Procesadores de Lenguajes

Memoria de actividades

Lidia Concepción Echeverría
lidiacon@ucm.es

Juan Ramón del Caño Vega
jdelcano@ucm.es

June 24, 2018

Contents

Introducción

La estructura del proyecto está dividida en fases. Cada fase contiene un directorio con todas las implementaciones requeridas.

En el caso de la fase 4, la sintaxis abstracta ha sido implementada una sola vez. Esta implementación es usada tanto por el constructor ascendente como por el descendente, por lo que cualquier cambio en ella se verá reflejado en ambas implementaciones. Por ello habrá que modificar el buildpath para que sea reconocida.

1 Primera fase

1.1 Clases léxicas

Nuestro lenguaje dispondrá de las siguientes clases léxicas:

1. Nombres de tipo: serán palabras reservadas en nuestro analizador léxico.
   - *num*: palabra reservada para tipar las variables numéricas.
   - *bool*: palabra reservada para tipar las variables booleanas.
2. Nombres de variable: *iden* serán los identificadores de las variables que declaremos. Deberán empezar por una letra, y pueden contener letras, dígitos o subrayados (_).
3. Separadores: dispondremos de dos tipos, uno para separar la sección de declaraciones de la sección de instrucciones, y otro para separar las declaraciones o instrucciones entre ellas.
   - *pcoma*: será el separador de declaraciones o instrucciones entre ellas mismas (;).
   - *end*: indicará el final de la sección de declaraciones y el comienzo de la sección de instrucciones (&&).
4. Expresiones: las usaremos para trabajar en la sección de instrucciones. Tenemos dos tipos.
   (a) Expresiones lógicas: de nuevo serán palabras reservadas.
      - *true*: valor lógico de verdad (true).
      - $false$: contrario de *true* (false).

   (b) Expresiones numéricas: *numero*, podrán comenzar por un signo (+ o -), seguido de uno o más dígitos. Si se tratase de números reales, aparecería un punto (.) seguido de uno o más dígitos. Finalmente, para ambos casos, podría aparecer una parte exponencial, con una e o E seguida opcionalmente de un signo (+ o -) y uno o más dígitos.
5. Operadores: para la sección de instrucciones dispondremos de los siguientes operadores.
   (a) Operador de asignación: *igual* (=) entre una variable y una expresión.

   (b) Operadores aritméticos:
      - *mas*: operador de suma (+).
      - *menos*: operador de resta y menos unario (-).
      - *por*: operador de multiplicación (*).
      - *div*: operador de división (/).

(c) Operadores lógicos: también serán palabras reservadas.

- *and*: operador de conjunción.
- *or*: operador de disyunción.
- *not*: operador de negación.

(d) Operadores relacionales.

- *mayor*: comprueba si una expresión es estrictamente mayor a otra (>).
- *menor*: comprueba si una expresión es estrictamente menor a otra (<).
- *mayorIgual*: comprueba si una expresión es mayor o igual a otra (>=).
- *menorIgual*: comprueba si una expresión es menor o igual a otra (<=).
- *equiv*: comprueba si dos expresiones tienen el mismo valor (==).
- *noEquiv*: comprueba si dos expresiones tienen distinto valor (! =).

(e) Paréntesis: modifican la prioridad de las operaciones entre *parAb* (( ) y *parCe* ( )).

1.2 Especificación formal

- **Definiciones auxiliares:**

$$letra \longrightarrow a|...|z|A|...|Z \tag{1}$$
$$digitoPositivo \longrightarrow 1|...|9 \tag{2}$$
$$digito \longrightarrow 0|digitoPositivo \tag{3}$$
$$parteDecimal \longrightarrow .digito^{+} \tag{4}$$
$$parteExponencial \longrightarrow (e|E)[+|-]digito^{+} \tag{5}$$

