Informe Compiladors

José Ruiz Bravo, 43150039S <joseruizbravo@gmail.com>,
Biel Moyà Alcover, 43142617E <bilibiel@gmail.com>,
Álvaro Medina i Ballester, 43176576X <alvaro@comiendolimones.com>

20 d'abril 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 *(fragment)*

1.1 Atributs

1.1.1 Fitxer *decls-d_atribut.ads*

```
1  -- ------------------------------------------------
2  --  Paquet de procediments dels atributs
3  -- ------------------------------------------------
4  --  Versio  :   0.2
5  --  Autors  :   Jose Ruiz Bravo
6  --              Biel Moya Alcover
7  --              Alvaro Medina Ballester
8  -- ------------------------------------------------
9  --      En aquest fitxer tenim implementats les
10 --  assignacions de cada tipus de token al tipus
11 --  atribut que li correspon. Cal destacar
12 --  l'utilitzacio de la taula de noms en els
13 --  casos d'identificadors i strings.
14 --
15 -- ------------------------------------------------
16
17 WITH     decls.Dgenerals,
```

```
18          decls.D_Taula_De_Noms,
19          decls.Dtnode,
20          Decls.Dtdesc;
21
22  USE     decls.Dgenerals,
23          decls.D_Taula_De_Noms,
24          decls.Dtnode,
25          Decls.dtdesc;
26
27  PACKAGE decls.d_atribut IS
28
29    TYPE atribut (t : tipus_atribut := atom) IS RECORD
30        lin, col : natural;
31        CASE t IS
32            WHEN atom       => NULL;
33            WHEN a_ident    => idn : id_nom;
34            WHEN A_Lit_C | A_Lit_N | A_Lit_S
35              => val : valor;
36            WHEN OTHERS     => a   : pnode;
37        END CASE;
38    END RECORD;
39
40  END decls.d_atribut;
```

2 Anàlisi Sintàctica *(fragment)*

2.1 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
```

```
40 %token op_resta
41 %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     M1 dec_procediment
69         {creaNode_programa($$, $1, $2, programa);}
70   ;
71
72 M1:
73     {creaNode($$, tnul);}
74   ;
75
76 dec_procediment:
77     pc_procedure encap pc_is
78         declaracions
79     pc_begin
80         bloc
81     pc_end id s_final
82     {creaNode_ID($8, $8, identificador);
```

```
83       creaNode($$, $2, $4, $6, $8, procediment);}
84    ;
85
86
87  encap:
88      id
89      {creaNode_ID($$, $1, identificador);}
90    |
91      pencap s_parentesitancat
92      {Remunta($$, $1);}
93    ;
94
95  pencap:
96      pencap s_final param
97      {creaNode($$, $1, $3, pencap);}
98    |
99      id s_parentesiobert param
100     {creaNode_ID($1, $1, identificador);
101      creaNode($$, $1, $3, pencap);}
102   ;
103
104 param:
105     id s_dospunts mode id
106     {creaNode_ID($1, $1, identificador);
107      creaNode_ID($4, $4, identificador);
108      creaNode($$, $1, $3, $4, Param);}
109   ;
110
111
112 mode:
113     pc_in
114     {creanode_mode($$, entra, mode);}
115   |
116     pc_out
117     {creanode_mode($$, surt, mode);}
118   |
119     pc_in pc_out
120     {creanode_mode($$, entrasurt, mode);}
121   ;
122
123 declaracions:
124     declaracions declaracio
125     {creaNode($$, $1, $2, declaracions);}
```

```
126   |
127     {creaNode($$, tnul);}
128   ;
129
130
131 -- DECLARACIONS
132 declaracio:
133     dec_var s_final
134     {Remunta($$, $1);}
135   |
136     dec_constant s_final
137     {Remunta($$, $1);}
138   |
139     dec_tipus s_final
140     {Remunta($$, $1);}
141   |
142     dec_procediment
143     {Remunta($$, $1);}
144   ;
145
146 dec_var:
147     id c_decl_var
148     {creaNode_ID($1, $1, identificador);
149      creaNode($$, $1, $2, dvariable);}
150   ;
151
152 c_decl_var:
153     s_dospunts id
154     {creaNode_ID($2, $2, identificador);
155      remunta($$, $2);}
156   |
157     s_coma id c_decl_var
158     {creaNode_ID($2, $2, identificador);
159      creaNode($$, $2, $3, declmultvar);}
160   ;
161
162 dec_constant:
163     id s_dospunts pc_constant id s_assignacio val
164     {creaNode_ID($1, $1, identificador);
165      creaNode_ID($4, $4, identificador);
166      creaNode($$, $1, $4, $6, dconstant);}
167   ;
168
```

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

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

```
255   |
256     sentencia s_final
257      {Remunta($$, $1);}
258   ;
259
260
261 -- SENTENCIES D'INSTRUCCIONS
262 sentencia:
263     sassig
264     {Remunta($$, $1);}
265   |
266     scond
267     {Remunta($$, $1);}
268   |
269     srep
270     {Remunta($$, $1);}
271   |
272     crida_proc
273     {Remunta($$, $1);}
274   ;
275
276 -- Sentencia assignacio
277 sassig:
278     referencia s_assignacio expressio
279     {creaNode($$, $1, $3, assignacio);}
280   ;
281
282 -- Sentencia condicional
283 scond:
284     pc_if expressio pc_then
285         bloc
286     pc_end pc_if
287     {creaNode($$, $2, $4, CondicionalS);}
288   |
289     pc_if expressio pc_then
290         bloc
291     pc_else
292         bloc
293     pc_end pc_if
294     {creaNode($$, $2, $4, $6, CondicionalC);}
295   ;
296
297 -- Sentencia bucle
```

```
298 srep:
299     pc_while expressio pc_loop
300         bloc
301     pc_end pc_loop
302     {creaNode($$, $2, $4, Repeticio);}
303   ;
304
305 -- Sentencia crida a procediment
306 crida_proc:
307     referencia
308     {Remunta($$, $1);}
309   ;
310
311 referencia:
312     id
313     {creaNode_ID($$, $1, identificador);}
314   |
315     referencia s_puntrec id
316     {creaNode_ID($3, $3, identificador);
317      creaNode($$, $1, $3, referencia);}
318   |
319     pri s_parentesitancat
320     {creaNode($$, $1, fireferencia);}
321   ;
322
323 pri:
324     referencia s_parentesiobert expressio
325     {creaNode($$, $1, $3, encappri);}
326   |
327     pri s_coma expressio
328     {creaNode($$, $1, $3, pri);}
329   ;
330
331
332 -- Expressions
333 expressio:
334     expressio pc_or expressio
335     {creaNode($$, $1, $3, Unio, Expressio);}
336   |
337     expressio pc_and expressio
338     {creaNode($$, $1, $3, Interseccio, Expressio);}
339   |
340     pc_not expressio     %prec pc_not
```

```
341     {creaNode($$, $2, Negacio, ExpressioUnaria);}
342   |
343     expressio op_menor expressio
344     {creaNode($$, $1, $3, Menor, Expressio);}
345   |
346     expressio op_menorigual expressio
347     {creaNode($$, $1, $3, Menorig, Expressio);}
348   |
349     expressio op_majorigual expressio
350     {creaNode($$, $1, $3, Majorig, Expressio);}
351   |
352     expressio op_major expressio
353     {creaNode($$, $1, $3, Major, Expressio);}
354   |
355     expressio op_igual expressio
356     {creaNode($$, $1, $3, Igual, Expressio);}
357   |
358     expressio op_distint expressio
359     {creaNode($$, $1, $3, Distint, Expressio);}
360   |
361     expressio op_suma expressio
362     {creaNode($$, $1, $3, Suma, Expressio);}
363   |
364     expressio op_resta expressio
365     {creaNode($$, $1, $3, Resta, Expressio);}
366   |
367     expressio op_multiplicacio expressio
368     {creaNode($$, $1, $3, Mult, Expressio);}
369   |
370     expressio op_divisio expressio
371     {creaNode($$, $1, $3, Div, Expressio);}
372   |
373     expressio pc_mod expressio
374     {creaNode($$, $1, $3, Modul, Expressio);}
375   |
376     op_resta expressio   %prec menys_unitari
377     {creaNode($$, $2, Resta, ExpressioUnaria);}
378   |
379     s_parentesiobert expressio s_parentesitancat
380     {Remunta($$, $2);}
381   |
382     referencia
383     {Remunta($$, $1);}
```

```
384   |
385     const
386     {creaNode_VAL($$, $1, Const, 1);}
387   ;
388
389
390 %%
391
392
393 PACKAGE pk_usintactica IS
394
395     PROCEDURE yyparse;
396
397
398 END pk_usintactica;
399
400
401
402 WITH     pk_usintactica_tokens,
403          pk_usintactica_shift_reduce,
404          pk_usintactica_goto,
405          pk_ulexica_io,
406          u_lexica,
407          decls.d_arbre,
408          decls.dtnode,
409          Ada.text_IO;
410
411 USE      pk_usintactica_tokens,
412          pk_usintactica_shift_reduce,
413          pk_usintactica_goto,
414          pk_ulexica_io,
415          u_lexica,
416          decls.d_arbre,
417          decls.dtnode,
418          ada, --no llevar mai
419          ada.text_io;
420
421 PACKAGE BODY pk_usintactica IS
422     PROCEDURE YYError (e : IN string) IS
423     BEGIN
424         Put_Line(e);
425         RAISE Syntax_Error;
426     END YYError;
```

```
427 ##
428 END pk_usintactica;
```

3 Anàlisi Semàntica

3.1 Taula de simbols

3.1.1 Fitxer *decls-dtsimbols.ads*

```
1  -- ------------------------------------------------
2  --  Declaracions taula de simbols
3  -- ------------------------------------------------
4  --  Versio  :   0.1
5  --  Autors  :   Jose Ruiz Bravo
6  --              Biel Moya Alcover
7  --              Alvaro Medina Ballester
8  -- ------------------------------------------------
9  --      Declaracions dels procediments de la
10 --  taula de simbols.
11 --
12 -- ------------------------------------------------
13
14
15 WITH     decls.dtdesc,
16         decls.dgenerals,
17         decls.d_taula_de_noms,
18         Ada.Text_IO;
19
20 USE      decls.dtdesc,
21         decls.dgenerals,
22         decls.d_taula_de_noms,
23         Ada.Text_IO;
24
25
26 PACKAGE decls.dtsimbols IS
27
28     --pragma pure;
29
30     TYPE tsimbols IS LIMITED PRIVATE;
31
32     --Serveix per al joc de proves
33     TYPE cursor_idx IS NEW Rang_despl;
34     TYPE cursor_arg IS NEW Rang_despl;
35
36     -- Operacions
37     -- VERSIO 1: llenguatge simple sense estructura
```

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

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

```
124       (ts : IN OUT tsimbols;
125        id : IN id_nom;
126         d : IN descrip);
127
128 PRIVATE
129
130
131    TYPE tipus_descripcio IS RECORD
132       np : nprof;
133        d : descrip;
134        s : rang_despl;
135    END RECORD;
136
137    TYPE tipus_expansio IS RECORD
138       np : nprof;
139        d : descrip;
140       id : id_nom;
141        s : rang_despl;
142    END RECORD;
143
144    TYPE taula_ambits IS ARRAY
145      (1 .. nprof'Last) OF rang_despl;
146
147    TYPE taula_expansio IS ARRAY
148      (1 .. rang_despl'Last) OF tipus_expansio;
149
150    TYPE taula_desc IS ARRAY
151      (1 .. id_nom'Last) OF tipus_descripcio;
152
153
154    --type cursor_idx is new rang_despl;
155    --type cursor_arg is new rang_despl;
156
157    TYPE tsimbols IS RECORD
158       tdesc : taula_desc;
159        texp : taula_expansio;
160      tambit : taula_ambits;
161        prof : nprof;
162    END RECORD;
163
164
165 END decls.dtsimbols;
```

3.1.2 Fitxer *decls-dtsimbols.adb*

