Universitat de les Illes Balears

Processadors del Llenguatge

Pràctica de Compiladors. El compilador "Compilemon".

Autors:
José Ruiz Bravo
<joseruizbravo@gmail.com>
Biel Moyà Alcover
<bilibiel@gmail.com>
Álvaro Medina Ballester
<alvaro@comiendolimones.com>

Professor:
Dr. Albert Llemosí Cases

6 de juliol de 2010

Resum

Compilador *compilemon* creat amb el llenguatge Ada. Està composat per un subconjunt bàsic d'instruccions en Ada conegudes com *lemonada*.

1 Anàlisi Lèxica

1.1 Descripció del lèxic: *pk_ulexica.l*

```
1  -- Macros
2  lletra          [A-Za-z]
3  digit           [0-9]
4  separadors      [" "\n\b\t\f]
5  car             ({lletra}|{digit})
6  caracter        \'{car}\'
7  car_string      [\040-\041\043-\176]
8  string          \"({car_string}|\"\")*\"
9
10
11 %%
12
13
14 -- Paraules clau
15 procedure       {mt_atom(tok_begin_line, tok_begin_col,
16                 yylval); return pc_procedure;}
17
18 begin           {mt_atom(tok_begin_line, tok_begin_col,
19                 yylval); return pc_begin;}
20
21 while           {mt_atom(tok_begin_line, tok_begin_col,
22                 yylval); return pc_while;}
23
24 if              {mt_atom(tok_begin_line, tok_begin_col,
25                 yylval); return pc_if;}
26
27 else            {mt_atom(tok_begin_line, tok_begin_col,
28                 yylval); return pc_else;}
29
30 end             {mt_atom(tok_begin_line, tok_begin_col,
31                 yylval); return pc_end;}
32
33 loop            {mt_atom(tok_begin_line, tok_begin_col,
```

```
34                  yylval); return pc_loop;}
35
36 constant          {mt_atom(tok_begin_line, tok_begin_col,
37                  yylval); return pc_constant;}
38
39 type              {mt_atom(tok_begin_line, tok_begin_col,
40                  yylval); return pc_type;}
41
42 array             {mt_atom(tok_begin_line, tok_begin_col,
43                  yylval); return pc_array;}
44
45 record            {mt_atom(tok_begin_line, tok_begin_col,
46                  yylval); return pc_record;}
47
48 is                {mt_atom(tok_begin_line, tok_begin_col,
49                  yylval); return pc_is;}
50
51 then              {mt_atom(tok_begin_line, tok_begin_col,
52                  yylval); return pc_then;}
53
54 not               {mt_atom(tok_begin_line, tok_begin_col,
55                  yylval); return pc_not;}
56
57 in                {mt_atom(tok_begin_line, tok_begin_col,
58                  yylval); return pc_in;}
59
60 out               {mt_atom(tok_begin_line, tok_begin_col,
61                  yylval); return pc_out;}
62
63 new               {mt_atom(tok_begin_line, tok_begin_col,
64                  yylval); return pc_new;}
65
66 null              {mt_atom(tok_begin_line, tok_begin_col,
67                  yylval); return pc_null;}
68
69 of                {mt_atom(tok_begin_line, tok_begin_col,
70                  yylval); return pc_of;}
71
72 mod               {mt_atom(tok_begin_line, tok_begin_col,
73                  yylval); return pc_mod;}
74
75 range             {mt_atom(tok_begin_line, tok_begin_col,
76                  yylval); return pc_range;}
```

```
77
78 and               {mt_atom(tok_begin_line, tok_begin_col,
79                   yylval); return pc_and;}
80
81 or                {mt_atom(tok_begin_line, tok_begin_col,
82                   yylval); return pc_or;}
83
84 --Simbols
85 ":="              {mt_atom(tok_begin_line, tok_begin_col,
86                   yylval); return s_assignacio;}
87
88 ":"               {mt_atom(tok_begin_line, tok_begin_col,
89                   yylval); return s_dospunts;}
90
91 ";"               {mt_atom(tok_begin_line, tok_begin_col,
92                   yylval); return s_final;}
93
94 ","               {mt_atom(tok_begin_line, tok_begin_col,
95                   yylval); return s_coma;}
96
97 "("               {mt_atom(tok_begin_line, tok_begin_col,
98                   yylval); return s_parentesiobert;}
99
100 ")"               {mt_atom(tok_begin_line, tok_begin_col,
101                   yylval); return s_parentesitancat;}
102
103 ".."              {mt_atom(tok_begin_line, tok_begin_col,
104                   yylval); return s_puntsrang;}
105
106 "."               {mt_atom(tok_begin_line, tok_begin_col,
107                   yylval); return s_puntrec;}
108
109 --Operadors
110 "<"               {mt_atom(tok_begin_line, tok_begin_col,
111                   yylval); return op_menor;}
112
113 "<="              {mt_atom(tok_begin_line, tok_begin_col,
114                   yylval); return op_menorigual;}
115
116 ">="              {mt_atom(tok_begin_line, tok_begin_col,
117                   yylval); return op_majorigual;}
118
119 ">"               {mt_atom(tok_begin_line, tok_begin_col,
```

```
120                 yylval); return op_major;}
121
122 "="               {mt_atom(tok_begin_line, tok_begin_col,
123                 yylval); return op_igual;}
124
125 "/="              {mt_atom(tok_begin_line, tok_begin_col,
126                 yylval); return op_distint;}
127
128 "+"               {mt_atom(tok_begin_line, tok_begin_col,
129                 yylval); return op_suma;}
130
131 "-"               {mt_atom(tok_begin_line, tok_begin_col,
132                 yylval); return op_resta;}
133
134 "*"               {mt_atom(tok_begin_line, tok_begin_col,
135                 yylval); return op_multiplicacio;}
136
137 "/"               {mt_atom(tok_begin_line, tok_begin_col,
138                 yylval); return op_divisio;}
139
140 --Digit
141 {digit}+          {mt_numero(tok_begin_line, tok_begin_col,
142                 yytext, yylval); return const;}
143
144 --Lletra
145 {caracter}        {mt_caracter(tok_begin_line, tok_begin_col,
146                 yytext, yylval); return const;}
147
148 --String  mirar a les declaracions
149 {string}          {mt_string(tok_begin_line, tok_begin_col,
150                 yytext, yylval); return const;}
151
152 --Identificador
153 {lletra}({car}|("_"{car}))*     {mt_identificador(tok_begin_line, tok_beg
154
155 --Comentaris
156 "--"[^\n]*              {null;}
157
158 --Separadors
159 [\t|\n|\ ]        {null;}
160
161 --Error
162 .         {return error;}
```

```
163
164
165 %%
166
167
168 with     decls.d_taula_de_noms,
169          pk_usintactica_tokens,
170          decls.d_atribut,
171          semantica.ctipus,
172          decls.dtdesc;
173
174 use      decls.d_taula_de_noms,
175          pk_usintactica_tokens,
176          decls.d_atribut,
177          semantica.ctipus,
178          decls.dtdesc;
179
180 package u_lexica is
181          function YYLex return token;
182 end u_lexica;
183
184
185 package body u_lexica is
186 ##
187 end u_lexica;
```

1.2 Taula de noms

1.2.1 Fitxer *decls-d_ taula_ de_ noms.ads*

```
1  -- DECLS-D_TAULA_DE_NOMS.ads
2  -- Declaracions de la taula de noms
3
4  WITH     Decls.Dgenerals,
5           Ada.Text_Io;
6
7  USE      Decls.Dgenerals,
8           Ada.Text_Io;
9
10
11 PACKAGE Decls.D_Taula_De_Noms IS
12
13     --pragma pure;
14
15     --Excepcions
16     E_Tids_Plena : EXCEPTION;
17     E_Tcar_Plena : EXCEPTION;
18
19     TYPE Taula_De_Noms IS LIMITED PRIVATE;
20
21     TYPE Id_Nom IS NEW Integer
22       RANGE 0 .. Max_Id;
23
24     TYPE Rang_Dispersio IS NEW Integer
25       RANGE 0 .. Max_Id;
26
27     TYPE Rang_Tcar IS NEW Integer
28       RANGE 0 .. (Long_Num_Ident*Max_Id);
29
30     -- Valor nul per al tipus id_nom
31     Id_Nul : CONSTANT Id_Nom := 0;
32
33     PROCEDURE Tbuida
34       (Tn : OUT Taula_De_Noms);
35
36     PROCEDURE Posa_Id
37       (Tn : IN OUT Taula_De_Noms;
38       Idn : OUT Id_Nom;
39       Nom : IN String);
40
```

```
41      PROCEDURE Posa_Tc
42        (Tn : IN OUT Taula_De_Noms;
43        Nom : IN String);
44
45      PROCEDURE Posa_Str
46        (Tn : IN OUT Taula_De_Noms;
47        Ids : OUT Rang_Tcar;
48          S : IN String);
49
50      FUNCTION Cons_Nom
51        (Tn : IN Taula_De_Noms;
52        Idn : IN Id_Nom)
53        RETURN String;
54
55      FUNCTION Cons_Str
56        (Tn : IN Taula_De_Noms;
57        Ids : IN Rang_Tcar)
58        RETURN String;
59
60      FUNCTION Fdisp_Tn
61        (Nom : IN String)
62        RETURN Rang_Dispersio;
63
64
65  PRIVATE
66
67      TYPE Taula_Dispersio IS ARRAY
68        (Rang_Dispersio) OF Id_Nom;
69
70      TYPE T_Identificador IS RECORD
71              Pos_Tcar : Rang_Tcar;
72               Seguent : Id_Nom;
73           Long_Paraula : Natural;
74      END RECORD;
75
76      TYPE Taula_Identificadors IS ARRAY
77        (1 .. Id_Nom'Last) OF T_Identificador;
78
79      TYPE Taula_Caracters IS ARRAY
80        (Rang_Tcar) OF Character;
81
82      TYPE taula_de_noms IS RECORD
83           Td : Taula_Dispersio;
```

```
84          Tid : Taula_Identificadors;
85           Tc : Taula_Caracters;
86          Nid : Id_Nom;
87         Ncar : Rang_Tcar;
88      END RECORD;
89
90  END Decls.D_Taula_De_Noms;
```

1.2.2 Fitxer *decls-d_taula_de_noms.adb*

```
1  -- DECLS-D_TAULA_DE_NOMS.adb
2  -- Procediments per a la taula de noms
3
4  PACKAGE BODY Decls.D_Taula_De_Noms IS
5
6     -- Donam els valors per defecte de cada camp.
7     PROCEDURE Tbuida
8       (Tn : OUT Taula_De_Noms) IS
9     BEGIN
10       FOR I IN Tn.Td'RANGE LOOP
11          Tn.Td(I) := Id_Nul;
12       END LOOP;
13
14       Tn.Nid := 1;
15       Tn.Ncar := 1;
16       Tn.Tid(1).Seguent := Id_Nul;
17    END Tbuida;
18
19
20    PROCEDURE Posa_Id
21      (Tn : IN OUT Taula_De_Noms;
22       idn : OUT Id_Nom;
23       nom : IN String) IS
24
25       -- Variable per el valor de la funcio de dispersio.
26       P_Tid : Rang_Dispersio;
27
28       -- Index per recorrer la taula d'identificadors.
29       Idx : Id_Nom;
30       Trobat : Boolean;
31
32       P : Taula_Identificadors RENAMES Tn.Tid;
33       D : Taula_Dispersio RENAMES Tn.Td;
34
35    BEGIN
36       P_Tid := Fdisp_Tn(Nom);
37       Idx := D(P_Tid);
38       Trobat := False;
39
40       WHILE NOT Trobat AND Idx/=Id_Nul LOOP
41          IF (Nom = Cons_Nom(Tn, Idx)) THEN
```

```
42              Idn := idx;
43              Trobat := True;
44           ELSE
45              Idx := p(Idx).Seguent;
46           END IF;
47        END LOOP;
48
49        IF NOT Trobat THEN
50           Idn := Tn.Nid;
51           P(Idn).Pos_Tcar := Tn.Ncar;
52           P(Idn).Seguent := D(P_Tid);
53           P(Idn).Long_Paraula := Nom'Length;
54           D(P_Tid) := Tn.Nid;
55           Posa_Tc(Tn, Nom);
56        END IF;
57     END Posa_Id;
58
59
60     PROCEDURE Posa_Tc
61       (Tn : IN OUT Taula_De_Noms;
62        Nom : IN String) IS
63     BEGIN
64        Tn.Nid := Tn.Nid + 1;
65        FOR I IN 1 .. Nom'Length LOOP
66           Tn.Tc(Tn.Ncar) := Nom(I);
67           Tn.Ncar := Tn.Ncar + 1;
68        END LOOP;
69     END Posa_Tc;
70
71
72     PROCEDURE Posa_Str
73       (Tn : IN OUT Taula_De_Noms;
74        Ids : OUT Rang_Tcar;
75        S : IN String) IS
76
77        -- Index per recorrer la taula de caracters.
78        Jdx : Rang_Tcar;
79        Long : Rang_Tcar RENAMES Tn.Ncar;
80
81     BEGIN
82        -- Excepcio per a controlar tc plena
83        IF (Long + S'Length) > Rang_Tcar'Last THEN
84           RAISE E_Tcar_Plena;
```

```
85        END IF;
86
87        -- Omplim la taula de caracters, desde la primera
88        -- posicio lliure 'tn.ncar' renombrat a 'long'.
89        Jdx := Long;
90        Ids := Long;
91
92        FOR I IN 1..S'Length LOOP
93           Tn.Tc(Jdx) := S(I);
94           Jdx := Jdx + 1;
95        END LOOP;
96        Long := Jdx + 1;
97        Tn.Tc(Jdx) := Ascii.Nul;
98
99     END Posa_Str;
100
101
102    FUNCTION Cons_Nom
103      (Tn : IN Taula_De_Noms;
104       Idn : IN Id_Nom)
105      RETURN String IS
106
107       It1, It2 : Rang_Tcar;
108
109    BEGIN
110       IF Idn /= Id_Nul THEN
111          It1 := Tn.Tid(Idn).Pos_Tcar;
112          It2 := Rang_Tcar(Tn.Tid(Idn).Long_Paraula);
113          It2 := It2 + It1 - 1;
114          RETURN String(Tn.Tc(It1 .. It2));
115       ELSE RETURN "Id_Nul";
116       END IF;
117    END Cons_Nom;
118
119    FUNCTION Cons_Str
120      (Tn : IN Taula_De_Noms;
121       Ids : IN Rang_Tcar)
122      RETURN String IS
123
124       Idx : Rang_Tcar;
125
126    BEGIN
127       Idx := Ids;
```

```
128         WHILE (Tn.Tc(Idx) /= Ascii.Nul) LOOP
129            Idx := Idx+1;
130         END LOOP;
131
132         RETURN String(Tn.Tc(Ids..Idx-1));
133
134      END Cons_Str;
135
136
137      FUNCTION Fdisp_Tn
138        (Nom : IN String)
139        RETURN Rang_Dispersio IS
140
141         A : ARRAY (Nom'RANGE) OF Integer;
142         R : ARRAY (1..2*Nom'Last) OF Integer;
143
144         K, C, M, N : Integer;
145
146         Base : CONSTANT Integer :=
147           Character'Pos(Character'Last)+1;
148
149      BEGIN
150         N := Nom'Last;
151         M := Nom'Length;
152
153         FOR I IN 1..N LOOP
154            A(I) := Character'Pos(Nom(I));
155         END LOOP;
156
157         FOR I IN 1..2*N LOOP
158            R(I) := 0;
159         END LOOP;
160
161         FOR I IN 1..N LOOP
162            C := 0; K := I - 1;
163            FOR J IN 1..N LOOP
164               C := C + R(K+J) + A(I) + A(J);
165               R(K+J) := C MOD Base;
166               C := C/Base;
167            END LOOP;
168            R(K+N+1) := R(K+N+1) + C;
169         END LOOP;
170
```

```
171         C := (R(N+1) * Base + R(N)) MOD (Max_Id);
172
173         RETURN Rang_Dispersio(C);
174
175      END Fdisp_Tn;
176
177
178 END Decls.D_Taula_De_Noms;
```

1.3 Atributs

1.3.1 Fitxer *decls-d_ atribut.ads*

```
1  -- DECLS-D_ATRIBUT.ads
2  -- Paquet de declaracions d'atributs
3
4  WITH     Decls.Dgenerals,
5           Decls.D_Taula_De_Noms,
6           Decls.Dtnode,
7           Decls.Dtdesc;
8
9  USE      Decls.Dgenerals,
10          Decls.D_Taula_De_Noms,
11          Decls.Dtnode,
12          Decls.Dtdesc;
13
14 PACKAGE Decls.D_Atribut IS
15
16   TYPE Atribut (T : Tipus_Atribut := Atom) IS RECORD
17       Lin, Col : Natural;
18       CASE T IS
19           WHEN Atom       => NULL;
20           WHEN A_Ident    => Idn : Id_Nom;
21           WHEN A_Lit_C | A_Lit_N | A_Lit_S
22             => Val : Valor;
23           WHEN OTHERS     => A : Pnode;
24       END CASE;
25   END RECORD;
26
27 END Decls.D_Atribut;
```

2 Anàlisi Sintàctica

2.1 Gramàtica del nostre llenguatge

programa → *procediment*

procediment → **PC_PROCEDURE** *encap* **PC_IS**
declaracions
PC_BEGIN
bloc
PC_END identificador;

encap → identificador *args*

args → (*lparam*)
| λ

lparam → *lparam* ; *param*
| *param*

param → identificador : *mode* identificador

mode → **PC_IN**
| **PC_OUT**
| **PC_IN PC_OUT**

declaracions → *declaracions declaracio*
| λ

declaracio → *dec_var*
| *dec_constant*
| *dec_tipus*
| *programa*

– Manual d'usuari variables
dec_var → *lid* : identificador;

lid → *lid*, identificador
| identificador

– Manual d'usuari constant

dec_constant → identificador : **PC_CONSTANT** identificador := *valor*;

valor → *lit*
| − *lit*

– Manual d'usuari tipus

dec_tipus → *dec_subrang*
| *dec_registre*
| *dec_coleccio*

dec_subrang → **PC_TYPE** identificador **PC_IS PC_NEW** identificador
PC_RANGE *valor* .. *valor*;

dec_registre → **PC_TYPE** identificador **PC_IS PC_RECORD**
ldc
PC_END PC_RECORD;

ldc → *ldc dc*
| *dc*

dc → identificador : identificador;

– Tipus colecció (*array*)

dec_coleccio → **PC_TYPE** identificador **PC_IS PC_ARRAY**
(*lid*) **PC_OF** identificador;

lid → *lid*, identificador
| identificador

– Bloc d'instruccions

bloc → *bloc sent*
| *sent*

sent → *sassig*
| *scond*
| *srep*
| *crida_proc*

sassig → *referencia* := *expressio*;

scond → **PC_IF** *expressio* **PC_THEN**
bloc
PC_END PC_IF;
| **PC_IF** *expressio* **PC_THEN**
bloc
PC_ELSE
bloc
PC_END PC_IF;

srep → **PC_WHILE** *expressio* **PC_LOOP**
bloc
PC_END PC_LOOP;

crida_proc → *referencia*;

referencia → identificador
| *referencia*.identificador
| *referencia* (*prparam*)

prparam → *expressio*
| *expressio, prparam*

expressio → *expressio* + *expressio*
| *expressio* − *expressio*
| *expressio* ∗ *expressio*
| *expressio* / *expressio*
| *expressio* **PC_MOD** *expressio*
| *expressio* > *expressio*
| *expressio* < *expressio*
| *expressio* ≥ *expressio*
| *expressio* ≤ *expressio*
| *expressio* ≠ *expressio*
| *expressio* = *expressio*
| − *expressio*
| *expressio* && *expressio*
| *expressio* || *expressio*
| **PC_NOT** *expressio*
| (*expressio*)
| *referencia*
| *lit*

2.2 Especificació *pk_usintactica.y*

```
--Token
%token pc_procedure
%token pc_begin
%token pc_while
%token pc_if
%token pc_else
%token pc_end
%token pc_loop
%token pc_constant
%token pc_type
%token pc_array
%token pc_record
%token pc_is
%token pc_then
%token pc_not
%token pc_in
%token pc_out
%token pc_new
%token pc_null
%token pc_of
%token pc_mod
%token pc_range
%token pc_or
%token pc_and
%token s_assignacio
%token s_dospunts
%token s_final
%token s_coma
%token s_parentesiobert
%token s_parentesitancat
%token s_puntsrang
%token s_puntrec
%token op_menor
%token op_menorigual
%token op_majorigual
%token op_major
%token op_igual
%token op_distint
%token op_suma
%token op_resta
%token op_multiplicacio
```

```
42 %token op_divisio
43 %token id
44 %token const
45
46 --Precedencia
47 %left pc_or
48 %left pc_and
49 %left pc_not
50 %nonassoc op_menor op_menorigual op_majorigual
51 op_major op_igual op_distint
52 %left op_suma
53 %left op_resta
54 %left op_multiplicacio op_divisio pc_mod
55 %left menys_unitari
56
57 --Definicio del tipus atribut
58 %WITH decls.d_atribut, decls.dtnode, decls.dgenerals;
59 %USE decls.d_atribut, decls.dtnode, decls.dgenerals;
60 {
61         SUBTYPE yystype IS decls.d_atribut.atribut;
62 }
63
64 %%
65
66 --Produccions de la gramatica del llenguatge
67 programa:
68     dec_procediment
69     {creaNode_programa($$, $1);}
70   ;
71
72 dec_procediment:
73     pc_procedure encap pc_is
74         declaracions
75     pc_begin
76         bloc
77     pc_end id s_final
78     {creaNode_ID($8, $8, identificador);
79      creaNode($$, $2, $4, $6, $8, procediment);}
80   ;
81
82
83 encap:
84     id
```

```
85      {creaNode_ID($$, $1, identificador);}
86    |
87      pencap s_parentesitancat
88      {Remunta($$, $1);}
89    ;
90
91  pencap:
92      pencap s_final param
93      {creaNode($$, $1, $3, pencap);}
94    |
95      id s_parentesiobert param
96      {creaNode_ID($1, $1, identificador);
97       creaNode($$, $1, $3, pencap);}
98    ;
99
100 param:
101     id s_dospunts mode id
102     {creaNode_ID($1, $1, identificador);
103      creaNode_ID($4, $4, identificador);
104      creaNode($$, $1, $3, $4, Param);}
105   ;
106
107
108 mode:
109     pc_in
110     {creanode_mode($$, entra, mode);}
111   |
112     pc_out
113     {creanode_mode($$, surt, mode);}
114   |
115     pc_in pc_out
116     {creanode_mode($$, entrasurt, mode);}
117   ;
118
119 declaracions:
120     declaracions declaracio
121     {creaNode($$, $1, $2, declaracions);}
122   |
123     {creaNode($$, tnul);}
124   ;
125
126
127 -- DECLARACIONS
```

```
128 declaracio:
129     dec_var s_final
130     {Remunta($$, $1);}
131   |
132     dec_constant s_final
133     {Remunta($$, $1);}
134   |
135     dec_tipus s_final
136     {Remunta($$, $1);}
137   |
138     dec_procediment
139     {Remunta($$, $1);}
140   ;
141
142 dec_var:
143     id c_decl_var
144     {creaNode_ID($1, $1, identificador);
145      creaNode($$, $1, $2, dvariable);}
146   ;
147
148 c_decl_var:
149     s_dospunts id
150     {creaNode_ID($2, $2, identificador);
151      remunta($$, $2);}
152   |
153     s_coma id c_decl_var
154     {creaNode_ID($2, $2, identificador);
155      creaNode($$, $2, $3, declmultvar);}
156   ;
157
158 dec_constant:
159     id s_dospunts pc_constant id s_assignacio val
160     {creaNode_ID($1, $1, identificador);
161      creaNode_ID($4, $4, identificador);
162      creaNode($$, $1, $4, $6, dconstant);}
163   ;
164
165 -- TIPUS
166 dec_tipus:
167     decl_coleccio
168     {Remunta($$, $1);}
169   |
170     decl_registre
```

```
171     {Remunta($$, $1);}
172   |
173     decl_subrang
174     {Remunta($$, $1);}
175   ;
176
177
178 -- TIPUS SUBRANG
179 decl_subrang:
180     pc_type id pc_is pc_new id pc_range val
181     s_puntsrang val
182     {creaNode_ID($2, $2, identificador);
183      creaNode_ID($5, $5, identificador);
184      creaNode($$, $2, $5, $7, $9, dsubrang);}
185   ;
186
187 val:
188     const
189     {creaNode_VAL($$, $1, const, 1);}
190   |
191     op_resta const
192     {creaNode_VAL($$, $2, const, 0);}
193   ;
194
195 limit:
196     const
197     {creaNode_VAL($$, $1, Const, 1);}
198   |
199     op_resta const
200     {creaNode_VAL($$, $2, const, 0);}
201   |
202     id
203     {creaNode_ID($$, $1, identificador);}
204   ;
205
206
207 -- TIPUS REGISTRE
208 decl_registre:
209     p_dregistre pc_end pc_record
210     {creaNode($$, $1, firecord);}
211   ;
212
213 p_dregistre:
```

```
214     p_dregistre id s_dospunts id s_final
215     {creaNode_ID($2, $2, identificador);
216      creaNode_ID($4, $4, identificador);
217      creaNode($$, $1, $2, $4, dencapregistre);}
218   |
219     pc_type id pc_is pc_record id s_dospunts id s_final
220     {creaNode_ID($2, $2, identificador);
221      creaNode_ID($5, $5, identificador);
222      creaNode_ID($7, $7, identificador);
223      creaNode($$, $2, $5, $7, Dregistre);}
224   ;
225
226
227 -- TIPUS COLECCIO
228 decl_coleccio:
229     p_dcoleccio s_parentesitancat pc_of id
230     {creaNode_ID($4, $4, identificador);
231      creaNode($$, $1, $4, Dcoleccio);}
232   ;
233
234 p_dcoleccio:
235     p_dcoleccio s_coma id
236     {creaNode_ID($3, $3, identificador);
237      creaNode($$, $1, $3, Pcoleccio);}
238   |
239     pc_type id pc_is pc_array s_parentesiobert id
240     {creaNode_ID($2, $2, identificador);
241      creaNode_ID($6, $6, identificador);
242      creaNode($$, $2, $6, Pdimcoleccio);}
243   ;
244
245
246 -- BLOC D'INSTRUCCIO
247 bloc:
248     bloc sentencia s_final
249      {creaNode($$, $1, $2, bloc);}
250
251   |
252     sentencia s_final
253      {Remunta($$, $1);}
254   ;
255
256
```

```
257 -- SENTENCIES D'INSTRUCCIONS
258 sentencia:
259     sassig
260     {Remunta($$, $1);}
261   |
262     scond
263     {Remunta($$, $1);}
264   |
265     srep
266     {Remunta($$, $1);}
267   |
268     crida_proc
269     {Remunta($$, $1);}
270   ;
271
272 -- Sentencia assignacio
273 sassig:
274     referencia s_assignacio expressio
275     {creaNode($$, $1, $3, assignacio);}
276   ;
277
278 -- Sentencia condicional
279 scond:
280     pc_if expressio pc_then
281         bloc
282     pc_end pc_if
283     {creaNode($$, $2, $4, CondicionalS);}
284   |
285     pc_if expressio pc_then
286         bloc
287     pc_else
288         bloc
289     pc_end pc_if
290     {creaNode($$, $2, $4, $6, CondicionalC);}
291   ;
292
293 -- Sentencia bucle
294 srep:
295     pc_while expressio pc_loop
296         bloc
297     pc_end pc_loop
298     {creaNode($$, $2, $4, Repeticio);}
299   ;
```

```
300
301 -- Sentencia crida a procediment
302 crida_proc:
303     referencia
304     {Remunta($$, $1);}
305   ;
306
307 referencia:
308     id
309     {creaNode_ID($$, $1, identificador);}
310   |
311     referencia s_puntrec id
312     {creaNode_ID($3, $3, identificador);
313      creaNode($$, $1, $3, referencia);}
314   |
315     pri s_parentesitancat
316     {creaNode($$, $1, fireferencia);}
317   ;
318
319 pri:
320     referencia s_parentesiobert expressio
321     {creaNode($$, $1, $3, encappri);}
322   |
323     pri s_coma expressio
324     {creaNode($$, $1, $3, pri);}
325   ;
326
327
328 -- Expressions
329 expressio:
330     expressio pc_or expressio
331     {creaNode($$, $1, $3, Unio, Expressio);}
332   |
333     expressio pc_and expressio
334     {creaNode($$, $1, $3, Interseccio, Expressio);}
335   |
336     pc_not expressio     %prec pc_not
337     {creaNode($$, $2, Negacio, ExpressioUnaria);}
338   |
339     expressio op_menor expressio
340     {creaNode($$, $1, $3, Menor, Expressio);}
341   |
342     expressio op_menorigual expressio
```

```
343     {creaNode($$, $1, $3, Menorig, Expressio);}
344   |
345     expressio op_majorigual expressio
346     {creaNode($$, $1, $3, Majorig, Expressio);}
347   |
348     expressio op_major expressio
349     {creaNode($$, $1, $3, Major, Expressio);}
350   |
351     expressio op_igual expressio
352     {creaNode($$, $1, $3, Igual, Expressio);}
353   |
354     expressio op_distint expressio
355     {creaNode($$, $1, $3, Distint, Expressio);}
356   |
357     expressio op_suma expressio
358     {creaNode($$, $1, $3, Suma, Expressio);}
359   |
360     expressio op_resta expressio
361     {creaNode($$, $1, $3, Resta, Expressio);}
362   |
363     expressio op_multiplicacio expressio
364     {creaNode($$, $1, $3, Mult, Expressio);}
365   |
366     expressio op_divisio expressio
367     {creaNode($$, $1, $3, Div, Expressio);}
368   |
369     expressio pc_mod expressio
370     {creaNode($$, $1, $3, Modul, Expressio);}
371   |
372     op_resta expressio   %prec menys_unitari
373     {creaNode($$, $2, Resta, ExpressioUnaria);}
374   |
375     s_parentesiobert expressio s_parentesitancat
376     {Remunta($$, $2);}
377   |
378     referencia
379     {Remunta($$, $1);}
380   |
381     const
382     {creaNode_VAL($$, $1, Const, 1);}
383   ;
384
385
```

```
386 %%
387
388
389 PACKAGE pk_usintactica IS
390
391     PROCEDURE yyparse;
392
393
394 END pk_usintactica;
395
396
397
398 WITH     pk_usintactica_tokens,
399          pk_usintactica_shift_reduce,
400          pk_usintactica_goto,
401          pk_ulexica_io,
402          u_lexica,
403          semantica,
404          decls.dtnode,
405          Ada.text_IO;
406
407 USE      pk_usintactica_tokens,
408          pk_usintactica_shift_reduce,
409          pk_usintactica_goto,
410          pk_ulexica_io,
411          u_lexica,
412          semantica,
413          decls.dtnode,
414          ada, --no llevar mai
415          ada.text_io;
416
417 PACKAGE BODY pk_usintactica IS
418     PROCEDURE YYError (e : IN string) IS
419     BEGIN
420         Put_Line(e);
421         RAISE Syntax_Error;
422     END YYError;
423 ##
424 END pk_usintactica;
```

