Práctica 3

Procesadores de Lenguajes

Borja Aday Guadalupe Luis
Diego Enrique de Miguel López
Grupo 13

Índice

1. Sintaxis abstracta.

Gramática Tiny 0	Sintaxis abstracta
Programa -> PDeclaraciones **&&** PIntrucciones PDeclaraciones -> LDecs LDecs -> LDecs **;** Dec LDecs -> Dec Dec -> Tipo **identificador** Tipo -> **int** Tipo -> **real** Tipo -> **bool** PInstrucciones -> LIns LIns -> LIns **;** Ins LIns -> Ins Ins -> **identificador =** E0 E0 -> E1 **+** E0 E0 -> E1 - E1 E0 -> E1 E1 -> E1 OpN1 E2 E1 -> E2 E2 -> E2 OpN2 E3 E2 -> E3 E3 -> E4 OpN3 E4 E3 -> E4 E4 -> **-** E5 E4 -> **not** E4 E4 -> E5 E5 -> **identificador** E5 -> **numEnt** E5 -> **numReal** E5 -> **true** E5 -> **false** E5 **-> (** E0 **)** OpN1 -> **and** OpN1 -> **or** OpN2 -> **<** OpN2 -> **>** OpN2 -> **<=** OpN2 -> **>=** OpN2 -> **==** OpN2 -> **!=** OpN3 -> ***** OpN3 -> /	programa: LDecs x LIns -> Programa decs_1: Dec -> LDecs decs_muchas: LDecs x Dec -> LDecs tipo_int: -> Tipo tipo_real: -> Tipo tipo_bool: -> Tipo dec: Tipo x string -> Dec ins_1: Ins -> LIns ins_muchas: LIns x Ins -> LIns ins: string x Exp -> Ins suma: Exp x Exp -> Exp resta: Exp x Exp -> Exp mul: Exp x Exp -> Exp div: Exp x Exp -> Exp and: Exp x Exp -> Exp or: Exp x Exp -> Exp menor: Exp x Exp -> Exp mayor: Exp x Exp -> Exp menor_igual: Exp x Exp -> Exp mayor_igual: Exp x Exp -> Exp igualdad: Exp x Exp -> Exp distinto: Exp x Exp -> Exp menos: Exp -> Exp not: Exp -> Exp identificador: string -> Exp numEnt: string -> Exp numReal: string -> Exp true: -> Exp false: -> Exp

2. Constructor de árboles de sintaxis abstracta (ASTs).

Constructor de árboles de sintaxis abstracta.
Gramática S-Atribuida

Programa -> PDeclaraciones **&&** PInstrucciones
  Programa.a = Programa(PDeclaraciones.a, PInstrucciones.a)

PDeclaraciones -> LDecs
  PDeclaraciones.a = LDecs.a

LDecs -> LDecs **;** Dec
  $LDecs_0$.a = decs_muchas($LDecs_1$.a, Dec.a)
LDecs -> Dec
  LDecs.a = decs_1(Dec.a)

Dec -> Tipo **identificador**
  Dec.a = dec(Tipo.a, **identificador**. lexema)
tipo
Tipo -> **int**
  Tipo.a = tipo_int()
Tipo -> **real**
  Tipo.a = tipo_real()
Tipo -> **bool**
  Tipo.a = tipo_bool()

PInstrucciones -> LIns
  PInstrucciones.a = LIns.a

LIns -> LIns **;** Ins
  $LIns_0$.a = ins_muchas($LIns_1$.a, Ins.a)
LIns -> Ins
  LIns.a = ins_1(Ins.a)

Ins -> **identificador =** E0
  Ins.a = ins(**identificador**.lexema, E0.a)

E0 -> E1 **+** E0
  $E0_0$.a = suma(E1.a, $E0_1$.a)
E0 -> E1 - E1
  E0.a = resta($E1_0$.a, $E1_1$.a)
E0 -> E1
  E0.a = E1.a

E1 -> E1 OpN1 E2
  $E1_0$.a = exp(OpN1.op, $E1_1$.a, E2.a)
E1 -> E2
  E1.a = E2.a

E2 -> E2 OpN2 E3
  $E2_0$.a = exp(OpN2.op, $E2_1$.a, E3.a)
E2 -> E3
  E2.a = E3.a

E3 -> E4 OpN3 E4
  E3.a = exp(OpN3.op, $E4_0$.a, $E4_1$.a)
E3 -> E4
  E3.a = E4.a

E4 -> **-** E5
  E4.a = menos(E5.a)
E4 -> **not** E4
  $E4_0$.a = not($E4_1$.a)
E4 -> E5
E4.a = E5.a

E5 -> **identificador**
  E5.a = identificador(**identificador**.lexema)
E5 -> **numEnt**
  E5.a = numEnt(**numEnt**. lexema)
E5 -> **numReal**
  E5.a = numReal(**numReal**. lexema)
E5 -> **true**
  E5.a = true()
E5 -> **false**
  E5.a = false()
E5 **-> (** E0 **)**
  E5.a = E0.a