```
1  -- ------------------------------------------------
2  --  Procediments taula de simbols
3  -- ------------------------------------------------
4  --  Versio  :   0.1
5  --  Autors  :   Jose Ruiz Bravo
6  --              Biel Moya Alcover
7  --              Alvaro Medina Ballester
8  -- ------------------------------------------------
9  --      Procediments per tractar la taula de
10 --      sImbols:
11 --          - Taula buida
12 --          - Posa
13 --          - Consulta
14 --          - Entra bloc
15 --          - Surt bloc
16 --          - Posa camp
17 --          - Consulta camp
18 --          - Posa Index
19 --          - Primer Index
20 --          - Successor Index
21 --          - Index valid?
22 --          - Consulta Index
23 --          - Posa argument
24 --          - Primer argument
25 --          - Successor argument
26 --          - Argument valid?
27 --          - Consulta argument
28 --          - Actualitza
29 --
30 -- ------------------------------------------------
31
32 PACKAGE BODY decls.dtsimbols IS
33
34    PROCEDURE printts
35      (ts : IN tsimbols) IS
36    BEGIN
37       New_Line;
38         Put_Line("tambit ----------");
39         FOR i IN 1 .. nprof'Last LOOP
40            Put_Line("tambit["&i'img&"] := "
41                        &ts.tambit(i)'img);
```

```
42          END LOOP;
43
44          Put_Line("");
45          Put_Line("tdesc ------------");
46          FOR i IN 1 .. (id_nom'Last-985) LOOP
47             Put("tdesc["&i'img&"] := (");
48             Put(ts.tdesc(i).np'img&", ");
49             CASE ts.tdesc(i).d.td IS
50                WHEN dnula => Put("dnula, ");
51                WHEN dtipus => Put("dtipus, ");
52                WHEN dvar => Put("dvar, ");
53                WHEN dproc => Put("dproc, ");
54                WHEN dconst => Put("dconst, ");
55                WHEN dargc => Put("dargc, ");
56                WHEN dcamp => Put("dcamp, ");
57             END CASE;
58             Put(ts.tdesc(i).s'img&")");
59             New_Line;
60          END LOOP;
61
62          New_Line;
63          Put_Line("texpansio --------");
64          FOR i IN 1 .. (rang_despl'Last-9985) LOOP
65             Put("texp["&i'img&"] := (");
66             Put(ts.texp(i).np'img&", ");
67             CASE ts.texp(i).d.td IS
68                WHEN dnula => Put("dnula, ");
69                WHEN dtipus => Put("dtipus, ");
70                WHEN dvar => Put("dvar, ");
71                WHEN dproc => Put("dproc, ");
72                WHEN dconst => Put("dconst, ");
73                WHEN dargc => Put("dargc, ");
74                WHEN dcamp => Put("dcamp, ");
75             END CASE;
76             Put(ts.texp(i).id'img&", ");
77             Put(ts.texp(i).s'img&")");
78             New_Line;
79          END LOOP;
80          Put_Line("PROFUNDITAT: "&ts.prof'img);
81    END printts;
82
83
84     -- VERSIO 1: llenguatge simple sense estructura
```

```
85      -- de blocs estil Fortran.
86    PROCEDURE tbuida
87      (ts : OUT tsimbols) IS
88       nul_desc : descrip(dnula);
89    BEGIN
90         ts.prof := 1;
91         ts.tambit(ts.prof) := nul_despl;
92         FOR i IN 1 .. id_nom'Last LOOP
93            ts.tdesc(i) := (nul_nprof, nul_desc,
94                            nul_despl);
95         END LOOP;
96     END tbuida;
97
98
99     PROCEDURE posa
100      (ts : IN OUT tsimbols;
101       id : IN id_nom;
102        d : IN descrip;
103        e : OUT boolean) IS
104        idespl : rang_despl;
105    BEGIN
106        e := (ts.tdesc(id).np = ts.prof);
107        IF NOT e THEN
108            ts.tambit(ts.prof) := ts.tambit(ts.prof) + 1;
109            idespl := ts.tambit(ts.prof);
110            ts.texp(idespl) := (ts.tdesc(id).np,
111                                ts.tdesc(id).d, id, 0);
112            ts.tdesc(id) := (ts.prof, d, 0);
113        END IF;
114    END posa;
115
116
117    FUNCTION cons
118      (ts : IN tsimbols;
119       id : IN id_nom)
120      RETURN descrip IS
121    BEGIN
122        RETURN ts.tdesc(id).d;
123    END cons;
124
125
126    -- VERSIO 2: Normal, llenguatge amb blocs estil
127    -- Pascal.
```

```
128     PROCEDURE entrabloc
129       (ts : IN OUT tsimbols) IS
130     BEGIN
131         ts.prof := ts.prof + 1;
132         ts.tambit(ts.prof) := ts.tambit(ts.prof - 1);
133     END entrabloc;
134
135
136     PROCEDURE surtbloc
137       (ts : IN OUT tsimbols) IS
138         idespl1 : rang_despl;
139         idespl2 : rang_despl;
140              id : id_nom;
141     BEGIN
142         idespl1 := ts.tambit(ts.prof);
143         ts.prof := ts.prof - 1;
144         idespl2 := ts.tambit(ts.prof) + 1;
145         FOR idespl IN REVERSE idespl1 .. idespl2 LOOP
146             IF ts.texp(idespl).np > 0 THEN
147                 id := ts.texp(idespl).id;
148                 ts.tdesc(id).d := ts.texp(idespl).d;
149                 ts.tdesc(id).np := ts.texp(idespl).np;
150                 ts.tdesc(id).s := ts.texp(idespl).s;
151             END IF;
152         END LOOP;
153     END surtbloc;
154
155
156     -- VERSIO 3: Blocs mes records.
157     PROCEDURE posacamp
158       (ts : IN OUT tsimbols;
159       idr : IN id_nom;
160       idc : IN id_nom;
161         d : IN descrip;
162         e : OUT boolean) IS
163            des : descrip;
164             td : descriptipus;
165              p : rang_despl;
166        itdespl : rang_despl;
167     BEGIN
168         des := ts.tdesc(idr).d;
169         IF des.td /= dtipus THEN e := TRUE; END IF;
170
```

```
171         td := des.dt;
172         IF td.tt /= tsrec THEN e := TRUE; END IF;
173
174         p := ts.tdesc(idr).s;
175         WHILE p /= 0 AND THEN ts.texp(p).id /= idc LOOP
176             p := ts.texp(p).s;
177         END LOOP;
178
179         e := (p /= 0);
180         IF NOT e THEN
181             ts.tambit(ts.prof) := ts.tambit(ts.prof) + 1;
182             itdespl := ts.tambit(ts.prof);
183             ts.texp(itdespl) := (nul_nprof, d, idc,
184                                  ts.tdesc(idr).s);
185             ts.tdesc(idr).s := itdespl;
186         END IF;
187     END posacamp;
188
189
190     FUNCTION conscamp
191       (ts : IN tsimbols;
192       idr : IN id_nom;
193       idc : IN id_nom) RETURN descrip IS
194                d : descrip;
195              td : tdescrip;
196               p : rang_despl;
197         descnula : descrip(dnula);
198     BEGIN
199         d := ts.tdesc(idr).d;
200         td := d.td;
201         p := ts.tdesc(idr).s;
202         WHILE p /= 0 AND THEN ts.texp(p).id /= idc LOOP
203             p := ts.texp(p).s;
204         END LOOP;
205
206         IF p = 0 THEN
207             RETURN descnula;
208         ELSE
209             RETURN ts.texp(p).d;
210         END IF;
211     END conscamp;
212
213
```

```
214     -- VERSIO 4: Arrays.
215     PROCEDURE posa_idx
216       (ts : IN OUT tsimbols;
217       ida : IN id_nom;
218       idi : IN id_nom;
219         e : OUT boolean) IS
220             d : descrip;
221            dt : descriptipus;
222             p : rang_despl;
223            pp : rang_despl;
224        idespl : rang_despl;
225     BEGIN
226        E := False;
227        d := ts.tdesc(ida).d;
228        IF d.td /= dtipus THEN e := TRUE; END IF;
229        dt := d.dt;
230        IF dt.tt /= tsarr THEN e := TRUE; END IF;
231
232        p := ts.tdesc(ida).s;
233        pp := 0;
234        WHILE p /= 0 LOOP -- Comprovar el 0
235            pp := p;
236            p := ts.texp(p).s;
237        END LOOP;
238
239        ts.tambit(ts.prof) := ts.tambit(ts.prof) + 1;
240        idespl := ts.tambit(ts.prof);
241        ts.texp(idespl) := (nul_nprof, (td => dnula),
242                            idi, 0);
243
244        IF pp /= 0 THEN
245           ts.texp(pp).s := idespl;
246        ELSE
247           ts.tdesc(ida).s := idespl;
248        END IF;
249     END posa_idx;
250
251
252     FUNCTION primer_idx
253       (ts : IN tsimbols;
254       ida : IN id_nom) RETURN cursor_idx IS
255     BEGIN
256         RETURN cursor_idx(ts.tdesc(ida).s);
```

```
257     END primer_idx;
258
259
260     FUNCTION idx_valid
261       (ci : IN cursor_idx) RETURN boolean IS
262     BEGIN
263         RETURN ci > 0;
264     END idx_valid;
265
266
267     FUNCTION succ_idx
268       (ts : IN tsimbols;
269        ci : IN cursor_idx) RETURN cursor_idx IS
270     BEGIN
271         IF idx_valid(ci) THEN
272            RETURN cursor_idx(ts.texp(rang_despl(ci)).s);
273         ELSE
274            RETURN 0; --Excepcio
275         END IF;
276     END succ_idx;
277
278
279     FUNCTION cons_idx
280       (ts : IN tsimbols;
281        ci : IN cursor_idx) RETURN id_nom IS
282     BEGIN
283         RETURN  ts.texp(rang_despl(ci)).id;
284     END cons_idx;
285
286
287     PROCEDURE posa_arg
288       (ts : IN OUT tsimbols;
289       idp : IN id_nom;
290       ida : IN id_nom;
291        da : IN descrip;
292         e : OUT boolean) IS
293         d : descrip;
294         p : rang_despl;
295        pp : rang_despl;
296    idespl : rang_despl;
297     BEGIN
298         e:= false;
299         d := ts.tdesc(idp).d;
```

```
300         IF d.td /= dproc THEN e := TRUE; END IF;
301
302         p := ts.tdesc(idp).s;
303         pp := 0;
304         WHILE p /= 0 LOOP
305             pp := p;
306             p := ts.texp(p).s;
307         END LOOP;
308
309         ts.tambit(ts.prof) := ts.tambit(ts.prof) + 1;
310         idespl := ts.tambit(ts.prof);
311         ts.texp(idespl) := (nul_nprof, da, ida, 0);
312
313         IF pp /= 0 THEN
314            ts.texp(pp).s := idespl;
315         ELSE
316            ts.tdesc(idp).s := idespl;
317         END IF;
318     END Posa_Arg;
319
320
321     FUNCTION Primer_Arg
322       (Ts : IN Tsimbols;
323       Idp : IN Id_Nom) RETURN Cursor_Arg IS
324     BEGIN
325        RETURN cursor_arg(ts.tdesc(idp).s);
326     END Primer_Arg;
327
328
329     FUNCTION Succ_Arg
330       (ts : IN tsimbols;
331        ca : IN cursor_arg) RETURN cursor_arg IS
332     BEGIN
333         IF arg_valid(ca) THEN
334            RETURN cursor_arg(ts.texp(rang_despl(ca)).s);
335         ELSE
336            RETURN 0; --Excepcio
337         END IF;
338     END Succ_Arg;
339
340
341     FUNCTION Arg_Valid
342       (Ca : IN Cursor_Arg) RETURN Boolean IS
```

```
343     BEGIN
344        RETURN Ca > 0;
345     END Arg_Valid;
346
347
348     PROCEDURE cons_arg
349       (ts : IN tsimbols;
350        ca : IN cursor_arg;
351       ida : OUT id_nom;
352        Dn : OUT Descrip) IS
353     BEGIN
354        Ida := ts.texp(rang_despl(ca)).id;
355        Dn := Ts.Texp(Rang_Despl(Ca)).D;
356     END Cons_Arg;
357
358
359     PROCEDURE Actualitza
360       (Ts : IN OUT Tsimbols;
361        Id : IN Id_Nom;
362         D : IN Descrip) IS
363     BEGIN
364        Ts.Tdesc(id).D := D;
365     END Actualitza;
366
367
368 END decls.dtsimbols;
```

3.2 Descripció

3.2.1 Fitxer *decls-dtdesc.ads*

