Procesadores de Lenguajes

Memoria de proyecto — Hito 4:Finalización del procesador

Grupo 14

Rodrigo Souto Santos

Leonardo Prado de Souza

Juan Andrés Hibjan Cardona

Izan Rodrigo Sanz

Grado en Ingeniería informática
Facultad de Informática
Universidad Complutense de Madrid

2024–2025

Índice general

1. Especificación del Procesamiento de Vinculación **2**
1.1. Funciones para la tabla de símbolos . . . 2
1.2. Funciones de procesamiento . . . 2
1.2.1. Declaraciones . . . 2
1.2.2. Tipos . . . 4
1.2.3. Instrucciones . . . 5
1.2.4. Expresiones . . . 6

2. Especificación del Procesamiento de Pre-tipado **9**
2.1. Funciones para el conjunto de campos . . . 9
2.2. Funciones de procesamiento . . . 9
2.2.1. Declaraciones . . . 9
2.2.2. Tipos . . . 10
2.2.3. Instrucciones . . . 11
2.2.4. Expresiones . . . 11

3. Especificación del Procesamiento de Comprobación de Tipos **13**
3.1. Funciones para el conjunto de pares de tipos . . . 13
3.2. Funciones de procesamiento . . . 13
3.2.1. Declaraciones . . . 13
3.2.2. Tipos . . . 14
3.2.3. Instrucciones . . . 14
3.2.4. Expresiones . . . 17
3.3. Funciones auxiliares . . . 20

4. Especificación del Procesamiento de Asignación de Espacio **24**
4.1. Funciones de procesamiento . . . 24
4.1.1. Declaraciones . . . 24
4.1.2. Tipos . . . 26
4.1.3. Instrucciones . . . 27
4.1.4. Expresiones . . . 28
4.2. Funciones auxiliares . . . 29

5. Descripción de las Instrucciones de la Máquina-p **30**

6. Especificación del Procesamiento de Etiquetado **31**

7. Especificación del Procesamiento de Generación de Código **32**

1 Especificación del Procesamiento de Vinculación

1.1. Funciones para la tabla de símbolos

- `creaTS()`: Crea una tabla de símbolos que no tiene aún ningún ámbito abierto.
- `abreAmbito(ts)`: Añade a la tabla de símbolos `ts` un nuevo ámbito, que tendrá como padre el ámbito más reciente (o $\perp$, si aún no se ha creado ningún ámbito).
- `contiene(ts,id)`: Comprueba si el ámbito actual de la tabla de símbolos `ts` contiene ya una entrada para el identificador `id`.
- `inserta(ts,id,dec)`: Inserta el identificador `id` en el ámbito actual de la tabla de símbolos `ts`, con la referencia al nodo `dec` como valor.
- `vinculoDe(ts,id)`: Recupera la referencia asociada a `id` en la tabla de símbolos `ts`. Para ello busca sucesivamente en la cadena de ámbitos, hasta que lo encuentra (si no está, devuelve $\perp$).
- `cierraAmbito(ts)`: Fija en `ts` el ámbito actual al ámbito padre del ámbito más reciente.

1.2. Funciones de procesamiento

var ts = **crearTS**()

vincula(**bloque**(*SecDecs*, *SecIs*)) :
    abreAmbito(*ts*)
    vincula(*SecDecs*)
    vincula(*SecIs*)
    cierraAmbito(*ts*)

1.2.1. Declaraciones

vincula(**si_decs**(*LDecs*)) :
    vincula1(*LDecs*)
    vincula2(*LDecs*)

vincula(**no_decs()**) : **noop**

vincula1(**muchas_decs**(*LDecs*, *Dec*)) :
    vincula1(*LDecs*)
    vincula1(*Dec*)

vincula2(**muchas_decs**(*LDecs*, *Dec*)) :
    vincula2(*LDecs*)
    vincula2(*Dec*)

vincula1(**una_dec**(*Dec*)) :
    vincula1(*Dec*)

vincula2(**una_dec**(*Dec*)) :
    vincula2(*Dec*)

vincula1(**dec_base**(*TipoNom*)) :
    let TipoNom = TipoNom(*Tipo*, *iden*) *in*

if **contiene**(*ts*, *iden*) *then*
error
else
inserta(*ts*, *iden*, $)
endif
end let
vincula1(*TipoNom*)

vincula2(**dec_base**(*TipoNom*)) :
vincula2(*TipoNom*)

vincula1(**dec_type**(*TipoNom*)) :
let TipoNom = *TipoNom*(*Tipo*, *iden*) *in*
if **contiene**(*ts*, *iden*) *then*
error
else
inserta(*ts*, *iden*, $)
endif
end let
vincula1(*TipoNom*)

vincula2(**dec_type**(*TipoNom*)) :
vincula2(*TipoNom*)

vincula1(**dec_proc**(*iden*, *ParamFs*, *Bloq*)) :
if **contiene**(*ts*, *iden*) *then*
error
else
inserta(*ts*, *iden*, $)
endif

vincula2(**dec_proc**(*iden*, *ParamFs*, *Bloq*)) :
abreAmbito(*ts*)
vincula1(*ParamFs*)
vincula2(*ParamFs*)
vincula(*Bloq*)
cierraAmbito(*ts*)

vincula1(**si_params_f**(*LParamFs*)) :
vincula1(*LParamFs*)

vincula2(**si_params_f**(*LParamFs*)) :
vincula2(*LParamFs*)

vincula1(**no_params_f()**) : **noop**

vincula2(**no_params_f()**) : **noop**

vincula1(**muchos_params_f**(*LParamFs*, *ParamF*)) :
vincula1(*LParamFs*)
vincula1(*ParamF*)

vincula2(**muchos_params_f**(*LParamFs*, *ParamF*)) :
vincula2(*LParamFs*)
vincula2(*ParamF*)

vincula1(**un_param_f**(*ParamF*)) :
vincula1(*ParamF*)

vincula2(**un_param_f**(*ParamF*)) :

vincula2(*ParamF*)

vincula1(**si_refparam_f**(*Tipo*, *iden*)) :
if **contiene**(*ts*, *iden*) *then*
error
else
inserta(*ts*, *iden*, $)
endif
vincula1(*Tipo*)

vincula2(**si_refparam_f**(*Tipo*, *iden*)) :
vincula2(*Tipo*)

vincula1(**no_refparam_f**(*Tipo*, *iden*)) :
if **contiene**(*ts*, *iden*) *then*
error
else
inserta(*ts*, *iden*, $)
endif
vincula1(*Tipo*)

vincula2(**no_refparam_f**(*Tipo*, *iden*)) :
vincula2(*Tipo*)

