Desarrollo de constructores de ASTs para Tiny(1)

Óscar Morujo Fernández

Sofía Capmany Fernández

G16

Especificación sintáctica:

PROGRAMA := LDECS **sep_di** LINST;
PROGRAMA := LINST;
LDECS := LDECS **scol** DEC;
LDECS := DEC;
DEC := **var** TIPO **id**;
DEC := **type** TIPO **id;**
DEC := **proc id pap** LPARAMS **pcierre** BLOQUE;
DEC := **proc id pap pcierre** BLOQUE;
LPARAMS := LPARAMS **coma** PARAM;
LPARAMS := PARAM;
PARAM := TIPO **amp id**;
PARAM := TIPO **id**;
TIPO := **r_bool;**
TIPO := **r_real;**
TIPO := **r_int ;**
TIPO := **r_string ;**
TIPO := **array cap ent ccierre of** TIPO;
TIPO := **record llap** CAMPOS **llcierre**;
TIPO := **pointer** TIPO**;**
TIPO := **id**;
CAMPOS := CAMPOS **scol** CAMPO;
CAMPOS := CAMPO;
CAMPO := TIPO **id**;
LINST := LINST **scol** INST;
LINST := INST;
INST := E0 **igual** E0;
INST := **if** E0 **then** AUX_LINST **endif**;
INST := **if** E0 **then** AUX_LINST **else** AUX_LINST **endif**;
INST := **while** E0 **do** AUX_LINST **endwhile**;
AUX_LINST := LINST;
AUX_LINST := **ƛ** ;
INST:= **read** E0;
INST := **write** E0;
INST := **nl**;
INST := **new** E0;
INST := **delete** E0;
INST := **call id pap** REAL_PARAMS **pcierre**;
INST := **call id pap pcierre**;

REAL_PARAMS := REAL_PARAMS **coma** E0;
REAL_PARAMS := E0;
INST := BLOQUE;
BLOQUE := **llap** PROGRAMA **llcierre**;
BLOQUE := **llap llcierre**;
E0 := E1 **mas** E0;
E0 := E1 **menos** E1;
E0 := E1;
E1 := E1 OP1AI E2;
E1 := E2;
E2 := E2 OP2AI E3;
E2 := E3;
E3 := E4 OP3NA E4;
E3 := E4;
E4 := **menos** E5;
E4 := **not** E4;
E4 := E5;
E5 := E5 **cap** E0 **ccierre** ;
E5 := E5 **flecha id**;
E5 := E5 **punto id**;
E5 := E6;
E6 := por E6
E6 := E7;
E7 := **pap** E0 **pcierre**;
E7 :=**null**;
E7 := **id**;
E7 := **false**;
E7 := **true**;
E7 := **cadena**;
E7 := **real**;
E7 := **ent**;
OP3NA := **por**;
OP3NA := **div**;
OP3NA := **mod**;
OP2AI := **bne**;
OP2AI := **beq**;
OP2AI := **ble**;
OP2AI := **bge**;
OP2AI := **blt**;
OP2AI :=**bgt**;
OP1AI := **and**;
OP1AI := **or**;

Sintaxis abstracta:

Géneros o conceptos sintácticos:

Prog, LDecs, LInst, Dec, Tipo, Param, LParams , Bloque, LCampos, Campo, Inst, Exp, LInst_aux, y LReal_params.

Constructoras:

prog_con_decs:	LDecs x LInst -> Prog
prog_sin_decs:	LInst -> Prog
decs_una:	Dec -> LDecs
decs_muchas:	LDecs x Dec -> LDecs
dec_var:	Tipo x String -> Dec
dec_type :	Tipo x String -> Dec
dec_proc_con_params:	String x LParams x Bloque -> Dec
dec_proc_sin_params:	String x Bloque -> Dec
l_params_uno:	Param -> Lparams
l_params_muchos:	LParams x Param -> Lparams
param_con_amp :	Tipo x String -> Param
param_sin_amp :	Tipo x String -> Param
tipo_Entero:	->Tipo
tipo_Real:	->Tipo
tipo_String:	->Tipo
tipo_Bool:	->Tipo
tipo_Id:	String -> Tipo
tipo_Puntero:	Tipo -> Tipo
tipo_Array:	String x Tipo -> Tipo
tipo_Reg:	LCampos -> Tipo
l_campos_uno:	Campo -> LCampos
l_campos_muchos:	LCampos x Campo ->LCampos
campo:	Tipo x string -> Campo
l_inst_muchas:	LInst x Inst -> LInst
l_inst_una :	Inst -> LInst
inst_asig:	Exp x Exp -> Inst
inst_if_then :	Exp x LInst_aux -> Inst
inst_if_then_else:	Exp x LInst_aux x LInst_aux ->Inst
linst_aux_vacia:	-> LInst_aux
linst_aux:	LInst -> LInst_aux
inst_while:	Exp x LInst_aux -> Inst
inst_read:	Exp -> Inst
inst_write:	Exp -> Inst
inst_nl:	-> Inst