3 Anàlisi Semàntica

3.1 Taula de simbols

3.1.1 Fitxer *decls-dtsimbols.ads*

```
1  -- DECLS-DTSIMBOLS.ads
2  -- Declaracions de taula de simbols
3
4  WITH     Decls.Dtdesc,
5           Decls.Dgenerals,
6           Decls.D_Taula_De_Noms,
7           Ada.Text_IO;
8
9  USE      Decls.Dtdesc,
10          Decls.Dgenerals,
11          Decls.D_Taula_De_Noms,
12          Ada.Text_IO;
13
14
15 PACKAGE Decls.Dtsimbols IS
16
17     --pragma pure;
18
19     TYPE Tsimbols IS PRIVATE;
20     TYPE Ttsimbols IS ARRAY
21           (Num_Proc) OF Tsimbols;
22
23     --Serveix per al joc de proves
24     TYPE Cursor_Idx IS NEW Rang_Despl;
25     TYPE Cursor_Arg IS NEW Rang_Despl;
26
27     -- Operacions
28     -- VERSIO 1: llenguatge simple sense estructura
29     -- de blocs estil Fortran.
30     PROCEDURE Printts
31       (Ts : IN Tsimbols;
32        Tn : IN Taula_De_Noms);
33
34     PROCEDURE Tbuida
35       (Ts : OUT Tsimbols);
36
37     PROCEDURE posa
```

```
38        (ts : IN OUT tsimbols;
39         id : IN id_nom;
40          d : IN descrip;
41          e : OUT boolean);
42
43      FUNCTION cons
44        (ts : IN tsimbols;
45         id : IN id_nom) RETURN descrip;
46
47      -- VERSIO 2: Normal, llenguatge amb blocs
48      -- estil Pascal.
49      PROCEDURE entrabloc
50        (ts : IN OUT tsimbols);
51
52      PROCEDURE surtbloc
53        (ts : IN OUT tsimbols;
54             tn : IN taula_de_noms);
55
56      -- VERSIO 3: Blocs mes records.
57      PROCEDURE posacamp
58        (ts : IN OUT tsimbols;
59        idr : IN id_nom;
60        idc : IN id_nom;
61          d : IN descrip;
62          e : OUT boolean);
63
64      FUNCTION conscamp
65        (ts : IN tsimbols;
66        idr : IN id_nom;
67        idc : IN id_nom) RETURN descrip;
68
69      -- VERSIO 4: Arrays.
70      PROCEDURE posa_idx
71        (ts : IN OUT tsimbols;
72        ida : IN id_nom;
73        idi : IN id_nom;
74          e : OUT boolean);
75
76      FUNCTION primer_idx
77        (ts : IN tsimbols;
78        ida : IN id_nom) RETURN cursor_idx;
79
80      FUNCTION idx_valid
```

```
81        (ci : IN cursor_idx) RETURN boolean;
82
83      FUNCTION succ_idx
84        (ts : IN tsimbols;
85         ci : IN cursor_idx) RETURN cursor_idx;
86
87      FUNCTION cons_idx
88        (ts : IN tsimbols;
89         ci : IN cursor_idx) RETURN id_nom;
90
91      -- VERSIO 5: Procediments
92      PROCEDURE posa_arg
93        (ts : IN OUT tsimbols;
94        idp : IN id_nom;
95        ida : IN id_nom;
96         da : IN descrip;
97          e : OUT boolean);
98
99      FUNCTION primer_arg
100       (ts : IN tsimbols;
101        idp : IN id_nom) RETURN cursor_arg;
102
103     FUNCTION Succ_Arg
104       (ts : IN tsimbols;
105        ca : IN cursor_arg) RETURN cursor_arg;
106
107     FUNCTION arg_valid
108       (Ca : IN Cursor_arg) RETURN boolean;
109
110     PROCEDURE cons_arg
111       (ts : IN tsimbols;
112        ca : IN cursor_arg;
113       ida : OUT id_nom;
114        dn : OUT descrip);
115
116     PROCEDURE actualitza
117       (ts : IN OUT tsimbols;
118        id : IN id_nom;
119         d : IN descrip);
120
121 PRIVATE
122
123
```

```
124     TYPE tipus_descripcio IS RECORD
125        np : nprof;
126         d : descrip;
127         s : rang_despl;
128     END RECORD;
129
130     TYPE tipus_expansio IS RECORD
131        np : nprof;
132         d : descrip;
133        id : id_nom;
134         s : rang_despl;
135     END RECORD;
136
137     TYPE taula_ambits IS ARRAY
138       (1 .. nprof'Last) OF rang_despl;
139
140     TYPE taula_expansio IS ARRAY
141       (1 .. rang_despl'Last) OF tipus_expansio;
142
143     TYPE taula_desc IS ARRAY
144       (1 .. id_nom'Last) OF tipus_descripcio;
145
146     TYPE tsimbols IS RECORD
147        tdesc : taula_desc;
148         texp : taula_expansio;
149       tambit : taula_ambits;
150         prof : nprof;
151     END RECORD;
152
153 END Decls.Dtsimbols;
```

3.1.2 Fitxer *decls-dtsimbols.adb*

```
1  -- DECLS-DTSIMBOLS.adb
2  -- Procediments de la taula de simbols
3
4  PACKAGE BODY Decls.Dtsimbols IS
5
6     PROCEDURE printts
7       (ts : IN tsimbols;
8        tn : IN taula_de_noms) IS
9     BEGIN
10       New_Line;
11       Put_Line("");
12       Put_Line("tdesc ------------");
13       FOR i IN 1 .. (id_nom'Last-970) LOOP
14          Put("tdesc["&i'img&"] := (");
15          Put(ts.tdesc(i).np'img&", ");
16          CASE ts.tdesc(i).d.td IS
17             WHEN dnula => Put("dnula, ");
18             WHEN dtipus => Put("dtipus, ");
19             WHEN dvar => Put("dvar, ");
20             WHEN dproc => Put("dproc, ");
21             WHEN dconst => Put("dconst, ");
22             WHEN dargc => Put("dargc, ");
23             WHEN dcamp => Put("dcamp, ");
24          END CASE;
25          Put(ts.tdesc(i).s'img&") "&cons_nom(tn,i));
26          New_Line;
27       END LOOP;
28       Put_Line("PROFUNDITAT: "&ts.prof'img);
29    END printts;
30
31
32    -- VERSIO 1: llenguatge simple sense estructura
33    -- de blocs estil Fortran.
34    PROCEDURE tbuida
35      (ts : OUT tsimbols) IS
36       nul_desc : descrip(dnula);
37    BEGIN
38       ts.prof := 1;
39       ts.tambit(ts.prof) := nul_despl;
40       FOR i IN 1 .. id_nom'Last LOOP
41          ts.tdesc(i) := (nul_nprof, nul_desc,
```

```
42                          nul_despl);
43       END LOOP;
44    END tbuida;
45
46
47    PROCEDURE posa
48      (ts : IN OUT tsimbols;
49       id : IN id_nom;
50       d : IN descrip;
51       e : OUT boolean) IS
52       idespl : rang_despl;
53    BEGIN
54       e := (ts.tdesc(id).np = ts.prof);
55       IF NOT e THEN
56          ts.tambit(ts.prof) := ts.tambit(ts.prof) + 1;
57          idespl := ts.tambit(ts.prof);
58          WHILE ts.texp(idespl).np = no_prof LOOP
59             idespl:= idespl +1;
60             ts.tambit(ts.prof) := 1 + ts.tambit(ts.prof);
61          END LOOP;
62          ts.texp(idespl) := (ts.tdesc(id).np,
63                              ts.tdesc(id).d, id, 0);
64          ts.tdesc(id) := (ts.prof, d, 0);
65       END IF;
66    END posa;
67
68
69    FUNCTION cons
70      (ts : IN tsimbols;
71       id : IN id_nom)
72      RETURN descrip IS
73    BEGIN
74       RETURN ts.tdesc(id).d;
75    END cons;
76
77
78    -- VERSIO 2: Normal, llenguatge amb blocs estil
79    -- Pascal.
80    PROCEDURE Entrabloc
81      (Ts : IN OUT Tsimbols) IS
82    BEGIN
83       Ts.Prof := Ts.Prof + 1;
84       Ts.Tambit(Ts.Prof) := Ts.Tambit(Ts.Prof - 1);
```

```
85    END Entrabloc;
86
87
88    PROCEDURE surtbloc
89      (ts : IN OUT tsimbols;
90       tn : IN taula_de_noms) IS
91       idespl1 : rang_despl;
92       idespl2 : rang_despl;
93       id : id_nom;
94    BEGIN
95       idespl1 := ts.tambit(ts.prof);
96       ts.prof := ts.prof - 1;
97       idespl2 := ts.tambit(ts.prof)+1;
98       FOR idespl IN REVERSE idespl2 .. idespl1 LOOP
99          IF ts.texp(idespl).np > no_prof THEN
100            id := ts.texp(idespl).id;
101            ts.tdesc(id).d := ts.texp(idespl).d;
102            ts.tdesc(id).np := ts.texp(idespl).np;
103            ts.tdesc(id).s := ts.texp(idespl).s;
104         END IF;
105      END LOOP;
106   END surtbloc;
107
108
109   -- VERSIO 3: Blocs mes records.
110   PROCEDURE posacamp
111     (ts : IN OUT tsimbols;
112      idr : IN id_nom;
113      idc : IN id_nom;
114      d : IN descrip;
115      e : OUT boolean) IS
116      des : descrip;
117      td : descriptipus;
118      p : rang_despl;
119      itdespl : rang_despl;
120   BEGIN
121      des := ts.tdesc(idr).d;
122      IF des.td /= dtipus THEN e := TRUE; END IF;
123
124      td := des.dt;
125      IF td.tt /= tsrec THEN e := TRUE; END IF;
126
127      p := ts.tdesc(idr).s;
```

```
128       WHILE p /= 0 AND THEN ts.texp(p).id /= idc LOOP
129          p := ts.texp(p).s;
130       END LOOP;
131
132       e := (p /= 0);
133       IF NOT e THEN
134          ts.tambit(ts.prof) := ts.tambit(ts.prof) + 1;
135          itdespl := ts.tambit(ts.prof);
136          ts.texp(itdespl) := (no_prof, d, idc,
137                               ts.tdesc(idr).s);
138          ts.tdesc(idr).s := itdespl;
139       END IF;
140    END posacamp;
141
142
143    FUNCTION conscamp
144      (ts : IN tsimbols;
145       idr : IN id_nom;
146       idc : IN id_nom) RETURN descrip IS
147       d : descrip;
148       td : tdescrip;
149       p : rang_despl;
150       descnula : descrip(dnula);
151    BEGIN
152       d := ts.tdesc(idr).d;
153       td := d.td;
154       p := ts.tdesc(idr).s;
155       WHILE p /= 0 AND THEN ts.texp(p).id /= idc LOOP
156          p := ts.texp(p).s;
157       END LOOP;
158
159       IF p = 0 THEN
160          RETURN descnula;
161       ELSE
162          RETURN ts.texp(p).d;
163       END IF;
164    END conscamp;
165
166
167    -- VERSIO 4: Arrays.
168    PROCEDURE posa_idx
169      (ts : IN OUT tsimbols;
170       ida : IN id_nom;
```

```
171         idi : IN id_nom;
172         e : OUT boolean) IS
173         d : descrip;
174         dt : descriptipus;
175         p : rang_despl;
176         pp : rang_despl;
177         idespl : rang_despl;
178      BEGIN
179         E := False;
180         d := ts.tdesc(ida).d;
181         IF d.td /= dtipus THEN e := TRUE; END IF;
182         dt := d.dt;
183         IF dt.tt /= tsarr THEN e := TRUE; END IF;
184
185         p := ts.tdesc(ida).s;
186         pp := 0;
187         WHILE p /= 0 LOOP
188            pp := p;
189            p := ts.texp(p).s;
190         END LOOP;
191
192         ts.tambit(ts.prof) := ts.tambit(ts.prof) + 1;
193         idespl := ts.tambit(ts.prof);
194         ts.texp(idespl) := (no_prof, (td => dnula),
195                             idi, 0);
196
197         IF pp /= 0 THEN
198            ts.texp(pp).s := idespl;
199         ELSE
200            ts.tdesc(ida).s := idespl;
201         END IF;
202      END posa_idx;
203
204
205      FUNCTION primer_idx
206        (ts : IN tsimbols;
207         ida : IN id_nom) RETURN cursor_idx IS
208      BEGIN
209         RETURN cursor_idx(ts.tdesc(ida).s);
210      END primer_idx;
211
212
213      FUNCTION idx_valid
```

```
214       (ci : IN cursor_idx) RETURN boolean IS
215     BEGIN
216        RETURN ci > 0;
217     END idx_valid;
218
219
220     FUNCTION succ_idx
221       (ts : IN tsimbols;
222        ci : IN cursor_idx) RETURN cursor_idx IS
223     BEGIN
224        IF idx_valid(ci) THEN
225           RETURN cursor_idx(ts.texp(rang_despl(ci)).s);
226        ELSE
227           RETURN 0; --Excepcio
228        END IF;
229     END succ_idx;
230
231
232     FUNCTION cons_idx
233       (ts : IN tsimbols;
234        ci : IN cursor_idx) RETURN id_nom IS
235     BEGIN
236        RETURN  ts.texp(rang_despl(ci)).id;
237     END cons_idx;
238
239
240     -- PROCEDIMENTS
241     PROCEDURE posa_arg
242       (ts : IN OUT tsimbols;
243        idp : IN id_nom;
244        ida : IN id_nom;
245        da : IN descrip;
246        e : OUT boolean) IS
247        d : descrip;
248        p : rang_despl;
249        pp : rang_despl;
250        idespl : rang_despl;
251     BEGIN
252        e:= false;
253        d := ts.tdesc(idp).d;
254        IF d.td /= dproc THEN e := TRUE; END IF;
255
256        p := ts.tdesc(idp).s;
```

```
257         pp := 0;
258         WHILE p /= 0 LOOP
259            pp := p;
260            p := ts.texp(p).s;
261         END LOOP;
262
263         ts.tambit(ts.prof) := ts.tambit(ts.prof) + 1;
264         idespl := ts.tambit(ts.prof);
265         ts.texp(idespl) := (no_prof, da, ida, 0);
266         IF pp /= 0 THEN
267            ts.texp(pp).s := idespl;
268         ELSE
269            ts.tdesc(idp).s := idespl;
270         END IF;
271      END Posa_Arg;
272
273
274      FUNCTION Primer_Arg
275        (Ts : IN Tsimbols;
276         Idp : IN Id_Nom) RETURN Cursor_Arg IS
277      BEGIN
278         RETURN cursor_arg(ts.tdesc(idp).s);
279      END Primer_Arg;
280
281
282      FUNCTION Succ_Arg
283        (ts : IN tsimbols;
284         ca : IN cursor_arg) RETURN cursor_arg IS
285      BEGIN
286         IF arg_valid(ca) THEN
287            RETURN cursor_arg(ts.texp(rang_despl(ca)).s);
288         ELSE
289            RETURN 0; --Excepcio
290         END IF;
291      END Succ_Arg;
292
293
294      FUNCTION Arg_Valid
295        (Ca : IN Cursor_Arg) RETURN Boolean IS
296      BEGIN
297         RETURN Ca > 0;
298      END Arg_Valid;
299
```

```
300
301     PROCEDURE cons_arg
302       (ts : IN tsimbols;
303        ca : IN cursor_arg;
304        ida : OUT id_nom;
305        Dn : OUT Descrip) IS
306     BEGIN
307        Ida := ts.texp(rang_despl(ca)).id;
308        Dn := Ts.Texp(Rang_Despl(Ca)).D;
309     END Cons_Arg;
310
311
312     PROCEDURE Actualitza
313       (Ts : IN OUT Tsimbols;
314        Id : IN Id_Nom;
315        D : IN Descrip) IS
316     BEGIN
317        Ts.Tdesc(id).D := D;
318     END Actualitza;
319
320 END Decls.Dtsimbols;
```

3.2 Descripció

3.2.1 Fitxer *decls-dtdesc.ads*

```
1  -- DECLS-DTDESC.ads
2  -- Declaracions de descripcio
3
4  WITH     Decls.Dgenerals,
5           Decls.D_Taula_De_Noms;
6
7  USE      Decls.Dgenerals,
8           Decls.D_Taula_De_Noms;
9
10 PACKAGE Decls.Dtdesc IS
11
12     --pragma pure;
13
14     -- Representa tambit
15     Max_Nprof : CONSTANT Integer := 25;
16     TYPE Nprof IS NEW Integer
17       RANGE -1 .. Max_Nprof;
18     Nul_Nprof : CONSTANT Nprof := 0;
19     No_Prof : CONSTANT Nprof := -1;
20
21     TYPE Despl IS NEW Integer;
22
23     -- Representa texpansio
24     TYPE Rang_Despl IS NEW Integer
25       RANGE 0 .. (Max_Id * Max_Nprof);
26     Nul_Despl : CONSTANT Rang_Despl := 0;
27
28     TYPE Tdescrip IS
29       (Dnula,
30        Dconst,
31        Dvar,
32        Dtipus,
33        Dproc,
34        Dcamp,
35        Dargc);
36
37     TYPE Tipussubjacent IS
38       (Tsbool,
39        Tscar,
40        Tsstr,
```

```
41          Tsent,
42          Tsrec,
43          Tsarr,
44          Tsnul);
45
46      TYPE Descriptipus (Tt: Tipussubjacent := Tsnul) IS
47          RECORD
48              Ocup : Despl;
49              CASE Tt IS
50                  WHEN Tsbool | Tscar | Tsent =>
51                     Linf, Lsup : Valor;
52                  WHEN Tsarr | Tsstr => Tcamp : Id_Nom;
53                     Base : Valor;
54                  WHEN Tsrec | Tsnul  => NULL;
55              END CASE;
56          END RECORD;
57
58      TYPE Descrip (Td : Tdescrip := Dnula) IS
59          RECORD
60              CASE Td IS
61                  WHEN Dnula  => NULL;
62                  WHEN Dtipus => Dt : Descriptipus;
63                  WHEN Dvar   => Tr : Id_Nom;
64                                 Nv : Num_Var;
65                  WHEN Dproc  => Np : Num_Proc;
66                  WHEN Dconst => Tc : Id_Nom;
67                                 Vc : Valor;
68                                 Nvc : Num_Var;
69                  WHEN Dargc  => Nvarg : Num_Var;
70                                  Targ : Id_Nom;
71                  WHEN Dcamp  => Tcamp : Id_Nom;
72                                   Dsp : Despl;
73              END CASE;
74          END RECORD;
75
76  END Decls.Dtdesc;
```

3.3 Semàntica

3.3.1 Fitxer *semantica.ads*

```
1  WITH Decls.Dgenerals,
2    Decls.Dtnode,
3    Decls.D_Taula_De_Noms,
4    Decls.D_Atribut,
5    Decls.dtsimbols,
6    Ada.Text_Io,
7    Decls.dtdesc;
8
9  USE Decls.Dgenerals,
10   Decls.Dtnode,
11   Decls.D_Taula_De_Noms,
12   Decls.D_Atribut,
13   Decls.dtsimbols,
14   Ada.Text_Io,
15   Decls.dtdesc;
16
17 PACKAGE Semantica IS
18
19    --Definicions basiques
20    TYPE tInstruccio IS
21      (--1 operand
22       Rtn,
23       Call,
24       Preamb,
25       Params,
26       Etiqueta,
27       Branc_Inc,
28       -- 2 operands
29       Negacio,
30       Op_Not,
31       Copia,
32       Paramc,
33       --3 operands
34       Suma,
35       Resta,
36       Producte,
37       Divisio,
38       Modul,
39       Op_And,
40       Op_Or,
```

```
41        Consindex,
42        Asigindex,
43        Menor,
44        Menorigual,
45        Igual,
46        Majorigual,
47        Major,
48        Diferent);
49
50     TYPE tCamp IS
51       (Proc,
52        Var,
53        Etiq,
54        Const);
55
56     TYPE Camp(tc : tCamp:=Const) IS RECORD
57        CASE Tc IS
58           WHEN Proc    => Idp : num_Proc;
59           WHEN Var     => Idv : num_var;
60           WHEN Etiq    => Ide : num_etiq;
61           WHEN Const   => Idc : num_var;
62           WHEN OTHERS => NULL;
63        END CASE;
64     END RECORD;
65
66     TYPE C3a IS RECORD
67        Instr : tInstruccio;
68        Camp1 : Camp;
69        Camp2 : Camp;
70        Camp3 : Camp;
71     END RECORD;
72
73     TYPE Tprocediment IS
74       (Intern,
75        Extern);
76
77     TYPE Info_Proc (Tp : Tprocediment := Intern) IS
78        RECORD
79           Ocup_Param : Despl;
80           CASE Tp IS
81              WHEN Intern =>
82                 Idn         : Id_Nom;
83                 Prof        : nprof;
```

```
84                Ocup_Var    : Despl;
85                Etiq : Num_Etiq;
86             WHEN Extern =>
87                Etiq_extern : Id_Nom;
88          END CASE;
89       END RECORD;
90
91    Info_Proc_Nul : Info_Proc := (Intern, 0, Id_Nul,
92                                  0, 0, Etiq_Nul);
93
94    TYPE Taula_P IS ARRAY
95      (Num_Proc) OF Info_Proc;
96
97    TYPE T_Procs IS RECORD
98       Tp : Taula_P;
99       Np : Num_Proc;
100   END RECORD;
101
102   Tp : T_Procs;
103
104   --Taula de variables
105   TYPE Info_Var IS RECORD
106      Id        : Id_Nom;
107      Np        : num_proc;
108      Ocup      : Despl;
109      Desp      : Despl;
110      Tsub      : Tipussubjacent;
111      Param     : Boolean;
112      Const     : Boolean;
113      Valconst  : Valor;
114   END RECORD;
115
116   Info_Var_Nul : Info_Var :=
117     (Id        => Id_Nul,
118      Np        => proc_nul,
119      Ocup      => 0,
120      Desp      => 0,
121      Tsub      => Tsnul,
122      Param     => False,
123      Const     => False,
124      Valconst  => 0);
125
126   TYPE taula_v IS ARRAY
```

```
127       (Num_Var) OF Info_Var;
128
129     TYPE T_Vars IS RECORD
130        Tv : taula_v;
131        Nv : num_var;
132     END RECORD;
133
134     Tv : T_Vars;
135     Ne : Num_Etiq := 0;
136     Arbre : Pnode;
137     -- Per els brancaments
138     Zero,
139     MenysU : num_Var;
140
141     -- Procediments
142     PROCEDURE Abuit
143       (P : OUT pnode);
144
145     PROCEDURE creaNode_programa
146       (P : OUT Atribut;
147        A : IN Atribut);
148
149     PROCEDURE creaNode
150       (p : OUT atribut;
151        fe, fd : IN atribut;
152        tn : IN Tipusnode);
153
154     PROCEDURE creaNode
155       (p : OUT atribut;
156        fe, fc, fd : IN atribut;
157        tn : IN Tipusnode);
158
159     PROCEDURE creaNode
160       (p : OUT atribut;
161        fe, fce, fc, fd : IN atribut;
162        tn : IN Tipusnode);
163
164     PROCEDURE creaNode
165       (p : OUT atribut;
166        f : IN atribut;
167        tn : IN Tipusnode);
168
169     PROCEDURE creaNode
```

```
170       (p : OUT atribut;
171        fe, fd: IN atribut;
172        op : IN operacio;
173        tn : IN Tipusnode);
174
175     PROCEDURE creaNode
176       (p : OUT atribut;
177        f : IN atribut;
178        op : IN operacio;
179        tn : IN Tipusnode);
180
181     PROCEDURE CreaNode_ID
182       (p : OUT atribut;
183        id : IN atribut;
184        tn : IN Tipusnode);
185
186     PROCEDURE CreaNode_VAL
187       (p : OUT atribut;
188        a : IN atribut;
189        tn : IN Tipusnode;
190        S : IN Valor);
191
192     PROCEDURE CreaNode_MODE
193       (P : OUT Atribut;
194        M : IN Mmode;
195        Tn : IN Tipusnode);
196
197     PROCEDURE creaNode
198       (P : OUT Atribut;
199        Tn : IN Tipusnode);
200
201     PROCEDURE Remunta
202       (P : OUT Atribut;
203        A : IN Atribut);
204
205     PROCEDURE Cons_Tnode
206       (P : IN Pnode;
207        Tn : OUT Tipusnode);
208
209     -- Procediments per a les Taules
210     PROCEDURE Noves_taules
211       (Tp : OUT T_Procs;
212        Tv : OUT T_Vars);
```

```
213
214     -- Procediments per Taula de Procediments
215     PROCEDURE Posa
216       (Tp  : IN OUT T_Procs;
217        Ip  : IN Info_Proc;
218        Idp : OUT num_Proc);
219
220     PROCEDURE Modif_Descripcio
221       (Tp  : IN OUT T_Procs;
222        Idp : IN num_proc;
223        Ip  : IN Info_Proc);
224
225     -- Procediments per Taula de Variables
226     PROCEDURE Posa
227       (Tv : IN OUT T_Vars;
228        Iv : IN Info_Var;
229        Idv : OUT num_var);
230
231 PRIVATE
232
233     Ts : Tsimbols;
234     Tts: Ttsimbols;
235     Tn : Taula_De_Noms;
236     Nv : Num_Var;
237     Np : Num_Proc;
238
239     Id_Puts : Num_Proc;
240     Id_Gets : Num_Proc;
241
242 END Semantica;
```

3.3.2 Fitxer *semantica.adb*

```
1  PACKAGE BODY Semantica IS
2
3     PROCEDURE Abuit
4       (P : OUT Pnode) IS
5     BEGIN
6        P := NULL;
7     END Abuit;
8
9
10    PROCEDURE Creanode_Programa
11      (P : OUT Atribut;
12       A : IN Atribut) IS
13    BEGIN
14       P := A;
15       Arbre := P.A;
16    END Creanode_Programa;
17
18
19    PROCEDURE Creanode
20      (P : OUT Atribut;
21       Fe, Fd : IN Atribut;
22       Tn : IN Tipusnode) IS
23       Paux : Pnode;
24    BEGIN
25       Paux := NEW Node(Tn);
26       Paux.Fe1 := Fe.A;
27       Paux.Fd1 := Fd.A;
28       P := (Nodearbre, 0, 0, Paux);
29    END Creanode;
30
31
32    PROCEDURE Creanode
33      (P : OUT Atribut;
34       Fe, Fc, Fd : IN Atribut;
35       Tn : IN Tipusnode) IS
36       Paux : Pnode;
37    BEGIN
38       Paux := NEW Node(Tn);
39       Paux.Fe2 := Fe.A;
40       Paux.Fd2 := Fd.A;
41       Paux.Fc2 := Fc.A;
```

```
42        P := (Nodearbre, 0, 0, Paux);
43    END Creanode;
44
45
46    PROCEDURE Creanode
47      (P : OUT Atribut;
48       Fe, Fd : IN Atribut;
49       Op : IN Operacio;
50       Tn : IN Tipusnode) IS
51       Paux : Pnode;
52    BEGIN
53       Paux := NEW Node(Tn);
54       Paux.Fe3 := Fe.A;
55       Paux.Fd3 := Fd.A;
56       Paux.Op3 := Op;
57       P := (Nodearbre, 0, 0, Paux);
58    END Creanode;
59
60
61    PROCEDURE Creanode
62      (P : OUT Atribut;
63       F : IN Atribut;
64       Op : IN Operacio;
65       Tn : IN Tipusnode) IS
66       Paux : Pnode;
67    BEGIN
68       Paux := NEW Node(Tn);
69       Paux.F4 := F.A;
70       Paux.Op4 := Op;
71       P := (Nodearbre, 0, 0, Paux);
72    END Creanode;
73
74
75    PROCEDURE Creanode
76      (P : OUT Atribut;
77       Fe, Fce, Fc, Fd : IN Atribut;
78       Tn : IN Tipusnode) IS
79       Paux : Pnode;
80    BEGIN
81       Paux := NEW Node(Tn);
82       Paux.Fe5 := Fe.A;
83       Paux.Fc5 := Fce.A;
84       Paux.Fd5 := Fc.A;
```

```
85        Paux.Fid5 := Fd.A;
86        P := (Nodearbre, 0, 0, Paux);
87     END Creanode;
88
89     PROCEDURE Creanode
90       (P : OUT atribut;
91        F : IN atribut;
92        Tn : IN Tipusnode) IS
93        Paux : Pnode;
94     BEGIN
95        Paux := NEW Node(Tn);
96        Paux.F6 := F.A;
97        P := (Nodearbre, 0, 0, Paux);
98     END Creanode;
99
100    -- Crea node per identificadors
101    PROCEDURE Creanode_Id
102      (P : OUT Atribut;
103       Id : IN Atribut;
104       Tn : IN Tipusnode) IS
105       Paux : Pnode;
106    BEGIN
107       Paux := NEW Node(Tn);
108       Paux.Id12 := Id.Idn;
109       Paux.L1 := Id.Lin;
110       Paux.C1 := Id.Col;
111       P := (Nodearbre, 0, 0, Paux);
112    END Creanode_Id;
113
114
115    PROCEDURE Creanode_Val
116      (P : OUT Atribut;
117       A : IN Atribut;
118       Tn : IN Tipusnode;
119       S : IN Valor) IS
120       Paux : Pnode;
121    BEGIN
122       Paux := NEW Node(Tn);
123       IF S = 0 THEN
124          Paux.Val := A.Val*(-1);
125       ELSE
126          Paux.Val := A.Val;
127       END IF;
```

```
128         Paux.Tconst := A.T;
129         Paux.L2 := A.Lin;
130         Paux.C2 := A.Col;
131         P := (Nodearbre, 0, 0, Paux);
132     END Creanode_Val;
133
134
135     PROCEDURE Creanode_Mode
136       (P : OUT Atribut;
137        M : IN mmode;
138        Tn : IN Tipusnode) IS
139        Paux : Pnode;
140     BEGIN
141        Paux := NEW Node(Tn);
142        Paux.M12 := M;
143        P := (NodeArbre, 0, 0, Paux);
144     END Creanode_Mode;
145
146
147     PROCEDURE Creanode
148       (P : OUT Atribut;
149        Tn : IN Tipusnode) IS
150        Paux : Pnode;
151     BEGIN
152        Paux := NEW Node(tn);
153        P := (NodeArbre, 0, 0, Paux);
154     END Creanode;
155
156
157     PROCEDURE Remunta
158       (P : OUT Atribut;
159        A : IN Atribut) IS
160     BEGIN
161        P := A;
162     END Remunta;
163
164
165     PROCEDURE Cons_Tnode
166       (P : IN Pnode;
167        Tn : OUT Tipusnode) IS
168     BEGIN
169        Tn := P.Tipus;
170     END Cons_Tnode;
```

```
171
172   -- Procediments per a les Taules
173   PROCEDURE Noves_taules
174     (Tp : OUT T_Procs;
175      Tv : OUT T_Vars) IS
176   BEGIN
177      Tp.Np := 0;
178      Tv.Nv := 0;
179   END Noves_taules;
180
181
182   -- Procediments per Taula de Procediments
183   PROCEDURE Posa
184     (Tp  : IN OUT T_Procs;
185      Ip  : IN Info_Proc;
186      Idp : OUT num_Proc) IS
187   BEGIN
188      Tp.Np := Tp.Np+1;
189      Tp.Tp(Tp.Np) := Ip;
190      Idp := Tp.Np;
191   END Posa;
192
193
194   PROCEDURE Modif_Descripcio
195     (Tp  : IN OUT T_Procs;
196      Idp : IN Num_Proc;
197      Ip  : IN Info_Proc) IS
198   BEGIN
199      Tp.Tp(Idp) := Ip;
200   END Modif_Descripcio;
201
202
203   -- Procediments per a la Taula de Variables
204   PROCEDURE Posa
205     (Tv : IN OUT T_Vars;
206      Iv : IN Info_Var;
207      Idv : OUT Num_Var) IS
208   BEGIN
209      Tv.Nv := Tv.Nv+1;
210      Tv.Tv(Tv.Nv) := Iv;
211      Idv := Tv.Nv;
212   END Posa;
213
```

```
214
215     FUNCTION Nova_Etiq RETURN Num_Etiq IS
216     BEGIN
217        Ne := Ne+1;
218        RETURN Ne;
219     END Nova_Etiq;
220
221 END Semantica;
```

