

# Desarrollo de constructores de ASTs descendentes y ascendentes

FACULTAD DE INFORMÁTICA



UNIVERSIDAD  
**COMPLUTENSE**  
MADRID

Asignatura de Procesadores de Lenguajes

Curso 2021-2022

Grupo 20

Adrián Martín Tiscar  
Gema Blanco Núñez

# 1. Especificación de la sintaxis abstracta

A continuación se realiza la enumeración de las firmas (cabeceras) de las funciones constructoras de ASTs.

## **Sintaxis abstracta:**

programa\_conDecs: Decs x Instrucciones  $\rightarrow$  Programa  
programa\_sinDecs: Instrucciones  $\rightarrow$  Programa  
decs\_varias: Decs x Dec  $\rightarrow$  Decs  
decs\_una: Dec  $\rightarrow$  Decs  
dec\_var: tipo x id  $\rightarrow$  Dec  
dec\_type: tipo x id  $\rightarrow$  Dec  
dec\_proc: id x ParamForm x bloque  
param\_form\_vacio:  $\rightarrow$  ParamForm  
param\_form: ListParamForm  $\rightarrow$  ParamForm  
list\_param\_form\_uno: Parametro  $\rightarrow$  ParamForm  
list\_param\_form\_varios: ListParamForm x Parametro  $\rightarrow$  ParamForm  
param\_valor: tipo x id  $\rightarrow$  Parametro  
param\_variable: tipo x id  $\rightarrow$  Parametro  
tipo\_int:  $\rightarrow$  tipo  
tipo\_real:  $\rightarrow$  tipo  
tipo\_bool:  $\rightarrow$  tipo  
tipo\_string:  $\rightarrow$  tipo  
tipo\_array: num x tipo  $\rightarrow$  tipo  
tipo\_record: campos  $\rightarrow$  tipo  
tipo\_pointer: tipo  $\rightarrow$  tipo  
campos\_varios: campos x campo  $\rightarrow$  campos  
campos\_uno: campo  $\rightarrow$  campos  
campo: tipo x string  $\rightarrow$  campo  
instrucciones\_varias: Instrucciones x Inst  $\rightarrow$  Instrucciones  
instrucciones\_una: Inst  $\rightarrow$  Instrucciones  
inst\_asig: Expresion x Expresion  $\rightarrow$  Inst  
inst\_IfThen: Expresion x InstOp  $\rightarrow$  Inst  
inst\_IfThenElse: Expresion x InstOp x InstOp  $\rightarrow$  Inst  
inst\_While: Expresion x InstOp  $\rightarrow$  Inst  
inst\_op\_varias: Instrucciones  $\rightarrow$  InstOp  
inst\_op\_vacio:  $\rightarrow$  InstOp  
inst\_Read: Expresion  $\rightarrow$  Inst  
inst\_Write: Expresion  $\rightarrow$  Inst  
inst\_NewLine:  $\rightarrow$  Inst  
inst\_new: Expresion  $\rightarrow$  Inst  
inst\_delete: Expresion  $\rightarrow$  Inst  
inst\_bloque: Bloque  $\rightarrow$  Inst  
bloque: Programa  $\rightarrow$  Bloque  
bloque\_vacio:  $\rightarrow$  Bloque  
inst\_Call: String x paramReales  $\rightarrow$  Inst

paramReales\_varios: ListaExpresiones  $\rightarrow$  paramReales  
 paramReales\_vacio:  $\rightarrow$  paramReales  
 listaExpresiones\_varias: ListaExpresiones x Expresion  $\rightarrow$  ListaExpresiones  
 listaExpresiones\_una: Expresion  $\rightarrow$  ListaExpresiones  
 suma: Expresion x Expresion  $\rightarrow$  Expresion  
 resta: Expresion x Expresion  $\rightarrow$  Expresion  
 and: Expresion x Expresion  $\rightarrow$  Expresion  
 or: Expresion x Expresion  $\rightarrow$  Expresion  
 mayor: Expresion x Expresion  $\rightarrow$  Expresion  
 mayor\_igual: Expresion x Expresion  $\rightarrow$  Expresion  
 menor: Expresion x Expresion  $\rightarrow$  Expresion  
 menor\_igual: Expresion x Expresion  $\rightarrow$  Expresion  
 comparacion: Expresion x Expresion  $\rightarrow$  Expresion  
 distinto: Expresion x Expresion  $\rightarrow$  Expresion  
 mul: Expresion x Expresion  $\rightarrow$  Expresion  
 div: Expresion x Expresion  $\rightarrow$  Expresion  
 mod: Expresion x Expresion  $\rightarrow$  Expresion  
 neg: Expresion  $\rightarrow$  Expresion  
 not: Expresion  $\rightarrow$  Expresion  
 indexacion: Expresion x Expresion  $\rightarrow$  Expresion  
 punto: Expresion x id  $\rightarrow$  Expresion  
 flecha: Expresion x id  $\rightarrow$  Expresion  
 indice: Expresion  $\rightarrow$  Expresion  
  
