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)
	cierraAmbito(ts)
	if contiene(ts,id) **then**
	error
	else
	inserta(ts,id,$)
	end if

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

	end if

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

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)): **noop**

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)): **noop**

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

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

tStruct: LCampos ➔ Tipo	**vincula**(**tStruct**(LCampos)):
	  recolectaDecs(LCampos)
	
	vincula2(**tStruct**(LCampos)):
	  vincula2(LCampos)

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

unCamp: Campo ➔ LCampos	**vincula**(**unCamp**(Campo)): **vincula**(Campo)

campo: Tipo ✖ **string** ➔ Campo	**vincula**(**campo**(Tipo, _)):
	  if Tipo != **tlden**(_) **then**
	  vincula(Tipo)
	  end if
	
	vincula2(**campo**(Tipo, _)):
	  if Tipo = **tlden**(id) **then**
	  $.vinculo = **vinculoDe**(ts,id)
	  if $.vinculo != **decType**(_,id) **then**
	  error
	  end if
	  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**(Opnd0,Opnd1)):
	  vincula(Opnd0)
	  vincula(Opnd1)

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**(Opnd0,Opnd1)):
	  vincula(Opnd0)
	  vincula(Opnd1)

div: Exp ✖ Exp ➔ Exp	**vincula**(**div**(Opnd0,Opnd1)):
	  vincula(Opnd0)
	  vincula(Opnd1)

mod: Exp ✖ Exp ➔ Exp	**vincula**(**mod**(Opnd0,Opnd1)):
	  vincula(Opnd0)
	  vincula(Opnd1)

neg: Exp ➔ Exp	**vincula**(**neg**(Opnd0)): **vincula**(Opnd0)

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**(Opnd0)): **vincula**(Opnd0)
litEnt: **string** ➔ Exp	**vincula**(**litEnt**(_)): **noop**
litReal: **string** ➔ Exp	**vincula**(**litReal**(_)): **noop**
iden: **string** ➔ Exp	**vincula**(**iden**(id)): $.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

Constructoras ASTs	Comprobación de tipos
prog: Bloq ➔ Prog	
bloq: Decs ✖ Instrs ➔ Bloq	
siDecs: LDecs ➔ Decs	-
noDecs: ➔ Decs	-
muchasDecs: LDecs ✖ Dec ➔ Ldecs	-
unaDec: Dec ➔ LDecs	-
decProc: **string** ✖ ParamForms ✖ Bloq ➔ Dec	-
decType: Tipo ✖ **string** ➔ Dec	-
decVar: Tipo ✖ **string** ➔ Dec	-
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 = ok
	end if

procInstr : **string** ✖ ParamReales ➔ Instr	**tipado**(**procInstr**(Id, ParamReales)): **tipado**(ParamReales) $.tipo = ParamReales.tipo
nlInstr : ➔ Instr	**tipado**(**nlInstr**()): $.tipo = ok

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

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

writeInstr : Exp ➔ Instr	**tipado**(**writeInstr**(Exp)): **tipado**(Exp) **if** Exp.tipo != **error then**

	$tipo = ok **end if**

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

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

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

ifInstr : Exp ✖ Bloq ➔ Instr	**tipado**(**ifInstr**(Exp, Bloq)):
	tipado(Exp);
	tipado(Bloq);
	if Exp.tipo == **tBool**() **then**
	$.tipo = Bloq.tipo
	else
	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) $.tipo = **ambosOk**(LExps.tipo, Exp.tipo)
unaExp: Exp ➔ LExps	**tipado**(**unaExp**(Exp)): **tipado**(Exp) $.tipo = Exp.tipo

