Práctica
Procesadores de Lenguajes

David Antuña Rodríguez
Javier Carrión García

Contenidos

1 Fase 1

1.1 Clases léxicas

La descripción de las clases léxicas identificadas se hará de manera informal, en lenguaje natural.

- **SPROG**

  Es el separador && que indica el fin de la sección de declaraciones y el comienzo de la de instrucciones.

- **LREAL**

  Empiezan con un signo (+ o -) opcional, a continuación aparecen uno o más dígitos cualesquiera. Seguida de esta parte puede aparecer una decimal que consta de un punto seguido de uno o más dígitos cualesquiera. Por último, tiene una E o e seguida de un signo (+ o -), opcional, y de uno o más dígitos cualesquiera.

- **ID**

  Comienza por una letra cualquiera y la sigue una secuencia de cero o más letras, dígitos o subrayado(_).

- **BOOL**

  Es una palabra reservada que se conforma por las letras minúsculas: b, o, o, l. En ese orden.

- **NUM**

  Palabra reservada formada por las letras minúsculas: n, u, m. En ese orden.

- **TRUE**

  Es una palabra reservada compuesta por las letras minúsculas: t, r, u, e. En ese orden.

- **FALSE**

  Palabra reservada que contiene las siguientes letras minúsculas: f, a, l, s, e. En ese orden.

- **PLUS**

  Representa ua suma, \+.

- **MINUS**

  Representa una resta, \-.

- **MUL**

  Representa la multiplicación, $\ast$.

- **DIV**

  Representa la división, /.

- **IS**

  Representación de la asignación, =.

- **EQ**

  Representa una comparación, ==.

- **GT**

  Representa el mayor que, >.

- **GEQ**

  Representa el mayor o igual que, >=.

- **LT**

  Representa el menor que, <.

- **LEQ**

  Representa el menor o igual que, <=.

- **NEQ**

  Representa una desigualdad, !=.

- **AND**

  Representa el operador lógico and.

- **OR**

  Representa el operador lógico or.

- **NOT**

  Representa el operador lógico not.

- **POP**

  Representa un paréntesis de apertura, (.

- **PCL**

  Representa un paréntesis de cierre,).

- **EOL**

  Representa el punto y coma como separador especial, ;.

1.2 Especificación formal

Vamos a utilizar DR_s para dar una descrición formal del lenguaje que conforman las clases léxicas del apartado 2.1.

(∗) SPROG ≡ &&

(∗) LREAL ≡ LENT PDEC? PEXP?
PDEC ≡ \. Dig∗ Dig
PEXP ≡ (E | e) LENT

LENT ≡ Sign? Dig∗ Dig
Sign ≡ [\+, \-]
Dig ≡ [0-9]

(∗) ID ≡ Letter (Letter | Dig | _)∗
Letter ≡ [a-z, A-Z]
Dig ≡ [0-9]

