TINY 0

Grupo 13
Gorka Silva Ramón
Sofía Capmany Fernández

ÍNDICE DE CONTENIDOS

1. ESPECIFICACIÓN SINTÁCTICA

PROG → LDEC **SEP_PROG** LINST
LDEC → LDEC **PTO_COMA** DEC
LDEC → DEC
DEC → TIPO **ID**

LINST → LINST **PTO_COMA** INST
LINST → INST
INST→ **ID ASIG** E0

E0 → E1 **MAS** E0
E0 → E1 **MENOS** E1
E0 → E1

E1 → E1 OPBAI1 E2
E1 → E2

E2 → E2 OPBAI2 E3
E2 → E3

E3 → E4 OPBNA E4
E3 → E4

E4 → **MENOS** E5
E4 → **R_NOT** E4
E4 → E5

E5 → **PAP** E0 **PCIERRE**
E5 → **R_TRUE**
E5 → **R_FALSE**
E5 → **ID**
E5 → **LIT_INT**
E5 → **LIT_REAL**

TIPO → **R_REAL**
TIPO → **R_INT**
TIPO → **R_BOOL**

OPBAI1 → **R_AND**
OPBAI1 → **R_OR**

OPBAI2 → **BLT**
OPBAI2 → **BGT**
OPBAI2 → **BLE**
OPBAI2 → **BGE**
OPBAI2 → **BEQ**
OPBAI2 → **BNE**

OPBNA → **POR**
OPBNA → **DIV**

2. SINTAXIS ABSTRACTA

2.1 Géneros o conceptos sintácticos

Prog, Ldecs, Dec, Linst, Inst, Tipo, Exp.

2.2 Constructoras

programa:	Ldecs x Linst → Prog
lista_dec_una:	Dec → Ldecs
lista_dec_muchas:	Ldecs x Dec → Ldecs
dec:	Tipo x String → Dec
lista_inst_una:	Inst → Linst
lista_inst_muchas:	Linst x Inst → Linst
inst:	String x Exp → Inst
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
and:	Exp x Exp → Exp
or:	Exp x Exp → Exp
not:	Exp → Exp
menos_unario:	Exp→ Exp
num_real:	String → Exp
num_int:	String → Exp
identificador:	String → Exp
r_false:	Exp
r_true:	Exp
r_int:	Tipo
r_real:	Tipo
r_bool:	Tipo

3. CONSTRUCTOR DE ÁRBOLES DE SINTAXIS ABSTRACTA

PROG := LDECS **sep_prog** LINST;
PROG.a = prog(LDECS.a, LINST.a)
LDECS := LDECS **pto_coma** DEC;
LDECS0.a = lista_dec_muchas ($LDECS_1$.a, DEC.a)
LDECS := DEC;
LDECS.a = lista_dec_una(DEC.a)
DEC := TIPO **id**;
DEC.a = dec(TIPO.a, id.lex)

LINST := LINST **pto_coma** INST;
$LINST_0$.a = lista_inst_muchas (LINST1.a, INST.a)
LINST := INST;
LINST.a = lista_inst_una (INST.a)
INST := **id asig** E0;
INST.a = inst(id.lex, E0.a)

E0 := E1 **mas** E0;
$E0_0$.a = exp("+",E1.a, E01.a)
E0 := E1 **menos** E1;
E0.a = exp("-",$E1_0$.a, $E1_1$.a)
E0 := E1;
E0.a = E1.a
E1 := E1 OPBAI1 E2;
$E1_0$.a = exp(OP1AI.op,$E1_1$.a, E2.a)
E1 := E2;
E1.a = E2.a
E2 := E2 OPBAI2 E3;
$E2_0$.a = exp(OP2AI.op,$E2_1$.a, E3.a)
E2 := E3;
E2.a = E3.a
E3 := E4 OPBNA E4;
E3.a = exp(OPBNA.op,$E4_0$.a, $E4_1$.a)
E3 := E4;
E3.a = E4.a
E4 := **menos** E5;
E4.a = menos_unario(E5.a)
E4 := **r_not** E4;
$E4_0$.a = not($E4_1$.a)
E4 := E5;
E4.a =E5.a
E5 := **lit_int**;
E5.a = num_int(lit_int.lex)
E5 := **lit_real**;
E5.a = num_real(lit_real.lex)
E5 := **id**;

