=LParams.nh

RLParams.nh=LParams.nh

RLParams.dirh=Param.tam

RLParam.tsph=ponID(LParams.tsph,Param.id,Param.props)

LParams.ts=RLParams.ts

LParams.dir=RLParams.dir

Lparam.params=RLParams.params

- **RLParams** → , Param **RLParams**

RLParams₁.paramsh=RLParams₀.paramsh || [Param.param]

Param.dirh=RLParams₀.dirh

Param.nh=RLParams₀.nh

RLParams₁.nh=RLParams₀.nh

RLParams₁.dirh=RLParams₀.dirh + Param.tam

RLParam₁.tsph=ponID(RLParams₀.tsph,Param.id,Param.props)

RLParams₀.ts=RLParams₁.ts

RLParams₀.dir=RLParams₁.dir

RLParam₀.params=RLParams₁.params

- **RLParams** → λ

RLParams.dir=RLParams.dirh

RLParams.params=RLParams.paramsh

RLParams.ts=RLParams.tsph

- **Bloque** → { **RBloque**

Bloque.fwd=false

RBloque.tsph=Bloque.tsph

RBloque.dirh=Bloque.dirh

RBloque.nh=Bloque.nh

Bloque.ts=RBloque.ts

- **RBloque** → Decs & Is }

Decs.tsph=RBloque.tsph

Decs.dirh=RBloque.dirh

Decs.nh=RBloque.nh

RBloque.ts=Decs.ts

Is.tsh=Decs.ts

- **RBloque \rightarrow & Is }**
Is.tsh=RBloque.tsph

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

- **Decs \rightarrow Dec RDecs**
Decs.err=RDecs.err
RDecs.errh=Dec.err v
(existeId(Decs.tsh,Dec.Id)^Decs.tsh[Dec.id].nivel=Decs.nh^
Dec.props.clase \diamond proc)
Decs.pend=RDecs.pend
RDecs.pendh=Dec.pend - (Si Dec.props.clase=tipo entonces {Dec.id} sino {})
Dec.tsh=Decs.tsh
RDecs.tsh=Dec.ts
RDecs.procpendh=Dec.procpend - (si Dec.fwd=false ^
Dec.props.clase=proc entonces {Dec.id} sino {})
Decs.procpend=RDecs.procpend
- **RDecs \rightarrow ; Dec RDecs**
RDecs₀.err=RDecs₁.err
RDecs₁.errh= RDecs₀.errh v Dec.err v (
existeId(RDecs₀.tsh,Dec.Id)^RDecs₀.tsh[Dec.id].nivel=RDecs₀.nh ^
Dec.props.clase \diamond proc) v (existeId(RDecs₀.tsh,Dec.Id) ^
RDecs₀.tsh[Dec.id].nivel=RDecs₀.nh
(\neg pendientes(Dec.id,RDecs₀.procpendh)) v (Dec.fwd))
RDecs₀.pend=RDecs₁.pend
RDecs₁.pendh=RDecs₀.pend U Dec.pend - (Si Dec.props.clase=tipo
entonces {Dec.id} sino {})

Dec.tsh=RDecs₀.tsh
RDecs₁.tsh=RDecs₀.ts
RDecs₁.procpendh=RDecs₀.procpendh U Dec.procpend - (si Dec.fwd=false ^
Dec.props.clase=proc entonces {Dec.id} sino {})
RDecs₀.procpend=RDecs₁.procpend
- **RDecs \rightarrow λ**
RDecs.pend=RDecs.pendh
RDecs.procpend=RDecs.procpendh
RDecs.err=RDecs.errh

- **Campos \rightarrow Campo RCampos**
Campos.err=RCampos.err
Campos.pend=RCampos.pend
Campo.tsh=Campos.tsh
RCampos.tsh=Campos.tsh
RCampos.errh=Campo.err
RCampos.pendh=Campo.pend
- **RCampos \rightarrow ; Campo RCampos**
RCampos₀.err=RCampos₁.err
RCampos₀.pend=RCampos₁.pend
Campo.tsh=RCampos₀.tsh
RCampos₁.tsh=RCampos₀.tsh
RCampos₁.errh=RCampos₀.errh v Campo.err v
esDuplicado(RCampos₀.camposh,Campo.id)
RCampos₁.pendh=RCampos₀.pendh U Campo.pend
- **RCampos \rightarrow λ**
RCampos.pend=RCampos.pendh
RCampos.err=RCampos.errh
- **LParams \rightarrow Param RLParams**
LParams.err=RLParams.err
RLParams.errh=Param.err
- **RLParams \rightarrow , Param RLParams**
RLParams₀.err=RLParams₁.err
RLParams₁.errh=RLParams₀.errh v Param.err v
(existeId(RLParams₀.tsph,Param.id
^RLParams₀.tsph[Param.id].nivel=RLParams₀.nh)
- **RLParams \rightarrow λ**
RLParams.err=RLParams.errh
- **Bloque \rightarrow { RBloque**
Bloque.err=RBloque.err
RBloque.tsh=Bloque.tsh
Bloque.procpend=RBloque.procpend
- **RBloque \rightarrow Decs & Is }**
RBloque.err=Decs.err v Is.err
Is.tsh=Decs.ts
RBloque.procpend=Decs.procpend

- **RBloque \rightarrow & Is }**
RBloque.err=Is.err
Is.tsh=RBloque.tsh
RBloque.procpnd={}
- **Is \rightarrow I RIs**
RIs.tsh=Is.tsh
I.tsh=Is.tsh
RIs.errh=I.err
Is.err=RIs.err
- **RIs \rightarrow ; I RIs**
I.tsh=RI₀.tsh
RI₁.tsh=RI₀.tsh
RI₁.errh=RI₀.errh \vee I.err
RI₀.err=RI₁.err
- **RIs \rightarrow λ**
RIs.err=RIs.errh
- **Exp0 \rightarrow Exp1 RExp0**
Exp0.tipo = RExp0.tipo
RExp0.tipoh = Exp1.tipo
Exp1.tsh = Exp0.tsh
RExp0.tsh = Exp0.tsh
RExp0.modoh=Exp1.modoh
Exp0.modoh=RExp0.modoh
- **RExp0 \rightarrow OP0 Exp1**
Exp1.tsh = RExp0.tsh
RExp0.tipo = tipoOp0(RExp0.tipoh, Exp1.tipo)
RExp0.modoh=val
- **RExp0 \rightarrow λ**
RExp0.tipo = Rexp0.tipoh
RExp0.modoh=RExp0.modoh
- **Exp1 \rightarrow Exp2 RExp1**
Exp1.tipo = RExp1.tipo
RExp1.tipoh = Exp2.tipo
Exp2.tsh = Exp1.tsh
RExp1.tsh = Exp1.tsh
RExp1.modoh=Exp2.modoh
Exp1.modoh=RExp1.modoh
- **RExp1 \rightarrow OP1 Exp2 RExp1**
Exp2.tsh = RExp1₀.tsh
RExp1₁.tsh = RExp1₀.tsh
RExp1₁.tipoh = tipoOp1(RExp1₀.tipoh, Exp2.tipo)

$\text{RExp1}_0.\text{tipo} = \text{RExp1}_1.\text{tipo}$

$\text{RExp1}_0.\text{modo} = \text{val}$

- **$\text{RExp1} \rightarrow \text{or Exp2 RExp1}$**

$\text{Exp2.tsh} = \text{RExp1}_0.\text{tsh}$

$\text{RExp1}_1.\text{tsh} = \text{RExp1}_0.\text{tsh}$

$\text{RExp1}_1.\text{tipoh} = \text{tipoOr}(\text{RExp1}_0.\text{tipoh}, \text{Exp2.tipo})$