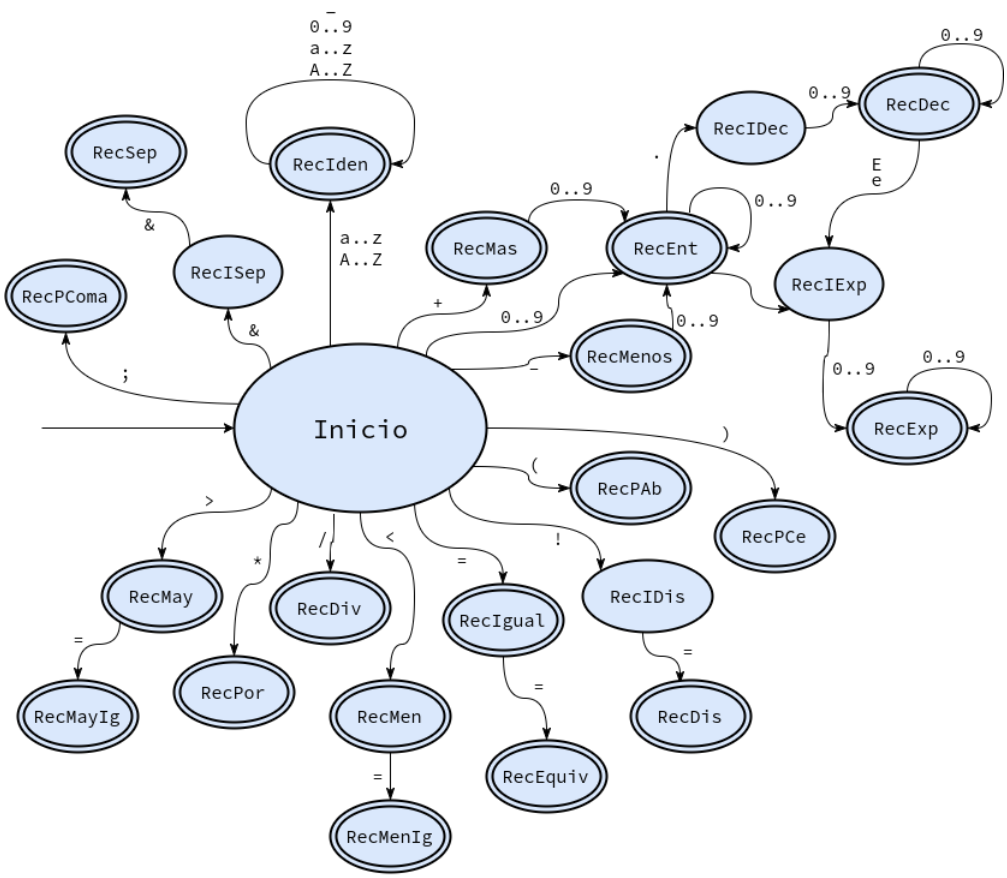
- **Definiciones de cadenas ignorables:**

$$separador \longrightarrow \textbf{SP—TAB—NL} \tag{6}$$

- **Definiciones léxicas:**

$$num \longrightarrow \textbf{num} \tag{7}$$
$$bool \longrightarrow \textbf{bool} \tag{8}$$
$$iden \longrightarrow letra(letra|digito|\_)^{*} \tag{9}$$
$$pcoma \longrightarrow ; \tag{10}$$
$$end \longrightarrow \&\& \tag{11}$$
$$igual \longrightarrow = \tag{12}$$
$$true \longrightarrow \textbf{true} \tag{13}$$
$$false \longrightarrow \textbf{false} \tag{14}$$
$$numero \longrightarrow [+|-]digito^{+}[parteDecimal][parteExponencial] \tag{15}$$
$$mas \longrightarrow + \tag{16}$$
$$menos \longrightarrow - \tag{17}$$
$$por \longrightarrow * \tag{18}$$
$$div \longrightarrow / \tag{19}$$
$$and \longrightarrow \textbf{and} \tag{20}$$
$$or \longrightarrow \textbf{or} \tag{21}$$
$$not \longrightarrow \textbf{not} \tag{22}$$
$$mayor \longrightarrow > \tag{23}$$
$$menor \longrightarrow < \tag{24}$$
$$mayorIgual \longrightarrow >= \tag{25}$$
$$menorIgual \longrightarrow <= \tag{26}$$
$$equiv \longrightarrow == \tag{27}$$
$$noEquiv \longrightarrow != \tag{28}$$
$$parAb \longrightarrow (\tag{29}$$
$$parCe \longrightarrow) \tag{30}$$