 id: string  $\rightarrow$  Expresion  
 numeroEntero: string  $\rightarrow$  Expresion  
 numeroReal: string  $\rightarrow$  Expresion  
 true:  $\rightarrow$  Expresion  
 false:  $\rightarrow$  Expresion  
 null:  $\rightarrow$  Expresion  
 literalCad: string  $\rightarrow$  Expresion

## 2. Especificación del constructor de ASTs mediante una gramática s-atribuida

### Constructor de árboles de sintaxis abstracta (ASTs):

Programa  $\rightarrow$  Decs '&&' Instrucciones  
     Programa.a = programa\_conDecs(Decs.a, Instrucciones.a)  
 Programa  $\rightarrow$  Instrucciones  
     Programa.a = programa\_sinDecs(Instrucciones.a)

Decs  $\rightarrow$  Decs ';' Dec  
     Decs<sub>0</sub>.a = decs\_varias(Decs<sub>0</sub>.a, Dec.a)  
 Decs  $\rightarrow$  Dec  
     Decs.a = decs\_una(Dec.a)

Dec  $\rightarrow$  var tipo id  
     Dec.a = dec\_var(tipo.a, id.lex)  
 Dec  $\rightarrow$  type tipo id  
     Dec.a = dec\_type(tipo.a, id.lex)  
 Dec  $\rightarrow$  proc id ParamForm bloque  
     Dec.a = dec\_proc(id.lex, ParamForm.a, bloque.a)  
 ParamForm  $\rightarrow$  '(' ListParamForm ')'  
     ParamForm.a = param\_form(ListParamForm.a)  
 ParamForm  $\rightarrow$  '(' ')'  
     ParamForm.a = param\_form\_vacio()

ListParamForm  $\rightarrow$  ListParamForm ';' Parametro  
     ListParamForm<sub>0</sub>.a = list\_param\_form\_varios(ListParamForm<sub>1</sub>.a, Parametro.a)  
 ListParamForm  $\rightarrow$  Parametro  
     ListParamForm.a = list\_param\_form\_uno(Parametro.a)

Parametro  $\rightarrow$  tipo '&' id  
     Parametro.a = param\_variable(tipo.a, id.lex)  
 Parametro  $\rightarrow$  tipo id  
     Parametro.a = param\_valor(tipo.a, id.lex)

bloque  $\rightarrow$  '{' Programa '  
     Bloque.a = bloque(Programa.a)  
 bloque  $\rightarrow$  '{' '}'  
     bloque.a = bloque\_vacio()

tipo  $\rightarrow$  int  
     tipo.a = tipo\_int()  
 tipo  $\rightarrow$  bool  
     tipo.a = tipo\_bool()  
 tipo  $\rightarrow$  real  
     tipo.a = tipo\_real()  
 tipo  $\rightarrow$  string  
     tipo.a = tipo\_string()  
 tipo  $\rightarrow$  id  
     tipo.a = id(id.lex)  
 tipo  $\rightarrow$  array '[' numeroEntero ']' of tipo  
     tipo<sub>0</sub>.a = tipo\_array(numeroEntero.lex, tipo<sub>1</sub>.a)  
 tipo  $\rightarrow$  record '{' campos '  
     tipo.a = tipo\_record(campos.a)  
 tipo  $\rightarrow$  pointer tipo  
     tipo<sub>0</sub>.a = tipo\_pointer(tipo<sub>1</sub>.a)

campos  $\rightarrow$  campos ';' campo  
campos<sub>0</sub>.a = campos\_varios(campos<sub>1</sub>.a, campo.a)  
campos  $\rightarrow$  campo  
campos.a = campos\_uno(campo.a)

campo  $\rightarrow$  tipo id  
campo.a = campo(tipo.a, id.lex)

