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 signaturas (cabeceras) de las funciones constructoras de ASTs.

Sintaxis abstracta:

programa_conDecs: Decs x Instrucciones → Programa
programa_sinDecs: Instrucciones → Programa
decs_varias: Decs x Dec → Decs
decs_una: Dec → Decs
dec_var: tipo x id → Dec
dec_type: tipo x id→ Dec
dec_proc: id x ParamForm x bloque
param_form_vacio: → ParamForm
param_form: ListParamForm → ParamForm
list_param_form_uno: Parametro -> ParamForm
list_param_form_varios: ListParamForm x Parametro → ParamForm
param_valor: tipo x id → Parametro
param_variable: tipo x id → Parametro
tipo_int: → tipo
tipo_real: → tipo
tipo_bool: → tipo
tipo_string: → tipo
tipo_array: num x tipo → tipo
tipo_record: campos → tipo
tipo_pointer: tipo → tipo
campos_varios: campos x campo → campos
campos_uno: campo → campos
campo: tipo x string → campo
instrucciones_varias: Instrucciones x Inst → Instrucciones
instrucciones_una: Inst → Instrucciones
inst_asig: Expresion x Expresion → Inst
inst_IfThen: Expresion x InstOp → Inst
inst_IfThenElse: Expresion x InstOp x InstOp → Inst
inst_While: Expresion x InstOp → Inst
inst_op_varias: Instrucciones → InstOp
inst_op_vacio: → InstOp
inst_Read: Expresion → Inst
inst_Write: Expresion → Inst
inst_NewLine: → Inst
inst_new: Expresion → Inst
inst_delete: Expresion → Inst
inst_bloque: Bloque → Inst
bloque: Programa → Bloque
bloque_vacio: → Bloque
inst_Call: String x paramReales → Inst

paramReales_varios: ListaExpresiones → paramReales
paramReales_vacio: → paramReales
listaExpresiones_varias: ListaExpresiones x Expresion → ListaExpresiones
listaExpresiones_una: Expresion → ListaExpresiones
suma: Expresion x Expresion → Expresion
resta: Expresion x Expresion → Expresion
and: Expresion x Expresion → Expresion
or: Expresion x Expresion → Expresion
mayor: Expresion x Expresion → Expresion
mayor_igual: Expresion x Expresion → Expresion
menor: Expresion x Expresion → Expresion
menor_igual: Expresion x Expresion → Expresion
comparacion: Expresion x Expresion → Expresion
distinto: Expresion x Expresion → Expresion
mul: Expresion x Expresion → Expresion
div: Expresion x Expresion → Expresion
mod: Expresion x Expresion → Expresion
neg: Expresion → Expresion
not: Expresion → Expresion
indexacion: Expresion x Expresion → Expresion
punto: Expresion x id → Expresion
flecha: Expresion x id → Expresion
indice: Expresion → Expresion

id: string → Expresion
numeroEntero: string → Expresion
numeroReal: string → Expresion
true: → Expresion
false: → Expresion
null: → Expresion
literalCad: string → Expresion

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

Constructor de árboles de sintaxis abstracta (ASTs):

Programa → Decs '&&' Instrucciones
    Programa.a = programa_conDecs(Decs.a, Instrucciones.a)
Programa → Instrucciones
    Programa.a = programa_sinDecs(Instrucciones.a)

Decs → Decs ‘;’ Dec
    $Decs_0$.a = decs_varias($Decs_0$.a, Dec.a)
Decs → Dec
    Decs.a = decs_una(Dec.a)

Dec → var tipo id
    Dec.a = dec_var(tipo.a,id.lex)
Dec → type tipo id
    Dec.a = dec_type(tipo.a,id.lex)
Dec → proc id ParamForm bloque
    Dec.a = dec_proc(id.lex, ParamForm.a, bloque.a)
ParamForm → ‘(‘ ListParamForm ‘)’
    ParamForm.a = param_form(ListParamForm.a)
ParamForm → ‘(‘ ‘)’
    ParamForm.a = param_form_vacio()

ListParamForm → ListParamForm ‘,’ Parametro
    $ListParamForm_0$.a = list_param_form_varios($ListParamForm_1$.a, Parametro.a)
ListParamForm → Parametro
    ListParamForm.a = list_param_form_uno(Parametro.a)