1.3 Diagrama de transiciones



Todos los estados finales vuelven al estado inicial para leer el siguiente token

1.4 Cómo usar JLex

En la implementación mediante JLex, la clase del analizador léxico es generada desde el archivo `Tiny.l`, usando `JLex.jar`. Para realizar cualquier cambio, debe modificarse el léxico definido en `Tiny.l`, y volver a generar el analizador léxico, mediante el comando:

```
java -cp /ruta/JLex.jar JLex.Main /ruta/Tiny.l
```

Esto generará nuestro AnalizadorLexico bajo el nombre `Tiny.l.java` por lo que tendremos que renombrarlo antes de usarlo.

2 Segunda fase

2.1 Tabla de operadores

	Tipo	Prioridad	Asociatividad
+	Binario infijo	0	Asociativo a izquierdas
-	Binario infijo	0	Asociativo a izquierdas
and	Binario infijo	1	Asociativo a derechas
or	Binario infijo	1	No asocia
<	Binario infijo	2	No asocia
>	Binario infijo	2	No asocia
<=	Binario infijo	2	No asocia
>=	Binario infijo	2	No asocia
==	Binario infijo	2	No asocia
!=	Binario infijo	2	No asocia
*	Binario infijo	3	Asociativo a izquierdas
/	Binario infijo	3	Asociativo a izquierdas
-	Unario prefijo	4	Asocia
not	Unario prefijo	4	No asocia

2.2 Gramática incontextual

$$G = (V_N, V_T, P, S)$$

$$V_N = \{S, Ds, D, T, Is, I, E0, E1, E2, E3, E4, E5, OP\}$$

$$V_F = \{+, -, and, or, >, <, >=, <=, ==, !=, *, /, not, (,), identificador, true, false\}$$

$$P = \begin{cases} S \longrightarrow Ds \ \&\& \ Is \\ \\ Ds \longrightarrow D \mid D; \ Ds \\ D \longrightarrow T \ identificador \\ T \longrightarrow num \mid bool \\ \\ Is \longrightarrow I \mid I; \ Is \\ I \longrightarrow identificador \ = \ E0 \\ \\ E0 \longrightarrow E0 + E1 \mid E0 - E1 \mid E1 \\ E1 \longrightarrow E2 \text{ and } E1 \mid E2 \text{ or } E2 \mid E2 \\ E2 \longrightarrow E3 \ OP \ E3 \mid E3 \\ E3 \longrightarrow E3 * E4 \mid E3/E4 \mid E4 \\ E4 \longrightarrow -E4 \mid \text{not } E5 \mid E5 \\ E5 \longrightarrow (E0) \mid identificador \mid numReal \mid \text{true} \mid \text{false} \\ \\ OP \longrightarrow < \ | \ > \ | \ <= \ | \ >= \ | \ == \ | \ != \end{cases}$$

2.3 Acondicionamiento para LL(1)

2.3.1 Eliminación de recursión por la izquierda

$$G = (V_N, V_T, P, S)$$

$$V_N = \{S, Ds, D, T, Is, I, E0, E0', E1, E2, E3, E3', E4, E5, OP\}$$

$$V_F = \{+, -, and, or, >, <, >=, <=, ==, !=, *, /, not, (,), identificador, true, false\}$$