```
1  -- ------------------------------------------------
2  --  Paquet de declaracions del tipus descripcio
3  -- ------------------------------------------------
4  --  Versio  :   0.1
5  --  Autors  :   Jose Ruiz Bravo
6  --              Biel Moya Alcover
7  --              Alvaro Medina Ballester
8  -- ------------------------------------------------
9  --      Declaracions del tipus descripcio.
10 --
11 -- ------------------------------------------------
12
13 WITH     decls.dgenerals,
14          decls.d_taula_de_noms;
15
16 USE      decls.dgenerals,
17          decls.d_taula_de_noms;
18
19 PACKAGE decls.dtdesc IS
20
21     --pragma pure;
22
23     -- Representa tambit
24     max_nprof : CONSTANT integer := 10;
25     TYPE nprof IS NEW integer
26       RANGE 0 .. max_nprof;
27     nul_nprof : CONSTANT nprof := 0;
28
29     TYPE despl IS NEW natural;
30
31     -- Representa texpansio
32     TYPE rang_despl IS NEW integer
33       RANGE 0 .. (max_id * max_nprof);
34     nul_despl : CONSTANT rang_despl := 0;
35
36     TYPE tdescrip IS
37       (dnula,
38        dconst,
39        dvar,
40        dtipus,
```

```
41          dproc,
42          dcamp,
43          dargc);
44
45      TYPE tipussubjacent IS
46        (tsbool,
47          tscar,
48          tsent,
49          tsrec,
50          tsarr,
51          tsnul);
52
53      TYPE descriptipus (tt: tipussubjacent := tsnul) IS
54          RECORD
55              ocup : despl;
56              CASE tt IS
57                  WHEN tsbool | tscar | tsent =>
58                     linf, lsup : valor;
59                  WHEN tsarr  => tcamp : id_nom;
60                  WHEN tsrec | tsnul  => NULL;
61              END CASE;
62          END RECORD;
63
64      TYPE descrip (td : tdescrip := dnula) IS
65          RECORD
66              CASE td IS
67                  WHEN dnula  => NULL;
68                  WHEN dtipus => dt: descriptipus;
69                  WHEN dvar   => tr: id_nom;
70                                 nv: num_var;
71                  WHEN dproc  => np: num_proc;
72                  WHEN dconst => tc: id_nom;
73                                 vc: valor;
74                                 --Nvc: Num_Var;
75                  WHEN dargc  => nvarg: num_var;
76                                  targ: id_nom;
77                  WHEN dcamp  => tcamp: id_nom;
78                                   dsp: rang_despl;
79              END CASE;
80          END RECORD;
81
82
83  END decls.dtdesc;
```

3.3 Comprovació de tipus

3.3.1 Fitxer *decls-dtnode.ads*

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

```
41        M1,
42        Repeticio, --
43        CondicionalS, --
44        CondicionalC, --
45        Expressio, --
46        ExpressioUnaria, -- Not E, -E
47        Pencap,
48        Procediment, -- Fe = encap, Fc = declaracions, Fd = bloc
49        Dvariable, --
50        Dconstant, --
51        Dcoleccio, --
52        Dregistre, --
53        Dencapregistre, --
54        Dsubrang, --
55        Identificador,
56        Const, --
57        Declaracions, --
58        Bloc,
59        Assignacio,
60        Referencia, --
61        Pri,
62        Param,
63        Pcoleccio, --
64        Pdimcoleccio, --
65        Declmultvar, --
66        Tnul,
67        Mode,
68        Encappri,
69        Firecord,
70        Fireferencia);
71
72    TYPE node (Tipus : Tipusnode := tnul) IS RECORD
73       CASE Tipus IS
74          WHEN m1 | tnul  => NULL;
75
76          WHEN programa | repeticio | condicionalS
77            | declaracions | bloc | assignacio | pri
78            | dcoleccio | Pdimcoleccio | Referencia
79            | pcoleccio | dvariable
80            | Declmultvar | encappri | Pencap => fe1, fd1: pnode;
81
82          WHEN CondicionalC | dconstant | dregistre
83            | Dencapregistre | Param => fe2, fc2, fd2: pnode;
```

```
84
85          WHEN expressio => fe3, fd3: pnode;
86                            op3: operacio;
87
88          WHEN ExpressioUnaria => f4: pnode;
89                                  op4: operacio;
90
91          WHEN procediment | dsubrang => fe5, fc5, fd5, fid5: pnode;
92
93          WHEN identificador => id12 : Id_Nom;
94                                l1, c1 : natural;
95
96          WHEN Firecord | Fireferencia => f6 : pnode;
97
98          WHEN const => val : valor;
99             l2, c2 : natural;
100            Tconst : Tipus_Atribut;
101
102         WHEN Mode => M12 : Mmode;
103
104       END CASE;
105    END RECORD;
106
107 END Decls.Dtnode;
```

3.3.2 Fitxer *decls-d_ arbre.ads*

```
WITH     Decls.Dgenerals,
         Decls.Dtnode,
         decls.D_Taula_De_Noms,
         decls.d_atribut,
         Ada.text_io;

USE      Decls.Dgenerals,
         Decls.Dtnode,
         decls.D_Taula_De_Noms,
         decls.d_atribut,
         Ada.text_io;

PACKAGE decls.d_arbre IS

   Arbre : Pnode;
   PROCEDURE Abuit
     (P : OUT pnode);

   PROCEDURE creaNode_programa
     (p : OUT atribut;
      fe, fd : IN atribut;
      tn : IN Tipusnode);

   PROCEDURE creaNode
     (p : OUT atribut;
      fe, fd : IN atribut;
      tn : IN Tipusnode);

   PROCEDURE creaNode
     (p : OUT atribut;
      fe, fc, fd : IN atribut;
      tn : IN Tipusnode);

   PROCEDURE creaNode
     (p : OUT atribut;
      fe, fce, fc, fd : IN atribut;
      tn : IN Tipusnode);

     PROCEDURE creaNode
     (p : OUT atribut;
      f : IN atribut;
```

```
42        tn : IN Tipusnode);
43
44    PROCEDURE creaNode
45      (p : OUT atribut;
46       fe, fd: IN atribut;
47       op : IN operacio;
48       tn : IN Tipusnode);
49
50    PROCEDURE creaNode
51      (p : OUT atribut;
52       f : IN atribut;
53       op : IN operacio;
54       tn : IN Tipusnode);
55
56    PROCEDURE CreaNode_ID
57      (p : OUT atribut;
58       id : IN atribut;
59       tn : IN Tipusnode);
60
61    PROCEDURE CreaNode_VAL
62      (p : OUT atribut;
63       a : IN atribut;
64       tn : IN Tipusnode;
65       S : IN Valor);
66
67    PROCEDURE CreaNode_MODE
68      (P : OUT Atribut;
69       M : IN Mmode;
70       Tn : IN Tipusnode);
71
72    PROCEDURE creaNode
73      (P : OUT Atribut;
74       Tn : IN Tipusnode);
75
76    PROCEDURE Remunta
77      (P : OUT Atribut;
78       A : IN Atribut);
79
80    PROCEDURE Cons_Tnode
81      (P : IN Pnode;
82      Tn : OUT Tipusnode);
83
84 END decls.d_arbre;
```

3.3.3 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       fe, fd : IN atribut;
13       tn : IN Tipusnode) IS
14       paux : pnode;
15    BEGIN
16       paux := NEW node(tn);
17       paux.fe1 := fe.a;
18       paux.fd1 := fd.a;
19       p := (nodeArbre, 0, 0, paux);
20       arbre := paux;
21    END creaNode_programa;
22
23
24    PROCEDURE creaNode
25      (p : OUT atribut;
26       fe,fd : IN atribut;
27       tn : IN Tipusnode) IS
28       paux : pnode;
29    BEGIN
30       paux := NEW node(tn);
31       paux.fe1 := fe.a;
32       paux.fd1 := fd.a;
33       p := (nodeArbre, 0, 0, paux);
34    END creaNode;
35
36
37    PROCEDURE CreaNode
38      (p : OUT atribut;
39       fe,fc,fd : IN atribut;
40       tn : IN Tipusnode) IS
41         paux : pnode;
```

```
42    BEGIN
43       paux := NEW node(tn);
44       paux.fe2 := fe.a;
45       paux.fd2 := fd.a;
46       paux.fc2 := fc.a;
47       p := (nodeArbre, 0, 0, paux);
48    END creaNode;
49
50
51    PROCEDURE creaNode
52      (p : OUT atribut;
53      fe, fd : IN atribut;
54      op : IN operacio;
55      tn : IN Tipusnode) IS
56        paux : pnode;
57    BEGIN
58       paux := NEW node(tn);
59       paux.fe3 := fe.a;
60       paux.fd3 := fd.a;
61       paux.op3 := op;
62       p := (nodeArbre, 0, 0, paux);
63    END creaNode;
64
65
66    PROCEDURE CreaNode
67      (p : OUT atribut;
68       f : IN atribut;
69      op : IN operacio;
70      tn : IN Tipusnode) IS
71        paux : pnode;
72    BEGIN
73       paux := NEW node(tn);
74       paux.f4 := f.a;
75       paux.op4 := op;
76       p := (nodeArbre, 0, 0, paux);
77    END creaNode;
78
79
80    PROCEDURE CreaNode
81       (p : OUT atribut;
82        fe, fce, fc, fd : IN atribut;
83        tn : IN Tipusnode) IS
84         paux : pnode;
```

```
85      BEGIN
86         paux := NEW node(tn);
87         paux.fe5 := fe.a;
88         paux.fc5 := fce.a;
89         paux.fd5 := fc.a;
90         paux.fid5 := fd.a;
91         p := (nodeArbre, 0, 0, paux);
92      END creaNode;
93
94      PROCEDURE creaNode
95       (p : OUT atribut;
96        f : IN atribut;
97        tn : IN Tipusnode) IS
98          paux : pnode;
99      BEGIN
100        paux := NEW node(tn);
101        paux.f6 := f.a;
102        p := (nodeArbre, 0, 0, paux);
103     END creaNode;
104
105     -- Crea node per identificadors
106     PROCEDURE CreaNode_ID
107       (p : OUT atribut;
108       id : IN atribut;
109       tn : IN Tipusnode) IS
110        paux : pnode;
111     BEGIN
112        paux := NEW node(tn);
113        paux.id12 := id.idn;
114        paux.l1 := id.lin;
115        paux.c1 := id.col;
116        p := (nodeArbre, 0, 0, paux);
117     END CreaNode_ID;
118
119
120     PROCEDURE CreaNode_VAL
121       (p : OUT atribut;
122        a : IN atribut;
123       tn : IN Tipusnode;
124        S : IN Valor) IS
125        paux : pnode;
126     BEGIN
127        paux := NEW node(tn);
```

```
128         IF S = 0 THEN
129            paux.val := A.Val*(-1);
130         ELSE
131            Paux.Val := A.Val;
132         END IF;
133         Paux.Tconst := A.T;
134         paux.l2 := a.lin;
135         paux.c2 := a.col;
136         p := (nodeArbre, 0, 0, paux);
137      END CreaNode_VAL;
138
139
140      PROCEDURE Creanode_MODE
141        (P : OUT Atribut;
142         M : IN mmode;
143         Tn : IN Tipusnode) IS
144         Paux : Pnode;
145      BEGIN
146         Paux := NEW Node(Tn);
147         Paux.M12 := M;
148         P := (NodeArbre, 0, 0, Paux);
149      END Creanode_Mode;
150
151
152      PROCEDURE creaNode
153        (P : OUT Atribut;
154        Tn : IN Tipusnode) IS
155        Paux : Pnode;
156      BEGIN
157        Paux := NEW Node(tn);
158        P := (NodeArbre, 0, 0, Paux);
159      END creaNode;
160
161
162      PROCEDURE Remunta
163        (P : OUT Atribut;
164         A : IN Atribut) IS
165      BEGIN
166         P := A;
167      END Remunta;
168
169
170      PROCEDURE Cons_Tnode
```

```
171         (P : IN Pnode;
172          Tn : OUT Tipusnode) IS
173       BEGIN
174          Tn := P.Tipus;
175       END Cons_Tnode;
176
177 END decls.d_arbre;
```