(∗) BOOL ≡ b o o l

(∗) NUM ≡ n u m

(∗) TRUE ≡ t r u e

(∗) FALSE ≡ f a l s e

(∗) PLUS ≡ \+

(∗) MINUS ≡ \-

(∗) MUL ≡ \∗

(∗) DIV ≡ /

(∗) IS ≡ =

(∗) EQ ≡ ==

(∗) GT ≡ >

(∗) GEQ ≡ >=

(∗) LT ≡ <

(∗) LEQ ≡ <=

(∗) NEQ ≡ !=

(∗) AND ≡ a n d

(∗) OR ≡ o r

(∗) NOT ≡ n o t

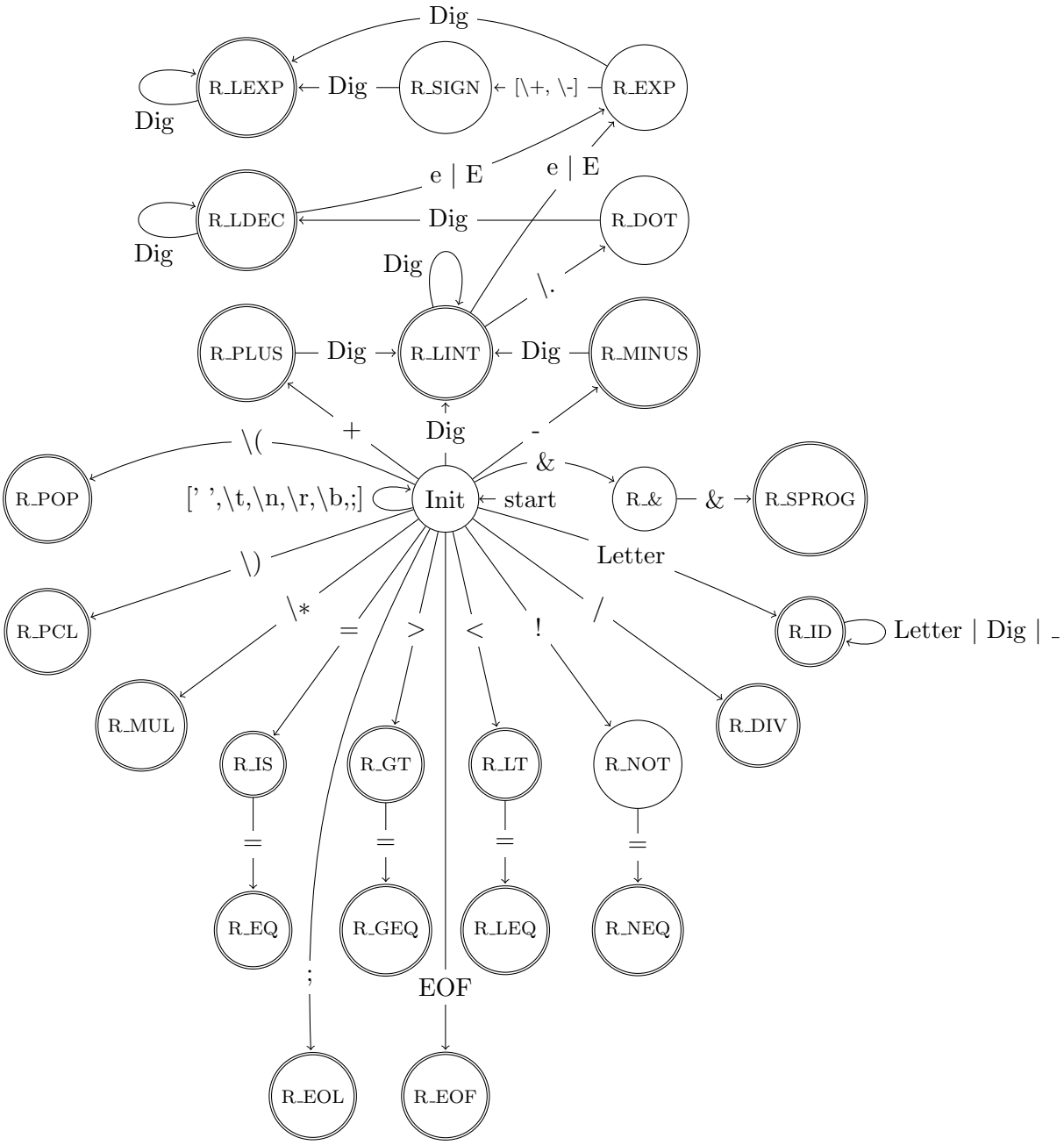
(∗) POP ≡ \(

(∗) PCL ≡ \)

(∗) EOL ≡ ;

[*I*] SEP ≡ [' ', \t, \n, \r, \b]

1.3 Diagrama de transiciones



2 Fase 2

2.1 Gramática incontextual

La especificación sintáctica utilizara la siguiente tabla de operadores.

Operador	Prioridad	Aridad	Tipo	Asociatividad
+, −	0	2	infijo	izq.
and	1	2	infijo	der.
or	1	2	infijo	no
<, >, ==, !=, <=, >=	2	2	infijo	no
∗, /	3	2	infijo	izq.
−	4	1	prefijo	si
not	4	1	prefijo	no

Con los operadores definidos procedemos a realizar la especificación.

S → Prog EOF
Prog → LDec SPROG LIns
LDec → LDec EOL Dec
LDec → Dec
LIns → LIns EOL Ins
LIns → Ins
Dec → NUM ID
Dec → BOOL ID
Ins → ID IS Exp0

Exp0 → Exp0 Op0 Exp1
Exp0 → Exp1
Exp1 → Exp2 AND Exp1
Exp1 → Exp2 OR Exp2
Exp1 → Exp2
Exp2 → Exp3 Op2 Exp3
Exp2 → Exp3
Exp3 → Exp3 Op3 Exp4
Exp3 → Exp4
Exp4 → MINUS Exp4
Exp4 → NOT Exp5
Exp4 → Exp5
Exp5 → LREAL
Exp5 → TRUE
Exp5 → FALSE
Exp5 → POP Exp0 PCL

Op0 → PLUS
Op0 → MINUS
Op2 → EQ
Op2 → GT
Op2 → GEQ
Op2 → LT
Op2 → LEQ
Op2 → NEQ
Op3 → MUL
Op3 → DIV

2.2 Gramática transformada

Vamos a transformar la gramática de la sección anterior para obtener una gramática LL(1) equivalente.