1.2.2. Tipos

vincula1(**tipo_nombre**(*Tipo*, *iden*)) :
vincula1(*Tipo*)

vincula2(**tipo_nombre**(*Tipo*, *iden*)) :
vincula2(*Tipo*)

vincula1(**tipo_array**(*Tipo*, *litEntero*)) :
if Tipo != *tipo_type*(_) *then*
vincula1(*Tipo*)
endif

vincula2(**tipo_array**(*Tipo*, *litEntero*)) :
if Tipo == *tipo_type*(*iden*) *then*
Tipo.vinculo = **vinculoDe**(*ts*, *iden*)
if Tipo.vinculo != *dec_type*(_) *then*
error
endif
else
vincula2(*Tipo*)
endif

vincula1(**tipo_indir**(*Tipo*)) :
if Tipo != *tipo_type*(_) *then*
vincula1(*Tipo*)
endif

vincula2(**tipo_indir**(*Tipo*)) :
if Tipo == *tipo_type*(*iden*) *then*
Tipo.vinculo = **vinculoDe**(*ts*, *iden*)
if Tipo.vinculo != *dec_type*(_) *then*
error
endif
else
vincula2(*Tipo*)

endif

vincula1(**tipo_struct**(*LCampos*)) :
vincula1(*LCampos*)

vincula2(**tipo_struct**(*LCampos*)) :
vincula2(*LCampos*)

vincula1(**tipo_int**()) : **noop**

vincula2(**tipo_int**()) : **noop**

vincula1(**tipo_real**()) : **noop**

vincula2(**tipo_real**()) : **noop**

vincula1(**tipo_bool**()) : **noop**

vincula2(**tipo_bool**()) : **noop**

vincula1(**tipo_string**()) : **noop**

vincula2(**tipo_string**()) : **noop**

vincula1(**tipo_type**(*iden*)) : **noop**

vincula2(**tipo_type**(*iden*)) :
$\$.vinculo = vinculoDe(ts, iden)$

vincula1(**muchos_campos**(*LCampos*, *TipoNom*)) :
vincula1(*LCampos*)
vincula1(*TipoNom*)

vincula2(**muchos_campos**(*LCampos*, *TipoNom*)) :
vincula2(*LCampos*)
vincula2(*TipoNom*)

vincula1(**un_campo**(*TipoNom*)) :
vincula1(*TipoNom*)

vincula2(**un_campo**(*TipoNom*)) :
vincula2(*TipoNom*)

1.2.3. Instrucciones

vincula(**si_ins**(*LIs*)) :
vincula(*LIs*)

vincula(**no_ins()**) : **noop**

vincula(**muchas_ins**(*LIs*, *I*)) :
vincula(*LIs*)
vincula(*I*)

vincula(**una_ins**(*I*)) :
vincula(*I*)

vincula(**ins_eval**(*Exp*)) :
vincula(*Exp*)

vincula(**ins_if**(*Exp*, *Bloq*)) :
    vincula(*Exp*)
    vincula(*Bloq*)

vincula(**ins_if_else**(*I*, *Bloq*)) :
    vincula(*I*)
    vincula(*Bloq*)

vincula(**ins_while**(*Exp*, *Bloq*)) :
    vincula(*Exp*)
    vincula(*Bloq*)

vincula(**ins_read**(*Exp*)) :
    vincula(*Exp*)

vincula(**ins_write**(*Exp*)) :
    vincula(*Exp*)

vincula(**ins_nl**()) : **noop**

vincula(**ins_new**(*Exp*)) :
    vincula(*Exp*)

vincula(**ins_delete**(*Exp*)) :
    vincula(*Exp*)

vincula(**ins_call**(*iden*, *ParamRs*)) :
    $.*vinculo* = **vinculoDe**(*ts*, *iden*)
    if $.*vinculo* == ⊥ *then*
        error
    endif
    vincula(*ParamRs*)

vincula(**ins_bloque**(*Bloq*)) :
    vincula(*Bloq*)

vincula(**si_params_r**(*LParamRs*)) :
    vincula(*LParamRs*)

vincula(**no_params_r()**) : **noop**

vincula(**muchos_params_r**(*LParamRs*, *Exp*)) :
    vincula(*LParamRs*)
    vincula(*Exp*)

vincula(**un_param_r**(*Exp*)) :
    vincula(*Exp*)

1.2.4. Expresiones

vincula(**exp_asig**(*Opnd*0, *Opnd*1)) :
    vinculaExpBin(*Opnd*0, *Opnd*1)

vincula(**exp_menor**(*Opnd*0, *Opnd*1)) :
    vinculaExpBin(*Opnd*0, *Opnd*1)

vincula(**exp_menor_ig**(*Opnd*0, *Opnd*1)) :
    vinculaExpBin(*Opnd*0, *Opnd*1)

vincula(**exp_mayor**(*Opnd*0, *Opnd*1)) :

vinculaExpBin(*Opnd*0, *Opnd*1)

vincula(**exp_mayor_ig**(*Opnd*0, *Opnd*1)) :
vinculaExpBin(*Opnd*0, *Opnd*1)

vincula(**exp_ig**(*Opnd*0, *Opnd*1)) :
vinculaExpBin(*Opnd*0, *Opnd*1)

vincula(**exp_dist**(*Opnd*0, *Opnd*1)) :
vinculaExpBin(*Opnd*0, *Opnd*1)

vincula(**exp_suma**(*Opnd*0, *Opnd*1)) :
vinculaExpBin(*Opnd*0, *Opnd*1)

vincula(**exp_resta**(*Opnd*0, *Opnd*1)) :
vinculaExpBin(*Opnd*0, *Opnd*1)

vincula(**exp_and**(*Opnd*0, *Opnd*1)) :
vinculaExpBin(*Opnd*0, *Opnd*1)

vincula(**exp_or**(*Opnd*0, *Opnd*1)) :
vinculaExpBin(*Opnd*0, *Opnd*1)

vincula(**exp_mul**(*Opnd*0, *Opnd*1)) :
vinculaExpBin(*Opnd*0, *Opnd*1)

vincula(**exp_div**(*Opnd*0, *Opnd*1)) :
vinculaExpBin(*Opnd*0, *Opnd*1)

