PRÁCTICA PL: TERCERA FASE II

Tiny(1)

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 (intrucción), Tipo (tipo), PFs/PF (parámetros formales), Campos (campos), Campo (campo), PR (parámetros reales), PInst (parte de instrucciones), Bloque (bloque)

prog_sin_decs: Insts → Prog
prog_con_decs: Decs × Insts → Prog
dec_una: Dec → Decs
decs_muchas: Decs × Dec → Decs
var: Tipo × string → Decs
type: Tipo × string → Decs
proc: string × PFs × Inst → Decs
param_f_sin: → PFs
param_f_con_una: PF → PFs
param_f_con_muchas: PFs × PF → PFs
param_f_ref: Tipo × string → PF
param_f_noref: Tipo × string → PF
tipo_array: Exp × Tipo → Tipo
tipo_record: Campos → Tipo
tipo_pointer: Tipo → Tipo
tipo_iden: string → Tipo
tipo_int: → Tipo
tipo_real: → Tipo
tipo_bool: → Tipo
tipo_string: → Tipo
campo_uno: Campo → Campos
campo_muchos: Campos × Campo→Campos
campo: Tipo × string → Campo
inst_una: Inst → Insts
insts_muchas: Insts × Inst → Insts
e_igual: Exp × Exp →Inst
if: Exp × PInst →Inst
ifelse: Exp × PInst × PInst→Inst
while: Exp × PInst →Inst
read: Exp →Inst
write: Exp →Inst
nl: →Inst
new: Exp →Inst
delete: Exp →Inst
call: string × PR →Inst
bl: Bloque →Inst

lista_sin: →PInst
lista_con: Insts →PInst
param_r_sin: → PR
param_r_con_una: Exp → PR
param_r_con_muchas: PR × Exp→ PR
bloque_sin: → Bloque
bloque_con: Prog → Bloque
entero: string → Exp
real: string → Exp
cadena: string → Exp
verdadero: → Exp
falso: → Exp
null: → Exp
identificador: string → Exp
suma: Exp × Exp → Exp
resta: Exp × Exp → Exp
and: Exp × Exp → Exp
or: Exp × Exp → Exp
menor: Exp × Exp → Exp
men_ig: Exp × Exp → Exp
mayor: Exp × Exp → Exp
may_ig: Exp × Exp → Exp
igual: Exp × Exp → Exp
desigual: Exp × Exp → Exp
mul: Exp × Exp → Exp
div: Exp × Exp → Exp
modulo: Exp × Exp → Exp
m_unario: Exp → Exp
not: Exp → Exp
indexacion: Exp × Exp → Exp
acc_registro: Exp × string→ Exp
indireccion: Exp → 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 -> Decs Insts
    Programa.a = prog(Decs.a, Insts.a)
Decs -> ε
    Decs.a = null

Decs -> LDecs &&
    Decs.a = LDecs.a
LDecs -> Dec
    LDecs.a = dec_una(Dec.a)
LDecs -> LDecs ; Dec
    $LDecs_0$.a = decs_muchas($LDecs_1$.a, Dec.a)
Dec -> var TypN Identificador
    Dec.a = var(TypN.a, Identificador.lex)
Dec -> type TypN Identificador
    Dec.a = type(TypN.a, Identificador.lex)
Dec -> proc Identificador ParamsF Bloq
    Dec.a = proc(Identificador.lex, ParamsF.a, Bloq.a)
ParamsF -> (LParamFOpc)
    ParamsF.a = LParamFOpc.a
LParamFOpc -> ε
    LParamFOpc.a = param_f_sin()
LParamFOpc -> LParamF
    LParamFOpc.a = LParamF.a
LParamF -> ParamF
    LParamF.a = param_f_con_una(ParamF.a)
LParamF -> LParamF , ParamF
    $LParamF_0$.a = param_f_con_muchas($LParamF_1$.a, ParamF.a)
ParamF -> TypN & Identificador
    ParamF.a = param_f_ref(TypN.a, Identificador.lex)
ParamF -> TypN Identificador
    ParamF.a = param_f_noref(TypN.a, Identificador.lex)
TypN -> BaseType
    TypN.a = BaseType.a
TypN -> array [LitEnt] of BaseType
    TypN.a = tipo_array(LitEnt.a, BaseType.a)
TypN -> record { LCampos }
    TypN.a = tipo_record(LCampos.a)
TypN -> pointer BaseType
    TypN.a = tipo_pointer(BaseType.a)
LCampos -> Campo
    LCampos.a = campo_uno(Campo.a)
LCampos -> LCampos ; Campo
    LCampos.a = campo_muchos(LCampos.a, Campo.a)
Campo -> TypN Identificador
    Campo.a = campo(TypN.a, Identificador.lex)
BaseType -> BasicType
    BaseType.a = BasicType.a
BaseType -> Identificador
    BaseType.a = tipo_iden(Identificador.lex)