3. Asignación de espacio

Constructoras ASTs	Asignación de espacio
	var dir **var** nivel
prog: Bloq ➔ Prog	**asigEspacio**(**prog**(Bloq)): **asigEspacio**(Bloq)

bloq: Decs ✖ Instrs ➔ Bloq	**asigEspacio**(**bloq**(Decs, Instrs)):
	  dir_ant = dir
	  asigEspacio(Decs)
	  asigEspacio2(Decs)
	  asigEspacio(Instrs)

	dir = dir_ant
siDecs: LDecs ➔ Decs	**asigEspacio**(**siDecs**(LDecs)): **asigEspacio**(LDecs) **asigEspacio2**(**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)): dir_ant = dir nivel++ $.nivel = nivel dir = 0 $.dir = dir **asigEspacio**(ParamForms) **asigEspacio2**(ParamForms) **asigEspacio**(Bloq) **asigEspacio2**(Bloq) $.tam = dir dir = dir_ant nivel-- **asigEspacio2**(**decProc**(Id, ParamForms, Bloq)): **noop**
decType: Tipo ✖ **string** ➔ Dec	**asigEspacio**(**decType**(Tipo,Id)): **asigTam**(Tipo) **asigEspacio2**(**decType**(Tipo,Id)): **asigTam2**(Tipo)
decVar: Tipo ✖ **string** ➔ Dec	**asigEspacio**(**decVar**(Tipo,Id)): $.dir = dir **asigTam**(Tipo) $.nivel = nivel dir += Tipo.tam

	asigEspacio2(**decVar**(Tipo,Id)): **asigTam2**(Tipo)

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)
	
	asigEspacio2(**paramFormRef**(Tipo, **string**)):
	  asigTam2(Tipo)

paramForm: Tipo ✖ **string** ➔ ParamForm	**asigEspacio**(**paramForm**(Tipo, **string**)):
	  asigTam(Tipo)
	
	asigEspacio2(**paramForm**(Tipo, **string**)):
	  asigTam2(Tipo)

tArray: Tipo ✖ **string** ➔ Tipo	**asigTam**(**tArray**(Tipo, litEnt)):
	  asigTam(Tipo)
	  $.tam = Tipo.tam * **int**(litEnt)
	
	asigTam2(**tArray**(_,_)): **noop**

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

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** $.tam = T.tam **end let** **asigTam2**(**tIden**(_)): **noop**
tStruct: LCampos ➔ Tipo	**asigTam**(**tStruct**(LCampos)): **asigTam**(LCampos) $.tam = LCampos.tam **asigTam2**(**tStruct**(_)): **noop**
muchosCamps: LCampos ✖ Campo ➔ LCampos	**asigEspacio**(**muchosCamps**(LCampos, Campo)): **asigEspacio**(LCampos) **asigEspacio**(Campo**)** $.tam = LCampos.tam + Campo.tam
unCamp: Campo ➔ LCampos	**asigEspacio**(**unCamp**(Campo)): **asigEspacio**(Campo) $.tam = Campo.tam
campo: Tipo ✖ **string** ➔ Campo	**asigEspacio**(**campo**(Tipo, **string**)): **asigTam**(Tipo) $.tam = Tipo.tam
siInstrs: LInstrs ➔ Instrs	**asigEspacio**(**siInstrs**(LInstrs)): **asigEspacio**(LInstrs)
noInstrs: ➔ Instrs	**asigEspacio**(**noInstrs**(LInstrs)): **noop**
muchasInstrs: LInstrs ✖ Instr ➔ Linstrs	**asigEspacio**(**muchasInstrs**(LInstrs, Instr)): **asigEspacio**(LInstrs) **asigEspacio**(Instr)
unaInstr: Instr ➔ Linstrs	**asigEspacio**(**unaInstr**(Instr)): **asigEspacio**(Instr)

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

4. Repertorio de instrucciones de código-p

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.

apilaDir(d): Añade el contenido de una dirección a la pila.

desapilaDir(d): Desapila el valor de la cima de la pila y lo almacena en la dirección dir.

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

desapilaInd: 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: reserva un espacio en memoria dada una dirección desapilada.

dealloc: borra el espacio de memoria reservado de una dirección desapilada.

irA(d): salta a la dirección d.

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

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

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.

apilad(n): Apila en la pila de evaluación el valor del display de nivel n.

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

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)

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

stop: Detiene la máquina.