3.3.4 Fitxer *decls-ctipus.ads*

```
1  WITH Ada.Text_Io,
2    Decls.Dgenerals,
3    Decls.Dtnode,
4    Decls.D_Arbre,
5    Decls.D_Taula_De_Noms,
6    Decls.D_Atribut,
7    Decls.Dtsimbols,
8    Decls.Dtdesc,
9    Decls.Missatges;
10
11 USE Ada.Text_Io,
12   Decls.Dgenerals,
13   Decls.Dtnode,
14   Decls.D_Arbre,
15   Decls.D_Taula_De_Noms,
16   Decls.D_Atribut,
17   Decls.Dtsimbols,
18   Decls.Dtdesc,
19   Decls.Missatges;
20
21
22 PACKAGE Decls.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    -- Taula de simbols
50    PROCEDURE Inicia_Enter;
51    PROCEDURE Inicia_Boolea;
52
53    -- Comprovacio de tipus
54    PROCEDURE Inicia_analisi;
55
56    PROCEDURE Ct_Programa
57      (A : IN Pnode);
58
59    PROCEDURE Ct_M1;
60
61    PROCEDURE Ct_Decprocediment
62      (A : IN Pnode);
63
64    PROCEDURE Ct_Encap
65      (A : IN Pnode;
66       I : OUT Id_Nom);
67
68    PROCEDURE Ct_Pencap
69      (A : IN Pnode;
70       I : OUT Id_Nom);
71
72    PROCEDURE Ct_Param
73      (A : IN Pnode;
74           I : IN Id_Nom);
75
76    PROCEDURE Ct_Declaracions
77      (A : IN Pnode);
78
79    PROCEDURE Ct_Decvar
80      (A : IN Pnode);
81
82    PROCEDURE Ct_Declsvar
83      (A : IN Pnode;
84       Idtipus : OUT Id_nom);
```

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

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

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

```
214         Idtipus : OUT Id_Nom;
215         Idbase : OUT Id_Nom);
216
217     PROCEDURE Ct_Ref_Pri
218       (A : IN Pnode;
219        T : OUT Tipussubjacent;
220        Id : OUT Id_Nom;
221        It_Idx : OUT Cursor_Idx);
222
223     PROCEDURE Ct_Ref_Pri
224       (A : IN Pnode;
225        T : OUT Tipussubjacent;
226        It_Arg : OUT Cursor_Arg);
227
228 PRIVATE
229
230     --Provisional
231     Ts : Tsimbols;
232     Tn : Taula_De_Noms;
233     nv : num_var;
234     np : num_proc;
235
236 END Decls.Ctipus;
```

