Fase 4: Procesador final de Tiny

Grupo 6

Javier García Viana

Ibon Malles Altolaguirre

David Peromingo Peromingo

Francisco Prieto Gallego

1. Vinculación

Constructoras ASTs	Vinculación
	var ts
prog: Bloq ➔ Prog	**vincula**(**prog**(Bloq)): ts = **creaTS**() **vincula**(Bloq)
bloq: Decs ✖ Instrs ➔ Bloq	**vincula**(**bloq**(Decs, Instrs)): **abreAmbito**(ts) **vincula**(Decs) **vincula**(Instrs) **cierraAmbito**(ts)
siDecs: LDecs ➔ Decs	**vincula**(**siDecs**(LDecs)): **vincula**(LDecs) **vincula2**(LDecs)
noDecs: ➔ Decs	**vincula**(**noDecs**()): **noop**
muchasDecs: LDecs ✖ Dec ➔ Ldecs	**vincula**(**muchasDecs**(LDecs,Dec)): **vincula**(LDecs) **vincula**(Dec) **vincula2**(**muchasDecs**(LDecs,Dec)): **vincula2**(LDecs) **vincula2**(Dec)
unaDec: Dec ➔ LDecs	**vincula**(**unaDec**(LDecs)): **vincula**(Dec) **vincula2**(**unaDec**(LDecs)): **vincula2**(Dec)

decProc: **string** ✖ ParamForms ✖ Bloq ➔ Dec	**vincula**(**decProg**(id, ParamForms, Bloq)):
	abreAmbito(ts)
	vincula(ParamForms)
	vincula(Bloq) // Dentro se hacen 2 pasadas
	cierraAmbito(ts)
	if contiene(ts,id) **then**
	error
	else
	inserta(ts,id,$)
	end if
	
	vincula2(**decProg**(id, ParamForms, Bloq)):
	vincula2(ParamForms)

decType: Tipo ✖ **string** ➔ Dec	**vincula**(**decType**(Tipo,id)): **vincula**(Tipo) **if contiene**(ts,id) **then** **error**

	else
	  inserta(ts,id,$)
	end if
	
	vincula2(**decType**(Tipo,id)):
	  vincula2(Tipo)

decVar: Tipo ✖ **string** ➔ Dec	**vincula**(**decVar**(Tipo,id)):
	  vincula(Tipo)
	  if contiene(ts,id) **then**
	  error
	  else
	  inserta(ts,id,$)
	  end if
	
	vincula2(**decVar**(Tipo,id)):
	  vincula2(Tipo)

siParam: LParams ➔ ParamForms	**vincula**(**siParam**(LParams)):
	  vincula(LParams)
	
	vincula2(**siParam**(LParams)):
	  vincula2(LParams)

noParam: ➔ ParamForms	**vincula**(**noParam**()): **noop** **vincula2**(**noParam**()): **noop**

muchosParams: LParams ✖ ParamForm➔ LParams	**vincula**(**muchosParams**(LParams, ParamForm)):
	  vincula(LParams)
	  vincula(ParamForm)
	
	vincula2(**muchosParams**(LParams, ParamForm)):
	  vincula2(LParams)
	  vincula2(ParamForm)

unParam: ParamForm ➔ LParams	**vincula**(**unParam**(ParamForm)):
	  vincula(ParamForm)
	
	vincula2(**unParam**(ParamForm)):
	  vincula2(ParamForm)

paramFormRef: Tipo ✖ **string** ➔ ParamForm	**vincula**(**paramFormRef**(Tipo, id)):
	  vincula(Tipo)
	  if contiene(ts,id) **then**
	  error
	  else
	  inserta(ts,id,$)
	  end if
	
	vincula2(**paramFormRef**(Tipo, id)):
	  vincula2(Tipo)

paramForm: Tipo ✖ **string** ➔ ParamForm	**vincula**(**paramForm**(Tipo, id)):

	vincula(Tipo)
	if contiene(ts,id) **then**
	error
	else
	inserta(ts,id,$)
	end if
	
	vincula2(**paramForm**(Tipo, id)):
	vincula2(Tipo)

tArray: Tipo ✖ **string** ➔ Tipo	**vincula**(**tArray**(Tipo,ent)):
	if Tipo != **tIden**(_) **then**
	vincula(Tipo)
	end if
	
	vincula2(**tArray**(Tipo,_)):
	if Tipo == **tIden**(id) **then**
	$.vinculo = **vinculoDe**(ts,id)
	else
	vincula2(Tipo)
	end if

tPunt: Tipo ➔ Tipo	**vincula**(**tPunt**(Tipo)):
	if Tipo != **tIden**(_) **then**
	vincula(Tipo)
	end if
	
	vincula2(**tPunt**(Tipo)):
	if Tipo = **tIden**(id) **then**
	$.vinculo = **vinculoDe**(ts,id)
	else
	vincula2(Tipo)
	end if

tInt: ➔ Tipo	**vincula**(**tInt**()): **noop** **vincula2**(**tInt**()): **noop**
tReal: ➔ Tipo	**vincula**(**tReal**()): **noop** **vincula2**(**tReal**()): **noop**
tBool: ➔ Tipo	**vincula**(**tBool**()): **noop** **vincula2**(**tBool**()): **noop**
tString: ➔ Tipo	**vincula**(**tString**()): **noop** **vincula2**(**tString**()): **noop**
tIden: **string** ➔ Tipo	**vincula**(**tIden**(id)): $.vinculo = **vinculoDe**(ts,id) **vincula2**(**tIden**(id)): **noop**
tStruct: LCampos ➔ Tipo	**vincula**(**tStruct**(LCampos)): **vincula**(LCampos) **vincula2**(**tStruct**(LCampos)):

	vincula2(LCampos)
muchosCamps: LCampos ✖ Campo ➔ LCampos	**vincula**(**muchosCamps**(LCampos,Campo)): **vincula**(LCampos) **vincula**(Campo) **vincula2**(**muchosCamps**(LCampos,Campo)): **vincula2**(LCampos) **vincula2**(Campo)
unCamp: Campo ➔ LCampos	**vincula**(**unCamp**(Campo)): **vincula**(Campo) **vincula2**(**unCamp**(Campo)): **vincula2**(Campo)

campo: Tipo ✖ **string** ➔ Campo	**vincula**(**campo**(Tipo, id)):
	if Tipo != **tIden**(_) **then**
	vincula(Tipo)
	end if
	
	vincula2(**campo**(Tipo, _)):
	if Tipo = **tIden**(id) **then**
	$.vinculo = **vinculoDe**(ts,id)
	else
	vincula2(Tipo)
	end if

siInstrs: LInstrs ➔ Instrs	**vincula**(**siInstrs**(LInstrs)): **vincula**(LInstrs)
noInstrs: ➔ Instrs	**vincula**(**noInstrs**()): **noop**
muchasInstrs: LInstrs ✖ Instr ➔ Linstrs	**vincula**(**muchasInstrs**(LInstrs,Instr)): **vincula**(LInstrs) **vincula**(Instr)
unaInstr: Instr ➔ Linstrs	**vincula**(**unaInstr**(Instr)): **vincula**(Instr)
arrobaInstr : Exp ➔ Instr	**vincula**(**arrobaInstr**(Exp)): **vincula**(Exp)
procInstr : **string** ✖ ParamReales ➔ Instr	**vincula**(**procInstr**(id, ParamReales)): $.vínculo = **vinculoDe**(ts,id) **vincula**(ParamReales)
nlInstr : ➔ Instr	**vincula**(**nlInstr**()): **noop**
newInstr : Exp ➔ Instr	**vincula**(**newInstr**(Exp)): **vincula**(Exp)
readInstr : Exp ➔ Instr	**vincula**(**readInstr**(Exp)): **vincula**(Exp)
writeInstr : Exp ➔ Instr	**vincula**(**writeInstr**(Exp)):

	vincula(Exp)
deleteInstr : Exp ➔ Instr	**vincula**(**deleteInstr**(Exp)): **vincula**(Exp)
whileInstr : Exp ✖ Bloq ➔ Instr	**vincula**(**whileInstr**(Exp, Bloq)): **vincula**(Exp) **vincula**(Bloq)
ifElseInstr : Exp ✖ Bloq ✖ Bloq ➔ Instr	**vincula**(**ifElseInstr**(Exp, Bloq1, Bloq2)): **vincula**(Exp) **vincula**(Bloq1) **vincula**(Bloq2)
ifInstr : Exp ✖ Bloq ➔ Instr	**vincula**(**ifInstr**(Exp, Bloq)): **vincula**(Exp) **vincula**(Bloq)
bloqueInstr: Bloq ➔ Instr	**vincula**(**bloqueInstr**(Bloq)): **vincula**(Bloq)
siExp: LExps ➔ ParamReales	**vincula**(**siExp**(LExps)): **vincula**(LExps)
noExp: ➔ ParamReales	**vincula**(**noExp**()): **noop**
muchasExp: LExps ✖ Exp ➔ LExps	**vincula**(**muchasExp**(LExps,Exp)): **vincula**(LExps) **vincula**(Exp)
unaExp: Exp ➔ LExps	**vincula**(**unaExp**(Exp)): **vincula**(Exp)
asignación: Exp ✖ Exp ➔ Exp	**vincula**(**asignacion**(Exp1,Exp2)): **vincula**(Exp1) **vincula**(Exp2)
suma: Exp ✖ Exp ➔ Exp	**vincula**(**suma**(Exp1, Exp2)): **vincula**(Exp1) **vincula**(Exp2)
and: Exp ✖ Exp ➔ Exp	**vincula**(**and**(Exp1, Exp2)): **vincula**(Exp1) **vincula**(Exp2)
or: Exp ✖ Exp ➔ Exp	**vincula**(**or**(Exp1, Exp2)): **vincula**(Exp1) **vincula**(Exp2)
resta: Exp ✖ Exp ➔ Exp	**vincula**(**resta**(Exp1, Exp2)): **vincula**(Exp1) **vincula**(Exp2)
menor: Exp ✖ Exp ➔ Exp	**vincula**(**menor**(Exp1, Exp2)): **vincula**(Exp1)