3.4 Comprovació de tipus

3.4.1 Fitxer *decls-dtnode.ads*

```
1  -- DECLS-DTNODE.ads
2  -- Declaracions del node
3
4  WITH Decls.Dgenerals,
5    Decls.D_Taula_De_Noms,
6    Decls.Dtdesc;
7
8  USE Decls.Dgenerals,
9    Decls.D_Taula_De_Noms,
10   Decls.Dtdesc;
11
12 PACKAGE Decls.Dtnode IS
13
14    --pragma pure;
15
16    TYPE Mmode IS
17      (Entra,
18       Surt,
19       Entrasurt);
20
21    TYPE Operacio IS
22      (Suma,
23       Resta,
24       Mult,
25       Div,
26       Menor,
27       Menorig,
28       Major,
29       Majorig,
30       Igual,
31       Distint,
32       Modul,
33       Unio,
34       Interseccio,
35       Negacio);
36
37
38    TYPE Node;
39
40    TYPE Pnode IS ACCESS Node;
```

```
41
42    TYPE Tipusnode IS
43      (Programa,
44       Repeticio,
45       CondicionalS,
46       CondicionalC,
47       Expressio,
48       ExpressioUnaria,
49       Pencap,
50       Procediment,
51       Dvariable,
52       Dconstant,
53       Dcoleccio,
54       Dregistre,
55       Dencapregistre,
56       Dsubrang,
57       Identificador,
58       Const,
59       Declaracions,
60       Bloc,
61       Assignacio,
62       Referencia,
63       Pri,
64       Param,
65       Pcoleccio,
66       Pdimcoleccio,
67       Declmultvar,
68       Tnul,
69       Mode,
70       Encappri,
71       Firecord,
72       Fireferencia);
73
74    TYPE node (Tipus : Tipusnode := tnul) IS RECORD
75       CASE Tipus IS
76          WHEN tnul  => NULL;
77
78          WHEN Programa =>
79             Proc : Pnode;
80
81          WHEN repeticio | condicionalS
82            | declaracions | bloc | assignacio | pri
83            | dcoleccio | Pdimcoleccio | Referencia
```

```
84              | pcoleccio | dvariable
85              | Declmultvar | encappri | Pencap =>
86               Fe1, Fd1 : Pnode;
87
88           WHEN CondicionalC | dconstant | dregistre
89             | Dencapregistre | Param => fe2, fc2, fd2: pnode;
90
91           WHEN expressio => fe3, fd3: pnode;
92                             op3: operacio;
93
94           WHEN ExpressioUnaria => f4: pnode;
95                                   op4: operacio;
96
97           WHEN Procediment | dsubrang =>
98              fe5, fc5, fd5, fid5: pnode;
99
100          WHEN identificador => id12 : Id_Nom;
101                                l1, c1 : natural;
102
103          WHEN Firecord | Fireferencia => f6 : pnode;
104
105          WHEN const => val : valor;
106             l2, c2 : natural;
107             Tconst : Tipus_Atribut;
108
109          WHEN Mode => M12 : Mmode;
110
111       END CASE;
112    END RECORD;
113
114 END Decls.Dtnode;
```

3.4.2 Fitxer *decls-d_ arbre.adb*

```
1  PACKAGE BODY Decls.D_Arbre IS
2
3     PROCEDURE Abuit
4       (P : OUT Pnode) IS
5     BEGIN
6         P := NULL;
7     END Abuit;
8
9
10    PROCEDURE Creanode_Programa
11      (P : OUT Atribut;
12       A : IN Atribut) IS
13    BEGIN
14       P := A;
15       Arbre := P.A;
16    END Creanode_Programa;
17
18
19    PROCEDURE Creanode
20      (P : OUT Atribut;
21       Fe,Fd : IN Atribut;
22       Tn : IN Tipusnode) IS
23       Paux : Pnode;
24    BEGIN
25       Paux := NEW Node(Tn);
26       Paux.Fe1 := Fe.A;
27       Paux.Fd1 := Fd.A;
28       P := (Nodearbre, 0, 0, Paux);
29    END Creanode;
30
31
32    PROCEDURE Creanode
33      (P : OUT Atribut;
34       Fe,Fc,Fd : IN Atribut;
35       Tn : IN Tipusnode) IS
36         Paux : Pnode;
37    BEGIN
38       Paux := NEW Node(Tn);
39       Paux.Fe2 := Fe.A;
40       Paux.Fd2 := Fd.A;
41       Paux.Fc2 := Fc.A;
```

```
42        P := (Nodearbre, 0, 0, Paux);
43    END Creanode;
44
45
46    PROCEDURE Creanode
47      (P : OUT Atribut;
48      Fe, Fd : IN Atribut;
49      Op : IN Operacio;
50      Tn : IN Tipusnode) IS
51        Paux : Pnode;
52    BEGIN
53        Paux := NEW Node(Tn);
54        Paux.Fe3 := Fe.A;
55        Paux.Fd3 := Fd.A;
56        Paux.Op3 := Op;
57        P := (Nodearbre, 0, 0, Paux);
58    END Creanode;
59
60
61    PROCEDURE Creanode
62      (P : OUT Atribut;
63       F : IN Atribut;
64      Op : IN Operacio;
65      Tn : IN Tipusnode) IS
66        Paux : Pnode;
67    BEGIN
68        Paux := NEW Node(Tn);
69        Paux.F4 := F.A;
70        Paux.Op4 := Op;
71        P := (Nodearbre, 0, 0, Paux);
72    END Creanode;
73
74
75    PROCEDURE Creanode
76       (P : OUT Atribut;
77        Fe, Fce, Fc, Fd : IN Atribut;
78        Tn : IN Tipusnode) IS
79         Paux : Pnode;
80     BEGIN
81        Paux := NEW Node(Tn);
82        Paux.Fe5 := Fe.A;
83        Paux.Fc5 := Fce.A;
84        Paux.Fd5 := Fc.A;
```

```
85          Paux.Fid5 := Fd.A;
86          P := (Nodearbre, 0, 0, Paux);
87      END Creanode;
88
89
90      PROCEDURE Creanode
91       (P : OUT Atribut;
92        F : IN Atribut;
93        Tn : IN Tipusnode) IS
94          Paux : Pnode;
95      BEGIN
96          Paux := NEW Node(Tn);
97          Paux.F6 := F.A;
98          P := (Nodearbre, 0, 0, Paux);
99      END Creanode;
100
101
102     PROCEDURE Creanode_Id
103       (P : OUT Atribut;
104       Id : IN Atribut;
105       Tn : IN Tipusnode) IS
106        Paux : Pnode;
107     BEGIN
108        Paux := NEW Node(Tn);
109        Paux.Id12 := Id.Idn;
110        Paux.L1 := Id.Lin;
111        Paux.C1 := Id.Col;
112        P := (Nodearbre, 0, 0, Paux);
113     END Creanode_Id;
114
115
116     PROCEDURE Creanode_Val
117       (P : OUT Atribut;
118        A : IN Atribut;
119       Tn : IN Tipusnode;
120        S : IN Valor) IS
121        Paux : Pnode;
122     BEGIN
123        Paux := NEW Node(Tn);
124        IF S = 0 THEN
125           Paux.Val := A.Val*(-1);
126        ELSE
127           Paux.Val := A.Val;
```

```
128         END IF;
129         Paux.Tconst := A.T;
130         Paux.L2 := A.Lin;
131         Paux.C2 := A.Col;
132         P := (Nodearbre, 0, 0, Paux);
133     END Creanode_Val;
134
135
136     PROCEDURE Creanode_Mode
137       (P : OUT Atribut;
138        M : IN Mmode;
139        Tn : IN Tipusnode) IS
140        Paux : Pnode;
141     BEGIN
142        Paux := NEW Node(Tn);
143        Paux.M12 := M;
144        P := (Nodearbre, 0, 0, Paux);
145     END Creanode_Mode;
146
147     PROCEDURE Creanode
148       (P : OUT Atribut;
149       Tn : IN Tipusnode) IS
150       Paux : Pnode;
151     BEGIN
152       Paux := NEW Node(Tn);
153       P := (Nodearbre, 0, 0, Paux);
154     END Creanode;
155
156
157     PROCEDURE Remunta
158       (P : OUT Atribut;
159        A : IN Atribut) IS
160     BEGIN
161        P := A;
162     END Remunta;
163
164
165     PROCEDURE Cons_Tnode
166       (P : IN Pnode;
167       Tn : OUT Tipusnode) IS
168     BEGIN
169        Tn := P.Tipus;
170     END Cons_Tnode;
```

```
171
172 END Decls.D_Arbre;
```

3.4.3 Fitxer *semantica-ctipus.ads*

```
1  WITH Ada.Text_Io,
2    Decls.Dgenerals,
3    Decls.Dtnode,
4    Semantica,
5    Decls.D_Taula_De_Noms,
6    Decls.D_Atribut,
7    Decls.dtsimbols,
8    Decls.Dtdesc,
9    Semantica.Missatges;
10
11 USE Ada.Text_Io,
12   Decls.Dgenerals,
13   Decls.Dtnode,
14   Semantica,
15   Decls.D_Taula_De_Noms,
16   Decls.D_Atribut,
17   Decls.Dtsimbols,
18   Decls.Dtdesc,
19   Semantica.Missatges;
20
21
22 PACKAGE Semantica.Ctipus IS
23
24    -- Rutines lexiques
25    PROCEDURE mt_atom
26      (l, c : IN natural;
27       a : OUT atribut);
28
29    PROCEDURE mt_identificador
30      (l, c : IN natural;
31       s : IN string;
32       a : OUT atribut);
33
34    PROCEDURE mt_string
35      (l, c : IN natural;
36       s : IN string;
37       a : OUT atribut);
38
39    PROCEDURE mt_caracter
40      (l, c : IN natural;
41       car : IN string;
```

```
42        a : OUT atribut);
43
44     PROCEDURE mt_numero
45       (l, c : IN natural;
46        s : IN string;
47        a : OUT atribut);
48
49     -- Comprovacio de tipus
50     PROCEDURE Inicia_analisi(nomFitxer: IN String);
51
52     PROCEDURE Ct_Programa
53       (A : IN Pnode);
54
55  PRIVATE
56
57     PROCEDURE Ct_Decprocediment
58       (A : IN Pnode);
59
60     PROCEDURE Ct_Encap
61       (A : IN Pnode;
62        I : OUT Id_Nom);
63
64     PROCEDURE Ct_Pencap
65       (A : IN Pnode;
66        I : OUT Id_Nom);
67
68     PROCEDURE Ct_Param
69       (A : IN Pnode;
70        I : IN Id_Nom);
71
72     PROCEDURE Ct_Declaracions
73       (A : IN Pnode);
74
75     PROCEDURE Ct_Decvar
76       (A : IN Pnode);
77
78     PROCEDURE Ct_Declsvar
79       (A : IN Pnode;
80        Idtipus : OUT Id_nom);
81
82     PROCEDURE Ct_Decconst
83       (A : IN Pnode);
84
```

```
85    PROCEDURE Ct_Deccol
86      (A : IN Pnode);
87
88    PROCEDURE Ct_Pcoleccio
89      (A : IN Pnode;
90       Idtipus_Array : IN Id_Nom;
91       Idarray : OUT Id_Nom;
92       Ncomponents : OUT Despl);
93
94    PROCEDURE Ct_Decregistre
95      (A : IN Pnode;
96       Idrecord : OUT Id_Nom;
97       Ocup: IN OUT despl);
98
99    PROCEDURE Ct_Dregistre_Camp
100     (Idrecord : IN Id_Nom;
101      Camp : IN Pnode;
102      Tcamp : IN Pnode;
103      Ocup: IN OUT Despl);
104
105   PROCEDURE Ct_Decsubrang
106     (A : IN Pnode);
107
108   PROCEDURE Ct_Expressio
109     (A : IN Pnode;
110      T : OUT Tipussubjacent;
111      Idtipus : OUT Id_Nom;
112      L, C : IN OUT Natural);
113
114   PROCEDURE Ct_Operand_Exp
115     (A : IN Pnode;
116      T : OUT Tipussubjacent;
117      Idtipus : OUT Id_Nom;
118      L, C : IN OUT Natural);
119
120   PROCEDURE Ct_Expressioc
121     (A : IN Pnode;
122      T : OUT Tipussubjacent;
123      Idtipus : OUT Id_Nom;
124      L, C : IN OUT Natural);
125
126   PROCEDURE Ct_Exp_Logica
127     (Tesq, Tdret : IN Tipussubjacent;
```

```
128       Idesq, Iddret : IN Id_Nom;
129       T : OUT Tipussubjacent;
130       Idtipus : OUT Id_Nom;
131       L, C : IN OUT Natural);
132
133    PROCEDURE Ct_Exp_Relacional
134      (Tesq, Tdret : IN Tipussubjacent;
135       Idesq, Iddret : IN Id_Nom;
136       T : OUT Tipussubjacent;
137       Idtipus : OUT Id_Nom;
138       L, C : IN OUT Natural);
139
140    PROCEDURE Ct_Exp_Aritmetica
141      (Tesq, Tdret : IN Tipussubjacent;
142       Idesq, Iddret : IN Id_Nom;
143       T : OUT Tipussubjacent;
144       Idtipus : OUT Id_Nom;
145       L, C : IN OUT Natural);
146
147    PROCEDURE Ct_Expressiou
148      (A : IN Pnode;
149       T : OUT Tipussubjacent;
150       Idtipus : OUT Id_Nom;
151       L, C : IN OUT Natural);
152
153    PROCEDURE Ct_Exp_Negacio
154      (Ts : IN Tipussubjacent;
155       Id : IN Id_Nom;
156       T : OUT Tipussubjacent;
157       Idtipus : OUT Id_Nom;
158       L, C : IN OUT Natural);
159
160    PROCEDURE Ct_Exp_Neglogica
161      (Ts : IN Tipussubjacent;
162       Id : IN Id_Nom;
163       T : OUT Tipussubjacent;
164       Idtipus : OUT Id_Nom;
165       L, C : IN OUT Natural);
166
167    PROCEDURE Ct_Constant
168      (A : IN Pnode;
169       T : OUT Tipussubjacent;
170       Idtipus : OUT Id_Nom;
```

```
171       L, C : IN OUT Natural);
172
173     PROCEDURE Ct_Identificador
174       (A : IN Pnode;
175        T : OUT Tipussubjacent;
176        Idtipus : OUT Id_Nom;
177        L, C : IN OUT Natural);
178
179     PROCEDURE Ct_Bloc
180       (A : IN Pnode);
181
182     PROCEDURE Ct_Srep
183       (A : IN Pnode);
184
185     PROCEDURE Ct_Sconds
186       (A : IN Pnode);
187
188     PROCEDURE Ct_Scondc
189       (A : IN Pnode);
190
191     PROCEDURE Ct_Referencia_Proc
192       (A : IN Pnode;
193        T : OUT Tipussubjacent;
194        Id : OUT Id_Nom);
195
196     PROCEDURE Ct_Referencia_Var
197       (A : IN Pnode;
198        T : OUT Tipussubjacent;
199        Id : OUT Id_Nom);
200
201     PROCEDURE Ct_Ref_Rec
202       (A : IN Pnode;
203        T : OUT Tipussubjacent;
204        Idtipus : OUT Id_Nom;
205        Idbase : OUT Id_Nom);
206
207     PROCEDURE Ct_Ref_Pri
208       (A : IN Pnode;
209        T : OUT Tipussubjacent;
210        Id : OUT Id_Nom;
211        It_Idx : OUT Cursor_Idx);
212
213     PROCEDURE Ct_Ref_Pri
```

```
214       (A : IN Pnode;
215        T : OUT Tipussubjacent;
216        It_Arg : OUT Cursor_Arg);
217
218
219 END Semantica.Ctipus;
```