inst_new:	Exp -> Inst
inst_delete:	Exp -> Inst
inst_call_sin_params:	String -> Inst
inst_call_con_params:	String x LReal_params -> Inst
l_real_params_uno:	Exp -> LReal_params
l_real_params_muchos:	LReal_params x Exp -> LReal_params
inst_compuesta:	Bloque -> Inst
bloque :	Prog -> Bloque
bloque_vacio:	->Bloque
suma:	Exp x Exp -> Exp
resta:	Exp x Exp -> Exp
mul:	Exp x Exp -> Exp
div:	Exp x Exp -> Exp
beq:	Exp x Exp -> Exp
bne:	Exp x Exp -> Exp
ble:	Exp x Exp -> Exp
bge:	Exp x Exp -> Exp
blt:	Exp x Exp -> Exp
bgt:	Exp x Exp -> Exp
c_and:	Exp x Exp -> Exp
c_or:	Exp x Exp -> Exp
c_mod:	Exp x Exp -> Exp
c_not:	Exp -> Exp
menos_unario:	Exp-> Exp
numEnt:	String -> Exp
numReal:	String -> Exp
identificador:	String -> Exp
c_false:	-> Exp
c_true:	-> Exp
c_null :	-> Exp
c_str:	String ->Exp
index :	Exp x Exp-> Exp
indireccion:	Exp -> Exp
punto:	Exp x String -> Exp
flecha:	Exp x String -> Exp

Constructor de árboles de sintaxis abstracta(ASTs):

PROGRAMA := LDECS **sep_di** LINST;

$PROGRAMA.a = programa\_con\_decs\ (LDecs.a, LINST.a)$

PROGRAMA := LINST;

$PROGRAMA.a = programa\_sin\_decs\ (LINST.a)$

LDECS := LDECS **scol** DEC;

$LDECS_0.a$ =decs_muchas($LDECS_1.a, DEC.a$)

LDECS := DEC;

$LDECS.a$ =decs_una($DEC.a$)

DEC := **var** TIPO **id**;

$DEC.a$ =dec_var(TIPO.a, id.lex)

DEC := **type** TIPO **id;**

$DEC.a$ =dec_type(TIPO.a, id.lex)

DEC := **proc id pap** LPARAMS **pcierre** BLOQUE;

$DEC.a$ =dec_proc_con_params(id.lex, LPARAMS.a,BLOQUE.a)

DEC := **proc id pap pcierre** BLOQUE;

$DEC.a$ =dec_proc_sin_params(id.lex,BLOQUE.a)

LPARAMS := LPARAMS **coma** PARAM;

$LPARAMS_0.a$ =l_params_muchos($LPARAMS_1.a,\ PARAM.a$)

LPARAMS := PARAM;

$LPARAMS.a$ =l_params_uno($PARAM.a$)

PARAM := TIPO **amp id**;

$PARAM.a$ =param_con_amp(TIPO.a,id.lex)

PARAM := TIPO **id**;

$PARAM.a$ =param_sin_amp(TIPO.a,id.lex)

TIPO := **r_bool;**

$TIPO.a$ = tipo_Bool()

TIPO := **r_real;**

$TIPO.a$ = tipo_Real()

TIPO := **r_int ;**

$TIPO.a$ = tipo_Entero()

TIPO := **r_string ;**

$TIPO.a$ = tipo_String()

TIPO := **array cap ent ccierre of** TIPO;

$TIPO_0.a$ = tipo_Array($ent.lex, TIPO_1.a$)

TIPO := **record llap** CAMPOS **llcierre**;

$TIPO.a$ = tipo_Reg(CAMPOS.a)

TIPO := **pointer** TIPO**;**

$TIPO_0.a$ = Tipo_Puntero($TIPO_1.a$)

TIPO := **id**;

$TIPO.a$ = tipo_Id(id.lex)

CAMPOS := CAMPOS **scol** CAMPO;

$CAMPOS_0.a$= l_campos_muchos($CAMPOS_1.a, CAMPO.a$)

CAMPOS := CAMPO;

$CAMPOS.a$= l_campos_uno(CAMPO.a)

CAMPO := TIPO **id**;

$CAMPO.a$ =campo(TIPO.a,id.lex)

LINST := LINST **scol** INST;

$LINST_0.a$ =l_inst_muchas($LINST_1.a, INST.a$)

LINST := INST;

$LINST.a$ =l_inst_una(INST.a)

INST := E0 **igual** E0;