BasicType -> int
BasicType.a = tipo_int()
BasicType -> realw
BasicType.a = tipo_real()
BasicType -> bool
BasicType.a = tipo_bool()
BasicType -> string
BasicType.a = tipo_string()
Insts -> Inst
Insts.a = inst_una(Inst.a)
Insts -> Insts ; Inst
$Insts_0$.a = insts_muchas($Insts_1$.a, Inst.a)
Inst -> E0 = E0
Inst.a = e_igual($E0_0$.a, $E0_1$.a)
Inst -> if E0 then LInsts endif
Inst.a = if(E0.a, LInsts.a)
Inst -> if E0 then LInsts else LInsts endif
Inst.a = ifelse(E0.a, $LInsts_0$.a, $LInsts_1$.a)
Inst -> while E0 do LInsts endwhile
Inst.a = while(E0.a, LInsts.a)
Inst -> read E0
Inst.a = read(E0.a)
Inst -> write E0
Inst.a = write(E0.a)
Inst-> nl
Inst.a = nl()
Inst -> new E0
Inst.a = new(E0.a)
Inst -> delete E0
Inst.a = delete(E0.a)
Inst -> call Identificador ParamsR
Inst.a = call(Identificador.lex, ParamsR.a)
Inst -> Bloq
Inst.a = bl(Bloq.a)
LInsts -> ε
LInsts.a = lista_sin()
LInsts -> Insts
LInsts.a = lista_con(Insts.a)
ParamsR -> (LParamROpc)
ParamsR.a = LParamROpc.a
LParamROpc -> ε
LParamROpc.a = param_r_sin()
LParamROpc -> LParamR
LParamROpc.a = LParamR.a

LParamR -> E0

LParamR.a = param_r_con_una(E0.a)

LParamR -> LParamR , E0

$LParamR_0$.a = param_r_con_muchas($LParamR_1$.a, E0.1)

Bloq -> { BloqOpc }

Bloq.a = BloqOpc.a

BloqOpc -> ε

BloqOpc.a = bloque_sin()

BloqOpc -> Programa

BloqOpc.a = bloque_con(Programa.a)

E0 -> E1 OpIn0AsocD E0

$E0_0$.a = opera_dos(OpIn0AsocD.op, E1.a, $E0_1$.a)

E0 -> E1 OpIn0NoAsoc E1

E0.a = opera_dos(OpIn0NoAsoc.op, $E1_0$.a, $E1_1$.a)

E0 -> E1

E0.a = E1.a

E1 -> E1 OpIn1AsocI E2

$E1_0$.a = opera_dos(OpIn1AsocI.op, $E1_1$.a, E2.a)

E1 -> E2

E1.a = E2.a

E2 -> E2 OpIn2AsocI E3

$E2_0$.a = opera_dos(OpIn2AsocI.op, $E2_1$.a, E3.a)

E2 -> E3

E2.a = E3.a

E3 -> E4 OpIn3NoAsoc E4

E3.a = opera_dos(OpIn3NoAsoc.op, $E4_0$.a, $E4_1$.a)

E3 -> E4

E3.a = E4.a

E4 -> OpPre4NoAsoc E5

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

E4 -> OpPre4Asoc E4

$E4_0$.a = opera_uno(OpPre4Asoc.op, $E4_1$.a)

E4 -> E5

E4.a = E5.a

E5 -> E5 OpPos5Asoc

$E5_0$.a = opera_opposcincoasoc(OpPos5Asoc.op, OpPos5Asoc.a, OpPos5Asoc.var ,$E5_1$.a)

E5 -> E6

E5.a = E6.a

E6 -> OpPre6Asoc E6

$E6_0$.a = opera_uno(OpPre6Asoc.op, $E6_1$.a)

E6 -> E7

E6.a = E7.a

E7 -> (E0)

E7.a = E0.a

E7 -> LitEnt

E7.a = entero(LitEnt.lex)

E7 -> LitReal

E7.a = real(LitReal.lex)

E7 -> LitCad

E7.a = cadena(LitCad.lex)

E7 -> true

E7.a = verdadero()

E7 -> false

E7.a = falso()

E7 -> Identificador

E7.a = identificador(Identificador.lex)

E7 -> null

E7.a = null()

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 = ‘/’

OpIn3NoAsoc -> %

OpIn3NoAsoc.op = ‘%’

OpPre4NoAsoc -> -

OpPre4NoAsoc.op = ‘-’

OpPre4Asoc -> not

OpPre4Asoc.op = ‘not’

OpPos5Asoc -> [E0]

OpPos5Asoc.op = ‘index’