$\text{RExp1}_0.\text{tipo} = \text{RExp1}_1.\text{tipo}$

$\text{RExp1}_0.\text{modo} = \text{val}$

- **$\text{RExp1} \rightarrow \lambda$**

$\text{RExp1.tipo} = \text{RExp1.tipoh}$

$\text{RExp1.modo} = \text{RExp1.modoh}$

- **$\text{Exp2} \rightarrow \text{Exp3 RExp2}$**

$\text{Exp2.tipo} = \text{RExp2.tipo}$

$\text{RExp2.tipoh} = \text{Exp3.tipo}$

$\text{Exp3.tsh} = \text{Exp2.tsh}$

$\text{RExp1.tsh} = \text{Exp2.tsh}$

$\text{RExp2.modoh} = \text{Exp3.modo}$

$\text{Exp2.modo} = \text{RExp2.modo}$

- **$\text{RExp2} \rightarrow \text{OP2 Exp3 RExp2}$**

$\text{Exp3.tsh} = \text{RExp2}_0.\text{tsh}$

$\text{RExp2}_1.\text{tsh} = \text{RExp2}_0.\text{tsh}$

$\text{RExp2}_1.\text{tipoh} = \text{tipoOp2}(\text{RExp2}_0.\text{tipoh}, \text{Exp3.tipo})$

$\text{RExp2}_0.\text{tipo} = \text{RExp2}_1.\text{tipo}$

$\text{RExp2}_0.\text{modo} = \text{val}$

- **$\text{RExp2} \rightarrow \text{and Exp3 RExp2}$**

$\text{Exp3.tsh} = \text{RExp2}_0.\text{tsh}$

$\text{RExp2}_1.\text{tsh} = \text{RExp2}_0.\text{tsh}$

$\text{RExp2}_1.\text{tipoh} = \text{tipoAnd}(\text{RExp2}_0.\text{tipoh}, \text{Exp3.tipo})$

$\text{RExp2}_0.\text{tipo} = \text{RExp2}_1.\text{tipo}$

$\text{RExp2}_0.\text{modo} = \text{val}$

- **$\text{RExp2} \rightarrow \lambda$**

$\text{RExp2.tipo} = \text{RExp2.tipoh}$

$\text{RExp2.modo} = \text{RExp2.modoh}$

- **$\text{Exp3} \rightarrow \text{Exp4 RExp3}$**

$\text{Exp3.tipo} = \text{RExp4.tipo}$

$\text{RExp3.tipoh} = \text{Exp4.tipo}$

$\text{Exp4.tsh} = \text{Exp3.tsh}$

$\text{RExp3.tsh} = \text{Exp3.tsh}$

$\text{RExp3.modoh} = \text{Exp4.modo}$

$\text{Exp3.modo} = \text{RExp3.modo}$

- **RExp3 → OP3 Exp4**
Exp4.tsh = RExp3.tsh
RExp3.tipo = tipoOp3(RExp3.tipoh, Exp4.tipo)
RExp3.modos=vals
- **RExp3 → λ**
RExp3.tipo = RExp3.tipoh
RExp3.modos=RExp3.modoh
- **Desig → Id RDesig**
Desig.tipo=RDesig.tipo
RDesig.tipoh=si existeId(Desig.tsh,id.lex) entonces
 si Desig.tsh[id.lex].clase=var entonces
 sigueRef(Desig.tsh[id.lex].tipo,Desig.tsh)
 si no <t:error>
 si no <t:error>
- **RDesig → → RDesig**
RDesig₀.tipo=RDesig₁.tipo
RDesig₁.tipoh=si RDesig₀.tipoh.t = punt entonces
 sigueRef(RDesig₀.tipoh.tbases,RDesig₀.tsh)
 si no <t:error>
- **RDesig → [Exp0] RDesig**
RDesig₀.tipo=RDesig₁.tipo
RDesig₁.tipoh= si RDesig₀.tipoh.t = array ^ (Exp0.tipo.t=natural v
 Exp0.tipo.t=integer)entonces
 sigueRef(RDesig₀.tipoh.tbases,RDesig₀.tsh)
 si no <t:error>
- **RDesig → .Id RDesig**
RDesig₀.tipo=RDesig₁.tipo
RDesig₁.tipoh= si RDesig₀.tipoh.t = reg entonces
 si esDuplicado(RDesig₀.tipoh.campos,Id.lex) entonces
 sigueRef(RDesig₀.tipoh.campos[id.lex].tipo,RDesig₀.tsh)
 si no <t:error>
 si no <t:error>
- **RDesig → λ**
RDesig.tipo=RDesig.tipoh
- **LArgs → Exp0 RLArgs**
LArgs.err=RLArgs.err
LArgs.nparams=RLArgs.nparams
RLArgs.tsh=LArgs.tsh
Exp0.tsh=LArgs.tsh
RLArgs.nparamsh=1
RLArgs.errh=|LArgs.paramsh|=0 v (LArgs.paramsh[1].modo=var ^

$\text{Exp0.mod0=val) } \vee$
 $\neg \text{compatibles(LArgs.paramsh[1].tipo,Exp0.tipo,LArgs.tsh)}$
 $\text{RLArgs.paramsh=LArgs.paramsh}$

- RLArgs \rightarrow , Exp0 RLArgs**
 $\text{RLArgs}_0.\text{err}=\text{RLArgs}_1.\text{err}$
 $\text{RLArgs}_0.\text{nparams}=\text{RLArgs}_1.\text{nparams}$
 $\text{RLArgs}_1.\text{tsh}=\text{RLArgs}_0.\text{tsh}$
 $\text{Exp0.tsh}=\text{RLArgs}_0.\text{tsh}$
 $\text{RLArgs}_1.\text{nparamsh}=1 + \text{RLArgs}_0.\text{nparamsh}$
 $\text{RLArgs}_1.\text{errh}=(\text{RLArgs}_0.\text{nparamsh}>\text{RLArgs}_0.\text{paramsh}) \vee$
 $(\text{RLArgs}_0.\text{paramsh}[\text{RLArgs}_0.\text{nparamsh}].\text{modo}=\text{var} \wedge$
 $\text{Exp0.mod0=val) } \vee$
 $\neg \text{compatibles(RLArgs}_0.\text{paramsh}[\text{RLArgs}_0.\text{nparamsh}].\text{tipo},$
 $\text{Exp0.tipo,RLArgs}_0.\text{tsh})$
 $\text{RLArgs}_1.\text{paramsh}=\text{RLArgs}_0.\text{paramsh}$
- RLArgs $\rightarrow \lambda$**
 $\text{RLArgs.err}=\text{RLArgs.errh}$
 $\text{RLArgs.nparams}=\text{RLArgs.nparamsh}$

