Desarrollo de constructores de ASTs para Tiny(1)

Óscar Morujo Fernández

Sofía Capmany Fernández

Especificación sintáctica:

```
PROGRAMA := LDECS sep di LINST;
                                                 REAL PARAMS := REAL PARAMS coma
PROGRAMA := LINST:
LDECS := LDECS scol DEC;
                                                 REAL PARAMS := E0;
LDECS := DEC;
                                                 INST := BLOQUE:
DEC := var TIPO id;
                                                 BLOQUE := Ilap PROGRAMA Ilcierre;
DEC := type TIPO id;
                                                 BLOQUE := Ilap Ilcierre;
DEC := proc id pap LPARAMS pcierre BLOQUE;
                                                 E0 := E1 \text{ mas } E0;
DEC := proc id pap pcierre BLOQUE;
                                                 E0 := E1 menos E1;
LPARAMS := LPARAMS coma PARAM;
                                                 E0 := E1;
LPARAMS := PARAM;
                                                 E1 := E1 OP1AI E2;
PARAM := TIPO amp id;
                                                 E1 := E2;
PARAM := TIPO id;
                                                 E2 := E2 OP2AI E3;
TIPO := r bool;
                                                 E2 := E3;
TIPO := r_real;
                                                 E3 := E4 OP3NA E4;
TIPO := r int;
                                                 E3 := E4;
TIPO := r_string;
                                                 E4 := menos E5;
TIPO := array cap ent ccierre of TIPO;
                                                 E4 := not E4:
TIPO := record llap CAMPOS llcierre;
                                                 E4 := E5;
TIPO := pointer TIPO;
                                                 E5 := E5 cap E0 ccierre;
TIPO := id;
                                                 E5 := E5 flecha id;
CAMPOS := CAMPOS scol CAMPO;
                                                 E5 := E5 punto id;
CAMPOS := CAMPO;
                                                 E5 := E6;
                                                 E6 := por E6
CAMPO := TIPO id:
LINST := LINST scol INST;
                                                 E6 := E7;
                                                 E7 := pap E0 pcierre;
LINST := INST;
INST := E0 iqual E0;
                                                 E7 :=null;
INST := if E0 then AUX LINST endif;
                                                 E7 := id;
INST := if E0 then AUX_LINST else AUX_LINST
                                                 E7 := false;
endif:
                                                 E7 := true;
INST := while E0 do AUX_LINST endwhile;
                                                 E7 := cadena;
AUX LINST := LINST;
                                                 E7 := real;
AUX LINST := X ;
                                                 E7 := ent:
INST:= read E0;
                                                 OP3NA := por;
INST := write E0;
                                                 OP3NA := div;
INST := nI:
                                                 OP3NA := mod:
INST := new E0;
                                                 OP2AI := bne;
INST := delete E0;
                                                 OP2AI := beq;
INST := call id pap REAL PARAMS pcierre;
                                                 OP2AI := ble;
INST := call id pap pcierre;
                                                 OP2AI := bge;
                                                 OP2AI := blt;
                                                 OP2Al :=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 -> LDecsdec_var:Tipo x String -> Decdec_type:Tipo x String -> Dec

dec_proc_con_params: String x LParams x Bloque -> Dec

dec_proc_sin_params: String x Bloque -> Dec

I_params_uno: Param -> Lparams

I_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_ld: String -> Tipo
tipo_Puntero: Tipo -> Tipo