OpPos5Asoc.a = E0.a

OpPos5Asoc.var = null

OpPos5Asoc -> .Identificador

OpPos5Asoc.op = ‘reg’

OpPos5Asoc.a = null

OpPos5Asoc.var = Identificador.lex

OpPos5Asoc -> ->Identificador

OpPos5Asoc.op = ‘reg’

OpPos5Asoc.a = null

OpPos5Asoc.var = Identificador.lex

OpPre6Asoc -> *

OpPre6Asoc = ‘*’

Funciones semánticas

```
fun prog(Dec, Ins){
      if (Dec == null) then return prog_sin_decs(Ins)
      else return prog_con_decs(Dec, Ins)
}

fun opera_uno(Op, Arg){
      switch Op
            case ‘-’: return m_unario(Arg)
            case ‘not’: return not(Arg)
            case ‘*’: return indireccion(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)
            cse ‘%’: return modulo(Arg0, Arg1)

}
```

```
fun  opera_opposcincoasoc(Op, Arg_a, Arg_v ,Arg1){
      switch Op
            case ‘index’: return indexacion(Arg1, Arg_a)
            case ‘reg’: return acc_registro(Arg1, Arg_v)
}
```

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 -> ε
    Decs.a = null
Decs -> LDecs &&
    Decs.a = LDecs.a
LDecs -> Dec RLDecs
    LDecs.a = RLDecs.a
    RLDecs.ah = dec_una(Dec.a)
RLDecs -> ; Dec RLDecs
    $RLDecs_0$.a = $RLDecs_1$.a
    $RLDecs_1$.ah = decs_muchas($RLDecs_0$.ah, Dec.a)
RLDecs -> ε
    RLDecs.a = RLDecs.ah
Dec -> var TypN Identificador
    Dec.a = var(TypN.a, Identificador.lex)
Dec -> type TypN Identificador
    Dec.a = type(TypN.a, Identificador.lex)
Dec -> proc Identificador ParamsF Bloq
    Dec.a = proc(Identificador.lex, ParamsF.a, Bloq.a)
ParamsF -> (LParamFOpc)
    ParamsF.a = LParamFOpc.a
LParamFOpc -> ε
    LParamFOpc.a = param_f_sin()
LParamFOpc -> LParamF
    LParamFOpc.a = LParamF.a
LParamF -> ParamF RLParamF
    LParamF.a = RLParamF.a
    RLParamF.ah = param_f_con_una(ParamF.a)
RLParamF -> , ParamF RLParamF
    $RLParamF_0$.a = $RLParamF_1$.a
    $RLParamF_1$.ah = param_f_con_muchas($RLParamF_0$.ah, ParamF.a)

RLParamF -> ε
    RLParamF.a = RLParamF.ah
ParamF -> TypN RParamF
    RParamF.ah = TypN.a
    ParamF.a = RParamF.a
RParamF -> & Identificador
    RParamF.a = param_f_ref(RParamF.ah, Identificador.lex)
RParamF -> Identificador
    RParamF.a = param_f_noref(RParamF.ah, Identificador.lex)
TypN -> BaseType
    TypN.a = BaseType.a
TypN -> array [LitEnt] of BaseType
    TypN.a = tipo_array(LitEnt.a, BaseType.a)
TypN -> record { LCampos }
    TypN.a = tipo_record(LCampos.a)
TypN -> pointer BaseType
    TypN.a = tipo_pointer(BaseType.a)
LCampos -> Campo RLCampos
    LCampos.a = RLCampos.a
    RLCampos.ah = campo_uno(Campo.a)
RLCampos -> ; Campo RLCampos
    $RLCampos_0$.a = $RLCampos_1$.a
    $RLCampos_1$.ah = campo_muchos($RLCampos_0$.ah, Campo.a)
RLCampos -> ε
    RLCampos.a = RLCampos.ah
Campo -> TypN Identificador
    Campo.a = campo(TypN.a, Identificador.lex)
BaseType -> BasicType
    BaseType.a = BasicType.a
BaseType -> Identificador
    BaseType.a = tipo_iden(Identificador.lex)
BasicType -> int
    BasicType.a = tipo_int()
BasicType -> realw
    BasicType.a = tipo_real()
BasicType -> bool
    BasicType.a = tipo_bool()
BasicType -> string
    BasicType.a = tipo_string()
Insts -> Inst RInsts
    Insts.a = RInsts.a
    RInsts.ah = inst_una(Inst.a)
RInsts -> ; Inst RInsts
    $RInsts_0$.a = $RInsts_1$.a

$RInsts_1$.ah = insts_muchas($RInsts_0$.ah, Inst.a)

