Desarrollo de constructores de ASTs

Procesadores de Lenguaje - GII (UCM)

Grupo 18

Jorge Ortega, Alejandro Tobías

Miguel Amato y Daniela Vidal

Índice

1. Especificación de la sintaxis abstracta de Tiny

a. Obtención de la gramática abstracta

- ☐ Se eliminan los terminales sin **carga semántica** (e.g.: símbolos de puntuación).
- ☐ Todos los Ei resultan ser equivalentes entre sí ⇒ pueden **fusionarse** en un no terminal común E.
- ☐ Expresiones como E → E op E dan lugar a **géneros discriminativos** ⇒ se simplifican.
- ☐ Los nodos que sean meros contenedores de otros son **géneros envoltorio** ⇒ se simplifican.

programa → declaraciones_opt instrucciones_opt

declaraciones_opt → lista_declaraciones
declaraciones_opt → ε
lista_declaraciones → lista_declaraciones declaración
lista_declaraciones → declaración
declaración → tipo **id**
declaración → **type** tipo **id**
declaración → **proc id** parametros_formales programa
parametros_formales → lista_parametros
parametros_formales → ε

lista_parametros → lista_parametros parametro
lista_parametros → parametro
parametro → tipo **id**
parametro → tipo **& id**

tipo → tipo [literal_ent]
tipo → **^**tipo
tipo → **int**

tipo → **real**

tipo → **bool**

tipo → **id**

tipo → **struct** lista_struct

lista_struct → lista_struct campo

lista_struct → campo

campo → tipo **id**

instrucciones_opt → lista_instrucciones

instrucciones_opt → ε

lista_instrucciones → lista_instrucciones instruccion

lista_instrucciones → instruccion

instruccion → E

instruccion → **if** E programa

instruccion → **if** E programa **else** programa

instruccion → **while** E programa

instruccion → **read** E

instruccion → **write** E

instruccion → **nl**

instruccion → **new** E

instruccion → **delete** E

instruccion → **call id** parametros_reales

instruccion → programa

parametros_reales → lista_parametros_reales

parametros_reales → ε

lista_parametros_reales → lista_parametros_reales E

lista_parametros_reales → E

E → E = E

E → E > E

E → E >= E

E → E < E

E → E <= E

E → E == E

E → E != E

E → E + E

E → E - E

E → E **and** E

E → E **or** E

E → E * E

E → E / E

E → E % E

E → - E

E → **not** E

E → E **[** E **]**

E → E **.id**

E → E **^**

E → **literal_ent**

E → **literal_real**

E → **literal_cadena**

E → **true**

E → **false**

E → **id**

E → **null**

b. Enumeración de las signaturas

No terminal	Género
programa	Prog
declaraciones_opt	Decs
lista_declaraciones	LDecs
declaración	Dec

parametros_formales	ParamF
lista_parametros	LParam
parametro	Param
tipo	Tipo
lista_struct	LStruct
campo	Campo
instrucciones_opt	InstrOpt
lista_instrucciones	LInstr
instruccion	Instr
parametros_reales	ParamR
lista_parametros_reales	LParamR
E	Exp

Regla	Constructor
programa → declaraciones_opt instrucciones_opt	prog: Decs x InstrOpt → Prog
declaraciones_opt → lista_declaraciones	si_decs: LDecs → Decs
declaraciones_opt → ε	no_decs: → Decs
lista_declaraciones → lista_declaraciones declaración	muchas_decs: LDecs x Dec → LDecs
lista_declaraciones → declaración	una_dec: Dec → LDecs
declaración → tipo id	dec_id: Tipo x **string** → Dec
declaración → type tipo id	dec_type: Tipo x **string** → Dec
declaración → proc id parametros_formales programa	dec_proc: **string** x ParamF x Prog → Dec
parametros_formales → lista_parametros	si_parF: LParam → ParamF

parametros_formales → ε	no_parF: → ParamF
lista_parametros → lista_parametros parametro	muchos_param: LParam x Param → LParam
lista_parametros → parametro	un_param: Param → LParam
parametro → tipo id	param_cop: Tipo x **string** → Param
parametro → tipo & id	param_ref: Tipo x **string** → Param
tipo → tipo [**literal_ent**]	tipo_array: Tipo x **string** → Tipo
tipo → ^tipo	tipo_punt: Tipo → Tipo
tipo → int	tipo_int: → Tipo
tipo → real	tipo_real: → Tipo
tipo → bool	tipo_bool: → Tipo
tipo → id	tipo_id: **string** → Tipo
tipo → string	tipo_string: → Tipo
tipo → struct lista_struct	tipo_struct: LStruct → Tipo
lista_struct → lista_struct campo	lista_struct: LStruct x Campo → LStruct
lista_struct → campo	info_struct: Campo → LStruct
campo → tipo id	campo: Tipo x **string** → Campo
instrucciones_opt → lista_instrucciones	si_instr: LInstr → InstrOpt
instrucciones_opt → ε	no_instr: → InstrOpt
lista_instrucciones → lista_instrucciones instruccion	muchas_instr: LInstr x Instr→ LInstr
lista_instrucciones → instruccion	una_instr: Instr → LInstr
instruccion → @ E	instr_eval: Exp → Instr
instruccion → if E programa	instr_if: Exp x Prog → Instr
instruccion → if E programa else programa	instr_else: Exp x Prog x Prog → Instr
instruccion → while E bloque	instr_wh: Exp x Prog → Instr

