Jorge Arévalo Echevarría

Ismail Azizi González

Sintaxis Abstracta

Constructora	**Explicación**
prog : Decs x Is → Prog	Construye un programa, dados una lista de declaraciones y una lista de instrucciones
decs_una : Dec → Decs	Construye una lista de declaraciones dada una declaración
decs_muchas : Decs x Dec → Decs	Construye una lista de declaraciones, dadas una lista de declaraciones y una declaración
decs_ninguna: → Decs	Construye una lista de declaraciones vacía
Is_una: I → Is	Construye una lista de instrucciones dada una instrucción
is_muchas: Is x I → Is	Construye una lista de instrucciones dada lista de instrucciones y una instrucción
is_niguna : → Is	Construye una lista de instrucciones vacía
is_asig : Exp x Exp → I	Construye una instrucción a partir de la expresión que representa la parte izquierda, y la expresión que representa la parte derecha
is_if_then: Exp x LIs → I	Construye una instrucción a partir de la expresión que representa la condicion(despues del if), y la lista de instrucciones que se realiza después del then
is_if_then_else: Exp x I x I → I	Construye una instrucción a partir de la expresión que representa la condicion(despues del if), la lista de instrucciones que se realiza después del then, y la lista de instrucciones que se realiza después del else
is_while : Exp x I → I	Construye una instrucción a partir de la expresión que representa la condición(después del while), y la lista de instrucciones que se realiza después del do
is_read: Exp → I	Construye una instrucción a partir de la expresión después de la palabra read.
is_write: Exp → I	Construye una instrucción a partir de la expresión después de la palabra write

is_nl : → I	Construye una instrucción a partir de la palabra clave nl
is_new : Exp → I	Construye una instrucción a partir de la expresión después de la palabra new
is_delete : Exp → I	Construye una instrucción a partir de la expresión después de la palabra delete
is_proc : Exp x Preals → I	Construye una instrucción a partir de la expresión y después de unos parámetros reales, para invocar a un procedimiento
is_seq : Decs x Is → I	Construye una instrucción a partir de una lista de declaraciones y después de una lista de instrucciones
to : → **TO**	Construye una terminacion Opcional con ;
to_ninguna : → **TO**	Construye una terminacion Opcional sin ;
pr_ninguno: → Preals	Construye una lista de parámetros reales vacía
pr_muchos: Preals x Exp → Preals	Construye una lista de varios parámetros reales
pr_uno: Exp → Preals	Construye una lista con un parámetro real
dec_var : Tipo x string → Dec	Construye una declaración de variable, dados su tipo y su nombre
dec_type : Tipo x string → Dec	Construye una declaración de tipo, dados el tipo y el nombre
dec_proc : string x Pforms x Decs x Is → Dec	Construye una declaración de procedimiento, dados un nombre(id), los parámetros formales, una lista de declaraciones y una sección de instrucciones.
pf_ninguno: → Pforms	Construye una lista de parámetros formales vacía
pf_muchos: Pforms x Pform → Pforms	Construye una lista de varios parámetros formales
pf_uno: Pform → Pforms	Construye una lista con un parámetro formal
p_valor: Tipo x string → Pform	Construye un parámetro formal por valor

p_var: Tipo x string → Pform	Construye un parámetro formal por variable
exp_menor : Exp x Exp → Exp	Construye una expresión a partir de una expresión, operador menor y otra expresión.
exp_mayor : Exp x Exp → Exp	Construye una expresión a partir de una expresión, operador mayor y otra expresión.
exp_menor_igual : Exp x Exp → Exp	Construye una expresión a partir de una expresión, operador menor igual y otra expresión.
exp_mayor_igual : Exp x Exp → Exp	Construye una expresión a partir de una expresión, operador mayor igual y otra expresión.
exp_igualdad : Exp x Exp → Exp	Construye una expresión a partir de una expresión, operador igualdad y otra expresión.
exp_distinto : Exp x Exp → Exp	Construye una expresión a partir de una expresión, operador distinto y otra expresión.
exp_suma : Exp x Exp → Exp	Construye una expresión a partir de una expresión, operador más y otra expresión.
exp_resta : Exp x Exp → Exp	Construye una expresión a partir de una expresión, operador menos y otra expresión.
exp_or: Exp x Exp → Exp	Construye una expresión a partir de una expresión, un operador or y otra expresión.
exp_and: Exp x Exp → Exp	Construye una expresión a partir de una expresión, un operador and y otra expresión.
exp_mul : Exp x Exp → Exp	Construye una expresión a partir de una expresión, operador multiplicación y otra expresión.
exp_div : Exp x Exp → Exp	Construye una expresión a partir de una expresión, operador división y otra expresión
exp_mod : Exp x Exp → Exp	Construye una expresión a partir de una expresión, operador porcentaje y otra expresión
exp_menos : Exp → Exp	Construye una expresión a partir de

	operador menos y una expresión
exp_not : Exp → Exp	Construye una expresión a partir de operador not y una expresión
exp_index : Exp → Exp	Construye una expresión a partir de operador de indexación de apertura, una expresión y un operador de indexación de cierre(Nivel 5 operador de indexación)
exp_acc_reg : Campo → Exp	Construye una expresión a partir de operador punto y un campo(Nivel 5 operador de acceso a registro)
exp_indireccion : →Exp	Construye una expresión a partir de operador ^ (Nivel 5 operador de indirreccion)
int : → Tipo	Construye el tipo int
real : → Tipo	Construye el tipo real
true : → Tipo	Construye el tipo bool con valor true
false : → Tipo	Construye el tipo bool con valor false
string : → Tipo	Construye el tipo string
ref: string → Tipo	Construye una referencia a un tipo declarado en una declaración de tipo
array : Tipo x int → Tipo	Construye un tipo array a partir del tipo base y del tamaño
campo : string x tipo →Campo	Construye un campo a partir de un id(string) seguido de un tipo
record : Campos → Tipo	Construye un tipo registro a partir de una lista de campos
campos_muchos : Campos x Campo → Campos	Construye una lista de campos, dados otra lista de campos y un último campo
campos_uno : Campo → Campos	Construye una lista de campos, dado un campo
pointer : Tipo → Tipo	Construye un tipo puntero, dado el tipo base
enteros : string → Exp	Construye una expresión que representa un literal entero numérico a partir de la

	cadena asociada a dicho literal
reales : string → Exp	Construye una expresión que representa un literal real numérico a partir de la cadena asociada a dicho literal
booleanos : string → Exp	Construye una expresión que representa un literal booleano a partir de la cadena asociada a dicho literal
cadena : string → Exp	Construye una expresión que representa un literal cadena a partir de la cadena asociada a dicho literal
id : string → Exp	Construye una expresión que representa una variable a partir del nombre de dicha variable
null : → Exp	Construye una expresión a partir de la cadena vacía
true : string → Exp	Construye una expresión a partir de la cadena true
false: string → Exp	Construye una expresión a partir de la cadena false
dref : Exp → Exp	Valor apuntado por un puntero (dereferencia)
acc : Exp x string → Exp	Acceso al campo (2º argumento) de un registro (1er argumento)
indx : Exp x Exp → Exp	Indexación de un elemento (2º argumento) en un array (1er argumento)