3.3.5 Fitxer *decls-ctipus.adb*

```
1  WITH U_Lexica;
2
3  USE U_Lexica;
4
5
6  PACKAGE BODY Decls.Ctipus IS
7
8     PROCEDURE mt_atom
9       (l, c : IN natural;
10           a : OUT atribut) IS
11    BEGIN
12        a := (atom, l, c);
13    END mt_atom;
14
15
16    PROCEDURE mt_identificador
17      (l, c : IN natural;
18           s : IN string;
19           a : OUT atribut) IS
20       id : id_nom;
21    BEGIN
22        id := id_nul;
23        posa_id(tn, id, s);
24        a := (a_ident, l, c, id);
25    END mt_identificador;
26
27
28    PROCEDURE mt_string
29      (l, c : IN natural;
30           s : IN string;
31           a : OUT atribut) IS
32        id : rang_tcar;
33    BEGIN
34        posa_str(tn, id, s);
35        a := (A_Lit_S, l, c, valor(id));
36    END mt_string;
37
38
39    PROCEDURE mt_caracter
40      (l, c : IN natural;
41        car : IN string;
```

```
42          a : OUT atribut) IS
43    BEGIN
44       a := (A_Lit_C, l, c, valor(car'First+1));
45    END mt_caracter;
46
47
48    PROCEDURE mt_numero
49      (l, c : IN natural;
50          s : IN string;
51          a : OUT atribut) IS
52    BEGIN
53        a := (A_Lit_N, l, c, valor(Integer'value(s)));
54    END mt_numero;
55
56
57    -- Taula de simbols
58    PROCEDURE Inicia_Enter IS
59       D : Descrip;
60       Dt : Descriptipus;
61       Idn : Id_Nom;
62       E : Boolean;
63    BEGIN
64       posa_id(tn, idn, "integer");
65       dt := (tsent, 4, valor(integer'first),
66              valor(integer'last));
67       d := (dtipus,dt);
68       posa(ts, idn, d, e);
69
70    END Inicia_Enter;
71
72
73    PROCEDURE Inicia_Boolea IS
74       D : Descrip;
75       Dt : Descriptipus;
76       Idb, Idt, Idf : Id_Nom;
77       E : Boolean;
78    BEGIN
79       Posa_Id(Tn, Idb, "boolean");
80       Dt := (Tsbool, 4, -1, 0);
81       D := (Dtipus, Dt);
82       Posa(Ts, Idb, D, E);
83
84       Posa_Id(Tn, Idt, "true");
```

```
85        Nv := Nv + 1;
86        --D := (Dconst, Idb, -1, Nv);
87        D := (Dconst, Idb, -1);
88        Posa(Ts, Idt, D, E);
89
90        Posa_Id(Tn, Idf, "false");
91        Nv := Nv + 1;
92        --D := (Dconst, Idb, 0, Nv);
93        D := (Dconst, Idb, 0);
94        Posa(Ts, Idf, D, E);
95     END Inicia_Boolea;
96
97
98     PROCEDURE Inicia_Caracter IS
99        D : Descrip;
100       Dt : Descriptipus;
101       Idn : Id_Nom;
102       E : Boolean;
103    BEGIN
104       Posa_Id(Tn, Idn, "character");
105       Dt := (Tscar, 4, Valor(Character'Pos(Character'First)),
106              Valor(Character'Pos(Character'Last)));
107       D := (Dtipus, Dt);
108       Posa(Ts, Idn, D, E);
109    END Inicia_Caracter;
110
111
112    PROCEDURE Inicia_analisi IS
113    BEGIN
114       nv := 0;
115       np := 0;
116       Tbuida(Tn);
117       Tbuida(Ts);
118       Inicia_Enter;
119       Inicia_Boolea;
120       Inicia_Caracter;
121       Obre_Fitxer;
122    END Inicia_analisi;
123
124
125    -- Procediments interns
126    PROCEDURE Posa_Idvar
127      (Idvar : IN Id_Nom;
```

```
128         Idtipus : IN Id_Nom;
129         L, C : IN Natural;
130         E : OUT Boolean) IS
131         Tassig : Descrip;
132      BEGIN
133         nv := nv + 1;
134         Tassig := (Dvar, Idtipus, Nv);
135         Posa(Ts, Idvar, Tassig, E);
136         IF E THEN
137            error(id_existent, L, C, cons_nom(tn, Idvar));
138         END IF;
139      END Posa_Idvar;
140
141
142      -- Comprovacio de tipus
143      PROCEDURE Ct_Programa
144        (A : IN Pnode) IS
145         d : Descrip;
146         Idproc : Id_nom RENAMES A.Fd1.Fid5.Id12;
147           ida : cursor_arg;
148      BEGIN
149         Ct_M1;
150         Ct_Decprocediment(A.Fd1);
151         ida := primer_arg(ts,Idproc);
152         IF (arg_valid(ida)) THEN
153            error(paramsPprincipal, cons_nom(tn, Idproc));
154         END IF;
155
156         Tanca_Fitxer;
157      END Ct_Programa;
158
159
160      PROCEDURE Ct_M1 IS
161      BEGIN
162         put_line("(DEBUG) M1 necessari per generacio de codi 3@");
163      END Ct_M1;
164
165
166      PROCEDURE Ct_Decprocediment
167        (A : IN Pnode) IS
168
169         Encap : Pnode RENAMES A.Fe5;
170         Decls : Pnode RENAMES A.Fc5;
```

```
171        Bloc : Pnode RENAMES A.Fd5;
172        Id : Pnode RENAMES A.Fid5;
173        Id_Inf : Id_Nom RENAMES A.Fid5.Id12;
174        Id_Sup : Id_Nom;
175        Tdecls : Tipusnode;
176
177     BEGIN
178        Put_line("CT_Decprocediment");
179        Ct_Encap(Encap, Id_Sup);
180
181        IF Id_Inf /= Id_Sup THEN
182           error(idProgDiferents, A.Fid5.l1, A.Fid5.c1,
183                 cons_nom(tn, Id_Sup));
184        END IF;
185
186        Cons_Tnode(Decls, Tdecls);
187        IF Tdecls /= Tnul THEN
188           Ct_Declaracions(Decls);
189        END IF;
190        Ct_Bloc(Bloc);
191        Surtbloc(Ts);
192
193     END Ct_Decprocediment;
194
195
196     PROCEDURE Ct_Encap
197       (A : IN Pnode;
198        I : OUT Id_Nom) IS
199
200        Tproc : Descrip;
201        E : Boolean;
202        Idx_Arg : cursor_arg;
203        ida : id_nom;
204        dn : descrip;
205
206     BEGIN
207        Put_line("CT_ENCAP");
208        IF A.Tipus = Pencap THEN
209           Ct_Pencap(A, I);
210           entrabloc(Ts);
211           idx_Arg := primer_arg(ts, I);
212           WHILE arg_valid(idx_Arg) LOOP
213              cons_arg(ts, idx_arg, ida, dn);
```

```
214             posa(ts, ida, dn, e);
215             IF E THEN
216                error(enregArg, 3, 3, cons_nom(tn, ida));
217             END IF;
218             idx_Arg := succ_arg(ts, idx_arg);
219          END LOOP;
220       ELSE
221          I := A.Id12;
222          np := np + 1;
223          Tproc := (Dproc, np);
224          Posa(Ts, I, Tproc, E);
225          IF E THEN
226             error(id_existent, A.l1, A.c1, cons_nom(tn, I));
227          END IF;
228          Entrabloc(Ts);
229       END IF;
230
231    END Ct_Encap;
232
233
234    PROCEDURE Ct_Pencap
235      (A : IN Pnode;
236       I : OUT Id_Nom) IS
237
238       Param : Pnode RENAMES A.Fd1;
239       Fesq : Pnode RENAMES A.Fe1;
240       Tproc : Descrip;
241       E : Boolean;
242
243    BEGIN
244       Put_line("CT_pencap: ");
245       IF Fesq.Tipus = Identificador THEN
246          np := np + 1;
247          Tproc := (Dproc, np);
248          Posa(Ts, Fesq.Id12, Tproc, E);
249          IF E THEN
250             error(id_existent, Fesq.l1, Fesq.c1,
251                   cons_nom(tn, Fesq.Id12));
252          END IF;
253          I := fesq.Id12;
254       ELSE
255          Ct_Pencap(Fesq, I);
256       END IF;
```

```
257       Ct_Param(Param, I);
258    END Ct_Pencap;
259
260
261    PROCEDURE Ct_Param
262      (A : IN Pnode;
263       I : IN Id_Nom) IS
264
265       idPar : id_nom RENAMES A.Fe2.id12;
266       mArg : mMode RENAMES A.Fc2.M12;
267       idTipus : id_nom RENAMES A.Fd2.id12;
268       d : Descrip;
269       dArg : Descrip;
270       E : boolean;
271
272    BEGIN
273
274       Put_line("CT_Param");
275       d := cons(ts, idtipus);
276       IF d.td /= dtipus THEN
277          error(tipusParam, A.Fd2.l1, A.Fd2.c1,
278                cons_nom(tn, idtipus));
279       END IF;
280
281       CASE mArg IS
282          WHEN Surt | Entrasurt =>
283             nv := nv + 1;
284             dArg := (dvar, idtipus, nv);
285          WHEN Entra =>
286             nv := nv + 1;
287             dArg := (dargc, nv, idtipus);
288          WHEN OTHERS =>
289             NULL;
290       END CASE;
291
292       posa_arg(ts, I, idPar, dArg, E);
293       IF E THEN
294          error(enregArg, A.Fe2.l1, A.Fe2.c1,
295                cons_nom(tn, idPar));
296       END IF;
297
298    END Ct_Param;
299
```

```
300
301   PROCEDURE Ct_Declaracions
302     (A : IN Pnode) IS
303
304      Decl : Pnode RENAMES A.Fd1;
305      Decls : Pnode RENAMES A.Fe1;
306      Tnode : Tipusnode;
307      Idrec : Id_Nom;
308      Ocup  : Despl;
309
310   BEGIN
311      Put_line("CT_DECLARACIONS");
312      IF Decls.Tipus = Declaracions THEN
313         Ct_Declaracions(Decls);
314      END IF;
315
316      Cons_Tnode(Decl, Tnode);
317      CASE Tnode IS
318         WHEN Dvariable   =>
319            Ct_Decvar(Decl);
320         WHEN Dconstant   =>
321            Ct_Decconst(Decl);
322         WHEN Dcoleccio   =>
323            Ct_Deccol(Decl);
324         WHEN Dregistre | Dencapregistre | Firecord =>
325            Ocup := 0;
326            Ct_Decregistre(Decl, Idrec,Ocup);
327         WHEN Dsubrang    =>
328            Ct_Decsubrang(Decl);
329         WHEN Procediment =>
330            Ct_Decprocediment(Decl);
331         WHEN OTHERS =>
332            Put_Line("ERROR CT_Declaracions:(DEBUG)tipus "&
333                     "declarat inexistent "&Tnode'Img);
334      END CASE;
335
336   END Ct_Declaracions;
337
338
339   PROCEDURE Ct_Decvar
340     (A : IN Pnode) IS
341
342      Dvariable : Pnode RENAMES A.Fd1;
```

```
343         Id : Id_Nom RENAMES A.Fe1.Id12;
344         L : Natural RENAMES A.Fe1.L1;
345         C : Natural  RENAMES A.Fe1.C1;
346         Tassig : Descrip;
347         Idtipus : Id_nom;
348         E : Boolean;
349
350      BEGIN
351         Put_line("CT_DECVAR");
352         Ct_Declsvar(Dvariable, Idtipus);
353         Posa_Idvar(Id, Idtipus, L, C, E);
354      END Ct_Decvar;
355
356
357      PROCEDURE Ct_Declsvar
358        (A : IN Pnode;
359         Idtipus : OUT Id_Nom) IS
360
361         Tnode : Tipusnode RENAMES A.Tipus;
362         E : Boolean;
363         Tdecl : Descrip;
364
365      BEGIN
366         Put_line("CT_DECLSVAR");
367         IF Tnode = Identificador THEN
368            Tdecl := Cons(Ts, A.Id12);
369            IF (Tdecl.Td /= Dtipus) THEN
370               error(tipusInexistent, A.l1, A.c1,
371                     cons_nom(tn, A.Id12));
372            END IF;
373            Idtipus := A.Id12;
374
375         ELSIF Tnode = Declmultvar THEN
376            Ct_Declsvar(A.Fd1, Idtipus);
377            Put_Line("CT_DECLSVAR:(DEBUG) diferents "&
378                     "variables amb mateix tipus...");
379            Posa_Idvar(A.Fd1.Id12, Idtipus, A.Fd1.L1,
380                     A.Fd1.C1, E);
381         END IF;
382
383      END Ct_Declsvar;
384
385
```

```
386   PROCEDURE Ct_Decconst
387     (A : IN Pnode) IS
388
389       Id : Id_Nom RENAMES A.Fe2.Id12;
390       Idtipus : Id_Nom RENAMES A.Fc2.Id12;
391       Val : Pnode RENAMES A.Fd2;
392       E : Boolean;
393       Tdecl : Descrip;
394       Tconst : Descrip;
395
396       -- variables per la crida a expressio
397       Tsubj : Tipussubjacent;
398       Ids : Id_Nom;
399       L, C : Natural := 0;
400
401   BEGIN
402
403       Tdecl := Cons(Ts, Idtipus);
404       IF (Tdecl.Td /= Dtipus) THEN
405          error(tipusInexistent, A.Fc2.l1, A.Fc2.c1,
406                cons_nom(tn, Idtipus));
407       ELSE
408          Ct_Constant(Val, Tsubj, Ids, L, C);
409          IF (Tsubj /= Tdecl.Dt.Tt) THEN
410             error(tipusSubDiferents, A.Fc2.l1, A.Fc2.c1,
411                   cons_nom(tn, Idtipus));
412          END IF;
413
414          IF (Val.Val < Tdecl.Dt.Linf) OR
415            (Val.Val > Tdecl.Dt.Lsup) THEN
416             error(rang_sobrepassat, A.Fe2.l1, A.Fe2.c1,
417                   cons_nom(tn, Id));
418          END IF;
419
420          Tconst := (dconst, IdTipus, Val.val);
421          Posa(Ts, Id, Tconst, E);
422          Put_Line("CT_CONST: (DEBUG)El valor de la "&
423                     "constant es: "&Val.val'img);
424          IF E THEN
425             error(id_existent, A.Fe2.l1, A.Fe2.c1,
426                   cons_nom(tn, Id));
427          END IF;
428       END IF;
```

```
429
430   END Ct_Decconst;
431
432
433   PROCEDURE Ct_Deccol
434     (A : IN Pnode) IS
435
436      Darray : Descrip;
437      Dtarray : Descrip;
438      Fesq : Pnode RENAMES A.Fe1;
439      Idtipus_Array : Id_Nom RENAMES A.Fd1.Id12;
440      Idarray : Id_Nom;
441      Ncomponents : Despl;
442
443   BEGIN
444      Dtarray := Cons(Ts, Idtipus_Array);
445      IF (Dtarray.Td /= Dtipus) THEN
446         error(tipusInexistent, A.Fd1.l1, A.Fd1.c1,
447               cons_nom(tn, Idtipus_Array));
448      ELSE
449         Ct_Pcoleccio(Fesq, Idtipus_Array, Idarray,
450                      Ncomponents);
451         Darray := Cons(Ts, Idarray);
452         Darray.Dt.Tcamp := Idtipus_Array;
453         Darray.Dt.Ocup := Ncomponents * Dtarray.Dt.Ocup;
454         Actualitza(Ts, Idarray, Darray);
455      END IF;
456   END Ct_Deccol;
457
458
459   PROCEDURE Ct_Pcoleccio
460     (A : IN Pnode;
461      Idtipus_Array : IN Id_Nom;
462      Idarray : OUT Id_Nom;
463      Ncomponents : OUT Despl) IS
464
465      Fesq : Pnode RENAMES A.Fe1;
466      Idrang : Id_Nom RENAMES A.Fd1.Id12;
467      E : Boolean;
468
469      Dtarray : Descriptipus;
470      Darray : Descrip;
471      Di : Descrip;
```

```
472
473    BEGIN
474       IF (A.Tipus = Pcoleccio) THEN
475          Ct_Pcoleccio(Fesq, Idtipus_Array, Idarray,
476                       Ncomponents);
477          Posa_Idx(Ts, Idarray, Idrang, E);
478
479          IF E THEN
480             error(posaIdxArray, A.Fd1.l1, A.Fd1.c1,
481                   cons_nom(tn, Idrang));
482          ELSE
483             Di := Cons(Ts, Idrang);
484             IF Di.td = Dtipus THEN
485                Ncomponents := Ncomponents +
486                  Despl(Di.Dt.Lsup - Di.Dt.Linf + 1);
487             ELSE
488                Error(Tipusidxerroniarray, A.Fd1.L1,
489                      A.Fd1.C1, Cons_Nom(Tn, Idrang));
490             END IF;
491          END IF;
492
493       ELSIF (A.Tipus = Pdimcoleccio) THEN
494          Dtarray := (Tsarr, 0, Idtipus_Array);
495          Darray := (Dtipus, Dtarray);
496          Idarray := Fesq.Id12;
497          Posa(Ts, Idarray, Darray, E);
498          IF E THEN
499             Error(tipusInexistent, Fesq.l1, Fesq.c1,
500                   cons_nom(tn, Idtipus_Array));
501             Ncomponents := 0;
502          END IF;
503
504          Di := cons(ts, Idrang);
505          IF NOT (Di.td = dtipus AND THEN
506                   Di.dt.tt <= tsent) THEN
507             Error(TipusIdxErroniArray, A.Fd1.l1, A.Fd1.c1,
508                   cons_nom(tn, Idrang));
509             Ncomponents := 0;
510          ELSE
511             Posa_Idx(Ts, Idarray, Idrang, E);
512             IF E THEN
513                Put_Line("ERROR CT-pdimcoleccio (DEBUG): "&
514                         "error al posa_idx, error "&