	vincula(Exp2)
mayor: Exp ✖ Exp ➔ Exp	**vincula**(**mayor**(Exp1, Exp2)): **vincula**(Exp1) **vincula**(Exp2)
menorIgual: Exp ✖ Exp ➔ Exp	**vincula**(**menorIgual**(Exp1, Exp2)): **vincula**(Exp1) **vincula**(Exp2)
mayorIgual: Exp ✖ Exp ➔ Exp	**vincula**(**mayorIgual**(Exp1, Exp2)): **vincula**(Exp1) **vincula**(Exp2)
igual: Exp ✖ Exp ➔ Exp	**vincula**(**igual**(Exp1, Exp2)): **vincula**(Exp1) **vincula**(Exp2)
desigual: Exp ✖ Exp ➔ Exp	**vincula**(**desigual**(Exp1, Exp2)): **vincula**(Exp1) **vincula**(Exp2)
mul: Exp ✖ Exp ➔ Exp	**vincula**(**mul**(Exp1,Exp2)): **vincula**(Exp1) **vincula**(Exp2)
div: Exp ✖ Exp ➔ Exp	**vincula**(**div**(Exp1,Exp2)): **vincula**(Exp1) **vincula**(Exp2)
mod: Exp ✖ Exp ➔ Exp	**vincula**(**mod**(Exp1,Exp2)): **vincula**(Exp1) **vincula**(Exp2)
neg: Exp ➔ Exp	**vincula**(**neg**(Exp)): **vincula**(Exp)
array: Exp ✖ Exp ➔ Exp	**vincula**(**array**(Exp1, Exp2)): **vincula**(Exp1) **vincula**(Exp2)
expCampo: Exp ✖ **string** ➔ Exp	**vincula**(**expCampo**(Exp,id)): **vincula**(Exp)
punt: Exp ➔ Exp	**vincula**(**punt**(Exp)): **vincula**(Exp)
not: Exp ➔ Exp	**vincula**(**not**(Exp)): **vincula**(Exp)
litEnt: **string** ➔ Exp	**vincula**(**litEnt**(_)): **noop**
litReal: **string** ➔ Exp	**vincula**(**litReal**(_)): **noop**
iden: **string** ➔ Exp	**vincula**(**iden**(id)):

	if !contiene(ts, id) **then**
	  error
	end if
	$.vínculo = **vinculoDe**(ts, id)

true: ➔ Exp	**vincula**(**true**()): **noop**
false: ➔ Exp	**vincula**(**false**()): **noop**
litCad: **string** ➔ Exp	**vincula**(**litCad**(_)): **noop**
null: ➔ Exp	**vincula**(**null**()): **noop**

2. Comprobación de tipos

2.1. Pretipado

Constructoras ASTs	Vinculación
	var pilaConjCampos = **pilaVacia**()
prog: Bloq ➔ Prog	**pretipado**(**prog**(Bloq)): **pretipado**(Bloq)
bloq: Decs ✖ Instrs ➔ Bloq	**pretipado**(**bloq**(Decs, _)): **pretipado**(Decs)
siDecs: LDecs ➔ Decs	**pretipado**(**siDecs**(LDecs)): **pretipado**(LDecs)
noDecs: ➔ Decs	**pretipado**(**noDecs**()): **noop**

muchasDecs: LDecs ✖ Dec ➔ Ldecs	**pretipado**(**muchasDecs**(LDecs, Dec)):
	  pretipado(LDecs)
	  pretipado(Dec)

unaDec: Dec ➔ LDecs	**pretipado**(**unaDec**(Dec)): **pretipado**(Dec)

decProc: **string** ✖ ParamForms ✖ Bloq ➔ Dec	**pretipado**(**decProc**(id, Params, Bloq)):
	  pretipado(Params)
	  pretipado(Bloq)

decType: Tipo ✖ **string** ➔ Dec	**pretipado**(**decType**(Tipo, _)): **pretipado**(Tipo)
decVar: Tipo ✖ **string** ➔ Dec	**pretipado**(**decVar**(Tipo, _)): **pretipado**(Tipo)

siParam: LParams ➔ ParamForms	**pretipado**(**siParam**(LParams)): **pretipado**(LParams)
noParam: ➔ ParamForms	**pretipado**(**noParams**()): **noop**

muchosParams: LParams ✖ ParamForm➔ LParams	**pretipado**(**muchosParams**(LParams, Param)):
	  pretipado(LParams)
	  pretipado(Param)

unParam: ParamForm ➔ LParams	**pretipado**(**unParam**(Param)): **pretipado**(Param)
paramFormRef: Tipo ✖ **string** ➔ ParamForm	**pretipado**(**paramFormRef**(Tipo, _)): **pretipado**(Tipo)
paramForm: Tipo ✖ **string** ➔ ParamForm	**pretipado**(**paramForm**(Tipo, _)): **pretipado**(Tipo)

tArray: Tipo ✖ **string** ➔ Tipo	**pretipado**(**tArray**(Tipo, ent)):
	  pretipado(Tipo)
	  if ent[0] == '-' **then**
	  error
	  end if

tPunt: Tipo ➔ Tipo	**pretipado**(**tPunt**(id)): **pretipado**(Tipo)
tInt: ➔ Tipo	**pretipado**(**tInt**()): **noop**
tReal: ➔ Tipo	**pretipado**(**tReal**()): **noop**
tBool: ➔ Tipo	**pretipado**(**tBool**()): **noop**
tString: ➔ Tipo	**pretipado**(**tString**()): **noop**

tIden: **string** ➔ Tipo	**pretipado**(**tIden**(id)):
	  if $.vinculo != **decType**(_,id) **then**
	  error
	  end if

tStruct: LCampos ➔ Tipo	**pretipado**(**tStruct**(LCampos)):
	  apila(pilaConjCampos, { })
	  pretipado(LCampos)
	  desapila(pilaConjCampos)

muchosCamps: LCampos ✖ Campo ➔ LCampos	**pretipado**(**muchosCamps**(LCamps, Camp)):
	  pretipado(LCamps)
	  pretipado(Camp)

unCamp: Campo ➔ LCampos	**pretipado**(**unCamp**(Campo)): **pretipado**(Campo)
campo: Tipo ✖ **string** ➔ Campo	**pretipado**(**campo**(Tipo, id)): **pretipado**(Tipo)

	if contiene(**cima**(conjCampos), id) **then**
	  error
	else
	  inserta(**cima**(conjCampos), id)
	end if

siInstrs: LInstrs ➔ Instrs	-
noInstrs: ➔ Instrs	-
muchasInstrs: LInstrs ✖ Instr ➔ Linstrs	-
unaInstr: Instr ➔ Linstrs	-
arrobaInstr : Exp ➔ Instr	-
procInstr : **string** ✖ ParamReales ➔ Instr	-
nlInstr : ➔ Instr	-
newInstr : Exp ➔ Instr	-
readInstr : Exp ➔ Instr	-
writeInstr : Exp ➔ Instr	-
deleteInstr : Exp ➔ Instr	-
whileInstr : Exp ✖ Bloq ➔ Instr	-
ifElseInstr : Exp ✖ Bloq ✖ Bloq ➔ Instr	-
ifInstr : Exp ✖ Bloq ➔ Instr	-
bloqueInstr: Bloq ➔ Instr	-
siExp: LExps ➔ ParamReales	-
noExp: ➔ ParamReales	-
muchasExp: LExps ✖ Exp ➔ LExps	-
unaExp: Exp ➔ LExps	-
asignación: Exp ✖ Exp ➔ Exp	-
suma: Exp ✖ Exp ➔ Exp	-
and: Exp ✖ Exp ➔ Exp	-
or: Exp ✖ Exp ➔ Exp	-
resta: Exp ✖ Exp ➔ Exp	-
menor: Exp ✖ Exp ➔ Exp	-
mayor: Exp ✖ Exp ➔ Exp	-

2.2. Tipado

Constructoras ASTs	Comprobación de tipos
	var Ø
prog: Bloq ➔ Prog	**tipado**(**prog**(Bloq)): **tipado**(Bloq) $.tipo = Bloq.tipo
bloq: Decs ✖ Instrs ➔ Bloq	**tipado**(**bloq**(Decs,Instrs)): **tipado**(Decs) **tipado**(Instrs) $.tipo = **ambosOk**(Decs.tipo, Instrs.tipo)
siDecs: LDecs ➔ Decs	**tipado**(**siDecs**(LDecs)):

	tipado(LDecs) $.tipo = LDecs.tipo
noDecs: ➔ Decs	**tipado**(**noDecs**()): $.tipo = **ok**
muchasDecs: LDecs ✖ Dec ➔ Ldecs	**tipado**(**muchasDecs**(LDecs, Dec)): **tipado**(LDecs) **tipado**(Dec) $.tipo = **ambosOk**(LDecs.tipo, Dec.tipo)
unaDec: Dec ➔ LDecs	**tipado**(**unaDec**(Dec)): **tipado**(Dec) $.tipo = Dec.tipo
decProc: **string** ✖ ParamForms ✖ Bloq ➔ Dec	**tipado**(**decProc**(id, ParamForms, Bloq)): **tipado**(Bloq) $.tipo = Bloq.tipo
decType: Tipo ✖ **string** ➔ Dec	**tipado**(**decType**(_,_)): $.tipo = **ok**
decVar: Tipo ✖ **string** ➔ Dec	**tipado**(**decVar**(_,_): $.tipo = **ok**
siParam: LParams ➔ ParamForms	-
noParam: ➔ ParamForms	-
muchosParams: LParams ✖ ParamForm➔ LParams	-
unParam: ParamForm ➔ LParams	-
paramFormRef: Tipo ✖ **string** ➔ ParamForm	-
paramForm: Tipo ✖ **string** ➔ ParamForm	-
tArray: Tipo ✖ **string** ➔ Tipo	-
tPunt: Tipo ➔ Tipo	-
tInt: ➔ Tipo	-
tReal: ➔ Tipo	-
tBool: ➔ Tipo	-
tString: ➔ Tipo	-
tIden: **string** ➔ Tipo	-
tStruct: LCampos ➔ Tipo	-
muchosCamps: LCampos ✖ Campo ➔ LCampos	-