tipo_Array:String x Tipo -> Tipotipo_Reg:LCampos -> TipoI_campos_uno:Campo -> LCampos

I_campos_muchos: LCampos x Campo ->LCampos

campo:Tipo x string -> CampoI_inst_muchas:Linst x Inst -> Linst

I_inst_una: Inst -> LInst
inst_asig: Exp x Exp -> Inst
inst_if_then: Exp x LInst_aux -> Inst

inst_if_then_else:
Exp x Llnst_aux x Llnst_aux ->Inst

linst_aux_vacia: -> Llnst_aux

linst_aux:LInst -> LInst_auxinst_while:Exp x LInst_aux -> Inst

inst_new: Exp -> Inst
inst_delete: Exp -> Inst
inst_call_sin_params: String -> Inst

inst_call_con_params: String x LReal_params -> Inst

I_real_params_uno:
Exp -> LReal_params

I_real_params_muchos: LReal_params x Exp -> LReal_params

inst_compuesta:Bloque -> Instbloque :Prog -> Bloque

bloque_vacio: ->Bloque

Exp x Exp -> Exp suma: Exp x Exp -> Exp resta: mul: Exp x Exp -> Exp div: Exp x Exp -> Exp beq: Exp x Exp -> Exp Exp x Exp -> Exp bne: Exp x Exp -> Exp ble: bge: Exp x Exp -> Exp blt: Exp x Exp -> Exp Exp x Exp -> Exp bgt: c_and: Exp x Exp -> Exp Exp x Exp -> Exp c_or: 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_ld(id.lex)
CAMPOS := CAMPOS scol CAMPO;
    CAMPOS_0. a = I_{campos_muchos(CAMPOS_1, a, CAMPO, a)}
CAMPOS := CAMPO;
    CAMPOS. \alpha= I_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 := X ;
    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 := nI;
    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 = | real\_params\_uno(E0.\ a)
INST := BLOQUE;
    INST. a =inst_compuesta(BLOQUE. a)
BLOQUE := Ilap PROGRAMA Ilcierre;
```

```
BLOQUE. a = bloque(PROGRAMA. a)
BLOQUE := Ilap Ilcierre;
     BLOQUE. a =bloque_vacio()
E0 := E1 \text{ mas } E0;
     E0_{0}. a = \exp_{0} sinaria("+",E1. a, E0_{1}. a)
E0 := E1 \text{ menos } E1;
     E0. a = \exp_{\text{binaria}(\text{"-"},E1_{0}.a,E1_{1}.a)}
E0 := E1;
     E0.a = E1.a
E1 := E1 OP1AI E2;
     E1_{0}. a = \exp_{\text{binaria}}(\text{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_{\text{binaria}}(\text{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<sub>0</sub>. a = c_not(E4<sub>1</sub>. 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 = " == "
OP2Al := ble;
    OP2AI.op = " <= "
OP2AI := bge;
    OP2AI.op = ">= "
OP2AI := bIt;
    OP2AI.op = " < "
OP2AI :=bqt;
    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)
                              return resta(Arg0,Arg1)
               case '-':
               case '*':
                              return mul(Arg0,Arg1)
               case '/':
                              return div(Arg0,Arg1)
                              return mod(Arg0,Arg1)
               case '%':
               case '==':
                              return beq(Arg0,Arg1)
                              return ble(Arg0,Arg1)
               case '<=':
               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 := proc id pap RES_DEC;
DEC. a = dec_proc_con_params(id.lex,
                                               RES_DEC. ah = id.lex
LPARAMS.a,BLOQUE.a)
                                               DEC.a = RES_DEC.a
DEC := proc id pap pcierre BLOQUE;
                                               RES DEC := LPARAMS pcierre BLOQUE;
DEC. a =dec_proc_sin_params(id.lex,BLOQUE.a)
                                               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 := TIPO RPARAM;
PARAM.a = param_con_amp(TIPO.a,id.lex)
                                               RPARAM.ah = TIPO.a
PARAM := TIPO id;
                                               PARAM.a = RPARAM.a
PARAM. a = param_sin_amp(TIPO.a,id.lex)
                                               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 := Ilap PROGRAMA Ilcierre; BLOQUE. a =bloque(PROGRAMA. a) BLOQUE := Ilap Ilcierre; BLOQUE. a =bloque_vacio()	BLOQUE := Ilap RBLOQUE; BLOQUE.a = RBLOQUE.a RBLOQUE := Ilcierre; RBLOQUE.a = bloque_vacio() RBLOQUE := PROGRAMA Ilcierre; 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 := \hbar ; RES0. a = RES0. ah