```

```
515                              "del compilador, array no creat,"&
516                              " idarr: "&Idarray'Img);
517              END IF;
518              Ncomponents := Despl(Di.Dt.Lsup
519                                   - Di.Dt.Linf + 1);
520           END IF;
521        END IF;
522
523     END Ct_Pcoleccio;
524
525
526     PROCEDURE Ct_Decregistre
527       (A : IN Pnode;
528        Idrecord : OUT Id_Nom;
529        Ocup: IN OUT despl) IS
530
531        Drecord : Descrip;
532        Dtrecord : Descriptipus;
533        E : Boolean;
534
535     BEGIN
536        IF (A.Tipus = Dregistre) THEN
537           Dtrecord := (Tsrec, 0);
538           Drecord := (Dtipus, Dtrecord);
539           Posa(Ts, A.Fe2.Id12, Drecord, E);
540           Idrecord := A.Fe2.Id12;
541           IF E THEN
542              error(id_existent, A.Fe2.l1, A.Fe2.c1,
543                    cons_nom(tn, Idrecord));
544              --Ver si hacer un error nuevo o no (no es
545              --una variable) es un tipo record
546           END IF;
547           Ct_Dregistre_Camp(A.Fe2.Id12, A.Fc2, A.Fd2,Ocup);
548
549        ELSIF (A.Tipus = Dencapregistre) THEN
550           Ct_Decregistre(A.Fe2, Idrecord, Ocup);
551           Ct_Dregistre_Camp(Idrecord, A.Fc2, A.Fd2,Ocup);
552
553        ELSIF (A.Tipus = Firecord) THEN
554           Ct_Decregistre(A.F6, Idrecord,Ocup);
555           Drecord := cons(ts,Idrecord);
556           Drecord.dt.ocup := ocup;
557           actualitza(ts, Idrecord,Drecord);
```

```
558        END IF;
559
560     END Ct_Decregistre;
561
562
563     PROCEDURE Ct_Dregistre_Camp
564       (Idrecord : IN Id_Nom;
565        Camp : IN Pnode;
566        Tcamp : IN Pnode;
567        Ocup: IN OUT Despl) IS
568
569        Idtcamp : Id_Nom RENAMES Tcamp.Id12;
570        Dtcamp : Descrip;
571        Idcamp : Id_Nom RENAMES Camp.Id12;
572        Desc_Camp : Descrip;
573        E : Boolean;
574
575     BEGIN
576        Dtcamp := Cons(Ts, Idtcamp);
577        IF (Dtcamp.Td /= Dtipus) THEN
578           error(tipusInexistent, Camp.l1, Camp.c1,
579                 cons_nom(tn, Idtcamp));
580        ELSE
581           Desc_Camp := (Dcamp, Idtcamp, Nul_Despl);
582           Posacamp(Ts, Idrecord, Idcamp, Desc_Camp, E);
583           Ocup := Ocup + Dtcamp.dt.ocup;
584           IF E THEN
585              error(idCampRecordExistent, Camp.l1,
586                    Camp.c1, cons_nom(tn, Idcamp));
587           END IF;
588        END IF;
589
590     END Ct_Dregistre_Camp;
591
592
593     PROCEDURE Ct_Decsubrang
594       (A : IN Pnode) IS
595
596        Idsubrang : Id_Nom RENAMES A.Fe5.Id12;
597        Idtsubrang : Id_Nom RENAMES A.Fc5.Id12;
598
599        Rang_Esq : Pnode RENAMES A.Fd5;
600        Rang_Dret : Pnode RENAMES A.Fid5;
```

```
601         Tsub : Tipussubjacent;
602
603         Tsesq : Tipussubjacent;
604         Tsdret : Tipussubjacent;
605         Idesq : Id_Nom;
606         Iddret : Id_Nom;
607         Valesq : Valor;
608         Valdret : Valor;
609
610         Tdecl : Descrip;
611         Tdescrip_decl : Descrip;
612         Tdescript_decl : Descriptipus;
613         L, C : Natural := 0;
614         E : Boolean;
615
616      BEGIN
617         Tdecl := Cons(Ts, Idtsubrang);
618         IF(Tdecl.Td /= Dtipus) THEN
619            error(tipusInexistent, A.Fc5.l1, A.Fc5.c1,
620                  cons_nom(tn, Idtsubrang));
621         ELSE
622            --Miram el fill esquerra
623            Ct_Constant(Rang_Esq, Tsesq, Idesq, L, C);
624            Valesq := Rang_Esq.val;
625
626            --Miram el fill dret
627            Ct_Constant(Rang_Dret, Tsdret, Iddret, L, C);
628            Valdret := Rang_Dret.val;
629
630            -- Comparam els tipus
631            IF (Tsesq /= Tsdret) THEN
632               Error(Tipussubdiferents, A.Fc5.L1, A.Fc5.C1,
633                     ""&Tsesq'Img&"/"&Tsdret'Img);
634            END IF;
635
636            Tsub := Tsesq;
637            IF (Tsub /= Tdecl.dt.tt) THEN
638               Error(Tipussubdiferents, A.Fc5.L1, A.Fc5.C1,
639                     ""&Tsub'Img&"/"&Tdecl.Dt.Tt'Img);
640            END IF;
641
642            IF (valesq > valdret) THEN
643               Error(ValEsqMajorDret, A.Fc5.L1, A.Fc5.C1,
```

```
644                  ""&Valesq'Img&" >"&Valdret'Img);
645          END IF;
646
647          IF (valesq < Tdecl.dt.Linf) THEN
648             Error(ValEsqMenor, A.Fc5.L1, A.Fc5.C1,
649                   Cons_Nom(Tn, Idtsubrang));
650          END IF;
651
652          IF (valdret > Tdecl.dt.Lsup) THEN
653             Error(ValDretMajor, A.Fc5.L1, A.Fc5.C1,
654                   Cons_Nom(Tn, Idtsubrang));
655          END IF;
656
657          CASE Tsub IS
658             WHEN tsent =>
659                Tdescript_decl := (tsent, 4, valesq,
660                                   valdret);
661             WHEN tscar =>
662                Tdescript_decl := (tscar, 4, valesq,
663                                   valdret);
664             WHEN OTHERS =>
665                Put_line("ERROR Ct_subrang: (Sub)Tipus no "&
666                         "valid per a un subrang");
667          END CASE;
668
669          Tdescrip_decl := (Dtipus, Tdescript_decl);
670          Posa(ts, Idsubrang, Tdescrip_decl, E);
671          IF E THEN
672             error(id_existent, A.Fe5.l1, A.Fe5.c1,
673                   cons_nom(tn, Idsubrang));
674          END IF;
675       END IF;
676
677    END Ct_Decsubrang;
678
679
680    PROCEDURE Ct_Expressio
681      (A : IN Pnode;
682       T : OUT Tipussubjacent;
683       Idtipus : OUT Id_Nom;
684       L, C : IN OUT Natural) IS
685
686       Tipus : Tipusnode RENAMES A.Tipus;
```

```
687       Tps : Tipussubjacent;
688       Id : Id_Nom;
689
690    BEGIN
691       Put_line("CT_EXP: "&Tipus'img );
692       CASE Tipus IS
693          WHEN Expressio =>
694             Ct_Expressioc(A, Tps, Id, L, C);
695          WHEN ExpressioUnaria =>
696             Ct_Expressiou(A, Tps, Id, L, C);
697          WHEN Identificador =>
698             Ct_Identificador(A, Tps, Id, L, C);
699          WHEN Const =>
700             Ct_Constant(A, Tps, Id, L, C);
701          WHEN Fireferencia | Referencia =>
702             Ct_Referencia_Var(A, Tps, Id); --falta L i C
703          WHEN OTHERS =>
704             Put_Line("ERROR CT-exp: tipus expressio no "&
705                      "trobat :S "&Tipus'Img);
706       END CASE;
707       T := Tps;
708       Idtipus := Id;
709
710       Put_Line("expressio: tsub: "&T'Img&" id: "&
711                Idtipus'Img);
712    END Ct_Expressio;
713
714
715    PROCEDURE Ct_Operand_Exp
716      (A : IN Pnode;
717       T : OUT Tipussubjacent;
718       Idtipus : OUT Id_Nom;
719       L, C : IN OUT Natural) IS
720
721       Tipus : Tipusnode RENAMES A.Tipus;
722
723    BEGIN
724       CASE Tipus IS
725          WHEN Expressio =>
726             Ct_Expressioc(A, T, Idtipus, L, C);
727          WHEN ExpressioUnaria =>
728             Ct_Expressiou(A, T, Idtipus, L, C);
729          WHEN Referencia | Fireferencia=>
```

```
730             --falta L i C
731             Ct_Referencia_var(A, T, IdTipus);
732             Put_Line("refe");
733          WHEN Const =>
734             Ct_Constant(A, T, Idtipus, L, C);
735             Put_line("CT_EXP_COMP const: "&Idtipus'img);
736          WHEN Identificador =>
737             Ct_Identificador(A, T, Idtipus, L, C);
738             Put_line("CT_EXP_COMP Id: "&Idtipus'img);
739          WHEN OTHERS =>
740             NULL;
741       END CASE;
742
743    END Ct_Operand_Exp;
744
745
746    PROCEDURE Ct_Expressioc
747      (A : IN Pnode;
748       T : OUT Tipussubjacent;
749       Idtipus : OUT Id_Nom;
750       L, C : IN OUT Natural) IS
751
752       Fesq : Pnode RENAMES A.Fe3;
753       Fdret : Pnode RENAMES A.Fd3;
754       Op : Operacio RENAMES A.Op3;
755
756       Tesq : Tipussubjacent;
757       Idesq : Id_Nom;
758       Tdret : Tipussubjacent;
759       Iddret : Id_Nom;
760
761    BEGIN
762       Put_line("CT_EXPRESSIOC");
763       --Analitzam l'operand esquerra
764       Ct_Operand_Exp(Fesq, Tesq, Idesq, L, C);
765       --Analitzam l'operand dret
766       Ct_Operand_Exp(Fdret, Tdret, Iddret, L, C);
767       -- Comparam els tipus
768       CASE Op IS
769          WHEN Unio | Interseccio =>
770             Ct_Exp_Logica(Tesq, Tdret, Idesq, Iddret, T,
771                           Idtipus, L, C);
772          WHEN Menor | Menorig | Major | Majorig
```

```
773             | Igual | Distint =>
774              Ct_Exp_Relacional(Tesq, Tdret, Idesq, Iddret,
775                                T, Idtipus, L, C);
776          WHEN Suma | Resta | Mult | Div | Modul =>
777             Ct_Exp_Aritmetica(Tesq, Tdret, Idesq, Iddret,
778                               T, Idtipus, L, C);
779          WHEN OTHERS =>
780             NULL;
781       END CASE;
782
783       Put_Line("ESQ: ts: "&Tesq'Img&" id: "&Idesq'Img);
784       Put_Line("DRT: ts: "&Tdret'Img&" id: "&Iddret'Img);
785
786    END Ct_Expressioc;
787
788
789    PROCEDURE Ct_Exp_Logica
790      (Tesq, Tdret : IN Tipussubjacent;
791       Idesq, Iddret : IN Id_Nom;
792       T : OUT Tipussubjacent;
793       Idtipus : OUT Id_Nom;
794       L, C : IN OUT Natural) IS
795
796    BEGIN
797       IF Tesq /= Tsbool THEN
798          Error(Tsub_No_Bool, L, C, "esquerra");
799       END IF;
800
801       IF Tdret /= Tsbool THEN
802          Error(Tsub_No_Bool, L, C, "dret");
803       END IF;
804
805       IF Idesq /= Id_Nul AND Iddret /= Id_Nul THEN
806          IF Idesq /= Iddret THEN
807             Error(Tops_Diferents, L, C, "");
808          END IF;
809       END IF;
810
811       IF Idesq = Id_Nul THEN
812          Idtipus := Iddret;
813       ELSE
814          Idtipus := Idesq;
815       END IF;
```

```
816
817       T := Tsbool;
818
819    END Ct_Exp_Logica;
820
821
822    PROCEDURE Ct_Exp_Relacional
823      (Tesq, Tdret : IN Tipussubjacent;
824       Idesq, Iddret : IN Id_Nom;
825       T : OUT Tipussubjacent;
826       Idtipus : OUT Id_Nom;
827       L, C : IN OUT Natural) IS
828
829    BEGIN
830       IF Tesq /= Tdret THEN
831          Error(Tsubs_Diferents, L, C, "");
832       END IF;
833
834       IF Tesq > Tsent THEN
835          Error(Tsub_No_Escalar, L, C, "esquerra");
836       END IF;
837
838       IF Tdret > Tsent THEN
839          Error(Tsub_No_Escalar, L, C, "dret");
840       END IF;
841
842       IF Idesq /= Id_Nul AND Iddret /= Id_Nul THEN
843          IF Idesq /= Iddret THEN
844             Error(Tops_Diferents, L, C, "");
845          END IF;
846       END IF;
847
848       T := Tsbool;
849       Idtipus := Id_Nul;
850
851    END Ct_Exp_Relacional;
852
853
854    PROCEDURE Ct_Exp_Aritmetica
855      (Tesq, Tdret : IN Tipussubjacent;
856       Idesq, Iddret : IN Id_Nom;
857       T : OUT Tipussubjacent;
858       Idtipus : OUT Id_Nom;
```

```
859         L, C : IN OUT Natural) IS
860
861      BEGIN
862         IF Tesq /= Tsent THEN
863            Error(Tsub_No_Sencer, L, C, "esquerra");
864         END IF;
865
866         IF Tdret /= Tsent THEN
867            Error(Tsub_No_Sencer, L, C, "dret");
868         END IF;
869
870         IF Idesq /= Id_Nul AND Iddret /= Id_Nul THEN
871            IF Idesq /= Iddret THEN
872               Error(Tops_Diferents, L, C, "");
873            END IF;
874         END IF;
875
876         T := Tsent;
877         IF Idesq = Id_Nul THEN
878            Idtipus := Iddret;
879         ELSE
880            Idtipus := Idesq;
881         END IF;
882
883      END Ct_Exp_Aritmetica;
884
885
886      PROCEDURE Ct_Expressiou
887        (A : IN Pnode;
888         T : OUT Tipussubjacent;
889         Idtipus : OUT Id_Nom;
890         L, C : IN OUT Natural) IS
891
892         Fdret : Pnode RENAMES A.F4;
893         Op : Operacio RENAMES A.Op4;
894         Tdret : Tipussubjacent;
895         Iddret : Id_Nom;
896
897      BEGIN
898         Put_line("CT_EXPRESSIOU");
899         Ct_Operand_Exp(Fdret, Tdret, Iddret, L, C);
900         CASE Op IS
901            WHEN Resta =>
```

```
902             Ct_Exp_Negacio(Tdret, Iddret, T, Idtipus,
903                            L, C);
904          WHEN Negacio =>
905             Ct_Exp_Neglogica(Tdret, Iddret, T, Idtipus,
906                              L, C);
907          WHEN OTHERS =>
908             NULL;
909       END CASE;
910
911       Put_Line("DRT: ts: "&T'Img&" id: "&Idtipus'Img);
912
913    END Ct_Expressiou;
914
915
916    PROCEDURE Ct_Exp_Negacio
917      (Ts : IN Tipussubjacent;
918       Id : IN Id_Nom;
919       T : OUT Tipussubjacent;
920       Idtipus : OUT Id_Nom;
921       L, C : IN OUT Natural) IS
922    BEGIN
923       IF Ts /= Tsent THEN
924          Error(Tsub_No_Sencer, L, C, "");
925       END IF;
926       Idtipus := Id;
927       T := Tsent;
928    END Ct_Exp_Negacio;
929
930
931    PROCEDURE Ct_Exp_Neglogica
932      (Ts : IN Tipussubjacent;
933       Id : IN Id_Nom;
934       T : OUT Tipussubjacent;
935       Idtipus : OUT Id_Nom;
936       L, C: IN OUT Natural) IS
937    BEGIN
938       IF Ts /= Tsbool THEN
939          Error(Tsub_No_Bool, L, C, "");
940       END IF;
941       Idtipus := Id;
942       T := Tsbool;
943    END Ct_Exp_Neglogica;
944
```

```
945
946   PROCEDURE Ct_Constant
947     (A : IN Pnode;
948      T : OUT Tipussubjacent;
949      Idtipus : OUT Id_Nom;
950      L, C : IN OUT Natural) IS
951
952      Tatr : Tipus_Atribut RENAMES A.Tconst;
953      Lin : Natural RENAMES A.L2;
954      Col : Natural RENAMES A.C2;
955      D : Descrip;
956
957   BEGIN
958      Put_line("CT_CONSTANT");
959      Idtipus := Id_Nul;
960      CASE (Tatr) IS
961         WHEN A_Lit_C =>
962            T := Tscar;
963         WHEN A_Lit_N =>
964            T := Tsent;
965         WHEN OTHERS =>
966            Put_Line("ERROR CT-constant: tipus constant "&
967                     "erroni");
968      END CASE;
969      L := Lin;
970      C := Col;
971
972   END Ct_Constant;
973
974
975   PROCEDURE Ct_Identificador
976     (A : IN Pnode;
977      T : OUT Tipussubjacent;
978      Idtipus : OUT Id_Nom;
979      L, C : IN OUT Natural) IS
980
981      Id : Id_Nom RENAMES A.Id12;
982      D : Descrip;
983      Desc : Tdescrip RENAMES D.Td;
984      Lin : Natural RENAMES A.L1;
985      Col : Natural RENAMES A.C1;
986
987      Carg : Cursor_Arg;
```

```
988
989     BEGIN
990        put_line(" CT_ID : "&Id'img);
991        D := Cons(Ts, Id);
992
993        CASE Desc IS