Parametro → tipo ‘&’ id
    Parametro.a = param_variable(tipo.a, id.lex)
Parametro → tipo id
    Parametro.a = param_valor(tipo.a, id.lex)

bloque → ‘{‘ Programa ‘}
    Bloque.a = bloque(Programa.a)
bloque → ‘{‘ ‘}’
    bloque.a = bloque_vacio()

tipo → int
    tipo.a = tipo_int()
tipo → bool
    tipo.a = tipo_bool()
tipo → real
    tipo.a = tipo_real()
tipo → string
    tipo.a = tipo_string()
tipo → id
    tipo.a = id(id.lex)
tipo → array ‘[‘ numeroEntero ‘]’ of tipo
    $tipo_0$.a = tipo_array(numeroEntero.lex, $tipo_1$.a)
tipo → record ‘{‘ campos ‘}
    tipo.a = tipo_record(campos.a)
tipo → pointer tipo
    $tipo_0$.a = tipo_pointer($tipo_1$.a)

campos → campos ‘;’ campo
    $campos_0$.a = campos_varios($campos_1$.a, campo.a)
campos → campo
    campos.a = campos_uno(campo.a)

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

Instrucciones → Instrucciones ‘;’ Inst
    $Instrucciones_0$.a = instrucciones_varias($Instrucciones_1$.a, Inst.a)
Instrucciones → Inst
    Instrucciones.a = instrucciones_una(Inst.a)

- Instrucción de asignación:

Inst → Expresion ‘=’ Expresion
    Inst.a = inst_asig(Expresion.a, Expresion.a)

- Instrucción if-then:

Inst → if Expresion then InstOp endif
    Inst.a = inst_IfThen(Expresion.a, InstOp.a)

- Instrucción if-then-else:

Inst → if Expresion then InstOp else InstOp endif
    Inst.a = inst_IfThenElse(Expresion.a, InstOp.a, InstOp.a)

- Instrucción while:

Inst → while Expresion do InstOp endwhile
    Inst.a = inst_While(Expresion.a, InstOp.a)

InstOp → Instrucciones
    InstOp.a = inst_op_varias(Instrucciones.a)
InstOp → ε
    InstOp.a = inst_op_vacio()

- Instrucción de lectura

Inst → read Expresion
    Ints.a = inst_Read(Expresion.a)

- Instrucción de escritura

Inst → write Expresion
    Inst.a = inst_Write(Expresion.a)

- Instrucción de nueva línea

Inst → nl
    Ints.a = inst_NewLine()

- Instrucción de reserva de memoria

Inst → 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
Inst.a = inst_Call(id.lex, paramReales.a)

paramReales → '(' ListaExpresiones ')'
paramReales.a = paramReales_varios(ListaExpresiones.a)
paramReales → '(' ')'
paramReales.a = paramReales_vacio()

ListaExpresiones → ListaExpresiones ',' Expresion
$ListaExpresiones_0$.a = listaExpresiones_varias($ListaExpresiones_1$.a, Expresion.a)
ListaExpresiones → Expresion
ListaExpresiones.a = listaExpresiones_una(Expresion.a)

- Instrucción compuesta

Inst → bloque
Inst.a = inst_bloque(bloque.a)