7.3 Acondicionamiento de la gramática para la traducción

- Decs \rightarrow Dec RDecs**
 $\text{Dec.etqh}=\text{Decs.etqh}$
 $\text{RDecs.etqh}=\text{Dec.etq}$
 $\text{Decs.etq}=\text{RDecs.etq}$
 $\text{Decs.cod}=\text{RDecs.cod}$
 $\text{RDecs.codh}=\text{Dec.cod}$
 $\text{Decs.n}=\max(\text{Dec.n,RDecs.n})$
- RDecs \rightarrow ; Dec RDecs**
 $\text{Dec.etqh}=\text{RDecs}_0.\text{etqh}$
 $\text{RDecs}_1.\text{etqh}=\text{Dec.etq}$
 $\text{Decs.etq}=\text{RDecs}_1.\text{etq}$
 $\text{Decs.cod}=\text{RDecs}_1.\text{cod}$
 $\text{RDecs}_1.\text{codh}=\text{RDecs}_0.\text{codh} \parallel \text{Dec.cod}$
 $\text{RDecs}_0.\text{n}=\max(\text{Dec.n,RDecs}_1.\text{n})$
- RDecs $\rightarrow \lambda$**
 $\text{RDecs.cod}=\text{RDecs.codh}$
 $\text{RDecs.etq}=\text{RDecs.etqh}$
 $\text{Rdec.n}=\text{RDecs.nh}$

- **Bloque \rightarrow { RBloque**
Bloque.n=RBloque.n
Bloque.cod=RBloque.cod
Bloque.etq=RBloque.etq
RBloque.etqh=Bloque.etqh
Bloque.inicio=RBloque.inicio
- **RBloque \rightarrow Decs & Is }**
Decs.etqh=RBloque.etqh
Is.etqh=Decs.etq + longPrologo
RBloque.inicio=Decs.etq
RBloque.etq=Is.etq+longEpilogo+1
RBloque.cod=Decs.cod || prologo(RBloque.nh,Decs.dir) || Is.cod ||
epilogo(RBloque.nh) || ir_ind
RBloque.n=Decs.n
- **RBloque \rightarrow & Is }**
Is.etqh=RBloque.etqh + longPrologo
RBloque.inicio=RBloque.etqh
RBloque.etq=Is.etq+longEpilogo+1
RBloque.cod=prologo(RBloque.nh,RBloque.dirh) || Is.cod ||
epilogo(RBloque.nh) || ir_ind
RBloque.n=RBloque.nh
- **Is \rightarrow I RIs**
I.etqh=Is.etqh
RIs.etqh=I.etq
Is.etq=RIs.etq
RIs.codh=I.cod
Is.cod=RIs.cod
- **RIs \rightarrow ; I RIs**
I.etqh=RIs₀.etqh
RIs₁.etqh=I.etq
RIs₀.etq=RIs₁.etq
RIs₁.codh=RIs₀.codh || I.cod
RIs₀.cod=RIs₁.cod
- **RIs \rightarrow λ**
RIs.etq=RIs.etqh
RIs.cod=RIs.codh
- **Exp0 \rightarrow Exp1 RExp0**
Exp0.cod=RExp0.cod
RExp0.codh=parchea(Exp1.cod,Exp1.listav,Exp1.listaf,Exp1.etq,Exp1.etq)
Exp1.etqh=Exp0.etqh
RExp0.etqh=Exp1.etq

```

Exp0.etq=RExp0.etq
Exp0.listav=RExp0.listav
Exp0.listaf=RExp0.listaf
RExp0.listavh=Exp1.listav
RExp0.listafh=Exp1.listaf
Exp1.parh=

```

- **RExp0 → OP0 Exp1**

```

RExp0.cod=RExp00.codh ||
  parchea(Exp1.cod,Exp1.listav,Exp1.listaf,Exp1.etq,Exp1.etq) || OP0.cod
Exp1.etqh=RExp0.etqh
RExp0.etq=Exp1.etq +1
RExp0.listav=[]
RExp0.listaf=[]
Exp1.parh=false

```
- **RExp0 → λ**

```

RExp0.etq=RExp0.etqh
RExp0.cod=RExp0.codh
RExp0.listav=RExp0.listavh
RExp0.listafh=RExp0.listafh

```
- **Exp1 → Exp2 RExp1**

```

Exp1.cod=RExp1.cod
Exp2.etqh=Exp1.etqh
RExp1.codh=Exp2.cod
Exp2.parh=
RExp1.etqh=Exp2.etq
Exp1.etq=RExp1.etq
RExp1.listavh=Exp2.listav
RExp1.listafh=Exp2.listaf
Exp1.listaf=RExp1.listaf
Exp1.listav=RExp1.listav

```
- **RExp1 → OP1 Exp2 RExp1**

```

RExp10.cod=RExp11.cod
Exp2.etqh=RExp10.etqh
RExp1.codh=parchea(RExp10.codh,RExp10.listavh,RExp10.listafh,RExp10.etqh,RExp10.
  etqh) ||parchea(Exp2.cod,Exp2.listav,Exp2.listaf,Exp2.etq,Exp2.etq) || Op1.cod
Exp2.parh=false
RExp11.etqh=Exp2.etq+1
RExp10.etq=RExp11.etq
RExp10.listaf=[]
RExp10.listav=[]

```

- **RExp1 → or Exp2 RExp1**

RExp1₀.cod=RExp1₁.cod

Exp2.etqh=RExp1₀.etqh+3

RExp1.codh=parchea(RExp1₀.codh,[],RExp1₀.listafh,?,RExp1₀.etqh+2) || copia ||
ir-v(?) ||desapila ||parchea(Exp2.cod,[],Exp2.listaf,?,Exp2.etq))

Exp2.parh=false

RExp1₁.etqh=Exp2.etq

RExp1₀.etq=RExp1₁.etq

RExp1₀.listaf=[]

RExp1₀.listav=RExp1₀.listavh || Exp2.listav || [RExp1₀.etqh + 1]

- **RExp1 → λ**

RExp1.cod=RExp1.codh

RExp1.etq=RExp1.etqh

RExp1.listaf=RExp1.listafh

RExp1.listav=RExp1.listavh

- **Exp2 → Exp3 RExp2**

Exp2.cod=RExp2.cod

Exp3.etqh=Exp2.etqh

RExp2.codh=Exp3.cod

Exp3.parh=

RExp2.etqh=Exp3.etq

Exp2.etq=RExp2.etq

RExp2.listavh=Exp3.listav

RExp2.listafh=Exp3.listaf

Exp2.listaf=RExp2.listaf

Exp2.listav=RExp2.listav

- **RExp2 → OP2 Exp3 RExp2**

RExp2₀.cod=RExp2₁.cod

Exp3.etqh=RExp2₀.etqh

RExp2₁.codh=parchea(RExp2₀.codh,RExp2₀.listavh,RExp2₀.listafh,RExp2₀.etqh,RExp2₀.
etqh) ||parchea(Exp3.cod,Exp3.listav,Exp3.listaf,Exp3.etq,Exp3.etq) || Op2.cod

Exp3.parh=false

RExp2₁.etqh=Exp3.etq+1

RExp2₀.etq=RExp2₁.etq

RExp2₀.listaf=[]

RExp2₀.listav=[]

- **RExp2 → and Exp3 RExp2**

RExp2₀.cod=RExp2₁.cod

Exp3.etqh=RExp2₀.etqh+3

RExp2₁.codh=parchea(RExp2₀.codh,RExp2₀.listavh,[],RExp2₀.etqh+2,?) || copia ||
ir-f(?) ||desapila ||parchea(Exp3.cod,Exp3.listav,[],Exp3.etq,?))

```

Exp3.parh=false
RExp21.etqh=Exp3.etq
RExp20.etq=RExp21.etq
RExp20.listaf=RExp20.listafh || Exp3.listaf || [RExp20.etqh+1]
RExp20.listav=[]

```

- **RExp2 → λ**

```

RExp2.cod=RExp2.codh
RExp2.etq=RExp2.etqh
RExp2.listaf=RExp2.listafh
RExp2.listav=RExp2.listavh