instruccion → read E	instr_rd: Exp → Instr
instruccion → write E	instr_wr: Exp → Instr
instruccion → nl	instr_nl: → Instr
instruccion → new E	instr_new: Exp → Instr
instruccion → delete E	instr_del: Exp → Instr
instruccion → call id parametros_reales	instr_call: **string** x ParamR → Instr
instruccion → programa	instr_comp: Prog → Instr
parametros_reales → lista_parametros_reales	si_param_re: LParamR → ParamR
parametros_reales → ε	no_param_re: → ParamR
lista_parametros_reales → lista_parametros_reales E	muchos_param_re: LParamR x Exp → LParamR
lista_parametros_reales → E	un_param_re: Exp → LParamR
E → E = E	asig: Exp x Exp → Exp
E → E > E	mayor: Exp x Exp → Exp
E → E >= E	mayor_igual: Exp x Exp → Exp
E → E < E	menor: Exp x Exp → Exp
E → E <= E	menor_igual: Exp x Exp → Exp
E → E == E	igual: Exp x Exp → Exp
E → E != E	distinto: Exp x Exp → Exp
E → E + E	suma: Exp x Exp → Exp
E → E - E	resta: Exp x Exp → Exp
E → E and E	and: Exp x Exp → Exp
E → E or E	or: Exp x Exp → Exp
E → E * E	mul: Exp x Exp → Exp
E → E / E	div: Exp x Exp → Exp
E → E % E	mod: Exp x Exp → Exp

E → - E	menos: Exp → Exp
E → not E	not: Exp → Exp
E → E [E]	index: Exp x Exp → Exp
E → E .id	reg: Exp x **string** → Exp
E → E ^	indir: Exp → Exp
E → literal_ent	literal_ent: **string** → Exp
E → literal_real	literal_real: **string** → Exp
E → literal_cadena	literal_cadena: **string** → Exp
E → true	true: → Exp
E → false	false: → Exp
E → id	id: **string** → Exp
E → null	null: → Exp

2. Especificación del constructor de ASTs ascendente

programa → { declaraciones_opt instrucciones_opt }

programa.a = **prog**(declaraciones_opt.a, instrucciones_opt.a)

declaraciones_opt → lista_declaraciones &&

declaraciones_opt.a = **si_decs**(lista_declaraciones.a)

declaraciones_opt → ε

declaraciones_opt.a = **no_decs()**

lista_declaraciones → lista_declaraciones ; declaracion

$lista\_declaraciones_0$.a=**muchas_decs**($lista\_declaraciones_1$.a, declaracion.a)

lista_declaraciones → declaracion

lista_declaraciones.a = **una_dec**(declaracion.a)

declaracion → declaracion_variable

declaracion.a = declaracion_variable.a

declaracion → declaracion_tipo

declaracion.a = declaracion_tipo.a

declaracion → declaracion_proc
    declaracion.a = declaracion_proc.a
declaracion_variable → tipo id
    declaracion_variable.a = **dec_id**(tipo.a, id.lex)
declaracion_tipo → type tipo id
    declaracion_tipo.a = **dec_type**(tipo.a, id.lex)
declaracion_proc → proc id (parametros_formales) programa
    declaracion_proc.a = **dec_proc**(id.lex, parametros_formales.a,
    programa.a)
parámetros_formales → lista_parametros
    parámetros_formales.a = **si_parF**(lista_parametros.a)
parametros_formales → ε
    parámetros_formales.a = **no_parF**()
lista_parametros → lista_parametros , parametro
    $lista\_parametros_0$.a = **muchos_param**($lista\_parametros_1$.a, parametro.a)
lista_parametros → parametro
    lista_parametros.a = **un_param(**parametro.a**)**
parametro → tipo id
    parametro.a = **param_cop**(tipo.a, id.lex)
parametro → tipo & id
    parametro.a = **param_ref**(tipo.a, id.lex)
tipo → tipo[literal_ent]
    tipo.a = **tipo_array**(tipo1.a, literal_ent.lex)
tipo → tipo1
    tipo.a = tipo1.a
tipo1 → tipo2
    tipo1.a = tipo2.a
tipo2 → ^ tipo1
    tipo2.a = **tipo_punt**(tipo1.a)
tipo2 → int
    tipo2.a = **tipo_int**()
tipo2 → real