unCamp: Campo ➔ LCampos	-
campo: Tipo ✖ **string** ➔ Campo	-

siInstrs: LInstrs ➔ Instrs	**tipado**(**siInstrs**(LInstrs)):
	  tipado(LInstrs)
	  $.tipo = LInstrs.tipo

noInstrs: ➔ Instrs	**tipado**(**noInstrs**()): $.tipo = **ok**

muchasInstrs: LInstrs ✖ Instr ➔ Linstrs	**tipado**(**muchasInstrs**(LInstrs,Instr)):
	  tipado(LInstrs)
	  tipado(Instr)
	  $.tipo = **ambosOk**(LInstrs.tipo, Instr.tipo)

unaInstr: Instr ➔ Linstrs	**tipado**(**unaInstr**(Instr)):
	  tipado(Instr)
	  $.tipo = Instr.tipo

arrobaInstr : Exp ➔ Instr	**tipado**(**arrobaInstr**(Exp)):
	  tipado(Exp)
	  if Exp.tipo == **error then**
	  $.tipo = **error**
	  else
	  $.tipo = **ok**
	  end if

procInstr : **string** ✖ ParamReales ➔ Instr	**tipado**(**procInstr**(id, PReales)):
	  tipado(PReales)
	  if PReales.tipo != ok **then**
	  $.tipo = **error**
	  else
	  let decProc(id, PForms, _) = $.vínculo **in**
	  $.tipo = **tipoParams**(PForms,PReales)
	  end let
	  end if

nlInstr : ➔ Instr	**tipado**(**nlInstr**()): $.tipo = **ok**

newInstr : Exp ➔ Instr	**tipado**(**newInstr**(Exp)):
	  tipado(Exp)
	  if ref!(Exp.tipo) == **tPunt**(_) **then**
	  $.tipo = **ok**
	  else
	  $.tipo = **error**
	  end if

readInstr : Exp ➔ Instr	**tipado**(**readInstr**(Exp)):
	  tipado(Exp)
	  if esDesignador(Exp) and (**ref!(**Exp.tipo) == **tInt**() or **ref!**(Exp.tipo) == **tReal**() or **ref!**(Exp.tipo) == **tString**()) **then**
	  $tipo = **ok**

	else
	$tipo = **error**
	end if

writeInstr : Exp ➔ Instr	**tipado**(**writeInstr**(Exp)):
	tipado(Exp)
	if ref!(Exp.tipo) == **tInt**() or **ref!**(Exp.tipo) == **tReal**() or **ref!**(Exp.tipo) == **tBool**() or **ref!**(Exp.tipo) == **tString**() **then**
	$.tipo = **ok**
	else
	$.tipo = **error**
	end if

deleteInstr : Exp ➔ Instr	**tipado**(**deleteInstr**(Exp)):
	tipado(Exp)
	if ref!(Exp.tipo) == **tPunt**(_) **then**
	$.tipo = **ok**
	else
	$.tipo = **error**
	end if

whileInstr : Exp ✖ Bloq ➔ Instr	**tipado**(**whileInstr**(Exp, Bloq)):
	tipado(Exp);
	tipado(Bloq);
	if Bloq.tipo == **ok** and **ref!**(Exp.tipo) == **tBool**() **then**
	$.tipo = **ok**
	else
	$.tipo = **error**
	end if

ifElseInstr : Exp ✖ Bloq ✖ Bloq ➔ Instr	**tipado**(**ifElseInstr**(Exp, Bloq1, Bloq2)):
	tipado(Exp);
	tipado(Bloq1);
	tipado(Bloq2);
	if Bloq1.tipo == **ok** and **ref!**(Exp.tipo) == **tBool()** and Bloq2.tipo == **ok** **then**
	$.tipo = **ok**
	else
	$.tipo = **error**
	end if

ifInstr : Exp ✖ Bloq ➔ Instr	**tipado**(**ifInstr**(Exp, Bloq)):
	tipado(Exp);
	tipado(Bloq);
	if Bloq.tipo == **ok** and **ref!**(Exp.tipo) == **tBool() then**
	$.tipo = **ok**
	else
	$.tipo = **error**
	end if

bloqueInstr: Bloq ➔ Instr	**tipado**(**bloqueInstr**(Bloq)):

	tipado(Bloq) $.tipo = Bloq.tipo
siExp: LExps ➔ ParamReales	**tipado**(**siExp**(LExps)): **tipado**(LExps) $.tipo = LExps.tipo
noExp: ➔ ParamReales	**tipado**(**noExp**()): $.tipo = **ok**

muchasExp: LExps ✖ Exp ➔ LExps	**tipado**(**muchasExps**(LExps, Exp)):
	tipado(LExps)
	tipado(Exp)
	if Exp.tipo != **error then**
	$.tipo = LExps.tipo
	else
	$.tipo = **error**
	end if

unaExp: Exp ➔ LExps	**tipado**(**unaExp**(Exp)):
	tipado(Exp)
	if Exp.tipo != **error then**
	$.tipo = **ok**
	else
	$.tipo = **error**
	end if

asignación: Exp ✖ Exp ➔ Exp	**tipado**(**asignacion**(E0,E1)):
	tipado(E0)
	tipado(E1)
	if (**esDesignador**(E0) and **compatibles**(E0.tipo, E1.tipo)) **then**
	$.tipo = E0.tipo
	else
	$.tipo = **error**
	end if

suma: Exp ✖ Exp ➔ Exp	**tipado**(**suma**(Exp1, Exp2)): $.tipo = **tipadoBinArit**(Exp1, Exp2)
and: Exp ✖ Exp ➔ Exp	**tipado**(**and**(Exp1, Exp2)): $.tipo = **tipadoBinLog**(Exp1, Exp2)
or: Exp ✖ Exp ➔ Exp	**tipado**(**or**(Exp1, Exp2)): $.tipo = **tipadoBinLog**(Exp1, Exp2)
resta: Exp ✖ Exp ➔ Exp	**tipado**(**resta**(Exp1, Exp2)): $.tipo = **tipadoBinArit**(Exp1, Exp2)
menor: Exp ✖ Exp ➔ Exp	**tipado**(**menor**(Exp1, Exp2)): $.tipo = **tipadoBinRel**(Exp1, Exp2)
mayor: Exp ✖ Exp ➔ Exp	**tipado**(**mayor**(Exp1, Exp2)): $.tipo = **tipadoBinRel**(Exp1, Exp2)

menorIgual: Exp ✖ Exp ➔ Exp	**tipado**(**menorIgual**(Exp1, Exp2)): $.tipo = **tipadoBinRel**(Exp1, Exp2)
mayorIgual: Exp ✖ Exp ➔ Exp	**tipado**(**mayorIgual**(Exp1, Exp2)): $.tipo = **tipadoBinRel**(Exp1, Exp2)
igual: Exp ✖ Exp ➔ Exp	**tipado**(**igual**(Exp1, Exp2)): $.tipo = **tipadoBinComp**(Exp1, Exp2)
desigual: Exp ✖ Exp ➔ Exp	**tipado**(**desigual**(Exp1, Exp2)): $.tipo = **tipadoBinComp**(Exp1, Exp2)
mul: Exp ✖ Exp ➔ Exp	**tipado**(**mul**(E0,E1)): $.tipo = **tipadoBinArit**(E0,E1)
div: Exp ✖ Exp ➔ Exp	**tipado**(**div**(E0,E1)): $.tipo = **tipadoBinArit**(E0,E1)

mod: Exp ✖ Exp ➔ Exp	**tipado**(**mod**(E0,E1)):
	tipado(E0)
	tipado(E1)
	let T0' = **ref!**(E0.tipo) and T1' = **ref!**(E1.tipo) **in**
	if T0' == T1' **and** T0' == **tInt**() **then**
	$.tipo = **tInt**()
	else
	$.tipo = **error**
	end let

neg: Exp ➔ Exp	**tipado**(**neg**(Exp)):
	tipado(Exp)
	let T' = **ref!**(Exp.tipo) **in**
	if T' == **tInt**() **then**
	$.tipo = **tInt**()
	else if T' == **tReal**() **then**
	$.tipo = **tReal**()
	else
	$.tipo = **error**
	end let

array: Exp ✖ Exp ➔ Exp	**tipado**(**array**(Exp1, Exp2)):
	tipado(Exp1)
	tipado(Exp2)
	let T1' = **ref!**(Exp1.tipo) **in**
	if T1' == **tArray**(T'', _) and **ref!**(Exp2.tipo) == **tInt**() **then**
	$.tipo = T''
	else
	$.tipo = **error**
	end let

expCampo: Exp ✖ **string** ➔ Exp	**tipado**(**expCampo**(Exp, id)): **tipado**(Exp) **if ref!**(Exp.tipo) == **tStruct**(LCampos) **then** $.tipo = **tieneCampo**(LCampos, id)

	else
	$.tipo = **error**
	end if

punt: Exp ➔ Exp	**tipado**(**punt**(Exp)):
	tipado(Exp)
	if ref!(Exp.tipo) == **tPunt**(T') **then**
	$.tipo = T'
	else
	$.tipo = **error**
	end if

not: Exp ➔ Exp	**tipado**(**not**(Exp)):
	tipado(Exp)
	if ref!(Exp.tipo) == **tBool**() **then**
	$.tipo = **tBool**()
	else
	$.tipo = error

litEnt: **string** ➔ Exp	**tipado**(**litEnt**(val)): $.tipo = **tInt**()
litReal: **string** ➔ Exp	**tipado**(**litReal**(val)): $.tipo = **tReal**()