vincula(**exp_mod**(*Opnd*0, *Opnd*1)) :
vinculaExpBin(*Opnd*0, *Opnd*1)

vincula(**exp_menos**(*Opnd*)) :
vincula(*Opnd*)

vincula(**exp_not**(*Opnd*)) :
vincula(*Opnd*)

vincula(**exp_index**(*Opnd*0, *Opnd*1)) :
vincula(*Opnd*0)
vincula(*Opnd*1)

vincula(**exp_reg**(*Opnd*, *iden*)) :
vincula(*Opnd*)

vincula(**exp_indir**(*Opnd*)) :
vincula(*Opnd*)

vincula(**exp_entero**(*litEntero*)) : **noop**

vincula(**exp_real**(*litReal*)) : **noop**

vincula(**exp_true**()) : **noop**

vincula(**exp_false**()) : **noop**

vincula(**exp_cadena**(*litCadena*)) : **noop**

vincula(**exp_iden**(*iden*)) :
$.*vinculo* = **vinculoDe**(*ts*, *iden*)

```
    if $.vinculo == ⊥ then
        error
    endif

vincula(exp_null()) : noop

vinculaExpBin(Opnd0, Opnd1) :
    vincula(Opnd0)
    vincula(Opnd1)
```

2 Especificación del Procesamiento de Pre-tipado

2.1. Funciones para el conjunto de campos

- `nuevoConjunto()`: Crea un conjunto vacío para almacenar los campos de un registro
- `contiene(set,id)`: Comprueba si el conjunto `set` contiene ya una entrada para el identificador `id`.
- `inserta(set,id)`: Inserta el identificador `id` en el conjunto `set`.

2.2. Funciones de procesamiento

var set

pretipado(**bloque**(*SecDecs*, *SecIs*)) :
    pretipado(*SecDecs*)
    pretipado(*SecIs*)

2.2.1. Declaraciones

pretipado(**si_decs**(*LDecs*)) :
    pretipado(*LDecs*)

pretipado(**no_decs()**) : **noop**

pretipado(**muchas_decs**(*LDecs*, *Dec*)) :
    pretipado(*LDecs*)
    pretipado(*Dec*)

pretipado(**una_dec**(*Dec*)) :
    pretipado(*Dec*)

pretipado(**dec_base**(*TipoNom*)) :
    pretipado(*TipoNom*)

pretipado(**dec_type**(*TipoNom*)) :
    pretipado(*TipoNom*)

pretipado(**dec_proc**(*iden*, *ParamFs*, *Bloq*)) :
    pretipado(*ParamFs*)
    pretipado(*Bloq*)

pretipado(**si_params_f**(*LParamFs*)) :
    pretipado(*LParamFs*)

pretipado(**no_params_f()**) : **noop**

pretipado(**muchos_params_f**(*LParamFs*, *ParamF*)) :
    pretipado(*LParamFs*)
    pretipado(*ParamF*)

pretipado(**un_param_f**(*ParamF*)) :
    pretipado(*ParamF*)

pretipado(**si_refparam_f**(*Tipo*, *iden*)) :
    pretipado(*Tipo*)

pretipado(**no_refparam_f**(*Tipo*, *iden*)) :
    pretipado(*Tipo*)

2.2.2. Tipos

pretipado(**tipo_nombre**(*Tipo*, *iden*)) :
    pretipado(*Tipo*)

pretipado(**tipo_array**(*Tipo*, *litEntero*)) :
    if litEntero $<$ 0 *then*
        error
    endif
    pretipado(*Tipo*)

pretipado(**tipo_indir**(*Tipo*)) :
    pretipado(*Tipo*)

pretipado(**tipo_struct**(*LCampos*)) :
    var tmp = *set*
    set = *nuevoConjunto*()
    pretipado(*LCampos*)
    set = *tmp*

pretipado(**tipo_int**()) : **noop**

pretipado(**tipo_real**()) : **noop**

pretipado(**tipo_bool**()) : **noop**

pretipado(**tipo_string**()) : **noop**

pretipado(**tipo_type**(*iden*)) :
    if $.*vinculo* != *dec_type*(_) *then*
        error
    endif

pretipado(**muchos_campos**(*LCampos*, *TipoNom*(*Tipo*, *iden*))) :
    pretipado(*LCampos*)
    pretipado(*TipoNom*)
    if contiene(*set*, *iden*) *then*
        error
    else
        inserta(*set*, *iden*)
    endif

pretipado(**un_campo**(*TipoNom*(*Tipo*, *iden*))) :
    pretipado(*TipoNom*)
    if contiene(*set*, *iden*) *then*
        error
    else
        inserta(*set*, *iden*)
    endif

2.2.3. Instrucciones

pretipado(**si_ins**(LIs)) :
    pretipado(LIs)

pretipado(**no_ins**()) : **noop**

pretipado(**muchas_ins**(LIs, I)) :
    pretipado(LIs)
    pretipado(I)

pretipado(**una_ins**(I)) :
    pretipado(I)

pretipado(**ins_eval**(Exp)) : **noop**

pretipado(**ins_if**(Exp, $Bloq$)) :
    pretipado($Bloq$)

pretipado(**ins_if_else**(I, $Bloq$)) :
    pretipado(I)
    pretipado($Bloq$)

pretipado(**ins_while**(Exp, $Bloq$)) :
    pretipado($Bloq$)

pretipado(**ins_read**(Exp)) : **noop**

pretipado(**ins_write**(Exp)) : **noop**

pretipado(**ins_nl**()) : **noop**

pretipado(**ins_new**(Exp)) : **noop**

pretipado(**ins_delete**(Exp)) : **noop**

pretipado(**ins_call**($iden$, $ParamRs$)) : **noop**

pretipado(**ins_bloque**($Bloq$)) :
    pretipado($Bloq$)