E5.a = identificador(id.lex)

E5 := **r_true**;

E5.a = r_true()

E5 := **r_false**;

E5.a = r_false()

E5 := **pap** E0 **pcierre**;

E5.a = E0.a

OPBNA := **por**;

OPBNA.op = "*"

OPBNA := **div**;

OPBNA.op = "/"

OPBAI2:= **bne**;

OP2AI.op = "!="

OPBAI2:= **beq**;

OP2AI.op = "=="

OPBAI2:= **ble**;

OP2AI.op = "<="

OPBAI2:= **bge**;

OP2AI.op = ">="

OPBAI2:= **blt**;

OP2AI.op = "<"

OPBAI2:= **bgt**;

OP2AI.op = ">"

OPBAI1:= **and**;

OP1AI.op = "and"

OPBAI1:= **or**;

OP1AI.op = "or"

TIPO := **r_int**;

TIPO.a = tipo_Entero()

TIPO := **r_bool**;

TIPO.a = tipo_Bool()

TIPO := **r_real**;

TIPO.a = tipo_Real()

Funciones semánticas

```
fun exp(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 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)
}
```

4. ACONDICIONAMIENTO PARA LA IMPLEMENTACIÓN DESCENDENTE

4.1 Factorizar

Sobre esta gramática factorizada hay que transformar también las correspondientes ecuaciones semánticas, de tal manera que se asegure la equivalencia semántica, aparte de la equivalencia sintáctica. Los cambios para la factorización de la grmática son los siguientes:

Gramática sin factorizar	Gramática factorizada
E0 := E1 **mas** E0; $E0_0.a = exp("+",E1.a, E01.a)$ E0 := E1 **menos** E1; $E0.a = exp("-",E1_0.a, E1_1.a)$ E0 := E1; $E0.a = E1.a$	E0 := E1 RE0; $RE0.ah = E1.a$ $E0.a = RE0.a$ RE0 := **mas** E0; $RE0.a = exp("+", RE0.ah, E0.a)$ RE0 := **menos** E1 $RE0.a = exp("-", RE0.ah, E1.a)$ RE0 := **ƛ** $RES0.a = RES0.ah$
E3 := E4 OPBNA E4; $E3.a = exp(OPBNA.op,E4_0.a, E4_1.a)$ E3 := E4; $E3.a = E4.a$	E3 := E4 RE3 $RE3.ah = E4.a$ $E3.a = RE3.a$ RE3 := OPBNA E4 RE3 $RE3.a=exp(OP3NA.op, RE3.ah, E4.a)$ RE3 := **ƛ** $RE3.a = RE3.ah$

4.2 Eliminar recursión a izquierdas

Con recursión a izquierdas	Sin recursión a izquierdas
LDECS := LDECS **pto_coma** DEC; LDECS0.a = lista_dec_muchas (LDECS$_1$.a, DEC.a) LDECS := DEC; LDECS.a = lista_dec_una(DEC.a)	LDEC := DEC RLDEC; RLDEC.ah =lista_dec_una(DEC.a) LDEC.a = RLDEC.a RLDEC := **pto_coma** DEC RLDEC; RLDEC$_1$.ah=lista_dec_muchas (RLDEC$_0$.ah,DEC.a) RLDEC$_0$.a =RLDEC$_1$.a RLDEC := **ƛ**; RLDEC.a = RLDEC.ah
LINST := LINST **pto_coma** INST; LINST$_0$.a = lista_inst_muchas (LINST1.a, INST.a) LINST := INST; LINST.a = lista_inst_una(INST.a)	LINST := INST RLINST; RLINST.ah =lista_inst_una(INST.a) LINST.a = RLINST.a RLINST := **pto_coma** INST RLINST; RLINST$_1$.ah = lista_inst_muchas (RLINST$_0$.ah, INST.a) RLINST$_0$.a = RLINST$_1$.a RLINST := **ƛ**; RLINST.a = RLINST.ah
E1 := E1 OPBAI1 E2; E1$_0$.a = exp(OP1AI.op, E1$_1$.a, E2.a) E1 := E2; E1.a = E2.a	E1 := E2 RE1; RE1.ah = E2.a E1.a =RE1.a RE1 := OPBAI1 E2 RE1; RE1$_1$.ah=exp(OP1AI.op, RE1$_0$.ah, E2.a) RE1 := **ƛ**; RE1.a = RE1.ah
E2 := E2 OPBAI2 E3; E2$_0$.a = exp(OP2AI.op, E2$_1$.a, E3.a) E2 := E3; E2.a = E3.a	E2 := E3 RE2; RE2.ah = E3.a E2.a =RE2.a RE2 := OPBAI2 E3 RE2; RE2$_1$.ah=exp(OP1AI.op, RE2$_0$.ah, E3.a) RE2 := **ƛ;** RE2.a = RE2.ah