iden: **string** ➔ Exp	**tipado**(**iden**(id)):
	if $.vinculo == **decVar**(T, id) **then**
	$.tipo = T
	else if $.vinculo == **paramForm**(T, id) **then**
	$.tipo = T
	else if $.vinculo == **paramFormRef**(T, id) **then**
	$.tipo = **tInt**()
	else
	$.tipo = **error**
	end if

true: ➔ Exp	**tipado**(**true**()): $.tipo = **tBool**()
false: ➔ Exp	**tipado**(**false**()): $.tipo = **tBool**()
litCad: **string** ➔ Exp	**tipado**(**litCad**(cad)): $.tipo = **tString**()
null: ➔ Exp	**tipado**(**null**()): $.tipo = **null**

ambosOk(T0,T1):
if T0 == ok T1 == ok **then**
return ok
else
return error	

```
   end if
```

```
tipadoBinArit(E0, E1):
   tipado(E0)
   tipado(E1)
   let T0' = ref!(E0.tipo) and  T1' = ref!(E1.tipo) in
      if T0' == T1' and T0' == tInt() then
         return tInt();
      else if (T0' == tReal() or T0'== tInt()) and (T1' == tReal() or T1'== tInt()) then
         return tReal()
      else
         return error
   end let
```

```
tipadoBinLog(E0, E1):
   tipado(E0)
   tipado(E1)
   let T0' = ref!(E0.tipo) and  T1' = ref!(E1.tipo) in
      if T0' == T1' and T0' == tBool() then
         return tBool()
      else
         return error
   end let
```

```
tipadoBinRel(E0, E1):
   tipado(E0)
   tipado(E1)
   let T0' = ref!(E0.tipo) and  T1' = ref!(E1.tipo) in
      if (T0' == tInt() or T0' == tReal()) and (T1' == tInt() or T1' == tReal()) then
         return tBool()
      else if T0' == T1' and T0' == tBool() then
         return tBool()
     else if T0' == T1' and T0' == tString() then
         return tBool()
      else
         return error
   end let
```

```
tipadoBinComp(E0, E1):
   tipado(E0)
   tipado(E1)
   if tipadoBinRel(E0.tipo, E1.tipo) != error then
      return tBool()
   end if
   let T0' = ref!(E0.tipo) and  T1' = ref!(E1.tipo) in
      if (T0' == tPunt() or T0' == null) and (T1' == tPunt() or T1' == null) then
         return tBool()
      else
         return error
   end let
```

```
ref!(T):
   if T==tIden() then
       let T.vinculo = decType(T', T) in
```

```
        return ref!(T')
    else
        return T
```

```
esDesignador(E):
    return E == iden(_) or E == array(_, _) or E == expCampo(_, _) or E == punt(_)
```

```
compatibles(T0, T1):
    Ø = {T0=T1}
    return unificables(T0, T1)
```

```
sonUnificables(T0, T1):
    if !contiene(Ø, T1=T2) then
        inserta(Ø, T1=T2)
        return unificables(T1, T2)
    else
        return true
    end if
```

```
unificables(T0, T1):
    let T0' = ref!(T0) and T1' = ref!(T1) in
        if T0' == T1' and (T0' == tBool() or T0' == tInt() or T0' == tString()) then
            return true;
        else if T0' == tReal() and (T1'== tInt() or T1' == tReal()) then
            return true
        else if T0' == tArray(T'',litEnt(e1)) and T1' == tArray(T''',litEnt(e2)) then
            if e1 == e2 then
                return sonUnificables(T'', T''')
            else
                return false
            end if
        else if T0' == tStruct(LC1) and T1' == tStruct(LC2) then
            return compatibles(LC1, LC2)
        else if (T0' == tPunt(T'') and T1 == null) then
            return true
        else if (T0' == tPunt(T'') and T1' == tPunt(T''')) then
            return sonUnificables(T'', T''')
        else
            return false
        end if
    end let
```

```
compatibles(unCamp(c1), unCamp(c2))
    let campo(T1,_) = c1 and campo(T2,_) = c2 in
        return unificables(T1, T2)
    end let

compatibles(muchosCamps(lc1, c1), muchosCamps(lc2, c1)):
    let campo(T1,_) = c1 and campo(T2,_) = c2 in
        return unificables(T1, T2) and compatibles(lc1, lc2)
    end let
```

```
tipoParams(PForm, PReales):
    if PForm == siParam(Params) and PReales == siExp(Exps) then
        return tipoSiParams(Params, Exps)
```

```
   else if PForm == noParam() and PReales == noExp() then
      return ok
   else
      return error
   end if

tipoSiParams(Params, Exps):
   if Params == unParam(Param) and Exps == unaExp(Exp) then
      return tipoParam(Param, Exp)
   else if Params == muchosParams(LP, P) and Exps == muchasExps(LE, E) then
      let tipo1 = tipoSiParams(LP, LE) and tipo2 = tipoParam(P, E) in
         return ambosOk(tipo1, tipo2)
      end let
   else
      return error
   end if

tipoParam(Param, Exp):
   if Param == paramForm(T, _) and compatibles(T, Exp.tipo) then
      return ok
   else if Param == paramFormRef(T, _) and esDesignador(Exp) and compatibles(T,
Exp.tipo) then
      if not (ref!(T) == tReal() and ref!(Exp.tipo) == tReal()) and (ref!(T) == tReal() or
ref!(Exp.tipo) == tReal()) then
         return error
      else
         return ok
      else if
   else
      return error
   end if
```

```
tieneCampo(muchosCamps(LCampos, Campo), id):
   if Campo == campo(T, id) then
      return T
   else
      return tieneCampo(LCampos, id)
   end if

tieneCampo(unCamp(Campo), id):
   if Campo == campo(T, id) then
      return T
   else
      return error
   end if
```

3. Asignación de espacio

Constructoras ASTs	Asignación de espacio
	var dir **var** nivel **var** maxDir

prog: Bloq ➔ Prog	**asigEspacio**(**prog**(Bloq)): nivel = 0 **asigEspacio**(Bloq) **asigEspacio2**(Bloq)
bloq: Decs ✖ Instrs ➔ Bloq	**asigEspacio**(**bloq**(Decs, Instrs)): **asigEspacio**(Decs) **asigEspacio**(Instrs) **asigEspacio2**(**bloq**(Decs, Instrs)): dir_ant = dir **asigEspacio2**(Decs) **asigEspacio2**(Instrs) dir = dir_ant
siDecs: LDecs ➔ Decs	**asigEspacio**(**siDecs**(LDecs)): **asigEspacio**(LDecs) **asigEspacio**(**siDecs**(LDecs)): **asigEspacio2**(LDecs)
noDecs: ➔ Decs	**asigEspacio**(**noDecs**()): **noop** **asigEspacio2**(**noDecs**()): **noop**
muchasDecs: LDecs ✖ Dec ➔ Ldecs	**asigEspacio**(**muchasDecs**(LDecs,Dec)): **asigEspacio**(LDecs) **asigEspacio**(Dec) **asigEspacio2**(**muchasDecs**(LDecs,Dec)): **asigEspacio2**(LDecs) **asigEspacio2**(Dec)
unaDec: Dec ➔ LDecs	**asigEspacio**(**unaDec**(Dec)): **asigEspacio**(Dec) **asigEspacio2**(**unaDec**(Dec)): **asigEspacio**(Dec)
decProc: **string** ✖ ParamForms ✖ Bloq ➔ Dec	**asigEspacio**(**decProc**(Id, ParamForms, Bloq)): nivel++ $.nivel = nivel **asigEspacio**(ParamForms) **asigEspacio**(Bloq) nivel-- **asigEspacio2**(**decProc**(Id, ParamForms, Bloq)): dir_ant = dir dir = 0 $.dir = dir **asigEspacio2**(ParamForms) **asigEspacio2**(Bloq) $.tam = dir dir = dir_ant

decType: Tipo ✖ **string** ➔ Dec	**asigEspacio**(**decType**(Tipo,Id)):
	  asigTam(Tipo)
	
	asigEspacio2(**decType**(Tipo,Id)):
	  asigTam2(Tipo)

decVar: Tipo ✖ **string** ➔ Dec	**asigEspacio**(**decVar**(Tipo,Id)):
	  asigTam(Tipo)
	  $.nivel = nivel
	
	asigEspacio2(**decVar**(Tipo,Id)):
	  $.dir = dir
	  asigTam2(Tipo)
	  incrDir(Tipo.tam)

siParam: LParams ➔ ParamForms	**asigEspacio**(**siParam**(LParams)):
	  asigEspacio(LParams)
	
	asigEspacio2(**siParam**(LParams)):
	  asigEspacio2(LParams)

noParam: ➔ ParamForms	**asigEspacio**(**noParam**()): **noop** **asigEspacio2**(**noParam**()): **noop**

muchosParams: LParams ✖ ParamForm➔ LParams	**asigEspacio**(**muchosParams**(LParams, ParamForm)):
	  asigEspacio(LParams)
	  asigEspacio(ParamForm)
	
	asigEspacio2(**muchosParams**(LParams, ParamForm)):
	  asigEspacio2(LParams)
	  asigEspacio2(ParamForm)

unParam: ParamForm ➔ LParams	**asigEspacio**(**unParam**(ParamForm)):
	  asigEspacio(ParamForm)
	
	asigEspacio2(**unParam**(ParamForm)):
	  asigEspacio2(ParamForm)

paramFormRef: Tipo ✖ **string** ➔ ParamForm	**asigEspacio**(**paramFormRef**(Tipo, **string**)):
	  asigTam(Tipo)
	  $.nivel = nivel
	
	asigEspacio2(**paramFormRef**(Tipo, **string**)):
	  $.dir = dir
	  asigTam2(Tipo)
	  incrDir(1)

paramForm: Tipo ✖ **string** ➔ ParamForm	**asigEspacio**(**paramForm**(Tipo, **string**)):
	  asigTam(Tipo)
	  $.nivel = nivel
	
	asigEspacio2(**paramForm**(Tipo, **string**)):
	  $.dir = dir

	asigTam2(Tipo) **incrDir**(Tipo.tam)