tipo2.a = **tipo_real**()

tipo2 → bool

tipo2.a = **tipo_bool**(bool.lex)

tipo2 → string

tipo2.a = **tipo_string**()

tipo2 → id

tipo2.a = **tipo_id**(id.lex)

tipo2 → tipo_struct

tipo2.a = tipo_struct.a

tipo_struct → struct { lista_struct }

tipo_struct.a = **tipo_struct**(lista_struct.a)

lista_struct → lista_struct , campo

$lista\_struct_0$.a = **lista_struct**($lista\_struct_1$.a, campo.a)

lista_struct → campo

lista_struct.a = **info_struct**(campo.a)

campo → tipo id

campo.a = **campo**(tipo.a, id.lex)

instrucciones_opt → lista_instrucciones

instrucciones_opt.a = **si_instr**(lista_instrucciones.a)

instrucciones_opt → ε

instrucciones_opt.a = **no_instr()**

lista_instrucciones → lista_instrucciones ; instruccion

$lista\_instrucciones_0$.a =**muchas_instr**($lista\_instrucciones_1$.a, instruccion.a)

lista_instrucciones → instruccion

lista_instrucciones.a = **una_instr**(instruccion.a)

instruccion → instruccion_eval

instruccion.a = instruccion_eval.a

instruccion → instruccion_if

instruccion.a = instruccion_if.a

instruccion → instruccion_while

instruccion.a = instruccion_while.a

instruccion → instruccion_read

instruccion.a = instruccion_read.a

instruccion → instruccion_write

instruccion.a = instruccion_write.a

instruccion → instruccion_nl

instruccion.a = instruccion_nl.a

instruccion → instruccion_reserva

instruccion.a = instruccion_reserva.a

instruccion → instruccion_libera

instruccion.a = instruccion_libera.a

instruccion → instruccion_call

instruccion.a = instruccion_call.a

instruccion → instruccion_compuesta

instruccion.a = instruccion_compuesta.a

instruccion_eval → @ E0

instruccion_eval.a = **instr_eval**(E0.a)

instruccion_if → if E0 programa

instruccion_if.a = **instr_if**(E0.a, programa.a)

instruccion_if → if E0 programa else programa

instruccion_if.a = **instr_else**(E0.a, programa.a, programa.a)

instruccion_while → while E0 programa

instruccion_while.a = **instr_wh**(E0.a, programa.a)

instruccion_read → read E0

instruccion_read.a = **instr_rd**(E0.a)

instruccion_write → write E0

instruccion_write.a = **instr_wr**(E0.a)

instruccion_nl → nl

instruccion_nl.a = **instr_nl**()

instruccion_reserva → new E0

instruccion_reserva.a = **instr_new**(E0.a)

instruccion_libera → delete E0

instruccion_libera.a = **instr_del**(E0.a)

instruccion_call → call id (parametros_reales)

instruccion_call.a = **instr_call**(id.lex, parametros_reales.a)

parametros_reales → lista_parametros_reales

parametros_reales.a = **si_param_re(**lista_parametros_reales.a)

parametros_reales → ε

parametros_reales.a = **no_param_re**()

lista_parametros_reales → lista_parametros_reales , E0

lista_parametros_reales.a = **muchos_param_re(**
lista_parametros_reales.a, E0.a)

lista_parametros_reales → E0

lista_parametros_reales.a = **un_param_re**(E0.a)

instruccion_compuesta → programa

instruccion_compuesta.a = **instr_comp**(programa.a)

E0 → E1 = E0

$E0_0$.a = **igual**(E1.a, $E0_1$.a)

E0 → E1

E0.a = E1.a

E1 → E1 op_relacional E2

$E1_0$.a = **op_rel**(op_relacional.op, $E1_1$.a, E2.a)

fun op_rel(op, opnd1, opnd2):

op = “>” → **return mayor**(opnd1, opnd2)

op = “>=” → **return mayor_igual**(opnd1, opnd2)

op = “<” → **return menor**(opnd1, opnd2)

op = “<=” → **return menor_igual**(opnd1, opnd2)

op = “==” → **return igual**(opnd1, opnd2)

op = “!=” → **return distinto**(opnd1, opnd2)

E1 → E2

E1.a = E2.a

E2 → E2 + E3

$E2_0$.a = **suma**($E2_1$.a, E3.a)