1) Vinculación

global ts //tabla de símbolos

vincula(**prog**(Ds,Is)) =
    vincula_decs_fase1(Ds)
    vincula_decs_fase2(Ds)
    vincula_is(Is)

// En la primera fase de vinculación de declaraciones, se recolectan los vínculos en la
// tabla de símbolos, y se vinculan todas las referencias a identificadores en los tipos,
// excepto aquellas precedidas por 'pointer'.

// primera pasada sobre declaraciones: se vinculan todos los identificadores, excepto
// los que aparecen en los 'refs' de 'pointer – ref'

vincula_decs_fase1(**decs_muchas**(Ds,D)) =
    vincula_decs_fase1(Ds)
    vincula_decs_fase1(D)

vincula_decs_fase1(**decs_una**(D)) =
    vincula_decs_fase1(D)

vincula_decs_fase1(**decs_ninguna**()) = skip //no hacer nada

recolecta(id, Dec) =
    si id_duplicado(ts,id) **entonces**
        error // id duplicado.
    si no
        añade(ts, id, Dec) // Se añade una entrada a la tabla de símbolos
                // que asocia la declaración al identificador

vincula_decs_fase1(**dec_var**(T,id)) =
    vincula_decs_fase1(T)
    recolecta(id,$) // se añade la declaración a la tabla de símbolos, controlando duplicados

vincula_decs_fase1(**dec_type**(T,id)) =
    vincula_decs_fase1(T)
    recolecta(id,$) // se añade la declaración a la tabla de símbolos, controlando duplicados

```
vincula_decs_fase1(dec_proc(id,Ps,Ds,Is)) =
        recolecta(id,$)
        abre_nivel(ts) // se abre el nivel del procedimiento en la tabla de símbolos
        vincula_decs_fase1(Ps)
        vincula_decs_fase1(Ds)
        vincula_decs_fase2(Ps)
        vincula_decs_fase2(Ds)
        vincula_is(Is)
        cierra_nivel(ts) // se elimina el nivel del procedimiento


vincula_decs_fase1(ref(id)) =
si existe_id(id,ts) entonces
        $.vinculo = valorDe(ts,id) // El tipo está declarado: se vincula
si no
        error // tipo no declarado

vincula_decs_fase1(int())=skip //nada que vincular
vincula_decs_fase1(real())=skip //nada que vincular
vincula_decs_fase1(bool())=skip //nada que vincular
vincula_decs_fase1(string())=skip //nada que vincular
vincula_decs_fase1(array(T,n)) =
        vincula_decs_fase1(T)

vincula_decs_fase1(record(Cs)) =
        vincula_decs_fase1(Cs)

vincula_decs_fase1(campo(Tipo, id)) =
        vincula_decs_fase1(Tipo)

vincula_decs_fase1(campos_uno(C)) =
        vincula_decs_fase1(C)

vincula_decs_fase1(campos_muchos(Cs,C)) =
        vincula_decs_fase1(Cs)
        vincula_decs_fase1(C)

vincula_decs_fase1(pointer(T)) =
        si T != ref(_) entonces // Se vincula únicamente si no se trata de un “pointer – ref”
                vincula_decs_fase1(T)


// En la segunda fase se vinculan las referencias precedidas por ‘pointer’ (los “pointer – ref”).
// Segunda pasada sobre declaraciones: se vinculan los ref en ‘pointer – ref’



vincula_decs_fase2(decs_muchas(Ds,D)) =
```

```
        vincula_decs_fase2(Ds)
        vincula_decs_fase2(D)


vincula_decs_fase2(decs_una(D)) =
        vincula_decs_fase2(D)

vincula_decs_fase2(decs_ninguna()) = skip //… porque ya se vinculó en la fase 1


vincula_decs_fase2(dec_var(T,id)) =
        vincula_decs_fase2(T)


vincula_decs_fase2(dec_type(T,id)) =
        vincula_decs_fase2(T)


vincula_decs_fase2(dec_proc(id,Ps,Ds,Is)) = skip

vincula_decs_fase2(ref(id)) = skip //… porque ya se vinculó en la fase 1

vincula_decs_fase2(int()) = skip
vincula_decs_fase2(real())= skip
vincula_decs_fase2(bool())= skip
vincula_decs_fase2(string())= skip

vincula_decs_fase2(array(T,n)) =
        vincula_decs_fase2(T)

vincula_decs_fase2(record(Cs)) =
        vincula_decs_fase2(Cs)

vincula_decs_fase2(campo(Tipo, id)) =
        vincula_decs_fase2(Tipo)

vincula_decs_fase2(campos_uno(C)) =
        vincula_decs_fase2(C)

vincula_decs_fase2(campos_muchos(Cs,C)) =
        vincula_decs_fase2(Cs)
        vincula_decs_fase2(C)
```

```
vincula_decs_fase2(pointer(T)) =
        si T = ref(id) entonces // Se vincula los “pointer – ref”
                si existe_id(ts, id) entonces
                        T.vinculo = valorDe(ts,id) //fija el vínculo del ref(id), cuelga de pointer
                 si no
                        error // id no declarado
        else
                vincula_decs_fase2(T)


// primera pasada parámetros formales

vincula_decs_fase1(pf_ninguno()) = skip //no hacer nada

vincula_decs_fase1(pf_uno(P)) =
        vincula_decs_fase1(P)

vincula_decs_fase1(pf_muchos(Ps,P)) =
        vincula_decs_fase1(Ps)
        vincula_decs_fase1(P)

vincula_decs_fase1(p_valor(T,id)) =
        vincula_decs_fase1(T)
        recolecta(id,$)

vincula_decs_fase1(p_var(T,id)) =
        vincula_decs_fase1(T)
        recolecta(id,$)

// segunda pasada parámetros formales

vincula_decs_fase2(pf_ninguno()) = skip //no hacer nada

vincula_decs_fase2(pf_uno(P)) =
        vincula_decs_fase2(P)

vincula_decs_fase2(pf_muchos(Ps,P)) =
        vincula_decs_fase2(Ps)
        vincula_decs_fase2(P)

vincula_decs_fase2(p_valor(T,id)) =
        vincula_decs_fase2(T)
        //no se recolecta

vincula_decs_fase1(p_var(T,id)) =
        vincula_decs_fase1(T)
        //no se recolecta
```