5. Etiquetado

Constructoras ASTs	Etiquetado
	var etq = 0 **var** procPendientes = **pilaVacia**()
prog: Bloq ➔ Prog	
bloq: Decs ✖ Instrs ➔ Bloq	
siDecs: LDecs ➔ Decs	-
noDecs: ➔ Decs	-
muchasDecs: LDecs ✖ Dec ➔ Ldecs	-
unaDec: Dec ➔ LDecs	-
decProc: **string** ✖ ParamForms ✖ Bloq ➔ Dec	-
decType: Tipo ✖ **string** ➔ Dec	-
decVar: Tipo ✖ **string** ➔ Dec	-
siParam: LParams ➔ ParamForms	-

noParam: ➔ ParamForms	-
muchosParams: LParams ✖ ParamForm➔ LParams	-
unParam: ParamForm ➔ LParams	-
paramFormRef: Tipo ✖ **string** ➔ ParamForm	**etiquetado**(**paramFormRef**(Tipo, _)): $.prim = etq **etiquetado**(Tipo) etq++ $.sig = etq
paramForm: Tipo ✖ **string** ➔ ParamForm	**etiquetado**(**paramForm**(Tipo, _)): $.prim = etq **etiquetado**(Tipo) etq++ $.sig = etq
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	**etiquetado**(**muchosCamps**(LCampos, Campo)): $.prim = etq **etiquetado**(LCampos) **etiquetado**(Campo) $.sig = etq
unCamp: Campo ➔ LCampos	**etiquetado**(**unCamp**(Campo)): $.prim = etq **etiquetado**(Campo) $.sig = etq
campo: Tipo ✖ **string** ➔ Campo	**etiquetado**(**Campo**(Tipo, _)): $.prim = etq **etiquetado**(Tipo) etq++ $.sig = etq
siInstrs: LInstrs ➔ Instrs	**etiquetado**(**siInstrs**(LInstrs)): $.prim = etq

	etiquetado(LInstrs) $.sig = etq
noInstrs: ➔ Instrs	**etiquetado**(**noInstrs**()): $.prim = etq 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, ParamReales)): $.prim = etq **etiquetado**(ParamReales) etq++ $.sig = etq
nlInstr : ➔ Instr	**etiquetado**(**nlInstr**()): $.prim = etq etq++ $.sig = etq
newInstr : Exp ➔ Instr	**etiquetado**(**newInstr**(Exp)): $.prim = etq **etiquetado**(Exp) etq++ $.sig = etq
readInstr : Exp ➔ Instr	**etiquetado**(**readInstr**(Exp)): $.prim = etq **etiquetado**(Exp) etq++ $.sig = etq
writeInstr : Exp ➔ Instr	**etiquetado**(**writeInstr**(Exp)): $.prim = etq **etiquetado**(Exp) etq++ $.sig = etq
deleteInstr : Exp ➔ Instr	**etiquetado**(**deleteInstr**(Exp)):

	$.prim = etq **etiquetado**(Exp) etq++ $.sig = etq
whileInstr : Exp ✖ Bloq ➔ Instr	**etiquetado**(**whileInstr**(Exp, Bloq)): $.prim = etq **etiquetado**(Exp) etq++ **etiquetado**(Bloq) etq++ $.sig = etq
ifElseInstr : Exp ✖ Bloq ✖ Bloq ➔ Instr	**etiquetado**(**ifElseInstr**(Exp,Bloq1, Bloq2)): $.prim = etq **etiquetado**(Exp) etq++ **etiquetado**(Bloq1) etq++ **etiquetado**(Bloq2) etq++ $.sig = etq
ifInstr : Exp ✖ Bloq ➔ Instr	**etiquetado**(**ifInstr**(Exp,Bloq)): $.prim = etq **etiquetado**(Exp) etq++ **etiquetado**(Bloq) etq++ $.sig = etq
bloqueInstr: Bloq ➔ Instr	**etiquetado**(**bloqueInstr**(Bloq)): $.prim = etq **etiquetado**(Bloq) etq++ $.sig = etq
siExp: LExps ➔ ParamReales	**etiquetado**(**siExp**(LExps)): $.prim = etq **etiquetado**(LExps) $.sig = etq
noExp: ➔ ParamReales	**etiquetado**(**noExp**(LExps)): $.prim = etq 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**(Exp,Exp)): $.prim = etq **etiquetado**(Exp) etq++ $.sig = etq
suma: Exp ✖ Exp ➔ Exp	**etiquetado**(**suma**(Exp1, Exp2)): $.prim = etq **etiquetadoBin**(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 **etiquetadoBin**(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 **etiquetadoBin**(Exp1, Exp2) etq++ $.sig = etq
div: Exp ✖ Exp ➔ Exp	**etiquetado**(**div**(Exp1, Exp2)): $.prim = etq **etiquetadoBin**(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) etq++ $.sig = etq
array: Exp ✖ Exp ➔ Exp	**etiquetado**(**array**(Exp1, Exp2)): $.prim = etq **etiquetado**(Exp1) **etiquetado**(Exp2) etq++ $.sig = etq
expCampo: Exp ✖ **string** ➔ Exp	**etiquetado**(**expCampo**(Exp, _)): $.prim = etq **etiquetado**(Exp) etq++ $.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) 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 etq++ $.sig = etq
true: ➔ Exp	
false: ➔ Exp	
litCad: **string** ➔ Exp	**etiquetado**(**litCad**(id)): $.prim = etq **etiquetadoAccVal**(id.vinculo) $.sig = etq
null: ➔ Exp	
	etiquetadoBin(Exp1, Exp2): **etiquetado**(Exp1) **etiquetadoAccVal**(Exp1) **etiquetado**(Exp2) **etiquetadoAccVal**(Exp2)