S → Prog EOF
Prog → SDec SPROG SIns
SDec → Dec LDec
LDec → EOL Dec LDec
LDec → ε
SIns → Ins LIns
LIns → EOL Ins LIns
LIns → ε
Dec → NUM ID
Dec → BOOL ID
Ins → ID IS Exp0

Exp0 → Exp1 R0
R0 → Op0 Exp1 R0
R0 → ε
Exp1 → Exp2 R1
R1 → AND Exp2 R1
R1 → OR Exp2
R1 → ε
Exp2 → Exp3 R2
R2 → Op2 Exp3 R2
R2 → ε
Exp3 → Exp4 R3
R3 → Op3 Exp4 R3
R3 → ε
Exp4 → MINUS Exp4
Exp4 → NOT Exp5
Exp4 → Exp5
Exp5 → LREAL
Exp5 → TRUE
Exp5 → FALSE
Exp5 → POP Exp0 PCL

Op0 → PLUS
Op0 → MINUS
Op2 → EQ
Op2 → GT
Op2 → GEQ
Op2 → LT
Op2 → LEQ
Op2 → NEQ
Op3 → MUL
Op3 → DIV

2.3 Primeros, siguientes y directores

Hemos utilizado la herramienta The Context Free Grammar Checker de la Universidad de Calgary para generar automáticamente los sets de símbolos. Para generarlos tan solo es necesario ir a la web y copiar el contenido del fichero transformed_grammar.md.

PRIM(S) = {NUM, BOOL}
PRIM(Prog) = {NUM, BOOL}
PRIM(SDec) = {NUM, BOOL}
PRIM(LDec) = {EOL, ε}
PRIM(SIns) = {ID}
PRIM(LIns) = {EOL, ε}
PRIM(Dec) = {NUM, BOOL}
PRIM(Ins) = {ID}
PRIM(R0) = {PLUS, MINUS, ε}
PRIM(Exp1) = {MINUS, NOT, LREAL, TRUE, FALSE, POP}
PRIM(R1) = {AND, OR, ε}
PRIM(Exp2) = {MINUS, NOT, LREAL, TRUE, FALSE, POP}
PRIM(R2) = {EQ, GT, GEQ, LT, LEQ, NEQ, ε}
PRIM(Exp3) = {MINUS, NOT, LREAL, TRUE, FALSE, POP}
PRIM(R3) = {MUL, DIV, ε}
PRIM(Exp4) = {MINUS, NOT, LREAL, TRUE, FALSE, POP}
PRIM(Exp5) = {LREAL, TRUE, FALSE, POP}
PRIM(Exp0) = {MINUS, NOT, LREAL, TRUE, FALSE, POP}
PRIM(Op0) = {PLUS, MINUS}
PRIM(Op2) = {EQ, GT, GEQ, LT, LEQ, NEQ}
PRIM(Op3) = {MUL, DIV}

SIG(S) = Ø
SIG(Prog) = {EOF}
SIG(SDec) = {SPROG}
SIG(LDec) = {SPROG}
SIG(SIns) = {EOF}
SIG(LIns) = {EOF}
SIG(Dec) = {EOL, SPROG}
SIG(Ins) = {EOL, EOF}
SIG(R0) = {PCL, EOL, EOF}
SIG(Exp1) = {PCL, PLUS, MINUS, EOL, EOF}
SIG(R1) = {PCL, PLUS, MINUS, EOL, EOF}
SIG(Exp2) = {PCL, AND, OR, PLUS, MINUS, EOL, EOF}
SIG(R2) = {PCL, AND, OR, PLUS, MINUS, EOL, EOF}
SIG(Exp3) = {PCL, EQ, GT, GEQ, LT, LEQ, NEQ, AND, OR, PLUS, MINUS, EOL, EOF}
SIG(R3) = {PCL, EQ, GT, GEQ, LT, LEQ, NEQ, AND, OR, PLUS, MINUS, EOL, EOF}