E2 → E3 - E3

E2.a = **resta**($E3_0$.a, $E3_1$.a)

E2 → E3

E2.a = E3.a

E3 → E3 and E4

$E3_0$.a = **and**($E3_1$.a, E4.a)

E3 → E4 or E4

E3.a =**or**($E4_0$.a, E_1.a)

E3 → E4

E3.a = E4.a

E4 → E4 op_nivel4 E5

$E4_0$.a = **op_mul**(op_nivel4.op, $E4_1$.a, E5.a)

fun op_mul(op,opnd1,opnd2):

op = "*" → **return mul**(opnd1,opnd2)

op = "/" → **return div**(opnd1,opnd2)

op = "%" → **return mod**(opnd1,opnd2)

E4 → E5

E4.a = E5.a

E5 → op_nivel5 E5

$E5_0$.a = **op_inv**(op_nivel5.op, $E5_1$.a)

E5 → E6

E5.a = E6.a

fun op_inv(op,opnd):

op = "-" → **return menos**(opnd)

op = "not" → **return not**(opnd)

E6 → E6 [E0]

$E6_0$.a = **index**($E6_1$.a, E0.a)

E6 → E6^

$E6_0$.a = **indir**($E6_1$.a)

E6 → E6 .id

$E6_0$.a = **reg**($E6_1$.a, id.lex)

E6 → E7

E6.a = E7.a

E7 → (E0)

E7.a = E0.a

E7 → op_basico
    E7.a = op_basico.a
op_relacional → >
    op_relacional.op = “>”
op_relacional → >=
    op_relacional.op = “>=”
op_relacional → <
    op_relacional.op = “<”
op_relacional → <=
    op_relacional.op = “<=”
op_relacional → ==
    op_relacional.op = “==”
op_relacional → !=
    op_relacional.op = “!=”
op_nivel4 → *
    op_nivel4.op = “*”
op_nivel4 → /
    op_nivel4.op = “/”
op_nivel4 → %
    op_nivel4.op = “%”
op_nivel5 → -
    op_nivel5.op = “-”
op_nivel5 → not
    op_nivel5.op = “not”
op_basico → literal_ent
    op_basico.a = **literal_ent**(literal_ent.lex)
op_basico → literal_real
    op_basico.a = **literal_real**(literal_real.lex)
op_basico → true
    op_basico.a = true()
op_basico → false
    op_basico.a = false()

op_basico → literal_cadena

op_basico.a = literal_cadena(literal_cadena.lex)

op_basico → id

op_basico.a = id(id.lex)

op_basico → null

op_basico.a = null()

3. Acondicionamiento para la implementación descendente

factorización

eliminación de recursión a izquierdas

factorización + eliminación de recursión a izquierdas

programa → { declaraciones_opt instrucciones_opt }

programa.a = **prog**(declaraciones_opt.a, instrucciones_opt.a)

declaraciones_opt → lista_declaraciones &&

declaraciones_opt.a = **si_decs**(lista_declaraciones.a)

declaraciones_opt → ε

declaraciones_opt.a = **no_decs()**

lista_declaraciones → declaracion lista_declaraciones_re

lista_declaraciones_re.ah = **una_dec(**declaracion.a**)**

lista_declaraciones.a = lista_declaraciones_re.a

lista_declaraciones_re → ; declaracion lista_declaraciones_re

$lista\_declaraciones\_re_1$.ah=**muchas_decs(**$lista\_declaraciones\_re_0$.ah, declaracion.a**)**

$lista\_declaraciones\_re_0$.a = $lista\_declaraciones\_re_1$.a

lista_declaraciones_re → ε

lista_declaraciones_re.a = lista_declaraciones_re.ah

declaracion → declaracion_variable

declaracion.a = declaracion_variable.a

declaracion → declaracion_tipo

declaracion.a = declaracion_tipo.a

declaracion → declaracion_proc

declaracion.a = declaracion_proc.a

declaracion_variable → tipo id

declaracion_variable.a = **dec_id**(tipo.a, id.lex)

declaracion_tipo → type tipo id

declaracion_tipo.a = **dec_type**(tipo.a, id.lex)

declaracion_proc → proc id (parametros_formales) programa

declaracion_proc.a = **dec_proc**(id.lex, parametros_formales.a, programa.a)

parámetros_formales → lista_parametros

parámetros_formales.a = **si_parF**(lista_parametros.a)

parametros_formales → ε

parámetros_formales.a = **no_parF**()

lista_parametros → parametro lista_parametros_re

lista_parametros_re.ah = **un_param**(parametro.a)

lista_parametros.a = lista_parametros_re.a

lista_parametros_re → , parametro lista_parametros_re

lista_parametros_re$_1$.ah=**muchos_param(**lista_parametros_re$_0$.ah, parametro.a**)**

