

PRÁCTICA PL: TERCERA FASE I

Tiny(0)

Integrantes:

David Davó Laviña

Ela Katherine Shepherd Arévalo

Grupo 12

ESPECIFICACIÓN SINTAXIS ABSTRACTA

Géneros: Prog(programa), Exp (expresión), Decs (declaraciones), Dec (declaración), Insts (instrucciones), Inst (instrucción), Tipo (tipo)

prog: Decs \times Insts \rightarrow Prog
dec_una: Dec \rightarrow Decs
decs_muchas: Decs \times Dec \rightarrow Decs
dec: Tipo \times string \rightarrow Dec
int: \rightarrow Tipo
realw: \rightarrow Tipo
bool: \rightarrow Tipo
inst_una: Inst \rightarrow Insts
insts_muchas: Insts \times Inst \rightarrow Insts
inst: string \times Exp \rightarrow Inst
suma: Exp \times Exp \rightarrow Exp
resta: Exp \times Exp \rightarrow Exp
and: Exp \times Exp \rightarrow Exp
or: Exp \times Exp \rightarrow Exp
menor: Exp \times Exp \rightarrow Exp
men_ig: Exp \times Exp \rightarrow Exp
mayor: Exp \times Exp \rightarrow Exp
may_ig: Exp \times Exp \rightarrow Exp
igual: Exp \times Exp \rightarrow Exp
desigual: Exp \times Exp \rightarrow Exp
mul: Exp \times Exp \rightarrow Exp
div: Exp \times Exp \rightarrow Exp
m_unario: Exp \rightarrow Exp
not: Exp \rightarrow Exp
entero: string \rightarrow Exp
real: string \rightarrow Exp
variable: string \rightarrow Exp
verdadero: \rightarrow Exp
falso: \rightarrow Exp

CONSTRUCTOR AST (GRAMÁTICA S-ATRIBUIDA)

El atributo “a” sintetizado se refiere al árbol de sintaxis abstracto de ese nodo, “lex” es el atributo léxico de los terminales, y “op” es un atributo que se refiere a un operador

Programa \rightarrow Decs && Insts

Programa.a = prog(Decs.a, Insts.a)

Decs -> Dec
 Decs.a = dec_una(Dec.a)

Decs -> Decs ; Dec
 Decs₀.a = decs_muchas(Decs₁.a, Dec.a)

Dec -> TypN VarN
 Dec0.a = dec(TypN.a, VarN.var)

Insts -> Inst
 Insts.a = inst_una(Inst.a)

Insts -> Insts ; Inst
 Insts₀.a = insts_muchas(Insts₁.a, Inst.a)

Inst -> VarN = E0
 Inst.a = inst(VarN.var, E0.a)

TypN -> int
 TypN.a = int()

TypN -> real
 TypN.a = realw()

TypN -> bool
 TypN.a = bool()

VarN -> Variable
 VarN.var = Variable.lex

E0 -> E1 OpIn0AsocD E0
 E0₀.a = opera_dos(OpIn0AsocD.op, E1.a, E0₁.a)

E0 -> E1 OpIn0NoAsoc E1
 E0.a = opera_dos(OpIn0NoAsoc.op, E1₀.a, E1₁.a)

E0 -> E1
 E0.a = E1.a

E1 -> E1 OpIn1AsocI E2
 E1₀.a = opera_dos(OpIn1AsocI.op, E1₁.a, E2.a)

E1 -> E2
 E1.a = E2.a

E2 -> E2 OpIn2AsocI E3
 E2₀.a = opera_dos(OpIn2AsocI.op, E2₁.a, E3.a)

E2 -> E3
 E2.a = E3.a

E3 -> E4 OpIn3NoAsoc E4
 E3.a = opera_dos(OpIn3NoAsoc.op, E4₀.a, E4₁.a)

E3 -> E4
 E3.a = E4.a

E4 -> OpPre4NoAsoc E5
 E4.a = opera_uno(OpPre4NoAsoc.op, E5.a)

E4 -> OpPre4Asoc E4
 E4₀.a = opera_uno(OpPre4Asoc.op, E4.a)

E4 -> E5
 E4.a = E5.a

```

E5 -> ( E0 )
    E5.a = E0.a
E5 -> LitEnt
    E5.a = entero(LitEnt.lex)
E5 -> LitReal
    E5.a = real(LitReal.lex)
E5 -> true
    E5.a = verdadero()
E5 -> false
    E5.a = falso()
E5 -> Variable
    E5.a = variable(Variable.lex)
OpIn0AsocD -> +
    OpIn0AsocD.op = '+'
OpIn0NoAsoc-> -
    OpIn0NoAsoc.op = '-'
OpIn1AsocI -> and
    OpIn1AsocI.op = 'and'
OpIn1AsocI -> or
    OpIn1AsocI.op = 'or'
OpIn2AsocI -> <
    OpIn2AsocI.op = '<'
OpIn2AsocI -> <=
    OpIn2AsocI.op = '<='
OpIn2AsocI -> >
    OpIn2AsocI.op = '>'
OpIn2AsocI -> >=
    OpIn2AsocI.op = '>='
OpIn2AsocI -> ==
    OpIn2AsocI.op = '=='
OpIn2AsocI -> !=
    OpIn2AsocI.op = '!='
OpIn3NoAsoc -> *
    OpIn3NoAsoc.op = '*'
OpIn3NoAsoc -> /
    OpIn3NoAsoc.op = '/'
OpPre4NoAsoc -> -
    OpPre4NoAsoc.op = '-'
OpPre4Asoc -> not
    OpPre4Asoc.op = 'not'