SIG(Exp4) = {PCL, MUL, DIV, EQ, GT, GEQ, LT, LEQ, NEQ, AND, OR, PLUS, MINUS, EOL, EOF}
SIG(Exp5) = {PCL, MUL, DIV, EQ, GT, GEQ, LT, LEQ, NEQ, AND, OR, PLUS, MINUS, EOL, EOF}
SIG(Exp0) = {PCL, EOL, EOF}
SIG(Op0) = {MINUS, NOT, LREAL, TRUE, FALSE, POP}
SIG(Op2) = {MINUS, NOT, LREAL, TRUE, FALSE, POP}
SIG(Op3) = {MINUS, NOT, LREAL, TRUE, FALSE, POP}

DIR(S → Prog EOF) = {NUM, BOOL}
DIR(Prog → SDec SPROG SIns) = {NUM, BOOL}
DIR(SDec → Dec LDec) = {NUM, BOOL}
DIR(LDec → EOL Dec LDec) = {EOL}
DIR(LDec → ε) = {SPROG}
DIR(SIns → Ins LIns) = {ID}
DIR(LIns → EOL Ins LIns) = {EOL}
DIR(LIns → ε) = {EOF}
DIR(Dec → NUM ID) = {NUM}
DIR(Dec → BOOL ID) = {BOOL}
DIR(Ins → ID EQ Exp0) = {ID}
DIR(Exp0 → Exp1 R0) = {MINUS, NOT, LREAL, TRUE, FALSE, POP}
DIR(R0 → Op0 Exp1 R0) = {PLUS, MINUS}
DIR(R0 → ε) = {PCL, EOL, EOF}
DIR(Exp1 → Exp2 R1) = {MINUS, NOT, LREAL, TRUE, FALSE, POP}
DIR(R1 → AND Exp2 R1) = {AND}
DIR(R1 → OR Exp2) = {OR}
DIR(R1 → ε) = {PCL, PLUS, MINUS, EOL, EOF}
DIR(Exp2 → Exp3 R2) = {MINUS, NOT, LREAL, TRUE, FALSE, POP}
DIR(R2 → Op2 Exp3 R2) = {EQ, GT, GEQ, LT, LEQ, NEQ}
DIR(R2 → ε) = {PCL, AND, OR, PLUS, MINUS, EOL, EOF}
DIR(Exp3 → Exp4 R3) = {MINUS, NOT, LREAL, TRUE, FALSE, POP}
DIR(R3 → Op3 Exp4 R3) = {MUL, DIV}
DIR(R3 → ε) = {PCL, EQ, GT, GEQ, LT, LEQ, NEQ, AND, OR, PLUS, MINUS, EOL, EOF}
DIR(Exp4 → MINUS Exp4) = {MINUS}
DIR(Exp4 → NOT Exp5) = {NOT}
DIR(Exp4 → Exp5) = {LREAL, TRUE, FALSE, POP}
DIR(Exp5 → LREAL) = {LREAl}
DIR(Exp5 → TRUE) = {TRUE}
DIR(Exp5 → FALSE) = {FALSE}
DIR(Exp5 → POP Exp0 PCL) = {POP}
DIR(Op0 → PLUS) = {PLUS}
DIR(Op0 → MINUS) = {MINUS}
DIR(Op2 → EQ) = {EQ}

DIR(Op2 → GT) = {GT}
DIR(Op2 → GEQ) = {GEQ}
DIR(Op2 → LT) = {LT}
DIR(Op2 → LEQ) = {LEQ}
DIR(Op2 → NEQ) = {NEQ}
DIR(Op3 → MUL) = {MUL}
DIR(Op3 → DIV) = {DIV}

3 Fase 4

3.1 Funciones constructoras

El primer paso para poder generar las funciones constructoras es simplificar la gramática incontextual generada en la fase 2.