```
- **Exp3 → Exp4 RExp3**

```

Exp3.cod=RExp3.cod
RExp3.codh=parchea(Exp4.cod,Exp4.listav,Exp4.listaf,Exp4.etq,Exp4.etq)
Exp4.etqh=Exp3.etqh
RExp3.etqh=Exp4.etq
Exp3.etq=RExp3.etq
Exp3.listav=RExp3.listav
Exp3.listaf=RExp3.listaf
RExp3.listavh=Exp4.listav
RExp3.listafh=Exp4.listaf
Exp4.parh=

```
- **RExp3 → OP3 Exp4**

```

RExp3.cod=RExp30.codh ||
  parchea(Exp4.cod,Exp4.listav,Exp4.listaf,Exp4.etq,Exp4.etq) || OP3.cod
Exp4.etqh=RExp3.etqh
RExp3.etq=Exp4.etq + 1
RExp3.listav=[]
RExp3.listaf=[]
Exp4.parh=false

```
- **RExp3 → λ**

```

RExp3.etq=RExp3.etqh
RExp3.cod=RExp3.codh
RExp3.listav=RExp3.listavh
RExp3.listaf=RExp3.listafh

```
- **Desig → Id RDesig**

```

Desig.cod=RDesig.cod
RDesig.codh=accessoVar(Desig.tsh[id.lex])
RDesig.etqh=Desig.etqh+longAccesoVar(Desig.tsh[id.lex])
Desig.etq=RDesig.etq

```

- **RDesig $\rightarrow \rightarrow$ RDesig**
RDesig₀.cod=RDesig₁.cod
RDesig₀.etq=RDesig₁.etq
RDesig₁.codh=RDesig₀.codh || apila_ind
RDesig₁.etqh=RDesig₀.etqh + 1
- **RDesig \rightarrow [Exp0] RDesig**
RDesig₀.cod=RDesig₁.cod
RDesig₀.etq=RDesig₁.etq
RDesig₁.codh=RDesig₀.codh || Exp0.cod || apila(RDesig₀.tipo.tbases.tam) ||
multiplica || suma
Exp0.etqh=RDesig₀.etqh
RDesig₁.etqh=Exp0.etq + 3
Exp0.parh=false
- **RDesig \rightarrow .Id RDesig**
RDesig₀.cod=RDesig₁.cod
RDesig₀.etq=RDesig₁.etq
RDesig₁.codh=RDesig₀.codh || apila(RDesig₀.tipo.campos[id.lex].desp) ||
suma
RDesig₁.etqh=RDesig₀.etqh + 2
- **RDesig $\rightarrow \lambda$**
RDesig.cod=RDesig.codh
RDesig.etq=RDesig.etqh
- **LArgs \rightarrow Exp0 RLArgs**
LArgs.cod=RLArgs.cod
LArgs.etq=RLArgs.etq
Exp0.etqh=LArgs.etqh + 1
RLArgs.etqh=Exp0.etqh + 1 +
longPasoParametro(Exp0.modos,LArgs.paramsh[1])
RLArgs.codh=copia || Exp0.cod || flip ||
pasoParametro(Exp0.modos,LArgs.paramsh[1])
Exp0.parh= LArgs.paramsh[1].modos=var
- **RLArgs \rightarrow , Exp0 RLArgs**
RLArgs₀.cod=RLArgs₁.cod
RLArgs₀.etq=RLArgs₁.etq
Exp0.etqh=RLArgs₀.etqh + 1
RLArgs₁.etqh=Exp0.etq + 1 +
longPasoParametro(Exp0.modos,RLArgs₀.paramsh[RLArgs₀.nparamsh])
RLArgs₁.codh=RLArgs₀.codh || copia || Exp0.cod || flip ||
pasoParametro(Exp0.modos,RLArgs₀.paramsh[RLArgs₀.nparamsh])
Exp0.parh= RLArgs₀.paramsh[RLArgs₀.nparamsh].modos=var

- $RLArgs \rightarrow \lambda$
 $RLArgs.cod = RLArgs.codh$
 $RLArgs.etq = RLArgs.etqh$

9. Esquema de traducción orientado al traductor

9.1. Variables globales

Variables globales usadas:

1. Cod: código generado hasta el momento (con el fin de no hacer copias del mismo).
2. Ts: tabla de símbolos (para no tener que hacer copias locales de la Ts en su propagación).
3. Etq: contador de instrucciones (no tiene sentido hacerla local).
4. Pend: conjunto de tipos pendientes por declarar.
5. PendProc: conjunto de procedimientos pendientes por declarar.
6. Err: indica si hay errores en las restricciones contextuales.

9.2. Nuevas operaciones y transformación de ecuaciones semánticas

Nuevas operaciones:

- emite(cod): concatena al código global el código *cod*.
- reconoce(token): reconocerá el token en la entrada.
- parchea(etq,valor): parchea la dirección *etq* del código global con *valor*

9.3. Esquema de traducción

- **Prog()**→
{ var n, tmpetq, dir
ts=creaTs(),
emite(inicio(n, dir)),
etq=longInicio,
emite(ir_a(tmpetq)) ,
etq=etq+1 }
Decs(0,0,n, dir)
{ tmpetq=etq
parchea(1,n+1)
parchea(3,n+1+dir)
parchea(5,tmpetq) }
reconoce(&)
Is()
{ err=err v pend<>{} v procpend<>{} }
- **Prog ()**→
{ ts=creaTs(),
emite(inicio(1,0)),
etq=longInicio }
reconoce(&)
Is()
- **Decs** (in dirh, in nh ,out n, out dir)→
{ var n1, n2, props, id, dir1, tam, dir, fwd }
Dec(dirh, nh, n1, props, id, tam, fwd)
{ ponId(ts, Dec.id, Dec.props)
err=err v (existeId(ts, id)^ts[id].nivel=nh^props.clase<>proc)
pend= pend - (Si props.clase=tipo entonces {id} sino {})
procpend=procpend - (si fwd=false ^ props.clase=proc entonces {id} sino {})
}
RDecs(dirh+tam, nh, n2, dir)
{ n=max(n1, n2) }
- **RDecs**(in dirh, in nh, out n, out dir)→
{ var n1, n2, props, id, tam, dir, fwd }
reconoce(;
Dec(dirh, nh, n1, props, id, tam, fwd)
{ ponId(ts, Dec.id, Dec.props)
err=err v (existeId(ts, id)^ts[id].nivel=nh^props.clase<>proc) v
(existeId(ts, id)^ ts[id].nivel=nh ^ (¬pendientes(id, procpend) v fwd))
pend= pend - (Si props.clase=tipo entonces {id} sino {})

```

procpend=procpnd - (si fwd=false ^ props.clase=proc entonces {id} else {})
}
RDecs(dirh+tam,nh,n2,dir)
{n=max(n1,n2)}
• RDecs (in dirh, in nh,out n,out dir)→
{var n,dir}
λ
{n=nh,dir=dirh}
• Dec (in dirh,in nh,out n,out props,out id,out tam,out fwd)→
{var n,props,id,tam,fwd}
DecVar(id,tipo)
{props=<clase:var, tipo:tipo, nivel:nh,dir:dirh>
tam=tipo.tam
n=nh
fwd=false
}
• Dec (in dirh,in nh,out n,out props,out id,out tam,out fwd)→
{var n,props,id,tam,fwd}
DecTipo(id,tipo)
{props=<clase:tipo, tipo:tipo, nivel:nh,dir:->
tam=0
n=nh
fwd=false
}
• Dec (in dirh,in nh,out n,out props,out id,out tam,out fwd)→
{var n,decprops,id,tam,props,ini,fwd}
DecProc(dirh,nh,id,n,decprops,ini,fwd)
{props=<clase:decprops.clase, tipo:decprops.tipo, nivel:nh,dir:-,inicio:ini>
tam=0
}
• DecVar(out id,out tipo) →
{var id,tipo}
Id(id)
reconoce(:)
Tipo(tipo)
{err=err v existeId(ts,id) v referenciaErronea(tipo,ts)}
• DecTipo(out id,out tipo) →
{var id,tipo}
reconoce(tipo)
Id(id)
reconoce(=)