994           WHEN Dvar =>
995              Idtipus := D.Tr;
996              D := Cons(Ts, Idtipus);
997              IF (D.Td = Dtipus) THEN
998                 T := D.Dt.Tt;
999              ELSE
1000                Error(Tipus_No_Desc, L, C, D.Td'Img);
1001             END IF;
1002
1003          WHEN Dconst =>
1004             Idtipus := D.Tc;
1005             D := Cons(Ts, Idtipus);
1006             IF (D.Td = Dtipus) THEN
1007                T := D.Dt.Tt;
1008             ELSE
1009                Error(Tipus_No_Desc, L, C, D.Td'Img);
1010             END IF;
1011
1012          WHEN Dproc =>
1013             Carg := Primer_Arg(Ts, Id);
1014             IF Arg_Valid(Carg) THEN
1015                T := Tsarr;
1016             ELSE
1017                T := Tsnul;
1018             END IF;
1019             Idtipus := Id;
1020
1021          WHEN OTHERS =>
1022             Error(Id_No_Reconegut, L, C, Desc'Img);
1023             Idtipus := Id;
1024             T := tsnul;
1025
1026       END CASE;
1027       L := Lin;
1028       C := Col;
1029
1030       Put_line("ct_id: Tipus: "&Idtipus'img);
```

```
1031
1032    END Ct_Identificador;
1033
1034
1035    PROCEDURE Ct_Bloc
1036      (A : IN Pnode) IS
1037
1038       D : Descrip;
1039       T : Tipussubjacent;
1040       Idbase : Id_Nom;
1041       Idtipus : Id_Nom;
1042
1043       Tsexp : Tipussubjacent;
1044       Idexp : Id_Nom;
1045       Tsvar : Tipussubjacent;
1046       Idvar : Id_Nom;
1047       L, C : Natural := 0;
1048
1049    BEGIN
1050       CASE (A.Tipus) IS
1051          WHEN Bloc =>
1052             Ct_Bloc(A.Fe1);
1053             Ct_Bloc(A.Fd1);
1054          WHEN Repeticio =>
1055             Ct_Srep(A);
1056          WHEN Identificador =>
1057             Put_Line("CT_Bloc : IDENTIFICADOR");
1058             Ct_Identificador(A, T, Idtipus, L, C);
1059             IF T /= Tsnul THEN
1060                Error(Id_No_Cridaproc, L, C,
1061                      Cons_Nom(Tn, A.Id12));
1062             END IF;
1063
1064          WHEN Fireferencia =>
1065             Ct_Referencia_Proc(A, T, Idbase);
1066          WHEN condicionalS =>
1067             Ct_Sconds(A);
1068          WHEN condicionalC =>
1069             Ct_Scondc(A);
1070          WHEN Assignacio =>
1071             Ct_Referencia_Var(A.Fe1, Tsvar, Idvar);
1072             Ct_Expressio(A.Fd1, Tsexp, Idexp, L, C);
1073             IF Tsvar /= Tsexp THEN
```

```
1074                Error(Assig_Tipus_Diferents, L, C, "");
1075             END IF;
1076             IF Idexp /= Id_Nul AND Idexp /= Idvar THEN
1077                Error(Assig_Tipus_Diferents, L, C, "");
1078             END IF;
1079          WHEN OTHERS =>
1080             Put_Line("blocothers"&A.Tipus'img);
1081       END CASE;
1082    END Ct_Bloc;
1083
1084
1085    PROCEDURE Ct_Srep
1086      (A : IN Pnode) IS
1087
1088       Tsexp : Tipussubjacent;
1089       Idtipus_exp : Id_Nom;
1090       Exp : Pnode RENAMES A.Fe1;
1091       Bloc : Pnode RENAMES A.fd1;
1092       L, C : Natural := 0;
1093
1094    BEGIN
1095       Ct_Expressio(Exp, Tsexp, Idtipus_Exp, L, C);
1096       IF tsexp /= tsbool THEN
1097          Error(Exp_No_Bool, L, C, "bucle");
1098       END IF;
1099       Ct_Bloc(Bloc);
1100    END Ct_Srep;
1101
1102
1103    PROCEDURE Ct_Sconds
1104      (A : IN Pnode) IS
1105
1106       Tsexp : Tipussubjacent;
1107       Idtipus_exp : Id_Nom;
1108       Cond : Pnode RENAMES A.Fe1;
1109       Bloc : Pnode RENAMES A.fd1;
1110       L, C : Natural := 0;
1111
1112    BEGIN
1113       Ct_Expressio(Cond, Tsexp, Idtipus_Exp, L, C);
1114       IF tsexp /= tsbool THEN
1115          Error(Exp_No_Bool, L, C, "condicional");
1116       END IF;
```

```
1117        Ct_Bloc(Bloc);
1118     END Ct_Sconds;
1119
1120
1121     PROCEDURE Ct_Scondc
1122       (A : IN Pnode) IS
1123
1124        Tsexp : Tipussubjacent;
1125        Idtipus_exp : Id_Nom;
1126        Cond : Pnode RENAMES A.Fe2;
1127        Bloc : Pnode RENAMES A.fc2;
1128        Blocelse : Pnode RENAMES A.fd2;
1129        L, C : Natural := 0;
1130
1131     BEGIN
1132        Ct_Expressio(Cond, Tsexp, Idtipus_Exp, L, C);
1133        IF tsexp /= tsbool THEN
1134           Error(Exp_No_Bool, L, C, "condicional compost");
1135        END IF;
1136        Ct_Bloc(Bloc);
1137        Ct_Bloc(Blocelse);
1138     END Ct_Scondc;
1139
1140
1141     PROCEDURE Ct_Referencia_Proc
1142       (A : IN Pnode;
1143        T : OUT Tipussubjacent;
1144        Id : OUT Id_Nom) IS
1145
1146        Tipus : Tipusnode RENAMES A.Tipus;
1147        It_Arg : Cursor_Arg;
1148        L, C : Natural := 0;
1149
1150     BEGIN
1151        CASE Tipus IS
1152           WHEN Identificador =>
1153              Ct_Identificador(A, T, Id, L, C);
1154           WHEN Referencia =>
1155              Error(Rec_No_Cridaproc, L, C, "");
1156           WHEN Fireferencia =>
1157              Ct_Ref_Pri(A.F6, T, It_Arg);
1158              IF Arg_Valid(It_Arg) THEN
1159                 Error(Falta_Param_Proc, L, C, "");
```

```
1160            END IF;
1161         WHEN OTHERS =>
1162            Put_Line("ERROR CT-referencia: node "&
1163                     "no reconegut");
1164      END CASE;
1165
1166   END Ct_Referencia_Proc;
1167
1168
1169
1170   PROCEDURE Ct_Referencia_Var
1171     (A : IN Pnode;
1172      T : OUT Tipussubjacent;
1173      Id : OUT Id_Nom) IS
1174
1175      Tipus : Tipusnode RENAMES A.Tipus;
1176      Idtipus : Id_Nom;
1177      It_Idx : Cursor_Idx;
1178      D : Descrip;
1179      L, C : Natural := 0;
1180
1181   BEGIN
1182      CASE Tipus IS
1183         WHEN Identificador =>
1184            Ct_Identificador(A, T, Id, L, C);
1185            D := Cons(Ts, Id);
1186            IF D.Td = Dproc THEN
1187               Error(Refvar_No_Proc, L, C, "");
1188            END IF;
1189         WHEN Referencia =>
1190            Ct_Ref_Rec(A, T, Id, Idtipus);
1191         WHEN Fireferencia =>
1192            Ct_Ref_Pri(A.F6, T, Id, It_Idx);
1193            IF Idx_Valid(It_Idx) THEN
1194               --falta L i C
1195               Error(Falta_Param_Array, L, C, "");
1196            END IF;
1197            IF T = Tsarr THEN
1198               D := Cons(Ts, Id);
1199               Id := D.Dt.Tcamp;
1200               D := Cons(Ts, Id);
1201               T := D.Dt.Tt;
1202            END IF;
```

```
1203            WHEN OTHERS =>
1204               Put_Line("ERROR CT-referencia: node no "&
1205                        "reconegut");
1206         END CASE;
1207
1208      END Ct_Referencia_Var;
1209
1210
1211      PROCEDURE Ct_Ref_Rec
1212        (A : IN Pnode;
1213         T : OUT Tipussubjacent;
1214         Idtipus : OUT Id_Nom;
1215         Idbase : OUT Id_Nom) IS
1216
1217         Fesq : Pnode RENAMES A.Fe1;
1218         Tesq : Tipussubjacent;
1219         Idbase_Esq : Id_Nom;
1220         Dcamp : Descrip;
1221         Dtcamp : Descrip;
1222         Idcamp : Id_Nom RENAMES A.Fd1.Id12;
1223         L, C : Natural := 0;
1224
1225      BEGIN
1226         Ct_Referencia_Var(Fesq, Tesq, Idbase_Esq);
1227         IF Tesq /= Tsrec THEN
1228            Error(Reccamp_No_Valid, L, C, ""); --falta L i C
1229         END IF;
1230
1231         Dcamp := Conscamp(Ts, Idbase_Esq, Idcamp);
1232         IF Dcamp.Td = Dnula THEN
1233            --comprovar nom
1234            Error(Idrec_No_Valid, L, C, Cons_Nom(Tn, Idcamp));
1235         END IF;
1236
1237         Idtipus := Dcamp.Tcamp;
1238         Dtcamp := Cons(Ts, Dcamp.Tcamp);
1239         T := Dtcamp.Dt.Tt;
1240         Idbase := Idbase_Esq;
1241
1242      END Ct_Ref_Rec;
1243
1244
1245      PROCEDURE Ct_Ref_Pri
```

```
1246      (A : IN Pnode;
1247       T : OUT Tipussubjacent;
1248       Id : OUT Id_Nom;
1249       It_Idx : OUT Cursor_Idx) IS
1250
1251       Tipus : Tipusnode RENAMES A.Tipus;
1252       Fesq : Pnode RENAMES A.Fe1;
1253       Fdret : Pnode RENAMES A.Fd1;
1254       Tsub : Tipussubjacent;
1255       Idvar : Id_Nom;
1256
1257       Tsref : Tipussubjacent;
1258       Idref : Id_Nom;
1259
1260       Id_Cursor : Id_Nom;
1261       Dtipoarg : Descrip;
1262       Dbase : Descrip;
1263       L, C : Natural := 0;
1264
1265    BEGIN
1266       CASE Tipus IS
1267          WHEN Pri =>
1268             Put_Line("CT-ref_pri: pri");
1269             Ct_Ref_Pri(Fesq, T, Id, It_Idx);
1270             Ct_Expressio(Fdret, Tsref, Idref, L, C);
1271             IF NOT Idx_Valid(It_Idx) THEN
1272                Error(Sobren_Parametres, L, C, "");
1273             ELSE
1274                Id_Cursor := Cons_Idx(Ts, It_Idx);
1275                Dtipoarg := Cons(ts, Id_Cursor);
1276                IF Idref = Id_Nul THEN
1277                   IF Dtipoarg.dt.tt /= Tsref THEN
1278                      Error(Tparam_No_Coincident,
1279                            L, C, "");
1280                   END IF;
1281                ELSIF Idref /= Id_cursor THEN
1282                   Error(Tparam_No_Coincident, L, C,
1283                         Cons_Nom(Tn, Idref)&"/"&
1284                           Cons_Nom(Tn, Id_Cursor));
1285                END IF;
1286                It_Idx := Succ_Idx(Ts, It_Idx);
1287             END IF;
1288
```

```
1289          WHEN Encappri =>
1290             Put_Line("CT-ref_pri: encappri");
1291             Ct_Referencia_Var(Fesq, Tsub, Idvar);
1292             Ct_Expressio(Fdret, Tsref, Idref, L, C);
1293             Dbase := Cons (ts, Idvar);
1294             IF Tsub = Tsarr THEN
1295                It_Idx := Primer_Idx(Ts, Idvar);
1296                IF Idx_Valid(It_Idx) THEN
1297                   Id_Cursor := Cons_Idx(Ts, It_Idx);
1298                   Dtipoarg := Cons(Ts, Id_Cursor);
1299                   IF Idref = Id_Nul THEN
1300                      IF Dtipoarg.dt.tt /= Tsref THEN
1301                         Error(Tparam_No_Coincident, L, C, "");
1302                      END IF;
1303                   ELSIF Idref /= Id_Cursor THEN
1304                      Error(Tparam_No_Coincident, L, C,
1305                            Cons_Nom(Tn, Idref)&"/"&
1306                              Cons_Nom(Tn, Id_Cursor));
1307                   END IF;
1308                END IF;
1309             ELSE
1310                Error(Tipus_No_Array, L, C, Tsub'Img);
1311             END IF;
1312             It_Idx := Succ_Idx(Ts, It_Idx);
1313             T := Tsub;
1314             Id := Idvar;
1315
1316          WHEN OTHERS =>
1317             Put_Line("ERROR CT-ref_pri: tipus no "&
1318                      "reconegut");
1319       END CASE;
1320
1321    END Ct_Ref_Pri;
1322
1323
1324    PROCEDURE Ct_Ref_Pri
1325      (A : IN Pnode;
1326       T : OUT Tipussubjacent;
1327       It_Arg : OUT Cursor_Arg) IS
1328
1329       Tipus : Tipusnode RENAMES A.Tipus;
1330       Fesq : Pnode RENAMES A.Fe1;
1331       Fdret : Pnode RENAMES A.Fd1;
```

```
1332        Tsub : Tipussubjacent;
1333        Id : Id_Nom;
1334
1335        Tsref : Tipussubjacent;
1336        Idref : Id_Nom;
1337
1338        Id_Cursor : Id_Nom;
1339        Dparam : Descrip;
1340        Dtipoarg : Descrip;
1341        Dbase : Descrip;
1342        L, C : Natural := 0;
1343
1344     BEGIN
1345        CASE Tipus IS
1346           WHEN Pri =>
1347              Put_Line("CT-ref_pri: pri");
1348              Ct_Ref_Pri(Fesq, T, It_Arg);
1349              Ct_Expressio(Fdret, Tsref, Idref, L, C);
1350
1351              IF NOT Arg_Valid(It_Arg) THEN
1352                 Error(Sobren_Parametres, L, C, "");
1353              ELSE
1354                 Cons_Arg(Ts, It_Arg, Id_Cursor, Dparam);
1355                 IF Idref = Id_Nul THEN
1356                    Dtipoarg := Cons(ts, Dparam.targ);
1357                    IF Dtipoarg.dt.tt /= Tsref THEN
1358                       Error(Tparam_No_Coincident, L,
1359                             C, Dtipoarg.Dt.Tt'Img);
1360                    END IF;
1361                 ELSIF Dparam.td = Dargc THEN
1362                    IF Idref /= Dparam.targ THEN
1363                       Error(Tparam_No_Coincident, L, C,
1364                                Cons_Nom(Tn, Idref)&"/"&
1365                                  Cons_Nom(Tn, Id_Cursor));
1366                    END IF;
1367                 ELSIF Dparam.td = Dvar THEN
1368                    IF Idref /= Dparam.tr THEN
1369                       Error(Tparam_No_Coincident, L, C,
1370                                Cons_Nom(Tn, Idref)&"/"&
1371                                  Cons_Nom(Tn, Id_Cursor));
1372                    END IF;
1373                 END IF;
1374                 It_Arg := succ_arg(ts, It_Arg);
```

```
1375              END IF;
1376
1377           WHEN Encappri =>
1378              Put_Line("CT-ref_pri: encappri");
1379              Ct_Referencia_Proc(Fesq, Tsub, Id);
1380              Ct_Expressio(Fdret, Tsref, Idref, L, C);
1381              Dbase := Cons (ts, id);
1382              IF Tsub = Tsarr AND Dbase.td = Dproc THEN
1383                 It_Arg := Primer_Arg(Ts, Id);
1384                 IF Arg_Valid(It_Arg) THEN
1385                    Cons_Arg(Ts, It_Arg, Id_Cursor, Dparam);
1386                    IF Idref = Id_Nul THEN
1387                       Dtipoarg := Cons(ts, Dparam.targ);
1388                       IF Dtipoarg.dt.tt /= Tsref THEN
1389                          Error(Tparam_No_Coincident,
1390                                L, C, "");
1391                       END IF;
1392                    ELSIF Dparam.td = Dargc THEN
1393                       IF Idref /= Dparam.targ THEN
1394                          Error(Tparam_No_Coincident, L, C,
1395                                Cons_Nom(Tn, Idref)&"/"&
1396                                  Cons_Nom(Tn, Id_Cursor));
1397                       END IF;
1398                    ELSIF Dparam.td = Dvar THEN
1399                       IF Idref /= Dparam.tr THEN
1400                          Error(Tparam_No_Coincident, L, C,
1401                                Cons_Nom(Tn, Idref)&"/"&
1402                                  Cons_Nom(Tn, Id_Cursor));
1403                       END IF;
1404                    END IF;
1405                 END IF;
1406              ELSE
1407                 Error(Tproc_No_Param, L, C, Tsub'Img);
1408              END IF;
1409              It_Arg := succ_arg(ts, It_Arg);
1410              T := Tsub;
1411           WHEN OTHERS =>
1412              Put_Line("ERROR CT-ref_pri: tipus no "&
1413                       "reconegut");
1414        END CASE;
1415     END Ct_Ref_Pri;
1416
1417  END Decls.Ctipus;
```