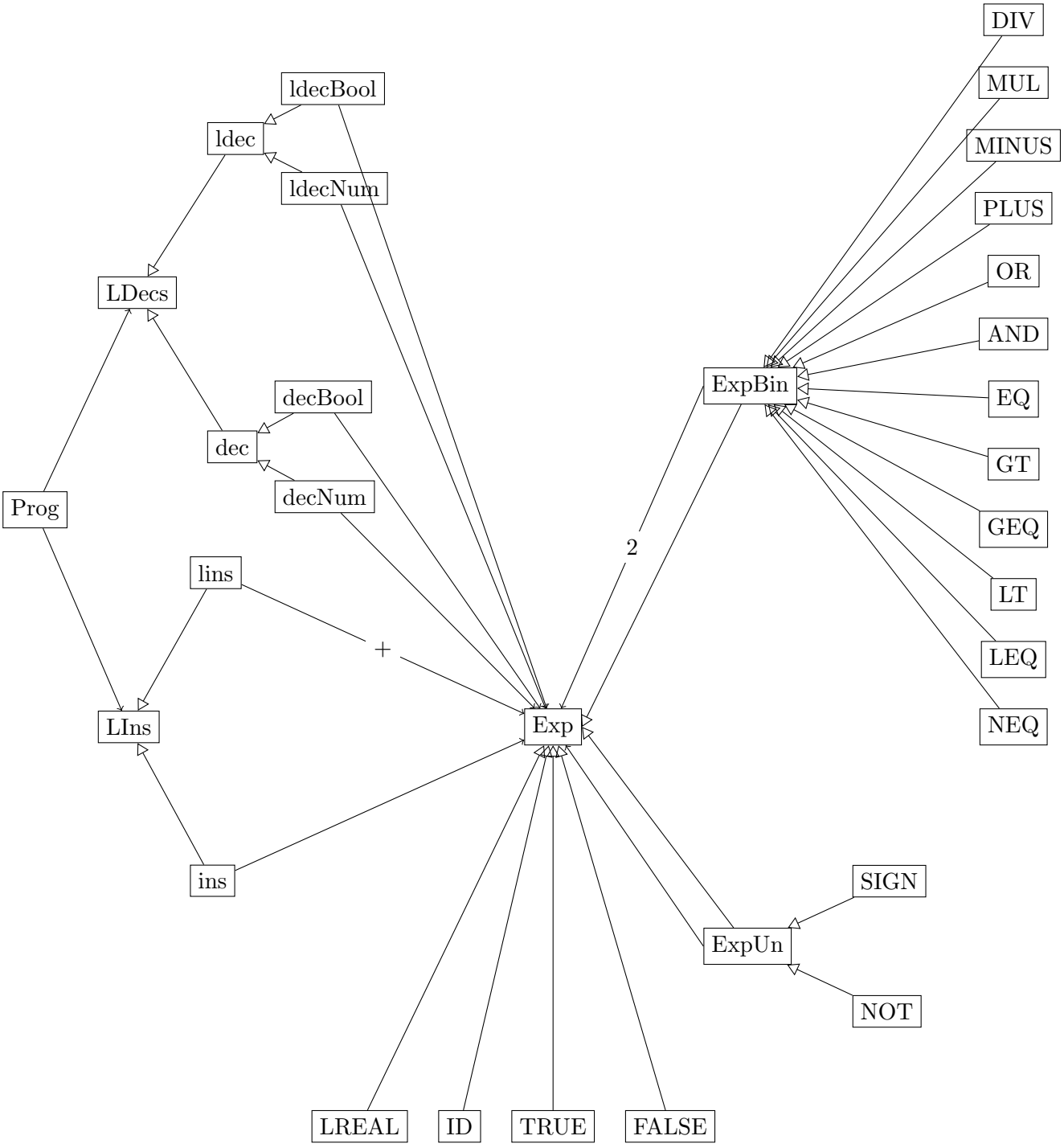
Prog → LDec SPROG LIns
LDec → LDec EOL NUM ID
| LDec EOL BOOL ID
| NUM ID
| BOOL ID
LIns → LIns EOL ID IS Exp
| ID IS Exp
Exp → Exp PLUS Exp
| Exp MINUS Exp
| Exp AND Exp
| Exp OR Exp
| Exp EQ Exp
| Exp GT Exp
| Exp GEQ Exp
| Exp LT Exp
| Exp LEQ Exp
| Exp NEQ Exp
| Exp MUL Exp
| Exp DIV Exp
| MINUS Exp
| NOT Exp
| LREAL
| TRUE
| FALSE
| POP Exp PCL

Por cada una de las reglas de esta gramática simplificada generaremos una función constructora.