$$P = \begin{cases} S \longrightarrow Ds \;\&\&\; Is \\ \\ Ds \longrightarrow D \mid D;\; Ds \\ D \longrightarrow T\ identificador \\ T \longrightarrow num \mid bool \\ \\ Is \longrightarrow I \mid I;\; Is \\ I \longrightarrow identificador\ =\ E0 \\ \\ E0 \longrightarrow E1\ E0' \\ E0' \longrightarrow +\ E1\ E0' \mid -\ E1\ E0' \mid \epsilon \\ E1 \longrightarrow E2 \text{ and } E1 \mid E2 \text{ or } E2 \mid E2 \\ E2 \longrightarrow E3\ OP\ E3 \mid E3 \\ E3 \longrightarrow E4\ E3' \\ E3' \longrightarrow *\ E4\ E3 \mid /\ E4\ E3 \mid \epsilon \\ E4 \longrightarrow -E4 \mid \text{not } E5 \mid E5 \\ E5 \longrightarrow (E0) \mid identificador \mid numReal \mid \text{true} \mid \text{false} \\ \\ OP \longrightarrow <\ \mid\ >\ \mid\ <=\ \mid\ >=\ \mid\ ==\ \mid\ != \end{cases}$$

2.3.2 Eliminación de factores a la izquierda

$$G = (V_N, V_T, P, S)$$

$$V_N = \{S, Ds, D, T, Is, I, E0, E0', E1, FE1, E2, FE2, E3, E3', E4, E5, OP\}$$

$$V_F = \{+, -, and, or, >, <, >=, <=, ==, !=, *, /, not, (,), identificador, true, false\}$$

$$P = \begin{cases} S \longrightarrow Ds \;\&\&\; Is \\ \\ Ds \longrightarrow D\ FD \\ D \longrightarrow T\ identificador \\ FD \longrightarrow \epsilon \mid ;\ D\ FD \\ T \longrightarrow num \mid bool \\ \\ Is \longrightarrow I\ FI \\ I \longrightarrow identificador\ =\ E0 \\ FI \longrightarrow \epsilon \mid ;\ I\ FI \\ \\ E0 \longrightarrow E1\ E0' \\ E0' \longrightarrow +\ E1\ E0' \mid -\ E1\ E0' \mid \epsilon \\ E1 \longrightarrow E2\ FE1 \\ FE1 \longrightarrow \text{and}\ E1 \mid \text{or}\ E2 \mid \epsilon \\ E2 \longrightarrow E3\ FE2 \\ FE2 \longrightarrow OP\ E3 \mid \epsilon \\ E3 \longrightarrow E4\ E3' \\ E3' \longrightarrow *\ E4\ E3 \mid /\ E4\ E3 \mid \epsilon \\ E4 \longrightarrow -E4 \mid \text{not}\ E5 \mid E5 \\ E5 \longrightarrow (E0) \mid identificador \mid numReal \mid \text{true} \mid \text{false} \\ \\ OP \longrightarrow < \mid > \mid <= \mid >= \mid == \mid != \end{cases}$$

2.4 Primeros, siguientes y directores

Productor	Primeros	Siguientes	
S	num, bool	⊘	
D	num, bool	;, &&	
FD	;	&&	
Ds	num, bool	&&	
T	num, bool	identificador	
I	identificador	;	
FI	;	⊘	
Is	identificador	⊘	
E0'	+, -	), ;	
FE1	and, or	), +, -, ;	
E1	-, not, (, identificador, numReal, true, false	), +, -, ;	
E2	-, not, (, identificador, numReal, true, false	), and, or, +, -, ;	
FE2	<, >, <=, >=, ==, !=	), and, or, +, -, ;	
E3	-, not, (, identificador, numReal, true, false	), <, >, <=, >=, ==, !=, and, or, +, -, ;	
E3'	*, /	), <, >, <=, >=, ==, !=, and, or, +, -, ;	
E4	-, not, (, identificador, numReal, true, false	), *, /, <, >, <=, >=, ==, !=, and, or, +, -, ;	
E5	(, identificador, numReal, true, false	), *, /, <, >, <=, >=, ==, !=, and, or, +, -, ;	
E0	-, not, (, identificador, numReal, true, false	), ;	
OP	<, >, <=, >=, ==, !=	-, not, (, identificador, numReal, true, false	