lista_parametros_re$_0$.a=lista_parametros_re$_1$.a

lista_parametros_re → ε

lista_parametros_re.a = lista_parametros_re.ah

parametro → tipo parametro_re

parametro_re.ah = tipo.a

parametro.a = parametro_re.a

parametro_re→ id

parametro_re.a = param_cop(parametro_re.ah, id.lex)

parametro_re→ & id

papametro.a = param_ref(parametro_re.ah, id.lex)

tipo → tipo1 tipo_re

tipo_re.ah = tipo1.a

tipo.a = tipo_re.a

tipo_re → [literal_ent]

tipo_re.a = **tipo_array**(tipo_re.ah,literal_ent.lex)

tipo_re → ε

tipo_re.a = tipo_re.ah

tipo1 → tipo2

tipo1.a = tipo2.a

tipo2 → ^ tipo1

tipo2.a = **tipo_punt**(tipo1.a)

tipo2 → int

tipo2.a = **tipo_int**()

tipo2 → real

tipo2.a = **tipo_real**()

tipo2 → bool

tipo2.a = **tipo_bool**()

tipo2 → string

tipo2.a = **tipo_string**()

tipo2 → id

tipo2.a = **tipo_id**(id.lex)

tipo2 → tipo_struct

tipo2.a = tipo_struct.a

tipo_struct → struct { lista_struct }

tipo_struct.a = **tipo_struct**(lista_struct.a)

lista_struct → campo lista_struct_re

lista_struct_re.ah = **info_struct**(campo.a)

lista_struct.a = lista_struct_re.a

lista_struct_re → , campo lista_struct_re

$lista\_struct\_re_1.ah$ = **lista_struct**($lista\_struct\_re_0.ah$, campo.a)

$lista\_struct\_re_0.a = lista\_struct\_re_1.a$

lista_struct_re → ε

lista_struct_re.a = lista_struct_re.ah

campo → tipo id

campo.a = **campo**(tipo.a, id.lex)

instrucciones_opt → lista_instrucciones

instrucciones_opt.a = **si_instr**(lista_instrucciones.a)

instrucciones_opt → ε

instrucciones_opt.a = **no_instr()**

lista_instrucciones → instruccion lista_instrucciones_re

lista_instrucciones_re.ah = **una_instr**(instruccion.a)

lista_instrucciones.a = lista_instrucciones_re.a

lista_instrucciones_re → ; instruccion lista_instrucciones_re

lista_instrucciones_re$_1$.ah= **muchas_instr**(lista_instrucciones_re$_0$.ah, instruccion.a)

lista_instrucciones_re$_0$.a = lista_instrucciones_re$_1$

lista_instrucciones_re → ε

lista_instrucciones_re.a = lista_instrucciones_re.ah

instruccion → instruccion_eval

instruccion.a = instruccion_eval.a

instruccion → instruccion_if

instruccion.a = instruccion_if.a

instruccion → instruccion_while

instruccion.a = instruccion_while.a

instruccion → instruccion_read

instruccion.a = instruccion_read.a

instruccion → instruccion_write

instruccion.a = instruccion_write.a

instruccion → instruccion_nl

instruccion.a = instruccion_nl.a

instruccion → instruccion_reserva

instruccion.a = instruccion_reserva.a

instruccion → instruccion_libera

instruccion.a = instruccion_libera.a

instruccion → instruccion_call

instruccion.a = instruccion_call.a

instruccion → instruccion_compuesta

instruccion.a = instruccion_compuesta.a

instruccion_eval → @ E0

instruccion_eval.a = **instr_eval**(E0.a)

instruccion_if → **if** E0 programa instruccion_if_re

instruccion_if_re.exp = E0.a

instruccion_if_re.prog = programa.a

instruccion_if.a = instruccion_if_re.a

instruccion_if_re → **else** programa

instruccion_if_re.a = **instr_else**(instruccion_if_re.exp, instruccion_if_re.prog, programa.a)

instruccion_if_re → ε

instruccion_if_re.a = **instr_if**(instruccion_if_re.exp, instruccion_if_re.prog)

instruccion_while → while E0 programa

instruccion_while.a = **instr_wh**(E0.a, programa.a)

instruccion_read → read E0

instruccion_read.a = **instr_rd**(E0.a)

instruccion_write → write E0

instruccion_write.a = **instr_wr**(E0.a)

instruccion_nl → nl

instruccion_nl.a = **instr_nl**()

instruccion_reserva → new E0

instruccion_reserva.a = **instr_new**(E0.a)

instruccion_libera → delete E0

instruccion_libera.a = **instr_del**(E0.a)

instruccion_call → call id (parametros_reales)