// vinculación en las instrucciones

```
vincula_is(is_muchas(Is,I)) =
        vincula_is(Is)
        vincula_is(I)

vincula_is(is_una(I)) =
        vincula_is(I)


vincula_is(is_asig(E0,E1)) =
        vincula_is(E0)
        vincula_is(E1)

vincula_is(is_if_then(E,Is)) =
        vincula_is(E)
        vincula_is(Is)

vincula_is(is_if_then_else(E,Is1,Is2)) =
        vincula_is(E)
        vincula_is(Is1)
        vincula_is(Is2)

vincula_is(is_while(E,Is)) =
        vincula_is(E)
        vincula_is(Is)

vincula_is(is_read(E)) =
        vincula_is(E)

vincula_is(is_write(E)) =
        vincula_is(E)

vincula_is(is_nl()) = skip

vincula_is(is_new(E)) =
        vincula_is(E)

vincula_is(is_delete(E)) =
        vincula_is(E)

vincula_is(is_proc(E,Ps)) =
        vincula_is(E)
        vincula_is(Ps)

vincula_is(is_seq(Decs,Is)) =
        abre_nivel(ts) // se abre el nivel del procedimiento en la tabla de símbolos
        vincula_decs_fase1(Decs)
```

```
        vincula_decs_fase2(Decs)
        vincula_is(Is)
        cierra_nivel(ts) // se elimina el nivel del procedimiento


vincula_is(exp_suma(E0,E1)) =
        vincula_is(E0)
        vincula_is(E1)

vincula_is(exp_resta(E0,E1)) =
        vincula_is(E0)
        vincula_is(E1)


vincula_is(exp_or(E0,E1)) =
        vincula_is(E0)
        vincula_is(E1)

vincula_is(exp_and(E0,E1)) =
        vincula_is(E0)
        vincula_is(E1)

vincula_is(exp_mul(E0,E1)) =
        vincula_is(E0)
        vincula_is(E1)

vincula_is(exp_div(E0,E1)) =
        vincula_is(E0)
        vincula_is(E1)

vincula_is(exp_mod(E0,E1)) =
        vincula_is(E0)
        vincula_is(E1)

vincula_is(exp_menos(E)) =
        vincula_is(E)

vincula_is(exp_not(E)) =
        vincula_is(E)

vincula_is(exp_index(E)) =
        vincula_is(E)

vincula_is(exp_acc_reg(c)) =
        vincula_is(c)//o skip

vincula_is(pr_ninguno()) = skip
```

```
vincula_is(pr_uno(E)) =
        vincula_is(E)


vincula_is(pr_muchos(Ps,E)) =
        vincula_is(Ps)
        vincula_is(E)

vincula_is(enteros(e)) = skip

vincula_is(reales(r)) = skip

vincula_is(booleanos(b)) = skip

vincula_is(cadena(c)) = skip

vincula_is(id(id)) =
        si existe_id(ts, id) entonces
                $.vinculo = valorDe(ts,id)
        si no
                error // id no declarado

vincula_is(dref(E)) =
        vincula_is(E)

vincula_is(acc(E,c)) =
        vincula_is(E)

vincula_is(indx(E0,E1)) =
        vincula_is(E0)
        vincula_is(E1)
```