```
E3 := E4 OP3NA E4;
                                                        E3 := E4 RES3:
E3. a = \exp_{\text{o}} \cdot a, E4 <sub>1</sub>. a)
                                                        RES3.ah = E4.a
                                                        E3.a = RES3.a
E3 := E4;
                                                        RES3 := OP3NA E4;
E3.a = E4.a
                                                        RES3. a = \exp_{\text{binaria}}(\text{OP3NA.op}, RES3. ah,
                                                        E4. a)
                                                        RES3:=1;
                                                        RE3. a = RES3. ah
E5 := E5 cap E0 ccierre;
                                                        E5 := E5 RES5;
E5_0. a = index(E5_1. a, E0. a)
                                                        RES5. ah = E5_1. a
E5 := E5 flecha id;
                                                        E5_0. a = RES5. a
E5_0. a = flecha(E5_1. a, id.lex)
                                                        E5 := E6:
E5 := E5 punto id;
                                                        E5. a = E6. a
E5_0. a = punto(E5_1. a,id.lex)
                                                        RES5 := cap E0 ccierre;
                                                        RES5. a = index(RES5. ah, E0. a)
E5 := E6;
                                                        RES5 := flecha id;
E5. a = E6. a
                                                        RES5. a = flecha(RES5. ah, id. lex)
                                                        RES5 := punto id;
                                                        RES5. a = punto(RES5. ah, id. lex)
```

<u>Segundo paso : Eliminar recursión a izquierdas</u>