Expresion → E0
Expresion.a = 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 OP1 E2
$E1_0$.a = exp(OP1.a, $E1_1$.a, E2.a)
E1 → E2
E1.a = E2.a
E2 → E2 OP2 E3
$E2_0$.a = exp(OP2.a, $E2_0$.a, E3.a)
E2 → E3
E2.a = E3.a
E3 → E4 OP3 E4
E3.a = exp(OP2.a, $E4_0$.a, $E4_0$.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 ']'
$E5_0$.a = indexacion($E5_1$.a, expr.a)
E5 -> E5 '.' id
$E5_0$.a = punto($E5_1$.a, id.lex)
E5 -> E5 '->' id
$E5_0$.a = flecha($E5_1$.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**
Decs → Decs ‘;’ Dec Decs.a = decs_varias(Decs.a, Dec.a)	Decs → Dec restoDecs restoDecs.ah = Dec.a Decs.a = restoDecs.a restoDecs → ; Dec restoDecs $restoDecs_1$.ah = decs_varias($restoDecs_0$.ah, Dec.a) $restoDecs_0$.a = $restoDecs_1$.a restoDecs → ε restoDecs.a = restoDecs.ah
ListParamForm → ListParamForm ‘,’ Parametro $ListParamForm_0$.a = list_param_form_varios($ListParamForm_1$.a, Parametro.a)	ListParamForm → Parametro restoListaParamForm restoListaParamForm.ah = Parametro.a ListParamForm.a = restoListaParamForm.a restoListaParamForm → ; Parametro restoListaParamForm $restoListaParamForm_1$.ah = list_param_form_varios($ListParamForm_0$.ah, Parametro.a) $restoListaParamForm_0$.a = $restoListaParamForm_1$.a restoListaParamForm → ε restoListaParamForm.a = restoListaParamForm.ah
campos → campos ‘;’ campo $campos_0$.a = campos_varios($campos_1$.a, campo.a) campos → campo campos.a = campos_uno(campo.a)	campos → campo restoCampos restoCampos.ah = campo.a campos.a = restoCampos.a restoCampos → ‘;’ campo restoCampos $restoCampos_1$.ah = campos_varios($restoCampos_0$.ah, campo.a) $restoCampos_0$.a = $restoCampos_1$.a restoCampos → ε restoCampos.a = restoCampos.ah
Instrucciones → Instrucciones ‘;’ Inst $Instrucciones_0$.a = inst_varias($Instrucciones_1$.a, Inst.a)	Instrucciones → Inst restoIns restoIns.ah = Inst.a Instrucciones.a = restoIns.a restoIns → ; Inst restoIns $restoIns_1$.ah = inst_varias($restoIns_0$.ah, Inst.a) $restoIns_0$.a = $restoIns_1$.a restoIns → ε restoIns.a = restoIns.ah
ListaExpresiones → ListaExpresiones ‘,’ Expresion $ListaExpresiones_0$.a = listaExpresiones_varias($ListaExpresiones_1$.a, Expresion.a) ListaExpresiones → Expresion	ListaExpresiones → Expresion RestoListaExpresiones RestoListaExpresiones.ah = Expresion.a ListaExpresiones.a = RestoListaExpresiones.a RestoListaExpresiones → ‘;’ Expresion RestoListaExpresiones

ListaExpresiones.a = listaExpresiones_una(Expresion.a)	RestoListaExpresiones$_1$.ah = listaExpresiones_varias(RestoListaExpresiones$_0$.ah, Expresion.a) RestoListaExpresiones$_0$.a = RestoListaExpresiones$_1$.a RestoListaExpresiones → ε RestoListaExpresiones.a = RestoListaExpresiones.ah
E1 → E1 OP1 E2 E1$_0$.a = exp(OP1.a, E1$_1$.a, E2.a)	E1 → E2 restoE1 restoE1.ah = E2.a E1.a = restoE1.a restoE1 → OP1 E2 restoE1 restoE1$_1$.ah = E2.a restoE1$_0$.a = exp(OP1.a, restoE1$_0$.ah, restoE1$_1$.a) restoE1 → ε restoE1.a = restoE1.ah
E2 → E2 OP2 E3 E2$_0$.a = exp(OP2.a, E2$_1$.a, E3.a)	E2 → E3 restoE2 restoE2.ah = E3.a E2.a = restoE2.a restoE2 → OP2 E3 restoE2 restoE2$_1$.ah = E3.a restoE2$_0$.a = exp(OP2.a, restoE2$_0$.ah, restoE2$_1$.a) restoE2 → ε restoE2.a = restoE2.ah
E5 -> E5 '[' expr ']' E5$_0$.a = indexacion(E5$_1$.a, expr.a) E5 -> E5 '.' id E5$_0$.a = punto(E5$_1$.a, id.lex) E5 -> E5 '->' id E5$_0$.a = flecha(E5$_1$.a, id.lex)	**Primero eliminamos factores comunes:** E5 → E5 restoE5 restoE5.ah = E5$_1$.a E5.a = restoE5.a restoE5 → '[' expr ']' restoE5.a = indexacion(restoE5.ah, expr.a) restoE5 → '.' id restoE5.a = punto(restoE5.ah, id.lex) restoE5 → '->' id restoE5.a = flecha(restoE5.ah, id.lex) **Y sobre esto eliminamos la recursion a izquierdas:** E5 -> E6 resto2E5 Res2E5.ah = E6.a E5.a = resto2E5.a resto2E5 -> restoE5 resto2E5 restoE5.ah = resto2E5$_0$.ah resto2E5$_1$.ah = restoE5.a resto2E5$_0$.a = resto2E5$_1$.a