2) Comprobación de tipos

```
chequeo_tipo(prog(Ds,Is)) =
        chequeo_tipo(Ds)
        chequeo_tipo(Is)
        $.tipo = ambos_ok(Ds.tipo,Is.tipo) //ambos_ok devuelve ok() si sus dos argumentos
                                           //son ok(), y error() en otro caso

chequeo_tipo(decs_ninguna()) = $.tipo = ok()

chequeo_tipo(decs_una(Dec)) =
        chequeo_tipo(Dec)
        $.tipo = Dec.tipo
```

```
chequeo_tipo(decs_muchas(Decs,Dec)) =
        chequeo_tipo(Decs)
        chequeo_tipo(Dec)
        $.tipo = ambos_ok(Decs.tipo, Dec.tipo)

chequeo_tipo(dec_var(T,id)) =
        chequeo_tipo(T) //se comprueba que los ref(…) refieran a declaraciones de tipos
        $.tipo = T.tipo

chequeo_tipo(dec_type(T,id)) =
        chequeo_tipo(T)
        $.tipo = T.tipo

chequeo_tipo(int()) =
        $.tipo = ok()

chequeo_tipo(real()) =
        $.tipo = ok()

chequeo_tipo(true()) =
        $.tipo = ok()

chequeo_tipo(false()) =
        $.tipo = ok()

chequeo_tipo(string()) =
        $.tipo = ok()

// Comprobación de que todos los refs refieren a declaraciones de tipo

chequeo_tipo(array(T,id)) =
        chequeo_tipo(T)
        si $.vinculo = dec_type(T,id)
                $.tipo = ok()
        si no
                error // referencia a una declaración que no es un tipo
                $.tipo = error()

chequeo_tipo(campo(id,T)) =
        chequeo_tipo(T)
        si $.vinculo = dec_type(T,id)
                $.tipo = ok()
        si no
                error // referencia a una declaración que no es un tipo
                $.tipo = error()

chequeo_tipo(record(c)) =
        chequeo_tipo(c)
```

```
        $.tipo = c.tipo

chequeo_tipo(ref(id)) =
        si $.vinculo = dec_type(T,id)
                $.tipo = ok()
        si no
                error // referencia a una declaración que no es un tipo
                $.tipo = error()

chequeo_tipo(pointer(T)) =
        chequeo_tipo(T)
        $.tipo = T.tipo

chequeo_tipo(dec_proc(id,Ps,Ds,Is)) =
        chequeo_tipo(Ps)
        chequeo_tipo(Ds)
        chequeo_tipo(Is)
        $.tipo = ambos_ok(Ps.tipo, ambos_ok(Ds.tipo,Is.tipo))

chequeo_tipo(pf_ninguna()) = $.tipo = ok()

chequeo_tipo(pf_una(P)) =
        chequeo_tipo(P)
        $.tipo = P.tipo

chequeo_tipo(pf_muchas(Ps,P)) =
        chequeo_tipo(Ps)
        chequeo_tipo(P)
        $.tipo = ambos_ok(Ps.tipo,P.tipo)

chequeo_tipo(p_valor(T,id)) =
        chequeo_tipo(T)
        $.tipo = T.tipo

chequeo_tipo(p_var(T,id)) =
        chequeo_tipo(T)
        $.tipo = T.tipo


chequeo_tipo(is_muchas(Is,I)) =
        chequeo_tipo(Is)
        chequeo_tipo(I)
        $.tipo = ambos_ok(Is.tipo,I.tipo)

chequeo_tipo(is_una(I)) =
        chequeo_tipo(I)
        $.tipo = I.tipo
```

```
chequeo_tipo (is_asig(E0,E1)) =
        chequeo_tipo(E0)
        chequeo_tipo(E1)
        si son_compatibles(E0.tipo,E1.tipo) && es_designador(E0) entonces
                $.tipo = ok()
        si no
                error //Informar de error de incompatibilidad de tipos, si procede (no informar)
                        // si alguno de los tipos involucrados es ya error
                $.tipo = error()

chequeo_tipo(is_if_then(E,Is)) =
        chequeo_tipo(E)
        si ref!(E) = bool() entonces
                chequeo_tipo(Is)
                $.tipo = ambos_ok(E.tipo,Is.tipo)
        si no
                error //Informar de error de incompatibilidad de tipos, si procede (no informar)
                        // si alguno de los tipos involucrados es ya error
                $.tipo = error()


chequeo_tipo(is_if_then_else(E,Is0,Is1)) =
        chequeo_tipo(E)
        si ref!(E) = bool() entonces
                chequeo_tipo(Is0)
                chequeo_tipo(Is1)
        $.tipo = ambos_ok(E.tipo,ambos_ok(Is.tipo,I.tipo))


        si no
                error //Informar de error de incompatibilidad de tipos, si procede (no informar)
                        // si alguno de los tipos involucrados es ya error
                $.tipo = error()


chequeo_tipo(is_while(E,Is)) =
        chequeo_tipo(E)
        si ref!(E) = bool() entonces
                chequeo_tipo(Is)
                $.tipo = ambos_ok(E.tipo,Is.tipo)
        si no
                error //Informar de error de incompatibilidad de tipos, si procede (no informar)
                        // si alguno de los tipos involucrados es ya error
                $.tipo = error()
```

```
chequeo_tipo(is_read(E)) =
        chequeo_tipo(E)
        si es_designador(E0) y (ref!(E0) = int() || ref!(E0) = real()
        || ref!(E0) = string()) entonces
                $.tipo = ok()
        si no
                error //Informar de error de incompatibilidad de tipos, si procede (no informar)
                        // si alguno de los tipos involucrados es ya error
                $.tipo = error()


chequeo_tipo(is_write(E)) =
        chequeo_tipo(E)
        si (ref!(E0) = int() || ref!(E0) = real() || ref!(E0) = string()) entonces
                $.tipo = ok()
        si no
                error //Informar de error de incompatibilidad de tipos, si procede (no informar)
                        // si alguno de los tipos involucrados es ya error
                $.tipo = error()



chequeo_tipo(is_nl()) = $.tipo = ok()

chequeo_tipo(is_new(E)) =
        chequeo_tipo(E)
        si (ref!(E0) = pointer(T)) entonces
                $.tipo = ok()
        si no
                error //Informar de error de incompatibilidad de tipos, si procede (no informar)
                        // si alguno de los tipos involucrados es ya error
                $.tipo = error()


chequeo_tipo(is_delete(E)) =
        chequeo_tipo(E)
        si (ref!(E0) = pointer(T)) entonces
                $.tipo = ok()
        si no
                error //Informar de error de incompatibilidad de tipos, si procede (no informar)
                        // si alguno de los tipos involucrados es ya error
                $.tipo = error()


chequeo_tipo(is_seq(Decs,Is)) =
        chequeo_tipo(Decs)
        chequeo_tipo(Is)
```

```
        $.tipo = ambos_ok(Descs.tipo,Is.tipo)

chequeo_tipo(is_proc(E,PRs)) =
        si E.vinculo = proc(id,PFs,Ds,Is) entonces
                si num_elems(PRs) !=  num_elems(PFs)
                        error // discrepancia en número de parámetros
                        $.tipo = error();
                else
                        $.tipo = chequeo_parametros(PRs,PFs)
                fin si
        si no
                error // id no es un procedimiento
                $.tipo = error()




chequeo_parametros(pr_ninguno(), pf_ninguno()) = return ok();

chequeo_parametros(pr_uno(PR), pf_uno(PF)) =
        return chequeo_parametro(PR,PF)

chequeo_parametros(pr_muchos(PRs,PR), pf_muchos(PFs,PF)) =
        return ambos_ok(chequeo_parametros(PRs,PFs),
                        chequeo_parametro(PR,PF))

chequeo_parametro(E,p_valor(T,id)) =
        chequeo_tipo(E)
        si son_compatibles(T,E.tipo)
                return ok()
        else
                error // señalar error, si procede (E.tipo no es ya error)
                return error()

chequeo_parametro(E,p_var(T,id)) =
        chequeo_tipo(E)
        si es_designador(E) && son_compatibles(T,E.tipo)
                return ok()
        else
                error // señalar error, si procede (E.tipo no es ya error)
                return error()

chequeo_tipo(id(id)) =
sea $.vinculo = Dec en
                si Dec = dec_var(T,id) || Dec = pf_valor(T,id) || Dec = pf_var(T,id) ||
                Dec = proc(id,PFs,Ds,Is) entonces
                $.tipo = T
        si no
```

```
                **error** // El identificador no es una variable ni un parámetro
                $.tipo = error()

chequeo_tipo(**enteros**(n)) =
        $.tipo = int()

chequeo_tipo(**reales**(n)) =
        $.tipo = real()

chequeo_tipo(**booleanos**(n)) =
        $.tipo = bool()

chequeo_tipo(**cadena**(n)) =
        $.tipo = string()

chequeo_tipo(**acc**(E,c)) =
        chequeo_tipo(E)
        **si ref!**(E.tipo) = **record**(Cs) **y existeCampo**(Cs,c) **entonces**
                $.tipo = **tipoDeCampo**(Cs,c)
        **si no**
                **error** // Informar del error que se ha producido (el tipo de E no es un registro,
                        // el campo no existe), siempre y cuando el tipo de E no sea
                        // ya error
                $.tipo = error()


chequeo_tipo(**indx**(E0,E1)) =