- Prog → LDec SPROG LIns
  prog: LDec x LIns → Prog
- LDec → LDec EOL NUM ID
  ldecNum: LDec x String → LDec
- LDec → LDec EOL BOOL ID
  ldecBool: LDec x String → LDec
- LDec → NUM ID
  decNum: String → LDec
- LDec → BOOL ID
  decBool: String → LDec
- LIns → LIns EOL ID IS Exp
  lins: LIns x String x Exp → LIns
- LIns → ID IS Exp
  ins: String x Exp → LIns
- Exp → Exp PLUS Exp
  plus: Exp x Exp → Exp
- Exp → Exp MINUS Exp
  minus: Exp x Exp → Exp
- Exp → Exp AND Exp
  and: Exp x Exp → Exp
- Exp → Exp OR Exp
  or: Exp x Exp → Exp
- Exp → Exp EQ Exp
  eq: Exp x Exp → Exp
- Exp → Exp GT Exp
  gt: Exp x Exp → Exp
- Exp → Exp GEQ Exp
  geq: Exp x Exp → Exp
- Exp → Exp LT Exp
  lt: Exp x Exp → Exp
- Exp → Exp LEQ Exp
  leq: Exp x Exp → Exp
- Exp → Exp NEQ Exp
  neq: Exp x Exp → Exp

- Exp → Exp MUL Exp
  mul: Exp x Exp → Exp
- Exp → Exp DIV Exp
  div: Exp x Exp → Exp
- Exp → MINUS Exp
  sign: Exp x Exp → Exp
- Exp → NOT Exp
  not: Exp x Exp → Exp
- Exp → LREAL
  num: String → Exp
- Exp → TRUE
  true: String → Exp
- Exp → FALSE
  false: String → Exp

3.2 Diagrama de clases



3.3 Gramática de atributos

S → Prog EOF
  S.a = Prog.a
Prog → LDec SPROG LIns
  Prog.a = prog(LDec.a, LIns.a)
LDec → LDec EOL Dec
  $LDec_0$.a = ldec($LDec_1$.a, Dec.a)
LDec → Dec
  LDec.a = Dec.a
LIns → LIns EOL Ins
  $LIns_0$.a = lins($LIns_1$.a, Ins.id, Ins.exp)
LIns → Ins
  LIns.a = ins(Ins.id, Ins.exp)
Dec → NUM ID
  Dec.a = decNum(ID.lex)
Dec → BOOL ID
  Dec.a = decBool(ID.lex)
Ins → ID IS Exp0
  Ins.id = ID.lex
  Ins.exp = Exp0.exp
Exp0 → Exp0 Op0 Exp1
  $Exp0_0$.a = mkexpb(Op0.op, $Exp0_1$.a, Exp1.a)
Exp0 → Exp1
  Exp0.a = Exp1.a
Exp1 → Exp2 AND Exp1
  $Exp1_0$.a = mkexpb('and', Exp2.a, $Exp1_1$.a)
Exp1 → Exp2 OR Exp2
  Exp1.a = mkexpb('or', $Exp2_0$.a, $Exp2_1$.a)
Exp1 → Exp2
  Exp1.a = Exp2.a
Exp2 → Exp3 Op2 Exp3
  Exp2.a = mkexpb(Op2.op, $Exp3_0$.a, $Exp3_1$.a)
Exp2 → Exp3
  Exp2.a = Exp3.a
Exp3 → Exp3 Op3 Exp4
  $Exp3_0$.a = mkexpb(Op3.a, $Exp3_1$.a, Exp4.a)
Exp3 → Exp4
  Exp3.a = Exp4.a
Exp4 → MINUS Exp4
  $Exp4_0$.a = mkexpu('-', $Exp4_1$.a)
Exp4 → NOT Exp5
  Exp4.a = mkexpu('not', Exp5.a)
Exp4 → Exp5

Exp4.a = Exp5.a
Exp5 → LREAL
Exp5.a = num(LREAL.lex)
Exp5 → TRUE
Exp5.a = true(TRUE.lex)
Exp5 → FALSE
Exp5.a = false(FALSE.lex)
Exp5 → POP Exp0 PCL
Exp5.a = Exp0.a
Op0 → PLUS
Op0.op = '+'
Op0 → MINUS
Op0.op = '-'
Op2 → EQ
Op2.op = '=='
Op2 → GT
Op2.op = '>'
Op2 → GEQ
Op2.op = '>='
Op2 → LT
Op2.op = '<'
Op2 → LEQ
Op2.op = '<='
Op2 → NEQ
Op2.op = '!='
Op3 → MUL
Op3.op = '*'
Op3 → DIV
Op3.op = '/'