```
LDECS := LDECS scol DEC;
                                            LDECS := DEC RDECS:
LDECS_0. a = decs_muchas(LDECS_1, a, DEC, a)
                                            RDECS. ah = decs_una(DEC. a)
                                            DECS. a = RDECS. a
LDECS := DEC;
                                            RDECS := scol DEC RDECS;
LDECS. a = decs_una(DEC. a)
                                            RDECS_1. ah = decs_muchas(RDECS_0. ah, DEC. a)
                                            RDECS_0. a = RDECS_1. a
                                            RDECS := 1 ;
                                            RDECS. a = RDECS. ah
LPARAMS := LPARAMS coma PARAM;
                                            LPARAMS:=PARAM RLPARAMS;
LPARAMS_0. a = l_params_muchos(
                                            RLPARAMS. ah = 1 params uno(PARAM. a)
                                            LPARAMS.a = RLPARAMS.a
LPARAMS_1. a, PARAM. a)
                                            RLPARAMS:= coma PARAM RLPARAMS;
LPARAMS := PARAM;
                                            RLPARAMS_1. ah = I_params_muchos(
LPARAMS. a = l_params_uno(PARAM. a)
```

CAMPOS := CAMPOS scol CAMPO; CAMPOS ₀ . a= l_campos_muchos(CAMPOS ₁ . a, CAMPO. a) CAMPOS := CAMPO; CAMPOS. a= l_campos_uno(CAMPO.a)	$RLPARAMS_{0}$. ah , $PARAM$. a) $RLPARAMS_{0}$. $a = RLPARAMS_{1}$. a $RLPARAMS:= \hbar$; $RLPARAMS:= \hbar$; $RLPARAMS$. $a = RLPARAMS$. ah $CAMPOS:= CAMPO RCAMPOS;$ $RCAMPOS$. $ah = I_{campos_{uno}(CAMPO.a)}$ $CAMPOS$. $a = RCAMPOS$. a $RCAMPOS:= scol CAMPO RCAMPOS;$ $RCAMPOS_{1}$. $ah = I_{campos_{muchos}(RCAMPOS_{0})$. ah , $CAMPO$. a) $RCAMPOS_{0}$. ah , $CAMPO$. a) $RCAMPOS_{0}$. ah , $CAMPOS_{1}$. ah $RCAMPOS:= \hbar$; $RCAMPOS:= \hbar$; $RCAMPOS$. ah
LINST := LINST scol INST; LINST ₀ . a = I_inst_muchas(LINS ₁ . a, INST. a) LINST := INST; LINST. a = I_inst_una(INST.a)	LINST := INST RLINST; $RLINST. ah = l_inst_una(INST.a)$ $LINST. a = RLINST.a$ $RLINST := scol INST RLINST;$ $RLINST_1. ah = l_inst_muchas(RLINST_0. ah, INST.a)$ $RLINST_0. a = RLINST_1. a$ $RLINST := \hbar$; $RLINST . a = RLINST . ah$
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; $RES_PARAMS. ah = l_real_params_uno(E0. a)$ $REAL_PARAMS. a = RES_PARAMS. a$ $RES_PARAMS := coma E0 RES_PARAMS;$ $RES_PARAMS_1. ah = l_real_params_muchos($ $RES_PARAMS_0. ah, E0. a)$ $RES_PARAMS_0. a = RES_PARAMS_1. a$ $RES_PARAMS := \hbar;$ $RES_PARAMS := \hbar;$ $RES_PARAMS. a = RES_PARAMS. ah$
E1 := E1 OP1Al E2; $E1_{0}$. $a = \exp_{\text{binaria}}(\text{OP1Al.op}, E1_{1}. a, E2. a)$ 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 := λ :
                                                 RES1.a = RES1.ah
E2 := E2 OP2AI E3:
                                                 E2 := E3 RES2:
                                                 RES2.ah = E3.a
E2_{0}. a = \exp_{\text{binaria}}(\text{OP2AI.op}, E2_{1}. a, E3. a)
                                                 E2.a = RES2.a
E2 := E3:
                                                 RES2 := OP2AI E3 RES2:
E2.a = E3.a
                                                 RES2_1. ah = exp\_binaria(OP2AI.op, RES2_0.ah, E3.a)
                                                 RES2_0. a = RES2_1. a
                                                 RES2:= ス;
                                                 RES2.a = RES2.ah
                                                 E5 := E6 RES RES5;
E5 := E5 RES5;
                                                 RES_RES5.ah = E6.a
RES5. ah = E5_1. a
                                                 E5. a = RES\_RES5. a
E5_0. a = RES5. a
                                                 RES RES5 := RES5 RES RES5 ;
E5 := E6;
                                                 RES5. ah = RES\_RES5_0. ah
E5. a = E6. a
                                                 RES_RES5_1. ah = RES5. a
RES5 := cap E0 ccierre;
RES5. a = index(RES5. ah, E0. a)
                                                 RES\_RES5_0. a = RES\_RES5_1. a
RES5 := flecha id;
                                                 RES_RES5 := \bar{\lambda};
RES5. a = flecha(RES5. ah, id. lex)
                                                 RES_RES5.a = RES_RES5.ah
RES5 := punto id;
RES5. a = punto(RES5. ah, id. lex)
```