        chequeo_tipo(E0)
        chequeo_tipo(E1)
        **si ref!**(E0.tipo) = **array**(T,_) **y ref!**(E1.tipo) = int() **entonces**
                $.tipo = T
        **si no**
                **error** // Informar del error que procesa cuando, E0 no es erróneo, y no es un
                        //array, o E0 es un array, E1 no es erróneo, pero no es de tipo entero.
                $.tipo = error()


chequeo_tipo(**dref**(E)) =
        chequeo_tipo(E)
        **si ref!**(E.tipo) = **pointer**(T) **entonces**
                $.tipo = T
        **si no**
                **error** // Informar del error, a no ser que el tipo de E sea ya error
                $.tipo = error()


chequeo_tipo(**exp_suma**(E0,E1)) =
        chequeo_tipo(E0)
```

```
        chequeo_tipo(E1)
        si ref!(E0) = int() y ref!(E1) = int() entonces
                $.tipo = int()
        else si ref!(E0) = real() y ref!(E1) = real() || ref!(E0) = real() y ref!(E1) = int() ||
                ref!(E0) = int() y ref!(E1) = real() entonces
                $.tipo = real()
        else
                error // Informar del error de tipos si procede (omitir mensaje si alguno de los
                        //tipos involucrados es ya error)
                $.tipo = error()

chequeo_tipo(exp_resta(E0,E1)) =
        chequeo_tipo(E0)
        chequeo_tipo(E1)
        si ref!(E0) = int() y ref!(E1) = int() entonces
                $.tipo = int()
        else si ref!(E0) = real() y ref!(E1) = real() || ref!(E0) = real() y ref!(E1) = int() ||
                ref!(E0) = int() y ref!(E1) = real() entonces
                $.tipo = real()
        else
                error // Informar del error de tipos si procede (omitir mensaje si alguno de los
                        //tipos involucrados es ya error)
                $.tipo = error()


chequeo_tipo(exp_mul(E0,E1)) =
        chequeo_tipo(E0)
        chequeo_tipo(E1)
        si ref!(E0) = int() y ref!(E1) = int() entonces
                $.tipo = int()
        else si ref!(E0) = real() y ref!(E1) = real() || ref!(E0) = real() y ref!(E1) = int() ||
                ref!(E0) = int() y ref!(E1) = real() entonces
                $.tipo = real()
        else
                error // Informar del error de tipos si procede (omitir mensaje si alguno de los
                        //tipos involucrados es ya error)
                $.tipo = error()

chequeo_tipo(exp_div(E0,E1)) =
        chequeo_tipo(E0)
        chequeo_tipo(E1)
        si ref!(E0) = int() y ref!(E1) = int() entonces
                $.tipo = int()
        else si ref!(E0) = real() y ref!(E1) = real() || ref!(E0) = real() y ref!(E1) = int() ||
                ref!(E0) = int() y ref!(E1) = real() entonces
                $.tipo = real()
        else
                error // Informar del error de tipos si procede (omitir mensaje si alguno de los
```

```
                    //tipos involucrados es ya error)
                $.tipo = error()

chequeo_tipo(exp_mod(E0,E1)) =
        chequeo_tipo(E0)
        chequeo_tipo(E1)
        si ref!(E0) = int() y ref!(E1) = int() entonces
                $.tipo = int()
        else
                error // Informar del error de tipos si procede (omitir mensaje si alguno de los
                    //tipos involucrados es ya error)
                $.tipo = error()
chequeo_tipo(exp_menos(E)) =
        chequeo_tipo(E)
        si ref!(E) = int() entonces
                $.tipo = int()
        else si ref!(E) = real() entonces
                $.tipo = real()
        else
                error // Informar del error de tipos si procede (omitir mensaje si alguno de los
                    //tipos involucrados es ya error)
                $.tipo = error()


chequeo_tipo(exp_not(E)) =
        chequeo_tipo(E)
        si ref!(E) = bool() entonces
                $.tipo = bool()
        else
                error // Informar del error de tipos si procede (omitir mensaje si alguno de los
                    //tipos involucrados es ya error)
                $.tipo = error()


chequeo_tipo(exp_or(E0,E1)) =
        chequeo_tipo(E0)
        chequeo_tipo(E1)
        si ref!(E0) = bool() y ref!(E1) = bool() entonces
                $.tipo = bool()
        else
                error // Informar del error de tipos si procede (omitir mensaje si alguno de los
                    //tipos involucrados es ya error)
                $.tipo = error()


chequeo_tipo(exp_and(E0,E1)) =
        chequeo_tipo(E0)
        chequeo_tipo(E1)
```

```
        si ref!(E0) = bool() y ref!(E1) = bool() entonces
                $.tipo = bool()
        else
                error // Informar del error de tipos si procede (omitir mensaje si alguno de los
                        //tipos involucrados es ya error)
                $.tipo = error()

chequeo_tipo(op_menor(E0,E1)) =
        chequeo_tipo(E0)
        chequeo_tipo(E1)
        si ( (ref!(E0) = int() || ref!(E0) = real()) y (ref!(E1) = real() || ref!(E1) = int() ) ||
        (ref!(E0) = bool() y ref!(E1) = bool() ) || (ref!(E0) = string() y ref!(E1) = string() ))
        entonces
                $.tipo = booll()
        else
                error // Informar del error de tipos si procede (omitir mensaje si alguno de los
                        //tipos involucrados es ya error)
                $.tipo = error()




chequeo_tipo(op_mayor(E0,E1)) =
        chequeo_tipo(E0)
        chequeo_tipo(E1)
        si ( (ref!(E0) = int() || ref!(E0) = real()) y (ref!(E1) = real() || ref!(E1) = int() ) ||
        (ref!(E0) = bool() y ref!(E1) = bool() ) || (ref!(E0) = string() y ref!(E1) = string() ))
        entonces
                $.tipo = booll()
        else
                error // Informar del error de tipos si procede (omitir mensaje si alguno de los
                        //tipos involucrados es ya error)
                $.tipo = error()

chequeo_tipo(op_menor_igual(E0,E1)) =
        chequeo_tipo(E0)
        chequeo_tipo(E1)
        si ( (ref!(E0) = int() || ref!(E0) = real()) y (ref!(E1) = real() || ref!(E1) = int() ) ||
        (ref!(E0) = bool() y ref!(E1) = bool() ) || (ref!(E0) = string() y ref!(E1) = string() ))
        entonces
                $.tipo = booll()
        else
                error // Informar del error de tipos si procede (omitir mensaje si alguno de los
                        //tipos involucrados es ya error)
                $.tipo = error()

chequeo_tipo(op_mayor_igual(E0,E1)) =
        chequeo_tipo(E0)
```

```
        chequeo_tipo(E1)
        si ( (ref!(E0) = int() || ref!(E0) = real()) y (ref!(E1) = real() || ref!(E1) = int() ) ||
        (ref!(E0) = bool() y ref!(E1) = bool() ) || (ref!(E0) = string() y ref!(E1) = string() ))
        entonces
                $.tipo = booll()
        else
                error // Informar del error de tipos si procede (omitir mensaje si alguno de los
                        //tipos involucrados es ya error)
                $.tipo = error()

chequeo_tipo(op_igual(E0,E1)) =
        chequeo_tipo(E0)
        chequeo_tipo(E1)
        si ( (ref!(E0) = int() || ref!(E0) = real()) y (ref!(E1) = real() || ref!(E1) = int() ) ||
        (ref!(E0) = bool() y ref!(E1) = bool() ) || (ref!(E0) = string() y ref!(E1) = string() ) ||
        (ref!(E0) =pointer(T) || ref!(E0) = null()) y (ref!(E1) =pointer(T) || ref!(E1) =null() ) )
        entonces
                $.tipo = booll()
        else
                error // Informar del error de tipos si procede (omitir mensaje si alguno de los
                        //tipos involucrados es ya error)
                $.tipo = error()
chequeo_tipo(op_distinto(E0,E1)) =
        chequeo_tipo(E0)
        chequeo_tipo(E1)
        si ( (ref!(E0) = int() || ref!(E0) = real()) y (ref!(E1) = real() || ref!(E1) = int() ) ||
        (ref!(E0) = bool() y ref!(E1) = bool() ) || (ref!(E0) = string() y ref!(E1) = string() ) ||
        (ref!(E0) =pointer(T) || ref!(E0) = null()) y (ref!(E1) =pointer(T) || ref!(E1) =null() ) )
        entonces
                $.tipo = booll()
        else
                error // Informar del error de tipos si procede (omitir mensaje si alguno de los
                        //tipos involucrados es ya error)
                $.tipo = error()




ambos_ok(to,t1) =
        si to=ok() && t1=ok() return ok()
        else return error()

son_compatibles(t0,t1) =
        si to == t1 return ok()
        else return error()
```