Productor	Directores
S ⟶ Ds && Is	num, bool
Ds ⟶ D FD	num, bool
D ⟶ T identificador	num, bool
FD ⟶ epsilon	&&
FD ⟶ ; D FD	;
T ⟶ num	num
T ⟶ bool	bool
Is ⟶ I FI	identificador
I ⟶ identificador = E0	identificador
FI ⟶ epsilon	-
FI ⟶ ; I FI	;
E0 ⟶ E1 E0'	-, not, (, identificador, numReal, true, false
E0' ⟶ + E1 E0'	+
E0' ⟶ - E1 E0'	-
E0' ⟶ epsilon	), -, ;
E1 ⟶ E2 FE1	-, not, (, identificador, numReal, true, false
FE1 ⟶ and E1	and
FE1 ⟶ or E2	or
FE1 ⟶ epsilon	+, -,), -, ;
E2 ⟶ E3 FE2	-, not, (, identificador, numReal, true, false
FE2 ⟶ OP E3	<, >, <=, >=, ==, !=
FE2 ⟶ epsilon	and, or, +, -,), -, ;
E3 ⟶ E4 E3'	-, not, (, identificador, numReal, true, false
E3' ⟶ * E4 E3'	*
E3' ⟶ / E4 E3'	/
E3' ⟶ epsilon	<, >, <=, >=, ==, !=, and, or, +, -,), -, ;, not, (, identificador, numReal, true, false
E4 ⟶ - E4	-
E4 ⟶ not E5	not
E4 ⟶ E5	(, identificador, numReal, true, false
E5 ⟶ (E0)	(
E5 ⟶ identificador	identificador
E5 ⟶ numReal	numReal
E5 ⟶ true	true
E5 ⟶ false	false
OP ⟶ <	<
OP ⟶ >	>
OP ⟶ <=	<=
OP ⟶ >=	>=
OP ⟶ ==	==
OP ⟶ !=	!=

2.5 Cómo usar Javacc

A partir del archivo `Tiny.jj` se generan todas clases Java necesarias para tener el analizador sintáctico funcional (excepto la clase Main). Al igual que para JLex, los cambios sobre la gramática deben hacerse sobre el archivo `Tiny.jj` y generar de nuevo las clases mediante el comando:

```
java -cp lib/javacc.jar javacc src/asint/Tiny.jj
```

3 Tercera fase

Para la implementación ascendente usaremos la gramática original diseñada en la fase 2.

3.1 Cómo usar CUP

Lo primero será generar el analizador léxico de la misma forma que en la fase 1, con la excepción de que esta vez nuestro `Tiny.l` deberá estar modificado para soportar la integración con CUP.

Después debemos generar el analizador sintáctico, los símbolos y el parser:

```
java -cp lib/cup.jar java_cup.Main -parser AnalizadorSintacticoTiny
-symbols ClaseLexica -nopositions src/asint/Tiny.cup
```

Una vez generado el parser y los símbolos, habrá que moverlos al paquete correspondiente (se generarán en la raíz).