OpN1 -> **and**
  OpN1.op = “and”
OpN1 -> **or**
  OpN1.op = “or”
OpN2 -> **<**
  OpN1.op = “<”
OpN2 -> **>**
  OpN1.op = “>”
OpN2 -> **<=**
  OpN1.op = “<=”
OpN2 -> **>=**
  OpN1.op = “>=”
OpN2 -> **==**
  OpN1.op = “==”
OpN2 -> **!=**
  OpN1.op = “!=”
OpN3 -> *****
  OpN1.op = “*”
OpN3 -> /
  OpN1.op = “/”

2.1. 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 igualdad(Arg0, Arg1)
    case “=!“: return distinto(Arg0, Arg1)
    case “*“: return mul(Arg0, Arg1)
    case “/“: return div(Arg0, Arg1)
}
```

3. Acondicionamiento para implementación descendente.

Constructor de árboles de sintaxis abstracta.
Gramática S-Atribuida

Programa -> PDeclaraciones **&&** PInstrucciones
  Programa.a = Programa(PDeclaraciones.a, PInstrucciones.a)

PDeclaraciones -> LDecs
  PDeclaraciones.a = LDecs.a

LDecs -> Dec RLDecs
  RLDecs.ah = decs_1(Dec.a)
  LDecs.a = RLDecs.a
RLDecs -> **;** Dec RLDecs
  $RLDecs_1$.ah = decs_muchas($RLDecs_0$.ah, Dec.a)
  $RLDecs_0$.a = $RLDecs_1$.a
RLDecs -> ε
  RLDecs.a = RLDecs.ah

Dec -> Tipo **identificador**
  Dec.a = dec(Tipo.a, **identificador**. lexema)

Tipo -> **int**
  Tipo.a = int()
Tipo -> **real**
  Tipo.a = real()
Tipo -> **bool**
  Tipo.a = bool()

PInstrucciones -> LIns
  PInstrucciones.a = LIns.a

LIns -> Dec RLIns
  RLIns.ah = ins_1(Ins.a)
  LIns.a = RLIns.a
RLIns -> **;** Ins RLIns
  $RLIns_1$.ah = ins_muchas($RLIns_0$.ah, Ins.a)
  $RLIns_0$.a = $RLIns_1$.a
RLIns -> ε
  RLIns.a = RLIns.ah

Ins -> **identificador =** E0
  Ins.a = ins(**identificador**. lexema, E0.a)

E0 -> E1 RE0
RE0.ah = E1.a
E0.a = RE0.a
RE0 -> **+** E0
RE0.a = suma(RE0.ah, E0.a)
RE0 -> **-** E1
RE0.a = resta(RE0.ah, E1.a)
RE0 -> ε
RE0.a = RE0.ah

E1 -> E2 RE1
  RE1.ah = E2.a
  E1.a = RE1.a
RE1 -> OpN1 E2 RE1
  $RE1_1$.ah = exp(opN1.op, $RE1_0$.ah, E2.a)
RE1 -> ε

RE1.a = RE1.ah

E2 -> E3 RE2
RE2.ah = E3.a
E2.a = RE2.a
RE2 -> OpN2 E3 RE2
$RE2_1$.ah = exp(opN2.op, $RE2_0$.ah, E3.a)
RE2 -> ε
RE2.a = RE2.ah

E3 -> E4 RE3
RE3.ah = E4.a
E3.a = RE3.a
RE3 -> OpN3 E4
RE3.a = exp(opN3.op, RE3.ah, E4.a)
RE3 -> ε
RE3.a = RE3.ah

E4 -> **-** E5
E4.a = menos(E5.a)
E4 -> **not** E4
$E4_0$.a = not($E4_1$.a)
E4 -> E5
E4.a = E5.a

E5 -> **identificador**
E5.a = identificador(**identificador**. lexema)
E5 -> **numEnt**
E5.a = numEnt(**numEnt**. lexema)
E5 -> **numReal**
E5.a = numReal(**numReal**. lexema)
E5 -> **true**
E5.a = true()
E5 -> **false**
E5.a = false()
E5 **-> (** E0 **)**
E5.a = E0.a

OpN1 -> **and**
OpN1.op = “and”
OpN1 -> **or**
OpN1.op = “or”
OpN2 -> **<**
OpN1.op = “<”
OpN2 -> **>**
OpN1.op = “>”
OpN2 -> **<=**
OpN1.op = “<=”
OpN2 -> **>=**
OpN1.op = “>=”
OpN2 -> **==**
OpN1.op = “==”
OpN2 -> **!=**
OpN1.op = “!=”
OpN3 -> ***
OpN1.op = “*”
OpN3 -> /
OpN1.op = “/”