pretipado(**si_params_r**($LParamRs$)) : **noop**

pretipado(**no_params_r**()) : **noop**

pretipado(**muchos_params_r**($LParamRs$, Exp)) : **noop**

pretipado(**un_param_r**(Exp)) : **noop**

2.2.4. Expresiones

pretipado(**exp_asig**($Opnd0$, $Opnd1$)) : **noop**

pretipado(**exp_menor**($Opnd0$, $Opnd1$)) : **noop**

pretipado(**exp_menor_ig**($Opnd0$, $Opnd1$)) : **noop**

pretipado(**exp_mayor**($Opnd0$, $Opnd1$)) : **noop**

pretipado(**exp_mayor_ig**($Opnd0$, $Opnd1$)) : **noop**

pretipado(**exp_ig**(*Opnd*0, *Opnd*1)) : **noop**

pretipado(**exp_dist**(*Opnd*0, *Opnd*1)) : **noop**

pretipado(**exp_suma**(*Opnd*0, *Opnd*1)) : **noop**

pretipado(**exp_resta**(*Opnd*0, *Opnd*1)) : **noop**

pretipado(**exp_and**(*Opnd*0, *Opnd*1)) : **noop**

pretipado(**exp_or**(*Opnd*0, *Opnd*1)) : **noop**

pretipado(**exp_mul**(*Opnd*0, *Opnd*1)) : **noop**

pretipado(**exp_div**(*Opnd*0, *Opnd*1)) : **noop**

pretipado(**exp_mod**(*Opnd*0, *Opnd*1)) : **noop**

pretipado(**exp_menos**(*Opnd*)) : **noop**

pretipado(**exp_not**(*Opnd*)) : **noop**

pretipado(**exp_index**(*Opnd*0, *Opnd*1)) : **noop**

pretipado(**exp_reg**(*Opnd*, *iden*)) : **noop**

pretipado(**exp_indir**(*Opnd*)) : **noop**

pretipado(**exp_entero**(*litEntero*)) : **noop**

pretipado(**exp_real**(*litReal*)) : **noop**

pretipado(**exp_true**()) : **noop**

pretipado(**exp_false**()) : **noop**

pretipado(**exp_cadena**(*litCadena*)) : **noop**

pretipado(**exp_iden**(*iden*)) : **noop**

pretipado(**exp_null**()) : **noop**

3 Especificación del Procesamiento de Comprobación de Tipos

3.1. Funciones para el conjunto de pares de tipos

- `nuevoConjunto()`: Crea un conjunto vacío para almacenar los pares de tipos
- `contiene(set,tipo0,tipo1)`: Comprueba si el conjunto `set` contiene ya una entrada para el par (`tipo0`, `tipo1`).
- `inserta(set,tipo0,tipo1)`: Inserta el par (`tipo0`, `tipo1`) en el conjunto `set`.

3.2. Funciones de procesamiento

var set

tipado(**bloque**(*SecDecs*, *SecIs*)) :
tipado(*SecDecs*)
tipado(*SecIs*)
$.*tipo* = **ambos_ok**(*SecDecs.tipo*, *SecIs.tipo*)

3.2.1. Declaraciones

tipado(**si_decs**(*LDecs*)) :
tipado(*LDecs*)
$.*tipo* = *LDecs.tipo*

tipado(**no_decs()**) :
$.*tipo* = *ok*

tipado(**muchas_decs**(*LDecs*, *Dec*)) :
tipado(*LDecs*)
tipado(*Dec*)
$.*tipo* = **ambos_ok**(*LDecs.tipo*, *Dec.tipo*)

tipado(**una_dec**(*Dec*)) :
tipado(*Dec*)
$.*tipo* = *Dec.tipo*

tipado(**dec_base**(*TipoNom*)) :
$.*tipo* = *ok*

tipado(**dec_type**(*TipoNom*)) :
$.*tipo* = *ok*

tipado(**dec_proc**(*iden*, *ParamFs*, *Bloq*)) :
tipado(*Bloq*)
$.*tipo* = *Bloq.tipo*

tipado(**si_params_f**(*LParamFs*)) : **noop**

tipado(**no_params_f()**) : **noop**

tipado(**muchos_params_f**(*LParamFs*, *ParamF*)) : **noop**

tipado(un_param_f(*ParamF***)) : noop**

tipado(si_refparam_f(*Tipo*, *iden***)) : noop**

tipado(no_refparam_f(*Tipo*, *iden***)) : noop**

3.2.2. Tipos

tipado(tipo_nombre(*Tipo*, *iden***)) : noop**

tipado(tipo_array(*Tipo*, *litEntero***)) : noop**

tipado(tipo_indir(*Tipo***)) : noop**

tipado(tipo_struct(*LCampos***)) : noop**

tipado(tipo_int()) : noop

tipado(tipo_real()) : noop

tipado(tipo_bool()) : noop

tipado(tipo_string()) : noop

tipado(tipo_type(*iden***)) : noop**

tipado(muchos_campos(*LCampos*, *TipoNom***)) : noop**

tipado(un_campo(*TipoNom***)) : noop**

3.2.3. Instrucciones

tipado(si_ins(*LIs***)) :**
    tipado(*LIs***)**
    $.*tipo* = *LIs.tipo*

tipado(no_ins()) :
    $.*tipo* = *ok*

tipado(muchas_ins(*LIs*, *I***)) :**
    tipado(*LIs***)**
    tipado(*I***)**
    $.*tipo* = **ambos_ok(***LIs.tipo*, *I.tipo***)**

tipado(una_ins(*I***)) :**
    tipado(*I***)**
    $.*tipo* = *I.tipo*

tipado(ins_eval(*Exp***)) :**
    tipado(*Exp***)**
    if Exp.tipo != *error then*
        $.*tipo* = *ok*
    else
        $.*tipo* = *error*
    endif

tipado(ins_if(*Exp*, *Bloq***)) :**
    tipado(*Exp***)**

```
    tipado(Bloq)
    if ref!(Exp.tipo) == tipo_bool() ∧ Bloq.tipo == ok then
        $.tipo = ok
    else
        $.tipo = error
        error
    endif

tipado(ins_if_else(I, Bloq)) :
    tipado(I)
    tipado(Bloq)
    $.tipo = ambos_ok(I.tipo, Bloq.tipo)

tipado(ins_while(Exp, Bloq)) :
    tipado(Exp)
    tipado(Bloq)
    if ref!(Exp.tipo) == tipo_bool() ∧ Bloq.tipo == ok then
        $.tipo = ok
    else
        $.tipo = error
        error
    endif

tipado(ins_read(Exp)) :
    tipado(Exp)
    let T = ref!(Exp.tipo) in
        if T == tipo_int() ∨
            T == tipo_real() ∨
            T == tipo_string() then
            if es_designador(Exp) then
                $.tipo = ok
            else
                $.tipo = error
                error
            endif
        else
            $.tipo = error
            error
        endif
    end let

tipado(ins_write(Exp)) :
    tipado(Exp)
    let T = ref!(Exp.tipo) in
        if (T == tipo_int() ∨
            T == tipo_real() ∨
            T == tipo_bool() ∨
            T == tipo_string()) then
            $.tipo = ok
        else
            $.tipo = error
            error
        endif
    end let

tipado(ins_nl()) :
    $.tipo = ok

tipado(ins_new(Exp)) :
    tipado(Exp)
```