$INST.a$ =inst_asig($E0_0.a, E0_1.a$)

INST := **if** E0 **then** AUX_LINST **endif**;

$INST.a$ =inst_if_then($E0.a, AUX\_LINST.a$)

INST := **if** E0 **then** AUX_LINST **else** AUX_LINST **endif**;

$INST.a$ =inst_if_then_else($E0.a, AUX\_LINST_0.a, AUX\_LINST_1.a$)

INST := **while** E0 **do** AUX_LINST **endwhile**;

$INST.a$ =inst_while($E0.a, AUX\_LINST.a$)

AUX_LINST := LINST;

$AUX\_LINST.a$ =linst_aux($LINST.a$)

AUX_LINST := **ƛ** ;

$AUX\_LINST.a$ =linst_aux_vacia()

INST:= **read** E0;

$INST.a$ =inst_read($E0.a$)

INST := **write** E0;

$INST.a$ =inst_write($E0.a$)

INST := **nl**;

$INST.a$ =inst_nl()

INST := **new** E0;

$INST.a$ =inst_new($E0.a$)

INST := **delete** E0;

$INST.a$ =inst_delete($E0.a$)

INST := **call id pap** REAL_PARAMS **pcierre**;

$INST.a$ =inst_call_con_params($id.lex$, $REAL\_PARAMS.a$)

INST := **call id pap pcierre**;

$INST.a$ =inst_call_sin_params($id.lex$)

REAL_PARAMS := REAL_PARAMS **coma** E0;

$REAL\_PARAMS_0.a$ =l_real_params_muchos($REAL\_PARAMS_1.a$, $E0.a$)

REAL_PARAMS := E0;

$REAL\_PARAMS.a$ =l_real_params_uno($E0.a$)

INST := BLOQUE;

$INST.a$ =inst_compuesta($BLOQUE.a$)

BLOQUE := **llap** PROGRAMA **llcierre**;

$BLOQUE.a$ =bloque($PROGRAMA.a$)

BLOQUE := **llap llcierre**;

$BLOQUE.a$ =bloque_vacio()

E0 := E1 **mas** E0;

$E0_0.a$ = exp_binaria("+",$E1.a$, $E0_1.a$)

E0 := E1 **menos** E1;

$E0.a$ = exp_binaria("-",$E1_0.a$, $E1_1.a$)

E0 := E1;

$E0.a = E1.a$

E1 := E1 OP1AI E2;

$E1_0.a$ =exp_binaria(OP1AI.op,$E1_1.a$, $E2.a$)

E1 := E2;

$E1.a = E2.a$

E2 := E2 OP2AI E3;

$E2_0.a$ =exp_binaria(OP2AI.op,$E2_1.a$, $E3.a$)

E2 := E3;

$E2.a = E3.a$

E3 := E4 OP3NA E4;

$E3.a$ =exp_binaria(OP3NA.op,$E4_0.a$, $E4_1.a$)

E3 := E4;

$E3.a = E4.a$

E4 := **menos** E5;

$E4.a$ = menos_unario(E5.a)

E4 := **not** E4;

$E4_0.a$ = c_not($E4_1.a$)

E4 := E5;

$E4.a = E5.a$

E5 := E5 **cap** E0 **ccierre** ;

$E5_0.a$ = index($E5_1.a$, $E0.a$)

E5 := E5 **flecha id**;

$E5_0.a$ = flecha($E5_1.a$,id.lex)

E5 := E5 **punto id**;

$E5_0.a$ = punto($E5_1.a$,id.lex)

E5 := E6;

$E5.a = E6.a$

E6 := por E6

$E6_0.a$ = indireccion($E6_1.a$)

E6 := E7;

$E6.a = E7.a$

E7 := **pap** E0 **pcierre**;

$E7.a = E0.a$

E7 :=**null**;

$E7.a$ =c_null()

E7 := **id**;

$E7.a$ =identificador(id.lex)

E7 := **false**;

$E7.a$ = c_false()

E7 := **true**;

$E7.a$ = c_true()

E7 := **cadena**;

$E7.a$ = c_str(cadena.lex)

E7 := **real**;

$E7.a$ = numReal(real.lex)

E7 := **ent**;

$E7.a$ = numEnt(ent.lex)

OP3NA := **por**;

$OP3NA.op$ = " * "

OP3NA := **div**;

$OP3NA.op$ = "/"

OP3NA := **mod**;

$OP3NA.op$ = "%"

OP2AI := **bne**;

$OP2AI.op$ = "!= "

OP2AI := **beq**;

$OP2AI.op$ = " == "

OP2AI := **ble**;

$OP2AI.op$ = " <= "

OP2AI := **bge**;

$OP2AI.op$ = " >= "

OP2AI := **blt**;