Instrucciones  $\rightarrow$  Instrucciones ';' Inst  
Instrucciones<sub>0</sub>.a = instrucciones\_varias(Instrucciones<sub>1</sub>.a, Inst.a)  
Instrucciones  $\rightarrow$  Inst  
Instrucciones.a = instrucciones\_una(Inst.a)

- Instrucción de asignación:

Inst  $\rightarrow$  Expresion '=' Expresion  
Inst.a = inst\_asig(Expresion.a, Expresion.a)

- Instrucción if-then:

Inst  $\rightarrow$  if Expresion then InstOp endif  
Inst.a = inst\_IfThen(Expresion.a, InstOp.a)

- Instrucción if-then-else:

Inst  $\rightarrow$  if Expresion then InstOp else InstOp endif  
Inst.a = inst\_IfThenElse(Expresion.a, InstOp.a, InstOp.a)

- Instrucción while:

Inst  $\rightarrow$  while Expresion do InstOp endwhile  
Inst.a = inst\_While(Expresion.a, InstOp.a)

InstOp  $\rightarrow$  Instrucciones  
InstOp.a = inst\_op\_varias(Instrucciones.a)  
InstOp  $\rightarrow$   $\epsilon$   
InstOp.a = inst\_op\_vacio()

- Instrucción de lectura

Inst  $\rightarrow$  read Expresion  
Inst.a = inst\_Read(Expresion.a)

- Instrucción de escritura

Inst  $\rightarrow$  write Expresion  
Inst.a = inst\_Write(Expresion.a)

- Instrucción de nueva línea

Inst  $\rightarrow$  nl  
Inst.a = inst\_NewLine()

- Instrucción de reserva de memoria

Inst  $\rightarrow$  new Expresion

Ints.a = inst\_new(Expresion.a)

- Instrucción de liberación de memoria

Inst → delete Expresion

Ints.a = inst\_delete(Expresion.a)

- Instrucción de invocación a procedimiento

Inst → call id paramReales

Ints.a = inst\_Call(id.lex, paramReales.a)

paramReales → '(' ListaExpresiones ')'

paramReales.a = paramReales\_varios(ListaExpresiones.a)

paramReales → '(' ' ')

paramReales.a = paramReales\_vacio()

ListaExpresiones → ListaExpresiones ',' Expresion

ListaExpresiones<sub>0</sub>.a = listaExpresiones\_varias(ListaExpresiones<sub>1</sub>.a, Expresion.a)

ListaExpresiones → Expresion

ListaExpresiones.a = listaExpresiones\_una(Expresion.a)

- Instrucción compuesta

Inst → bloque

Ints.a = inst\_bloque(bloque.a)

Expresion → E0

Expresion.a = E0.a

E0 → E1 '+' E0

E0<sub>0</sub>.a = suma(E1.a, E0<sub>1</sub>.a)

E0 → E1 '-' E1

E0.a = resta(E1<sub>0</sub>.a, E1<sub>1</sub>.a)

E0 → E1

E0.a = E1.a

E1 → E1 OP1 E2

E1<sub>0</sub>.a = exp(OP1.a, E1<sub>1</sub>.a, E2.a)

E1 → E2

E1.a = E2.a

E2 → E2 OP2 E3

E2<sub>0</sub>.a = exp(OP2.a, E2<sub>0</sub>.a, E3.a)

E2 → E3

E2.a = E3.a

E3 → E4 OP3 E4

E3.a = exp(OP2.a, E4<sub>0</sub>.a, E4<sub>0</sub>.a)

E3 → E4

E3.a = E4.a

E4 → '-' E5

E4.a = neg(E5.a)

E4 → not E4

E4.a = not(E4.a)

E4 → E5  
E4.a = E5.a

E5 → E5 '[' expr ']'  
E5<sub>0</sub>.a = indexacion(E5<sub>1</sub>.a, expr.a)

E5 → E5 '.' id  
E5<sub>0</sub>.a = punto(E5<sub>1</sub>.a, id.lex)

E5 → E5 '->' id  
E5<sub>0</sub>.a = flecha(E5<sub>1</sub>.a, id.lex)

E5 → E6  
E5.a = E6.a

E6 → '\*' E6  
E6.a = indice(E6.a)

E6 → E7  
E6.a = E7.a

E7 → numeroEntero  
E7.a = numeroEntero(numeroEntero.lex)