if **ref!**(*Exp.tipo*) == *tipo_indir*(_) *then*
$.*tipo* = *ok*
else
$.*tipo* = *error*
error
endif

tipado(**ins_delete**(*Exp*)) :
tipado(*Exp*)
if **ref!**(*Exp.tipo*) == *tipo_indir*(_) *then*
$.*tipo* = *ok*
else
$.*tipo* = *error*
error
endif

tipado(**ins_call**(*iden*, *ParamRs*)) :
tipado(*ParamRs*)
if $.*vinculo* != *dec_proc*(_, *ParamFs*, _) *then*
if ParamRs == *no_params_r*() ∧ *ParamFs* == *no_params_f*() *then*
$.*tipo* = *ok*
else if ParamRs == *si_params_r*(*LParamRs*) ∧ *ParamFs* == *si_params_f*(*LParamsFs*) *then*
if **num_params**(*LParamRs*, *LParamFs*) *then*
if **compatibles_params**(*LParamRs*, *LParamFs*) *then*
$.*tipo* = *ok*
else
$.*tipo* = *error*
endif
else
$.*tipo* = *error*
error
endif
else
$.*tipo* = *error*
error
endif

tipado(**ins_bloque**(*Bloq*)) :
tipado(*Bloq*)
$.*tipo* = *Bloq.tipo*

tipado(**si_params_r**(*LParamRs*)) :
tipado(*LParamRs*)

tipado(**no_params_r**()) : **noop**

tipado(**muchos_params_r**(*LParamRs*, *Exp*)) :
tipado(*LParamRs*)
tipado(*Exp*)

tipado(**un_param_r**(*Exp*)) :
tipado(*Exp*)

3.2.4. Expresiones

tipado(**exp_asig**(*Opnd*0, *Opnd*1)) :
    tipado(*Opnd*0)
    tipado(*Opnd*1)
    if **es_designador**(*Opnd*0) *then*
        if **compatibles**(*Opnd*0.*tipo*, *Opnd*1.*tipo*) *then*
            $.*tipo* = *ok*
        else
            $.*tipo* = *error*
            error
        endif
    else
        $.*tipo* = *error*
        error
    endif

tipado(**exp_menor**(*Opnd*0, *Opnd*1)) :
    tipado_rel(*Opnd*0, *Opnd*1, $)

tipado(**exp_menor_ig**(*Opnd*0, *Opnd*1)) :
    tipado_rel(*Opnd*0, *Opnd*1, $)

tipado(**exp_mayor**(*Opnd*0, *Opnd*1)) :
    tipado_rel(*Opnd*0, *Opnd*1, $)

tipado(**exp_mayor_ig**(*Opnd*0, *Opnd*1)) :
    tipado_rel(*Opnd*0, *Opnd*1, $)

tipado(**exp_ig**(*Opnd*0, *Opnd*1)) :
    tipado_rel(*Opnd*0, *Opnd*1, $)

tipado(**exp_dist**(*Opnd*0, *Opnd*1)) :
    tipado_rel(*Opnd*0, *Opnd*1, $)

tipado(**exp_suma**(*Opnd*0, *Opnd*1)) :
    tipado_arit(*Opnd*0, *Opnd*1, $)

tipado(**exp_resta**(*Opnd*0, *Opnd*1)) :
    tipado_arit(*Opnd*0, *Opnd*1, $)

tipado(**exp_and**(*Opnd*0, *Opnd*1)) :
    tipado_logic(*Opnd*0, *Opnd*1, $)

tipado(**exp_or**(*Opnd*0, *Opnd*1)) :
    tipado_logic(*Opnd*0, *Opnd*1, $)

tipado(**exp_mul**(*Opnd*0, *Opnd*1)) :
    tipado_arit(*Opnd*0, *Opnd*1, $)

tipado(**exp_div**(*Opnd*0, *Opnd*1)) :
    tipado_arit(*Opnd*0, *Opnd*1, $)

tipado(**exp_mod**(*Opnd*0, *Opnd*1)) :
    tipado(*Opnd*0)
    tipado(*Opnd*1)
    if **ref!**(*Opnd*0.*tipo*) == *tipo_int*() ∧
        ref!(*Opnd*1.*tipo*) == *tipo_int*() *then*
          $.*tipo* = *tipo_int*()
    else

```
            $.tipo = error
            aviso_error_bin(T0, T1)
        endif

tipado(exp_menos(Opnd)) :
        tipado(Opnd)
        let T = ref!(Opnd.tipo) in
            if T == tipo_int() then
                $.tipo = tipo_int()
            else if T == tipo_real() then
                $.tipo = tipo_real()
            else
                $.tipo = error
                aviso_error_un(T)
            endif
        end let

tipado(exp_not(Opnd)) :
        tipado(Opnd)
        if ref!(Opnd.tipo) == tipo_bool() then
            $.tipo = tipo_bool()
        else
            $.tipo = error
            aviso_error_un(T)
        endif

tipado(exp_index(Opnd0, Opnd1)) :
        tipado(Opnd0)
        tipado(Opnd1)
        if ref!(Opnd0.tipo) == tipo_array(Tipo, _) ∧
             ref!(Opnd1.tipo) == tipo_int() then
            $.tipo = tipo_array(Tipo, _)
        else
            $.tipo = error
            error
        endif

tipado(exp_reg(Opnd, iden)) :
        tipado(Opnd)
        if ref!(Opnd.tipo) == tipo_struct(LCampos) then
            let C = campo_struct(LCampos, iden) in
                if C == error then
                    error
                endif
                $.tipo = C
            end let
        else
            $.tipo = error
            error
        endif

tipado(exp_indir(Opnd)) :
        tipado(Opnd)
        if ref!(Opnd.tipo) == tipo_indir(Tipo) then
            $.tipo = tipo_indir(Tipo)
        else
            $.tipo = error
            error
        endif
```