```



```

Tipo(tipo)
{err=err v existeId(ts,id) v  referenciaErronea(tipo,ts)
}

• Tipo (out tipo)→
{var tipo}
reconoce(boolean)
{tipo=<t:boolean,tam:1>}

• Tipo (out tipo)→
{var tipo}
reconoce(character)
{tipo=<t:character,tam:1>}

• Tipo(out tipo) →
{var tipo}
reconoce(natural)
{tipo=<t:natural,tam:1>}

• Tipo (out tipo)→
{var tipo}
reconoce(integer)
{tipo=<t:integer,tam:1>}

• Tipo(out tipo) →
{var tipo}
reconoce(float)
{tipo=<t:float,tam:1>}

• Tipo (out tipo)→
{var tipo,id}
Id(id)
{
err= err v si existeId(ts,id) entonces ts[id].clase<>tipo si no false
tipo=<t:ref, id:id, tam:ts[id].tipo.tam>
pend=pend U si ¬existeId(ts,id) entonces {id} si no {}
}

• Tipo(out tipo) →
{var tipo, tipo1}
reconoce(array)
reconoce()
NNat (lex)
reconoce()
reconoce(of)
Tipo(tipo1)
{tipo=<t:array, nElems: lex, tBase: tipo1, tam: lex * tipo1.tam>
err=err v referenciaErronea(tipo1,ts)}

```

- **Tipo**(out tipo) →
{var tipo,tipo1}
reconoce(pointer)
Tipo(tipo1)
{tipo=<t: punt, tBase: tipo1, tam: 1>}
- **Tipo**(out tipo) →
{var tipo,campos,tam}
reconoce(record)
reconoce()
Campos(campos,tam)
reconoce()
{tipo=<t:reg, campos: campos, tam: tam>}
- **Campos** (out campos,out tam)→
{var campos,tam,campo,tam1,id}
Campo(0,tam1,campo,id)
RCampos(tam1,[campo],tam,campos)
- **RCampos**(in desph,in camposh,out tam,out campos) →
{var tam,campos,tam1,campo,camposh1}
reconoce();
Campo(desph,tam1,campo,err1,id)
{err=err v esDuplicado(camposh,id)
camposh1=camposh || [campo]}
RCampos(desph+tam1,camposh1,tam,campos)
- **RCampos**(in desph,in camposh,out tam,out campos) →
{var tam,campos}
λ
{tam=desph,campos=camposh}
- **Campo**(in desph,out tam,out campo,out id) →
{var tam,campo,id,tipo}
Tipo(tipo)
Id(id)
{campo=<id:id, tipo: tipo, desp: desph>
tam=tipo.tam
err= err v referenciaErronea(tipo,ts)}
- **DecProc** (in dirh,in nh,out id,out n,out decprops,out ini,out fwd)→
{var id,n,decprops,ini,fwd,init}
reconoce(procedure)
Id (id)
reconoce()
{ creaTs(ts)}
Params(dirh,nh+1,params,dir)

reconoce()

{decprops=<clase:proc, tipo:

<t:proc,params:params>,nivel:nh+1,dir:-,inicio:init>

ponId(ts,id,decprops)}

Bloque(dir,nh+1,n,init,fwd)

{err=err v (existeId(ts,id) ^ ts[id].nivel=nh+1) v

procpnd=procpnd U si Bloque.fwd=true entonces {id.lex} si no {}

ini=<inicio:init>

ts[id].props.inicio=init

}

- **Params**(in dirh,in nh,out params,out dir) →

{var params,dir}

LParams(nh,params,dir)

- **Params** (in dirh,in nh,out params,out dir) →

{var params,dir}

λ

{params=[],dir=0}

- **LParams**(in nh,out params,out dir) →

{var params,dir,paramsh,param,tam,id,props}

Param(0,nh,param,tam,id,props)

{paramsh=[param]}

ponId(ts,id,props)}

RLParams(tam,nh,paramsh,dir,params)

- **RLParams**(in dirh,in nh,in paramsh,out dir,out params) →

{var dir,params,param,tam,id,props,paramsh1}

reconoce(,)

Param(dirh,nh,param,tam,id,props,err1)

{paramsh1=paramsh || [param]}

ponId(ts,id,props)

err=err v (existeId(ts,id)^ts[id].nivel=nh)}

RLParams(dirh+tam,nh,paramsh1,dir,params)

- **RLParams**(in dirh,in nh,in paramsh,out dir,out params) →

{var dir,params}

λ

{dir=dirh,params=paramsh}

- **Param**(in dirh in nh,out param,out tam,out id,out props) →

{var param,tam,id,props,clase,modo,tipo}

Modo(clase,modo)

Tipo(tipo)

Id (id)

{props=<clase:clase,tipo:tipo,nivel:nh,dir:dirh>,

param=<modo:modo,tipo:tipo>,
tam= si modo= variable entonces 1 si no tipo.tam}

- **Modo** (out clase,out modo)→
{var clase, modo}
reconoce(var)
{clase=pvar,modo=variable}
- **Modo**(out clase,out modo) →
{ var clase, modo}
 λ
{ clase=var,modo=valor}
- **Bloque**(in dirh,in nh,out n,out init,out fwd) →
{var n,init,fwd}
reconoce(forward)
{ fwd=true
n=nh
init=-}
- **Bloque** (in dirh,in nh,out n,out init,out fwd)→
{ var n,init,fwd}
reconoce({}
RBloque(dirh,nh,n,init)
{ fwd=false}
- **RBloque** (in dirh,in nh,out n,out inicio)→
{ var n,inicio,dir}
Decs(dirh,nh,dir,n)
reconoce(&)
{ inicio=etq
etq=etq+longPrologo
emite(prologo(nh,dir))}
Is ()
reconoce({}
{ etq=etq+longEpilogo+1
emite(epilogo(nh))
emite(ir_ind)}
- **RBloque**(in dirh,in nh,out n,out inicio) →
{ var n,inicio}
reconoce(&)
{ inicio=etq
etq=etq+longPrologo
emite(prologo(nh,dirh))}
Is()
reconoce({}

```

    {etq=etq+longEpilogo+1
    emite(epilogo(nh))
    emite(ir_ind)
    n=nh}
• Is() →
  I()
  RIs()
• RIs() →
  reconoce( ;)
  I()
  RIs()
• RIs() →  $\lambda$ 
• I() → IASig()
• I() → ILec()
• I() → IEsc()
• I() → IIf()
• I() → IWhile()
• I() → IFor()
• I() → ICall()
• I() → INew()
• I() → IDelete()
• IASig() →
  {var tipo,tipo1,listav,listaf,modo}
  Desig(tipo)
  reconoce(:=)
  Exp0(false,tipo1,listav,listaf,modo)
  {err=err v ¬compatibles(tipo,tipo1,ts)}
• Exp0(in parh,out tipo,out listav,out listaf,out modo) →
  {var tipo,tipo1,listav,listaf,listav1,listaf1,modo,modo1}
  Exp1(?,tipo1,listav1,listaf1,modo1)
  {parchea(listav1,listaf1,etq,etq)}
  RExp0(modo1,tipo1,listav1,listaf1,listav,listaf,modo,tipo)
• RExp0(in modoh,in tipoh,in listavh,in listafh,out listav,out listaf,out modo,out
  tipo) →
  {var listav,listaf,op,tipo,listav1,listaf1,modo,tipo1,modo1}
  OP0(op)
  Exp1(false,tipo1,listav1,listaf1,modo1)
  {parchea(listav1,listaf1,etq,etq)}
  emite(op)
  etq=etq+1