tArray: Tipo ✖ **string** ➔ Tipo	**asigTam**(**tArray**(Tipo, litEnt)):
	if Tipo != **tIden**(_) **then**
	asigTam(Tipo)
	$.tam = Tipo.tam * **int**(litEnt)
	end if
	
	asigTam2(**tArray**(Tipo, litEnt)):
	if Tipo == **tIden**(_) **then**
	asigTam2(Tipo)
	$.tam = Tipo.tam * **int**(litEnt)
	end if

tPunt: Tipo ➔ Tipo	**asigTam**(**tPunt**(Tipo)):
	if Tipo != **tIden**(_) **then**
	asigTam(Tipo)
	end if
	$.tam = 1
	
	asigTam2(**tPunt**(Tipo)):
	if Tipo == tIden(_) **then**
	asigTam2(Tipo)
	end if

tInt: ➔ Tipo	**asigTam**(**tInt**()): $.tam = 1 **asigTam2**(**tInt**()): **noop**
tReal: ➔ Tipo	**asigTam**(**tReal**()): $.tam = 1 **asigTam2**(**tReal**()): **noop**
tBool: ➔ Tipo	**asigTam**(**tBool**()): $.tam = 1 **asigTam2**(**tBool**()): **noop**
tString: ➔ Tipo	**asigTam**(**tString**()): $.tam = 1 **asigTam2**(**tString**()): **noop**
tIden: **string** ➔ Tipo	**asigTam**(**tIden**(id)): **let decType**(T, id) = $.vinculo **in** **asigTam**(T) $.tam = T.tam **end let** **asigTam2**(**tIden**(_)): **noop**
tStruct: LCampos ➔ Tipo	**asigTam**(**tStruct**(LCampos)):

	asigTam(LCampos) **asigTam2**(**tStruct**(LCampos)): dir_ant = dir dir = 0 **asigTam2**(LCampos) $.tam = dir dir = dir_ant
muchosCamps: LCampos ✖ Campo ➔ LCampos	**asigTam**(**muchosCamps**(LCampos, Campo)): **asigTam**(LCampos) **asigTam**(Campo) **asigTam2**(**muchosCamps**(LCampos, Campo)): **asigTam2**(LCampos) **asigTam2**(Campo) $.tam = LCampos.tam + Campo.tam
unCamp: Campo ➔ LCampos	**asigTam**(**unCamp**(Campo)): **asigTam**(Campo) **asigTam2**(**unCamp**(Campo)): **asigTam2**(Campo) $.tam = Campo.tam

campo: Tipo ✖ **string** ➔ Campo	**asigTam**(**campo**(Tipo, **string**)):
	if Tipo != **tIden**(_) **then**
	asigTam(Tipo)
	$.tam = Tipo.tam
	end if
	
	asigTam2(**campo**(Tipo, **string**)):
	if Tipo == **tIden**(_) **then**
	asigTam2(Tipo)
	$.tam = Tipo.tam
	end if
	$.dir = dir
	dir += $.tam

siInstrs: LInstrs ➔ Instrs	**asigEspacio**(**siInstrs**(LInstrs)): **asigEspacio**(LInstrs) **asigEspacio2**(**siInstrs**(LInstrs)): **asigEspacio2**(LInstrs)
noInstrs: ➔ Instrs	**asigEspacio**(**noInstrs**(LInstrs)): **noop** **asigEspacio2**(**noInstrs**(LInstrs)): **noop**
muchasInstrs: LInstrs ✖ Instr ➔ Linstrs	**asigEspacio**(**muchasInstrs**(LInstrs, Instr)): **asigEspacio**(LInstrs) **asigEspacio**(Instr) **asigEspacio2**(**muchasInstrs**(LInstrs, Instr)): **asigEspacio2**(LInstrs)

	asigEspacio2(Instr)
unaInstr: Instr ➔ Linstrs	**asigEspacio**(**unaInstr**(Instr)): **asigEspacio**(Instr) **asigEspacio2**(**unaInstr**(Instr)): **asigEspacio2**(Instr)
arrobaInstr : Exp ➔ Instr	**asigEspacio**(**arrobaInstr**(Exp)): **noop** **asigEspacio2**(**arrobaInstr**(Exp)): **noop**
procInstr : **string** ✖ ParamReales ➔ Instr	**asigEspacio**(**procInstr**(_, _)): **noop** **asigEspacio2**(**procInstr**(_, _)): **noop**
nlInstr : ➔ Instr	**asigEspacio**(**nlInstr**()): **noop** **asigEspacio2**(**nlInstr**()): **noop**
newInstr : Exp ➔ Instr	**asigEspacio**(**newInstr**(Exp)): **noop** **asigEspacio2**(**newInstr**(Exp)): **noop**
readInstr : Exp ➔ Instr	**asigEspacio**(**readInstr**(Exp)): **noop** **asigEspacio2**(**readInstr**(Exp)): **noop**
writeInstr : Exp ➔ Instr	**asigEspacio**(**writeInstr**(Exp)): **noop** **asigEspacio2**(**writeInstr**(Exp)): **noop**
deleteInstr : Exp ➔ Instr	**asigEspacio**(**deleteInstr**(Exp)): **noop** **asigEspacio2**(**deleteInstr**(Exp)): **noop**
whileInstr : Exp ✖ Bloq ➔ Instr	**asigEspacio**(**whileInstr**(Exp, Bloq)): **asigEspacio**(Bloq) **asigEspacio2**(**whileInstr**(Exp, Bloq)): **asigEspacio2**(Bloq)
ifElseInstr : Exp ✖ Bloq ✖ Bloq ➔ Instr	**asigEspacio**(**ifInstr**(Exp, Bloq1, Bloq2)): **asigEspacio**(Bloq1) **asigEspacio**(Bloq2) **asigEspacio2**(**ifInstr**(Exp, Bloq1, Bloq2)): **asigEspacio2**(Bloq1) **asigEspacio2**(Bloq2)
ifInstr : Exp ✖ Bloq ➔ Instr	**asigEspacio**(**ifInstr**(Exp, Bloq)): **asigEspacio**(Bloq) **asigEspacio2**(**ifInstr**(Exp, Bloq)): **asigEspacio2**(Bloq)
bloqueInstr: Bloq ➔ Instr	**asigEspacio**(**bloqueInstr**(Bloq)): **asigEspacio**(Bloq) **asigEspacio2**(**bloqueInstr**(Bloq)): **asigEspacio2**(Bloq)
siExp: LExps ➔ ParamReales	-

noExp: ➔ ParamReales	-
muchasExp: LExps ✖ Exp ➔ LExps	-
unaExp: Exp ➔ LExps	-
asignación: Exp ✖ Exp ➔ Exp	-
suma: Exp ✖ Exp ➔ Exp	-
and: Exp ✖ Exp ➔ Exp	-
or: Exp ✖ Exp ➔ Exp	-
resta: Exp ✖ Exp ➔ Exp	-
menor: Exp ✖ Exp ➔ Exp	-
mayor: Exp ✖ Exp ➔ Exp	-
menorIgual: Exp ✖ Exp ➔ Exp	-
mayorIgual: Exp ✖ Exp ➔ Exp	-
igual: Exp ✖ Exp ➔ Exp	-
desigual: Exp ✖ Exp ➔ Exp	-
mul: Exp ✖ Exp ➔ Exp	-
div: Exp ✖ Exp ➔ Exp	-
mod: Exp ✖ Exp ➔ Exp	-
neg: Exp ➔ Exp	-
array: Exp ✖ Exp ➔ Exp	-
expCampo: Exp ✖ **string** ➔ Exp	-
punt: Exp ➔ Exp	-
not: Exp ➔ Exp	-
litEnt: **string** ➔ Exp	-
litReal: **string** ➔ Exp	-
iden: **string** ➔ Exp	-
true: ➔ Exp	-
false: ➔ Exp	-
litCad: **string** ➔ Exp	-
null: ➔ Exp	-

	incrDir(n):
	  dir += n
	  maxDir = **max**(MaxDir, dir)

4. Repertorio de instrucciones de código-p

desapila: desapila y elimina el valor en la cima de la pila.

apilaInt(literalEntero): Añade un entero a la cima de la pila.

apilaReal(literalReal): Añade un real a la cima de la pila.

apilaString(string): Añade un string a la cima de la pila.

apilaBool(bool): Añade un booleano a la cima de la pila.

castReal: Desapila un valor entero de la cima de la pila y lo vuelve a apilar como real.

fetch: Desapila una dirección d de la cima de la pila, y apila el valor de la celda d de la memoria de datos.

store: Desapila de la pila un valor v y una dirección d, y almacena v en la celda d de la memoria de datos.

copia(n): Desapila una dirección d0 y una dirección d1, y copia el contenido de n celdas desde d0 a n celdas desde d1.

suma: Desapila los dos últimos valores de la pila y apila el resultado de sumar esos dos valores.

resta: Desapila los dos últimos valores de la pila y apila el resultado de restar esos dos valores.

mul: Desapila los dos últimos valores de la pila y apila el resultado de multiplicar esos dos valores.

div: Desapila los dos últimos valores de la pila y apila el resultado de dividir esos dos valores.

mod: Desapila los dos últimos valores de la pila y apila el módulo resultado de dividir esos dos valores.

and: aplica AND lógico sobre los dos valores en la cima de la pila

or: aplica OR lógico sobre los dos valores en la cima de la pila

mayor: desapila los dos valores en la cima de la pila y comprueba si el primero es mayor que el segundo.

menor: desapila los dos valores en la cima de la pila y comprueba si el primero es menor que el segundo.

mayorIgual: desapila los dos valores en la cima de la pila y comprueba si el primero es mayor o igual que el segundo.

menorIgual: desapila los dos valores en la cima de la pila y comprueba si el primero es menor o igual que el segundo.

igual: desapila los dos valores en la cima de la pila y comprueba si son iguales.

desigual: desapila los dos valores en la cima de la pila y comprueba si son distintos.