E7 → numeroReal  
E7.a = numeroReal(numeroReal.lex)

E7 → id  
E7.a = id(id.lex)

E7 → true  
E7.a = true()

E7 → false  
E7.a = false()

E7 → literalCad  
E7.a = literalCad(literalCad.lex)

E7 → null  
E7.a = null()

E7 → (E0)  
E7.a = E0.a

OP1 → and  
OP1.a = 'and'

OP1 → or  
OP1.a = 'or'

OP2 → '>'  
OP2.a = '>'

OP2 → '>='  
OP2.a = '>='

OP2 → '<'  
OP2.a = '<'

OP2 → '<='  
OP2.a = '<='

```

OP2 → '=='
      OP2.a = '=='
OP2 → '!='
      OP2.a = '!='
OP3 → '*'
      OP3.a = '*'
OP3 → '/'
      OP3.a = '/'
OP3 → '%'
      OP3.a = '%'

```

### **Funciones semánticas:**

```

fun exp(op, arg0, arg1){
  switch op
    case 'and': return and(arg0, arg1)
    case 'or': return or(arg0, arg1)
    case '<': return menor(arg0, arg1)
    case '>': return mayor(arg0, arg1)
    case '<=': return menor_igual(arg0, arg1)
    case '>=': return mayor_igual(arg0, arg1)
    case '==': return equivalente(arg0, arg1)
    case '!=': return distinto(arg0, arg1)
    case '*': return mul(arg0, arg1)
    case '/': return div(arg0, arg1)
}

```

## 3. Acondicionamiento de la especificación

A continuación se realiza el condicionamiento de la especificación para permitir la implementación descendente. Se aplican dos transformaciones:

- Eliminación de factores comunes
- Eliminación de recursión a izquierdas

En la siguiente tabla se muestran las reglas sin acondicionar y sus respectivas transformaciones.



## Eliminación de recursión a izquierdas

Sin acondicionar	Acondicionada
<p>Decs <math>\rightarrow</math> Decs ';' Dec  Decs.a = decs_varias(Decs.a, Dec.a)</p>	<p>Decs <math>\rightarrow</math> Dec restoDecs  restoDecs.ah = Dec.a  Decs.a = restoDecs.a  restoDecs <math>\rightarrow</math> ; Dec restoDecs  restoDecs<sub>1</sub>.ah = decs_varias(restoDecs<sub>0</sub>.ah, Dec.a)  restoDecs<sub>0</sub>.a = restoDecs<sub>1</sub>.a  restoDecs <math>\rightarrow</math> <math>\epsilon</math>  restoDecs.a = restoDecs.ah</p>
<p>ListParamForm <math>\rightarrow</math> ListParamForm ';' Parametro  ListParamForm<sub>0</sub>.a =  list_param_form_varios(ListParamForm<sub>1</sub>.a, Parametro.a)</p>	<p>ListParamForm <math>\rightarrow</math> Parametro restoListaParamForm  restoListaParamForm.ah = Parametro.a  ListParamForm.a = restoListaParamForm.a  restoListaParamForm <math>\rightarrow</math> ; Parametro restoListaParamForm  restoListaParamForm<sub>1</sub>.ah =  list_param_form_varios(ListParamForm<sub>0</sub>.ah, Parametro.a)  restoListaParamForm<sub>0</sub>.a = restoListaParamForm<sub>1</sub>.a  restoListaParamForm <math>\rightarrow</math> <math>\epsilon</math>  restoListaParamForm.a = restoListaParamForm.ah</p>
<p>campos <math>\rightarrow</math> campos ';' campo  campos<sub>0</sub>.a = campos_varios(campos<sub>1</sub>.a,  campo.a)  campos <math>\rightarrow</math> campo  campos.a = campos_uno(campo.a)</p>	<p>campos <math>\rightarrow</math> campo restoCampos  restoCampos.ah = campo.a  campos.a = restoCampos.a  restoCampos <math>\rightarrow</math> ';' campo restoCampos  restoCampos<sub>1</sub>.ah =  campos_varios(restoCampos<sub>0</sub>.ah, campo.a)  restoCampos<sub>0</sub>.a = restoCampos<sub>1</sub>.a  restoCampos <math>\rightarrow</math> <math>\epsilon</math>  restoCampos.a = restoCampos.ah</p>
<p>Instrucciones <math>\rightarrow</math> Instrucciones ';' Inst  Instrucciones<sub>0</sub>.a = inst_varias(Instrucciones<sub>1</sub>.a,  Inst.a)</p>	<p>Instrucciones <math>\rightarrow</math> Inst restolns  restolns.ah = Inst.a  Instrucciones.a = restolns.a  restolns <math>\rightarrow</math> ; Inst restolns  restolns<sub>1</sub>.ah = inst_varias(restolns<sub>0</sub>.ah, Inst.a)  restolns<sub>0</sub>.a = restolns<sub>1</sub>.a  restolns <math>\rightarrow</math> <math>\epsilon</math>  restolns.a = restolns.ah</p>
<p>ListaExpresiones <math>\rightarrow</math> ListaExpresiones ';' Expresion  ListaExpresiones<sub>0</sub>.a =  listaExpresiones_varias(ListaExpresiones<sub>1</sub>.a,  Expresion.a)  ListaExpresiones <math>\rightarrow</math> Expresion</p>	<p>ListaExpresiones <math>\rightarrow</math> Expresion RestoListaExpresiones  RestoListaExpresiones.ah = Expresion.a  ListaExpresiones.a = RestoListaExpresiones.a  RestoListaExpresiones <math>\rightarrow</math> ';' Expresion  RestoListaExpresiones</p>