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)

REAL_PARAMS := E0 RES_PARAMS;

RES_PARAMS. ah = | real_params_uno(E0. a | )

REAL_PARAMS. a = RES_PARAMS. a

RES_PARAMS := coma E0 RES_PARAMS;

LDECS := DEC RDECS;
```

RDECS. ah = decs una(DEC. a)RES_PARAMS_.ah = | real_params_muchos() DECS.a = RDECS.aRES_PARAMS₀. ah, E0. a) RDECS := scol DEC RDECS; RES_PARAMS_0 . $a = RES_PARAMS_1$. a $RDECS_1$. $ah = decs_muchas(RDECS_0, ah, DEC. a)$ RES PARAMS := 1 : $RDECS_0$. $a = RDECS_1$. a $RES_PARAMS.$ $a = RES_PARAMS.$ ahRDECS := 1\lambda : RDECS, a = RDECS, ahINST := BLOQUE; INST. a = inst compuesta(BLOQUE. a)DEC := var TIPO id; DEC. a = dec var(TIPO.a, id.lex)BLOQUE := Ilap RBLOQUE; DEC := type TIPO id; BLOOUE. a = RBLOOUE. a $DEC. a = dec_type(TIPO.a, id.lex)$ RBLOQUE := Ilcierre; RBLOQUE.a =bloque vacio() DEC := proc id pap RES DEC; RBLOQUE := PROGRAMA Ilcierre; $RES_DEC. ah = id.lex$ RBLOQUE.a =bloque(PROGRAMA.a) DEC.a = RES DEC.aRES_DEC := LPARAMS pcierre BLOQUE; E0 := E1 RES0: $RES_DEC. a = dec_proc_con_params(RES_DEC. ah,$ RES0. ah = E1. aLPARAMS.a, BLOQUE.a) E0.a = RES0.aRES DEC := pcierre BLOQUE; RES0 := mas E0; RES_DEC. a =dec_proc_sin_params(RES_DEC. ah RES0. $a = \exp_{\text{binaria}}("+", RES0. ah, E0. a)$,BLOQUE.a) RES0:= menos E1; RES0. $a = \exp_{\text{binaria}}(\text{"-"}, RES0. ah, E1. a)$ LPARAMS:=PARAM RLPARAMS; RES0 := λ : $RLPARAMS. ah = l_params_uno(PARAM. a)$ RES0. a = RES0. ahLPARAMS. a = RLPARAMS. aRLPARAMS:= coma PARAM RLPARAMS; E1 := E2 RES1; $RLPARAMS_1$. $ah = l_params_muchos($ RES1. ah = E2. aRLPARAMS ah, PARAM a) E1.a = RES1.aRES1 := OP1AI E2 RES1; $RLPARAMS_0$. $a = RLPARAMS_1$. a $RES1_1$. $ah = exp_binaria(OP1AI.op, RES1_2.ah)$ RLPARAMS:= 1 ; $RES1_0$. $a = RES1_1$. aRLPARAMS . a = RLPARAMS . ahRES1 := λ ; RES1.a = RES1.ahPARAM := TIPO RPARAM; RPARAM.ah = TIPO.aE2 := E3 RES2; PARAM.a = RPARAM.aRES2. ah = E3. aRPARAM := id;E2. a = RES2. a $RPARAM. a = param_sin_amp(RPARAM. ah, id. lex)$ RES2 := OP2AI E3 RES2; RPARAM := amp id; $RES2_1$. $ah = exp_binaria(OP2AI.op, RES2_0.a)$ $RPARAM. a = param_con_amp(RPARAM. ah, id. lex)$ TIPO := r bool; $RES2_0$. $a = RES2_1$. aTIPO.a = tipo Bool()RES2:= λ ; $TIPO := r_real;$ RES2. a = RES2. ahTIPO.a = tipo Real()

```
E3 := E4 RES3:
TIPO := r int;
TIPO. a = tipo_Entero()
                                                        RES3.ah = E4.a
TIPO := r_string;
                                                        E3.a = RES3.a
TIPO. a = tipo_String()
                                                        RES3 := OP3NA E4:
TIPO := array cap ent ccierre of TIPO;
                                                        RES3. a = \exp_{\text{binaria}}(\text{OP3NA.op}, RES3. ah,
TIPO_0. a = tipo_Array(ent. lex, TIPO_1. a)
                                                        E4.a)
                                                        RES3:=λ :
TIPO := record llap CAMPOS llcierre;
                                                        RE3. a = RES3. ah
TIPO.a = tipo Reg(CAMPOS.a)
TIPO := pointer TIPO;
                                                        E4 := menos E5:
TIPO_0. a = Tipo_Puntero(TIPO_1.a)
                                                        E4. a = menos_unario(E5.a)
TIPO := id:
                                                        E4 := not E4;
TIPO.a = tipo Id(id.lex)
                                                        E4 <sub>0</sub>. a = c_not(E4 <sub>1</sub>. a)
CAMPOS:= CAMPO RCAMPOS;
                                                        E4 := E5:
RCAMPOS . ah = I_campos_uno(CAMPO. a)
                                                        E4. a = E5. a
CAMPOS \cdot a = RCAMPOS \cdot a
RCAMPOS:= scol CAMPO RCAMPOS;
                                                        RES5 := cap E0 ccierre;
RCAMPOS<sub>1</sub>. ah= l_campos_muchos(
                                                        RES5. a = index(RES5. ah, E0. a)
                                                        RES5 := flecha id;
RCAMPOS<sub>0</sub>. ah, CAMPO. a)
                                                        RES5. \alpha = flecha(RES5. ah, id. lex)
RCAMPOS_0. a = RCAMPOS_1. a
                                                        RES5 := punto id;
RCAMPOS:= 1 ;
                                                        RES5. a = punto(RES5. ah, id. lex)
RCAMPOS \cdot a = RCAMPOS_0 \cdot ah
                                                        E5 := E6 RES RES5:
                                                        RES RES5. ah = E6. a
CAMPO := TIPO id;
                                                        E5. a = RES_RES5. a
CAMPO.a = campo(TIPO.a,id.lex)
                                                        RES_RES5 := RES5 RES_RES5 ;
                                                        RES5. ah = RES_RES5_0. ah
LINST := INST RLINST;
RLINST. ah = l inst una(INST.a)
                                                        RES_RES5_1. ah = RES5. a
LINST.a = RLINST.a
                                                        RES\_RES5_0. a = RES\_RES5_1. a
RLINST := scol INST RLINST;
                                                        RES RES5 := 1 :
RLINST<sub>1</sub>. ah = | inst_muchas(RLINST<sub>0</sub>. ah, INST.a)
                                                        RES_RES5. a = RES_RES5. ah
RLINST_0. a = RLINST_1. a
RLINST := \lambda;
                                                        E6 := por E6
RLINST . a = RLINST . ah
                                                        E6_0. a = indireccion(E6_1. a)
INST := E0 igual E0;
                                                        E6 := E7:
INST. a = inst_asig(E0_0. a, E0_1. a)
                                                        E6. a = E7. a
                                                        E7 := pap E0 pcierre;
                                                        E7. a = E0. a
INST := if E0 then AUX LINST RES IF;
                                                        E7 :=null:
RES_IF. ah1 = E0. a
                                                        E7. a = c null()
RES\ IF.\ ah2\ =AUX\ LINST.\ ah
                                                        E7 := id;
INST.a = RES IF.a
                                                        E7. a = identificador(id.lex)
RES IF := else AUX LINST endif;
                                                        E7 := false:
```