not: desapila el valor en la cima de la pila y cambia su valor booleano.

menosUnario: desapila el valor en la cima de la pila y cambia su signo.

alloc(t): reserva espacio en memoria dado un tipo t, y apila la dirección resultante.

dealloc(t): desapila una dirección d, y recibe un tipo t. Se notifica que la zona de memoria que comienza en d y que permite almacenar valores del tipo t queda liberada.

irA(d): salta a la dirección d.

irF(d): desapila un valor y salta a la dirección d si es false. Error si no es booleano.

irD: Desapila una dirección d de la pila, y realiza un salto incondicional a dicha dirección.

activa(n,t,d): Reserva espacio en el segmento de pila de registros de activación para ejecutar un procedimiento que tiene nivel de anidamiento n y tamaño de datos locales t. Así mismo, almacena en la zona de control de dicho registro d como dirección de retorno. También almacena en dicha zona de control el valor del display de nivel n. Por último, apila en la pila de evaluación la dirección de comienzo de los datos en el registro creado.

desactiva(n,t): Libera el espacio ocupado por el registro de activación actual, restaurando adecuadamente el estado de la máquina. n es el nivel de anidamiento del procedimiento asociado y t el tamaño de los datos locales. De esta forma, la instrucción apila en la pila de evaluación la dirección de retorno, restaura el valor del display de nivel n al antiguo valor guardado en el registro y decrementa el puntero de pila de registros de activación en el tamaño ocupado por el registro.

dup: Consulta el valor v de la cima de la pila de evaluación, y apila de nuevo dicho valor (es decir, duplica la cima de la pila de evaluación).

stop: Detiene la máquina.

apilaDisp(n): Apila el valor del display de nivel n

desapilaDisp(n): Desapila una dirección d de la pila en el display de nivel n.

read: Lee un dato de la entrada estándar y lo apila.

write: Desapila un dato y escribe su valor en la salida estándar.

idx(t): desapila una dirección d y un valor i, y recibe un tamaño t. Considera que, a partir de d, comienza un array cuyos elementos son valores de tamaño t, y devuelve la dirección de comienzo del elemento i-esimo de dicho array.

5. Etiquetado

Constructoras ASTs	Etiquetado
	var etq = 0 **var** procPendientes = **pilaVacia**()
prog: Bloq ➔ Prog	**etiquetado**(**prog**(Bloq)): $.prim = etq **etiquetado**(Bloq) etq++ **while** !**esVacia**(procPendientes) p = **desapila**(procPendientes) **let decProc**(id,Param,Bq) = p **in** p.prim = etq etq++ **etiquetado**(Bq) **etiquetadoLiberaParam**(Param) etq += 2 p.sig = etq **end let** **end while** $.sig = etq
bloq: Decs ✖ Instrs ➔ Bloq	**etiquetado**(**bloq**(Decs,Instrs)): $.prim = etq **recolectaProcs**(Decs) **etiquetado**(Instrs) $.sig = etq
siDecs: LDecs ➔ Decs	**recolectaProcs**(**siDecs**(LDecs)): **recolectaProcs**(LDecs)
noDecs: ➔ Decs	**recolectaProcs**(**noDecs**()): **noop**
muchasDecs: LDecs ✖ Dec ➔ Ldecs	**recolectaProcs**(**muchasDecs**(Decs,Dec)): **recolectaProcs**(Decs) **recolectaProcs**(Dec)
unaDec: Dec ➔ LDecs	**recolectaProcs**(**unaDec**(Dec)): **recolectaProcs**(Dec)
decProc: **string** ✖ ParamForms ✖ Bloq ➔ Dec	**recolectaProcs**(**decProc**(_,_,_)): **apila**(procPendientes, $)
decType: Tipo ✖ **string** ➔ Dec	**recolectaProcs**(**decType**(_,_)): **noop**
decVar: Tipo ✖ **string** ➔ Dec	**recolectaProcs**(**decVar**(_,_)): **noop**

siParam: LParams ➔ ParamForms	-
noParam: ➔ ParamForms	-
muchosParams: LParams ✖ ParamForm➔ LParams	-
unParam: ParamForm ➔ LParams	-
paramFormRef: Tipo ✖ **string** ➔ ParamForm	-
paramForm: Tipo ✖ **string** ➔ ParamForm	-
tArray: Tipo ✖ **string** ➔ Tipo	-
tPunt: Tipo ➔ Tipo	-
tInt: ➔ Tipo	-
tReal: ➔ Tipo	-
tBool: ➔ Tipo	-
tString: ➔ Tipo	-
tIden: **string** ➔ Tipo	-
tStruct: LCampos ➔ Tipo	-
muchosCamps: LCampos ✖ Campo ➔ LCampos	-
unCamp: Campo ➔ LCampos	-
campo: Tipo ✖ **string** ➔ Campo	-
siInstrs: LInstrs ➔ Instrs	**etiquetado**(**siInstrs**(LInstrs)): $.prim = etq **etiquetado**(LInstrs) $.sig = etq
noInstrs: ➔ Instrs	**etiquetado**(**noInstrs**()): $.prim = etq $.sig = etq
muchasInstrs: LInstrs ✖ Instr ➔ Linstrs	**etiquetado**(**muchasInstrs**(LInstrs,Instr)): $.prim = etq **etiquetado**(LInstrs) **etiquetado**(Instr) $.sig = etq
unaInstr: Instr ➔ Linstrs	**etiquetado**(**unaInstr**(Instr)): $.prim = etq **etiquetado**(Instr) $.sig = etq

arrobaInstr : Exp ➔ Instr	**etiquetado**(**arrobaInstr**(Exp)): $.prim = etq **etiquetado**(Exp) etq++ $.sig = etq
procInstr : **string** ✖ ParamReales ➔ Instr	**etiquetado**(**procInstr**(id, PReales)): $.prim = etq etq++ **let decProc**(id, PForm) = $.vínculo **in** **etiquetadoPasoParams**(PForm, PReales) **end let** etq++ $.sig = etq
nlInstr : ➔ Instr	**etiquetado**(**nlInstr**()): $.prim = etq etq += 2 $.sig = etq
newInstr : Exp ➔ Instr	**etiquetado**(**newInstr**(Exp)): $.prim = etq **etiquetado**(Exp) etq += 2 $.sig = etq
readInstr : Exp ➔ Instr	**etiquetado**(**readInstr**(Exp)): $.prim = etq **etiquetado**(Exp) etq += 2 $.sig = etq
writeInstr : Exp ➔ Instr	**etiquetado**(**writeInstr**(Exp)): $.prim = etq **etiquetado**(Exp) **etiquetadoAccVal**(Exp) etq++ $.sig = etq
deleteInstr : Exp ➔ Instr	**etiquetado**(**deleteInstr**(Exp)): $.prim = etq **etiquetado**(Exp) etq += 2 $.sig = etq
whileInstr : Exp ✖ Bloq ➔ Instr	**etiquetado**(**whileInstr**(Exp, Bloq)): $.prim = etq **etiquetado**(Exp) **etiquetadoAccVal**(Exp) etq++ **etiquetado**(Bloq) etq++ $.sig = etq

ifElseInstr : Exp ✖ Bloq ✖ Bloq ➔ Instr	**etiquetado**(**ifElseInstr**(Exp,Bloq1, Bloq2)): $.prim = etq **etiquetado**(Exp) **etiquetadoAccVal**(Exp) etq++ **etiquetado**(Bloq1) etq++ **etiquetado**(Bloq2) $.sig = etq
ifInstr : Exp ✖ Bloq ➔ Instr	**etiquetado**(**ifInstr**(Exp,Bloq)): $.prim = etq **etiquetado**(Exp) **etiquetadoAccVal**(Exp) etq++ **etiquetado**(Bloq) $.sig = etq
bloqueInstr: Bloq ➔ Instr	**etiquetado**(**bloqueInstr**(Bloq)): $.prim = etq **etiquetado**(Bloq) $.sig = etq
siExp: LExps ➔ ParamReales	**etiquetado**(**siExp**(LExps)): $.prim = etq **etiquetado**(LExps) $.sig = etq
noExp: ➔ ParamReales	**etiquetado**(**noExp**(LExps)): $.prim = etq $.sig = etq
muchasExp: LExps ✖ Exp ➔ LExps	**etiquetado**(**muchasExp**(LExps, Exp)): $.prim = etq **etiquetado**(LExps) **etiquetado**(Exp) $.sig = etq
unaExp: Exp ➔ LExps	**etiquetado**(**unaExp**(Exp)): $.prim = etq **etiquetado**(Exp) $.sig = etq
asignación: Exp ✖ Exp ➔ Exp	**etiquetado**(**asig**(Exp1,Exp2)): $.prim = etq **etiquetado**(Exp1) etq++ **etiquetado**(Exp2) **if ref!**(Exp1.tipo) == **tReal**() && **ref!**(Exp2.tipo) == **tInt**() **then** **etiquetadoAccVal**(Exp2) etq++ **end if** etq++