3) Asignación de espacio

global dir=0 // contador de direcciones
global nivel=0 // nivel de anidamiento

asigna_espacio(**prog**(Decs,Is)) =
    asigna_espacio(Decs)

asigna_espacio(**decs_ninguna**()) = **skip**

asigna_espacio(**decs_una**(Dec)) = asigna_espacio(Dec)

asigna_espacio(**decs_muchas**(Decs,Dec)) =
    asigna_espacio(Decs)
    asigna_espacio(Dec)

asigna_espacio(**dec_variable**(Tipo, id)) =
    $.dir = dir
    $.nivel = nivel
    asigna_espacio_tipo(Tipo)
    dir = dir + Tipo.tamaño

asigna_espacio(**dec_tipo**(Tipo, id)) =
    asigna_espacio_tipo(Tipo)

asigna_espacio(**dec_proc**(id,Preals,Decs,Is)) =
    ant_dir = dir
    nivel = nivel + 1
    $.nivel = nivel
    dir = 0
    asigna_espacio(Preals)
    asigna_espacio(Decs)
    $.tam_datos = dir
    dir = ant_dir
    nivel = nivel - 1

asigna_espacio(**pr_ninguno**()) = **skip**
asigna_espacio(**pr_uno**(Preal)) = asigna_espacio(Preal)
asigna_espacio(**pr_muchos**(Preals, Preal)) =
    asigna_espacio(Preals)
    asigna_espacio(Preal)

asigna_espacio(**pf_ninguno**()) = skip
asigna_espacio(**pf_uno**(Pform)) = asigna_espacio(Pform)
asigna_espacio(**pf_muchos**(Pforms, Pform)) =
    asigna_espacio(Pforms)
    asigna_espacio(Pform)

asigna_espacio(**pf_valor**(Tipo, id)) =
    $.dir = dir
    $.nivel = nivel
    asigna_espacio_tipo(Tipo)
    dir = dir + T.tam

asigna_espacio(**pf_var**(Tipo, id)) =
    $.dir = dir
    $.nivel = nivel
    asigna_espacio_tipo(Tipo)
    dir = dir + 1

asigna_espacio_tipo1(**campos_uno**(Campo)) =
    asigna_espacio(Campo)
asigna_espacio(**campos_muchos**(Campos, Campo) =
    asigna_espacio(Campos)
    asigna_espacio(Campo)

asigna_espacio(**campo(**Tipo, id**)**) =
    $.dir = dir
    $.nivel = nivel
    asigna_espacio_tipo(Tipo)
    dir = dir + Tipo.tamaño

asigna_espacio(**array_uno**(Elem)) =
    asigna_espacio(Elem)

asigna_espacio(**array_muchos**(Elems, Elem)) =
    asigna_espacio(Elems)
    asigna_espacio(Elem)

asigna_espacio(**elem(**Tipo**)**) =
    $.dir = dir
    $.nivel = nivel
    asigna_espacio_tipo(Tipo)
    dir = dir + Tipo.tamaño