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

	etiquetadoPasoParam(**proc**(_,_,_,_),PReal): etq += 3 **etiquetado**(PReal) etq++

6. Generación de código

Constructoras ASTs	Código
	var procPendientes = **pilaVacia**()
prog: Bloq ➔ Prog	
bloq: Decs ✖ Instrs ➔ Bloq	
siDecs: LDecs ➔ Decs	-
noDecs: ➔ Decs	-
muchasDecs: LDecs ✖ Dec ➔ Ldecs	-
unaDec: Dec ➔ LDecs	-
decProc: **string** ✖ ParamForms ✖ Bloq ➔ Dec	
decType: Tipo ✖ **string** ➔ Dec	
decVar: Tipo ✖ **string** ➔ Dec	
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 irA($.prim)

procInstr : **string** ✖ ParamReales ➔ Instr	**genCod**(**procInstr**(ParamReales)): **genCod**(ParamReales)
nlInstr : ➔ Instr	**genCod**(**nlInstr**(Instr)): **noop**

newInstr : Exp ➔ Instr	**genCod**(**newInstr**(Instr)):
	  genCod(Exp)
	  emit irA($.prim)

readInstr : Exp ➔ Instr	**genCod**(**readInstr**(Instr)):
	  genCod(Exp)
	  emit irA($.prim)

writeInstr : Exp ➔ Instr	**genCod**(**whileInstr**(Instr)):
	  genCod(Exp)
	  emit irA($.prim)

deleteInstr : Exp ➔ Instr	**genCod**(**deleteInstr**(Exp)):
	  genCod(Exp)
	  emit irA($.prim)

whileInstr : Exp ✖ Bloq ➔ Instr	**genCod**(**whileInstr**(Exp,Bloq)):
	  genCod(Exp)
	  emit irF($.sig)
	  genCod(Bloq)
	  emit irA($.prim)

ifElseInstr : Exp ✖ Bloq ✖ Bloq ➔ Instr	**genCod**(**ifElseInstr**(Exp,Bloq1,Bloq2)):
	  genCod(Exp)
	  emit irF($.sig)
	  genCod(Bloq1)
	  genCod(Bloq2)

ifInstr : Exp ✖ Bloq ➔ Instr	**genCod**(**ifInstr**(Exp,Bloq)):
	  genCod(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(E0)
	  genCod(E1)
	  genAsig(E1)