	$.sig = etq
suma: Exp ✖ Exp ➔ Exp	**etiquetado**(**suma**(Exp1, Exp2)): $.prim = etq **etiquetadoBinAritm**(Exp1, Exp2) etq++ $.sig = etq
and: Exp ✖ Exp ➔ Exp	**etiquetado**(**and**(Exp1, Exp2)): $.prim = etq **etiquetadoBin**(Exp1, Exp2) etq++ $.sig = etq
or: Exp ✖ Exp ➔ Exp	**etiquetado**(**or**(Exp1, Exp2)): $.prim = etq **etiquetadoBin**(Exp1, Exp2) etq++ $.sig = etq
resta: Exp ✖ Exp ➔ Exp	**etiquetado**(**resta**(Exp1, Exp2)): $.prim = etq **etiquetadoBinAritm**(Exp1, Exp2) etq++ $.sig = etq
menor: Exp ✖ Exp ➔ Exp	**etiquetado**(**menor**(Exp1, Exp2)): $.prim = etq **etiquetadoBin**(Exp1, Exp2) etq++ $.sig = etq
mayor: Exp ✖ Exp ➔ Exp	**etiquetado**(**mayor**(Exp1, Exp2)): $.prim = etq **etiquetadoBin**(Exp1, Exp2) etq++ $.sig = etq
menorIgual: Exp ✖ Exp ➔ Exp	**etiquetado**(**menorIgual**(Exp1, Exp2)): $.prim = etq **etiquetadoBin**(Exp1, Exp2) etq++ $.sig = etq
mayorIgual: Exp ✖ Exp ➔ Exp	**etiquetado**(**mayorIgual**(Exp1, Exp2)): $.prim = etq **etiquetadoBin**(Exp1, Exp2) etq++ $.sig = etq
igual: Exp ✖ Exp ➔ Exp	**etiquetado**(**igual**(Exp1, Exp2)): $.prim = etq **etiquetadoBin**(Exp1, Exp2) etq++ $.sig = etq

desigual: Exp ✖ Exp ➔ Exp	**etiquetado**(**desigual**(Exp1, Exp2)): $.prim = etq **etiquetadoBin**(Exp1, Exp2) etq++ $.sig = etq
mul: Exp ✖ Exp ➔ Exp	**etiquetado**(**mul**(Exp1, Exp2)): $.prim = etq **etiquetadoBinAritm**(Exp1, Exp2) etq++ $.sig = etq
div: Exp ✖ Exp ➔ Exp	**etiquetado**(**div**(Exp1, Exp2)): $.prim = etq **etiquetadoBinAritm**(Exp1, Exp2) etq++ $.sig = etq
mod: Exp ✖ Exp ➔ Exp	**etiquetado**(**mod**(Exp1, Exp2)): $.prim = etq **etiquetadoBin**(Exp1, Exp2) etq++ $.sig = etq
neg: Exp ➔ Exp	**etiquetado**(**neg**(Exp)): $.prim = etq **etiquetado**(Exp) **etiquetadoAccVal**(Exp) etq++ $.sig = etq
array: Exp ✖ Exp ➔ Exp	**etiquetado**(**array**(Exp1, Exp2)): $.prim = etq **etiquetado**(Exp1) **etiquetado**(Exp2) **etiquetadoAccVal**(Exp2) etq++ $.sig = etq
expCampo: Exp ✖ **string** ➔ Exp	**etiquetado**(**expCampo**(Exp, _)): $.prim = etq **etiquetado**(Exp) etq += 2 $.sig = etq
punt: Exp ➔ Exp	**etiquetado**(**punt**(Exp)): $.prim = etq **etiquetado**(Exp) etq++ $.sig = etq
not: Exp ➔ Exp	**etiquetado**(**not**(Exp)): $.prim = etq **etiquetado**(Exp) **etiquetadoAccVal**(Exp)

	etq++ $.sig = etq
litEnt: **string** ➔ Exp	**etiquetado**(**litEnt**(_)): $.prim = etq etq++ $.sig = etq
litReal: **string** ➔ Exp	**etiquetado**(**litReal**(_)): $.prim = etq etq++ $.sig = etq
iden: **string** ➔ Exp	**etiquetado**(**iden**(_)): $.prim = etq **etiquetadoAccId**($.vínculo) $.sig = etq
true: ➔ Exp	**etiquetado**(**true**()): $.prim = etq etq++ $.sig = etq
false: ➔ Exp	**etiquetado**(**false**()): $.prim = etq etq++ $.sig = etq
litCad: **string** ➔ Exp	**etiquetado**(**litCad**(id)): $.prim = etq etq++ $.sig = etq
null: ➔ Exp	**etiquetado**(**null**()): $.prim = etq etq++ $.sig = etq

	etiquetadoAccVal(E):
	if esDesignador(E) **then**
	etq++
	end if

	etiquetadoAccId(**decVar**(T,id)):
	etq++
	if $.nivel > 0 **then**
	etq += 2
	end if
	
	etiquetadoAccId(**paramForm**(T,id)):
	etq += 3
	
	etiquetadoAccId(**paramFormRef**(T,id)):
	etq += 4

```
etiquetadoBin(Exp1, Exp2):
   etiquetado(Exp1)
   etiquetadoAccVal(Exp1)
   etiquetado(Exp2)
   etiquetadoAccVal(Exp2)
```

```
etiquetadoBinAritm(Exp1, Exp2, E):
   etiquetado(Exp1)
   etiquetadoAccVal(Exp1)
   castAritm(Exp1, E)
   etiquetado(Exp2)
   etiquetadoAccVal(Exp2)
   castAritm(Exp2, E)

castAritm(Exp, E):
   if ref!(E.tipo) == tReal() && ref!(Exp.tipo) == tInt() then
      etq++
   end if
```

```
etiquetadoPasoParams(siParam(LParams), siExp(LExps)):
   etiquetadoPasoParams(LParams, LExps)

etiquetadoPasoParams(noParam(), noExp()): noop

etiquetadoPasoParams(unParam(Param), unaExp(Exp)):
   etiquetadoPasoParam(Param, Exp)

etiquetadoPasoParams(muchosParams(LParam, Param), muchasExps(LExps, Exp)):
   etiquetadoPasoParams(LParam, LExps)
   etiquetadoPasoParam(Param, Exp)

etiquetadoPasoParams(_, _): noop

etiquetadoPasoParam(paramFormRef(T, _), Exp):
   etq += 7
   etiquetado(Exp)
   etq++

etiquetadoPasoParam(paramForm(T, _), Exp):
   etq += 7
   etiquetado(Exp)
   if ref!(T) == tReal() && ref!(Exp.tipo) == tInt() then
      etiquetadoAccVal(Exp)
      etq++
   end if
   etq++
```

```
etiquetadoLiberaParams(siParam(LParams)):
   etiquetadoLiberaParam(LParams)

etiquetadoLiberaParams(noParam()): noop

etiquetadoLiberaParams(unParam(Param)):
   etiquetadoLiberaParam(Param)

etiquetadoLiberaParams(muchosParams(LParam, Param)):
```

```
    etiquetadoLiberaParams(LParam)
    etiquetadoLiberaParam(Param)

etiquetadoLiberaParam(Param):
    etq += 5
```

6. Generación de código

Constructoras ASTs	Código
	var procPendientes = **pilaVacia**()

prog: Bloq ➔ Prog	**genCod**(**prog**(Bloq)):
	  genCod(Bloq)
	  emit stop()
	  while !esVacia(procPendientes)
	  p = desapila(procPendientes)
	  let decProc(id,Param,Bloque) = p **in**
	  emit desapilaDisp(p.nivel)
	  genCod(Bloque)
	  genLiberaParams(Param)
	  emit desactiva(p.nivel, p.tam)
	  emit irD()
	  end let
	  end while

bloq: Decs ✖ Instrs ➔ Bloq	**genCod**(**bloq**(Decs, Instrs)):
	  recolectaProcs(Decs)
	  genCod(Instrs)

siDecs: LDecs ➔ Decs	**recolectaProcs**(**siDecs**(LDecs)): **recolectaProcs**(LDecs)
noDecs: ➔ Decs	**recolectaProcs**(**noDecs**()): **noop**

muchasDecs: LDecs ✖ Dec ➔ Ldecs	**recolectaProcs**(**muchasDecs**(Decs,Dec)):
	  recolectaProcs(Decs)
	  recolectaProcs(Dec)

unaDec: Dec ➔ LDecs	**recolectaProcs**(**unaDec**(Dec)): **recolectaProcs**(Dec)
decProc: **string** ✖ ParamForms ✖ Bloq ➔ Dec	**recolectaProcs**(**decProc**(_,_,_)): **apila**(procPendientes, $)
decType: Tipo ✖ **string** ➔ Dec	**recolectaProcs**(**decType**(_,_)): **noop**
decVar: Tipo ✖ **string** ➔ Dec	**recolectaProcs**(**decVar**(_,_)): **noop**
siParam: LParams ➔ ParamForms	-

noParam: ➔ ParamForms	-
muchosParams: LParams ✖ ParamForm➔ LParams	-
unParam: ParamForm ➔ LParams	-
paramFormRef: Tipo ✖ **string** ➔ ParamForm	-
paramForm: Tipo ✖ **string** ➔ ParamForm	-
tArray: Tipo ✖ **string** ➔ Tipo	-
tPunt: Tipo ➔ Tipo	-
tInt: ➔ Tipo	-
tReal: ➔ Tipo	-
tBool: ➔ Tipo	-
tString: ➔ Tipo	-
tIden: **string** ➔ Tipo	-
tStruct: LCampos ➔ Tipo	-
muchosCamps: LCampos ✖ Campo ➔ LCampos	-
unCamp: Campo ➔ LCampos	-
campo: Tipo ✖ **string** ➔ Campo	-
siInstrs: LInstrs ➔ Instrs	**genCod**(**siInstrs**(LInstrs)): **genCod**(LInstrs)
noInstrs: ➔ Instrs	**genCod**(**noInstrs**()): **noop**

muchasInstrs: LInstrs ✖ Instr ➔ Linstrs	**genCod**(**muchasInstrs**(LInstrs, Instr)):
	  genCod(LInstrs)
	  genCod(Instr)

unaInstr: Instr ➔ Linstrs	**genCod**(**unaInstr**(Instr)): **genCod**(Instr)

arrobaInstr : Exp ➔ Instr	**genCod**(**arrobaInstr**(Exp)):
	  genCod(Exp)
	  emit desapila()

procInstr : **string** ✖ ParamReales ➔ Instr	**genCod**(**procInstr**(Id, PReales)):
	  emit activa($.vinculo.nivel,$.vinculo.tam,$.sig)
	  let decProc(id, PForm, Bloq) = $.vínculo **in**
	  genPasoParams(PForm, PReales)
	  emit irA(Bloq.prim)