3.4.4 Fitxer *semantica-ctipus.adb*

```
1  WITH U_Lexica;
2
3  USE U_Lexica;
4
5  PACKAGE BODY Semantica.Ctipus IS
6
7     PROCEDURE Mt_Atom
8       (L, C : IN Natural;
9        A : OUT Atribut) IS
10    BEGIN
11       A := (Atom, L, C);
12    END Mt_Atom;
13
14
15    PROCEDURE Mt_Identificador
16      (L, C : IN Natural;
17       S : IN String;
18       A : OUT Atribut) IS
19       Id : Id_Nom;
20    BEGIN
21       Id := Id_Nul;
22       Posa_Id(Tn, Id, S);
23       A := (A_Ident, L, C, Id);
24    END Mt_Identificador;
25
26
27    PROCEDURE Mt_String
28      (L, C : IN Natural;
29       S : IN String;
30       A : OUT Atribut) IS
31       Id : Rang_Tcar;
32    BEGIN
33       Posa_Str(Tn, Id, S);
34       A := (A_Lit_S, L, C, Valor(Id));
35    END Mt_String;
36
37
38    PROCEDURE Mt_Caracter
39      (L, C : IN Natural;
40       Car : IN String;
41       A : OUT Atribut) IS
```

```
42    BEGIN
43       A := (A_Lit_C, L, C, Valor(Character'Pos(Car(Car'First+1))));
44    END Mt_Caracter;
45
46
47    PROCEDURE Mt_Numero
48      (L, C : IN Natural;
49       S : IN String;
50       A : OUT Atribut) IS
51    BEGIN
52       A := (A_Lit_N, L, C, Valor(Integer'Value(S)));
53    END Mt_Numero;
54
55
56    -- Taula de simbols
57    PROCEDURE Inicia_Enter IS
58       D : Descrip;
59       Dt : Descriptipus;
60       Idn, Ida, Idint : Id_Nom;
61       E : Boolean;
62
63       Ipr : Info_Proc;
64       Idpr : Num_Proc;
65       Iv : Info_Var;
66       Idv : Num_Var;
67
68    BEGIN
69       -- Constants inicials
70       Posa_Id(Tn, Idn, "_zero");
71       Iv := (Idn, Tp.Np, Integer'Size/8, 0, Tsent,
72              False, True, 0);
73       Nv := Nv + 1;
74       Posa(Tv, Iv, Zero);
75
76       Posa_Id(Tn, Idn, "_menysu");
77       Iv := (Idn, Tp.Np, Integer'Size/8, 0, Tsent,
78              False, True, -1);
79       Posa(Tv, Iv, Menysu);
80       Nv := Nv + 1;
81
82       -- "Integer"
83       Posa_Id(Tn, Idint, "integer");
84       Dt := (Tsent, Integer'Size/8, Valor(Integer'First),
```

```
85              Valor(Integer'Last));
86        D := (Dtipus, Dt);
87        Posa(Ts, Idint, D, E);
88
89        -- "puti"
90        Posa_Id(Tn, Idn, "puti");
91        Ipr := (Extern, 4, Idn);
92        Posa(Tp, Ipr, Idpr);
93        Np := Np + 1;
94        D := (Dproc, Idpr);
95        Posa(Ts, Idn, D, E);
96
97        Posa_Id(Tn, Ida, "_arg_puti");
98        Iv := (Ida, Idpr, Integer'Size/8, Ipr.Ocup_Param,
99              Tsent, True, False, 0);
100       Posa(Tv, Iv, Idv);
101       Nv := Nv + 1;
102       D := (Dargc, Idv, Idint);
103       Posa(Ts, Ida, D, E);
104       Posa_Arg(Ts, Idn, Ida, D, E);
105       Ipr.Ocup_Param := Ipr.Ocup_Param + Iv.Ocup;
106
107       -- "geti"
108       Posa_Id(Tn, Idn, "geti");
109       Ipr := (Extern, 4, Idn);
110       Posa(Tp, Ipr, Idpr);
111       Np := Np + 1;
112       D := (Dproc, Idpr);
113       Posa(Ts, Idn, D, E);
114
115       Posa_Id(Tn, Ida, "_arg_geti");
116       Iv := (Ida, Idpr, Integer'Size/8, Ipr.Ocup_Param, Tsent,
117             True, False, 0);
118       Posa(Tv, Iv, Idv);
119       Nv := Nv + 1;
120       D := (Dargc, Idv, Idint);
121       Posa(Ts, Ida, D, E);
122       Posa_Arg(Ts, Idn, Ida, D, E);
123       Ipr.Ocup_Param := Ipr.Ocup_Param + Iv.Ocup;
124
125   END Inicia_Enter;
126
127
```

```
128    PROCEDURE Inicia_Boolea IS
129       D : Descrip;
130       Dt : Descriptipus;
131       Idb, Idt, Idf : Id_Nom;
132       E : Boolean;
133
134       Iv : Info_Var;
135       Idv : Num_Var;
136    BEGIN
137       Posa_Id(Tn, Idb, "boolean");
138       Dt := (Tsbool, Integer'Size/8, -1, 0);
139       D := (Dtipus, Dt);
140       Posa(Ts, Idb, D, E);
141
142       Posa_Id(Tn, Idt, "true");
143       Iv := (Idt, 0, Integer'Size/8, 0, Tsbool, False,
144              True, -1);
145       Posa(Tv, Iv, Idv);
146       Nv := Nv+1;
147       D := (Dconst, Idb, -1, Nv);
148       Posa(Ts, Idt, D, E);
149
150       Posa_Id(Tn, Idf, "false");
151       Iv.Id := Idf;
152       Iv.Valconst := 0;
153       Posa(Tv, Iv, Idv);
154       Nv := Nv+1;
155       D := (Dconst, Idb, 0, Nv);
156       Posa(Ts, Idf, D, E);
157    END Inicia_Boolea;
158
159
160    PROCEDURE Inicia_Caracter IS
161       D : Descrip;
162       Dt : Descriptipus;
163       Idn, Idstring, Ida, Idchar : Id_Nom;
164       E : Boolean;
165       Ipr : Info_Proc;
166       --       Ide : Num_Etiq;
167       Idpr : Num_Proc;
168       --       Ie : Info_Etiq;
169       Iv : Info_Var;
170       Idv : Num_Var;
```

```
171     BEGIN
172        -- "character"
173        Posa_Id(Tn, Idchar, "character");
174        Dt := (Tscar, 4, Valor(Character'Pos(Character'First)),
175               Valor(Character'Pos(Character'Last)));
176        D := (Dtipus, Dt);
177        Posa(Ts, Idchar, D, E);
178
179        -- "string"
180        Posa_Id(Tn, Idstring, "string");
181        Dt := (Tsstr, 4, Idchar, 0); --0 es la base
182        D := (Dtipus, Dt);
183        Posa(Ts, Idstring, D, E);
184
185        -- putc
186        Posa_Id(Tn, Idn, "putc");
187        Ipr := (Extern, 4, Idn);
188        Posa(Tp, Ipr, Idpr);
189        Np := Np + 1;
190
191        D := (Dproc, Idpr);
192        Posa(Ts, Idn, D, E);
193
194        Posa_Id(Tn, Ida, "_arg_putc");
195        Iv := (Ida, Idpr, Integer'Size/8, Ipr.Ocup_Param, Tscar,
196               True, False, 0);
197        Posa(Tv, Iv, Idv);
198        nv:= nv +1;
199        D := (Dargc, Idv, Idchar);
200        Posa(Ts, Ida, D, E);
201        Posa_Arg(Ts, Idn, Ida, D, E);
202        Ipr.Ocup_Param := Ipr.Ocup_Param + Iv.Ocup;
203
204        -- getc
205        Posa_Id(Tn, Idn, "getc");
206        Ipr := (Extern, 4, Idn);
207        Posa(Tp, Ipr, Idpr);
208        Np := Np + 1;
209
210        D := (Dproc, Idpr);
211        Posa(Ts, Idn, D, E);
212
213        Posa_Id(Tn, Ida, "_arg_getc");
```

```
214         Iv := (Ida, Idpr, Integer'Size/8, Ipr.Ocup_Param, Tscar,
215                True, False, 0);
216         Posa(Tv, Iv, Idv);
217         nv:= nv +1;
218         D := (Dargc, Idv, Idchar);
219         Posa(Ts, Ida, D, E);
220         Posa_Arg(Ts, Idn, Ida, D, E);
221         Ipr.Ocup_Param := Ipr.Ocup_Param + Iv.Ocup;
222
223         -- getcc
224         Posa_Id(Tn, Idn, "getcc");
225         Ipr := (Extern, 4, Idn);
226         Posa(Tp, Ipr, Idpr);
227         Np := Np + 1;
228         D := (Dproc, Idpr);
229         Posa(Ts, Idn, D, E);
230
231         Posa_Id(Tn, Ida, "_arg_getcc");
232         Iv := (Ida, Idpr, 1, Ipr.Ocup_Param, Tscar,
233                True, False, 0);
234         Posa(Tv, Iv, Idv);
235         nv:= nv +1;
236         D := (Dargc, Idv, Idchar);
237         Posa(Ts, Ida, D, E);
238         Posa_Arg(Ts, Idn, Ida, D, E);
239
240         -- puts
241         Posa_Id(Tn, Idn, "puts");
242         Ipr := (Extern, 4, Idn);
243         Posa(Tp, Ipr, Idpr);
244         Np := Np + 1;
245         Id_Puts := Idpr;
246
247         D := (Dproc, Idpr);
248         Posa(Ts, Idn, D, E);
249
250         --arg_puts
251         Posa_Id(Tn, Ida, "_arg_puts");
252         Iv := (Ida, Idpr, 4, Ipr.Ocup_Param, Tsstr,
253                True, False, 0);--16 * Integer'Size
254         Posa(Tv, Iv, Idv);
255         nv:= nv +1;
256         D := (Dargc, Idv, Idstring);
```

```
257         Posa(Ts, Ida, D, E);
258         Posa_Arg(Ts, Idn, Ida, D, E);
259         Ipr.Ocup_Param := Ipr.Ocup_Param + Iv.Ocup;
260
261         -- gets
262         Posa_Id(Tn, Idn, "gets");
263         Ipr := (Extern, 4, Idn);
264         Posa(Tp, Ipr, Idpr);
265         Np := Np + 1;
266         Id_Gets := Idpr;
267
268         D := (Dproc, Idpr);
269         Posa(Ts, Idn, D, E);
270         --arg_gets
271         Posa_Id(Tn, Ida, "_arg_gets");
272         Iv := (Ida, Idpr, 4, Ipr.Ocup_Param, Tsstr,
273                True, False, 0);
274         Posa(Tv, Iv, Idv);
275         Nv := Nv +1;
276         D := (Dargc, Idv, Idstring);
277         Posa(Ts, Ida, D, E);
278         Posa_Arg(Ts, Idn, Ida, D, E);
279         Ipr.Ocup_Param := Ipr.Ocup_Param + Iv.Ocup;
280
281         -- nova linea
282         Posa_Id(Tn, Idn, "new_line");
283         Ipr := (Extern, 0, Idn);
284         Posa(Tp, Ipr, Idpr);
285         Np := Np + 1;
286         D := (Dproc, Idpr);
287         Posa(Ts, Idn, D, E);
288     END Inicia_Caracter;
289
290
291     PROCEDURE Inicia_Analisi
292       (Nomfitxer : IN String) IS
293     BEGIN
294         Nv := 0;
295         Np := 0;
296         Tbuida(Tn);
297         Tbuida(Ts);
298         -- Iniciam les Taules
299         Noves_Taules(Tp, Tv);
```

```
300        Inicia_Enter;
301        Inicia_Boolea;
302        Inicia_Caracter;
303        Obre_Fitxer(nomFitxer);
304     END Inicia_analisi;
305
306     -- Procediments interns
307     PROCEDURE Posa_Idvar
308       (Idvar : IN Id_Nom;
309        Idtipus : IN Id_Nom;
310        L, C : IN Natural;
311        E : OUT Boolean) IS
312        Tassig : Descrip;
313     BEGIN
314        Nv := Nv + 1;
315        Tassig := (Dvar, Idtipus, Nv);
316        Posa(Ts, Idvar, Tassig, E);
317        IF E THEN
318           Error(Id_Existent, L, C, Cons_Nom(Tn, Idvar));
319           Esem := True;
320        END IF;
321     END Posa_Idvar;
322
323
324     -- Comprovacio de tipus
325     PROCEDURE Ct_Programa
326       (A : IN Pnode) IS
327        D : Descrip;
328        Idproc : Id_nom RENAMES A.Fid5.Id12;
329        Ida : Cursor_Arg;
330     BEGIN
331        Ct_Decprocediment(A);
332        Ida := Primer_Arg(Ts, Idproc);
333        IF (Arg_Valid(Ida)) THEN
334           Error(Paramspprincipal, Cons_Nom(Tn, Idproc));
335           Esem := True;
336        END IF;
337        Tts(Proc_Nul) := Ts;
338        Tanca_Fitxer;
339
340     END Ct_Programa;
341
342
```

```
343    PROCEDURE Ct_Decprocediment
344      (A : IN Pnode) IS
345
346       Encap : Pnode RENAMES A.Fe5;
347       Decls : Pnode RENAMES A.Fc5;
348       Bloc : Pnode RENAMES A.Fd5;
349       Id : Pnode RENAMES A.Fid5;
350       Id_Inf : Id_Nom RENAMES A.Fid5.Id12;
351       Id_Sup : Id_Nom;
352       Tdecls : Tipusnode;
353       np_propi : num_proc;
354
355    BEGIN
356       Ct_Encap(Encap, Id_Sup);
357       Np_Propi := Np;
358       IF Id_Inf /= Id_Sup THEN
359          Error(Idprogdiferents, A.Fid5.l1, A.Fid5.c1,
360                Cons_Nom(Tn, Id_Sup));
361          Esem := True;
362       END IF;
363
364       Cons_Tnode(Decls, Tdecls);
365       IF Tdecls /= Tnul THEN
366          Ct_Declaracions(Decls);
367       END IF;
368       Ct_Bloc(Bloc);
369       Tts(Np_Propi) := Ts;
370
371       Surtbloc(Ts,tn);
372
373    END Ct_Decprocediment;
374
375
376    PROCEDURE Ct_Encap
377      (A : IN Pnode;
378       I : OUT Id_Nom) IS
379
380       Tproc : Descrip;
381       E : Boolean;
382       Idx_Arg : Cursor_Arg;
383       Ida : Id_Nom;
384       Dn : Descrip;
385
```

```
386    BEGIN
387
388       IF A.Tipus = Pencap THEN
389          Ct_Pencap(A, I);
390          Idx_Arg := Primer_Arg(Ts, I);
391          WHILE Arg_Valid(Idx_Arg) LOOP
392             Cons_Arg(Ts, Idx_Arg, Ida, Dn);
393             Posa(Ts, Ida, Dn, E);
394             IF E THEN
395                Error(Enregarg, 3, 3, Cons_Nom(Tn, Ida));
396                Esem := True;
397             END IF;
398             Idx_Arg := Succ_Arg(Ts, Idx_Arg);
399          END LOOP;
400
401       ELSE
402          I := A.Id12;
403          Np := Np + 1;
404          Tproc := (Dproc, Np);
405          Posa(Ts, I, Tproc, E);
406          Entrabloc(Ts);
407          IF E THEN
408             Error(Id_Existent, A.l1, A.C1, Cons_Nom(Tn, I));
409             Esem := True;
410          END IF;
411
412       END IF;
413
414    END Ct_Encap;
415
416
417    PROCEDURE Ct_Pencap
418      (A : IN Pnode;
419       I : OUT Id_Nom) IS
420
421       Param : Pnode RENAMES A.Fd1;
422       Fesq : Pnode RENAMES A.Fe1;
423       Tproc : Descrip;
424       E : Boolean;
425
426    BEGIN
427
428       IF Fesq.Tipus = Identificador THEN
```

```
429          Np := Np + 1;
430          Tproc := (Dproc, Np);
431          Posa(Ts, Fesq.Id12, Tproc, E);
432          IF E THEN
433             Error(Id_Existent, Fesq.L1, Fesq.C1,
434                   Cons_Nom(Tn, Fesq.Id12));
435             Esem := True;
436          END IF;
437          Entrabloc(Ts);
438          I := Fesq.Id12;
439       ELSE
440          Ct_Pencap(Fesq, I);
441
442       END IF;
443       Ct_Param(Param, I);
444    END Ct_Pencap;
445
446
447    PROCEDURE Ct_Param
448      (A : IN Pnode;
449       I : IN Id_Nom) IS
450
451       Idpar : Id_Nom RENAMES A.Fe2.id12;
452       Marg : Mmode RENAMES A.Fc2.M12;
453       Idtipus : Id_Nom RENAMES A.Fd2.id12;
454       D : Descrip;
455       Darg : Descrip;
456       E : Boolean;
457
458    BEGIN
459       D := Cons(Ts, Idtipus);
460       IF D.Td /= Dtipus THEN
461          Error(Tipusparam, A.Fd2.l1, A.Fd2.c1,
462                Cons_Nom(Tn, Idtipus));
463          Esem := True;
464       END IF;
465
466       CASE Marg IS
467          WHEN Surt | Entrasurt =>
468             Nv := Nv + 1;
469             Darg := (Dvar, Idtipus, Nv);
470          WHEN Entra =>
471             Nv := Nv + 1;
```

```
472             Darg := (Dargc, Nv, Idtipus);
473          WHEN OTHERS =>
474             NULL;
475       END CASE;
476
477       Posa_Arg(Ts, I, Idpar, Darg, E);
478       IF E THEN
479          Error(Enregarg, A.Fe2.l1, A.Fe2.c1,
480                Cons_Nom(Tn, IdPar));
481          Esem := True;
482       END IF;
483
484    END Ct_Param;
485
486
487    PROCEDURE Ct_Declaracions
488      (A : IN Pnode) IS
489
490       Decl : Pnode RENAMES A.Fd1;
491       Decls : Pnode RENAMES A.Fe1;
492       Tnode : Tipusnode;
493       Idrec : Id_Nom;
494       Ocup  : Despl;
495
496    BEGIN
497
498       IF Decls.Tipus = Declaracions THEN
499          Ct_Declaracions(Decls);
500       END IF;
501
502       Cons_Tnode(Decl, Tnode);
503       CASE Tnode IS
504          WHEN Dvariable   =>
505             Ct_Decvar(Decl);
506          WHEN Dconstant   =>
507             Ct_Decconst(Decl);
508          WHEN Dcoleccio   =>
509             Ct_Deccol(Decl);
510          WHEN Dregistre | Dencapregistre | Firecord =>
511             Ocup := 0;
512             Ct_Decregistre(Decl, Idrec,Ocup);
513          WHEN Dsubrang    =>
514             Ct_Decsubrang(Decl);
```

```
515          WHEN Procediment =>
516             Ct_Decprocediment(Decl);
517          WHEN OTHERS =>
518             Esem := True;
519             NULL;
520       END CASE;
521
522    END Ct_Declaracions;
523
524
525    PROCEDURE Ct_Decvar
526      (A : IN Pnode) IS
527
528       Dvariable : Pnode RENAMES A.Fd1;
529       Id : Id_Nom RENAMES A.Fe1.Id12;
530       L : Natural RENAMES A.Fe1.L1;
531       C : Natural  RENAMES A.Fe1.C1;
532       Tassig : Descrip;
533       Idtipus : Id_nom;
534       E : Boolean;
535
536    BEGIN
537       Ct_Declsvar(Dvariable, Idtipus);
538       Posa_Idvar(Id, Idtipus, L, C, E);
539    END Ct_Decvar;
540
541
542    PROCEDURE Ct_Declsvar
543      (A : IN Pnode;
544       Idtipus : OUT Id_Nom) IS
545
546       Tnode : Tipusnode RENAMES A.Tipus;
547       E : Boolean;
548       Tdecl : Descrip;
549
550    BEGIN
551
552       IF Tnode = Identificador THEN
553          Tdecl := Cons(Ts, A.Id12);
554          IF (Tdecl.Td /= Dtipus) THEN
555             Error(Tipusinexistent, A.L1, A.C1,
556                   Cons_Nom(Tn, A.Id12));
557             Esem := True;
```

```
558            END IF;
559            Idtipus := A.Id12;
560
561         ELSIF Tnode = Declmultvar THEN
562            Ct_Declsvar(A.Fd1, Idtipus);
563            Posa_Idvar(A.Fe1.Id12, Idtipus, A.Fe1.L1,
564                       A.Fe1.C1, E);
565         END IF;
566
567      END Ct_Declsvar;
568
569
570      PROCEDURE Ct_Decconst
571        (A : IN Pnode) IS
572
573         Id : Id_Nom RENAMES A.Fe2.Id12;
574         Idtipus : Id_Nom RENAMES A.Fc2.Id12;
575         Val : Pnode RENAMES A.Fd2;
576         E : Boolean;
577         Tdecl : Descrip;
578         Tconst : Descrip;
579
580         Tsubj : Tipussubjacent;
581         Ids : Id_Nom;
582         L, C : Natural := 0;
583
584      BEGIN
585
586         Tdecl := Cons(Ts, Idtipus);
587         IF (Tdecl.Td /= Dtipus) THEN
588            Error(Tipusinexistent, A.Fc2.L1, A.Fc2.C1,
589                  Cons_Nom(Tn, Idtipus));
590            Esem := True;
591         ELSE
592            Ct_Constant(Val, Tsubj, Ids, L, C);
593            IF (Tsubj /= Tdecl.Dt.Tt) THEN
594               Error(Tipussubdiferents, A.Fc2.L1, A.Fc2.C1,
595                     Cons_Nom(Tn, Idtipus));
596               Esem := True;
597            END IF;
598
599            IF (Tdecl.Dt.Tt > Tsent) THEN
600               Error(Tsub_No_Escalar, A.Fc2.L1, A.Fc2.C1,
```

```
601                  Cons_Nom(Tn, Idtipus));
602            Esem := True;
603         END IF;
604
605         IF (Tsubj = Tsent OR Tsubj = Tsbool OR Tsubj = Tscar) THEN
606            IF (Val.Val < Tdecl.Dt.Linf) OR
607              (Val.Val > Tdecl.Dt.Lsup) THEN
608               Error(Rang_Sobrepassat, A.Fe2.L1, A.Fe2.C1,
609                     Cons_Nom(Tn, Id));
610               Esem := True;
611            END IF;
612         END IF;
613
614         Nv := Nv + 1;
615         Tconst := (Dconst, Idtipus, Val.Val, Nv);
616         Posa(Ts, Id, Tconst, E);
617
618         IF E THEN
619            Error(Id_Existent, A.Fe2.L1, A.Fe2.C1,
620                  Cons_Nom(Tn, Id));
621            Esem := True;
622         END IF;
623      END IF;
624
625   END Ct_Decconst;
626
627
628   PROCEDURE Ct_Deccol
629     (A : IN Pnode) IS
630
631      Darray : Descrip;
632      Dtarray : Descrip;
633      Fesq : Pnode RENAMES A.Fe1;
634      Idtipus_Array : Id_Nom RENAMES A.Fd1.Id12;
635      Idarray : Id_Nom;
636      Ncomponents : Despl;
637
638   BEGIN
639      Dtarray := Cons(Ts, Idtipus_Array);
640      IF (Dtarray.Td /= Dtipus) THEN
641         Error(Tipusinexistent, A.Fd1.L1, A.Fd1.C1,
642               Cons_Nom(Tn, Idtipus_Array));
643         Esem := True;
```

```
644         ELSE
645            Ct_Pcoleccio(Fesq, Idtipus_Array, Idarray,
646                         Ncomponents);
647            Darray := Cons(Ts, Idarray);
648            Darray.Dt.Tcamp := Idtipus_Array;
649            Darray.Dt.Ocup := Ncomponents * Dtarray.Dt.Ocup;
650            Actualitza(Ts, Idarray, Darray);
651         END IF;
652      END Ct_Deccol;
653
654
655      PROCEDURE Ct_Pcoleccio
656        (A : IN Pnode;
657         Idtipus_Array : IN Id_Nom;
658         Idarray : OUT Id_Nom;
659         Ncomponents : OUT Despl) IS
660
661         Fesq : Pnode RENAMES A.Fe1;
662         Idrang : Id_Nom RENAMES A.Fd1.Id12;
663         E : Boolean;
664
665         Dtarray : Descriptipus;
666         Darray : Descrip;
667         Di : Descrip;
668
669      BEGIN
670         IF (A.Tipus = Pcoleccio) THEN
671            Ct_Pcoleccio(Fesq, Idtipus_Array, Idarray,
672                         Ncomponents);
673            Posa_Idx(Ts, Idarray, Idrang, E);
674
675            IF E THEN
676               Error(Posaidxarray, A.Fd1.L1, A.Fd1.C1,
677                     Cons_Nom(Tn, Idrang));
678               Esem := True;
679            ELSE
680               Di := Cons(Ts, Idrang);
681               IF Di.Td = Dtipus THEN
682                  Ncomponents := Ncomponents *
683                    Despl(Di.Dt.Lsup - Di.Dt.Linf + 1);
684               ELSE
685                  Error(Tipusidxerroniarray, A.Fd1.L1,
686                        A.Fd1.C1, Cons_Nom(Tn, Idrang));
```

```
687                Esem := True;
688             END IF;
689          END IF;
690
691       ELSIF (A.Tipus = Pdimcoleccio) THEN
692          Dtarray := (Tsarr, 0, Idtipus_Array, 0);
693          Darray := (Dtipus, Dtarray);
694          Idarray := Fesq.Id12;
695          Posa(Ts, Idarray, Darray, E);
696          IF E THEN
697             Error(Tipusinexistent, Fesq.L1, Fesq.C1,
698                   Cons_Nom(Tn, Idtipus_Array));
699             Esem := True;
700             Ncomponents := 0;
701          END IF;
702
703          Di := Cons(Ts, Idrang);
704          IF NOT (Di.Td = Dtipus AND THEN
705                  Di.Dt.Tt <= Tsent) THEN
706             Error(Tipusidxerroniarray, A.Fd1.L1, A.Fd1.C1,
707                   Cons_Nom(Tn, Idrang));
708             Esem := True;
709             Ncomponents := 0;
710          ELSE
711             Posa_Idx(Ts, Idarray, Idrang, E);
712             IF E THEN
713                Put_Line("ERROR CT-pdimcoleccio (DEBUG): "&
714                         "error al posa_idx, error "&
715                         "del compilador, array no creat,"&
716                         " idarr: "&Idarray'Img);
717                Esem := True;
718             END IF;
719
720             Ncomponents := Despl(Di.Dt.Lsup
721                                  - Di.Dt.Linf + 1);
722          END IF;
723       END IF;
724
725    END Ct_Pcoleccio;
726
727
728    PROCEDURE Ct_Decregistre
729      (A : IN Pnode;
```

```
730       Idrecord : OUT Id_Nom;
731       Ocup: IN OUT despl) IS
732
733       Drecord : Descrip;
734       Dtrecord : Descriptipus;
735       E : Boolean;
736
737    BEGIN
738       IF (A.Tipus = Dregistre) THEN
739          Dtrecord := (Tsrec, 0);
740          Drecord := (Dtipus, Dtrecord);
741          Posa(Ts, A.Fe2.Id12, Drecord, E);
742          Idrecord := A.Fe2.Id12;
743          IF E THEN
744             Error(Id_Existent, A.Fe2.L1, A.Fe2.C1,
745                   Cons_Nom(Tn, Idrecord));
746             Esem := True;
747          END IF;
748          Ct_Dregistre_Camp(A.Fe2.Id12, A.Fc2, A.Fd2, Ocup);
749
750       ELSIF (A.Tipus = Dencapregistre) THEN
751          Ct_Decregistre(A.Fe2, Idrecord, Ocup);
752          Ct_Dregistre_Camp(Idrecord, A.Fc2, A.Fd2, Ocup);
753
754       ELSIF (A.Tipus = Firecord) THEN
755          Ct_Decregistre(A.F6, Idrecord, Ocup);
756          Drecord := Cons(Ts, Idrecord);
757          Drecord.Dt.Ocup := Ocup;
758          Actualitza(Ts, Idrecord, Drecord);
759       END IF;
760
761    END Ct_Decregistre;
762
763
764    PROCEDURE Ct_Dregistre_Camp
765      (Idrecord : IN Id_Nom;
766       Camp : IN Pnode;
767       Tcamp : IN Pnode;
768       Ocup: IN OUT Despl) IS
769
770       Idtcamp : Id_Nom RENAMES Tcamp.Id12;
771       Dtcamp : Descrip;
772       Idcamp : Id_Nom RENAMES Camp.Id12;
```

```
773         Desc_Camp : Descrip;
774         E : Boolean;
775
776      BEGIN
777         Dtcamp := Cons(Ts, Idtcamp);
778         IF (Dtcamp.Td /= Dtipus) THEN
779            Error(Tipusinexistent, Camp.L1, Camp.C1,
780                  Cons_Nom(Tn, Idtcamp));
781            Esem := True;
782         ELSE
783
784            Desc_Camp := (Dcamp, Idtcamp, Ocup);
785            Posacamp(Ts, Idrecord, Idcamp, Desc_Camp, E);
786            Ocup := Ocup + Dtcamp.Dt.Ocup;
787            IF E THEN
788               Error(IdCampRecordExistent, Camp.L1,
789                     Camp.C1, Cons_Nom(Tn, Idcamp));
790               Esem := True;
791            END IF;
792         END IF;
793
794      END Ct_Dregistre_Camp;
795
796
797      PROCEDURE Ct_Decsubrang
798        (A : IN Pnode) IS
799
800         Idsubrang : Id_Nom RENAMES A.Fe5.Id12;
801         Idtsubrang : Id_Nom RENAMES A.Fc5.Id12;
802
803         Rang_Esq : Pnode RENAMES A.Fd5;
804         Rang_Dret : Pnode RENAMES A.Fid5;
805         Tsub : Tipussubjacent;
806
807         Tsesq : Tipussubjacent;
808         Tsdret : Tipussubjacent;
809         Idesq : Id_Nom;
810         Iddret : Id_Nom;
811         Valesq : Valor;
812         Valdret : Valor;
813
814         Tdecl : Descrip;
815         Tdescrip_decl : Descrip;
```

```
816         Tdescript_decl : Descriptipus;
817         L, C : Natural := 0;
818         E : Boolean;
819
820      BEGIN
821         Tdecl := Cons(Ts, Idtsubrang);
822         IF(Tdecl.Td /= Dtipus) THEN
823            Error(TipusInexistent, A.Fc5.L1, A.Fc5.C1,
824                  Cons_Nom(Tn, Idtsubrang));
825            Esem := True;
826         ELSE
827            --Miram el fill esquerra
828            Ct_Constant(Rang_Esq, Tsesq, Idesq, L, C);
829            Valesq := Rang_Esq.Val;
830
831            --Miram el fill dret
832            Ct_Constant(Rang_Dret, Tsdret, Iddret, L, C);
833            Valdret := Rang_Dret.Val;
834
835            -- Comparam els tipus
836            IF (Tsesq /= Tsdret) THEN
837               Error(Tipussubdiferents, A.Fc5.L1, A.Fc5.C1,
838                     ""&Tsesq'Img&"/"&Tsdret'Img);
839               Esem := True;
840            END IF;
841
842            Tsub := Tsesq;
843            IF (Tsub /= Tdecl.Dt.Tt) THEN
844               Error(Tipussubdiferents, A.Fc5.L1, A.Fc5.C1,
845                     ""&Tsub'Img&"/"&Tdecl.Dt.Tt'Img);
846               Esem := True;
847            END IF;
848
849            IF (Valesq > Valdret) THEN
850               Error(ValEsqMajorDret, A.Fc5.L1, A.Fc5.C1,
851                     ""&Valesq'Img&" >"&Valdret'Img);
852               Esem := True;
853            END IF;
854
855            IF (Valesq < Tdecl.Dt.Linf) THEN
856               Error(ValEsqMenor, A.Fc5.L1, A.Fc5.C1,
857                     Cons_Nom(Tn, Idtsubrang));
858               Esem := True;
```

```
859           END IF;
860
861           IF (Valdret > Tdecl.Dt.Lsup) THEN
862              Error(ValDretMajor, A.Fc5.L1, A.Fc5.C1,
863                    Cons_Nom(Tn, Idtsubrang));
864              Esem := True;
865           END IF;
866
867           CASE Tsub IS
868              WHEN Tsent =>
869                 Tdescript_Decl := (Tsent, 4, Valesq,
870                                    Valdret);
871              WHEN Tscar =>
872                 Tdescript_Decl := (Tscar, 4, Valesq,
873                                    Valdret);
874              WHEN OTHERS =>
875                 Put_Line("ERROR Ct_subrang: (Sub)Tipus no "&
876                          "valid per a un subrang");
877                 Esem := True;
878           END CASE;
879
880           Tdescrip_Decl := (Dtipus, Tdescript_Decl);
881           Posa(Ts, Idsubrang, Tdescrip_Decl, E);
882           IF E THEN
883              Error(Id_Existent, A.Fe5.L1, A.Fe5.C1,
884                    Cons_Nom(Tn, Idsubrang));
885              Esem := True;
886           END IF;
887        END IF;
888
889     END Ct_Decsubrang;
890
891
892     PROCEDURE Ct_Expressio
893       (A : IN Pnode;
894        T : OUT Tipussubjacent;
895        Idtipus : OUT Id_Nom;
896        L, C : IN OUT Natural) IS
897
898        Tipus : Tipusnode RENAMES A.Tipus;
899        Tps : Tipussubjacent;
900        Id : Id_Nom;
901
```

```
902    BEGIN
903
904       CASE Tipus IS
905          WHEN Expressio =>
906             Ct_Expressioc(A, Tps, Id, L, C);
907          WHEN ExpressioUnaria =>
908             Ct_Expressiou(A, Tps, Id, L, C);
909          WHEN Identificador =>
910             Ct_Identificador(A, Tps, Id, L, C);
911          WHEN Const =>
912             Ct_Constant(A, Tps, Id, L, C);
913          WHEN Fireferencia | Referencia =>
914             Ct_Referencia_Var(A, Tps, Id);
915          WHEN OTHERS =>
916             Put_Line("ERROR CT-exp: tipus expressio no "&
917                      "trobat :S "&Tipus'Img);
918             Esem := True;
919       END CASE;
920       T := Tps;
921       Idtipus := Id;
922
923    END Ct_Expressio;
924
925
926    PROCEDURE Ct_Operand_Exp
927      (A : IN Pnode;
928       T : OUT Tipussubjacent;
929       Idtipus : OUT Id_Nom;
930       L, C : IN OUT Natural) IS
931
932       Tipus : Tipusnode RENAMES A.Tipus;
933
934    BEGIN
935       CASE Tipus IS
936          WHEN Expressio =>
937             Ct_Expressioc(A, T, Idtipus, L, C);
938          WHEN ExpressioUnaria =>
939             Ct_Expressiou(A, T, Idtipus, L, C);
940          WHEN Referencia | Fireferencia=>
941             Ct_Referencia_var(A, T, IdTipus);
942          WHEN Const =>
943             Ct_Constant(A, T, Idtipus, L, C);
944          WHEN Identificador =>
```

```
945             Ct_Identificador(A, T, Idtipus, L, C);
946
947          WHEN OTHERS =>
948             Esem := True;
949             NULL;
950       END CASE;
951
952    END Ct_Operand_Exp;
953
954
955    PROCEDURE Ct_Expressioc
956      (A : IN Pnode;
957       T : OUT Tipussubjacent;
958       Idtipus : OUT Id_Nom;
959       L, C : IN OUT Natural) IS
960
961       Fesq : Pnode RENAMES A.Fe3;
962       Fdret : Pnode RENAMES A.Fd3;
963       Op : Operacio RENAMES A.Op3;
964
965       Tesq : Tipussubjacent;
966       Idesq : Id_Nom;
967       Tdret : Tipussubjacent;
968       Iddret : Id_Nom;
969
970    BEGIN
971       --Analitzam l'operand esquerra
972       Ct_Operand_Exp(Fesq, Tesq, Idesq, L, C);
973       --Analitzam l'operand dret
974       Ct_Operand_Exp(Fdret, Tdret, Iddret, L, C);
975       -- Comparam els tipus
976       CASE Op IS
977          WHEN Unio | Interseccio =>
978             Ct_Exp_Logica(Tesq, Tdret, Idesq, Iddret, T,
979                           Idtipus, L, C);
980          WHEN Menor | Menorig | Major | Majorig
981            | Igual | Distint =>
982             Ct_Exp_Relacional(Tesq, Tdret, Idesq, Iddret,
983                               T, Idtipus, L, C);
984          WHEN Suma | Resta | Mult | Div | Modul =>
985             Ct_Exp_Aritmetica(Tesq, Tdret, Idesq, Iddret,
986                               T, Idtipus, L, C);
987          WHEN OTHERS =>
```

```
988             Esem := True;
989             NULL;
990       END CASE;
991
992    END Ct_Expressioc;
993
994
995    PROCEDURE Ct_Exp_Logica
996      (Tesq, Tdret : IN Tipussubjacent;
997       Idesq, Iddret : IN Id_Nom;
998       T : OUT Tipussubjacent;
999       Idtipus : OUT Id_Nom;
1000      L, C : IN OUT Natural) IS
1001
1002   BEGIN
1003      IF Tesq /= Tsbool THEN
1004         Error(Tsub_No_Bool, L, C, "esquerra");
1005         Esem := True;
1006      END IF;
1007
1008      IF Tdret /= Tsbool THEN
1009         Error(Tsub_No_Bool, L, C, "dret");
1010         Esem := True;
1011      END IF;
1012
1013      IF Idesq /= Id_Nul AND Iddret /= Id_Nul THEN
1014         IF Idesq /= Iddret THEN
1015            Error(Tops_Diferents, L, C, "");
1016            Esem := True;
1017         END IF;
1018      END IF;
1019
1020      IF Idesq = Id_Nul THEN
1021         Idtipus := Iddret;
1022      ELSE
1023         Idtipus := Idesq;
1024      END IF;
1025
1026      T := Tsbool;
1027
1028   END Ct_Exp_Logica;
1029
1030
```

```
1031    PROCEDURE Ct_Exp_Relacional
1032      (Tesq, Tdret : IN Tipussubjacent;
1033       Idesq, Iddret : IN Id_Nom;
1034       T : OUT Tipussubjacent;
1035       Idtipus : OUT Id_Nom;
1036       L, C : IN OUT Natural) IS
1037
1038    BEGIN
1039       IF Tesq /= Tdret THEN
1040          Error(Tsubs_Diferents, L, C, "");
1041          Esem := True;
1042       END IF;
1043
1044       IF Tesq > Tsent THEN
1045          Error(Tsub_No_Escalar, L, C, "esquerra");
1046          Esem := True;
1047       END IF;
1048
1049       IF Tdret > Tsent THEN
1050          Error(Tsub_No_Escalar, L, C, "dret");
1051          Esem := True;
1052       END IF;
1053
1054       IF Idesq /= Id_Nul AND Iddret /= Id_Nul THEN
1055          IF Idesq /= Iddret THEN
1056             Error(Tops_Diferents, L, C, "");
1057             Esem := True;
1058          END IF;
1059       END IF;
1060
1061       T := Tsbool;
1062       Idtipus := Id_Nul;
1063
1064    END Ct_Exp_Relacional;
1065
1066
1067    PROCEDURE Ct_Exp_Aritmetica
1068      (Tesq, Tdret : IN Tipussubjacent;
1069       Idesq, Iddret : IN Id_Nom;
1070       T : OUT Tipussubjacent;
1071       Idtipus : OUT Id_Nom;
1072       L, C : IN OUT Natural) IS
1073
```

```
1074    BEGIN
1075       IF Tesq /= Tsent THEN
1076          Error(Tsub_No_Sencer, L, C, "esquerra");
1077          Esem := True;
1078       END IF;
1079
1080       IF Tdret /= Tsent THEN
1081          Error(Tsub_No_Sencer, L, C, "dret");
1082          Esem := True;
1083       END IF;
1084
1085       IF Idesq /= Id_Nul AND Iddret /= Id_Nul THEN
1086          IF Idesq /= Iddret THEN
1087             Error(Tops_Diferents, L, C, "");
1088             Esem := True;
1089          END IF;
1090       END IF;
1091
1092       T := Tsent;
1093       IF Idesq = Id_Nul THEN
1094          Idtipus := Iddret;
1095       ELSE
1096          Idtipus := Idesq;
1097       END IF;
1098
1099    END Ct_Exp_Aritmetica;
1100
1101
1102    PROCEDURE Ct_Expressiou
1103      (A : IN Pnode;
1104       T : OUT Tipussubjacent;
1105       Idtipus : OUT Id_Nom;
1106       L, C : IN OUT Natural) IS
1107
1108       Fdret : Pnode RENAMES A.F4;
1109       Op : Operacio RENAMES A.Op4;
1110       Tdret : Tipussubjacent;
1111       Iddret : Id_Nom;
1112
1113    BEGIN
1114       Ct_Operand_Exp(Fdret, Tdret, Iddret, L, C);
1115       CASE Op IS
1116          WHEN Resta =>
```

```
1117              Ct_Exp_Negacio(Tdret, Iddret, T, Idtipus,
1118                             L, C);
1119           WHEN Negacio =>
1120              Ct_Exp_Neglogica(Tdret, Iddret, T, Idtipus,
1121                               L, C);
1122           WHEN OTHERS =>
1123              Esem := True;
1124              NULL;
1125       END CASE;
1126    END Ct_Expressiou;
1127
1128
1129    PROCEDURE Ct_Exp_Negacio
1130      (Ts : IN Tipussubjacent;
1131       Id : IN Id_Nom;
1132       T : OUT Tipussubjacent;
1133       Idtipus : OUT Id_Nom;
1134       L, C : IN OUT Natural) IS
1135    BEGIN
1136       IF Ts /= Tsent THEN
1137          Error(Tsub_No_Sencer, L, C, "");
1138          Esem := True;
1139       END IF;
1140       Idtipus := Id;
1141       T := Tsent;
1142    END Ct_Exp_Negacio;
1143
1144
1145    PROCEDURE Ct_Exp_Neglogica
1146      (Ts : IN Tipussubjacent;
1147       Id : IN Id_Nom;
1148       T : OUT Tipussubjacent;
1149       Idtipus : OUT Id_Nom;
1150       L, C: IN OUT Natural) IS
1151    BEGIN
1152       IF Ts /= Tsbool THEN
1153          Error(Tsub_No_Bool, L, C, "");
1154          Esem := True;
1155       END IF;
1156       Idtipus := Id;
1157       T := Tsbool;
1158    END Ct_Exp_Neglogica;
1159
```

```
1160
1161    PROCEDURE Ct_Constant
1162      (A : IN Pnode;
1163       T : OUT Tipussubjacent;
1164       Idtipus : OUT Id_Nom;
1165       L, C : IN OUT Natural) IS
1166
1167       Tatr : Tipus_Atribut RENAMES A.Tconst;
1168       Lin : Natural RENAMES A.L2;
1169       Col : Natural RENAMES A.C2;
1170       D : Descrip;
1171
1172    BEGIN
1173
1174       Idtipus := Id_Nul;
1175       CASE (Tatr) IS
1176          WHEN A_Lit_C =>
1177             T := Tscar;
1178          WHEN A_Lit_N =>
1179             T := Tsent;
1180          WHEN A_Lit_S =>
1181             T := Tsstr;
1182          WHEN OTHERS =>
1183             Put_Line("ERROR CT-constant: tipus constant "&
1184                      "erroni");
1185             Esem := True;
1186       END CASE;
1187       L := Lin;
1188       C := Col;
1189
1190    END Ct_Constant;
1191
1192
1193    PROCEDURE Ct_Identificador
1194      (A : IN Pnode;
1195       T : OUT Tipussubjacent;
1196       Idtipus : OUT Id_Nom;
1197       L, C : IN OUT Natural) IS
1198
1199       Id : Id_Nom RENAMES A.Id12;
1200       D : Descrip;
1201       Desc : Tdescrip RENAMES D.Td;
1202       Lin : Natural RENAMES A.L1;
```

```
1203        Col : Natural RENAMES A.C1;
1204
1205        Carg : Cursor_Arg;
1206
1207     BEGIN
1208
1209        D := Cons(Ts, Id);
1210
1211        CASE Desc IS
1212           WHEN Dvar =>
1213              Idtipus := D.Tr;
1214              D := Cons(Ts, Idtipus);
1215              IF (D.Td = Dtipus) THEN
1216                 T := D.Dt.Tt;
1217              ELSE
1218                 Error(Tipus_No_Desc, L, C, D.Td'Img);
1219                 Esem := True;
1220              END IF;
1221
1222           WHEN Dconst =>
1223              Idtipus := D.Tc;
1224              D := Cons(Ts, Idtipus);
1225              IF (D.Td = Dtipus) THEN
1226                 T := D.Dt.Tt;
1227              ELSE
1228                 Error(Tipus_No_Desc, L, C, D.Td'Img);
1229                 Esem := True;
1230              END IF;
1231
1232           WHEN Dproc =>
1233              Carg := Primer_Arg(Ts, Id);
1234              IF Arg_Valid(Carg) THEN
1235                 T := Tsarr;
1236              ELSE
1237                 T := Tsnul;
1238              END IF;
1239              Idtipus := Id;
1240           WHEN Dargc =>
1241              Idtipus := D.Targ;
1242              D := Cons(Ts, Idtipus);
1243              IF (D.Td = Dtipus) THEN
1244                 T := D.Dt.Tt;
1245              ELSE
```

```
1246                Error(Tipus_No_Desc, L, C, D.Td'Img);
1247                Esem := True;
1248             END IF;
1249
1250          WHEN OTHERS =>
1251             Error(Id_No_Reconegut, L, C, Desc'Img);
1252             Esem := True;
1253             Idtipus := Id;
1254             T := tsnul;
1255
1256       END CASE;
1257       L := Lin;
1258       C := Col;
1259
1260
1261
1262    END Ct_Identificador;
1263
1264
1265    PROCEDURE Ct_Bloc
1266      (A : IN Pnode) IS
1267
1268       D : Descrip;
1269       T : Tipussubjacent;
1270       Idbase : Id_Nom;
1271       Idtipus : Id_Nom;
1272
1273       Tsexp : Tipussubjacent;
1274       Idexp : Id_Nom;
1275       Tsvar : Tipussubjacent;
1276       Idvar : Id_Nom;
1277       L, C : Natural := 0;
1278
1279    BEGIN
1280       CASE (A.Tipus) IS
1281          WHEN Bloc =>
1282             Ct_Bloc(A.Fe1);
1283             Ct_Bloc(A.Fd1);
1284          WHEN Repeticio =>
1285             Ct_Srep(A);
1286          WHEN Identificador =>
1287
1288             Ct_Identificador(A, T, Idtipus, L, C);
```

```
1289              IF T /= Tsnul THEN
1290                 Error(Id_No_Cridaproc, L, C,
1291                       Cons_Nom(Tn, A.Id12));
1292                 Esem := True;
1293              END IF;
1294
1295           WHEN Fireferencia =>
1296              Ct_Referencia_Proc(A, T, Idbase);
1297           WHEN condicionalS =>
1298              Ct_Sconds(A);
1299           WHEN condicionalC =>
1300              Ct_Scondc(A);
1301           WHEN Assignacio =>
1302              Ct_Referencia_Var(A.Fe1, Tsvar, Idvar);
1303              Ct_Expressio(A.Fd1, Tsexp, Idexp, L, C);
1304              IF Tsvar /= Tsexp THEN
1305                 Error(Assig_Tipus_Diferents, L, C, "");
1306                 Esem := True;
1307              END IF;
1308              IF Idexp /= Id_Nul AND Idexp /= Idvar THEN
1309                 Error(Assig_Tipus_Diferents, L, C, "");
1310                 Esem := True;
1311              END IF;
1312           WHEN OTHERS =>
1313
1314              Esem := True;
1315       END CASE;
1316    END Ct_Bloc;
1317
1318
1319    PROCEDURE Ct_Srep
1320      (A : IN Pnode) IS
1321
1322       Tsexp : Tipussubjacent;
1323       Idtipus_exp : Id_Nom;
1324       Exp : Pnode RENAMES A.Fe1;
1325       Bloc : Pnode RENAMES A.fd1;
1326       L, C : Natural := 0;
1327
1328    BEGIN
1329       Ct_Expressio(Exp, Tsexp, Idtipus_Exp, L, C);
1330       IF tsexp /= tsbool THEN
1331          Error(Exp_No_Bool, L, C, "bucle");
```

```
1332            Esem := True;
1333         END IF;
1334         Ct_Bloc(Bloc);
1335      END Ct_Srep;
1336
1337
1338      PROCEDURE Ct_Sconds
1339        (A : IN Pnode) IS
1340
1341         Tsexp : Tipussubjacent;
1342         Idtipus_exp : Id_Nom;
1343         Cond : Pnode RENAMES A.Fe1;
1344         Bloc : Pnode RENAMES A.fd1;
1345         L, C : Natural := 0;
1346
1347      BEGIN
1348         Ct_Expressio(Cond, Tsexp, Idtipus_Exp, L, C);
1349         IF tsexp /= tsbool THEN
1350            Error(Exp_No_Bool, L, C, "condicional");
1351            Esem := True;
1352         END IF;
1353         Ct_Bloc(Bloc);
1354      END Ct_Sconds;
1355
1356
1357      PROCEDURE Ct_Scondc
1358        (A : IN Pnode) IS
1359
1360         Tsexp : Tipussubjacent;
1361         Idtipus_exp : Id_Nom;
1362         Cond : Pnode RENAMES A.Fe2;
1363         Bloc : Pnode RENAMES A.fc2;
1364         Blocelse : Pnode RENAMES A.fd2;
1365         L, C : Natural := 0;
1366
1367      BEGIN
1368         Ct_Expressio(Cond, Tsexp, Idtipus_Exp, L, C);
1369         IF tsexp /= tsbool THEN
1370            Error(Exp_No_Bool, L, C, "condicional compost");
1371            Esem := True;
1372         END IF;
1373         Ct_Bloc(Bloc);
1374         Ct_Bloc(Blocelse);
```

```
1375    END Ct_Scondc;
1376
1377
1378    PROCEDURE Ct_Referencia_Proc
1379      (A : IN Pnode;
1380       T : OUT Tipussubjacent;
1381       Id : OUT Id_Nom) IS
1382
1383       Tipus : Tipusnode RENAMES A.Tipus;
1384       It_Arg : Cursor_Arg;
1385       L, C : Natural := 0;
1386
1387    BEGIN
1388       CASE Tipus IS
1389          WHEN Identificador =>
1390             Ct_Identificador(A, T, Id, L, C);
1391          WHEN Referencia =>
1392             Error(Rec_No_Cridaproc, L, C, "");
1393             Esem := True;
1394          WHEN Fireferencia =>
1395             Ct_Ref_Pri(A.F6, T, It_Arg);
1396             IF Arg_Valid(It_Arg) THEN
1397                Error(Falta_Param_Proc, L, C, "");
1398                Esem := True;
1399             END IF;
1400          WHEN OTHERS =>
1401             Put_Line("ERROR CT-referencia: node "&
1402                      "no reconegut");
1403             Esem := True;
1404       END CASE;
1405
1406    END Ct_Referencia_Proc;
1407
1408
1409
1410    PROCEDURE Ct_Referencia_Var
1411      (A : IN Pnode;
1412       T : OUT Tipussubjacent;
1413       Id : OUT Id_Nom) IS
1414
1415       Tipus : Tipusnode RENAMES A.Tipus;
1416       Idtipus : Id_Nom;
1417       It_Idx : Cursor_Idx;
```

```
1418        D : Descrip;
1419        L, C : Natural := 0;
1420
1421     BEGIN
1422        CASE Tipus IS
1423           WHEN Identificador =>
1424              Ct_Identificador(A, T, Id, L, C);
1425              D := Cons(Ts, Id);
1426              IF D.Td = Dproc THEN
1427                 Error(Refvar_No_Proc, L, C, "");
1428                 Esem := True;
1429              END IF;
1430           WHEN Referencia =>
1431              Ct_Ref_Rec(A, T, Id, Idtipus);
1432           WHEN Fireferencia =>
1433              Ct_Ref_Pri(A.F6, T, Id, It_Idx);
1434              IF Idx_Valid(It_Idx) THEN
1435                 Error(Falta_Param_Array, L, C, "");
1436                 Esem := True;
1437              END IF;
1438              IF T = Tsarr THEN
1439                 D := Cons(Ts, Id);
1440                 Id := D.Dt.Tcamp;
1441                 D := Cons(Ts, Id);
1442                 T := D.Dt.Tt;
1443              END IF;
1444           WHEN OTHERS =>
1445              Esem := True;
1446              NULL;
1447        END CASE;
1448
1449     END Ct_Referencia_Var;
1450
1451
1452     PROCEDURE Ct_Ref_Rec
1453       (A : IN Pnode;
1454        T : OUT Tipussubjacent;
1455        Idtipus : OUT Id_Nom;
1456        Idbase : OUT Id_Nom) IS
1457
1458        Fesq : Pnode RENAMES A.Fe1;
1459        Tesq : Tipussubjacent;
1460        Idbase_Esq : Id_Nom;
```

```
1461        Dcamp : Descrip;
1462        Dtcamp : Descrip;
1463        Idcamp : Id_Nom RENAMES A.Fd1.Id12;
1464        L, C : Natural := 0;
1465
1466     BEGIN
1467        Ct_Referencia_Var(Fesq, Tesq, Idbase_Esq);
1468        IF Tesq /= Tsrec THEN
1469           Error(Reccamp_No_Valid, L, C, "");
1470           Esem := True;
1471        END IF;
1472
1473        Dcamp := Conscamp(Ts, Idbase_Esq, Idcamp);
1474        IF Dcamp.Td = Dnula THEN
1475           Error(Idrec_No_Valid, L, C, Cons_Nom(Tn, Idcamp));
1476           Esem := True;
1477        END IF;
1478
1479        Idtipus := Dcamp.Tcamp;
1480        Dtcamp := Cons(Ts, Dcamp.Tcamp);
1481        T := Dtcamp.Dt.Tt;
1482        Idbase := Idbase_Esq;
1483
1484     END Ct_Ref_Rec;
1485
1486
1487     PROCEDURE Ct_Ref_Pri
1488       (A : IN Pnode;
1489        T : OUT Tipussubjacent;
1490        Id : OUT Id_Nom;
1491        It_Idx : OUT Cursor_Idx) IS
1492
1493        Tipus : Tipusnode RENAMES A.Tipus;
1494        Fesq : Pnode RENAMES A.Fe1;
1495        Fdret : Pnode RENAMES A.Fd1;
1496        Tsub : Tipussubjacent;
1497        Idvar : Id_Nom;
1498
1499        Tsref : Tipussubjacent;
1500        Idref : Id_Nom;
1501
1502        Id_Cursor : Id_Nom;
1503        Dtipoarg : Descrip;
```

```
1504        Dbase : Descrip;
1505        L, C : Natural := 0;
1506
1507     BEGIN
1508        CASE Tipus IS
1509           WHEN Pri =>
1510              Ct_Ref_Pri(Fesq, T, Id, It_Idx);
1511              Ct_Expressio(Fdret, Tsref, Idref, L, C);
1512              IF NOT Idx_Valid(It_Idx) THEN
1513                 Error(Sobren_Parametres, L, C, "");
1514                 Esem := True;
1515              ELSE
1516                 Id_Cursor := Cons_Idx(Ts, It_Idx);
1517                 Dtipoarg := Cons(Ts, Id_Cursor);
1518                 IF Idref = Id_Nul THEN
1519                    IF Dtipoarg.Dt.Tt /= Tsref THEN
1520                       Error(Tparam_No_Coincident,
1521                             L, C, "");
1522                       Esem := True;
1523                    END IF;
1524                 ELSIF Idref /= Id_cursor THEN
1525                    Error(Tparam_No_Coincident, L, C,
1526                          Cons_Nom(Tn, Idref)&"/"&
1527                            Cons_Nom(Tn, Id_Cursor));
1528                    Esem := True;
1529                 END IF;
1530                 It_Idx := Succ_Idx(Ts, It_Idx);
1531              END IF;
1532
1533           WHEN Encappri =>
1534              Ct_Referencia_Var(Fesq, Tsub, Idvar);
1535              Ct_Expressio(Fdret, Tsref, Idref, L, C);
1536              Dbase := Cons (Ts, Idvar);
1537              IF Tsub = Tsarr THEN
1538                 It_Idx := Primer_Idx(Ts, Idvar);
1539                 IF Idx_Valid(It_Idx) THEN
1540                    Id_Cursor := Cons_Idx(Ts, It_Idx);
1541                    Dtipoarg := Cons(Ts, Id_Cursor);
1542                    IF Idref = Id_Nul THEN
1543                       IF Dtipoarg.Dt.Tt /= Tsref THEN
1544                          Error(Tparam_No_Coincident, L, C,
1545                                "");
1546                          Esem := True;
```

```
1547                        END IF;
1548                     ELSIF Idref /= Id_Cursor THEN
1549                        Error(Tparam_No_Coincident, L, C,
1550                              Cons_Nom(Tn, Idref)&"/"&
1551                                Cons_Nom(Tn, Id_Cursor));
1552                        Esem := True;
1553                     END IF;
1554                  END IF;
1555               ELSE
1556                  Error(Tipus_No_Array, L, C, Tsub'Img);
1557                  Esem := True;
1558               END IF;
1559               It_Idx := Succ_Idx(Ts, It_Idx);
1560               T := Tsub;
1561               Id := Idvar;
1562
1563            WHEN OTHERS =>
1564               Esem := True;
1565               NULL;
1566         END CASE;
1567
1568      END Ct_Ref_Pri;
1569
1570
1571      PROCEDURE Ct_Ref_Pri
1572        (A : IN Pnode;
1573         T : OUT Tipussubjacent;
1574         It_Arg : OUT Cursor_Arg) IS
1575
1576         Tipus : Tipusnode RENAMES A.Tipus;
1577         Fesq : Pnode RENAMES A.Fe1;
1578         Fdret : Pnode RENAMES A.Fd1;
1579         Tsub : Tipussubjacent;
1580         Id : Id_Nom;
1581
1582         Tsref : Tipussubjacent;
1583         Idref : Id_Nom;
1584
1585         Id_Cursor : Id_Nom;
1586         Dparam : Descrip;
1587         Dtipoarg : Descrip;
1588         Dbase : Descrip;
1589         L, C : Natural := 0;
```

```
1590
1591    BEGIN
1592       CASE Tipus IS
1593          WHEN Pri => -- pri , E
1594             Ct_Ref_Pri(Fesq, T, It_Arg);
1595             Ct_Expressio(Fdret, Tsref, Idref, L, C);
1596             IF NOT Arg_Valid(It_Arg) THEN
1597                Error(Sobren_Parametres, L, C, "");
1598                Esem := True;
1599             ELSE
1600                Cons_Arg(Ts, It_Arg, Id_Cursor, Dparam);
1601                IF Idref = Id_Nul THEN
1602                   Dtipoarg := Cons(ts, Dparam.targ);
1603                   IF Dtipoarg.Dt.Tt /= Tsref THEN
1604                      Error(Tparam_No_Coincident, L,
1605                            C, Dtipoarg.Dt.Tt'Img);
1606                      Esem := True;
1607                   END IF;
1608                ELSIF Dparam.td = Dargc THEN
1609                   IF Idref /= Dparam.targ THEN
1610                      Error(Tparam_No_Coincident, L, C,
1611                            Cons_Nom(Tn, Idref)&"/"&
1612                              Cons_Nom(Tn, Id_Cursor));
1613                      Esem := True;
1614                   END IF;
1615                ELSIF Dparam.td = Dvar THEN
1616                   IF Idref /= Dparam.Tr THEN
1617                      Error(Tparam_No_Coincident, L, C,
1618                            Cons_Nom(Tn, Idref)&"/"&
1619                              Cons_Nom(Tn, Id_Cursor));
1620                      Esem := True;
1621                   END IF;
1622                END IF;
1623                It_Arg := Succ_Arg(Ts, It_Arg);
1624             END IF;
1625
1626          WHEN Encappri => -- r(E
1627             Ct_Referencia_Proc(Fesq, Tsub, Id);
1628             Ct_Expressio(Fdret, Tsref, Idref, L, C);
1629             Dbase := Cons (Ts, Id);
1630             IF Tsub = Tsarr AND Dbase.td = Dproc THEN
1631                It_Arg := Primer_Arg(Ts, Id);
1632                IF Arg_Valid(It_Arg) THEN
```

```
1633                    Cons_Arg(Ts, It_Arg, Id_Cursor, Dparam);
1634
1635                    IF Idref = Id_Nul THEN
1636                       IF(Dtipoarg.Td /= Dnula) THEN
1637                          Dtipoarg := Cons(Ts, Dparam.Targ);
1638                          IF Dtipoarg.Dt.Tt /= Tsref THEN
1639                             Error(Tparam_No_Coincident,
1640                                   L, C, "");
1641                             Esem := True;
1642                          END IF;
1643                       END IF;
1644                    ELSIF Dparam.Td = Dargc THEN
1645                       IF Idref /= Dparam.Targ THEN
1646                          Error(Tparam_No_Coincident, L, C,
1647                                Cons_Nom(Tn, Idref)&"/"&
1648                                  Cons_Nom(Tn, Id_Cursor));
1649                          Esem := True;
1650                       END IF;
1651                    ELSIF Dparam.Td = Dvar THEN
1652                       IF Idref /= Dparam.Tr THEN
1653                          Error(Tparam_No_Coincident, L, C,
1654                                Cons_Nom(Tn, Idref)&"/"&
1655                                  Cons_Nom(Tn, Id_Cursor));
1656                          Esem := True;
1657                       END IF;
1658                    END IF;
1659                 END IF;
1660                 It_Arg := Succ_Arg(Ts, It_Arg);
1661              ELSE
1662                 Error(Tproc_No_Param, L, C, Tsub'Img);
1663                 Esem := True;
1664              END IF;
1665
1666              T := Tsub;
1667           WHEN OTHERS =>
1668              Esem := True;
1669        END CASE;
1670     END Ct_Ref_Pri;
1671
1672 END Semantica.Ctipus;
```