<p>ListaExpresiones.a = listaExpresiones_una(Expresion.a)</p>	<p>RestoListaExpresiones<sub>1</sub>.ah = listaExpresiones_varias(RestoListaExpresiones<sub>0</sub>.ah, Expresion.a) RestoListaExpresiones<sub>0</sub>.a = RestoListaExpresiones<sub>1</sub>.a RestoListaExpresiones → ε RestoListaExpresiones.a = RestoListaExpresiones.ah</p>
<p>E1 → E1 OP1 E2 E1<sub>0</sub>.a = exp(OP1.a, E1<sub>1</sub>.a, E2.a)</p>	<p>E1 → E2 restoE1 restoE1.ah = E2.a E1.a = restoE1.a restoE1 → OP1 E2 restoE1 restoE1<sub>1</sub>.ah = E2.a restoE1<sub>0</sub>.a = exp(OP1.a, restoE1<sub>0</sub>.ah, restoE1<sub>1</sub>.a) restoE1 → ε restoE1.a = restoE1.ah</p>
<p>E2 → E2 OP2 E3 E2<sub>0</sub>.a = exp(OP2.a, E2<sub>1</sub>.a, E3.a)</p>	<p>E2 → E3 restoE2 restoE2.ah = E3.a E2.a = restoE2.a restoE2 → OP2 E3 restoE2 restoE2<sub>1</sub>.ah = E3.a restoE2<sub>0</sub>.a = exp(OP2.a, restoE2<sub>0</sub>.ah, restoE2<sub>1</sub>.a) restoE2 → ε restoE2.a = restoE2.ah</p>
<p>E5 → E5 '[' expr '] E5<sub>0</sub>.a = indexacion(E5<sub>1</sub>.a, expr.a) E5 → E5 '.' id E5<sub>0</sub>.a = punto(E5<sub>1</sub>.a, id.lex) E5 → E5 '-&gt;' id E5<sub>0</sub>.a = flecha(E5<sub>1</sub>.a, id.lex)</p>	<p><b>Primero eliminamos factores comunes:</b> E5 → E5 restoE5 restoE5.ah = E5<sub>1</sub>.a E5.a = restoE5.a restoE5 → '[' expr '] restoE5.a = indexacion(restoE5.ah, expr.a) restoE5 → '.' id restoE5.a = punto(restoE5.ah, id.lex) restoE5 → '-&gt;' id restoE5.a = flecha(restoE5.ah, id.lex)</p> <p><b>Y sobre esto eliminamos la recursion a izquierdas:</b></p> <p>E5 → E6 resto2E5 Res2E5.ah = E6.a E5.a = resto2E5.a resto2E5 → restoE5 resto2E5 restoE5.ah = resto2E5<sub>0</sub>.ah resto2E5<sub>1</sub>.ah = restoE5.a resto2E5<sub>0</sub>.a = resto2E5<sub>1</sub>.a</p>