instruccion_call.a = **instr_call**(id.lex, parametros_reales.a)

parametros_reales → lista_parametros_reales

parametros_reales.a = **si_param_re(**lista_parametros_reales.a)

parametros_reales → ε

parametros_reales.a = **no_param_re**()

lista_parametros_reales → E0 lista_parametros_reales_re

lista_parametros_reales_re_0.ah = **un_param_re(**
lista_parametros_reales_re_1.ah, E0.a)
lista_parametros_reales_re_0.a = lista_parametros_reales_re_1.a

lista_parametros_reales_re → , E0 lista_parametros_reales_re

lista_parametros_reales_re_1.ah = **muchos_param_re**(
lista_parametros_reales_re_0.ah, E0.a)
lista_parametros_reales_re_0.a = lista_parametros_reales_re_1.a

lista_parametros_reales_re → ε

lista_parametros_reales_re.a = lista_parametros_reales_re.ah

instruccion_compuesta → programa
instruccion_compuesta.a = **instr_comp**(programa.a)

E0 → E1 E0RE
E0RE.ah = E1.a
E0.a = E0RE.a

E0RE → = E0
E0RE.a = **asig**(E0RE.ah, E0.a)

E0RE→ ε
E0RE.a = E0RE.ah

E1 → E2 E1RE
E1RE.ah = E2.a
E.1.a = E1RE.a

E1RE → op_relacional E2 E1RE
E1RE$_1$.ah = **op_rel**(op_relacional.op, E1RE$_0$.ah, E2.a)

E1RE → ε
E1RE.a = E1RE.ah

fun op_rel(op, opnd1, opnd2):
op = “>” → **return mayor**(opnd1, opnd2)
op = “>=” → **return mayor_igual**(opnd1, opnd2)
op = “<” → **return menor**(opnd1, opnd2)
op = “<=” → **return menor_igual**(opnd1, opnd2)
op = “==” → **return igual**(opnd1, opnd2)

op = “!=” → **return distinto**(opnd1, opnd2)

E2 → E3 E2RE E2RE’

E2RE.ah = E3.a

E2RE’.ah = E2RE.a

E2.a = E2RE’.a

E2RE’→ + E3 E2RE’

$E2RE'_1$.ah = **suma**($E2RE'_0$.ah, E3.a)

E2RE’→ ε

E2RE’.a = E2RE’.ah

E2RE→ - E3

E2RE.a = **resta**(E2RE.ah, E3.a)

E2RE→ ε

E2RE.a = E2RE.ah

E3 → E4 E3RE

E3RE.ah = E4.a

E3.a = E3RE.a

E3RE → **and** E3

E3RE.a = **and**(E3RE.ah,E3.a)

E3RE → **or** E4

E3RE.a = **or**(E3RE.ah, E3.a)

E3RE → ε

E3RE.a = E3RE.ah

E4 → E5 E4RE

E4RE.ah = E5.a

E4.a = E4RE.a

E4RE → op_nivel4 E5 E4RE

$E4RE_1$.ah = **op_mul**(op_nivel4.op, $E4RE_0$ .ah, E5.a)

E4RE0 .a = E4RE1.a

E4RE → ε

E4RE.a = E4RE.ah

fun op_mul(op,opnd1,opnd2):

op = "*" → **return mul**(opnd1,opnd2)

op = "/" → **return div**(opnd1,opnd2)

op = "%" → **return mod**(opnd1,opnd2)

E5 → op_nivel5 E5

$E5_0$.a = **op_inv**(op_nivel5.op, $E5_1$.a)

E5 → E6

E5.a = E6.a

fun op_inv(op,opnd):

op = "-" → **return menos**(opnd)

op = "not" → **return not**(opnd)

E6 → E7 E6RE

E6RE.ah = E7.a

E6.a = E6RE.a

E6RE → [E0] E6RE

$E6RE_1$.ah = **index**($E6RE_0$.ah, E0.a)

$E6RE_0$.a = $E6RE_1$.a

E6RE →^ E6RE

$E6RE_1$.ah = **indir**($E6RE_0$.ah)

$E6RE_0$.a = $E6RE_1$.a

E6RE → .id E6RE

$E6RE_1$.ah = **reg**($E6RE_0$.ah, id.lex)

$E6RE_0$.a = $E6RE_1$.a

E6RE → ε

E6RE.a = E6RE.ah

E7 → (E0)