asigna_espacio_tipo(Tipo) =
    si indefinido(Tipo.tam) **entonces**
        asigna_espacio_tipo1(Tipo)
        / Primera pasada en la asignación de espacio en una definición de tipo. Se determinan todos los tamaños, a excepción de los de 'ref' que son hijos de 'pointer'.*/*
        asigna_espacio_tipo2(Tipo)
        / Segunda pasada en la asignación de espacio en una definición de tipo. Se calculan los tamaños de los nodos 'ref' que son hijos de 'pointer', y, si procede, los de los tipos referidos. */*

```
asigna_espacio_tipo1(int()) =
        $.tam = 1

asigna_espacio_tipo1(real()) =
        $.tam = 1

asigna_espacio_tipo1(bool()) =
        $.tam = 1

asigna_espacio_tipo1(string()) =
        $.tam = 1

asigna_espacio_tipo1(ref(id)) =
        asigna_espacio_tipo1($.vinculo)
        sea $.vinculo = dec_tipo(Tipo,_) en
                $.tam = T.tam

asigna_espacio_tipo1(pointer(T)) =
        $.tam = 1
        si Tipo ≠ ref(_)
                asigna_espacio_tipo1(T)
                /* Si el tipo base es una referencia a una declaración de tipo, no podemos estar
                seguros de que no sea a la declaración del tipo que estamos procesando
                actualmente. Por tanto, si tratamos de determinar aquí el tamaño, podemos entrar en
                bucle infinito: dejamos su determinación a una segunda pasada, en la que sabremos
                ya el tamaño de T*/

asigna_espacio_tipo2(int()) = skip
asigna_espacio_tipo2(real()) = skip
asigna_espacio_tipo2(bool()) = skip
asigna_espacio_tipo2(string()) = skip
asigna_espacio_tipo2(ref(id)) = skip

asigna_espacio_tipo2(pointer(Tipo)) =
        si Tipo = ref(id)
                sea Tipo.vinculo = dec_tipo(Tipo',_)
                        asigna_espacio_tipo(Tipo')
                        /* Ahora sí podemos determinar el tamaño del tipo base, ya que, (i) bien ha
                        sido ya procesado, (ii) bien es el que se está procesando ahora, (iii) bien
                        aparece más adelante, y aún no ha sido procesado. En los dos primeros
                        casos, se conocerá ya su tamaño. En el último caso, se pasará a determinar
                        el mismo.*/
                $.tam= T'.tam
```

4) Instrucciones máquina-p

Traducción de los designadores

- **Variable Global (x)**
  apila(dirección de x)
- **Variable local o parámetro por valor (y)**
  apilad(nivel de y)
  apila(dirección de y)
  suma
- **Parámetro por variable(z)**
  apilad(nivel de z)
  apila(dirección de z)
  suma
  apilaind
- **(* D)**
  <<<Código para D>>>
  apilaind
- **D.c**
  apila(desplazamiento de c)
  suma
- **D[E]**
  <<<Código para D>>>
  <<<Código para E>>>>
  (apilaind)? (sólo si E es un designador)
  apila(tamaño tipo base de D)
  mul
  suma

Traducción de las expresiones

- **Literal l**
  apila<tipo de l>(l)
- **E0 op E1**
  <<<<Código para E0 >>>
  (apilaind)? (sólo si E0 es un designador)
  <<<<Código para E1 >>>
  (apilaind)? (sólo si E1 es un designador)
  <<<<Instrucción que realiza la operación **op**>>>>
- **op E**
  <<<<Código para E >>>
  (apilaind)? (sólo si E es un designador)
  <<<<Instrucción que realiza la operación **op**>>>>

Traducción de la asignación

- **D = E**
  <<<Código para D>>>
  <<<Código para E>>>
  (mueve(tamaño de D) si E es un designador; desapilaind si E no es un designador)

Traducción estructuras de control:

- **while E do I**
  <<<Código para E>>>
  irf(dirección siguiente instrucción al while)
  <<<Código para I>>>>
  ira(dirección primera instrucción del while)
- **if E then I**
  <<<Código para E>>>
  irf(dirección siguiente instrucción al if)
  <<<Código para I>>>>
- **if E then I0 else I1**
  <<<Código para E>>>
  irf(dirección primera instrucción para I1)
  <<<Código para I0>>>>
  ira(dirección siguiente instrucción tras I1)
  <<<Código para I1>>>

Traducción llamadas a procedimientos:

- **p(E0, …, En)**
  activa(nivel de p, tamaño datos de p, siguiente dirección a la llamada)
  dup
  apila(dirección parámetro formal 0)
  suma
  <<<Código para E0>>>
  <<<Instrucción de paso: mueve, o desapilaind, dependiendo del parámetro formal y del parámetro real>>>
  ….
  dup
  apila(dirección parámetro formal n)
  suma
  <<<Código para En>>>
  <<<Instrucción de paso: mueve, o desapilaind >>>
  desapilad(nivel de p)
  ira(dirección de comienzo de p)

Instrucción de paso:

- **Parámetro real = designador && parámetro formal por valor**
  mueve(tamaño parámetro formal)
- **Parámetro real = designador && parámetro formal por variable**
  desapilaind
- **Parámetro real ≠ designador && parámetro formal por valor**
  desapilaind