4.3 Gramática acondicionada

PROG := LDECS **sep_prog** LINST;
PROG.a = prog(LDECS.a, LINST.a)
LDEC := DEC RLDEC;
RLDEC.ah =lista_dec_una(DEC.a)
LDEC.a = RLDEC.a
RLDEC := **pto_coma** DEC RLDEC;
$RLDEC_1$.ah=lista_dec_muchas ($RLDEC_0$.ah,DEC.a)
$RLDEC_0$.a =$RLDEC_1$.a
RLDEC := **ƛ**;
RLDEC.a = RLDEC.ah
DEC := TIPO **id**;
DEC.a = dec(TIPO.a, id.lex)

LINST := INST RLINST;
RLINST.ah =lista_inst_una(INST.a)
LINST.a = RLINST.a
RLINST := **pto_coma** INST RLINST;
$RLINST_1$.ah = lista_inst_muchas ($RLINST_0$.ah, INST.a)
$RLINST_0$.a = $RLINST_1$.a
RLINST := **ƛ**;
RLINST.a = RLINST.ah
INST := **id asig** E0;
INST.a = inst(id.lex, E0.a)

E0 := E1 RE0;
RE0.ah = E1.a
E0.a = RE0.a
RE0 := **mas** E0;
RE0.a = exp("+", RE0.ah, E0.a)
RE0 := **menos** E1
RE0.a = exp("-", RE0.ah, E1.a)
RE0 := **ƛ**
RES0.a = RES0.ah

E1 := E2 RE1;
RE1.ah = E2.a
E1.a =RE1.a
RE1 := OPBAI1 E2 RE1;
$RE1_1$.ah=exp(OP1AI.op, $RE1_0$.ah, E2.a)
RE1 := **ƛ**;
RE1.a = RE1.ah

E2 := E3 RE2;

RE2.ah = E3.a
E2.a =RE2.a

RE2 := OPBAI2 E3 RE2;
$RE2_1$.ah=exp(OP1AI.op, $RE2_0$.ah, E3.a)

RE2 := ƛ;
RE2.a = RE2.ah

E3 := E4 RE3
RE3.ah = E4.a
E3.a =RE3.a

RE3 := OPBNA E4 RE3
RE3.a=exp(OP3NA.op, RE3.ah, E4.a)

RE3 := ƛ
RE3.a =RE3.ah

E4 := **menos** E5;
E4.a = menos_unario(E5.a)

E4 := **r_not** E4;
$E4_0$.a = not($E4_1$.a)

E4 := E5;
E4.a =E5.a

E5 := **lit_int**;
E5.a = num_int(lit_int.lex)

E5 := **lit_real**;
E5.a = num_real(lit_real.lex)

E5 := **id**;
E5.a = identificador(id.lex)

E5 := **r_true**;
E5.a = r_true()

E5 := **r_false**;
E5.a = r_false()

E5 := **pap** E0 **pcierre**;
E5.a = E0.a

OPBNA := **por**;
OPBNA.op = "*"

OPBNA := **div**;
OPBNA.op = "/"

OPBAI2:= **bne**;
OP2AI.op = "!="

OPBAI2:= **beq**;
OP2AI.op = "=="

OPBAI2:= **ble**;
OP2AI.op = "<="

OPBAI2:= **bge**;
OP2AI.op = ">="

OPBAI2:= **blt**;

OP2AI.op = "<"

OPBAI2:=**bgt**;

OP2AI.op = ">"

OPBAI1:= **and**;

OP1AI.op = "and"

OPBAI1:= **or**;

OP1AI.op = "or"

TIPO := **t_int**;

TIPO.a = tipo_Entero()

TIPO := **t_bool**;

TIPO.a = tipo_Bool()

TIPO := **t_real**;

TIPO.a = tipo_Real()