E7.a = E0.a

E7 → op_basico

E7.a = op_basico.a

op_relacional → >

op_relacional.op = “>”

op_relacional → >=

op_relacional.op = “>=”

op_relacional → <

op_relacional.op = “<”

op_relacional → <=

op_relacional.op = “<=”

op_relacional → ==

op_relacional.op = “==”

op_relacional → !=

op_relacional.op = “!=”

op_nivel4 → *

op_nivel4.op = “*”

op_nivel4 → /

op_nivel4.op = “/”

op_nivel4 → %

op_nivel4.op = “%”

op_nivel5 → -

op_nivel5.op = “-”

op_nivel5 → not

op_nivel5.op = “not”

op_basico → literal_ent

op_basico.a = **literal_ent**(literal_ent.lex)

op_basico → literal_real

op_basico.a = **literal_real**(literal_real.lex)

op_basico → true

op_basico.a = true()

op_basico → false

op_basico.a = false()

op_basico → literal_cadena

op_basico.a = **literal_cadena**(literal_cadena.lex)

op_basico → id

op_basico.a = id(id.lex)

op_basico → null

op_basico.a = null()

4. Especificación del procesamiento

```
imprime(prog(Prog)):
        imprime(Prog)
        print "<EOF>"
        nl
imprime(prog(Decs , InstrOpt )):
        print "{"
        nl
        imprime(Decs)
        imprime(InstrOpt)
        print "}"
        nl
imprime(si_decs(LDecs )):
        imprime(LDecs )
        print "&&"
        nl
imprime(no_decs()): noop
imprime(muchas_decs(LDecs , Dec):
        imprime(LDecs)
        print ";"
        nl
        imprime(Dec)
imprime(una_dec(Dec):
        imprime(Dec)
imprime(dec_id(Tipo, Id)):
        imprime(Tipo)
        print Id
        nl
imprime(dec_type(Tipo, Id)):
        print "<type>"
        nl
```

```
        imprime(Tipo)
        print " "
        nl
imprime(dec_proc(Id, ParamF, Prog)):
        print "<proc>"
        nl
        print Id
        nl
        print "("
        nl
        imprime(ParamF)
        print ")"
        nl
        imprime(Prog)
imprime(tipo_array(Tipo, String)):
        imprime(tipo)
        print "["
        nl
        print Ent
        nl
        print  "]"
        nl
imprime(tipo_punt(Tipo):
        print "^"
        nl
        print Tipo
imprime(tipo_int()):
        print "<int>"
        nl
imprime(tipo_real()):
        print "<real>"
        nl
```

```
imprime(tipo_string()):
        print “<string>”
        nl

imprime(tipo_bool()):
        print "<bool>"
        nl
imprime(tipo_id(Id)):
        print Id
        nl

imprime(tipo_struct(LStruct)):
        print "<struct>"
        nl
        print "{"
        nl
        imprime(LStruct)
        print "}"
        nl
imprime(lista_struct(LStruct , Campo)):
        imprime(LStruct)
        print ","
        nl
        imprime(Campo)
imprime(info_struct(Campo)):
        imprime(Campo)
imprime(campo(Tipo, Id)):
        imprime(Tipo)
        print Id
        nl
imprime(si_parF(LParam)):
        imprime(LParam)
```

```
imprime(no_parF()): noop
imprime(muchos_param(LParam, Param)):
        imprime(LParam)
        print ","
        nl
        imprime(Param)
imprime(un_param(Param)):
        imprime(Param)
imprime(param_cop(Tipo, Id)):
        imprime(Tipo)
        print Id
        nl
imprime(param_ref(Tipo, Id)):
        imprime(Tipo)
        print "&"
        nl
        print Id
        nl
imprime(si_instr(LInstr)):
        imprime(LInstr )
imprime(no_instr()): noop
imprime(muchas_instr(LInstr , Exp):
        imprime(LInstr)
        print ";"
        nl
        imprime(Exp)
imprime(una_instr(Exp):
        imprime(Exp)
imprime(instr_eval(Exp)):
        print "@"
        nl
        imprime(Exp)
```

```
imprime(instr_if(Exp, Bloque)):
        print "<if>"
        nl
        imprime(Exp)
        imprime(Bloque)
imprime(instr_else(Exp, Bloque, Bloque)):
        print "<if>"
        nl
        imprime(Exp)
        imprime(Bloque)
        print "<else>"
        nl
        imprime(Bloque)
imprime(instr_wh(Exp, Bloque)):
        print "<while>"
        nl
        imprime(Exp)