RInsts -> ε

RInsts.a = RInsts.ah

Inst -> E0 = E0

Inst.a = e_igual($E0_0$.a, $E0_1$.a)

Inst -> if E0 then LInsts RInst

RInst.ah = LInsts.a

RInst.ahh = E0.a

Inst.a = RInst.a

RInst -> endif

RInst.a = if(RInst.ahh, RInst.ah)

RInst -> else LInsts endif

RInst.a = ifelse(RInst.ahh, RInst.ah, LInsts.a)

Inst -> while E0 do LInsts endwhile

Inst.a = while(E0.a, LInsts.a)

Inst -> read E0

Inst.a = read(E0.a)

Inst -> write E0

Inst.a = write(E0.a)

Inst-> nl

Inst.a = nl()

Inst -> new E0

Inst.a = new(E0.a)

Inst -> delete E0

Inst.a = delete(E0.a)

Inst -> call Identificador ParamsR

Inst.a = call(Identificador.lex, ParamsR.a)

Inst -> Bloq

Inst.a = bl(Bloq.a)

LInsts -> ε

LInsts.a = lista_sin()

LInsts -> Insts

LInsts.a = lista_con(Insts.a)

ParamsR -> (LParamROpc)

ParamsR.a = LParamROpc.a

LParamROpc -> ε

LParamROpc.a = param_f_sin()

LParamROpc -> LParamR

LParamROpc.a = LParamR.a

LParamR -> E0 RLParamR

LParamR.a = RLParamR.a

RLParamR.ah = param_r_con_una(E0.a)

RLParamR -> , E0 RLParamR

$RLParamR_0$.a = $RLParamR_1$.a

$RLParamR_1$.ah = param_r_con_muchas($RLParamR_0$.ah, E0.a)

RLParamR -> ε

RLParamR.a = RLParamR.ah

Bloq -> { BloqOpc }

Bloq.a = BloqOpc.a

BloqOpc -> ε

BloqOpc.a = bloque_sin()

BloqOpc -> Programa

BloqOpc.a = bloque_con(Programa.a)

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

$RE1_0$.a = $RE1_1$.a

$RE1_1$.ah = opera_dos(OpIn1AsocI.op, $RE1_0$.ah ,E2.a)

RE1 -> ε

RE1.a = RE1.ah

E2 -> E3 RE2

E2.a = RE2.a

RE2.ah = E3.a

RE2 -> OpIn2AsocI E3 RE2

$RE2_0$.a = $RE2_1$.a

$RE2_1$.ah = opera_dos(OpIn2AsocI.op, $RE2_0$.ah ,E3.a)

RE2 -> ε

RE2.a = RE2.ah

E3 -> E4 RE3

E3.a = RE3.a

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

$E4_0$.a = opera_uno(OpPre4Asoc.op, $E4_1$.a)

E4 -> E5

E4.a = E5.a

E5 -> E6 RE5

E5.a = RE5.a

RE5.ah = E6.a

RE5 -> OpPos5Asoc RE5

$RE5_0$.a = $RE5_1$.a

$RE5_1$.ah = opera_opposcincoasoc(OpPos5Asoc.op, OpPos5Asoc.a, OpPos5Asoc.var, $RE5_0$.ah)

RE5 -> ε

RE5.a = RE5.ah

E6 -> OpPre6Asoc E6

$E6_0$.a = opera_uno(OpPre6Asoc.op, $E6_1$.a)

E6 -> E7

E6.a = E7.a

E7 -> (E0)

E7.a = E0.a

E7 -> LitEnt

E7.a = entero(LitEnt.lex)

E7 -> LitReal

E7.a = real(LitReal.lex)

E7 -> LitCad

E7.a = cadena(LitCad.lex)

E7 -> true

E7.a = verdadero()

E7 -> false

E7.a = falso()

E7 -> Identificador

E7.a = identificador(Identificador.lex)

E7 -> null

E7.a = null()

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 = ‘/’
OpIn3NoAsoc -> %
    OpIn3NoAsoc.op = ‘%’
OpPre4NoAsoc -> -
    OpPre4NoAsoc.op = ‘-’
OpPre4Asoc -> not
    OpPre4Asoc.op = ‘not’
OpPos5Asoc -> [E0]
    OpPos5Asoc.op = ‘index’
    OpPos5Asoc.a = E0.a
    OpPos5Asoc.var = null
OpPos5Asoc -> .Identificador
    OpPos5Asoc.op = ‘reg’
    OpPos5Asoc.a = null
    OpPos5Asoc.var = Identificador.lex
OpPos5Asoc -> ->Identificador
    OpPos5Asoc.op = ‘reg’
    OpPos5Asoc.a = null
    OpPos5Asoc.var = Identificador.lex
OpPre6Asoc -> *
    OpPre6Asoc = ‘*’