tipado(**exp_entero**(*litEntero*)) :
    $\$.tipo = tipo\_int()$

tipado(**exp_real**(*litReal*)) :
    $\$.tipo = tipo\_real()$

tipado(**exp_true**()) :
    $\$.tipo = tipo\_bool()$

tipado(**exp_false**()) :
    $\$.tipo = tipo\_bool()$

tipado(**exp_cadena**(*litCadena*)) :
    $\$.tipo = tipo\_string()$

tipado(**exp_iden**(*iden*)) :
    if $\$.vinculo = dec\_base(TipoNom)$ *then*
        let $TipoNom = TipoNom(Tipo, iden)$ *in*
            $\$.tipo = Tipo$
        end let
    else if $\$.vinculo = si\_refparam\_f(Tipo, \_) \vee \$.vinculo = no\_refparam\_f(Tipo, \_)$ *then*
        $\$.tipo = Tipo$
    else
        $\$.tipo = error$
        error
    endif

tipado(**exp_null**()) :
    $\$.tipo = null$

tipado_arit($Opnd0$, $Opnd1$, Exp) :
    tipado($Opnd0$)
    tipado($Opnd1$)
    let $T0 =$ **ref!**($Opnd0.tipo$) $\wedge$ $T1 =$ **ref!**($Opnd1.tipo$) *in*
        if $T0 == tipo\_int() \wedge$
            $T1 == tipo\_int()$ *then*
            $Exp.tipo = tipo\_int()$
        else if $(T0 == tipo\_real() \vee$
            $T0 == tipo\_int()) \wedge$
            $(T1 == tipo\_real() \vee$
            $T1 == tipo\_int())$ *then*
            $Exp.tipo = tipo\_real()$
        else
            $Exp.tipo = error$
            aviso_error_bin($T0$, $T1$)
        endif
    end let

tipado_logic($Opnd0$, $Opnd1$, Exp) :
    tipado($Opnd0$)
    tipado($Opnd1$)
    let $T0 =$ **ref!**($Opnd0.tipo$) $\wedge$ $T1 =$ **ref!**($Opnd1.tipo$) *in*
        if $T0 == tipo\_bool() \wedge$
            $T1 == tipo\_bool()$ *then*
            $Exp.tipo = tipo\_bool()$
        else
            $Exp.tipo = error$
            aviso_error_bin($T0$, $T1$)
        endif
    end let

```
tipado_rel(Opnd0, Opnd1, Exp) :
    tipado(Opnd0)
    tipado(Opnd1)
    let T0 = ref!(Opnd0.tipo) ∧ T1 = ref!(Opnd1.tipo) in
        if (T0 == tipo_int() ∨
            T0 == tipo_real()) ∧
           (T1 == tipo_int() ∨
            T1 == tipo_real()) then
              Exp.tipo = tipo_bool()
        else if T0 == tipo_bool() ∧
            T1 == tipo_bool() then
              Exp.tipo = tipo_bool()
        else if T0 == tipo_string() ∧
            T1 == tipo_string() then
              Exp.tipo = tipo_string()
        else
            if Exp == exp_ig(_, _) ∨
               Exp == exp_dist(_, _) then
                  tipado_rel_indir(Opnd0, Opnd1, Exp)
            else
                  Exp.tipo = error
                  aviso_error_bin(T0, T1)
            endif
        endif
    end let

tipado_rel_indir(T0, T1, Exp) :
    if (T0 == tipo_indir() ∨
        T0 == null) ∧
        (T1 == tipo_indir() ∧
        T1 == null) then
          Exp.tipo = tipo_bool()
      else
          Exp.tipo = error
          aviso_error_bin(T0, T1)
    endif
```

3.3. Funciones auxiliares

```
aviso_error_bin(T0, T1) :
    if T0 != error ∧ T1 != error then
          error
    endif

aviso_error_un(T) :
    if T != error then
          error
    endif

ambos_ok(T0, T1) :
    if T0 == ok ∧ T1 == ok then
          return ok
    else
          return error
    endif

ref!(T) :
    if T == tipo_type(_) then
```

```
            let T.vinculo = dec_type(Tipo, iden) in
                  return ref!(Tipo)
            end let
      else
            return T
      endif

es_designador(E) :
      return E == exp_iden(_) ∨
               E == exp_index(_, _) ∨
               E == exp_reg(_, _) ∨
               E == exp_indir(_)

compatibles(T0, T1) :
      set = nuevoConjunto()
      inserta(set, T0, T1)
      return unificables(T0, T1)

unificables(T0, T1) :
      let T0' = ref!(T0) ∧ T1' = ref!(T1) in
            if T0' == tipo_array(Tipo0, _) ∧ T1' == tipo_array(Tipo1, _) then
                  return son_unificables(Tipo0, Tipo1)
            else if T0' == tipo_struct(LCampos0) ∧ T1' == tipo_struct(LCampos1) then
                  return son_unificables_struct(LCampos0, LCampos1)
            else if T0' == tipo_indir(_) ∧ T1' == null then
                  return true
            else if T0' == tipo_indir(Tipo0) ∧ T1' == tipo_indir(Tipo1) then
                  return son_unificables(Tipo0, Tipo1)
            else if T0' == T1' ∨ (T0' == tipo_real() ∧ T1' == tipo_int()) then
                  return true
            else
                  return false
            endif
      end let

son_unificables_struct(LCampos0, LCampos1) :
      if LCampos0 == un_campo(TipoNom0) ∧ LCampos1 == un_campo(TipoNom1) then
            let TipoNom0 = TipoNom(Tipo0, _) ∧ TipoNom1 = TipoNom(Tipo1, _)
                  return son_unificables(Tipo0, Tipo1)
            end let
      else if LCampos0 == muchos_campos(LCampos0, TipoNom0) ∧
            LCampos1 == muchos_campos(LCampos1, TipoNom1) then
            let TipoNom0 = TipoNom(Tipo0, _) ∧ TipoNom1 = TipoNom(Tipo1, _)
                  return son_unificables(Tipo0, Tipo1) ∧
                        son_unificables_struct(LCampos0, LCampos1)
            end let
      else
            return false
      endif

son_unificables(T0, T1) :
      if contiene(set, T0, T1) then
            return true
      else
            inserta(set, T0, T1)
            return unificables(T0, T1)
      endif

campo_struct(LCampos, iden) :
      if LCampos == un_campo(TipoNom0) then
```

```
            let TipoNom0 = TipoNom(Tipo0, iden0)
                if iden == iden0 then
                        return Tipo0
                else
                        return error
                endif
            end let
        else if LCampos == muchos_campos(LCampos0, TipoNom0) then
            let TipoNom0 = TipoNom(Tipo0, iden0)
                if iden == iden0 then
                        return Tipo0
                else
                        return campo_struct(LCampos0, iden)
                endif
            end let
        endif

num_params(LParamRs, LParamFs) :
        if LParamRs == un_param_r(_) ∧ LParamFs == un_param_f(_) then
            return true
        else if LParamRs == muchos_params_r(LParamRs0, _) ∧
            LParamFs == muchos_params_f(LParamFs0, _) then
            return num_params(LParamRs0 LParamFs0)
        else
            return false
        endif

compatibles_params(LParamRs, LParamFs) :
        if LParamRs == un_param_r(Exp) ∧ LParamFs == un_param_f(ParamF) then
            return param_r_f(Exp, ParamF)
        else if LParamRs == muchos_params_r(LParamRs0, Exp) ∧
            LParamFs == muchos_params_f(LParamFs0, Tipo) then
            return param_r_f(Exp, ParamF) ∧
                    compatibles_params(LParamRs0 LParamFs0)
        endif

param_r_f(Exp, ParamF) :
        if ParamF == si_refparam_f(Tipo, _) then
            if es_designador(Exp) then
                    if ParamF == tipo_real() then
                        if Exp.tipo == tipo_real() then
                                return true
                        else
                                error
                                return false
                        endif
                    else
                        if compatibles(Tipo, Exp.tipo) then
                                return true
                        else
                                error
                                return false
                        endif
                    endif
            else
                    error
                    return false
            endif
        else
            if compatibles(Tipo, Exp.tipo) then
```

```
            return true
        else
            error
            return false
        endif
    endif
```