```

Funciones semánticas

```
fun opera_uno(Op, Arg){
  switch Op
    case '-': return m_unario(Arg)
    case 'not': return not(Arg)
}

fun opera_dos(Op, Arg0, Arg1){
  switch Op
    case '+': return suma(Arg0, Arg1)
    case '-': return resta(Arg0, Arg1)
    case 'and': return and(Arg0, Arg1)
    case 'or': return or(Arg0, Arg1)
    case '<': return menor(Arg0, Arg1)
    case '<=': return men_ig(Arg0, Arg1)
    case '>': return mayor(Arg0, Arg1)
    case '>=': return may_ig(Arg0, Arg1)
    case '==': return igual(Arg0, Arg1)
    case '!=': return desigual(Arg0, Arg1)
    case '*': return mul(Arg0, Arg1)
    case '/': return div(Arg0, Arg1)
}
```

ACONDICIONAMIENTO

En el paso anterior, las reglas en **negrita** tienen recursión a izquierdas y las reglas en *cursiva* tienen un factor común. Añadimos un atributo heredado cuando realizamos las transformaciones.

Programa -> Decs && Insts

Programa.a = prog(Decs.a, Insts.a)

Decs -> Dec RDecs

Decs.a = RDecs.a

RDecs.ah = dec_una(Dec.a)

RDecs -> ; Dec RDecs

RDecs₀.a = RDecs₁.a

RDecs₁.ah = decs_muchas(RDecs₀.ah, Dec.a)

RDecs -> ε

RDecs.a = RDecs.ah

Dec -> TypN VarN

Dec0.a = dec(TypN.a, VarN.var)

Insts -> Inst RInsts

Insts.a = RInsts.a

RInsts.ah = inst_una(Inst.a)

```

RInsts -> ; Inst RInsts
    RInsts0.a = RInsts1.a
    RInsts1.ah = insts_muchas(RInsts0.ah, Inst.a)
RInsts -> ε
    RInsts.a = RInsts.ah
Inst -> VarN = E0
    Inst.a = inst(VarN.var, E0.a)
TypN -> int
    TypN.a = int()
TypN -> real
    TypN.a = realw()
TypN -> bool
    TypN.a = bool()
VarN -> Variable    VarN.var = Variable.lex
E0 -> E1 RE0
    E0.a = RE0.a
    RE0.ah = E1.a
RE0 -> OpIn0AsocD E0
    RE0.a = opera_dos(OpIn0AsocD.op, RE0.ah, E0.a)
RE0 -> OpIn0NoAsoc E1
    RE0.a = opera_dos(OpIn0NoAsoc.op, RE0.ah, E1.a)

RE0 -> ε
    RE0.a = RE0.ah
E1 -> E2 RE1
    E1.a = RE1.a
    RE1.ah = E2.a
RE1 -> OpIn1AsocI E2 RE1
    RE10.a = RE11.a
    RE11.ah = opera_dos(OpIn1AsocI.op, RE10.ah ,E2.a)
RE1 -> ε
    RE1.a = RE1.ah
E2 -> E3 RE2
    E2.a = RE2.a
    RE2.ah = E3.a
RE2 -> OpIn2AsocI E3 RE2
    RE20.a = RE21.a
    RE21.ah = opera_dos(OpIn2AsocI.op, RE20.ah ,E3.a)
RE2 -> ε
    RE2.a = RE2.ah
E3 -> E4 RE3
    E3.a = RE3.ah
    RE3.ah = E4.a

```

RE3 -> OpIn3NoAsoc E4
 RE3.a = opera_dos(OpIn3NoAsoc.op, RE3.ah, E4.a)
 RE3 -> ε
 RE3.a = RE3.ah
 E4 -> OpPre4NoAsoc E5
 E4.a = opera_uno(OpPre4NoAsoc.op, E5.a)
 E4 -> OpPre4Asoc E4
 E4₀.a = opera_uno(OpPre4Asoc.op, E4.a)
 E4 -> E5
 E4.a = E5.a
 E5 -> (E0)
 E5.a = E0.a
 E5 -> LitEnt
 E5.a = entero(LitEnt.lex)
 E5 -> LitReal
 E5.a = real(LitReal.lex)
 E5 -> true
 E5.a = verdadero()
 E5 -> false
 E5.a = falso()
 E5 -> Variable
 E5.a = variable(Variable.lex)
 OpIn0AsocD -> +
 OpIn0AsocD.op = '+'
 OpIn0NoAsoc -> -
 OpIn0NoAsoc.op = '-'
 OpIn1AsocI -> and
 OpIn1AsocI.op = 'and'
 OpIn1AsocI -> or
 OpIn1AsocI.op = 'or'
 OpIn2AsocI -> <
 OpIn2AsocI.op = '<'
 OpIn2AsocI -> <=
 OpIn2AsocI.op = '<='
 OpIn2AsocI -> >
 OpIn2AsocI.op = '>'
 OpIn2AsocI -> >=
 OpIn2AsocI.op = '>='
 OpIn2AsocI -> ==
 OpIn2AsocI.op = '=='
 OpIn2AsocI -> !=
 OpIn2AsocI.op = '!='
 OpIn3NoAsoc -> *
 OpIn3NoAsoc.op = '*'

OpIn3NoAsoc -> /
 OpIn3NoAsoc.op = '/'
OpPre4NoAsoc -> -
 OpPre4NoAsoc.op = '-'
OpPre4Asoc -> not
 OpPre4Asoc.op = 'not'