$OP2AI.op$ = " < "

OP2AI :=**bgt**;

$OP2AI.op$ = " > "

OP1AI := **and**;

$OP1AI.op$ = "and"

OP1AI := **or**;

$OP1AI.op$ = "or"

Funciones semánticas:

- fun programa(LDECS.a,LINST.a) {
    if LDECS.a != null
      return prog_con_decs(LDECS.a,LINST.a)
    else
      return prog_sin_decs(LINST.a)
  }

- fun exp_binaria(Op,Arg0,Arg1) {
  switch (Op){
  case '+': return suma(Arg0,Arg1)
  case '-': return resta(Arg0,Arg1)
  case '*': return mul(Arg0,Arg1)
  case '/': return div(Arg0,Arg1)
  case '%': return mod(Arg0,Arg1)
  case '==': return beq(Arg0,Arg1)
  case '<=': return ble(Arg0,Arg1)
  case '>=': return bge(Arg0,Arg1)
  case '!=': return bne(Arg0,Arg1)
  case '<': return blt(Arg0,Arg1)
  case '>': return bgt(Arg0,Arg1)
  case 'and': return and(Arg0,Arg1)
  case 'or': return or(Arg0,Arg1)
  }
  }

Acondicionamiento para implementación descendente:

Primer paso: Factorizar.

DEC := **proc id pap** LPARAMS **pcierre** BLOQUE; $DEC.a$ =dec_proc_con_params(id.lex, LPARAMS.a,BLOQUE.a) DEC := **proc id pap pcierre** BLOQUE; $DEC.a$ =dec_proc_sin_params(id.lex,BLOQUE.a)	DEC := **proc id pap** RES_DEC; $RES\_DEC.ah$ =id.lex $DEC.a = RES\_DEC.a$ RES_DEC := LPARAMS **pcierre** BLOQUE; $RES\_DEC.a$ =dec_proc_con_params($RES\_DEC.ah$, LPARAMS.a,BLOQUE.a) RES_DEC := **pcierre** BLOQUE; $RES\_DEC.a$ =dec_proc_sin_params($RES\_DEC.ah$,BLOQUE.a)
PARAM := TIPO **amp id**; $PARAM.a$ =param_con_amp(TIPO.a,id.lex) PARAM := TIPO **id**; $PARAM.a$ =param_sin_amp(TIPO.a,id.lex)	PARAM := TIPO RPARAM; $RPARAM.ah = TIPO.a$ $PARAM.a = RPARAM.a$ RPARAM := **id**;

	$RPARAM.a$ =param_sin_amp($RPARAM.ah, id.lex$) RPARAM := **amp id**; $RPARAM.a$ =param_con_amp($RPARAM.ah, id.lex$)
INST := **if** E0 **then** AUX_LINST **endif**; $INST.a$ =inst_if_then($E0.a, AUX\_LINST.a$) INST := **if** E0 **then** AUX_LINST **else** AUX_LINST **endif**; $INST.a$ =inst_if_then_else($E0.a, AUX\_LINST_0.a, AUX\_LINST_1.a$)	INST := **if** E0 **then** AUX_LINST RES_IF; $RES\_IF.ah1$ =$E0.a$ $RES\_IF.ah2$ =$AUX\_LINST.ah$ $INST.a$ = $RES\_IF.a$ RES_IF := **else** AUX_LINST **endif** ; $RES\_IF.a$ =inst_if_then_else($RES\_IF.ah1, RES\_IF.ah2, AUXLINST.a$) RES_IF := **endif** ; $RES\_IF.a$ =inst_if_then($RES\_IF.ah1, RES\_IF.ah2$)
INST := **call id pap** REAL_PARAMS **pcierre**; $INST.a$ =inst_call_con_params($id.lex$, $REAL\_PARAMS.a$) INST := **call id pap pcierre** $INST.a$ =inst_call_sin_params($id.lex$)	INST := **call id pap** RES_CALL; $RES\_CALL.ah$ =id.lex $INST.a$ = $RES\_CALL.a$ RES_CALL := **pcierre;** $RES\_CALL.a$ =inst_call_sin_params($RES\_CALL.ah$) RES_CALL := REAL_PARAMS **pcierre;** $RES\_CALL.a$ =inst_call_con_params($RES\_CALL.ah, REAL\_PARAMS.a$)
BLOQUE := **llap** PROGRAMA **llcierre**; $BLOQUE.a$ =bloque($PROGRAMA.a$) BLOQUE := **llap llcierre**; $BLOQUE.a$ =bloque_vacio()	BLOQUE := **llap** RBLOQUE; $BLOQUE.a$ = $RBLOQUE.a$ RBLOQUE := **llcierre**; $RBLOQUE.a$ =bloque_vacio() RBLOQUE := PROGRAMA **llcierre**; $RBLOQUE.a$ =bloque(PROGRAMA.a)
E0 := E1 **mas** E0; $E0_0.a$ = exp_binaria("+",$E1.a, E0_1.a$) E0 := E1 **menos** E1; $E0.a$ = exp_binaria("-",$E1_0.a, E1_1.a$) E0 := E1; $E0.a$ = $E1.a$	E0 := E1 RES0; $RES0.ah$ = $E1.a$ $E0.a$ = $RES0.a$ RES0 := **mas** E0; $RES0.a$ = exp_binaria("+",$RES0.ah, E0.a$) RES0:= **menos** E1; $RES0.a$ = exp_binaria("-",$RES0.ah, E1.a$) RES0 := **ƛ** ; $RES0.a$ =$RES0.ah$