4 Cuarta fase

4.1 Funciones constructoras

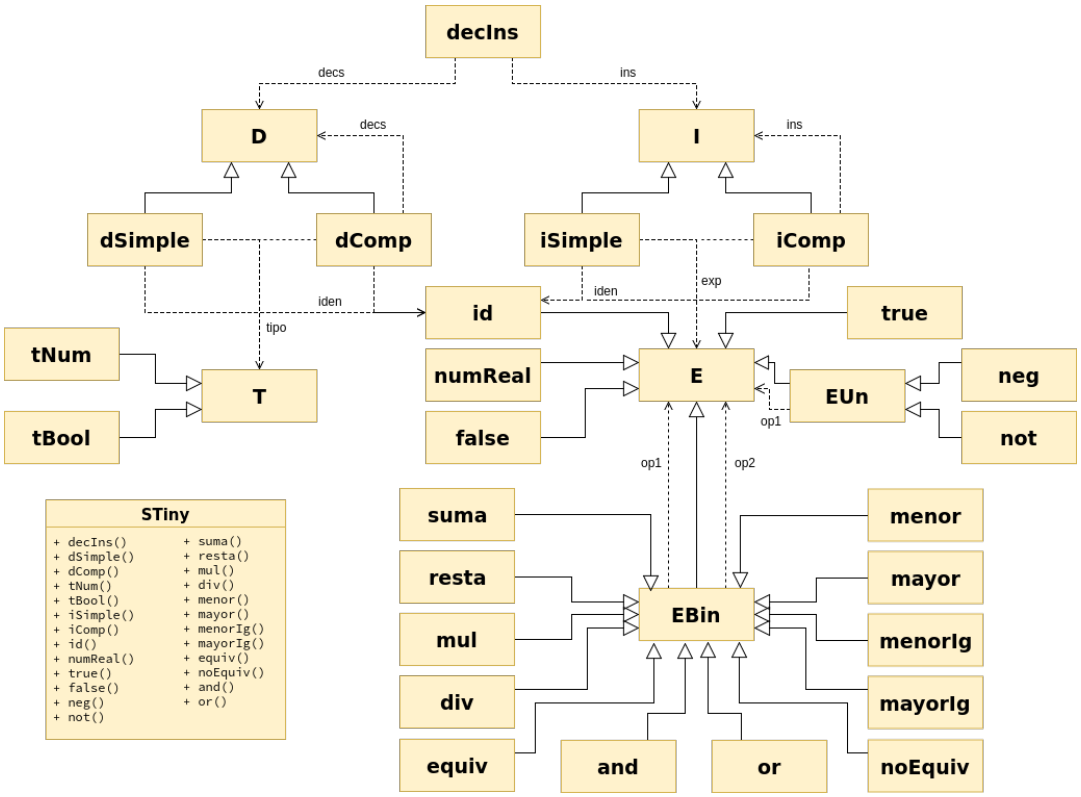
4.1.1 Simplificación de la gramática

$$P = \begin{cases} S \longrightarrow Ds \;\&\&\; Is \\ \\ Ds \longrightarrow T\ identificador \mid Ds;\ T\ identificador \\ T \longrightarrow num \mid bool \\ \\ Is \longrightarrow identificador\ =\ E \mid Is;\ identificador\ =\ E \\ E \longrightarrow E + E \mid E - E \mid E \text{ and } E \mid E \text{ or } E \mid E\ <\ E \mid E\ >\ E \mid E\ <=\ E \\ \qquad \mid E\ >=\ E \mid E\ ==\ E \mid E\ !=\ E \mid E * E \mid E/E \mid\ -E \mid \text{not } E \mid (E) \\ \qquad \mid identificador \mid numReal \mid \text{true} \mid \text{false} \\ \\ OP \longrightarrow < \mid > \mid <= \mid >= \mid == \mid != \end{cases}$$