```
RES_IF. a = inst_if_then_else(
                                                    E7.a = c_false()
RES_IF. ah1, RES_IF. ah2, AUXLINST. a)
                                                    E7 := true;
RES IF := endif;
                                                    E7. a = c true()
RES\_IF. a = inst if then(RES\_IF. ah1, RES\_IF. ah2)
                                                    E7 := cadena:
                                                    E7. a = c_str(cadena.lex)
INST := while E0 do AUX LINST endwhile;
                                                    E7 := real;
INST. a = inst while(E0. a, AUX\_LINST. a)
                                                    E7. a = numReal(real.lex)
AUX LINST := LINST;
                                                    E7 := ent:
AUX\_LINST.a = linst aux(LINST.a)
                                                    E7. a = numEnt(ent.lex)
AUX LINST := 1 ;
                                                    OP3NA := por;
                                                    OP3NA.op = "*"
AUX_LINST. a = linst_aux_vacia()
INST:= read E0;
                                                    OP3NA := div;
INST. a = inst read(E0. a)
                                                    OP3NA.op = "/"
INST := write E0;
                                                    OP3NA := mod;
INST. a = inst_write(E0. a)
                                                    OP3NA.op = "%"
INST := nI;
                                                    OP2AI := bne;
INST. a = inst nl()
                                                    OP2AI. op = "! = "
INST := new E0;
                                                    OP2AI := beq;
INST. a = inst_new(E0. a)
                                                    OP2AI.op = " == "
INST := delete E0;
                                                    OP2AI := ble;
                                                    OP2AI.op = " <= "
INST. a = inst delete(E0. a)
                                                    OP2Al := bge;
INST := call id pap RES_CALL;
                                                    OP2AI.op = ">= "
RES_CALL.ah = id.lex
                                                    OP2AI := blt;
INST. a = RES_CALL. a
                                                    OP2AI.op = " < "
RES_CALL := pcierre;
                                                    OP2Al :=bgt;
RES_CALL. a = inst call sin params(RES_CALL. ah)
                                                    OP2AI.op = " > "
RES CALL := REAL PARAMS pcierre;
                                                    OP1AI := and;
RES\_CALL. a = inst call con params(
                                                    OP1AI.op = "and"
RES_CALL. ah, REAL_PARAMS. a)
                                                    OP1AI := or;
                                                    OP1AI.op = "or"
```