	end let
nlInstr : ➔ Instr	**genCod**(**nlInstr**(Exp)): **emit** apilaString(**"\n"**) **emit** write()
newInstr : Exp ➔ Instr	**genCod**(**newInstr**(Exp)): **genCod**(Exp) **let ref!**(Exp.tipo) = **tPunt**(Tipo) **in** **emit** alloc(Tipo.tam) **end let** **emit** store()
readInstr : Exp ➔ Instr	**genCod**(**readInstr**(Exp)): **genCod**(Exp) **emit** read() **emit** store()
writeInstr : Exp ➔ Instr	**genCod**(**writeInstr**(Exp)): **genCod**(Exp) **genAccVal**(Exp) **emit** write()
deleteInstr : Exp ➔ Instr	**genCod**(**deleteInstr**(Exp)): **genCod**(Exp) **emit** fetch() **emit** dealloc(Exp.tipo.tam)
whileInstr : Exp ✖ Bloq ➔ Instr	**genCod**(**whileInstr**(Exp,Bloq)): **genCod**(Exp) **genAccVal**(Exp) **emit** irF($.sig) **genCod**(Bloq) **emit** irA($.prim)
ifElseInstr : Exp ✖ Bloq ✖ Bloq ➔ Instr	**genCod**(**ifElseInstr**(Exp,Bloq1,Bloq2)): **genCod**(Exp) **genAccVal**(Exp) **emit** irF(Bloq2.prim) **genCod**(Bloq1) **emit** irA($.sig) **genCod**(Bloq2)
ifInstr : Exp ✖ Bloq ➔ Instr	**genCod**(**ifInstr**(Exp,Bloq)): **genCod**(Exp) **genAccVal**(Exp) **emit** irF($.sig) **genCod**(Bloq)
bloqueInstr: Bloq ➔ Instr	**genCod**(**bloqueInstr**(Bloq)): **genCod**(Bloq)
siExp: LExps ➔ ParamReales	**genCod**(**siExp**(LExps)): **genCod**(LExps)

noExp: ➔ ParamReales	**genCod**(**noExp**()): **noop**

muchasExp: LExps ✖ Exp ➔ LExps	**genCod**(**muchasExp**(LExps, Exp)):
	  genCod(LExps)
	  genCod(Exp)

unaExp: Exp ➔ LExps	**genCod**(**unaExp**(Exp)): **genCod**(Exp)

asignación: Exp ✖ Exp ➔ Exp	**genCod**(**asignación**(Exp1, Exp2)):
	  genCod(Exp1)
	  emit dup()
	  genCod(Exp2)
	  if ref!(Exp1.tipo) == **tReal**() && **ref!**(Exp2.tipo) == **tInt**() **then**
	  genAccVal(Exp2)
	  emit castReal()
	  emit store()
	  else if esDesignador(Exp2) **then**
	  emit copia(Exp2.tipo.tam)
	  else
	  emit store()
	  end

suma: Exp ✖ Exp ➔ Exp	**genCod**(**suma**(Exp1, Exp2)):
	  genCodBinAritm(Exp1, Exp2, $)
	  emit suma

and: Exp ✖ Exp ➔ Exp	**genCod**(**resta**(Exp1, Exp2)):
	  genCodBin(Exp1, Exp2)
	  emit and

or: Exp ✖ Exp ➔ Exp	**genCod**(**and**(Exp1, Exp2)):
	  genCodBin(Exp1, Exp2)
	  emit or

resta: Exp ✖ Exp ➔ Exp	**genCod**(**or**(Exp1, Exp2)):
	  genCodBinAritm(Exp1, Exp2, $)
	  emit resta

menor: Exp ✖ Exp ➔ Exp	**genCod**(**menor**(Exp1, Exp2)):
	  genCodBin(Exp1, Exp2)
	  emit menor

mayor: Exp ✖ Exp ➔ Exp	**genCod**(**mayor**(Exp1, Exp2)):
	  genCodBin(Exp1, Exp2)
	  emit mayor

menorIgual: Exp ✖ Exp ➔ Exp	**genCod**(**menorIgual**(Exp1, Exp2)):
	  genCodBin(Exp1, Exp2)
	  emit menorIgual

mayorIgual: Exp ✖ Exp ➔ Exp	**genCod**(**mayorIgual**(Exp1, Exp2)):
	  genCodBin(Exp1, Exp2)
	  emit mayorIgual

igual: Exp ✖ Exp ➔ Exp	**genCod**(**igual**(Exp1, Exp2)):
	  genCodBin(Exp1, Exp2)
	  emit igual

desigual: Exp ✖ Exp ➔ Exp	**genCod**(**desigual**(Exp1, Exp2)):
	  genCodBin(Exp1, Exp2)
	  emit desigual

mul: Exp ✖ Exp ➔ Exp	**genCod**(**mul**(Exp1, Exp2)):
	  genCodBinAritm(Exp1, Exp2, $)
	  emit mul

div: Exp ✖ Exp ➔ Exp	**genCod**(**div**(Exp1, Exp2)):
	  genCodBinAritm(Exp1, Exp2, $)
	  emit div

mod: Exp ✖ Exp ➔ Exp	**genCod**(**mod**(Exp1, Exp2)):
	  genCodBin(Exp1, Exp2)
	  emit mod

neg: Exp ➔ Exp	**genCod**(**neg**(Exp)):
	  genCod(Exp)
	  genAccVal(Exp)
	  emit neg

array: Exp ✖ Exp ➔ Exp	**genCod**(**array**(Exp1, Exp2)):
	  genCod(Exp1)
	  genCod(Exp2)
	  genAccVal(Exp2)
	  let tArray(T, _) = **ref!**(Exp1.tipo) **in**
	  emit idx(T.tam)
	  end let

expCampo: Exp ✖ **string** ➔ Exp	**genCod**(**expCampo**(Exp, id)):
	  genCod(Exp)
	  let tStruct(LCamps) = **ref!**(Exp.tipo) **in**
	  emit apilaInt(**desplazamiento**(LCamps, id))
	  end let
	  emit suma

punt: Exp ➔ Exp	**genCod**(**punt**(Exp)):
	  genCod(Exp)
	  emit fetch()

not: Exp ➔ Exp	**genCod**(**neg**(Exp)):
	  genCod(Exp)
	  genAccVal(Exp)
	  emit not

litEnt: **string** ➔ Exp	**genCod**(**litEnt**(elem)): **emit** apilaInt(elem)
litReal: **string** ➔ Exp	**genCod**(**litReal**(elem)): **emit** apilaReal(elem)

iden: **string** ➔ Exp	**genCod**(**iden**(id)): **genAccId**($.vinculo)
true: ➔ Exp	**genCode**(**true**()): **emit** apilaBool(true)
false: ➔ Exp	**genCode**(**false**()): **emit** apilaBool(false)
litCad: **string** ➔ Exp	**genCode**(**litCad**(cad)): **emit** apilaString(cad)
null: ➔ Exp	**genCode**(**null**()): **emit** apilaInt(-1)

genAccVal(E):
  if esDesignador(**ref!**(E)) **then**
    emit fetch()
  end if

genAccId(**decVar**(T,id)):
  if $.nivel = 0 **then**
    emit apilaInt($.dir)
  else
    genAccVar($)
  end if

genAccId(**paramForm**(T,id)):
  genAccVar($)

genAccId(**paramFormRef**(T,id)):
  genAccVar($)
  emit fetch()

genAccVar(V):
  emit apilaDisp(V.nivel)
  emit apilaInt(V.dir)
  emit suma()

genCodBin(Exp1, Exp2):
  genCod(Exp1)
  genAccVal(Exp1)
  genCod(Exp2)
  genAccVal(Exp2)

genCodBinAritm(Exp1, Exp2, E):
  genCod(Exp1)
  genAccVal(Exp1)
  castAritm(Exp1, E)
  genCod(Exp2)
  genAccVal(Exp2)
  castAritm(Exp1, E)

castAritm(Exp, E):
  if ref!(E.tipo) == **tReal**() && **ref!**(Exp.tipo) == **tInt**() **then**

```
   emit castReal()
```

```
genPasoParams(siParam(LParams), siExp(LExps)):
   genPasoParams(LParams, LExps)

genPasoParams(noParam(), noExp()): noop

genPasoParams(unParam(Param), unaExp(Exp)):
   genPasoParam(Param, Exp)

genPasoParams(muchosParams(LParam, Param), muchasExps(LExps, Exp)):
   genPasoParams(LParam, LExps)
   genPasoParam(Param, Exp)

genPasoParam(Param = paramFormRef(T, _), Exp):
   emit dup
   emit apilaInt(Param.dir)
   emit suma
   emit dup
   emit alloc(1)
   emit store()
   emit fetch()
   genCod(Exp)
   emit store()

genPasoParam(Param = paramForm(T, _), Exp):
   emit dup
   emit apilaInt(Param.dir)
   emit suma
   emit dup
   emit alloc(Param.tipo.tam)
   emit store()
   emit fetch()
   genCod(Exp)
   if ref!(T) == tReal() && ref!(Exp.tipo) == tInt() then
      genAccVal(Exp)
      emit castReal()
      emit store()
   else if esDesignador(Exp) then
      emit copia(Param.tipo.tam)
   else
      emit store()
   end if
```

```
genLiberaParams(siParam(LParams)):
   genLiberaParam(LParams)

genLiberaParams(noParam()): noop

genLiberaParams(unParam(Param)):
   genLiberaParam(Param)

genLiberaParams(muchosParams(LParam, Param)):
   genLiberaParams(LParam)
   genLiberaParam(Param)
```

```
genLiberaParam(Param):
   emit dup
   emit apilaInt(Param.dir)
   emit suma
   emit fetch()
   if Param == paramForm(_, _) then
      emit dealloc(tInt())
   else
      emit dealloc(Param.tipo)
   end if
```

```
desplazamiento(unCamp(Campo), id):
   return Campo.dir
desplazamiento(muchosCamps(LCamps, Campo), id):
   if Campo = campo(_, id) then
      return Campo.dir
   else
      return desplazamiento(LCamps, id)
   end if
```