4.1.2 Constructoras y tipos

Regla	**Constructora**
S ⟶ Ds && Is	**decIns:** Ds x Is ⟶ S
Ds ⟶ T identificador	**dSimple:** T x **string** ⟶ Ds
Ds ⟶ Ds; T identificador	**dCompuesta:** Ds x T x **string** ⟶ Ds
T ⟶ num	**tNum: num** ⟶ T
T ⟶ bool	**tBool: bool** ⟶ T
Is ⟶ identificador = E	**iSimple: string** x E ⟶ Is
Is ⟶ Is; identificador = E	**iCompuesta:** Is x **string** x E ⟶ Is
E ⟶ E + E	**suma:** E x E ⟶ E
E ⟶ E - E	**resta:** E x E ⟶ E
E ⟶ E and E	**mul:** E x E ⟶ E
E ⟶ E or E	**div:** E x E ⟶ E
E ⟶ E < E	**and:** E x E ⟶ E
E ⟶ E > E	**or:** E x E ⟶ E
E ⟶ E <= E	**menor:** E x E ⟶ E
E ⟶ E >= E	**mayor:** E x E ⟶ E
E ⟶ E == E	**menorIg:** E x E ⟶ E
E ⟶ E != E	**mayorIg:** E x E ⟶ E
E ⟶ E * E	**equiv:** E x E ⟶ E
E ⟶ E / E	**noEquiv:** E x E ⟶ E
E ⟶ - E	**neg:** E x E ⟶ E
E ⟶ not E	**not:** E x E ⟶ E
E ⟶ identificador	**id: string** ⟶ E
E ⟶ numReal	**real: string** ⟶ E
E ⟶ true	**true** ⟶ E
E ⟶ false	**false** ⟶ E

4.2 Diagrama de clases



4.3 Gramática de atributos para el constructor de árboles

S $\longrightarrow$ Ds && Is	S.a = decIns(Ds.a, Is.a)
Ds $\longrightarrow$ D Ds $\longrightarrow$ D; Ds	Ds.a = dSimple(D.tipo, D.iden) Ds.a = dCompuesta(Ds'.a, D.tipo, D.iden)
D $\longrightarrow$ T identificador	D.tipo = T.a D.iden = identificador.lex
T $\longrightarrow$ num T $\longrightarrow$ bool	T.a = tNum() T.a = tBool()
Is $\longrightarrow$ I Is $\longrightarrow$ I; Is	Is.a = iSimple(I.iden, I.exp) Is.a = iCompuesta(Is'.a, I.iden, I.exp)
I $\longrightarrow$ identificador = E0	I.iden = identificador.lex I.exp = E0.a
E0 $\longrightarrow$ E0 + E1 E0 $\longrightarrow$ E0 - E1 E0 $\longrightarrow$ E1	E0.a = suma(E0'.a, E1.a) E0.a = resta(E0'.a, E1.a) E0.a = E1.a
E1 $\longrightarrow$ E2 and E1 E1 $\longrightarrow$ E2 or E2 E1 $\longrightarrow$ E2	E1.a = and(E2.a, E1'a) E1.a = or(E2.a, E2'.a) E1.a = E2.a
E2 $\longrightarrow$ E3 OP E3 E2 $\longrightarrow$ E3	E2.a = mkexp(OP.op, E3.a, E3'.a) E2.a = E3.a
E3 $\longrightarrow$ E3 * E4 E3 $\longrightarrow$ E3 / E4 E3 $\longrightarrow$ E4	E3.a = mul(E3'.a, E4.a) E3.a = div(E3'.a, E4.a) E3.a = E4.a
E4 $\longrightarrow$ - E4 E4 $\longrightarrow$ not E5 E4 $\longrightarrow$ E5	E4.a = neg(E4'.a) E4.a = not(E5.a) E4.a = E5.a
E5 $\longrightarrow$ (E0) E5 $\longrightarrow$ identificador E5 $\longrightarrow$ numReal E5 $\longrightarrow$ true E5 $\longrightarrow$ false	E5.a = E0.a E5.a = id(identificador.lex) E5.a = numReal(numReal.lex) E5.a = true() E5.a = false()
OP $\longrightarrow$ < OP $\longrightarrow$ > OP $\longrightarrow$ <= OP $\longrightarrow$ >= OP $\longrightarrow$ == OP $\longrightarrow$! =	OP.op = < OP.op = > OP.op = <= OP.op = >= OP.op = == OP.op = ! =

4.3.1 Función `mkexp()`

Data: OP: op; E: opnd1, opnd2
Result: E
switch *op* **do**
  case < **do**
    return `menor(opnd1, opnd2)`
  end
  case > **do**
    return `mayor(opnd1, opnd2)`
  end
  case <= **do**
    return `menorIg(opnd1, opnd2)`
  end
  case >= **do**
    return `mayorIg(opnd1, opnd2)`
  end
  case == **do**
    return `equiv(opnd1, opnd2)`
  end
  case ! = **do**
    return `noEquiv(opnd1, opnd2)`
  end
end