3.5 Missatges d'error

3.5.1 Fitxer *semantica-missatges.ads*

```
1  WITH decls.dgenerals,
2    Ada.Text_IO;
3
4  USE decls.dgenerals,
5    Ada.Text_IO;
6
7  PACKAGE Semantica.Missatges IS
8
9     TYPE Terror IS
10      (paramsPprincipal,
11       id_existent,
12       idProgDiferents,
13       tipusParam,
14       paramRepetit,
15       enregArg,
16       tipusInexistent,
17       tipusSubIncorrecte,
18       rang_sobrepassat,
19       idCampRecordExistent,
20       TsubjRangDif,
21       TsubjDifTipus,
22       ValEsqMajorDret,
23       ValEsqMenor,
24       ValDretMajor,
25       TsubNoValid,
26       argNoProc,
27       tipusSubDiferents,
28       posaIdxArray,
29       TipusIdxErroniArray,
30       Tsub_No_Bool,
31       Tops_Diferents,
32       Tsubs_Diferents,
33       Tsub_No_Escalar,
34       Tsub_No_Sencer,
35       Tipus_No_Desc,
36       Id_No_Reconegut,
37       Id_No_Cridaproc,
38       Assig_Tipus_Diferents,
39       Exp_No_Bool,
40       Rec_No_Cridaproc,
```

```
41        Falta_Param_Proc,
42        Refvar_No_Proc,
43        Falta_Param_Array,
44        Reccamp_No_Valid,
45        Idrec_No_Valid,
46        Sobren_Parametres,
47        Tparam_No_Coincident,
48        Tipus_No_Array,
49        Tproc_No_Param);
50
51    PROCEDURE Obre_Fitxer
52      (nomFitxer: IN String);
53
54    PROCEDURE Tanca_Fitxer;
55
56    PROCEDURE Error
57      (Te : IN Terror;
58       L, C : IN Natural;
59       Id : String);
60
61    PROCEDURE Error
62      (Te : IN Terror;
63       Id : String);
64
65    PROCEDURE Impressio
66      (Msj : IN String);
67
68 PRIVATE
69
70    Log_File : File_Type;
71
72 END Semantica.Missatges;
```

3.5.2 Fitxer *semantica-missatges.adb*

```
PACKAGE BODY Semantica.Missatges IS

   PROCEDURE Obre_Fitxer
     (Nomfitxer : IN String) IS
   BEGIN
      Create(Log_File, Out_File, Nomfitxer&".log");
   END Obre_Fitxer;

   PROCEDURE Tanca_Fitxer IS
   BEGIN
      Close(log_file);
   END Tanca_Fitxer;

   PROCEDURE Impressio
     (Msj : IN String) IS
   BEGIN
      put_line(log_file, "ERROR CompiLEMON: " & msj);
      put_line("ERROR CompiLEMON: " & msj);
   END Impressio;

   PROCEDURE Error
     (Te : IN Terror;
      L, C : IN Natural;
      Id : String) IS
   BEGIN
      CASE te IS
         WHEN id_existent =>
            Impressio("l:"&l'img&" c:"&c'img&
                      " L'identificador '"&id&
                      "' ja existeix");
         WHEN idProgDiferents =>
            Impressio("l:"&l'img&" c:"&c'img&
                      " Possible escritura "&
                      "erronea de '"&Id&"'");
         WHEN tipusParam =>
            Impressio("l: "&l'img&" c: "&c'img&
                      " El tipus del parametre "&
                         id&" es incorrecte");
         WHEN enregArg =>
            Impressio("l: "&l'img&" c: "&c'img&
                      " Error al enregistrar"&
```

```
42                      " l'argument");
43          WHEN paramRepetit =>
44             Impressio("l: "&l'img&" c: "&c'img&
45                      " El param "&id&
46                      " es troba repetit");
47          WHEN tipusSubDiferents =>
48             Impressio("l: "&l'img&" c: "&c'img&
49                      " Tipus subjacents "&
50                      "diferents "&id);
51          WHEN tipusInexistent =>
52             Impressio("l: "&l'img&" c: "&c'img&
53                      " El tipus "&id&" no "&
54                      "existeix o no es correcte");
55          WHEN tipusSubIncorrecte =>
56             --Aqui donam prioritat al tipus que
57             --declaram per sobre el tipus
58             -- assignat si son diferents l'erroni
59             --es el que assignam
60             Impressio("l: "&l'img&" c: "&c'img&
61                      " El tipus "&id&
62                      " no es correspon amb el tipus"&
63                      " de la variable");
64          WHEN rang_sobrepassat =>
65             Impressio("l: "&l'img&" c: "&c'img&" El "&
66                      "valor de la constant "&
67                      id&" surt del rang");
68          WHEN idCampRecordExistent =>
69             Impressio("l: "&l'img&" c: "&c'img&
70                      " Ja existeix un camp "&
71                      id&" en aquest record");
72          WHEN TsubjRangDif =>
73             Impressio("l: "&l'img&" c: "&c'img&
74                      "Els Tsubjacents dels "&
75                      "limits del subtipus "&id&
76                      " son diferents");
77          WHEN ValEsqMajorDret =>
78             Impressio("l: "&l'img&" c: "&c'img&
79                      " El valor del limit "&
80                      "Esquerra no pot esser major"&
81                      " que el Dret en "&
82                      "la declaracio del subrang: "&id);
83          WHEN TsubjDifTipus =>
84             Impressio("l: "&l'img&" c: "&c'img&
```

```
85                      " Els Tsubjacents dels "&
86                      "limits del subtipus"&id&
87                      " son diferents al "&
88                      "tipus assignat");
89            WHEN ValEsqMenor =>
90               Impressio("l: "&l'img&" c: "&c'img&
91                      " El valor esquerra es menor"&
92                      "al permes en el subtipus"&id);
93            WHEN ValDretMajor =>
94               Impressio("l: "&l'img&" c: "&c'img&
95                      " El valor dret es major"&
96                      "al permes en el subtipus"&id);
97            WHEN TsubNoValid =>
98               Impressio("l: "&l'img&" c: "&c'img&
99                      " Tipus subjacent no valid"&
100                     " per al subrang"&id);
101           WHEN argNoProc =>
102              Impressio("l: "&l'img&" c: "&c'img&
103                     " L'identificador de "&
104                     "l'argument no es un procediment"
105                     &id);
106           WHEN posaIdxArray =>
107              Impressio("l: "&l'img&" c: "&c'img&
108                     " Error al enregistrar "&
109                     "l'index "&id&" en un array");
110           WHEN tipusIdxErroniArray =>
111              Impressio("l: "&l'img&" c: "&c'img&
112                     " L'index d'un array nomes"&
113                     " pot esser d'un tipus, "&
114                     "aquest es d' "&Id);
115           WHEN Tsub_No_Bool =>
116              Impressio("l:"&L'Img&" c:"&C'Img&
117                     " L'operand "&id&" no es de "&
118                     "tipus boolea");
119           WHEN Tops_Diferents =>
120              Impressio("l:"&L'Img&" c:"&C'Img&
121                     " Els tipus dels operands son"&
122                     " diferents");
123           WHEN Tsubs_Diferents =>
124              Impressio("l:"&L'Img&" c:"&C'Img&
125                     " Els tipus subjacents son "&
126                     "diferents");
127           WHEN Tsub_No_Escalar =>
```

```
128             Impressio("l:"&L'Img&" c:"&C'Img&
129                       " El tipus subjacent "&id&
130                       " no es escalar");
131          WHEN Tsub_No_Sencer =>
132             Impressio("l:"&L'Img&" c:"&C'Img&
133                       " El tipus subjacent "&Id&
134                       " no es sencer");
135          WHEN Tipus_No_Desc =>
136             Impressio("l:"&L'Img&" c:"&C'Img&
137                       " El tipus no es una "&
138                       "descripcio de tipus "&Id);
139          WHEN Id_No_Reconegut =>
140             Impressio("l:"&L'Img&" c:"&C'Img&
141                       " L'identificador "&Id&
142                       " no es reconegut");
143          WHEN Id_No_Cridaproc =>
144             Impressio("l:"&L'Img&" c:"&C'Img&
145                       " L'identificador "&Id&
146                       " nomes pot representar una"&
147                       " crida a procediment"&
148                       " sense parametres");
149          WHEN Assig_Tipus_Diferents =>
150             Impressio("l:"&L'Img&" c:"&C'Img&
151                       " L'assignacio es de tipus "&
152                       "diferents");
153          WHEN Exp_No_Bool =>
154             Impressio("l:"&L'Img&" c:"&C'Img&
155                       " L'expressio per un "&Id&
156                       " ha d'esser booleana");
157          WHEN Rec_No_Cridaproc =>
158             Impressio("l:"&L'Img&" c:"&C'Img&
159                       " No es pot utilitzar un "&
160                       "record com una crida a"&
161                       " procediment");
162          WHEN Falta_Param_Proc =>
163             Impressio("l:"&L'Img&" c:"&C'Img&
164                       " Falten parametres al "&
165                       "procediment");
166          WHEN Refvar_No_Proc =>
167             Impressio("l:"&L'Img&" c:"&C'Img&
168                       " No pot esser un procediment");
169          WHEN Falta_Param_Array =>
170             Impressio("l:"&L'Img&" c:"&C'Img&
```

```
171                         " Falten parametres a l'array");
172          WHEN Reccamp_No_Valid =>
173             Impressio("l:"&L'Img&" c:"&C'Img&
174                         " Camp no valid en l'acces "&
175                         "a referencia");
176          WHEN Idrec_No_Valid =>
177             Impressio("l:"&L'Img&" c:"&C'Img&
178                         " '"&Id&"' no es un nom de "&
179                         "camp valid");
180          WHEN Sobren_Parametres =>
181             Impressio("l:"&L'Img&" c:"&C'Img&
182                         " Sobren parametres");
183          WHEN Tparam_No_Coincident =>
184             Impressio("l:"&L'Img&" c:"&C'Img&
185                         " El tipus del parametre no "&
186                         "coincideix amb el tipus "&
187                         "demanat "&Id);
188          WHEN Tipus_No_Array =>
189             Impressio("l:"&L'Img&" c:"&C'Img&
190                         " El tipus '"&Id&"' no es un "&
191                         "array");
192          WHEN Tproc_No_Param =>
193             Impressio("l:"&L'Img&" c:"&C'Img&
194                         " El tipus no es un "&
195                         "procediment amb parametres "&Id);
196          WHEN OTHERS => NULL;
197       END CASE;
198    END Error;
199
200    PROCEDURE Error
201      (Te : IN Terror;
202       Id : String) IS
203    BEGIN
204       CASE Te IS
205          WHEN paramsPprincipal =>
206             Impressio("El programa principal "&
207                         "no pot tenir parametres");
208          WHEN id_existent =>
209             Impressio("l'identificador ja existeix");
210          WHEN OTHERS => NULL;
211       END CASE;
212    END Error;
213
```

```
214 END Semantica.Missatges;
```

4 Generació de codi intermedi

4.1 Codi de 3 adreces

4.1.1 Fitxer *semantica-declsc3a.ads*

```
1  WITH Decls.Dgenerals,
2    Semantica,
3    Decls.Dtdesc,
4    Ada.Sequential_Io,
5    Ada.Text_Io,
6    Decls.D_Taula_De_Noms,
7    Semantica.Ctipus,
8    Ada.Strings,
9    Ada.Strings.Fixed,
10   Ada.Strings.Maps;
11
12 USE Decls.Dgenerals,
13   Semantica,
14   Decls.Dtdesc,
15   Decls.D_Taula_De_Noms,
16   Semantica.Ctipus,
17   Ada.Strings,
18   Ada.Strings.Fixed,
19   Ada.Strings.Maps;
20
21 PACKAGE Semantica.Declsc3a IS
22
23    --taula procediments
24    PROCEDURE Nouproc
25      (Tp  : IN OUT T_Procs;
26       Idp : OUT num_proc);
27
28    FUNCTION Consulta
29      (Tp  : IN  T_Procs;
30       Idp : IN  num_proc) RETURN Info_Proc;
31
32    FUNCTION Consulta
33      (Tv  : IN T_Vars;
34       Idv : IN num_var) RETURN Info_Var;
35
36    PROCEDURE Modif_Descripcio
37      (Tv  : IN OUT T_Vars;
```

```
38        Idv : IN num_var;
39        Iv  : IN Info_Var);
40
41     PROCEDURE Novavar
42       (Tv   : IN OUT T_Vars;
43        Idpr : IN num_proc;
44        Idv  : OUT num_var);
45
46     PROCEDURE Novaconst
47       (Tv   : IN OUT T_Vars;
48        Vc   : IN Valor;
49        Tsub : IN tipussubjacent;
50        Idpr : IN num_proc;
51        Idc  : OUT num_var);
52
53     --Taula d'etiquetes
54     FUNCTION Nova_Etiq RETURN num_Etiq;
55
56     FUNCTION Etiqueta
57       (Ipr : IN Info_Proc) RETURN String;
58
59     --Fitxers
60     PROCEDURE Crea_Fitxer
61       (Nom_Fitxer : IN String);
62     PROCEDURE Obrir_Fitxer
63       (Nom_Fitxer : IN String);
64     PROCEDURE Tanca_Fitxer;
65     PROCEDURE Llegir_Fitxer
66       (Instruccio : OUT c3a);
67     PROCEDURE Escriure_Fitxer
68       (Instruccio : IN c3a);
69     FUNCTION Fi_Fitxer RETURN Boolean;
70
71  PRIVATE
72     PACKAGE Fitxer_Seq IS NEW Ada.Sequential_Io(c3a);
73     USE Fitxer_Seq;
74     F3as : Fitxer_Seq.File_Type;
75     F3at : Ada.Text_Io.File_Type;
76
77  END Semantica.Declsc3a;
```

4.1.2 Fitxer *semantica-declsc3a.adb*

```
1  PACKAGE BODY Semantica.Declsc3a IS
2
3     -- Taula Procediments
4     PROCEDURE Nouproc
5       (Tp  : IN OUT T_Procs;
6        Idp : OUT Num_Proc) IS
7     BEGIN
8        Posa(Tp, Info_Proc_Nul, Idp);
9     END Nouproc;
10
11
12    FUNCTION Consulta
13      (Tp  : IN T_Procs;
14       Idp : IN Num_Proc) RETURN Info_Proc IS
15    BEGIN
16       RETURN Tp.Tp(Idp);
17    END Consulta;
18
19
20    -- Taula Variables
21    FUNCTION Consulta
22      (Tv  : IN T_Vars;
23       Idv : IN Num_Var) RETURN Info_Var IS
24    BEGIN
25       RETURN Tv.Tv(Idv);
26    END Consulta;
27
28
29    PROCEDURE Modif_Descripcio
30      (Tv  : IN OUT T_Vars;
31       Idv : IN Num_Var;
32       Iv  : IN Info_Var) IS
33    BEGIN
34       Tv.Tv(Idv) := Iv;
35    END Modif_Descripcio;
36
37
38    PROCEDURE Novavar
39      (Tv   : IN OUT T_Vars;
40       Idpr : IN Num_Proc;
41       Idv  : OUT Num_Var) IS
```

```
42
43      Ip         : Info_Proc := Info_Proc_Nul;
44      Iv         : Info_Var  := Info_Var_Nul;
45      Numvar     : Integer   := Integer (Tv.Nv) + 1;
46      Nomvar     : String    := "_var" &
47        Integer'Image(Numvar);
48      Idn        : Id_Nom;
49
50    BEGIN
51      Nomvar(Nomvar'First + 4) := '_' ;
52      Posa_Id(Tn, Idn, Nomvar);
53      Ip:=Consulta(Tp, Idpr);
54      Iv:=(Id       => Idn,
55           Np       => Idpr,
56           Ocup     => Integer'Size / 8,
57           Desp     => 0,
58           Tsub     => Tsent,
59           Param    => False,
60           Const    => False,
61           Valconst => 0);
62
63      Ip.Ocup_Var := Ip.Ocup_Var + Iv.Ocup;
64      Posa(Tv, Iv, Idv);
65      Modif_Descripcio(Tp, Idpr, Ip);
66
67    END Novavar;
68
69
70    PROCEDURE Novaconst
71      (Tv    : IN OUT T_Vars;
72       Vc    : IN Valor;
73       Tsub  : IN Tipussubjacent;
74       Idpr  : IN Num_Proc;
75       Idc   : OUT Num_Var) IS
76
77      Idn       : Id_Nom;
78      E         : Boolean;
79      Iv        : Info_Var;
80      D         : Descrip;
81      Ocup      : Despl;
82      Nconst    : Num_Var := Tv.Nv + 1;
83      Nomconst  : String  := "_cnt" & Nconst'img;
84
```

```
85    BEGIN
86
87       Nomconst(Nomconst'First + 4) := '_';
88
89       IF Tsub=Tsarr THEN
90          Ocup:=16*Integer'Size;
91          Nomconst(2..4):="str";
92       ELSE
93          Ocup:=Integer'Size/8;
94       END IF;
95
96       Posa_Id(Tn, Idn, Nomconst);
97
98       Iv:=(Id        => Idn,
99            Np        => Idpr,
100           Ocup      => Integer'Size / 8,
101           Desp      => 0,
102           Tsub      => Tsub,
103           Param     => False,
104           Const     => True,
105           Valconst  => Vc);
106
107      Posa(Tv, Iv, Idc);
108
109      D:=(Dconst,
110          Id_Nul,
111          Vc,
112          Nconst);
113
114      Posa(Ts, Idn, D, E);
115
116   END Novaconst;
117
118   FUNCTION Nova_Etiq RETURN Num_Etiq IS
119   BEGIN
120      Ne := Ne + 1;
121      RETURN Ne;
122   END Nova_Etiq;
123
124   FUNCTION Etiqueta
125     (Idpr : IN num_Proc) RETURN String IS
126      Nomproc : String := Cons_Nom
127        (Tn, Consulta(Tp, Idpr).Idn);
```

```
128    BEGIN
129       RETURN "_" & Trim(Nomproc, Both);
130    END Etiqueta;
131
132
133    FUNCTION Etiqueta
134      (N : IN Integer) RETURN String IS
135       Text : String := "_etq" & Integer'Image (N);
136    BEGIN
137       Text(Text'First+4):='_';
138       RETURN Trim(Text, Both);
139    END Etiqueta;
140
141
142    FUNCTION Etiqueta
143      (Ipr : IN Info_Proc) RETURN String IS
144    BEGIN
145       CASE Ipr.Tp IS
146          WHEN Intern =>
147             RETURN "_etq_" & Trim(Ipr.Etiq'Img, Both);
148          WHEN Extern =>
149             RETURN "_" &
150               Trim(Cons_Nom(Tn, Ipr.Etiq_Extern), Both);
151       END CASE;
152    END Etiqueta;
153
154    --Fitxers
155    PROCEDURE Crea_Fitxer
156      (Nom_Fitxer : IN String) IS
157    BEGIN
158       Create(F3as, Out_File, Nom_Fitxer&".c3as");
159       Create(F3at, Out_File, Nom_Fitxer&".c3at");
160    END Crea_Fitxer;
161
162
163    PROCEDURE Obrir_Fitxer
164      (Nom_Fitxer : IN String) IS
165    BEGIN
166       Open(F3as, In_File, Nom_Fitxer&".c3as");
167    END Obrir_Fitxer;
168
169
170    PROCEDURE Tanca_Fitxer IS
```

```
171    BEGIN
172       Close(F3as);
173    END Tanca_Fitxer;
174
175
176    PROCEDURE Llegir_Fitxer
177      (Instruccio : OUT c3a) IS
178    BEGIN
179       Read(F3as, Instruccio);
180    END Llegir_Fitxer;
181
182
183    PROCEDURE Escriure_Fitxer
184      (Instruccio : IN c3a) IS
185    BEGIN
186
187       -- Escriptura a arxiu binari
188       Write(F3as, Instruccio);
189       -- Escriptura a arxiu de text
190       Put(F3at, Instruccio.Instr'Img & Ascii.Ht);
191
192       IF Instruccio.Instr <= Branc_Inc THEN
193          -- 1 operand
194          CASE Instruccio.Camp1.Tc IS
195             WHEN Proc =>
196                Put_Line(F3at, Instruccio.Camp1.Idp'Img);
197             WHEN Var =>
198                Put_Line(F3at, Instruccio.Camp1.Idv'Img);
199             WHEN Const =>
200                Put_Line(F3at, Instruccio.Camp1.Idc'Img);
201             WHEN Etiq =>
202                Put_Line(F3at, Instruccio.Camp1.Ide'Img);
203             WHEN OTHERS =>
204                NULL;
205          END CASE;
206
207       ELSIF Instruccio.Instr <= Paramc THEN
208          -- 2 operands
209          CASE Instruccio.Camp1.Tc IS
210             WHEN Proc =>
211                Put(F3at, Instruccio.Camp1.Idp'Img &
212                      Ascii.Ht);
213             WHEN Var =>
```

```
214                Put(F3at, Instruccio.Camp1.Idv'Img &
215                      Ascii.Ht);
216             WHEN Const =>
217                Put(F3at, Instruccio.Camp1.Idc'Img &
218                      Ascii.Ht);
219             WHEN Etiq =>
220                Put(F3at, Instruccio.Camp1.Ide'Img &
221                      Ascii.Ht);
222             WHEN OTHERS =>
223                NULL;
224          END CASE;
225
226          CASE Instruccio.Camp2.Tc IS
227             WHEN Proc =>
228                Put_Line(F3at, Instruccio.Camp2.Idp'Img);
229             WHEN Var =>
230                Put_Line(F3at, Instruccio.Camp2.Idv'Img);
231             WHEN Const =>
232                Put_Line(F3at, Instruccio.Camp2.Idc'Img);
233             WHEN Etiq =>
234                Put_Line(F3at, Instruccio.Camp1.Ide'Img);
235             WHEN OTHERS =>
236                NULL;
237          END CASE;
238
239       ELSE
240          -- 3 operands
241          CASE Instruccio.Camp1.Tc IS
242             WHEN Proc =>
243                Put(F3at, Instruccio.Camp1.Idp'Img &
244                      Ascii.Ht);
245             WHEN Var =>
246                Put(F3at, Instruccio.Camp1.Idv'Img &
247                      Ascii.Ht);
248             WHEN Const =>
249                Put(F3at, Instruccio.Camp1.Idc'Img &
250                      Ascii.Ht);
251             WHEN Etiq =>
252                Put(F3at, Instruccio.Camp1.Ide'Img &
253                      Ascii.Ht);
254             WHEN OTHERS =>
255                NULL;
256          END CASE;
```

```
257
258            CASE Instruccio.Camp2.Tc IS
259               WHEN Proc =>
260                  Put(F3at, Instruccio.Camp2.Idp'Img &
261                        Ascii.Ht);
262               WHEN Var =>
263                  Put(F3at, Instruccio.Camp2.Idv'Img &
264                        Ascii.Ht);
265               WHEN Const =>
266                  Put(F3at, Instruccio.Camp2.Idc'Img &
267                        Ascii.Ht);
268               WHEN Etiq =>
269                  Put(F3at, Instruccio.Camp1.Ide'Img &
270                        Ascii.Ht);
271               WHEN OTHERS =>
272                  NULL;
273            END CASE;
274
275            CASE Instruccio.Camp3.Tc IS
276               WHEN Proc =>
277                  Put_Line(F3at, Instruccio.Camp3.Idp'Img);
278               WHEN Var =>
279                  Put_Line(F3at, Instruccio.Camp3.Idv'Img);
280               WHEN Const =>
281                  Put_Line(F3at, Instruccio.Camp3.Idc'Img);
282               WHEN Etiq =>
283                  Put_Line(F3at, Instruccio.Camp3.Ide'Img);
284               WHEN OTHERS =>
285                  NULL;
286            END CASE;
287
288         END IF;
289      END Escriure_Fitxer;
290
291
292      FUNCTION Fi_Fitxer RETURN Boolean IS
293      BEGIN
294         RETURN End_Of_File(F3as);
295      END Fi_Fitxer;
296
297
298 END Semantica.Declsc3a;
```

