7. EXPRESSIONS

7.1 Procedure call

When a procedure is called, we must distinguish between four kinds of procedures: non-formal-non-virtual, virtual, formal and external. Their characteristics are as follows:

For a non-formal-non-virtual procedure, the compiler may check that the number of actual parameters is correct and that the actual parameters are compatible with the formal parameters. Value parameters may be transmitted by compiler generated (in-line) coding.

For a virtual procedure, these checks must be made at runtime. The compiler does not know which parameters are called by value. In fact, the compiler does not even know that a matching virtual procedure exists. The access to the procedure must be through the prototype of the object where the procedure is local.

For a formal procedure, much of the same checking must be done as for virtual procedures.

For an external procedure, the checking of ref parameters may be simplified as compared to formal procedures, since ref parameters to an external procedure may only be qualified by a system class.*)

In the formal description, a ref (driver) parameter has been used instead of a display index because it is easier to understand. The use of a display index would be simpler in an implementation.

We have, when calling a non-formal (non-virtual or virtual) procedure, P, three different cases:

*) This point is currently being clarified by the SIMULA Standards Group.

1. P is normal, i.e. its name is statically visible without connection. In this case, only one driver is required. The static link (drp) of this driver is found in DDISPLAY(bl) at the level of the block where P is declared.

2. P is connected, i.e. its name has become visible through a connection. Two drivers are required. The first driver is an ordinary procedure driver with a static link to the other extra driver. For the second driver con is true, pex and drex are none, obj is found in DISPLAY (blc) where blc is the level of the connection block where the object in which P is declared is connected, and drp is found in DDISPLAY(blc).cdrp where blc is defined as above, or alternatively as DISPLAY[bld-1].MDP where bld is block level of class declaration.

3. P is remote, i.e. accessed by remote referencing, and the class of the element expression, **X**, preceding the dot preceding P is C (P is thus local to C). **X** must be supplied by the compiler as a parameter to the runtime system. Two drivers are required: an ordinary procedure driver with a static link (drp) to the extra driver. The extra driver has a static link (drp) found in ddisplay (blc) where blc is the apparent block level of the class C, con is true, pex and drex are none and obj is **X**.

7.1.1 Non-formal non-virtual procedure

A procedure call P(A1,.....,An) is assumed to create the following in-line coding in the compiled program:

1. A call on a call procedure runtime routine to generate the date storage for the procedure, insert PP, create necessary drivers and prepare to accept value parameters and name parameter descriptors through in-line coding.

2. In-line coding to compute and store dynamic parameter descriptors and value parameters.

During the value parameter evaluation, the procedure itself has two drivers.

The first is an ordinary procedure driver, the second is a thunk driver which is deleted when all value parameters have been computed and stored.

If the number of value parameters is zero, the second driver is not created.

The contents of the thunk driver are as follows:

obj	pointer to B.I. B where the procedure is called
drp	equal to drp of B
pex	none
drex	pointer to the procedure driver
acs	none
md	false

con and cdrp are inherited from the driver of B

When all parameters and descriptors have been evaluated, the second driver is discarded and display is updated to point to the static environment of the procedure.

The compiler generated coding for the procedure is entered. In this coding, the local declarations are performed and the procedure body entered.

The parameter "acs" is used to indicate that an accumulator stack may be present. The corresponding actual parameter will be either none or a generating expression. This does not imply that the accumulator stack need be saved with in-line coding. In an implementation, acs will probably be a description of the accumulator stack, and the save function is performed by the runtime system.

```
procedure ENTER;
     begin ref (driver) y;
        y :- CD;
        CD :- CD.drex;
        CD.pex :- exit;
        deletenotice (y);
        update display;
        go to CD.obj.PP.prefix[0]. declare;
end ENTER;
```

Note: procedure prototype have prefix [0] == prototype of procedure.