4 Especificación del Procesamiento de Asignación de Espacio

4.1. Funciones de procesamiento

var dir = 0
var max_dir = 0
var nivel = 0
var campos

asig_espacio(**bloque**(*SecDecs*, *SecIs*)) :
    var dir_ant = dir
    asig_espacio(*SecDecs*)
    asig_espacio(*SecIs*)
    dir = dir_ant

4.1.1. Declaraciones

asig_espacio(**si_decs**(*LDecs*)) :
    asig_espacio1(*LDecs*)
    asig_espacio2(*LDecs*)

asig_espacio(**no_decs**()) : **noop**

asig_espacio1(**muchas_decs**(*LDecs*, *Dec*)) :
    asig_espacio1(*LDecs*)
    asig_espacio1(*Dec*)

asig_espacio2(**muchas_decs**(*LDecs*, *Dec*)) :
    asig_espacio2(*LDecs*)
    asig_espacio2(*Dec*)

asig_espacio1(**una_dec**(*Dec*)) :
    asig_espacio1(*Dec*)

asig_espacio2(**una_dec**(*Dec*)) :
    asig_espacio2(*Dec*)

asig_espacio1(**dec_base**(*TipoNom*)) :
    let TipoNom = TipoNom(Tipo, iden) in
        asig_tam1(*Tipo*)
        $.*dir* = *dir*
        $.*nivel* = *nivel*
        inc_dir(*Tipo.tam*)
    end let

asig_espacio2(**dec_base**(*TipoNom*)) :
    let TipoNom = TipoNom(Tipo, iden) in
        asig_tam1(*TipoNom*)
    end let

asig_espacio1(**dec_type**(*TipoNom*)) :
    let TipoNom = TipoNom(Tipo, iden) in
        asig_tam1(*Tipo*)

$\$.dir = dir$
$\$.nivel = nivel$
$inc\_dir(Tipo.tam)$
end let

asig_espacio2(**dec_type**($TipoNom$)) :
let $TipoNom = TipoNom(Tipo,\ iden)$ *in*
asig_tam1($TipoNom$)
end let

asig_espacio1(**dec_proc**($iden,\ ParamFs,\ Bloq$)) :
$var\ dir\_ant = dir$
$var\ max\_dir\_ant = max\_dir$
$nivel++$
$\$.nivel = nivel$
$dir = 0$
$max\_dir = 0$
asig_espacio1($ParamFs$)
asig_espacio2($ParamFs$)
asig_espacio($Bloq$)
$\$.tam = max\_dir$
$dir = dir\_ant$
$max\_dir = max\_dir\_ant$
$nivel--$

asig_espacio2(**dec_proc**($iden,\ ParamFs,\ Bloq$)) : **noop**

asig_espacio1(**si_params_f**($LParamFs$)) :
asig_espacio1($LParamFs$)

asig_espacio2(**si_params_f**($LParamFs$)) :
asig_espacio2($LParamFs$)

asig_espacio1(**no_params_f()**) : **noop**

asig_espacio2(**no_params_f()**) : **noop**

asig_espacio1(**muchos_params_f**($LParamFs,\ ParamF$)) :
asig_espacio1($LParamFs$)
asig_espacio1($ParamF$)

asig_espacio2(**muchos_params_f**($LParamFs,\ ParamF$)) :
asig_espacio2($LParamFs$)
asig_espacio2($ParamF$)

asig_espacio1(**un_param_f**($ParamF$)) :
asig_espacio1($ParamF$)

asig_espacio2(**un_param_f**($ParamF$)) :
asig_espacio2($ParamF$)

asig_espacio1(**si_refparam_f**($Tipo,\ iden$)) :
asig_tam1($Tipo$)
$\$.dir = dir$
$\$.nivel = nivel$
$inc\_dir(1)$

asig_espacio2(**si_refparam_f**($Tipo,\ iden$)) :
asig_espacio2($Tipo$)

asig_espacio1(**no_refparam_f**(*Tipo*, *iden*)) :
    asig_tam1(*Tipo*)
    $\$.dir = dir$
    $\$.nivel = nivel$
    inc_dir(*Tipo.tam*)

asig_espacio2(**no_refparam_f**(*Tipo*, *iden*)) :
    asig_espacio2(*Tipo*)