Las funciones mkexpb y mkexpu son las encargadas de construir las expresiones apropiadas para operadores binarios y unarios, respectivamente.

```
fun mkexpb (op, opd1, opd2) {
    switch(op) {
        '+': return plus(opd1, opd2)
        '-': return minus(opd1, opd2)
        '*': return mul(opd1, opd2)
        '/': return div(opd1, opd2)
        'and': return and(opd1, opd2)
        'or': return or(opd1, opd2)
        '==': return eq(opd1, opd2)
        '>': return gt(opd1, opd2)
        '>=': return geq(opd1, opd2)
```

```
        '<': return lt(opd1, opd2)
        '<=': return leq(opd1, opd2)
        '!=': return neq(opd1, opd2)
    }
}
fun mkexpu (op, opd) {
    switch(op) {
        '-': return sign(opd)
        'not': return not(opd)
    }
}
```

3.4 Acondicionamiento para implementación descendente

S → Prog EOF
  S.a = Prog.a
Prog → SDec SPROG SIns
  Prog.a = prog(LDec.a, LIns.a)
SDec → Dec LDec
  LDec.ah = Dec.a
  SDec.a = LDec.a
LDec → EOL Dec LDec
  $LDec_1$.a = ldec($LDec_1$.ah, Dec.a)
  $LDec_0$.a = $LDec_1$.a
LDec → ε
  LDec.a = LDec.ah
SIns → Ins LIns
  LIns.ah = ins(Ins.id, Ins.exp)
  SIns.a = LIns.a
LIns → EOL Ins LIns
  $LIns_1$.a = lins($LIns_1$.ah, Ins.id, Ins.exp)
  $LIns_0$.a = $LIns_1$.a
LIns → ε
  LIns.a = LIns.ah
Dec → NUM ID
  Dec.a = decNum(ID.lex)
Dec → BOOL ID
  Dec.a = decBool(ID.lex)
Ins → ID IS Exp0
  Ins.id = ID.lex
  Ins.exp = Exp0.exp
Exp0 → Exp1 R0
  R0.ah = Exp1.a

Exp0.a = R0.a

R0 → Op0 Exp1 R0

$R0_1$.ah = mkexpb(Op0.op, $R0_0$.ah, Exp1.a)

$R0_0$.a = $R0_1$.a

R0 → ε

R0.a = R0.ah

Exp1 → Exp2 R1

R1.ah = Exp2.a

Exp1.a = R1.a

R1 → AND Exp2 R1

$R1_1$.ah = mkexpb('and', $R1_0$.ah, Exp2.a)

$R1_0$.a = $R1_1$.a

R1 → OR Exp2

R1.a = mkexpb('or', R1.ah, Exp2.a)

R1 → ε

R1.a = R1.ah

Exp2 → Exp3 R2

R2.ah = Exp3.a

Exp2.a = R2.a

R2 → Op2 Exp3 R2

$R2_1$.ah = mkexpb(Op2.op, $R2_0$.ah, Exp3.a)

$R2_0$.a = $R2_1$.a

R2 → ε

R2.a = R2.ah

Exp3 → Exp4 R3

R3.ah = Exp4.a

Exp3.a = R3.a

R3 → Op3 Exp4 R3

$R3_1$.ah = mkexpb(Op3.op, $R3_0$.ah, Exp4.a)

$R3_0$.a = $R3_1$.a

R3 → ε

R3.a = R3.ah

Exp4 → MINUS Exp4

$Exp4_0$.a = mkexpu('-', $Exp4_1$.a)

Exp4 → NOT Exp5

Exp4.a = mkexpu('not', Exp5.a)

Exp4 → Exp5

Exp4.a = Exp5.a

Exp5 → LREAL

Exp5.a = num(LREAL.lex)

Exp5 → TRUE

Exp5.a = true(TRUE.lex)

Exp5 → FALSE

Exp5.a = false(FALSE.lex)

Exp5 → POP Exp0 PCL

Exp5.a = Exp0.a

Op0 → PLUS

Op0.op = '+'

Op0 → MINUS

Op0.op = '-'

Op2 → EQ

Op2.op = '=='

Op2 → GT

Op2.op = '>'

Op2 → GEQ

Op2.op = '>='

Op2 → LT

Op2.op = '<'

Op2 → LEQ

Op2.op = '<='

Op2 → NEQ

Op2.op = '!='

Op3 → MUL

Op3.op = '*'

Op3 → DIV

Op3.op = '/'