	resto2E5 -> ε Rest2E5.a = Rest2E5.ah restoE5 → '[' expr ']' restoE5.a = indexacion(restoE5.ah, expr) restoE5 → '.' id restoE5.a = punto(restoE5.ah, id.lex) restoE5 → '->' id restoE5.a = flecha(restoE5.ah, id.lex)
Eliminación de factores comunes	
Sin acondicionar	**Acondicionada**
ParamForm → '(' ListParamForm ')' ParamForm.a = param_form(ListParamForm.a) ParamForm → '(' ')' ParamForm.a = param_form_vacio()	ParamForm → '(' restoParamForm ParamForm.a = restoParamForm.a restoParamForm → ListParamForm ')' restoParamForm.a = param_form(ListParamForm.a) restoParamForm → ')' restoParamForm.a = param_form_vacio()
Parametro → tipo '&' id Parametro.a = param(tipo.a, id.lex) Parametro → tipo id Parametro.a = param(tipo.a, id.lex)	Parametro → tipo restoParametro restoParametro.ah = tipo.a Parametro.a = restoParametro.a restoParametro → '&' id restoParametro.a = param(restoParametro.ah, id.lex) restoParametro → id restoParametro.a = param(restoParametro.ah, id.lex)
Inst → if Expresion then InstOp endif Inst.a = inst_IfThen(Expresion.a, InstOp.a) - Instrucción if-then-else: Inst → if Expresion then InstOp else InstOp endif Inst.a = inst_IfThenElse(Expresion.a, InstOp.a, InstOp.a)	Inst → If Expresion then InstOp restoIf restoIf.ahex = Expresion.a restoIf.ahop = InstOp.a Inst.a = restoIf.a restoIf → endif restoIf.a = inst_IfThen(restoIf.ahex, restoIf.ahop) restoIf → else InstOp endif restoIf.a = inst_IfThenElse(restoIf.ahex, restoIf.ahop, InstOp.a)
paramReales → '(' ListaExpresiones ')' paramReales.a = paramReales_varios(ListaExpresiones.a) paramReales → '(' ')' paramReales.a = paramReales_vacio()	paramReales → '(' restoParamReales restoParamReales.a = restoParamReales.a restoParamReales → ListaExpresiones ')' restoParamReales.a = paramReales_varios(ListaExpresiones.a) restoParamReales → ')' restoParamReales .a = paramReales_vacio()
E0 → E1 '+' E0 $E0_0$.a = suma(E1.a, $E0_1$.a) E0 → E1 '-' E1 E0.a = resta($E1_1$.a, $E1_2$.a)	E0 → E1 restoE0 restoE0.ah = E1.a E0.a = restoE0.a restoE0 → '+' E0 restoE0.a = suma(restoE0.ah, E0.a)

E0 → E1 E0.a = E1.a	restoE0 → '-' E1 restoE0.a = resta(restoE0.ah, E1.a) restoE0 → ε restoE0.a = restoE0.ah
E3 → E4 OP3 E4 E3.a = exp(OP3.a, $E4_1$.a, $E4_2$.a) E3 → E4 E3.a = E4.a	E3 → E4 restoE3 restoE3.ah = E4.a E3.a = restoE3.a restoE3 → OP3 E4 restoE3.a = exp(OP3.a, restoE3.ah, E4.a) restoE3 → ε 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. Sí hemos conseguido implementar el analizador ascendente. Aún así, en la entrega se observa todos los avances que hicimos en el analizador descendente.