4.1.2. Tipos

asig_tam1(**tipo_array**(*Tipo*, *litEntero*)) :
    asig_tam1(*Tipo*)
    $\$.tam = Tipo.tam * litEntero$

asig_tam2(**tipo_array**(*Tipo*, *litEntero*)) : **noop**

asig_tam1(**tipo_indir**(*Tipo*)) :
    if $Tipo\ != tipo\_type(\_)$ *then*
        asig_tam1(*Tipo*)
    endif
    $\$.tam = 1$

asig_tam2(**tipo_indir**(*Tipo*)) :
    if $Tipo == tipo\_type(iden)$ *then*
        let $Tipo.vinculo = dec\_type(TipoNom) \wedge TipoNom = TipoNom(T, iden)$ *in*
            $Tipo.tam = T.tam$
        end let
    else
        asig_tam2(*Tipo*)
    endif

asig_tam1(**tipo_struct**(*LCampos*)) :
    $campos = 0$
    asig_tam1(*LCampos*)
    $\$.tam = campos$

asig_tam2(**tipo_struct**(*LCampos*)) :
    asig_tam2(*LCampos*)

asig_tam1(**tipo_int**()) :
    $\$.tam = 1$

asig_tam2(**tipo_int**()) : **noop**

asig_tam1(**tipo_real**()) :
    $\$.tam = 1$

asig_tam2(**tipo_real**()) : **noop**

asig_tam1(**tipo_bool**()) :
    $\$.tam = 1$

asig_tam2(**tipo_bool**()) : **noop**

asig_tam1(**tipo_string**()) :
    $\$.tam = 1$

asig_tam2(**tipo_string**()) : **noop**

asig_tam1(**tipo_type**(*iden*)) :
    let $.*vinculo* = *dec_type*(*TipoNom*) ∧ *TipoNom* = *TipoNom*(*T*, *iden*) *in*
        $.*tam* = *T.tam*
    end let

asig_tam2(**tipo_type**(*iden*)) : **noop**

asig_tam1(**muchos_campos**(*LCampos*, *TipoNom*)) :
    asig_tam1(*LCampos*)
    let TipoNom = *TipoNom*(*T*, *iden*) *in*
        asig_tam1(*T*)
        campos += *T.tam*
    end let

asig_tam2(**muchos_campos**(*LCampos*, *TipoNom*)) :
    asig_tam2(*LCampos*)
    let TipoNom = *TipoNom*(*T*, *iden*) *in*
        asig_tam2(*T*)
    end let

asig_tam1(**un_campo**(*TipoNom*)) :
    let TipoNom = *TipoNom*(*T*, *iden*) *in*
        asig_tam1(*T*)
        campos += *T.tam*
    end let

asig_tam2(**un_campo**(*TipoNom*)) :
    let TipoNom = *TipoNom*(*T*, *iden*) *in*
        asig_tam2(*T*)
    end let

4.1.3. Instrucciones

asig_espacio(**si_ins**(*LIs*)) :
    asig_espacio(*LIs*)

asig_espacio(**no_ins**()) : **noop**

asig_espacio(**muchas_ins**(*LIs*, *I*)) :
    asig_espacio(*LIs*)
    asig_espacio(*I*)

asig_espacio(**una_ins**(*I*)) :
    asig_espacio(*I*)

asig_espacio(**ins_eval**(*Exp*)) : **noop**

asig_espacio(**ins_if**(*Exp*, *Bloq*)) :
    asig_espacio(*Bloq*)

asig_espacio(**ins_if_else**(*I*, *Bloq*)) :
    asig_espacio(*I*)
    asig_espacio(*Bloq*)

asig_espacio(**ins_while**(*Exp*, *Bloq*)) :
    asig_espacio(*Bloq*)

asig_espacio(**ins_read**(*Exp*)) : **noop**

asig_espacio(**ins_write**(*Exp*)) : **noop**

asig_espacio(**ins_nl**()) : **noop**

asig_espacio(**ins_new**(*Exp*)) : **noop**

asig_espacio(**ins_delete**(*Exp*)) : **noop**

asig_espacio(**ins_call**(*iden*, *ParamRs*)) : **noop**

asig_espacio(**ins_bloque**(*Bloq*)) :
asig_espacio(*Bloq*)

asig_espacio(**si_params_r**(*LParamRs*)) : **noop**

asig_espacio(**no_params_r**()) : **noop**

asig_espacio(**muchos_params_r**(*LParamRs*, *Exp*)) : **noop**

asig_espacio(**un_param_r**(*Exp*)) : **noop**

4.1.4. Expresiones

asig_espacio(**exp_asig**(*Opnd*0, *Opnd*1)) : **noop**

asig_espacio(**exp_menor**(*Opnd*0, *Opnd*1)) : **noop**

asig_espacio(**exp_menor_ig**(*Opnd*0, *Opnd*1)) : **noop**

asig_espacio(**exp_mayor**(*Opnd*0, *Opnd*1)) : **noop**

asig_espacio(**exp_mayor_ig**(*Opnd*0, *Opnd*1)) : **noop**

asig_espacio(**exp_ig**(*Opnd*0, *Opnd*1)) : **noop**

asig_espacio(**exp_dist**(*Opnd*0, *Opnd*1)) : **noop**

asig_espacio(**exp_suma**(*Opnd*0, *Opnd*1)) : **noop**

asig_espacio(**exp_resta**(*Opnd*0, *Opnd*1)) : **noop**

asig_espacio(**exp_and**(*Opnd*0, *Opnd*1)) : **noop**

asig_espacio(**exp_or**(*Opnd*0, *Opnd*1)) : **noop**

asig_espacio(**exp_mul**(*Opnd*0, *Opnd*1)) : **noop**

asig_espacio(**exp_div**(*Opnd*0, *Opnd*1)) : **noop**

asig_espacio(**exp_mod**(*Opnd*0, *Opnd*1)) : **noop**

asig_espacio(**exp_menos**(*Opnd*)) : **noop**

asig_espacio(**exp_not**(*Opnd*)) : **noop**

asig_espacio(**exp_index**(*Opnd*0, *Opnd*1)) : **noop**

asig_espacio(**exp_reg**(*Opnd*, *iden*)) : **noop**

asig_espacio(**exp_indir**(*Opnd*)) : **noop**

asig_espacio(**exp_entero**(*litEntero*)) : **noop**

asig_espacio(**exp_real**(*litReal*)) : **noop**

asig_espacio(**exp_true**()) : **noop**

asig_espacio(**exp_false**()) : **noop**

asig_espacio(**exp_cadena**(*litCadena*)) : **noop**

asig_espacio(**exp_iden**(*iden*)) : **noop**

asig_espacio(**exp_null**()) : **noop**

4.2. Funciones auxiliares

inc_dir(*inc*) :
$dir += inc$
$if\ dir > max\_dir\ then$
$max\_dir = dir$
$endif$

5 Descripción de las Instrucciones de la Máquina-p

6 Especificación del Procesamiento de Etiquetado

7 Especificación del Procesamiento de Generación de Código