E3 := E4 OP3NA E4; $E3.a$ =exp_binaria(OP3NA.op,$E4_0.a$, $E4_1.a$) E3 := E4; $E3.a = E4.a$	E3 := E4 RES3; $RES3.ah = E4.a$ $E3.a = RES3.a$ RES3 := OP3NA E4; $RES3.a$ =exp_binaria(OP3NA.op,$RES3.ah$, $E4.a$) RES3:=**ƛ** ; $RE3.a = RES3.ah$
E5 := E5 **cap** E0 **ccierre** ; $E5_0.a$ = index($E5_1.a$, $E0.a$) E5 := E5 **flecha id**; $E5_0.a$ = flecha($E5_1.a$,id.lex) E5 := E5 **punto id**; $E5_0.a$ = punto($E5_1.a$,id.lex) E5 := E6; $E5.a = E6.a$	E5 := E5 RES5; $RES5.ah = E5_1.a$ $E5_0.a = RES5.a$ E5 := E6; $E5.a = E6.a$ RES5 := **cap** E0 **ccierre**; $RES5.a$ = index($RES5.ah$, $E0.a$) RES5 := **flecha id**; $RES5.a$ = flecha($RES5.ah$, $id.lex$) RES5 := **punto id**; $RES5.a$ = punto($RES5.ah$, $id.lex$)

Segundo paso : Eliminar recursión a izquierdas

LDECS := LDECS **scol** DEC; $LDECS_0.a$ =decs_muchas($LDECS_1.a, DEC.a$) LDECS := DEC; $LDECS.a$ =decs_una($DEC.a$)	LDECS := DEC RDECS; $RDECS.ah$ =decs_una($DEC.a$) $DECS.a$ =$RDECS.a$ RDECS := **scol** DEC RDECS; $RDECS_1.ah$ =decs_muchas($RDECS_0.ah, DEC.a$) $RDECS_0.a$ =$RDECS_1.a$ RDECS := **ƛ** ; $RDECS.a$ =$RDECS.ah$
LPARAMS := LPARAMS **coma** PARAM; $LPARAMS_0.a$ =l_params_muchos($LPARAMS_1.a$, $PARAM.a$) LPARAMS := PARAM; $LPARAMS.a$ =l_params_uno($PARAM.a$)	LPARAMS:=PARAM RLPARAMS; $RLPARAMS.ah$ =l_params_uno($PARAM.a$) $LPARAMS.a$ =$RLPARAMS.a$ RLPARAMS:= **coma** PARAM RLPARAMS; $RLPARAMS_1.ah$ =l_params_muchos(