4.4 Acondicionamiento para implementación descendente

S ⟶ Ds && Is	S.a = decIns(Ds.a, Is.a)
Ds ⟶ D FD	FD.ah = dSimple(D.tipo, D.iden) Ds.a = FD.a
D ⟶ T identificador	D.tipo = T.a D.iden = identificador.lex
FD ⟶ ; D FD	$FD_1.a = dCompuesta(FD_0.ah, Ds.tipo, Ds.iden)$ $FD_0.a = FD_1.a$
FD ⟶ epsilon	FD.a = FD.ah
T ⟶ num T ⟶ bool	T.a = tNum() T.a = tBool()
Is ⟶ I FI	FI.ah = iSimple(I.iden, I.exp) Is.a = FI.a
I ⟶ identificador = E0	I.iden = identificador.lex I.exp = E0.a
FI ⟶ ; I FI	$FI_1.a = iCompuesta(FI_0.ah, Is.tipo, Is.iden)$ $FI_0.a = FI_1.a$
FI ⟶ epsilon	FI.a = FI.ah

E0 $\longrightarrow$ E1 E0'	E0'.ah = E1.a E0.a = E0'.a
E0' $\longrightarrow$ + E1 E0'	$E0'_1.ah = suma(E0'_0.ah, E1.a)$ $E0'_0.a = E0'_1.a$
E0' $\longrightarrow$ - E1 E0'	$E0'_1.ah = resta(E0'_0.ah, E1.a)$ $E0'_0.a = E0'_1.a$
E0' $\longrightarrow$ epsilon	E0'.a = E0'.ah
E1 $\longrightarrow$ E2 FE1	FE1.ah = E2.a E1.a = FE1.a
FE1 $\longrightarrow$ and E1	FE1.a = and(FE1.ah, E1.a)
FE1 $\longrightarrow$ or E2	FE1.a = or(FE1.ah, E1.a)
FE1 $\longrightarrow$ epsilon	FE1.a = FE1.ah
E2 $\longrightarrow$ E3 FE2	FE2.ah = E3.a E2.a = FE2.a
FE2 $\longrightarrow$ OP E3	FE2.a = mkexp(OP.op, FE2.ah, E3.a)
FE2 $\longrightarrow$ epsilon	FE2.a = FE2.ah
E3 $\longrightarrow$ E4 E3'	E3'.ah = E4.a E3.a = E3'.a
E3' $\longrightarrow$ * E4 E3'	$E3'_1.ah = mul(E3'_0.ah, E4.a)$ $E3'_0.a = E3'_1.a$
E3' $\longrightarrow$ / E4 E3'	$E3'_1.ah = div(E3'_0.ah, E4.a)$ $E3'_0.a = E3'_1.a$
E3' $\longrightarrow$ epsilon	E3'.a = E3'.ah
E4 $\longrightarrow$ - E4	E4.a = neg(E4'.a)
E4 $\longrightarrow$ not E5	E4.a = not(E5.a)
E4 $\longrightarrow$ E5	E4.a = E5.a
E5 $\longrightarrow$ (E0)	E5.a = E0.a
E5 $\longrightarrow$ identificador	E5.a = id(identificador.lex)
E5 $\longrightarrow$ numReal	E5.a = numReal(numReal.lex)
E5 $\longrightarrow$ true	E5.a = true()
E5 $\longrightarrow$ false	E5.a = false()
OP $\longrightarrow$ <	OP.op = <
OP $\longrightarrow$ >	OP.op = >
OP $\longrightarrow$ <=	OP.op = <=
OP $\longrightarrow$ >=	OP.op = >=
OP $\longrightarrow$ ==	OP.op = ==
OP $\longrightarrow$! =	OP.op = ! =