        imprime(Bloque)
imprime(instr_rd(Exp)):
        print "<read>"
        nl
        imprime(Exp)
imprime(instr_wr(Exp)):
        print "<write>"
        nl
        imprime(Exp)
imprime(instr_nl):
        nl
imprime(instr_new(Exp)):
        print "<new>"
        nl
        imprime(Exp)
```

```
imprime(instr_del(Exp)):
        print "<delete>"
        nl
        imprime(Exp)
imprime(instr_call(Id, ParamR )):
        print "<call>"
        nl
        print Id
        nl
        print "("
        nl
        imprime(ParamR )
        print ")"
        nl
imprime(instr_comp(Bloque)):
        imprime(Bloque)
imprime(si_param_re(LParamR)):
        imprime(LParamR )
imprime(no_param_re()): noop
imprime(muchos_param_re(LParamR , Exp):
        imprime(LParamR )
        print ","
        nl
        imprime(Exp)
imprime(un_param_re(Exp):
        imprime(Exp)
imprime(asig(Opnd0, Opnd1)):
        imprimeExpBin(Opnd0, "=", Opnd1,1,0)
imprime(mayor(Opnd0, Opnd1)):
        imprimeExpBin(Opnd0, ">", Opnd1,1,2)
imprime(menor(Opnd0, Opnd1)):
        imprimeExpBin(Opnd0, "<", Opnd1,1,2)
```

```
imprime(mayor_igual(Opnd0, Opnd1)):
        imprimeExpBin(Opnd0, ">=", Opnd1,1,2)
imprime(menor_igual(Opnd0, Opnd1)):
        imprimeExpBin(Opnd0, "<=", Opnd1,1,2)
imprime(igual(Opnd0, Opnd1)):
        imprimeExpBin(Opnd0, "==", Opnd1,1,2)
imprime(distinto(Opnd0, Opnd1)):
        imprimeExpBin(Opnd0, "!=", Opnd1,1,2)
imprime(suma(Opnd0, Opnd1)):
        imprimeExpBin(Opnd0, "+", Opnd1,2,3)
imprime(resta(Opnd0, Opnd1)):
        imprimeExpBin(Opnd0, "-", Opnd1,3,3)
imprime(and(Opnd0, Opnd1)):
        imprimeExpBin(Opnd0, "and", Opnd1,4,3)
imprime(or(Opnd0, Opnd1)):
        imprimeExpBin(Opnd0, "or", Opnd1,4,4)
imprime(mul(Opnd0, Opnd1)):
        imprimeExpBin(Opnd0, "*", Opnd1,4,5)
imprime(mod(Opnd0, Opnd1)):
        imprimeExpBin(Opnd0, "%", Opnd1,4,5)
imprime(div(Opnd0, Opnd1)):
        imprimeExpBin(Opnd0, "/", Opnd1,4,5)
imprime(menos(Opnd)):
        imprimeExpPre(Opnd, "-", 5)
imprime(not(Opnd)):
        imprimeExpPre(Opnd, "not", 5)
imprime(index(Opnd0, Opnd1)):
        imprimeOpnd(Opnd0, 6)
        print "["
        nl
        imprimeOpnd(Opnd1, 6)
        print "]"
```

```
        nl

imprime(reg(Opnd, Id)):
        imprimeOpnd(Opnd, 6)
        print "."
        nl
        print Id
        nl
imprime(indir(Opnd)):
        print ^
        nl
        print Opnd
        nl
imprimeExpBin(Opnd0, Op, Opnd1, np0, np1):
        imprimeOpnd(Opnd0, np0)
        print Op
        nl
        imprimeOpnd(Opnd1, np1)
imprimeExpPre(Opnd, Op, np):
        print Op
        nl
        imprimeOpnd(Opnd, np)
imprimeOpnd(Opnd,MinPrior):
        if prioridad(Opnd) < MinPrior
        print “(“
        nl
        end if
        imprime(Opnd)
        if prioridad(Opnd) < MinPrior
        print “)“
        nl
        end if
```

```
prioridad(asig(_,_)): return 0
prioridad(mayor(_,_)): return 1
prioridad(menor(_,_)): return 1
prioridad(mayor_igual(_,_)): return 1
prioridad(menor_igual(_,_)): return 1
prioridad(igual(_,_)): return 1
prioridad(distinto(_,_)): return 1
prioridad(suma(_,_)): return 2
prioridad(resta(_,_)): return 2
prioridad(and(_,_)): return 3
prioridad(or(_,_)): return 3
prioridad(mod(_,_): return 4
prioridad(mul(_,_)): return 4
prioridad(div(_,_)): return 4
prioridad(menos unario(_)): return 5
prioridad(not(_)): return 5
prioridad(index(_,_)): return 6
prioridad(reg(_,_)): return 6
prioridad(indir(_)): return 6
imprime(literal_real(R)):
        print R
        nl
imprime(literal_ent(Ent)):
        print Ent
        nl
imprime(true()):
        print "<true>"
        nl
imprime(false()):
        print "<false>"
        nl
imprime(literal_cadena(S)):
```

```
        print S
        nl
imprime(id(Id)):
        print Id
        nl
imprime(null()):
        print "<null>"
        nl
```