	<p>resto2E5 <math>\rightarrow \epsilon</math>  Rest2E5.a = Rest2E5.ah  restoE5 <math>\rightarrow</math> '[' expr ']  restoE5.a = indexacion(restoE5.ah, expr)  restoE5 <math>\rightarrow</math> '.' id  restoE5.a = punto(restoE5.ah, id.lex)  restoE5 <math>\rightarrow</math> '-&gt;' id  restoE5.a = flecha(restoE5.ah, id.lex)</p>
<b>Eliminación de factores comunes</b>	
<b>Sin acondicionar</b>	<b>Acondicionada</b>
ParamForm $\rightarrow$ '(' ListParamForm ')' ParamForm.a = param_form(ListParamForm.a) ParamForm $\rightarrow$ '(' ' ParamForm.a = param_form_vacio()	ParamForm $\rightarrow$ '(' restoParamForm ParamForm.a = restoParamForm.a restoParamForm $\rightarrow$ ListParamForm ')' restoParamForm.a = param_form(ListParamForm.a) restoParamForm $\rightarrow$ ')' restoParamForm.a = param_form_vacio()
Parametro $\rightarrow$ tipo '&' id Parametro.a = param(tipo.a, id.lex) Parametro $\rightarrow$ tipo id Parametro.a = param(tipo.a, id.lex)	Parametro $\rightarrow$ tipo restoParametro restoParametro.ah = tipo.a Parametro.a = restoParametro.a restoParametro $\rightarrow$ '&' id restoParametro.a = param(restoParametro.ah, id.lex) restoParametro $\rightarrow$ id restoParametro.a = param(restoParametro.ah, id.lex)
Inst $\rightarrow$ if Expresion then InstOp endif Inst.a = inst_IfThen(Expresion.a, InstOp.a)  - Instrucción if-then-else: Inst $\rightarrow$ if Expresion then InstOp else InstOp endif Inst.a = inst_IfThenElse(Expresion.a, InstOp.a, InstOp.a)	Inst $\rightarrow$ If Expresion then InstOp restolf restolf.ahex = Expresion.a restolf.ahop = InstOp.a Inst.a = restolf.a restolf $\rightarrow$ endif restolf.a = inst_IfThen(restolf.ahex, restolf.ahop) restolf $\rightarrow$ else InstOp endif restolf.a = inst_IfThenElse(restolf.ahex, restolf.ahop, InstOp.a)
paramReales $\rightarrow$ '(' ListaExpresiones ')' paramReales.a = paramReales_varios(ListaExpresiones.a) paramReales $\rightarrow$ '(' ' paramReales.a = paramReales_vacio()	paramReales $\rightarrow$ '(' restoParamReales restoParamReales.a = restoParamReales.a restoParamReales $\rightarrow$ ListaExpresiones ')' restoParamReales.a = paramReales_varios(ListaExpresiones.a) restoParamReales $\rightarrow$ ')' restoParamReales.a = paramReales_vacio()
E0 $\rightarrow$ E1 '+' E0 E0.a = suma(E1.a, E0.a) E0 $\rightarrow$ E1 '-' E1 E0.a = resta(E1.a, E1.a)	E0 $\rightarrow$ E1 restoE0 restoE0.ah = E1.a E0.a = restoE0.a restoE0 $\rightarrow$ '+' E0 restoE0.a = suma(restoE0.ah, E0.a)

$E0 \rightarrow E1$ $E0.a = E1.a$	$restoE0 \rightarrow '-' E1$ $restoE0.a = resta(restoE0.ah, E1.a)$ $restoE0 \rightarrow \epsilon$ $restoE0.a = restoE0.ah$
$E3 \rightarrow E4 \text{ OP3 } E4$ $E3.a = \exp(OP3.a, E4_1.a, E4_2.a)$ $E3 \rightarrow E4$ $E3.a = E4.a$	$E3 \rightarrow E4 \text{ restoE3}$ $restoE3.ah = E4.a$ $E3.a = restoE3.a$ $restoE3 \rightarrow OP3 E4$ $restoE3.a = \exp(OP3.a, restoE3.ah, E4.a)$ $restoE3 \rightarrow \epsilon$ $restoE3.a = restoE3.ah$

**NOTA:** no hemos logrado terminar de implementar una versión funcional del analizador descendente para tiny1 por errores que no supimos solucionar ni cómo avanzar, especialmente por atascarnos con la recursión a izquierdas. Sí hemos conseguido implementar el analizador ascendente. Aún así, en la entrega se observa todos los avances que hicimos en el analizador descendente.