```

```

listav=[]
listaf=[]
tipo=tipoOp0(tipoh,tipo1)
modo=valor}

```

- RExp0** (in modoh,in tipoh,in listavh,in listafh,out listav,out listaf,out modo,out tipo)

→

{ var listav,listaf,modo,tipo}

λ

{ listav=listavh,listaf=listafh,tipo=tipoh,modo=modoh}
- Exp1** (in parh,out tipo,out listav,out listaf,out modo)→

{ var tipo,listav,listaf,modo,listav1,listaf1,modo1,tipo1}

Exp2 (?,tipo1,listav1,listaf1,modo1)

RExp1(modo1,tipo1,listav1,listaf1,listav,listaf,modo,tipo)
- RExp1** (in modoh,in tipoh,in listavh,in listafh,out listav,out listaf,out modo,out tipo)

→

{ var listav,listaf,op,tipo,listav1,listaf1,modo,lv,lf,modo1,tipo1,tipoh1,mod}

OP1 (op)

{parchea(listavh,listafh,etq,etq)}

Exp2 (false,tipo1,listav1,listaf1,modo1)

{ parchea(listav1,listaf1,etq,etq)

emite(op)

etq=etq+1

tipoh1=tipoOp1(tipoh,tipo1)}

RExp1(val,tipoh1,listav1,listaf1,lv,lf,mod,tipo)

{ listaf=[]

listav=[]

modo=valor}
- RExp1** (in modoh,in tipoh,in listavh,in listafh,out listav,out listaf,out modo,out tipo)

→

{ var listav,listaf,tipo,listav1,listaf1,modo,lv,lf,tmpetq,modo1,tipo1,tipoh1,mod}

reconoce(or)

{ tmpetq=etq

parchea([],listafh,?,etq+2)

emite(copia)

emite(ir_v(?))

emite(desapila)

etq=etq+3}

Exp2 (false,tipo1,listav1,listaf1,modo1)

{ parchea([],listaf1,?,etq)

tipoh1=tipoOr(tipoh,tipo1)}

RExp1(val,tipoh1,listav1,listaf1,lv,lf,mod,tipo)

```

{ listaf=[]
listav=listavh || listav1 || [tmpetq+1]
modo=valor}

```

- **RExp1** (in modoh,in tipoh,in listavh,in listafh,out listav,out listaf,out modo,out tipo)→
{ var listav,listaf,modo,tipo}
λ
{ listaf=listafh,listav=listavh,modo=modoh,tipo=tipoh}
- **Exp2** (in parh,out tipo,out listav,out listaf,out modo) →
{ var tipo,listav,listaf,modo,listav1,listaf1,tipo1,modo1}
Exp3 (?.tipo1,listav1,listaf1,modo1)
RExp2(modo1,tipo1,listav1,listaf1,listav,listaf,modo,tipo)
- **RExp2** (in modoh,in tipoh,in listavh,in listafh,out listav,out listaf,out modo,out tipo)→
{ var listav,listaf,tipo1,listav1,listaf1,modo1,lv,lf,modo,tipo,tipoh1,mod}
OP2 (op)
{ parchea(listavh,listafh,etq,etq)}
Exp3 (false,tipo1,listav1,listaf1,modo1)
{ parchea(listav1,listaf1,etq,etq)
emite(op)
etq=etq+1
tipoh1=tipoOp2(tipoh,tipo1)}
RExp2(val,tipoh1,listav1,listaf1,lv,lf,mod,tipo)
{ listaf=[],listav=[],modo=valor}
- **RExp2** (in modoh,in tipoh,in listavh,in listafh,out listav,out listaf,out modo,out tipo)→
{ var listav,listaf,tipo1,listav1,listaf1,modo1,lv,lf,tmpetq,tipo,modo,tipoh1}
reconoce(and)
{ tmpetq=etq
parchea(listavh,[],etq+2,?)
emite(copia)
emite(ir_f(?))
emite(desapila)
etq=etq+3}
Exp3 (false,tipo1,listav1,listaf1,modo1)
{ tipoh1=tipoAnd(tipoh,tipo1)
parchea(listav1,[],etq,?)}
RExp2(val,tipoh1,listav1,listaf1,lv,lf,mod,tipo)
{ listav=[],
listaf=listafh || listaf1 || [tmpetq+1]
modo=valor}

- **RExp2** (in modoh,in tipoh,in listavh,in listafh,out listav,out listaf,out modo,out tipo)→
{ var listav,listaf,modo,tipo}
λ
{ listav=listavh,listaf=listafh,modo=modoh,tipo=tipoh}
- **Exp3** (in parh,out tipo,out listav,out listaf,out modo)→
{ var tipo,listav,listaf,modo,tipo1,listav1,listaf1,modo1}
Exp4 (?.tipo1,listav1,listaf1,modo1)
{ parchea(listav1,listaf1,etq,etq)}
RExp3(modo1,tipo1,listav1,listaf1,listav,listaf,modo,tipo)
- **RExp3** (in modoh,in tipoh,in listavh,in listafh,out listav,out listaf,out modo,out tipo)→
{ var op,tipo1,listav1,listaf1,modo1,listav,listaf,modo,tipo}
OP3 (op)
Exp4(false,tipo1,listav1,listaf1,modo1)
{ parchea(listav1,listaf1,etq,etq)
emite(op)
etq=etq+1
listav=[]
listaf=[]
modo=valor
tipo=tipoOp3(tipoh,tipo1)}
• **RExp3**(in modoh,in tipoh,in listavh,in listafh,out listav,out listaf,out modo,out tipo)
→
{ var listav,listaf}
λ
{ listav=listavh,listaf=listafh,modo=modoh,tipo=tipoh}
- **Exp4** (in parh,out tipo,out listav,out listaf,out modo)→
{ var listav,listaf,tipo,modo,listav1,listaf1,tipo1,modo1,op}
OP4A(op)
Exp4(false,tipo1,listav1,listaf1,modo1)
{ tipo=tipoOP4A(op,tipo1)
modo=valor
parchea(listav1,listaf1,etq,etq)
emite(op)
etq=etq+1
listav=[]
listaf=[]}
- **Exp4** (in parh,out tipo,out listav,out listaf,out modo)→
{ var listav,listaf,tipo,modo,listav1,listaf1,tipo1,modo1,op}
OP4NA (op)


```

Exp5(false,tipo1,listav1,listaf1,modo1)
{tipo=tipoOp4NA(op,tipo1)
modo=valor
parchea(listav1,listaf1,etq,etq)
emite(op)
etq=etq+1
listav=[]
listaf=[]}