	$RLPARAMS_0.ah,\ PARAM.a)$ $RLPARAMS_0.a = RLPARAMS_1.a$ RLPARAMS:= **ƛ** ; $RLPARAMS.a = RLPARAMS.ah$
CAMPOS := CAMPOS **scol** CAMPO; $CAMPOS_0.a$= l_campos_muchos($CAMPOS_1.a, CAMPO.a$) CAMPOS := CAMPO; $CAMPOS.a$= l_campos_uno(CAMPO.a)	CAMPOS:= CAMPO RCAMPOS; $RCAMPOS.ah$= l_campos_uno($CAMPO.a$) $CAMPOS.a = RCAMPOS.a$ RCAMPOS:= **scol** CAMPO RCAMPOS; $RCAMPOS_1.ah$= l_campos_muchos($RCAMPOS_0.ah, CAMPO.a$) $RCAMPOS_0.a = RCAMPOS_1.a$ RCAMPOS:= **ƛ** ; $RCAMPOS.a = RCAMPOS_0.ah$
LINST := LINST **scol** INST; $LINST_0.a$ =l_inst_muchas($LINS_1.a, INST.a$) LINST := INST; $LINST.a$ =l_inst_una(INST.a)	LINST := INST RLINST; $RLINST.ah$ =l_inst_una(INST.a) $LINST.a$ =RLINST.a RLINST := **scol** INST RLINST; $RLINST_1.ah$ =l_inst_muchas($RLINST_0.ah$,INST.a) $RLINST_0.a = RLINST_1.a$ RLINST := **ƛ** ; $RLINST.a = RLINST.ah$
REAL_PARAMS := REAL_PARAMS **coma** E0; $REAL\_PARAMS_0.a$ =l_real_params_muchos($REAL\_PARAMS_1.a, E0.a$) REAL_PARAMS := E0; $REAL\_PARAMS.a$ =l_real_params_uno($E0.a$)	REAL_PARAMS := E0 RES_PARAMS; $RES\_PARAMS.ah$ =l_real_params_uno($E0.a$) $REAL\_PARAMS.a = RES\_PARAMS.a$ RES_PARAMS := **coma** E0 RES_PARAMS; $RES\_PARAMS_1.ah$ =l_real_params_muchos($RES\_PARAMS_0.ah, E0.a$) $RES\_PARAMS_0.a = RES\_PARAMS_1.a$ RES_PARAMS := **ƛ** ; $RES\_PARAMS.a = RES\_PARAMS.ah$
E1 := E1 OP1AI E2; $E1_0.a$ =exp_binaria(OP1AI.op,$E1_1.a, E2.a$) E1 := E2; $E1.a = E2.a$	E1 := E2 RES1; $RES1.ah = E2.a$ $E1.a = RES1.a$ RES1 := OP1AI E2 RES1; $RES1_1.ah = exp\_binaria(OP1AI.op, RES1_0.ah, E2.a)$ $RES1_0.a = RES1_1.a$

	RES1 := **ƛ** ; $RES1.a = RES1.ah$
E2 := E2 OP2AI E3; $E2_0.a$ =exp_binaria(OP2AI.op,$E2_1.a, E3.a$) E2 := E3; $E2.a = E3.a$	E2 := E3 RES2; $RES2.ah = E3.a$ $E2.a = RES2.a$ RES2 := OP2AI E3 RES2; $RES2_1.ah = exp\_binaria(OP2AI.op, RES2_0.ah, E3.a)$ $RES2_0.a = RES2_1.a$ RES2:= **ƛ** ; $RES2.a = RES2.ah$
E5 := E5 RES5; $RES5.ah = E5_1.a$ $E5_0.a = RES5.a$ E5 := E6; $E5.a = E6.a$ RES5 := **cap** E0 **ccierre**; $RES5.a$ = index($RES5.ah, E0.a$) RES5 := **flecha id**; $RES5.a$ = flecha($RES5.ah, id.lex$) RES5 := **punto id**; $RES5.a$ = punto($RES5.ah, id.lex$)	E5 := E6 RES_RES5; $RES\_RES5.ah = E6.a$ $E5.a = RES\_RES5.a$ RES_RES5 := RES5 RES_RES5 ; $RES5.ah = RES\_RES5_0.ah$ $RES\_RES5_1.ah = RES5.a$ $RES\_RES5_0.a = RES\_RES5_1.a$ RES_RES5 := **ƛ** ; $RES\_RES5.a = RES\_RES5.ah$

Gramática transformada:

Reglas que son producto de factorizar
Reglas que son producto de eliminar recursión a izquierdas
Reglas que son producto de factorizar y eliminar recursión a izquierdas

PROGRAMA := LDECS **sep_di** LINST; $PROGRAMA.a = programa\_con\_decs\ (LDECS.a, LINS$ PROGRAMA := LINST; $PROGRAMA.a = programa\_sin\_decs\ (LINST.a)$ LDECS := DEC RDECS;	REAL_PARAMS := E0 RES_PARAMS; $RES\_PARAMS.ah$ =l_real_params_uno($E0.a$) $REAL\_PARAMS.a = RES\_PARAMS.a$ RES_PARAMS := **coma** E0 RES_PARAMS;

$RDECS.ah$ =decs_una($DEC.a$)
$DECS.a$ =$RDECS.a$
RDECS := **scol** DEC RDECS;
$RDECS_1.ah$ =decs_muchas($RDECS_0.ah, DEC.a$)
$RDECS_0.a$ =$RDECS_1.a$
RDECS := ƛ ;
$RDECS.a$ =$RDECS.ah$

DEC := **var** TIPO **id**;
$DEC.a$ =dec_var(TIPO.a, id.lex)
DEC := **type** TIPO **id;**
$DEC.a$ =dec_type(TIPO.a, id.lex)

DEC := **proc id pap** RES_DEC;
$RES\_DEC.ah$ =id.lex
$DEC.a$ = $RES\_DEC.a$
RES_DEC := LPARAMS **pcierre** BLOQUE;
$RES\_DEC.a$ =dec_proc_con_params($RES\_DEC.ah$, LPARAMS.a,BLOQUE.a)
RES_DEC := **pcierre** BLOQUE;
$RES\_DEC.a$ =dec_proc_sin_params($RES\_DEC.ah$,BLOQUE.a)