4.2 Piles

4.2.1 Fitxer *piles.ads*

```
1  GENERIC
2
3     TYPE Tipus_Element IS PRIVATE;
4
5  PACKAGE pilas IS
6
7     Memoria_Agotada : EXCEPTION;
8
9     TYPE Pila IS LIMITED PRIVATE;
10
11    PROCEDURE Pila_Buida
12      (P : OUT Pila);
13
14    FUNCTION Es_Buida
15      (P : Pila) RETURN Boolean;
16
17    PROCEDURE Cim
18      (P : IN Pila;
19       Element : OUT Tipus_Element);
20
21    PROCEDURE Empilar
22      (P : IN OUT Pila;
23       Element : IN Tipus_Element);
24
25    PROCEDURE Desempilar
26      (P : IN OUT Pila);
27
28    PROCEDURE Destruir
29      (P : IN OUT Pila);
30
31 PRIVATE
32
33    TYPE Component;
34    TYPE Pila IS ACCESS Component;
35
36 END pilas;
```

4.2.2 Fitxer *piles.adb*

```
1  WITH Ada.Unchecked_Deallocation;
2
3  PACKAGE BODY pilas IS
4
5     TYPE Component IS RECORD
6        Cim    : Tipus_Element;
7        Resta : Pila;
8     END RECORD;
9
10    PROCEDURE Allibera_Memoria IS
11       NEW Ada.Unchecked_Deallocation
12      (Object => Component,
13       Name => Pila);
14
15    PROCEDURE Pila_Buida
16      (P : OUT Pila) IS
17    BEGIN
18       P := NULL;
19    END Pila_Buida;
20
21
22    FUNCTION Es_Buida
23      (P : Pila) RETURN Boolean IS
24    BEGIN
25       RETURN P = NULL;
26    END Es_Buida;
27
28
29    PROCEDURE Cim
30      (P : IN Pila;
31       Element : OUT Tipus_Element) IS
32    BEGIN
33       PRAGMA Assert
34         (P /= NULL, "Intent d'accedir a cim de pila buida");
35       Element := P.ALL.Cim;
36    END Cim;
37
38
39
40    PROCEDURE Empilar
41      (P : IN OUT Pila;
```

```
42          Element : IN Tipus_Element) IS
43       BEGIN
44          P := NEW Component'(Cim => Element, Resta => P);
45       EXCEPTION
46          WHEN Storage_Error => RAISE Memoria_Agotada;
47       END Empilar;
48
49
50       PROCEDURE Desempilar
51         (P : IN OUT Pila) IS
52          Antic : Pila;
53       BEGIN
54          PRAGMA Assert
55            (P /= NULL, "Intent de desempilar una pila buida");
56          Antic := P;
57          P := P.ALL.Resta;
58          Allibera_Memoria(Antic);
59       END Desempilar;
60
61
62       PROCEDURE Destruir
63         (P : IN OUT Pila) IS
64          Antic : Pila;
65       BEGIN
66          WHILE P /= NULL LOOP
67             Antic := P;
68             P := P.ALL.Resta;
69             Allibera_Memoria(Antic);
70          END LOOP;
71       END Destruir;
72
73    END pilas;
```

4.3 Generació de codi intermedi

4.3.1 Fitxer *semantica-gci.ads*

```
1  WITH Semantica.Declsc3a,
2    Pilas,
3    Decls.Dgenerals;
4
5  USE Semantica.Declsc3a;
6
7  PACKAGE Semantica.gci IS
8
9     Camp_Nul : CONSTANT Camp := (Tc => Const, Idc => Var_Nul);
10
11    TYPE T_Param IS RECORD
12       Base, Despl : num_Var;
13    END RECORD;
14
15    PROCEDURE Genera
16      (Instr : IN tInstruccio;
17       C1    : IN Camp := Camp_Nul;
18       C2    : IN Camp := Camp_Nul;
19       C3    : IN Camp := Camp_Nul);
20
21    PROCEDURE Inicia_Generacio
22      (nomFitxer : IN String);
23
24    PROCEDURE Gci_Decprocediment
25      (A : IN Pnode);
26
27    PROCEDURE gci_Programa
28      (A : IN Pnode);
29
30    PROCEDURE Gci_Encap
31      (A : IN Pnode;
32       I : IN Id_Nom);
33
34    PROCEDURE gci_Pencap
35      (A : IN Pnode);
36
37    PROCEDURE gci_Param
38      (A : IN Pnode);
39
40    PROCEDURE gci_Declaracions
```

```
41      (A : IN Pnode);
42
43    PROCEDURE gci_Decvar
44      (A : IN Pnode);
45
46    PROCEDURE gci_Declsvar
47      (A : IN Pnode);
48
49    PROCEDURE gci_Decconst
50      (A : IN Pnode);
51
52    PROCEDURE gci_Deccol
53      (A : IN Pnode);
54
55    PROCEDURE gci_Pcoleccio
56      (A : IN Pnode;
57       base: IN OUT Valor;
58       Idarray : OUT Id_nom);
59
60    PROCEDURE gci_Bloc
61      (A : IN Pnode);
62
63    PROCEDURE Gci_Assignacio
64      (Idref, Iddref, Idrexp, Iddexp: IN num_var);
65
66    --Procediments
67    PROCEDURE gci_Referencia_Proc
68      (A : IN Pnode;
69       Idproc : OUT num_proc);
70
71    PROCEDURE gci_Ref_Pri
72      (A : IN Pnode;
73       Idproc : OUT num_proc);
74
75    PROCEDURE gci_Identificador
76      (A : IN Pnode;
77       Idres, Iddesp: OUT num_var;
78       Idtipus : OUT Id_Nom);
79
80    PROCEDURE gci_Constant
81      (A : IN Pnode;
82       Idres : OUT Num_var);
83
```

```
84    PROCEDURE gci_Expressio
85      (A : IN Pnode;
86       Idr, Idd: OUT num_var);
87
88    PROCEDURE gci_Expressioc
89      (A : IN Pnode;
90       Idres,Idresdesp: OUT num_var);
91
92    PROCEDURE gci_Exp_Relacional
93      (IdResE, IdResD, IddespE, IddespD : IN num_var;
94       IdResultExp, IddespExp :   OUT num_var;
95       Op : IN Operacio);
96
97    PROCEDURE gci_Exp_Logica
98      (IdResE, IdResD, IddespE, IddespD : IN num_var;
99       IdResultExp, IddespExp :   OUT num_var;
100      Op : IN Operacio);
101
102   PROCEDURE gci_Exp_Aritmetica
103     (IdResE, IdResD, IddespE, IddespD : IN num_var;
104      IdResultExp, IddespExp : OUT num_var;
105      Op : IN Operacio);
106
107   PROCEDURE gci_Expressiou
108     (A : IN Pnode;
109      Idr, Idd : OUT num_var);
110
111   PROCEDURE gci_Exp_Negacio
112     (idRes, Iddesp : IN num_var;
113      IdresultExp, IddespExp : OUT num_var);
114
115   PROCEDURE gci_Exp_Neglogica
116     (idRes, Iddesp : IN num_var;
117      IdresultExp, IddespExp : OUT num_var);
118
119   PROCEDURE gci_Referencia_Var
120     (A : IN Pnode;
121      Idres, Iddesp: OUT Num_Var;
122      Idtipus : OUT Id_Nom);
123
124   --Arrays
125   PROCEDURE gci_Ref_Pri
126     (A : IN Pnode;
```

```
127         Idres, Iddesp, Idbase : OUT Num_var;
128         Idtipus : OUT Id_Nom;
129         It_Idx : OUT Cursor_Idx);
130
131     PROCEDURE gci_Ref_Rec
132       (A : IN Pnode;
133         Idres, Iddesp: OUT num_var;
134         Idtipus : OUT Id_Nom);
135
136     PROCEDURE gci_Sconds
137       (A : IN Pnode);
138
139     PROCEDURE gci_Scondc
140       (A : IN Pnode);
141
142     PROCEDURE gci_Srep
143       (A : IN Pnode);
144
145     PROCEDURE Calcula_Despls;
146
147 PRIVATE
148
149     Nprofunditat : nprof;
150
151 END Semantica.gci;
```