```

- **Exp4** (in parh,out tipo,out listav,out listaf,out modo)→
{ var listav,listaf,tipo,modo}
Exp5(parh,tipo,listav,listaf,modo)
- **Exp5** (in parh,out tipo,out listav,out listaf,out modo)→
{ var listav,listaf,tipo,modo}
reconoce()
Exp0 (parh,tipo,listav,listaf,modo)
reconoce())
- **Exp5** (in parh,out tipo,out listav,out listaf,out modo) →
{ var lex,tipo,listav,listaf,modo}
NNat (lex)
{tipo=<t:natural>
modo=valor
listav=[]
listaf=[]
emite(apila(lex))
etq=etq+1 }
- **Exp5**(in parh,out tipo,out listav,out listaf,out modo) →
{ var lex,tipo,listav,listaf,modo}
NFloat (lex)
{tipo=<t:float>
modo=valor
listav=[]
listaf=[]
emite(apila(lex))
etq=etq+1 }
- **Exp5** (in parh,out tipo,out listav,out listaf,out modo)→
{ var lex,lex1,tipo,listav,listaf,modo}
Signo(lex)
NNat (lex1)
{tipo=<t:integer>
modo=valor
listav=[]

```

listaf=[]
emite(apila(lex1))
si lex= - entonces emite(inv)
etq=etq + 1
si lex=- entonces etq=etq +1}

```

- **Exp5** (in parh,out tipo,out listav,out listaf,out modo)→
{var tipo,listav,listaf,modo}
reconoce(true)
{tipo=<t:boolean>
modo=valor
listav=[]
listaf=[]
emite(apila(true))
etq=etq+1}
- **Exp5** (in parh,out tipo,out listav,out listaf,out modo)→
{var tipo,listav,listaf,modo}
reconoce(false)
{tipo=<t:boolean>
modo=valor
listav=[]
listaf=[]
emite(apila(false))
etq=etq+1}
- **Exp5** (in parh,out tipo,out listav,out listaf,out modo)→
{var tipo,listav,listaf,modo}
reconoce(null)
{tipo=<t:punt,tbase,null,tam:0>
modo=valor
listav=[]
listaf=[]
emite(apila(null))
etq=etq+1}
- **Exp5** (in parh,out tipo,out listav,out listaf,out modo)→
{var lex,tipo,listav,listaf,modo}
char (lex)
{tipo=<t:character>
modo=valor
listav=[]
listaf=[]
emite(apila(lex))
etq=etq+1}

- **Exp5** (in parh,out tipo,out listav,out listaf,out modo)→
 {var tipo,listav,listaf,modo}
Desig(tipo)
 {modo=variable
 si compatibleTipoBasico(tipo,ts) $\wedge \neg$ parh entonces emite(apila_ind)
 si compatibleTipoBasico(tipo,ts) $\wedge \neg$ parh entonces etq=etq+1
 listav=[]
 listaf=[]}
- **Exp5** (in parh,out tipo,out listav,out listaf,out modo)→
 {var tipo,modo,listav,listaf}
reconoce(|)
Exp0(false,tipo1,listav1,listaf1,modo)
reconoce(|)
 {tipo=valorAbs(tipo1)
 modo=valor
 emite(abs)
 etq=etq+1
 listav=[]
 listaf=[]}
- **Desig (out tipo)→**
 {var tipo,tipo1,id}
Id (id)
 {tipoh=si existeId(ts,id) entonces
 si ts[id].clase=var entonces
 sigueRef(ts[id].tipo,ts)
 si no <t:error>
 si no <t:error>
 emite(accesoVar(ts[id]))
 etq=etq+longAccesoVar(ts[id])}
RDesig(tipoh,tipo)
- **RDesig** (in tipoh,out tipo)→
 {var tipo,tipo1}
reconoce(→)
 {tipoh1=si tipoh.t = punt entonces
 sigueRef(tipoh.tbases,ts)
 si no <t:error>
 emite(apila_ind)
 etq=etq+1}
RDesig(tipoh1,tipo)
- **RDesig** (in tipoh,out tipo)→
 {var tipo,tipo1,tipo1,modo}

```

reconoce()
Exp0(false,tipoh,listav,listaf,modo)
reconoce()
{tipoh1=si tipoh.t = array ^ (tipoh1.t=natural v tipoh1.t=integer) entonces
    sigueRef(tipoh.tbases,ts)
    si no <t:error>
emite(apila(tipoh.tbases.tam))
emite(multiplica)
emite(suma)
etq=etq+3}
RDesig(tipoh1,tipoh)
• RDesig (in tipoh,out tipo) →
{var tipo,tipoh1,id}
reconoce(.)
Id (id)
{tipoh1=si tipoh.t = reg entonces
    si esDuplicado(tipoh.campos,id) entonces
        sigueRef(tipoh.campos[id].tipo,ts)
    si no <t:error>
        si no <t:error>
emite(apila(tipoh.campos[id].desp))
emite(suma)
etq=etq+2}
RDesig(tipoh1,tipoh)
• RDesig (in tipoh,out tipo)→
{var tipo}
λ
{tipo=tipoh}
• ILec ()→
reconoce(in)
reconoce( )
Id (id)
reconoce( )
{err=err v i ¬existeId( ts, id ) entonces true si no false
emite(read)
emite(apila_dir(ts[id].dir)
etq=etq+2}
• IEsc() →
{var tipo,listav,listaf,modo}
reconoce(out)
reconoce( )

```

```

Exp0(false,tipo,listav,listaf,modo)
reconoce( )
{emite(write)
etq=etq+1
err=err v si tipo=err entonces true si no false}

• IIf() →
{var tipo,etqi,tmpetq}
reconoce(if)
Exp0(false,tipo,listav,listaf)
reconoce(then)
{parchea(listav,listaf,etq,etq)
emite(ir_f(etqi))
etq=etq+1
tmpetq=etq}
BloqueI()
PElse(etqi)
{parchea(tmpetq,etqi)
err= err v tipo.t<>boolean}

• PElse(out etqi) →
{var etqi,tmpetq}
reconoce(else)
{emite(ir_a(tmpetq))
etq=etq+1
etqi=etq}
BloqueI()
{tmpetq=etq
parchea(etqi,tmpetq)}

• PElse(out etqi) →
{var etqi}
λ
{etqi=etq}

• IWhile() →
{var tipo,listav,listaf,tmpetq,principio,modo,parchetq}
reconoce(while)
{principio=etq}
Exp0(false,tipo,listav,listaf,modo)
{parchea(listav,listaf,etq,etq)}
reconoce(do)
{emite(ir_f(tmpetq))
etq=etq+1
parchetq=etq}

```

```

BloqueI ()
{emite(ir_a(principio))
etq=etq+1
tmpetq=etq
parchea(parchetq,tmpetq)
err = err v tipo.t<>boolean}

```

- IFor** () →

```

{var tipo1,tipo2,listav1,listav2,listaf1,listaf2,tmpetq,principio,modo1,modo2,id,
parchetq}
reconoce(for)
Id (id)
reconoce(=)
Exp0 (false,tipo1,listav1,listaf1,modo1)
{principio=etq
parchea(listav1,listaf1,etq,etq)
emite(copia)
emite(desapila_dir(ts[id].dir))}
reconoce(to)
{etq=etq+2}
Exp0 (false,tipo2,listav2,listaf2,modo2)
{parchea(listav2,listaf2,etq,etq)
emite(menorigual)
emite(ir_f(tmpetq))}
reconoce(do)
{etq=etq+2
parchetq=etq}
BloqueI ()
{emite(apila_dir(ts[id.lex].dir))
emite(apila(1))
emite(suma)
emite(ir_a(principio))
err= err v ( ¬(tipo1.t=natural ^ tipo2.t=natural) ^ ¬(tipo1.t=integer ^
tipo2.t=integer) ^ ¬compatibles(ts[id.lex].tipo,tipo1,ts))v ¬existeId( ts, id.lex )

etq=etq+4
tmpetq=etq
parchea(parchetq,tmpetq)}