LPARAMS:=PARAM RLPARAMS;
$RLPARAMS.ah$ =l_params_uno($PARAM.a$)
$LPARAMS.a$ =$RLPARAMS.a$
RLPARAMS:= **coma** PARAM RLPARAMS;
$RLPARAMS_1.ah$ =l_params_muchos($RLPARAMS_0.ah$, $PARAM.a$)
$RLPARAMS_0.a$ =$RLPARAMS_1.a$
RLPARAMS:= ƛ ;
$RLPARAMS\ .a$ =$RLPARAMS\ .ah$

PARAM := TIPO RPARAM;
$RPARAM.ah$ = $TIPO.a$
$PARAM.a$ = $RPARAM.a$
RPARAM := **id**;
$RPARAM.a$ =param_sin_amp($RPARAM.ah, id.lex$)
RPARAM := **amp id**;
$RPARAM.a$ =param_con_amp($RPARAM.ah, id.lex$)
TIPO := **r_bool;**
$TIPO.a$ = tipo_Bool()
TIPO := **r_real;**
$TIPO.a$ = tipo_Real()

$RES\_PARAMS_1.ah$ =l_real_params_muchos($RES\_PARAMS_0.ah$, $E0.a$)
$RES\_PARAMS_0.a$ = $RES\_PARAMS_1.a$
RES_PARAMS := ƛ ;
$RES\_PARAMS.a$ =$RES\_PARAMS.ah$

INST := BLOQUE;
$INST.a$ =inst_compuesta($BLOQUE.a$)

BLOQUE := **llap** RBLOQUE;
$BLOQUE.a$ = $RBLOQUE.a$
RBLOQUE := **llcierre**;
$RBLOQUE.a$ =bloque_vacio()
RBLOQUE := PROGRAMA **llcierre**;
$RBLOQUE.a$ =bloque(PROGRAMA.a)

E0 := E1 RES0;
$RES0.ah$ = $E1.a$
$E0.a$ = $RES0.a$
RES0 := **mas** E0;
$RES0.a$ = exp_binaria("+",$RES0.ah$, $E0.a$)
RES0:= **menos** E1;
$RES0.a$ = exp_binaria("-",$RES0.ah$, $E1.a$)
RES0 := ƛ ;
$RES0.a$ =$RES0.ah$

E1 := E2 RES1;
$RES1.ah$ = $E2.a$
$E1.a$ = $RES1.a$
RES1 := OP1AI E2 RES1;
$RES1_1.ah$ = $exp\_binaria(OP1AI.op, RES1_0.ah$
$RES1_0.a$ = $RES1_1.a$
RES1 := ƛ ;
$RES1.a$ = $RES1.ah$

E2 := E3 RES2;
$RES2.ah$ = $E3.a$
$E2.a$ = $RES2.a$
RES2 := OP2AI E3 RES2;
$RES2_1.ah$ = $exp\_binaria(OP2AI.op, RES2_0.a$
$RES2_0.a$ = $RES2_1.a$
RES2:= ƛ ;
$RES2.a$ = $RES2.ah$

TIPO := **r_int ;**

$TIPO.a = tipo\_Entero()$

TIPO := **r_string ;**

$TIPO.a = tipo\_String()$

TIPO := **array cap ent ccierre of** TIPO;

$TIPO_0.a = tipo\_Array(ent.lex, TIPO_1.a)$

TIPO := **record llap** CAMPOS **llcierre**;

$TIPO.a = tipo\_Reg(CAMPOS.a)$

TIPO := **pointer** TIPO**;**

$TIPO_0.a = Tipo\_Puntero(TIPO_1.a)$

TIPO := **id**;

$TIPO.a = tipo\_Id(id.lex)$

CAMPOS:= CAMPO RCAMPOS;

$RCAMPOS.ah = l\_campos\_uno(CAMPO.a)$

$CAMPOS.a = RCAMPOS.a$

RCAMPOS:= **scol** CAMPO RCAMPOS;

$RCAMPOS_1.ah = l\_campos\_muchos($

$RCAMPOS_0.ah, CAMPO.a)$

$RCAMPOS_0.a = RCAMPOS_1.a$

RCAMPOS:= ƛ ;

$RCAMPOS.a = RCAMPOS_0.ah$

CAMPO := TIPO **id**;

$CAMPO.a = campo(TIPO.a, id.lex)$

LINST := INST RLINST;

$RLINST.ah = l\_inst\_una(INST.a)$

$LINST.a = RLINST.a$

RLINST := **scol** INST RLINST;