4.4 Generació de codi intermedi

4.4.1 Fitxer *semantica-gci.adb*

```
1  PACKAGE BODY Semantica.Gci IS
2
3     PACKAGE Pila_Proc IS NEW Pilas (Num_Proc);
4     USE Pila_Proc;
5     Pproc : Pila_Proc.Pila;
6
7     PACKAGE Pila_Param IS NEW Pilas (T_Param);
8     USE Pila_Param;
9     Pparam : Pila_Param.Pila;
10
11    PROCEDURE Genera
12      (instr : IN Tinstruccio;
13       C1    : IN Camp := Camp_Nul;
14       C2    : IN Camp := Camp_Nul;
15       C3    : IN Camp := Camp_Nul) IS
16    BEGIN
17       Escriure_Fitxer((instr, C1, C2, C3));
18    END Genera;
19
20
21    PROCEDURE inicia_Generacio
22      (Nomfitxer : IN String) IS
23    BEGIN
24       IF NOT Esem THEN
25          Crea_Fitxer(Nomfitxer);
26          Pila_Buida(Pproc);
27          Pila_Buida(Pparam);
28          Empilar(Pproc, Proc_Nul);
29       END IF;
30    END inicia_Generacio;
31
32
33    PROCEDURE Gci_Programa
34      (A : IN Pnode) IS
35    BEGIN
36       Nprofunditat := 1;
37       Empilar(Pproc, Proc_Nul);
38       Tv.Nv := Nv;
39       Gci_Decprocediment(A);
40       Calcula_Despls;
```

```
41        Tanca_Fitxer;
42    END Gci_Programa;
43
44
45    PROCEDURE Gci_Decprocediment
46      (A : IN Pnode) IS
47
48        Encap : Pnode RENAMES A.Fe5;
49        Decls : Pnode RENAMES A.Fc5;
50        Bloc : Pnode RENAMES A.Fd5;
51        Id : Pnode RENAMES A.Fid5;
52        Id_Proc : Id_Nom RENAMES A.Fid5.Id12;
53
54        Eip : Num_Etiq;
55        C1 : Camp;
56
57        Ipr : info_Proc;
58        Dproc : Descrip;
59
60        Idprinvocador,
61        Idprinvocat,
62        Nproc : Num_Proc;
63
64    BEGIN
65
66        Gci_Encap(Encap, Id_Proc);
67        Eip := Nova_Etiq;
68        Cim(Pproc, Nproc);
69        Dproc := Cons(Tts(Nproc), Id_Proc);
70
71        Ipr := (intern, 0, Id_Proc, Nprofunditat, 0, Eip);
72        Nprofunditat := Nprofunditat + 1;
73        Modif_Descripcio(Tp, Dproc.Np, Ipr);
74
75        IF Decls.Tipus = Declaracions THEN
76           Gci_Declaracions(Decls);
77        END IF;
78
79        C1 := (Etiq, Eip);
80        Genera(Etiqueta, C1);
81        C1:=(Proc, Dproc.Np);
82        Genera(Preamb, C1);
83        Gci_Bloc(Bloc);
```

```
84        Nprofunditat := Nprofunditat - 1;
85
86        --Rtn
87        Cim(Pproc, Idprinvocat);
88        C1:=(Proc, Idprinvocat);
89        Genera(Rtn, C1);
90
91        Desempilar(Pproc);
92        Cim(Pproc, Idprinvocador);
93
94     END Gci_Decprocediment;
95
96
97     PROCEDURE Gci_Encap
98       (A : IN Pnode;
99        I : IN Id_Nom) IS
100       Dproc : Descrip;
101       Idproc : Num_Proc;
102    BEGIN
103       IF A.Tipus = Pencap THEN
104          Gci_Pencap(A);
105       ELSE
106          Cim(Pproc, Idproc);
107          Dproc := Cons(Tts(Idproc), I);
108          Empilar(Pproc, Dproc.Np);
109       END IF;
110
111    END Gci_Encap;
112
113
114    PROCEDURE Gci_Pencap
115      (A : IN Pnode) IS
116
117       Param : Pnode RENAMES A.Fd1;
118       Fesq : Pnode RENAMES A.Fe1;
119       Dproc : Descrip;
120       Idproc : Num_Proc;
121
122    BEGIN
123       IF Fesq.Tipus = Identificador THEN
124          Cim(Pproc, Idproc);
125          Dproc := Cons(Tts(Idproc), Fesq.Id12);
126          Empilar(Pproc, Dproc.Np);
```

```
127          Gci_Param(Param);
128       ELSE
129          Gci_Pencap(Fesq);
130          Gci_Param(Param);
131       END IF;
132    END Gci_Pencap;
133
134
135    PROCEDURE Gci_Param
136      (A : IN Pnode) IS
137
138       Idpar : Id_Nom RENAMES A.Fe2.Id12;
139       D, Dtipus: Descrip;
140       Idproc : Num_Proc;
141       Iv : info_Var;
142
143    BEGIN
144       Cim(Pproc, Idproc);
145       D := Cons(Tts(Idproc), Idpar);
146
147       CASE D.Td IS
148          WHEN Dvar =>
149             Dtipus:=Cons(Tts(Idproc),D.Tr);
150             Iv := (Idpar,
151                    Idproc,
152                    Dtipus.Dt.Ocup,
153                    0,
154                    Dtipus.Dt.Tt,
155                    True,
156                    False,
157                    0);
158             Modif_Descripcio(Tv, D.Nv, Iv);
159
160          WHEN Dargc =>
161             Dtipus:=Cons(Tts(Idproc),D.Targ);
162             Iv := (Idpar,
163                    Idproc,
164                    Dtipus.Dt.Ocup,
165                    0,
166                    Dtipus.Dt.Tt,
167                    True,
168                    False,
169                    0);
```

```
170             Modif_Descripcio(Tv, D.Nvarg, Iv);
171          WHEN OTHERS =>
172             NULL;
173       END CASE;
174    END Gci_Param;
175
176
177    PROCEDURE Gci_Declaracions
178      (A : IN Pnode) IS
179
180       Decl : Pnode RENAMES A.Fd1;
181       Decls : Pnode RENAMES A.Fe1;
182
183    BEGIN
184       IF Decls.Tipus = Declaracions THEN
185          Gci_Declaracions(Decls);
186       END IF;
187
188       CASE Decl.Tipus IS
189          WHEN Dvariable   =>
190             Gci_Decvar(Decl);
191          WHEN Dconstant   =>
192             Null;
193          WHEN Dcoleccio   =>
194             Gci_Deccol(Decl);
195          WHEN Dregistre | Dencapregistre | Firecord =>
196             Null;
197          WHEN Dsubrang    =>
198             Null;
199          WHEN Procediment =>
200             Gci_Decprocediment(Decl);
201          WHEN OTHERS =>
202             Null;
203       END CASE;
204    END Gci_Declaracions;
205
206
207    PROCEDURE Gci_Decvar
208      (A : IN Pnode) IS
209
210       Dvariable : Pnode RENAMES A.Fd1;
211       Id : Id_Nom RENAMES A.Fe1.Id12;
212       Ivar : info_Var := info_Var_Nul;
```

```
213        Desc,Desctipus : Descrip;
214        Idproc : Num_Proc;
215
216     BEGIN
217        Gci_Declsvar(Dvariable);
218        Cim(Pproc, Idproc);
219        Desc:= Cons(Tts(Idproc),Id);
220        Desctipus := Cons(Tts(Idproc),Desc.Tr);
221        Ivar := (Id,
222                 Idproc,
223                 Desctipus.Dt.Ocup,
224                 0,
225                 Desctipus.Dt.Tt,
226                 False,
227                 False,
228                 0);
229        Modif_Descripcio(Tv, Desc.Nv, Ivar);
230     END Gci_Decvar;
231
232     PROCEDURE Gci_Declsvar
233       (A : IN Pnode) IS
234
235        Tnode : Tipusnode RENAMES A.Tipus;
236        Ivar : info_Var := info_Var_Nul;
237        Desc,Desctipus : Descrip;
238        Idproc : Num_Proc;
239
240     BEGIN
241        IF Tnode = Declmultvar THEN
242           Gci_Declsvar(A.Fd1);
243           Cim(Pproc, Idproc);
244           Desc:= Cons(Tts(Idproc),A.Fe1.Id12);
245           Desctipus := Cons(Tts(Idproc),Desc.Tr);
246           Ivar := (A.Fe1.Id12,
247                    Idproc,
248                    Desctipus.Dt.Ocup,
249                    0,
250                    Desctipus.Dt.Tt,
251                    False,
252                    False,
253                    0);
254           Modif_Descripcio(Tv, Desc.Nv, Ivar);
255        END IF;
```

```
256     END Gci_Declsvar;
257
258
259     PROCEDURE Gci_Decconst
260       (A : IN Pnode) IS
261
262        Id : Id_Nom RENAMES A.Fe2.Id12;
263        Val : Pnode RENAMES A.Fd2;
264        Iconst : info_Var := info_Var_Nul;
265        Desc, Desctipus : Descrip;
266        Idproc : Num_Proc;
267
268     BEGIN
269        Cim(Pproc, Idproc);
270        Desc:= Cons(Tts(Idproc),A.Fd1.Id12);
271        Desctipus := Cons(Tts(Idproc),Desc.Tr);
272        Iconst := (Id,
273                   Idproc,
274                   Desctipus.Dt.Ocup,
275                   0,
276                   Desctipus.Dt.Tt,
277                   False,
278                   True,
279                   Val.Val);
280        Modif_Descripcio(Tv, Desc.Nv, Iconst);
281     END Gci_Decconst;
282
283
284     PROCEDURE Gci_Deccol
285       (A : IN Pnode) IS
286
287        Darray : Descrip;
288        Fesq : Pnode RENAMES A.Fe1;
289        Idarray : Id_Nom;
290        Base : Valor := 0;
291        Idproc : Num_Proc;
292        T1 : Num_Var;
293
294     BEGIN
295        Gci_Pcoleccio(Fesq,Base,Idarray);
296        Cim(Pproc, Idproc);
297        Darray := Cons(Tts(Idproc),Idarray);
298        Novaconst(Tv, Base, Tsent, Idproc, T1);
```

```
299        Darray.Dt.Base := Base;
300        Actualitza(Tts(Idproc), Idarray, Darray);
301     END Gci_Deccol;
302
303
304     PROCEDURE Gci_Pcoleccio
305       (A : IN Pnode;
306        Base: IN OUT Valor;
307        Idarray : OUT Id_Nom) IS
308
309        Fesq : Pnode RENAMES A.Fe1;
310        Id   : Id_Nom RENAMES A.Fd1.Id12;
311        Ncomp : Valor;
312        Dtcamp : Descrip;
313        Idproc : Num_Proc;
314
315     BEGIN
316
317        Cim(Pproc, Idproc);
318        IF (A.Tipus = Pcoleccio) THEN
319           Gci_Pcoleccio(Fesq, Base, Idarray);
320           Dtcamp := Cons(Tts(Idproc),Id);
321           Ncomp :=  Dtcamp.Dt.Lsup - Dtcamp.Dt.Linf + 1;
322           Base := (Base * Ncomp) + Dtcamp.Dt.Linf;
323
324        ELSIF (A.Tipus = Pdimcoleccio) THEN
325           Dtcamp := Cons(Tts(Idproc),Id);
326           Idarray := Fesq.Id12;
327           Base := Dtcamp.Dt.Linf;
328        END IF;
329     END Gci_Pcoleccio;
330
331
332     PROCEDURE Gci_Bloc
333       (A : IN Pnode) IS
334
335        D : Descrip;
336        Idbase : Num_Proc;
337        Idtipus : Id_Nom;
338        Idres,
339        Iddesp,
340        Idr,
341        Idd: Num_Var;
```

```
342
343    BEGIN
344       CASE (A.Tipus) IS
345          WHEN Bloc =>
346             Gci_Bloc(A.Fe1);
347             Gci_Bloc(A.Fd1);
348          WHEN Repeticio =>
349             Gci_Srep(A);
350          WHEN Identificador =>
351             Gci_Identificador(A, Idres, Iddesp, Idtipus);
352          WHEN Fireferencia =>
353             Gci_Referencia_Proc(A, Idbase);
354          WHEN Condicionals =>
355             Gci_Sconds(A);
356          WHEN Condicionalc =>
357             Gci_Scondc(A);
358          WHEN Assignacio =>
359             Gci_Referencia_Var(A.Fe1, Idr, Idd, Idtipus);
360             Gci_Expressio(A.Fd1, Idres, Iddesp);
361             Gci_Assignacio(Idr, Idd, Idres, Iddesp);
362          WHEN OTHERS => NULL;
363       END CASE;
364    END Gci_Bloc;
365
366
367    PROCEDURE Gci_Assignacio
368      (Idref, Iddref, Idrexp, Iddexp: IN Num_Var) IS
369       C1,
370       C2,
371       C3,
372       C4,
373       C5 : Camp;
374       T : Num_Var;
375       Idproc : Num_Proc;
376
377    BEGIN
378       C1:=(Var, Idref);
379       C2:=(Var, Iddref);
380       C3:=(Var, Idrexp);
381       C4:=(Var, Iddexp);
382       Cim(Pproc, Idproc);
383
384       IF Iddref = Var_Nul THEN
```

```
385          IF Iddexp = Var_Nul THEN
386             Genera(Copia, C1, C3);
387          ELSE
388             Genera(Consindex, C1, C3, C4);
389          END IF;
390       ELSE
391          IF Iddexp = Var_Nul THEN
392             Genera(Asigindex, C1, C2, C3);
393          ELSE
394             Novavar(Tv, Idproc, T);
395             C5:=(Var, T);
396             Genera(Consindex, C5, C3, C4);
397             Genera(Asigindex, C1, C2, C5);
398          END IF;
399       END IF;
400    END Gci_Assignacio;
401
402
403    --Procediments
404    PROCEDURE Gci_Referencia_Proc
405      (A : IN Pnode;
406       Idproc : OUT Num_Proc) IS
407
408       Tipus : Tipusnode RENAMES A.Tipus;
409       Dproc : Descrip;
410       Prm : T_Param;
411       C1, C2 : Camp;
412
413    BEGIN
414       CASE Tipus IS
415          WHEN Identificador => --R -> Id
416             Idproc:= Proc_Nul;
417             Cim(Pproc,Idproc);
418             Dproc := Cons(Tts(Idproc), A.Id12);
419             Idproc := Dproc.Np;
420
421          WHEN Fireferencia => -- R -> Pri)
422             Gci_Ref_Pri(A.F6, Idproc);
423
424             WHILE NOT Es_Buida(Pparam) LOOP
425                Cim(Pparam, Prm);
426                C1:=(Var, Prm.Base);
427                C2:=(Var, Prm.Despl);
```

```
428
429                IF Prm.Despl=Var_Nul THEN
430                   Genera(Params, C1);
431                ELSE
432                   Genera(Paramc, C1, C2);
433                END IF;
434                Desempilar(Pparam);
435             END LOOP;
436
437             C1:=(Proc, Idproc);
438             Genera(Call, C1);
439
440          WHEN OTHERS =>
441             Put_Line("Error (Debug)");
442       END CASE;
443    END Gci_Referencia_Proc;
444
445
446    PROCEDURE Gci_Ref_Pri
447      (A : IN Pnode;
448       Idproc : OUT Num_Proc) IS
449
450       Tipus : Tipusnode RENAMES A.Tipus;
451       Fesq : Pnode RENAMES A.Fe1;
452       Fdret : Pnode RENAMES A.Fd1;
453
454       Idres, Iddesp : Num_Var;
455       Prm:  T_Param;
456
457    BEGIN
458       CASE Tipus IS
459          WHEN Pri => --Pri -> Pri,E
460             Gci_Ref_Pri(Fesq,Idproc);
461             Gci_Expressio(Fdret, Idres, Iddesp);
462             Prm.Base := Idres;
463             Prm.Despl := Iddesp;
464             Empilar(Pparam, Prm);
465
466          WHEN Encappri => -- Pri -> R(E
467             Gci_Referencia_Proc(Fesq, Idproc);
468             Gci_Expressio(Fdret, Idres, Iddesp);
469             Prm.Base := Idres;
470             Prm.Despl := Iddesp;
```

```
471             Empilar(Pparam, Prm);
472
473          WHEN OTHERS =>
474             Put_Line("Error (Debug)");
475       END CASE;
476    END Gci_Ref_Pri;
477
478
479    PROCEDURE Gci_Identificador
480      (A : IN Pnode;
481       Idres, Iddesp: OUT Num_Var;
482       Idtipus : OUT Id_Nom) IS
483
484       D , Descconst: Descrip;
485       Id : Id_Nom RENAMES A.Id12;
486       Desc : Tdescrip RENAMES D.Td;
487
488       Idv, T1 : Num_Var := Var_Nul;
489       Idproc : Num_Proc := Proc_Nul;
490       C1, C2: Camp;
491
492       Iv : info_Var;
493
494    BEGIN
495       Cim(Pproc, Idproc);
496       D := Cons(Tts(Idproc), Id);
497       CASE Desc IS
498          WHEN Dvar => -- R -> Id
499             Idres := D.Nv;
500             Iddesp := Var_Nul;
501             Idtipus := D.Tr;
502
503          WHEN Dconst =>
504             Descconst := Cons(Tts(Idproc), D.Tc);
505             Iv := (Id,
506                    Idproc,
507                    Descconst.Dt.Ocup,
508                    0,
509                    Descconst.Dt.Tt,
510                    False,
511                    True,
512                    D.Vc);
513             Modif_Descripcio(Tv, D.Nvc, Iv);
```

```
514             Novavar(Tv, Idproc, T1);
515
516             C1:=(Var, T1);
517             C2:=(Const, D.Nvc);
518             Genera(Copia, C1, C2);
519             Idres:= T1;
520             Iddesp:= Var_Nul;
521             Idtipus:= D.Tc;
522
523          WHEN Dargc =>
524             Novavar(Tv, Idproc, T1);
525             C1:=(Var, T1);
526             C2:=(Var, D.Nvarg);
527             Genera(Copia, C1, C2);
528             Idres := T1;
529             Iddesp := Var_Nul;
530             Idtipus := D.Targ;
531
532          WHEN Dproc =>
533             D:=Cons(Tts(Idproc), Id);
534             C1:=(Proc, D.Np);
535             Genera(Call, C1);
536
537          WHEN OTHERS =>
538             Put_Line("Es Un Altre Tipus Al Gci Identificador");
539
540       END CASE;
541    END Gci_Identificador;
542
543
544    PROCEDURE Gci_Constant
545      (A : IN Pnode;
546       Idres : OUT Num_Var) IS
547
548       Tatr : Tipus_Atribut RENAMES A.Tconst;
549       Idproc : Num_Proc;
550       T : Tipussubjacent;
551       T1 : Num_Var;
552       C1,
553       C2 : Camp;
554
555    BEGIN
556       Cim(Pproc, Idproc);
```

```
557         CASE (Tatr) IS
558            WHEN A_Lit_C =>
559               T := Tscar;
560            WHEN A_Lit_N =>
561               T := Tsent;
562            WHEN A_Lit_S =>
563               T := Tsstr;
564            WHEN OTHERS => NULL;
565         END CASE;
566
567         Novaconst(Tv, A.Val, T, Idproc, T1);
568         Novavar(Tv, Idproc, Idres);
569         C1:=(Const, T1);
570         C2:=(Var, Idres);
571         Genera(Copia, C2, C1);
572      END Gci_Constant;
573
574
575      PROCEDURE Gci_Expressio
576        (A : IN Pnode;
577         Idr, Idd: OUT Num_Var) IS
578
579         Tipus : Tipusnode RENAMES A.Tipus;
580         Idtipus : Id_Nom;
581         Desc : Descrip;
582
583      BEGIN
584         Idd := Var_Nul;
585         CASE Tipus IS
586            WHEN Expressio =>
587               Gci_Expressioc(A, Idr, Idd);
588            WHEN Expressiounaria =>
589               Gci_Expressiou(A, Idr, Idd);
590            WHEN Identificador =>
591               Gci_Identificador(A, Idr, Idd, Idtipus);
592            WHEN Const =>
593               Gci_Constant(A, Idr);
594            WHEN Fireferencia | Referencia =>
595               Gci_Referencia_Var(A, Idr, Idd, Idtipus);
596            WHEN OTHERS =>
597               Put_Line("Error (Debug)");
598         END CASE;
599      END Gci_Expressio;
```

```
600
601
602     PROCEDURE Gci_Expressioc
603       (A : IN Pnode;
604        Idres,Idresdesp: OUT Num_Var) IS
605
606        Fesq : Pnode RENAMES A.Fe3;
607        Fdret : Pnode RENAMES A.Fd3;
608        Op : Operacio RENAMES A.Op3;
609        Idesq,
610        Iddret,
611        Iddespe,
612        Iddespd : Num_Var;
613
614     BEGIN
615        --Analitzam L'Operand Esquerra
616        Gci_Expressio(Fesq, Idesq,Iddespe);
617        --Analitzam L'Operand Dret
618        Gci_Expressio(Fdret, Iddret,Iddespd);
619        -- Comparam Els Tipus
620        CASE Op IS
621           WHEN Unio | interseccio =>
622              Gci_Exp_Logica(Idesq, Iddret, Iddespe,
623                             Iddespd, Idres, Idresdesp, Op);
624           WHEN Menor | Menorig | Major | Majorig
625             | Igual | Distint =>
626              Gci_Exp_Relacional(Idesq, Iddret, Iddespe,
627                                 Iddespd, Idres, Idresdesp, Op);
628           WHEN Suma | Resta | Mult | Div | Modul =>
629              Gci_Exp_Aritmetica(Idesq, Iddret, Iddespe,
630                                 Iddespd, Idres, Idresdesp, Op);
631           WHEN OTHERS =>
632              Null;
633        END CASE;
634     END Gci_Expressioc;
635
636
637     PROCEDURE Gci_Exp_Relacional
638       (Idrese, Idresd, Iddespe, Iddespd : IN Num_Var;
639        Idresultexp, Iddespexp : OUT Num_Var;
640        Op : IN Operacio) IS
641
642        T1,
```

```
643         T2,
644         T3 : Num_Var := Var_Nul;
645
646         Emig,
647         Efi : Num_Etiq;
648
649         C1,
650         C2,
651         C3 : Camp;
652         Idproc : Num_Proc := Proc_Nul;
653
654     BEGIN
655         IF Iddespe = Var_Nul THEN
656             T1:= Idrese;
657         ELSE
658             Cim(Pproc, Idproc);
659             Novavar(Tv, Idproc, T1);
660             C1:=(Var, T1);
661             C2:=(Var, Idrese);
662             C3:=(Var, Iddespe);
663             Genera(Consindex,C1,C2,C3);
664         END IF;
665
666         IF Iddespd = Var_Nul THEN
667             T2 := Idresd;
668         ELSE
669             Cim(Pproc, Idproc);
670             Novavar(Tv, Idproc, T2);
671             C1:=(Var, T2);
672             C2:=(Var, Idresd);
673             C3:=(Var, Iddespd);
674             Genera(Consindex,C1,C2,C3);
675         END IF;
676
677         Emig:=Nova_Etiq;
678         Efi:=Nova_Etiq;
679         C1:=(Var, T1);
680         C2:=(Var, T2);
681         C3:=(Etiq, Emig);
682
683         CASE Op IS
684             WHEN Menor => Genera(Menor, C1, C2, C3);
685             WHEN Menorig => Genera(Menorigual, C1, C2, C3);
```

```
686           WHEN Igual => Genera(Igual,C1,C2,C3);
687           WHEN Majorig => Genera(Majorigual, C1, C2, C3);
688           WHEN Major => Genera(Major, C1, C2, C3);
689           WHEN Distint => Genera(Diferent, C1, C2, C3);
690           WHEN OTHERS => Null;
691        END CASE;
692
693        Cim(Pproc, Idproc);
694        Novavar(Tv, Idproc, T3);
695        C1 := (Var, T3 );
696        C2 := (Const, Zero);
697        Genera(Copia, C1, C2);
698        C3 := (Etiq, Efi);
699        Genera(Branc_inc, C3);
700        C3.Ide := Emig;
701        Genera(Etiqueta, C3);
702        C2.Idc := Menysu;
703        Genera(Copia, C1, C2);
704        C3.Ide := Efi;
705        Genera(Etiqueta, C3);
706        Idresultexp := T3;
707        Iddespexp := Var_Nul;
708     END Gci_Exp_Relacional;
709
710
711     PROCEDURE Gci_Exp_Logica
712       (Idrese, Idresd, Iddespe, Iddespd : IN Num_Var;
713        Idresultexp, Iddespexp :   OUT Num_Var;
714        Op : IN Operacio) Is
715
716        T1,
717        T2,
718        T3 : Num_Var := Var_Nul;
719        C1,
720        C2,
721        C3 : Camp;
722        Idproc : Num_Proc := Proc_Nul;
723
724     BEGIN
725        IF Iddespe = Var_Nul THEN
726           T1:= Idrese;
727        ELSE
728           Cim(Pproc, Idproc);
```

```
729          Novavar(Tv, Idproc, T1);
730          C1 := (Var, T1);
731          C2 := (Var, Idrese);
732          C3 := (Var, Iddespe);
733          Genera(Consindex,C1,C2,C3);
734       END IF;
735
736       IF Iddespd = Var_Nul THEN
737          T2 := Idresd;
738       ELSE
739          Cim(Pproc, Idproc);
740          Novavar(Tv, Idproc, T2);
741          C1 := (Var, T2);
742          C2 := (Var, Idresd);
743          C3 := (Var, Iddespd);
744          Genera(Consindex, C1, C2, C3);
745       END IF;
746
747       Cim(Pproc, Idproc);
748       Novavar(Tv, Idproc, T3);
749       C1 := (Var, T3);
750       C2 := (Var, T1);
751       C3 := (Var, T2);
752
753       CASE Op IS
754          WHEN Unio => Genera(Op_Or,C1,C2,C3);
755          WHEN interseccio => Genera(Op_And,C1,C2,C3);
756          WHEN OTHERS => Null;
757       END CASE;
758
759       Idresultexp := T3;
760       Iddespexp := Var_Nul;
761    END Gci_Exp_Logica;
762
763
764    PROCEDURE Gci_Exp_Aritmetica
765      (Idrese, Idresd, Iddespe, Iddespd : IN Num_Var;
766       Idresultexp, Iddespexp : OUT Num_Var;
767       Op : IN Operacio) IS
768
769       T1,
770       T2,
771       T3 : Num_Var := Var_Nul;
```

```
772         C1,
773         C2,
774         C3 : Camp;
775         Idproc : Num_Proc := Proc_Nul;
776
777     BEGIN
778         IF Iddespe = Var_Nul THEN
779             T1:= Idrese;
780         ELSE
781             Cim(Pproc, Idproc);
782             Novavar(Tv, Idproc, T1);
783             C1 := (Var, T1);
784             C2 := (Var, Idrese);
785             C3 := (Var, Iddespe);
786             Genera(Consindex,C1,C2,C3);
787         END IF;
788
789         IF Iddespd = Var_Nul THEN
790             T2 := Idresd;
791         ELSE
792             Cim(Pproc, Idproc);
793             Novavar(Tv, Idproc, T2);
794             C1 := (Var, T2);
795             C2 := (Var, Idresd);
796             C3 := (Var, Iddespd);
797             Genera(Consindex,C1,C2,C3);
798         END IF;
799
800         Cim(Pproc, Idproc);
801         Novavar(Tv, Idproc, T3);
802         C1 := (Var, T3);
803         C2:=(Var, T1);
804         C3:=(Var, T2);
805
806         CASE Op IS
807             WHEN Suma => Genera(Suma, C1, C2, C3);
808             WHEN Resta => Genera(Resta, C1, C2, C3);
809             WHEN Mult => Genera(Producte, C1, C2, C3);
810             WHEN Div => Genera(Divisio, C1, C2, C3);
811             WHEN Modul => Genera(Modul, C1, C2, C3);
812             WHEN OTHERS => Null;
813         END CASE;
814
```

```
815        Idresultexp := T3;
816        Iddespexp := Var_Nul;
817
818     END Gci_Exp_Aritmetica;
819
820
821     PROCEDURE Gci_Expressiou
822       (A : IN Pnode;
823        Idr, Idd : OUT Num_Var) IS
824
825        Fdret : Pnode RENAMES A.F4;
826        Op : Operacio RENAMES A.Op4;
827        Idru, Iddu : Num_Var;
828
829     BEGIN
830        Gci_Expressio(Fdret, Idru, Iddu);
831        CASE Op IS
832           WHEN Resta =>
833              Gci_Exp_Negacio(Idru, Iddu, Idr, Idd);
834           WHEN Negacio =>
835              Gci_Exp_Neglogica(Idru, Iddu, Idr, Idd);
836           WHEN OTHERS =>
837              NULL;
838        END CASE;
839     END Gci_Expressiou;
840
841
842     PROCEDURE Gci_Exp_Negacio
843       (Idres, Iddesp : IN Num_Var;
844        Idresultexp, Iddespexp : OUT Num_Var) IS
845
846        T1,
847        T2 : Num_Var := Var_Nul;
848        C1,
849        C2,
850        C3 : Camp;
851        Idproc : Num_Proc := Proc_Nul;
852
853     BEGIN
854        Cim(Pproc, Idproc);
855        IF Iddesp = Var_Nul THEN
856           T1:= Idres;
857        ELSE
```

```
858            Novavar(Tv, Idproc, T1);
859            C1 := (Var, T1);
860            C2 := (Var, Idres);
861            C3 := (Var, Iddesp);
862            Genera(Consindex, C1, C2, C3);
863         END IF;
864
865         Novavar(Tv, Idproc, T2);
866         C1 := (Var, T2);
867         C2 := (Var, T1);
868
869         Genera(Negacio, C1, C2);
870         Idresultexp := T2;
871         Iddespexp := Var_Nul;
872      END Gci_Exp_Negacio;
873
874
875      PROCEDURE Gci_Exp_Neglogica
876        (Idres, Iddesp : IN Num_Var;
877         Idresultexp, Iddespexp : OUT Num_Var) IS
878
879         T1,
880         T2 : Num_Var := Var_Nul;
881         C1,
882         C2,
883         C3 : Camp;
884         Idproc : Num_Proc := Proc_Nul;
885
886      BEGIN
887         Cim(Pproc, Idproc);
888         IF Iddesp = Var_Nul THEN
889            T1:= Idres;
890         ELSE
891            Novavar(Tv, Idproc, T1);
892            C1 := (Var, T1);
893            C2 := (Var, Idres);
894            C3 := (Var, Iddesp);
895            Genera(Consindex, C1, C2, C3);
896         END IF;
897
898         Novavar(Tv, Idproc, T2);
899         C1 := (Var, T2);
900         C2 := (Var, T1);
```

```
901
902        Genera(Op_Not, C1, C2);
903        Idresultexp := T2;
904        Iddespexp := Var_Nul;
905
906     END Gci_Exp_Neglogica;
907
908
909     PROCEDURE Gci_Referencia_Var
910       (A : IN Pnode;
911        Idres, Iddesp: OUT Num_Var;
912        Idtipus : OUT Id_Nom) IS
913
914        Tipus : Tipusnode RENAMES A.Tipus;
915        Idbase : Num_Var;
916        It_Idx : Cursor_Idx;
917        Da, Dtc : Descrip;
918        T1,T2,T3,T4,T5,T6,T7: Num_Var := Var_Nul;
919        Idproc : Num_Proc := Proc_Nul;
920        C1, C2, C3: Camp;
921
922     BEGIN
923        CASE Tipus IS
924           WHEN Identificador =>
925              Gci_Identificador(A, Idres, Iddesp, Idtipus);
926
927           WHEN Referencia => -- R -> R.Id
928              Gci_Ref_Rec(A, Idres, Iddesp, Idtipus);
929
930           WHEN Fireferencia => --R -> Ref_Pri)
931              Gci_Ref_Pri(A.F6, Idres, Iddesp, Idbase,
932                          Idtipus, It_Idx);
933
934              Cim(Pproc, Idproc);
935              Dtc := Cons(Tts(Idproc),Idtipus);
936              Idtipus := Dtc.Dt.Tcamp;
937              Novavar(Tv,Idproc, T7);
938              Novaconst(Tv, Valor(Dtc.Dt.Base), Tsent,
939                        Idproc, T3);
940
941              C1 := (Var, T7);
942              C2 := (Var, Iddesp);
943              C3 := (Const, T3);
```

```
944             Genera(Resta, C1, C2, C3);
945             Novavar(Tv, Idproc, T1);
946             Novaconst(Tv, Valor(integer'Size/8), Tsent,
947                       Idproc, T6);
948
949             C1 := (Var, T1);
950             C2 := (Var, T7);
951             C3 := (Const, T6);
952             Genera(Producte, C1, C2, C3);
953             Novavar(Tv, Idproc, T2);
954
955             IF Idbase = Var_Nul THEN
956                Iddesp := T1;
957             ELSE
958                Novavar(Tv, Idproc, T4);
959                Novaconst(Tv, Valor(Dtc.Dt.Ocup), Tsent,
960                          Idproc, T5);
961
962                C1 := (Var, T4);
963                C2 := (Const, T5);
964                C3 := (Var, T2);
965                Genera(Suma, C1, C2, C3);
966             END IF;
967          WHEN OTHERS => Null;
968       END CASE;
969
970    END Gci_Referencia_Var;
971
972
973    --Arrays
974    PROCEDURE Gci_Ref_Pri
975      (A : IN Pnode;
976       Idres, Iddesp, Idbase : OUT Num_Var;
977       Idtipus : OUT Id_Nom;
978       It_Idx : OUT Cursor_Idx) IS
979
980       Tipus : Tipusnode RENAMES A.Tipus;
981       Fesq : Pnode RENAMES A.Fe1;
982       Fdret : Pnode RENAMES A.Fd1;
983
984       Idrese, Iddespe : Num_Var := Var_Nul;
985
986       T0,T1,T2,T3 : Num_Var := Var_Nul;
```

```
987         C1, C2, C3 : Camp;
988         Idproc : Num_Proc := Proc_Nul;
989         Di : Id_Nom;
990         Dti: Descrip;
991         Ni : Valor;
992
993     BEGIN
994         CASE Tipus IS
995             WHEN Pri => --Pri -> Pri, E
996                 Cim(Pproc, Idproc);
997
998                 Gci_Ref_Pri(Fesq, Idres, Iddesp, Idbase,
999                             Idtipus, It_Idx);
1000                Gci_Expressio(Fdret, Idrese, Iddespe);
1001
1002                It_Idx := Succ_Idx(Tts(Idproc), It_Idx);
1003
1004                Di := Cons_Idx(Tts(Idproc), It_Idx);
1005                Dti := Cons(Tts(Idproc),Di);
1006                Ni := Dti.Dt.Lsup - Dti.Dt.Linf + 1;
1007
1008                Novaconst(Tv, Ni, Tsent, Idproc, T0);
1009                Novavar(Tv, Idproc, T1);
1010                C1 := (Var, T1);
1011                C2 := (Var, Iddesp);
1012                C3 := (Const, T0);
1013                Genera(Producte, C1, C2, C3);
1014                Novavar(Tv, Idproc, T2);
1015
1016                IF Iddespe = Var_Nul THEN
1017                    C1 := (Var, T2);
1018                    C2 := (Var, T1);
1019                    C3 := (Var, Idrese);
1020                    Genera(Suma, C1, C2, C3);
1021                ELSE
1022                    C1 := (Var, T3);
1023                    C2 := (Var, Idrese);
1024                    C3 := (Var, Iddespe);
1025                    Genera(Suma, C1, C2, C3);
1026
1027                    C1 := (Var, T2);
1028                    C2 := (Var, T1);
1029                    C3 := (Var, T3);
```

```
1030                Genera(Suma, C1, C2, C3);
1031             END IF;
1032             Iddesp := T2;
1033
1034          WHEN Encappri => -- Encappri --> R(E
1035             Cim(Pproc, Idproc);
1036
1037             Gci_Referencia_Var(Fesq, Idres, Idbase,
1038                                Idtipus);
1039             Gci_Expressio(Fdret, Idrese, Iddespe);
1040             It_Idx := Primer_Idx(Tts(Idproc), Idtipus);
1041
1042             IF Iddespe = Var_Nul THEN
1043                Iddesp:= Idrese;
1044             ELSE
1045                Novavar(Tv, Idproc, T1);
1046                C1:=(Var, T1);
1047                C2:=(Var, Idrese);
1048                C3:=(Var, Iddespe);
1049                Genera(Suma, C1, C2, C3);
1050                Iddesp:=T1;
1051             END IF;
1052
1053          WHEN OTHERS =>
1054             Put_Line("Error (Debug)");
1055       END CASE;
1056    END Gci_Ref_Pri;
1057
1058
1059    PROCEDURE Gci_Ref_Rec
1060      (A : IN Pnode;
1061       Idres, Iddesp: OUT Num_Var;
1062       Idtipus : OUT Id_Nom) IS
1063
1064       Fesq : Pnode RENAMES A.Fe1;
1065       Dcmp : Descrip;
1066       Dtcamp : Descrip;
1067       Idcamp : Id_Nom RENAMES A.Fd1.Id12;
1068
1069       Numconstant : Num_Var := Var_Nul;
1070
1071       T1 :Num_Var := Var_Nul;
1072       C1,
```

```
1073        C2,
1074        C3 : Camp;
1075        Dtipus_Camp : Descrip;
1076        Idproc : Num_Proc;
1077
1078     BEGIN
1079        Gci_Referencia_Var(Fesq, Idres, Iddesp,
1080                           Idtipus);
1081        Cim(Pproc, Idproc);
1082        Dcmp := Conscamp(Tts(Idproc), Idtipus,
1083                         Idcamp);
1084        Idtipus:= Dcmp.Tcamp;
1085        Dtipus_Camp := Cons(Ts, Idtipus);
1086        Novaconst(Tv, Valor(Dcmp.Dsp*4),
1087                  Tsent, Idproc, Numconstant);
1088        IF Iddesp = Var_Nul THEN
1089           Iddesp:=Numconstant;
1090        ELSE
1091           Novavar(Tv, Idproc, T1);
1092           C1:=(Var, T1);
1093           C2:=(Var, Iddesp);
1094           C3:=(Const, Numconstant);
1095           Genera (Suma, C1, C2, C3);
1096           Iddesp:= T1;
1097        END IF;
1098     END Gci_Ref_Rec;
1099
1100
1101     PROCEDURE Gci_Sconds
1102       (A : IN Pnode) IS
1103
1104        Cond : Pnode RENAMES A.Fe1;
1105        Bloc : Pnode RENAMES A.Fd1;
1106
1107        Idres, Iddesp : Num_Var;
1108
1109        C1, C2, C3 : Camp;
1110        Efals: Num_Etiq;
1111
1112        Idproc : Num_Proc;
1113        T1 : Num_Var := Var_Nul;
1114
1115     BEGIN
```

```
1116        Efals := Nova_Etiq;
1117        Gci_Expressio(Cond, Idres, Iddesp);
1118        IF Iddesp = Var_Nul THEN
1119           C2 := (Var, Idres);
1120        ELSE
1121           Cim(Pproc, Idproc);
1122           Novavar(Tv, Idproc, T1);
1123           C1 := (Var, T1);
1124           C2 := (Var, Idres);
1125           C3 := (Var, Iddesp);
1126           Genera(Consindex, C1, C2, C3);
1127           C2 := (Var, T1);
1128        END IF;
1129
1130        C3 := (Etiq, Efals);
1131        C1 := (Const, Zero);
1132        Genera(Igual, C2, C1, C3);
1133        Gci_Bloc(Bloc);
1134        Genera(Etiqueta, C3);
1135     END Gci_Sconds;
1136
1137
1138     PROCEDURE Gci_Scondc
1139       (A : IN Pnode) IS
1140
1141        Cond : Pnode RENAMES A.Fe2;
1142        Bloc : Pnode RENAMES A.Fc2;
1143        Blocelse : Pnode RENAMES A.Fd2;
1144
1145        Idres, Iddesp : Num_Var;
1146
1147        C1, C2, C3 : Camp;
1148        Efals, Efinal: Num_Etiq;
1149
1150        Idproc : Num_Proc;
1151        T1 : Num_Var := Var_Nul;
1152
1153     BEGIN
1154        Efals := Nova_Etiq;
1155        Efinal := Nova_Etiq;
1156        Gci_Expressio(Cond, Idres, Iddesp);
1157
1158        IF Iddesp = Var_Nul THEN
```

```
1159            C2 := (Var, Idres);
1160         ELSE
1161            Cim(Pproc, Idproc);
1162            Novavar(Tv, Idproc, T1);
1163            C1 := (Var, T1);
1164            C2 := (Var, Idres);
1165            C3 := (Var, Iddesp);
1166            Genera(Consindex, C1, C2, C3);
1167            C2 := (Var, T1);
1168         END IF;
1169
1170         C3 := (Etiq, Efals);
1171         C1 := (Const, Zero);
1172         Genera(Igual, C2, C1, C3);
1173
1174         Gci_Bloc(Bloc);
1175         C1 := (Etiq, Efinal);
1176         Genera(Branc_inc, C1);
1177
1178         C1.Ide:=Efals;
1179         Genera(Etiqueta, C1);
1180
1181         Gci_Bloc(Blocelse);
1182         C1 := (Etiq, Efinal);
1183         Genera(Etiqueta, C1);
1184
1185      END Gci_Scondc;
1186
1187
1188      PROCEDURE Gci_Srep
1189        (A : IN Pnode) IS
1190
1191         Exp : Pnode RENAMES A.Fe1;
1192         Bloc : Pnode RENAMES A.Fd1;
1193
1194         Idres, Iddesp : Num_Var;
1195
1196         C1, C2, C3 : Camp;
1197         Einicial, Efinal: Num_Etiq;
1198
1199         Idproc : Num_Proc;
1200         T1 : Num_Var := Var_Nul;
1201
```

```
1202    BEGIN
1203       Einicial := Nova_Etiq;
1204       Efinal := Nova_Etiq;
1205       C1 := (Etiq, Einicial);
1206       Genera(Etiqueta, C1);
1207
1208       Gci_Expressio(Exp, Idres, Iddesp);
1209       C1 := (Etiq, Efinal);
1210
1211       IF Iddesp = Var_Nul THEN
1212          C2 := (Var, Idres);
1213       ELSE
1214          Cim(Pproc, Idproc);
1215          Novavar(Tv, Idproc, T1);
1216          C1 := (Var, T1);
1217          C2 := (Var, Idres);
1218          C3 := (Var, Iddesp);
1219          Genera(Consindex, C1, C2, C3);
1220          C2 := (Var, T1);
1221       END IF;
1222
1223       C3 := (Const, Zero);
1224       Genera(Igual, C2, C3, C1);
1225
1226       Gci_Bloc(Bloc);
1227       C1 := (Etiq, Einicial);
1228       Genera(Branc_inc, C1);
1229       C1.Ide := Efinal;
1230       Genera(Etiqueta, C1);
1231
1232    END Gci_Srep;
1233
1234
1235    -- Calcula Desplacaments
1236    PROCEDURE Calcula_Despls IS
1237       Idpr     : Num_Proc;
1238       Ocup_Var : Despl;
1239
1240    BEGIN
1241       FOR P IN Num_Proc LOOP
1242          IF Tp.Tp(P).Tp = intern THEN
1243             Tp.Tp(P).Ocup_Var := 0;
1244          END IF;
```

```
1245        END LOOP;
1246
1247        FOR V IN Num_Var RANGE 1..Tv.Nv LOOP
1248           IF Tv.Tv(V).Param THEN
1249              Idpr := Tv.Tv(V).Np;
1250              IF Tp.Tp(Idpr).Tp = intern THEN
1251                 Tv.Tv(V).Desp := Tp.Tp(Idpr).Ocup_Param + 12;
1252                 Tp.Tp(Idpr).Ocup_Param :=
1253                   Despl(Tp.Tp(Idpr).Ocup_Param) + 4;
1254              END IF;
1255           ELSE
1256              Idpr := Tv.Tv(V).Np;
1257              IF Tp.Tp(Idpr).Tp = intern THEN
1258                 Ocup_Var := Tv.Tv(V).Ocup;
1259                 Tp.Tp(Idpr).Ocup_Var :=
1260                   Tp.Tp(Idpr).Ocup_Var + Ocup_Var;
1261                 Tv.Tv(V).Desp :=
1262                   Despl(Tp.Tp(Idpr).Ocup_Var* (-1));
1263              END IF;
1264           END IF;
1265        END LOOP;
1266
1267     END Calcula_Despls;
1268
1269
1270  END Semantica.Gci;
```

5 Assemblador

5.1 Generació de codi assemblador

5.1.1 Fitxer *semantica-assemblador.ads*

```
1  WITH Ada.Text_Io,
2    Ada.Strings,
3    Ada.Strings.Fixed,
4    Ada.Strings.Maps;
5
6  USE Ada.Text_Io,
7    Ada.Strings,
8    Ada.Strings.Fixed,
9    Ada.Strings.Maps;
10
11 PACKAGE Semantica.Assemblador IS
12     Error_Assemblador : EXCEPTION;
13     PROCEDURE Genera_Assemblador
14       (Nom_Fitxer : IN String);
15
16 PRIVATE
17     Nproc : Num_Proc := 0;
18     Fitxer_Asmbl : File_Type;
19 END Semantica.Assemblador;
```