suma: Exp ✖ Exp ➔ Exp	**genCod**(**suma**(Exp1, Exp2)): **genCodBin**(Exp1, Exp2, suma)
and: Exp ✖ Exp ➔ Exp	**genCod**(**resta**(Exp1, Exp2)): **genCodBin**(Exp1, Exp2, resta)
or: Exp ✖ Exp ➔ Exp	**genCod**(**and**(Exp1, Exp2)): **genCodBin**(Exp1, Exp2, and)
resta: Exp ✖ Exp ➔ Exp	**genCod**(**or**(Exp1, Exp2)): **genCodBin**(Exp1, Exp2, or)
menor: Exp ✖ Exp ➔ Exp	**genCod**(**menor**(Exp1, Exp2)): **genCodBin**(Exp1, Exp2, menor)
mayor: Exp ✖ Exp ➔ Exp	**genCod**(**mayor**(Exp1, Exp2)): **genCodBin**(Exp1, Exp2, mayor)
menorIgual: Exp ✖ Exp ➔ Exp	**genCod**(**menorIgual**(Exp1, Exp2)): **genCodBin**(Exp1, Exp2, menorIgual)
mayorIgual: Exp ✖ Exp ➔ Exp	**genCod**(**mayorIgual**(Exp1, Exp2)): **genCodBin**(Exp1, Exp2, mayorIgual)
igual: Exp ✖ Exp ➔ Exp	**genCod**(**igual**(Exp1, Exp2)): **genCodBin**(Exp1, Exp2, igual)
desigual: Exp ✖ Exp ➔ Exp	**genCod**(**desigual**(Exp1, Exp2)): **genCodBin**(Exp1, Exp2, desigual)
mul: Exp ✖ Exp ➔ Exp	**genCod**(**mul**(Exp1, Exp2)):

	genCodBin(Exp1, Exp2, mul)
div: Exp ✖ Exp ➔ Exp	**genCod**(**div**(Exp1, Exp2)): **genCodBin**(Exp1, Exp2, div)
mod: Exp ✖ Exp ➔ Exp	**genCod**(**mod**(Exp1, Exp2)): **genCodBin**(Exp1, Exp2, mod)
neg: Exp ➔ Exp	
array: Exp ✖ Exp ➔ Exp	**genCod**(**array**(Exp1, Exp2)): **genCod**(Exp1) **genCod**(Exp2) **genAccVal**(Exp2) **let tArray**(T, _) = **ref!**(Exp1.tipo) **in** **emit** apilaInt(T.tam) **end let** **emit** mul **emit** suma
expCampo: Exp ✖ **string** ➔ Exp	**genCod**(**expCampo**(Exp, id)): **genCod**(Exp) **let tStruct**(LCampos) = **ref!**(Exp.tipo) **in** **emit** apilaInt(**desplazamiento**(LCampos)) **end let** **emit** suma
punt: Exp ➔ Exp	**genCod**(**punt**(Exp)): **genCod**(Exp) **emit** apilaInd()
not: Exp ➔ Exp	
litEnt: **string** ➔ Exp	
litReal: **string** ➔ Exp	
iden: **string** ➔ Exp	
true: ➔ Exp	
false: ➔ Exp	
litCad: **string** ➔ Exp	
null: ➔ Exp	

	genAsig(E):
	if esDesignador(**ref!**(E)) **then**
	emit copia(E.tipo.tam)
	else
	emit desapilaInd()
	end

	genAccVal(E):

	if esDesignador(**ref!**(E)) **then**
	  emit apilaInd()
	end if

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

	desplazamiento(**unCamp**(Campo)):
	  emit apilaInt(Campo.tipo.tam)
	
	desplazamiento(**muchosCamps**(LC, Campo)):
	  desplazamiento(LC)
	  emit apilaInt(Campo.tipo.tam)
	  emit suma

	recolectaProcs(**siDecs**(LDecs)):
	  recolectaProcs(LDecs)
	
	recolectaProcs(**noDecs**()): **noop**
	
	recolectaProcs(**muchasDecs**(Decs,Dec)):
	  recolectaProcs(Decs)
	  recolectaProcs(Dec)
	
	recolectaProcs(**unaDec**(Dec)):
	  recolectaProcs(Dec)
	
	recolectaProcs(**decVar**(_,_)): **noop**
	
	recolectaProcs(**decType**(_,_)): **noop**
	
	recolectaProcs(**decProc**(_,_,_,_)):
	  apila(procPendientes, $)