3.4 Missatges d'error

3.4.1 Fitxer *decls-missatges.ads*

```
1  WITH     decls.dgenerals,
2           Ada.Text_IO;
3
4  USE      decls.dgenerals,
5           Ada.Text_IO;
6
7  PACKAGE Decls.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
53    PROCEDURE Tanca_Fitxer;
54
55    PROCEDURE Error
56      (Te : IN Terror;
57       L, C : IN Natural;
58       Id : String);
59
60    PROCEDURE Error
61      (Te : IN Terror;
62       Id : String);
63
64    PROCEDURE Impressio
65      (Msj : IN String);
66
67 PRIVATE
68
69    Log_File : File_Type;
70
71 END Decls.Missatges;
```

3.4.2 Fitxer *decls-missatges.adb*

```
PACKAGE BODY decls.missatges IS

   PROCEDURE Obre_Fitxer IS
   BEGIN
      Create(log_file, out_file, "Fitxer_errors.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, msj);
      put_line(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"&
                      " l'argument");
```

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

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

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

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

4 Declaracions i altres paquets

4.1 Fitxer *decls.ads*

```
1  -- ------------------------------------------------
2  --  Paquet de Declaracions
3  -- ------------------------------------------------
4  --  Versio  :   0.1
5  --  Autors  :   Jose Ruiz Bravo
6  --              Biel Moya Alcover
7  --              Alvaro Medina Ballester
8  -- ------------------------------------------------
9  --  Paquet de declaracions pare.
10 --
11 -- ------------------------------------------------
12
13 PACKAGE decls IS
14
15    --pragma pure;
16
17
18 END decls;
```

4.2 Fitxer *decls-dgenerals.ads*

```
1  -- ------------------------------------------------
2  --  Paquet de declaracions generals
3  -- ------------------------------------------------
4  --  Versio  :   0.3
5  --  Autors  :   Jose Ruiz Bravo
6  --              Biel Moya Alcover
7  --              Alvaro Medina Ballester
8  -- ------------------------------------------------
9  --  Declaracions generals.
10 --
11 -- ------------------------------------------------
12
13 PACKAGE decls.dgenerals IS
14
15     --pragma pure;
16
17     max_id : CONSTANT integer := 1000;
18     long_num_ident : CONSTANT integer := 40;
19
20     max_var : CONSTANT integer := 1000;
21     TYPE num_var IS NEW natural
22       RANGE 0 .. max_var;
23
24     max_proc : CONSTANT integer := 100;
25     TYPE num_proc IS NEW natural
26       RANGE 0 .. max_proc;
27
28     TYPE valor IS NEW Integer
29       RANGE integer'First.. integer'Last;
30
31     TYPE tipus_atribut IS
32       (atom,
33        a_ident,
34        A_Lit_C,
35        A_Lit_N,
36        A_Lit_S,
37        nodeArbre);
38
39
40 END decls.dgenerals;
```

5 Proves i programa principal

5.1 Fitxer *compilemon.adb*, programa principal

```
1  -- ------------------------------------------------
2  --  Programa de prova
3  -- ------------------------------------------------
4  --  Versio  :   0.2
5  --  Autors  :   Jose Ruiz Bravo
6  --              Biel Moya Alcover
7  --              Alvaro Medina Ballester
8  -- ------------------------------------------------
9  --      Programa per comprovar les funcionalitats
10 --  del lexic i la taula de noms.
11 --
12 -- ------------------------------------------------
13
14 WITH     Ada.Text_IO,
15          Ada.Command_Line,
16          decls.d_taula_de_noms,
17          decls.dgenerals,
18          decls.dtsimbols,
19          decls.dtdesc,
20          pk_usintactica_tokens,
21          pk_ulexica_io,
22          u_lexica,
23          Pk_Usintactica,
24          Decls.D_atribut,
25          decls.d_arbre,
26          decls.Dtnode,
27          Decls.Ctipus;
28
29 USE      Ada.Text_IO,
30          Ada.Command_Line,
31          decls.d_taula_de_noms,
32          decls.dgenerals,
33          decls.dtsimbols,
34          decls.dtdesc,
35          pk_usintactica_tokens,
36          pk_ulexica_io,
37          u_lexica,
38          Pk_Usintactica,
39          decls.d_arbre,
```

```
40          decls.Dtnode,
41          Decls.Ctipus;
42
43 PROCEDURE compilemon IS
44 BEGIN
45    Open_Input(Argument(1));
46
47    --PROVISIONAL
48    Inicia_analisi;
49
50    yyparse;
51    Ct_Programa(Arbre);
52    Close_Input;
53
54    EXCEPTION
55       WHEN Syntax_Error =>
56          Put_Line("ERROR: Error a la linea "
57                   &yy_line_number'img&
58                   " i columna "&yy_begin_column'img);
59
60 END compilemon;
```

5.1.1 Prova 3: fitxer *prova3.lem*

```
1  PROCEDURE provaid IS
2
3          --h:integer;
4          --type num is new integer range 0..2;
5          --type bili is array (h, num) of integer;
6
7          --procedure provaid(pepe : in integer; bili : in out integer) is
8          --begin
9          --      h := 1;
10         --end provaid;
11
12         --procedure proaid is
13         --begin
14         --      h := 1;
15         --end proaid;
16
17         --proaid : integer;
18
19         --type p is record
20         --      m : boolean;
21         --      l : integer;
22         --end record;
23
24         e : boolean;
25
26
27 BEGIN
28
29         --h:=0;
30         e := false;
31         IF (e - e) THEN
32         e := true;
33         END IF;
34
35 END provaid;
```

Índex