5.1.2 Fitxer *semantica-assemblador.adb*

```
1  WITH Semantica.Declsc3a,
2    Decls.Dtdesc,
3    Decls.Dgenerals;
4
5  USE Semantica.Declsc3a,
6    Decls.Dtdesc,
7    Decls.Dgenerals;
8
9  PACKAGE BODY Semantica.Assemblador IS
10
11    -- Caracter TAB per els comentaris
12    Tab : CONSTANT Character := Ascii.Ht;
13
14    -- Variable per mantenir la profunditat
15    Prof_Actual : Nprof;
16
17    -- Procediments per inserir les instruccions
18    -- de codi assemblador. Tenim per els tres
19    -- tipus de instruccions C3@:
20    --    Instruccio amb 0 operadors
21    PROCEDURE Instr_0_Op
22      (Instruccio : IN String) IS
23    BEGIN
24       Put_Line(Fitxer_Asmbl, Tab & Instruccio);
25    END Instr_0_Op;
26
27    --    Instruccio amb 1 operador
28    PROCEDURE Instr_1_Op
29      (Instruccio,
30         Operand : IN String) IS
31    BEGIN
32       Put_Line(Fitxer_Asmbl, Tab & Instruccio &
33                   Tab & Operand);
34    END Instr_1_Op;
35
36    --    Instruccio amb 2 operadors
37    PROCEDURE Instr_2_Op
38      (Instruccio,
39         Operand1,
40         Operand2 : IN String) IS
41    BEGIN
```

```
42          Put_Line(Fitxer_Asmbl, Tab & Instruccio &
43                   Tab & Operand1 & ", " &
44                   Operand2);
45      END Instr_2_Op;
46
47      -- Comentaris
48      PROCEDURE Comentari
49        (Comentari : IN String) IS
50      BEGIN
51          Put_Line(Fitxer_Asmbl, Tab & " # " & Tab &
52                   Comentari);
53      END Comentari;
54
55      -- Etiquetes
56      PROCEDURE Etiqueta
57        (Etiqueta : IN String) IS
58      BEGIN
59          Put_Line(Fitxer_Asmbl, Trim(Etiqueta, Both) & ": nop");
60      END Etiqueta;
61
62      -- LD a, %eax
63      PROCEDURE Ld
64        (Org : IN Camp; --per ara usam camp
65         Dst : IN String) IS
66
67          Ivar : Info_Var;
68          Prof_Var : Nprof;
69          Vc : Valor;
70
71          Dpa : Integer;
72          Da : Despl RENAMES Ivar.Desp;
73
74      BEGIN
75          CASE Org.Tc IS
76             WHEN Var =>
77                Ivar := Consulta(Tv, Org.Idv);
78                Prof_Var := Consulta(Tp, Ivar.Np).Prof;
79
80                -- 'a' es una variable constant
81                IF Ivar.Const THEN
82                   Comentari("LD variable constant" & Org.Idv'Img &
83                             ", " & Dst & " i TS " &
84                             Ivar.Tsub'Img);
```

```
85                IF Ivar.Tsub = Tsstr THEN
86                   Instr_2_Op("movl", "$_cnt_" &
87                              Trim(Org.Idv'Img,
88                                   Both), Dst);
89                ELSE
90                   Instr_2_Op("movl", "$" &
91                              Trim(Ivar.Valconst'Img,
92                                   Both), Dst);
93                END IF;
94
95                -- 'a' es local
96             ELSIF Prof_Var = Prof_Actual THEN
97                -- 'a' es parametre local
98                IF Ivar.Param THEN
99                   Comentari("LD parametre local");
100                  Instr_2_Op("movl", Da'Img & "(%ebp)",
101                             "%esi");
102                  Instr_2_Op("movl", "(%esi)", Dst);
103                  -- 'a' es variable local
104               ELSE
105                  Comentari("LD variable local");
106                  Instr_2_Op("movl", Trim(Da'Img, Both) &
107                             "(%ebp)", Dst);
108               END IF;
109               -- 'a' es global
110            ELSIF Prof_Var < Prof_Actual THEN
111               -- 'a' es parametre global
112               IF Ivar.Param THEN
113                  Comentari("LD parametre global");
114                  Instr_2_Op("movl", "$DISP", "%esi");
115                  Dpa := 4*Integer(Prof_Var);
116                  Instr_2_Op("movl", Trim(Dpa'Img, Both) &
117                             "(%esi)", "%esi");
118                  Instr_2_Op("movl", Trim(Da'Img, Both) &
119                             "(%esi)", "%esi");
120                  Instr_2_Op("movl", "(%esi)", Dst);
121                  -- 'a' es variable global
122               ELSE
123                  Comentari("LD variable global");
124                  Instr_2_Op("movl", "$DISP", "%esi");
125                  Dpa := 4*Integer(Prof_Var);
126                  Instr_2_Op("movl", Trim(Dpa'Img, Both) &
127                             "(%esi)", "%esi");
```

```
128                  Instr_2_Op("movl", Trim(Da'Img, Both) &
129                                     "(%esi)", Dst);
130               END IF;
131            ELSE
132               RAISE Error_Assemblador;
133            END IF;
134
135         WHEN Const =>
136            -- 'a' es una constant
137            Ivar := Consulta(Tv, Org.Idc);
138            Vc := Ivar.Valconst;
139            IF Ivar.Tsub = Tsstr THEN
140               Comentari("LD es una constant string" &
141                           Org.Idc'Img
142                           & ", " & Dst);
143               Instr_2_Op("movl", "$_cnt_" &
144                           Trim(Org.Idc'Img,
145                                Both), Dst);
146            ELSE
147               Comentari("LD es una constant " &
148                           Org.Idc'Img
149                           & ", " & Dst);
150               Instr_2_Op("movl", "$" &
151                           Trim(Vc'Img, Both), Dst);
152            END IF;
153         WHEN OTHERS =>
154            RAISE Error_Assemblador;
155      END CASE;
156   END Ld;
157
158
159   -- ST %eax, a
160   PROCEDURE St
161     (Org : IN String;
162      Dst : IN Camp) IS
163
164      Prof_Var : Nprof;
165      Idst : Info_Var;
166
167      Dpa : Integer;
168      Da : Despl RENAMES Idst.Desp;
169
170   BEGIN
```

```
171         IF Dst.Tc /= Var THEN
172            RAISE Error_Assemblador;
173         END IF;
174
175         Idst := Consulta(Tv, Dst.Idv);
176         Prof_Var := Consulta(Tp, Idst.Np).Prof;
177         -- 'a' es local
178         IF Prof_Var = Prof_Actual THEN
179            -- 'a' es una variable local
180            IF NOT Idst.Param THEN
181               Comentari("ST a una variable local");
182               Instr_2_Op("movl", Org, Trim(Da'Img, Both) &
183                          "(%ebp)");
184               -- 'a' es un parametre local
185            ELSE
186               Comentari("ST a un parametre local");
187               Instr_2_Op("movl", Trim(Da'Img, Both) &
188                          "(%ebp)", "%edi");
189               Instr_2_Op("movl", Org, "(%edi)");
190            END IF;
191            -- 'a' es global
192         ELSIF Prof_Var < Prof_Actual THEN
193            -- 'a' es una variable global
194            IF NOT Idst.Param THEN
195               Comentari("ST a una variable global");
196               Instr_2_Op("movl", "$DISP", "%esi");
197               Dpa := 4*Integer(Prof_Var);
198               Instr_2_Op("addl", "$" & Trim(Dpa'Img, Both),
199                          "%esi");
200               Instr_2_Op("movl", "(%esi)", "%edi");
201               Instr_2_Op("movl", Org, Trim(Da'Img, Both) &
202                          "(%edi)");
203               -- 'a' es un parametre global
204            ELSE
205               Comentari("ST a un parametre global");
206               Instr_2_Op("movl", "$DISP", "%esi");
207               Dpa := 4*Integer(Prof_Var);
208               Instr_2_Op("addl", "$" & Trim(Dpa'Img, Both),
209                          "%esi");
210               Instr_2_Op("movl", "(%esi)", "%esi");
211               Instr_2_Op("movl", Trim(Da'Img, Both) &
212                          "(%esi)", "%edi");
213               Instr_2_Op("movl", Org, "(%edi)");
```

```
214            END IF;
215         ELSE
216            RAISE Error_Assemblador;
217         END IF;
218      END St;
219
220
221      -- LDA a, %eax
222      PROCEDURE Ldaddr
223        (Org : IN Camp;
224         Dst : IN String) IS
225
226         Ivar : Info_Var;
227         Prof_Var : Nprof;
228
229         Dpa : Integer;
230         Da : Despl RENAMES Ivar.Desp;
231
232      BEGIN
233         Ivar := Consulta(Tv, Org.Idv);
234         CASE Org.Tc IS
235            -- 'a' es constant
236            WHEN Const =>
237               Comentari("LDADDR amb a constant" &
238                           Org.Idc'Img & ", " & Dst);
239               Instr_2_Op("movl", "$" &
240                            Trim(Cons_Nom(Tn, Ivar.Id), Both), Dst);
241
242               -- 'a' es una variable
243            WHEN Var =>
244               Prof_Var := Consulta(Tp, Ivar.Np).Prof;
245               -- 'a' es una variable constant
246               IF Ivar.Const THEN
247                  Comentari("LDADDR amb var. constant" &
248                              Org.Idc'Img &
249                              ", " & Dst);
250                  Instr_2_Op("movl", "$" &
251                               Trim(Cons_Nom(Tn, Ivar.Id), Both),
252                             Dst);
253
254                  -- 'a' es local
255               ELSIF Prof_Var = Prof_Actual THEN
256                  -- 'a' es una variable local
```

```
257                IF NOT Ivar.Param THEN
258                   Comentari("LDADDR amb variable local");
259                   Instr_2_Op("leal", Trim(Da'Img, Both) &
260                                 "(%ebp)", Dst);
261                   -- 'a' es un parametre local
262                ELSE
263                   Comentari("LDADDR amb parametre local");
264                   Instr_2_Op("movl", Trim(Da'Img, Both) &
265                                 "(%ebp)", Dst);
266                END IF;
267
268                -- 'a' es global
269             ELSIF Prof_Var < Prof_Actual THEN
270                -- 'a' es una variable global
271                IF NOT Ivar.Param THEN
272                   Comentari("LDADDR amb variable global");
273                   Instr_2_Op("movl", "$DISP", "%esi");
274                   Dpa := 4*Integer(Prof_Var);
275                   Instr_2_Op("movl", Trim(Dpa'Img, Both) &
276                                 "(%esi)", "%esi");
277                   Instr_2_Op("leal", Trim(Da'Img, Both) &
278                                 "(%esi)", Dst);
279                   -- 'a' es un parametre global
280                ELSE
281                   Comentari("LDADDR amb parametre global");
282                   Instr_2_Op("movl", "$DISP", "%esi");
283                   Dpa := 4*Integer(Prof_Var);
284                   Instr_2_Op("movl", Trim(Dpa'Img, Both) &
285                                 "(%esi)", "%esi");
286                   Instr_2_Op("movl", Trim(Da'Img, Both) &
287                                 "(%esi)", Dst);
288                END IF;
289             ELSE
290                RAISE Error_Assemblador;
291             END IF;
292
293          WHEN OTHERS =>
294             RAISE Error_Assemblador;
295       END CASE;
296    END Ldaddr;
297
298
299    PROCEDURE Gce_Inicialitza
```

```
300       (Nom_Fitxer : IN String) IS
301        Iv : Info_Var;
302    BEGIN
303       Create(Fitxer_Asmbl, Out_File, Nom_Fitxer & ".s");
304       Obrir_Fitxer(Nom_Fitxer); --dc3a
305
306       --1) Constants
307       Put_Line(Fitxer_Asmbl, ".section .data");
308       FOR I IN Num_Var RANGE 1..Tv.Nv LOOP
309          Iv := Consulta(Tv, I);
310          IF Iv.Const THEN
311             IF Iv.Tsub = Tsstr THEN
312                --Si es un String
313                --s1 : .asciiz "El nombre de a's es"
314                Put_Line(Fitxer_Asmbl, Tab &
315                            Cons_Nom(Tn, Iv.Id)
316                            & " : .asciz " &
317                            Trim(Cons_Str(Tn, rang_tcar(Iv.Valconst)),
318                                 Both));
319             ELSIF Iv.Tsub = Tsent OR Iv.Tsub = Tsbool THEN
320                --Si es un numeric
321                --c3 : .long 3
322                Put_Line(Fitxer_Asmbl, Tab &
323                            Cons_Nom(Tn, Iv.Id) &
324                            " : .long " &
325                            Trim(Iv.Valconst'Img, Both));
326             ELSE
327                --Si es un caracter
328                --cc2 : .ascii "A"
329                Put_Line(Fitxer_Asmbl, Tab &
330                            Cons_Nom(Tn, Iv.Id) &
331                            " : .ascii """ &
332                            Trim(Iv.Valconst'Img, Both) & """");
333             END IF;
334          END IF;
335       END LOOP;
336
337       --2) Variables comuns
338       New_Line(Fitxer_Asmbl);
339       Put_Line(Fitxer_Asmbl, ".section .bss");
340       Put_Line(Fitxer_Asmbl, Tab & ".comm DISP, 100");
341
342       --3) Instruccions
```

```
343         New_Line(Fitxer_Asmbl);
344         Put_Line(Fitxer_Asmbl, ".section .text");
345         Put_Line(Fitxer_Asmbl, Tab & ".global main");
346         New_Line(Fitxer_Asmbl);
347         Put_Line(Fitxer_Asmbl, "main:");
348         Put_Line(Fitxer_Asmbl, Tab & "jmp _etq_1");
349
350      END Gce_Inicialitza;
351
352
353      PROCEDURE Gce_Genera IS
354
355         Ic3a : c3a;
356         Txt : String (1..10);
357         Dpn : Integer;
358         Ide : Num_Etiq;
359         Ipr : Info_Proc;
360
361      BEGIN
362         WHILE NOT Fi_Fitxer LOOP
363            Ipr := Info_Proc_Nul;
364            Ide := Etiq_Nul;
365            Llegir_Fitxer(Ic3a);
366            Txt := (OTHERS => ' ');
367            Dpn := 0;
368
369            CASE Ic3a.Instr IS
370
371               WHEN Rtn =>
372                  IF Ic3a.Camp1.Tc /= Proc THEN
373                     RAISE Error_Assemblador;
374                  END IF;
375                  New_Line(Fitxer_Asmbl);
376                  Comentari("Return " & Ic3a.Camp1.Idp'Img);
377                  Np := Np - 1;
378                  Ipr := Consulta(Tp, Ic3a.Camp1.Idp);
379                  Instr_2_Op("movl", "%ebp", "%esp");
380                  Instr_1_Op("popl", "%ebp");
381                  Instr_2_Op("movl", "$DISP", "%edi");
382                  Dpn := 4*Integer(Ipr.Prof);
383                  Instr_1_Op("popl", Trim(Dpn'Img, Both) &
384                                 "(%edi)");
385                  Instr_0_Op("ret");
```

```
386
387             WHEN Call =>
388                IF Ic3a.Camp1.Tc /= Proc THEN
389                   RAISE Error_Assemblador;
390                END IF;
391                New_Line(Fitxer_Asmbl);
392                Comentari("Call " & Ic3a.Camp1.Idp'Img);
393                Ipr := Consulta(Tp, Ic3a.Camp1.Idp);
394                IF Ic3a.Camp1.Idp = Id_Puts  OR
395                  Ic3a.Camp1.Idp = Id_Gets THEN
396                   Comentari("Crida a 'gets' o 'puts'");
397                   Instr_1_Op("popl", "%eax");
398                   Instr_2_Op("movl", "(%eax)", "%eax");
399                   Instr_1_Op("pushl", "%eax");
400                END IF;
401                Instr_1_Op("call", Trim(Etiqueta(Ipr), Both));
402                Instr_2_Op("addl", "$" & Trim(Ipr.Ocup_Param'Img,
403                                              Both), "%esp");
404
405                --end if;
406
407             WHEN Preamb =>
408                IF Ic3a.Camp1.Tc /= Proc THEN
409                   RAISE Error_Assemblador;
410                END IF;
411                New_Line(Fitxer_Asmbl);
412                Comentari("Preambul "& Ic3a.Camp1.Idp'Img);
413                Nproc := Nproc + 1;
414                Ipr := Consulta(Tp, Ic3a.Camp1.Idp);
415                Prof_Actual := Ipr.Prof;
416                Instr_2_Op("movl", "$DISP", "%esi");
417                Dpn := 4*Integer(Ipr.Prof);
418                Instr_1_Op("pushl", Trim(Dpn'Img, Both) &
419                             "(%esi)");
420                Instr_1_Op("pushl", "%ebp");
421                Instr_2_Op("movl", "%esp", "%ebp");
422                Instr_2_Op("movl", "%ebp", Trim(Dpn'Img, Both) &
423                             "(%esi)");
424
425                Instr_2_Op("subl", "$" & Trim(Ipr.Ocup_Var'Img,
426                                              Both), "%esp");
427
428             WHEN Params =>
```

```
429                New_Line(Fitxer_Asmbl);
430                Comentari("Parametre Simple");
431                Ldaddr(Ic3a.Camp1, "%eax");
432                Instr_1_Op("pushl", "%eax");
433
434             WHEN Etiqueta =>
435                IF Ic3a.Camp1.Tc /= Etiq THEN
436                   RAISE Error_Assemblador;
437                END IF;
438                New_Line(Fitxer_Asmbl);
439                Comentari("Etiqueta " & Ic3a.Camp1.Ide'Img);
440                Etiqueta("_etq_" & Trim(Ic3a.Camp1.Ide'Img, Both));
441
442             WHEN Branc_Inc =>
443                IF Ic3a.Camp1.Tc /= Etiq THEN
444                   RAISE Error_Assemblador;
445                END IF;
446                New_Line(Fitxer_Asmbl);
447                Comentari("Brancament Incondicional " &
448                            Ic3a.Camp1.Ide'Img);
449                Instr_1_Op("jmp", "_etq_" &
450                            Trim(Ic3a.Camp1.Ide'Img, Both));
451
452                -- 2 Operands
453             WHEN Negacio =>
454                New_Line(Fitxer_Asmbl);
455                Comentari("Negacio");
456                Instr_2_Op("xorl", "%eax", "%eax");
457                Ld(Ic3a.Camp2, "%ebx");
458                Instr_2_Op("subl", "%ebx", "%eax");
459                St("%eax", Ic3a.Camp1);
460
461             WHEN Op_Not =>
462                New_Line(Fitxer_Asmbl);
463                Comentari("Not");
464                Ld(Ic3a.Camp2, "%eax");
465                Instr_1_Op("notl", "%eax");
466                St("%eax", Ic3a.Camp1);
467
468             WHEN Copia =>
469                New_Line(Fitxer_Asmbl);
470                Comentari("Copia");
471                Ld(Ic3a.Camp2, "%eax");
```

```
472                St("%eax", Ic3a.Camp1);
473
474             WHEN Paramc =>
475                New_Line(Fitxer_Asmbl);
476                Comentari("Parametre Compost");
477                Ldaddr(Ic3a.Camp1, "%eax");
478                Ld(Ic3a.Camp2, "%ebx");
479                Instr_2_Op("addl", "%ebx", "%eax");
480                Instr_1_Op("pushl", "%eax");
481
482                -- 3 Operands
483             WHEN Suma =>
484                New_Line(Fitxer_Asmbl);
485                Comentari("Suma");
486                Ld(Ic3a.Camp2, "%eax");
487                Ld(Ic3a.Camp3, "%ebx");
488                Instr_2_Op("addl", "%eax", "%ebx");
489                St("%ebx", Ic3a.Camp1);
490
491             WHEN Resta =>
492                New_Line(Fitxer_Asmbl);
493                Comentari("Resta");
494                Ld(Ic3a.Camp2, "%eax");
495                Ld(Ic3a.Camp3, "%ebx");
496                Instr_2_Op("subl", "%ebx", "%eax");
497                St("%eax", Ic3a.Camp1);
498
499             WHEN Producte =>
500                New_Line(Fitxer_Asmbl);
501                Comentari("Producte");
502                Ld(Ic3a.Camp2, "%eax");
503                Ld(Ic3a.Camp3, "%ebx");
504                Instr_2_Op("imull", "%eax", "%ebx");
505                St("%ebx", Ic3a.Camp1);
506
507             WHEN Divisio =>
508                New_Line(Fitxer_Asmbl);
509                Comentari("Divisio");
510                Ld(Ic3a.Camp2, "%eax");
511                Instr_2_Op("movl", "%eax", "%edx");
512                Instr_2_Op("sarl", "$31", "%edx");
513                Ld(Ic3a.Camp3, "%ebx");
514                Instr_1_Op("idivl", "%ebx");
```

```
515                St("%eax", Ic3a.Camp1);
516
517             WHEN Modul =>
518                New_Line(Fitxer_Asmbl);
519                Comentari("Modul");
520                Ld(Ic3a.Camp2, "%eax");
521                Instr_2_Op("movl", "%eax", "%edx");
522                Instr_2_Op("sarl", "$31", "%edx");
523                Ld(Ic3a.Camp3, "%ebx");
524                Instr_1_Op("idivl", "%ebx");
525                St("%edx", Ic3a.Camp1);
526
527             WHEN Op_And =>
528                New_Line(Fitxer_Asmbl);
529                Comentari("AND");
530                Ld(Ic3a.Camp2, "%eax");
531                Ld(Ic3a.Camp3, "%ebx");
532                Instr_2_Op("andl", "%ebx", "%eax");
533                St("%eax", Ic3a.Camp1);
534
535             WHEN Op_Or =>
536                New_Line(Fitxer_Asmbl);
537                Comentari("OR");
538                Ld(Ic3a.Camp2, "%eax");
539                Ld(Ic3a.Camp3, "%ebx");
540                Instr_2_Op("orl", "%ebx", "%eax");
541                St("%eax", Ic3a.Camp1);
542
543             WHEN Consindex =>
544                New_Line(Fitxer_Asmbl);
545                Comentari("Consulta index");
546                Ldaddr(Ic3a.Camp2, "%esi");
547                Ld(Ic3a.Camp3, "%eax");
548                Instr_2_Op("addl", "%eax", "%esi");
549                Instr_2_Op("movl", "(%esi)", "%eax");
550                St("%eax", Ic3a.Camp1);
551
552             WHEN Asigindex =>
553                New_Line(Fitxer_Asmbl);
554                Comentari("Assignacio d'index");
555                Ldaddr(Ic3a.Camp1, "%edi");
556                Ld(Ic3a.Camp2, "%eax");
557                Instr_2_Op("addl", "%eax", "%edi");
```

```
558                Ld(Ic3a.Camp3, "%eax");
559                Instr_2_Op("movl", "%eax","(%edi)");
560
561             WHEN Menor =>
562                IF Ic3a.Camp3.Tc /= Etiq THEN
563                   RAISE Error_Assemblador;
564                END IF;
565                Comentari("IF Menor");
566                Ide := Nova_Etiq;
567                Ld(Ic3a.Camp1, "%eax");
568                Ld(Ic3a.Camp2, "%ebx");
569                Instr_2_Op("cmpl", "%ebx", "%eax");
570                Instr_1_Op("jge", "_etq_" &
571                           Trim(Ide'Img, Both));
572                Instr_1_Op("jmp", "_etq_" &
573                           Trim(Ic3a.Camp3.Ide'Img, Both));
574                Etiqueta("_etq_" & Trim(Ide'Img, Both));
575
576             WHEN Menorigual =>
577                IF Ic3a.Camp3.Tc /= Etiq THEN
578                   RAISE Error_Assemblador;
579                END IF;
580                Comentari("IF Menor o Igual");
581                Ide := Nova_Etiq;
582                Ld(Ic3a.Camp1, "%eax");
583                Ld(Ic3a.Camp2, "%ebx");
584                Instr_2_Op("cmpl", "%ebx", "%eax");
585                Instr_1_Op("jg", "_etq_" &
586                           Trim(Ide'Img, Both));
587                Instr_1_Op("jmp", "_etq_" &
588                           Trim(Ic3a.Camp3.Ide'Img, Both));
589                Etiqueta("_etq_" & Trim(Ide'Img, Both));
590
591             WHEN Igual =>
592                IF Ic3a.Camp3.Tc /= Etiq THEN
593                   RAISE Error_Assemblador;
594                END IF;
595                Comentari("IF Igual");
596                Ide := Nova_Etiq;
597                Ld(Ic3a.Camp1, "%eax");
598                Ld(Ic3a.Camp2, "%ebx");
599                Instr_2_Op("cmpl", "%ebx", "%eax");
600                Instr_1_Op("jne", "_etq_" &
```

```
601                              Trim(Ide'Img, Both));
602                 Instr_1_Op("jmp", "_etq_" &
603                              Trim(Ic3a.Camp3.Ide'Img, Both));
604                 Etiqueta("_etq_" & Trim(Ide'Img, Both));
605
606              WHEN Majorigual =>
607                 IF Ic3a.Camp3.Tc /= Etiq THEN
608                    RAISE Error_Assemblador;
609                 END IF;
610                 Comentari("IF Major o Igual");
611                 Ide := Nova_Etiq;
612                 Ld(Ic3a.Camp1, "%eax");
613                 Ld(Ic3a.Camp2, "%ebx");
614                 Instr_2_Op("cmpl", "%ebx", "%eax");
615                 Instr_1_Op("jl", "_etq_" &
616                              Trim(Ide'Img, Both));
617                 Instr_1_Op("jmp", "_etq_" &
618                              Trim(Ic3a.Camp3.Ide'Img, Both));
619                 Etiqueta("_etq_" & Trim(Ide'Img, Both));
620
621              WHEN Major =>
622                 IF Ic3a.Camp3.Tc /= Etiq THEN
623                    RAISE Error_Assemblador;
624                 END IF;
625                 Comentari("IF Major");
626                 Ide := Nova_Etiq;
627                 Ld(Ic3a.Camp1, "%eax");
628                 Ld(Ic3a.Camp2, "%ebx");
629                 Instr_2_Op("cmpl", "%ebx", "%eax");
630                 Instr_1_Op("jle", "_etq_" &
631                              Trim(Ide'Img, Both));
632                 Instr_1_Op("jmp", "_etq_" &
633                              Trim(Ic3a.Camp3.Ide'Img, Both));
634                 Etiqueta("_etq_" & Trim(Ide'Img, Both));
635
636              WHEN Diferent =>
637                 IF Ic3a.Camp3.Tc /= Etiq THEN
638                    RAISE Error_Assemblador;
639                 END IF;
640                 Comentari("IF Diferent");
641                 Ide := Nova_Etiq;
642                 Ld(Ic3a.Camp1, "%eax");
643                 Ld(Ic3a.Camp2, "%ebx");
```

```
644                Instr_2_Op("cmpl", "%ebx", "%eax");
645                Instr_1_Op("je", "_etq_" &
646                           Trim(Ide'Img, Both));
647                Instr_1_Op("jmp", "_etq_" &
648                           Trim(Ic3a.Camp3.Ide'Img, Both));
649                Etiqueta("_etq_" & Trim(Ide'Img, Both));
650
651             WHEN OTHERS =>
652                RAISE Error_Assemblador;
653          END CASE;
654       END LOOP;
655
656    END Gce_Genera;
657
658
659    PROCEDURE Gce_Finalitza IS
660    BEGIN
661       Tanca_Fitxer; --dc3a
662       Close(Fitxer_Asmbl);
663    EXCEPTION
664       WHEN OTHERS=>
665          NULL;
666    END Gce_Finalitza;
667
668
669    PROCEDURE Genera_Assemblador
670      (Nom_Fitxer : IN String) IS
671    BEGIN
672       IF Esem THEN
673          RAISE Error_Assemblador;
674       END IF;
675       Gce_Inicialitza(Nom_Fitxer);
676       Gce_Genera;
677       Gce_Finalitza;
678    EXCEPTION
679       WHEN Error_Assemblador =>
680          Comentari("Error assemblador");
681    END Genera_Assemblador;
682
683
684 END Semantica.Assemblador;
```

6 Proves i programa principal

6.1 Fitxer *compilemon.adb*, programa principal

```
1  -- COMPILEMON.adb
2  -- Programa per compilar el compilador
3
4  WITH Ada.Text_IO,
5    Ada.Command_Line,
6    Decls.D_Taula_De_Noms,
7    Decls.Dgenerals,
8    Decls.Dtdesc,
9    Pk_Usintactica_Tokens,
10   Pk_Ulexica_Io,
11   U_Lexica,
12   Pk_Usintactica,
13   Decls.D_Atribut,
14   Semantica,
15   Decls.Dtnode,
16   Semantica.Ctipus,
17   Semantica.Declsc3a,
18   Semantica.Gci,
19   Semantica.Assemblador;
20
21 USE Ada.Text_IO,
22   Ada.Command_Line,
23   Decls.D_Taula_De_Noms,
24   Decls.Dgenerals,
25   Decls.Dtdesc,
26   Pk_Usintactica_Tokens,
27   Pk_Ulexica_Io,
28   U_Lexica,
29   Pk_Usintactica,
30   Decls.D_Atribut,
31   Semantica,
32   Decls.Dtnode,
33   Semantica.Ctipus,
34   Semantica.Declsc3a,
35   Semantica.Gci,
36   Semantica.Assemblador;
37
38
39 PROCEDURE Compilemon IS
```

```
40
41 BEGIN
42    Open_Input(Argument(1));
43    Inicia_analisi(Argument(1));
44    yyparse;
45
46    --Comprovacio de tipus
47    Ct_Programa(Arbre);
48
49    IF NOT esem THEN
50       -- Generacio de codi intermedi
51       Inicia_Generacio(Argument(1));
52       Gci_Programa(Arbre);
53
54       -- Generacio de codi assemblador
55       Genera_Assemblador(Argument(1));
56    END IF;
57
58    Close_Input;
59
60 EXCEPTION
61    WHEN Syntax_Error =>
62       Put_Line("ERROR CompiLEMON: Error a la linea "
63                &yy_line_number'img&
64                " i columna "&yy_begin_column'img);
65 END compilemon;
```

7 Declaracions i altres paquets

7.1 Fitxer *decls.ads*

```
1 -- DECLS.ads
2 -- Paquet de declaracions
3
4 PACKAGE decls IS
5
6    --pragma pure;
7
8
9 END decls;
```

7.2 Fitxer *decls-dgenerals.ads*

```
1  -- DECLS-DGENERALS.ads
2  -- Paquet de declaracions generals
3
4  PACKAGE Decls.Dgenerals IS
5
6      --pragma pure;
7
8      Max_Id : CONSTANT Integer := 1000;
9      Long_Num_Ident : CONSTANT Integer := 40;
10
11     Max_Var : CONSTANT Integer := 1000;
12     TYPE Num_Var IS NEW Natural
13       RANGE 0 .. Max_Var;
14     Var_Nul : Num_Var := 0;
15
16     Max_Proc : CONSTANT Integer := 100;
17     TYPE Num_Proc IS NEW Natural
18       RANGE 0 .. Max_Proc;
19     Proc_Nul : Num_Proc := 0;
20
21     Max_Etiquetes : CONSTANT Integer := 4000;
22     TYPE Num_Etiq IS NEW Integer
23       RANGE 0 .. Max_Etiquetes;
24     Etiq_Nul : Num_Etiq := 0;
25
26     TYPE Tipus_Etiq IS
27       (Etiq_Num,
28        Etiq_Proc);
29
30     TYPE valor IS NEW Integer
31       RANGE Integer'First..Integer'Last;
32
33     TYPE tipus_atribut IS
34       (Atom,
35        A_Ident,
36        A_Lit_C,
37        A_Lit_N,
38        A_Lit_S,
39        NodeArbre);
40
41    Esem : Boolean := False;
```

```
42
43 END Decls.Dgenerals;
```

8 Jocs de proves

8.1 Arrays de records

8.1.1 Fitxer *prova1.lem*

```
1  -- PROVA1.lem
2  -- Array de records.
3
4  PROCEDURE prova1 IS
5
6          TYPE tipussubjacent IS RECORD
7
8                  --linf : integer;
9                  lsup : integer;
10                 ts : character;
11                 o : string;
12         END RECORD;
13
14         TYPE rang IS NEW integer RANGE 0..3;
15
16         TYPE descrip_tipus IS ARRAY (rang) OF tipussubjacent;
17         d : descrip_tipus;
18
19         i : rang;
20
21 BEGIN
22         d(0).ts := 'n';
23         --d(0).linf := 0;
24         d(0).lsup := 0;
25         d(0).o := "merda";
26
27         d(1).ts := 'b';
28         --d(1).linf := -1;
29         d(1).lsup := 0;
30         d(1).o := "caca";
31
32         d(2).ts := 'c';
33         --d(2).linf := 1;
34         d(2).lsup := 4;
35         d(2).o := "cacota";
36
37         d(3).ts := 's';
```

```
38          --d(3).linf := 4;
39          d(3).lsup := 256;
40          d(3).o := "pichac0";
41
42          i := 0;
43          WHILE (i <= 3) LOOP
44                  putc(d(i).ts);
45                  new_line;
46                  --puti(d(i).linf);
47                  --new_line;
48                  puti(d(i).lsup);
49                  puts(d(i).o);
50                  new_line;
51                  new_line;
52                  i := i + 1;
53          END LOOP;
54  END prova1;
```

8.2 Suma de matrius

8.2.1 Fitxer *prova2.lem*

```
1  -- PROVA2.lem
2  -- Prova amb suma de matrius 3x3
3
4  PROCEDURE prova2 IS
5
6          TYPE rang IS NEW integer RANGE 0..2;
7          TYPE matriu IS ARRAY (rang, rang) OF integer;
8          a, b, c : matriu;
9
10         i, j : rang;
11         aux_a, aux_b : integer;
12
13 BEGIN
14
15         -- Exemple extret de l'article 'Matrices'
16         -- de la Wikipedia
17
18         -- Matriu 'a':
19         -- 1 3 2
20         -- 1 0 0
21         -- 1 2 2
22         a(0, 0) := 1;
23         a(0, 1) := 3;
24         a(0, 2) := 2;
25         a(1, 0) := 1;
26         a(1, 1) := 0;
27         a(1, 2) := 0;
28         a(2, 0) := 1;
29         a(2, 1) := 2;
30         a(2, 2) := 2;
31
32         -- Matriu 'b'
33         -- 1 0 5
34         -- 7 5 0
35         -- 2 1 1
36         b(0, 0) := 1;
37         b(0, 1) := 0;
38         b(0, 2) := 5;
39         b(1, 0) := 7;
40         b(1, 1) := 5;
```

```
41          b(1, 2) := 0;
42          b(2, 0) := 2;
43          b(2, 1) := 1;
44          b(2, 2) := 1;
45
46          -- Sumam
47          i := 0;
48          j := 0;
49          WHILE (i <= 2) LOOP
50                  WHILE (j <= 2) LOOP
51                          c(i, j) := a(i, j) + b(i, j);
52                          puti(c(i, j));
53                          j := j + 1;
54                  END LOOP;
55                  new_line;
56                  j := 0;
57                  i := i + 1;
58          END LOOP;
59
60  END prova2;
```

8.3 Vector amb rangs negatius

8.3.1 Fitxer *prova3.lem*

```
1  -- PROVA3.lem
2  -- Vectors amb rangs negatius
3
4  PROCEDURE prova3 IS
5          TYPE rang IS NEW integer RANGE -2..5;
6          TYPE vector IS ARRAY (rang) OF integer;
7          v : vector;
8          i : rang;
9  BEGIN
10
11         i := -2;
12         WHILE (i <=5) LOOP
13                 v(i) := 7;
14                 i := i + 1;
15         END LOOP;
16
17         i := -2;
18         WHILE (i <= 5) LOOP
19                 puti(v(i));
20                 new_line;
21                 i := i + 1;
22         END LOOP;
23
24 END prova3;
```

8.4 Algoritme d'ordenació *Quicksort*

8.4.1 Fitxer *prova4.lem*

```
1  PROCEDURE qsort IS
2
3    TYPE a_range IS NEW integer RANGE 0..24;
4    v_len : CONSTANT a_range := 20;
5    TYPE a_value IS ARRAY (a_range) OF integer;
6    values : a_value;
7    s : string;
8
9    PROCEDURE omple_vector IS
10     i : a_range;
11     val : integer;
12     max : CONSTANT integer := 251;
13     tmp: integer;
14   BEGIN
15     i := 0;
16     tmp := 19;
17     WHILE i <= v_len LOOP
18       values(i) := tmp;
19
20       tmp := tmp +50;
21       tmp := tmp*tmp;
22       tmp := tmp MOD max;
23
24       i := i +1;
25         puti(tmp);
26         new_line;
27     END LOOP;
28   END omple_vector;
29
30   PROCEDURE put_vect(vect: OUT a_value) IS
31     i : a_range;
32         ax : integer;
33         s: string;
34   BEGIN
35     i := 0;
36         ax := 0;
37         s := ",  ";
38     WHILE i <= v_len LOOP
39                 ax := vect(i);
40         puti(ax);
```

```
41          puts(s);
42                  i := i+1;
43      END LOOP;
44    END put_vect;
45
46    PROCEDURE swap(left: IN a_range; right: IN a_range) IS
47    tmp : integer;
48    BEGIN
49    tmp := values(left);
50    values(left) := values(right);
51    values(right) := tmp;
52    END swap;
53
54     PROCEDURE sort(inferior: IN a_range; superior: IN a_range) IS
55       i: a_range;
56       j: a_range;
57       pivot: integer;
58       tmp: a_range;
59
60     BEGIN
61
62       IF inferior < superior THEN
63         tmp := (inferior + superior) / 2;
64         pivot := values(tmp);
65         swap(inferior, tmp);
66         i := inferior + 1;
67         j := superior;
68         WHILE i <= j LOOP
69           IF values(i) <= pivot THEN
70             i := i + 1;
71           ELSE
72             swap(i, j);
73             j := j - 1;
74           END IF;
75         END LOOP;
76         swap(inferior, j);
77         IF j > inferior THEN
78           sort(inferior, j - 1);
79         END IF;
80         IF j < superior THEN
81           sort(j + 1, superior);
82         END IF;
83       END IF;
```

```
84     END sort;
85
86 BEGIN
87   omple_vector;
88   s:= "vector desordenat";
89   puts(s);
90   new_line;
91   put_vect(values);
92   new_line;
93   sort(0, v_len);
94   s:="vector ordenat";
95   puts(s);
96   new_line;
97   put_vect(values);
98   new_line;
99 END qsort;
```

Índex