```
- BloqueI** () →

```

reconoce({}
Is ()
reconoce({}

```

- **BloqueI ()** →
I ()
- **ICall ()** →
{var paramsh,id,tmpetq,parchetq}
Id (id)
reconoce ()
{paramsh=ts[id.lex].tipo.params
parchetq=etq
etq=etq+longApilaDirRetorno
emite(apilaDirRetorno(tmpetq))}
Args(paramsh)
reconoce ()
{err = err v \neg existeId(ts,id) v ts[id].clase \Diamond proc
etq=etq+1
emite(ir_a(ts[id].inicio))
tmpetq=etq
parchea(parchetq+4,tmpetq)}
- **Args (in paramsh)**→
{var nparams}
{emite(inicio_paso_param)
etq=etq+longInicioPasoParam}
LArgs(paramsh,nparams)
{emite(fin_paso_param)
err= err v |paramsh| \Diamond nparams
etq=etq+longFinPasoParam}
- **Args (in paramsh)**→
 λ
{err=err v |paramsh|>0}
- **LArgs(in paramsh,out nparams)** →
{var nparams}
{etq=etq+1
emite(copia)}
Exp0((paramsh[1].modo=var),tipo,listav,listaf,modo)
{etq=etq+1+longPasoParametro(modo,paramsh[1])
emite(flip)
emite(pasoParametro(modo,paramsh[1]))}
RLArgs(1,paramsh,nparams)
{err= err v |paramsh|=0 v (paramsh[1].modo=var \wedge modo=val) v
 \neg compatibles(paramsh[1].tipo,tipo,ts)}
- **RLArgs (in nparamsh,in paramsh,out nparams)**→
{var nparams,listav,listaf,tipo,modo}

```

reconoce(,)
{etq=etq+1
emite(copia)}
Exp0((paramsh[nparamsh].modo=var),tipo,listav.listaf,modo)
{err=err v (nparamsh>|paramsh|) v (paramsh[nparamsh].modo=var ^
modo=val) v ¬compatibles(paramsh[nparamsh].tipo,tipo,ts)
etq=etq+1+longPasoParametro(modo,paramsh[nparamsh])
emite(flip)
emite(pasoParametro(modo,paramsh[nparamsh]))}
RLArgs(1+nparamsh,paramsh,nparams)
• RLArgs (in nparamsh,in paramsh,out nparams)→
{var nparams}
λ
{nparams=nparamsh}
• INew () →
{var tipo}
reconoce(new)
Desig(tipo)
{emite(new(si tipo.t = ref entonces ts[tipo.id].tam sino tipo.tbase.tam))
emite(desapila_ind)
etq=etq+2
err=err v tipo.t<▷punt}
• IDelete () →
{var tipo}
reconoce(dispose)
Desig(tipo)
{err = err v tipo.t<▷punt
emite(dispose(si tipo.t = ref entonces ts[tipo.id].tam si no tipo.tbase.tam))
etq=etq+1}
• OP0 (out op)→
{var op}
reconoce(<)
{op=<}
• OP0 (out op)→
{var op}
reconoce(>)
{op=>}
• OP0 (out op)→
{var op}
reconoce(<=)
{op=<=}

```


- **OP0** (out op)→
{ var op}
reconoce(>=)
{ op=>=}
- **OP0** (out op)→
{ var op}
reconoce(==)
{ op= ==}
- **OP0** (out op)→
{ var op}
reconoce(=)
{emite(igual)
op= =}
- **OP1** (out op)→
{ var op}
reconoce(+)
{ op= +}
- **OP1** (out op)→
{ var op}
reconoce(-)
{ op= -}
- **OP2** (out op)→
{ var op}
reconoce(/)
{ op= /}
- **OP2** (out op)→
{ var op}
reconoce(*)
{ op= *}
- **OP2** (out op)→
{ var op}
reconoce(%)
{ op= %}
- **OP3** (out op)→
{ var op}
reconoce(>>)
{ op= >>}
- **OP3** (out op)→
{ var op}
reconoce(<<)

- { op= <<}
- **OP4A** (out op)→
 { var op}
reconoce(not)
 { op=not}
- **OP4A** (out op)→
 { var op}
reconoce(-)
 { op= -}
- **OP4NA** (out op)→
 { var op}
reconoce(float)
 { op= cfloat}
- **OP4NA** (out op)→
 { var op}
reconoce(nat)
 { op= cnat}
- **OP4NA** (out op)→
 { var op}
reconoce(int)
 { op= cint}
- **OP4NA** (out op)→
 { var op}
reconoce(char)
 { op= cchar}

10 Formato de representación del código P

El archivo de código máquina contiene un ArrayList de instrucciones, dichas instrucciones con de una clase implementada llamada Instruction. Se decidió por esta estructura de Java ya que es serializable (permite guardarse directamente en un archivo) y se puede recorrer de una manera eficiente durante la ejecución.

La clase Instruction consta de una función a aplicar y un vector de argumentos para dicha función.

A su vez la función es de un tipo enumerado Funtion que implementa el patrón de diseño Strategy obligando a cada tipo del enumerado a implementar un método exec, esto permite añadir nuevas funciones a la máquina virtual de una manera rápida.

11 Notas sobre la implementación

11.1. Descripción de archivos

Default package

MainCompiler: Realiza la llamada del programa traductor.

MainVM: Realiza la llamada del intérprete.

Common

































Function: Enumerado de funciones de la máquina virtual.

Instruction: Instrucciones bytecode.

Compiler

Campo: campos de los registros

Clases

- ▾  (default package)
 -  MainCompiler.java
 -  MainVM.java
- ▾  common
 -  Function.java
 -  Instruction.java
- ▾  Parser
 -  Campo.java
 -  Clases.java
 - Codigo.java
 -  Data.java
 -  Elems.java
 -  Modo.java
 -  Modos.java
 -  Nodo.java
 -  Oper.java
 -  Param.java
 -  Parser.java
 -  Props.java
 -  SymbolTable.java
 -  Table.java
 -  Type.java
 -  Types.java
 -  TypeTables.java
- ▾  Scanner
 -  CatLexica.java
 -  Estado.java
 -  Reservadas.java
 -  Scanner.java
- ▾  vm
 -  Memory.java
 -  VirtualMachine.java

Codigo: Código que va traduciendo el parser

Data: información de un identificador en la tabla de símbolos

Elems: Elementos de un tipo

Modo: Modo de un parámetro

Nodo

Oper: Operadores

Param: Parámetros

Parser: Analizador sintáctico

Props: Propiedades de una declaración

SymbolTable: tabla de símbolos para el traductor.

Tabla: Tabla de símbolos de un nivel

Type: Tipo

TypeTables: tablas comprobación tipos.

Parser

Parser: Analizador sintáctico

Type: Tipos de datos del lenguaje.

TypeTable: Tablas de comprobación de tipos.

Oper: Tipos de operadores.

Scanner

Estado: estados del analizador léxico

Scanner: analizador léxico

CatLexica: Categorías léxicas

Reservadas: Palabras reservadas

Vm

VirtualMachine: máquina pila virtual

Memory: Memoria de la máquina pila.

11.2. Otras notas

Patrones de diseño utilizados: Singleton, Status, Strategy.