5) Etiquetado

```
global etq=0
global procs = pila_vacia()
etiqueta(prog(Decs,Is)) =
        etiqueta(Is)
        recolecta_procs(Decs)
        mientras(! es_vacia(procs)))
        P = pop(procs)
        etiqueta(P)

etiqueta(is_ninguna()) = skip

etiqueta(is_una(I)) = etiqueta(I)

etiqueta(is_muchas(Is,I)) =
        gen_cod(Is)
        gen_cod(I)

etiqueta(asig(E0,E1)) =
        etiqueta(E0)
        etiqueta(E1)
        etq = etq + 1

etiqueta(call(id,Preals)) =
        etq = etq + 1
        sea $.vinculo = proc(id,Pforms,Decs,Is) en
                etiqueta_params(Preals,Pforms)
        etq = etq + 2
        $.dir_sig = etq

etiqueta_params(pr_ninguno(),pf_ninguno()) = skip

etiqueta_params(pr_uno(Preal), pf_uno(Pform)) = etiqueta_paso(Preal,Pform)

etiqueta(pr_muchos(Preals,Preal), , pf_muchos(Pforms,Pform)) =
        etiqueta_params(Preals,Pforms)
        etiqueta_paso(Preal,Pform)

etiqueta_paso(E,Pform) =
        etq = etq + 3
        gen_cod(E)
        etq = etq + 1

etiqueta(num(n)) =
        etq = etq + 1
```

```
gen_cod(id(v)) =
        si $.vinculo.nivel = 0
                etq = etq + 1
        si no
                etq = etq + 3
                si $.vinculo = pf_var(Tipo,v)
                        etq = etq + 1
etiqueta(prim(E)) = etiqueta(E)

etiqueta(seg(E)) =
        etiqueta(E)
        etq = etq + 2

etiqueta(dref(E)) =
        etiqueta(E)
        etq = etq + 1

etiqueta(proc(id,Preals,Decs,Is)) =
        $.dir_inic = etq
        etiqueta(Is)
        etq = etq + 2
        recolecta_procs(Ds)

recolecta_procs(decs_ninguna()) = skip
recolecta_procs(decs_una(Dec)) = recolecta_procs(Dec)
recolecta_procs(decs_muchas(Decs,Dec)) =
        recolecta_procs(Decs)
        recolecta_procs(Dec)

recolecta_procs(dec_var(Tipo,id)) = skip
recolecta_procs(dec_tipo(Tipo,id)) = skip
recolecta_procs(dec_proc(id,Preals,Decs,Is)) =
        push($,procs)
        //se añade el procedimiento a la pila de procedimientos pendientes de traducción
```

6) Generación de código

global procs = **pila_vacia()** // procedimientos pendientes de traducir

```
gen_cod(prog(Ds,Is)) =
        gen_cod(Is)
        recolecta_procs(Ds)
        // Este bucle provoca que se vayan traduciendo sucesivamente los
        // procedimientos
        mientras(! es_vacia(procs)))
                P = pop(procs)
                gen_cod(P)


gen_cod(is_ninguna()) = skip

gen_cod(is_una(I)) = gen_cod(I)

gen_cod(is_muchas(Is,I)) =
        gen_cod(Is)
        gen_cod(I)

gen_cod(is_asig(E0,E1)) =
        gen_cod(E0)
        gen_cod(E1)
        gen_ins_asig(E1) // mueve o desapilaind, dependiendo de si
                         // E1 es o no un designador

gen_cod(is_if_then(E,Is)) =
        gen_cod(E)
        si ( E)
                gen_cod(Is)

gen_cod(is_if_then_else(E,Is1,Is2)) =
        gen_cod(E)
        si ( E)
                gen_cod(Is1)
        si no
                gen_cod(Is2)

gen_cod(is_while(E,Is)) =
        gen_cod(E)
        mientras ( E)
                gen_cod(Is)
```

```
gen_cod(is_read(E)) =
        gen_cod(E)
        gen_ins(apilaint())

gen_cod(is_write(E)) =
        gen_cod(E)

gen_cod(is_nl(E)) = skip

gen_cod(is_new(E)) =
        gen_cod(E)
        gen_ins(apilaint())

gen_cod(is_delete(E)) =
        gen_cod(E)
        gen_ins(desapilaint())

gen_cod(is_proc(E,Ps)) =
        gen_cod(E)
        gen_ins(activa($.vinculo.nivel,$.vinculo.tam_datos,$.dir_sig))
        sea $.vinculo = proc(id,PsF,Ds,Is) en
                gen_cod_params(Ps,PsF)
        fin sea
        gen_ins(desapilad($.vinculo.nivel))
        gen_ins(ir_a($.vinculo.dir_inic))

gen_cod(is_seq(Ds,Is)) =
        gen_cod(Is)
        recolecta_procs(Ds)
        // Este bucle provoca que se vayan traduciendo sucesivamente los
        // procedimientos
        mientras(! es_vacia(procs)))
                P = pop(procs)
                gen_cod(P)

gen_cod_params(pr_ninguno(),pf_ninguno()) = skip

gen_cod_params(pr_uno(P), pf_uno(PF)) = gen_cod_paso(P,PF)

gen_cod_params(pr_muchos(Ps,P), pf_muchos(PsF,PF)) =
        gen_cod_params(Ps,PsF)
        gen_cod_paso(P,PF)

gen_cod_paso(E,PF) =
```

```
        gen_ins(dup())
        gen_ins(apilaint(PF.dir))
        gen_ins(suma())
        gen_cod(E)
        gen_ins_paso(E,PF) //mueve si E no es designador, y PF es por
                           // valor, desapilaind en otro caso

gen_cod(enteros(n)) =
        gen_ins(apilaint(n))

gen_cod(reales(n)) =
        gen_ins(apilaint(n))

gen_cod(booleanos(n)) =
        gen_ins(apilaint(n))

gen_cod(cadenas(n)) =
        gen_ins(apilaint(n))

gen_cod(id(v)) =
        si $.vinculo.nivel = 0
                gen_ins(apilaint($.vinculo.dir))
        si no
                gen_ins(apilad($.vinculo.nivel))
                gen_ins(apilaint($.vinculo.dir))
                gen_ins(suma())
                si $.vinculo = pf_var(T,v)
                        gen_ins(apilaind())

gen_cod(dref(E)) =
        gen_cod(E)
        gen_ins(apilaint())

gen_cod(acc(E,v)) =
        gen_cod(E)
        gen_ins(apilaint())

gen_cod(indx(E0,E1)) =
        gen_cod(E0)
        gen_ins(apilaint())
        gen_cod(E1)
        gen_ins(apilaint())


gen_cod(proc(id,PFs,Ds,Is)) =
        gen_cod(Is)
        gen_ins(desactiva($.nivel,$.tam))
        gen_ins(irind())
```

recolecta_procs(Ds)

recolecta_procs(**decs_ninguna**()) = **skip**

recolecta_procs(**decs_una**(Dec)) = recolecta_procs(Dec)

recolecta_procs(**decs_muchas**(Decs,Dec)) =
recolecta_procs(Decs)
recolecta_procs(Dec)

recolecta_procs(**dec_var**(T,id)) = **skip**

recolecta_procs(**dec_tipo**(T,id)) = **skip**

recolecta_procs(**dec_proc**(id,Ps,Ds,Is)) =
push($,procs) // se añade el procedimiento a la pila de procedimientos pendientes de traducción