$RLINST_1.ah = l\_inst\_muchas(RLINST_0.ah, INST.a)$

$RLINST_0.a = RLINST_1.a$

RLINST := ƛ ;

$RLINST.a = RLINST.ah$

INST := E0 **igual** E0;

$INST.a = inst\_asig(E0_0.a, E0_1.a)$

INST := **if** E0 **then** AUX_LINST RES_IF;

$RES\_IF.ah1 = E0.a$

$RES\_IF.ah2 = AUX\_LINST.ah$

$INST.a = RES\_IF.a$

RES_IF := **else** AUX_LINST **endif** ;

E3 := E4 RES3;

$RES3.ah = E4.a$

$E3.a = RES3.a$

RES3 := OP3NA E4;

$RES3.a = exp\_binaria(OP3NA.op, RES3.ah, E4.a)$

RES3:=ƛ ;

$RE3.a = RES3.ah$

E4 := **menos** E5;

$E4.a = menos\_unario(E5.a)$

E4 := **not** E4;

$E4_0.a = c\_not(E4_1.a)$

E4 := E5;

$E4.a = E5.a$

RES5 := **cap** E0 **ccierre**;

$RES5.a = index(RES5.ah, E0.a)$

RES5 := **flecha id**;

$RES5.a = flecha(RES5.ah, id.lex)$

RES5 := **punto id**;

$RES5.a = punto(RES5.ah, id.lex)$

E5 := E6 RES_RES5;

$RES\_RES5.ah = E6.a$

$E5.a = RES\_RES5.a$

RES_RES5 := RES5 RES_RES5 ;

$RES5.ah = RES\_RES5_0.ah$

$RES\_RES5_1.ah = RES5.a$

$RES\_RES5_0.a = RES\_RES5_1.a$

RES_RES5 := ƛ ;

$RES\_RES5.a = RES\_RES5.ah$

E6 := **por** E6

$E6_0.a = indireccion(E6_1.a)$

E6 := E7;

$E6.a = E7.a$

E7 := **pap** E0 **pcierre**;

$E7.a = E0.a$

E7 :=**null**;

$E7.a = c\_null()$

E7 := **id**;

$E7.a = identificador(id.lex)$

E7 := **false**;

$RES\_IF.a$ =inst_if_then_else(
$RES\_IF.ah1, RES\_IF.ah2, AUXLINST.a)$
RES_IF := **endif** ;
$RES\_IF.a$ =inst_if_then($RES\_IF.ah1, RES\_IF.ah2$)

INST := **while** E0 **do** AUX_LINST **endwhile**;
$INST.a$ =inst_while($E0.a, AUX\_LINST.a$)
AUX_LINST := LINST;
$AUX\_LINST.a$ =linst_aux($LINST.a$)
AUX_LINST := **ƛ** ;
$AUX\_LINST.a$ =linst_aux_vacia()
INST:= **read** E0;
$INST.a$ =inst_read($E0.a$)
INST := **write** E0;
$INST.a$ =inst_write($E0.a$)
INST := **nl**;
$INST.a$ =inst_nl()
INST := **new** E0;
$INST.a$ =inst_new($E0.a$)
INST := **delete** E0;
$INST.a$ =inst_delete($E0.a$)

INST := **call id pap** RES_CALL;
$RES\_CALL.ah$ =id.lex
$INST.a = RES\_CALL.a$
RES_CALL := **pcierre**;
$RES\_CALL.a$ =inst_call_sin_params($RES\_CALL.ah$)
RES_CALL := REAL_PARAMS **pcierre**;
$RES\_CALL.a$ =inst_call_con_params(
$RES\_CALL.ah, REAL\_PARAMS.a$)

$E7.a$ = c_false()
E7 := **true**;
$E7.a$ = c_true()
E7 := **cadena**;
$E7.a$ = c_str(cadena.lex)
E7 := **real**;
$E7.a$ = numReal(real.lex)
E7 := **ent**;
$E7.a$ = numEnt(ent.lex)
OP3NA := **por**;
$OP3NA.op$ = " * "
OP3NA := **div**;
$OP3NA.op$ = "/"
OP3NA := **mod**;
$OP3NA.op$ = "%"
OP2AI := **bne**;
$OP2AI.op$ = "! = "
OP2AI := **beq**;
$OP2AI.op$ = " == "
OP2AI := **ble**;
$OP2AI.op$ = " <= "
OP2AI := **bge**;
$OP2AI.op$ = " >= "
OP2AI := **blt**;
$OP2AI.op$ = " < "
OP2AI :=**bgt**;
$OP2AI.op$ = " > "
OP1AI := **and**;
$OP1AI.op$ = "and"
OP1AI := **or**;
$OP1AI.op$ = "or"