A call on the subroutine "ENTER" is compiled at the end of the in-line coding for parameter transmission to procedures and classes.

```
procedure ENTPROC;
     begin ref (driver) y;
     CD.obj.MDP :- CD;
     if CD.obj.PP.nrp = 0 then begin update display;
                                  go to CD.obj.PP.declare
                                end;
     y:- CD.drex;
     CD :- new driver (y.obj,y.drp,none,CD,none,false,
                                                    y.level);
     CD.con := y.con; CD.cdrp :- y.cdrp;
     DDISPLAY[y.level] :- CD;
     go to CD.drex.pex;
end ENTPROC;
```

If the procedure has no parameters, "ENTPROC" will cause the in-line coding for declarations local to the procedure to be entered. This in-line coding continues with the first statement of the body.

```
procedure CPR (p,acs);
     ref (prototype) p; ref (object) acs;
     begin
     comment call procedure;
        CD :- new driver (new object(p),DDISPLAY[p.level-1],
                                        exit,CD,acs,true,p.level);
        ENTPROC;
     end CPR;
```

A driver for the procedure is created and the procedure is entered using "entproc" (described above).

```
procedure CCP (p,c,acs);
     ref (prototype) p; ref (object) acs;
     ref (driver) c;
     begin ref (driver) x;
     comment call connected procedure;
         x :- new driver(c.obj,c.cdrp,none,none,none,false,
              p.level-1);
         x.con := true;
         CD :- new driver(new object(p),x,exit,CD,acs,true,
                                                 p.level);
         CD.dot := true; ENTPROC;
     end CCP;
```

CCP is used when a procedure belonging to a connected object is called. It creates a substitute driver for the connected object, the driver for the procedure and the procedure universe, and enters the procedure using "ENTPROC" (described above).

```
procedure CDP (p,c,slc,acs);
     ref (prototype) p; ref (object) c,acs;
     ref (driver)slc;
     begin ref (driver) x;
     comment call dot procedure;
```

```
        x :- new driver (c,slc,none,none,none,false,p.level-1);
        x.con := true;
        CD :- new driver (new object(p),x,exit,CD,acs,true,
                                                    p.level);
        CD.dot := true;
        ENTPROC;
    end CDP;
```

A substitute driver is created for the object where the procedure is declared.

A driver for the procedure is created with a static link to this substitute driver. Dot of the procedure driver is true to indicate that both dynamic and static links must be followed by the store collapse, and that the substitute driver should be deleted at procedure exit.

The procedure is entered through ENTPROC.

7.1.2 Virtual procedure

The number of parameters and their types are not checked by the compiler for a call on a virtual procedure. The checking is left to the runtime system subroutines.

A virtual procedure, V, when called, may be of either of the cases mentioned for non-formal - non-virtual procedures.

A call on a virtual procedure V(A1,.....,An), is assumed to create the following in-line coding in the compiled program:

1. A call on a "call virtual procedure" runtime routine to:

    1. Generate the data storage for the procedure.
    2. Insert PP.
    3. Create necessary drivers.
    4. Check parameter numbers and types.
    5. Compute value parameters.
    6. Insert name parameter descriptors.
    7. Enter the procedure (starting with local declarations).

2. The number of parameters

3. Static parameter descriptor (spd) for each parameter.

```
procedure entvirt (p,dr,dot,acs);
   ref (prototype) p; ref (driver) dr;
   ref (object) acs; Boolean dot;
   begin check number of parameters etc;
      CD :- new driver (new object(p),dr,exit,CD,acs,true,
                                                  p.level);
      CD.dot := dot;
      CD.obj.MDP :- CD;
      store descriptors for name parameters;
      store value parameters;
      update display;
      go to p.declare
   end entvirt;

procedure CVP (cl,index,acs);
   integer index;
   ref (object) acs; ref (driver) cl;
   begin ref (prototype) p,q;
      p :- cl.obj.PP;
      q :- p.progaddr (index) qua prototype;
      if q == none then error ("cvp",1);
      entvirt (q,cl,false,acs)
   end CVP;

procedure CCVP(c,index,acs);
   integer index;
   ref (object) acs; ref (driver) c;
   begin ref (prototype) p,q;
       p :- c.obj.PP;
       q :- p.progaddr (index) qua prototype;
       if q == none then error ("ccvp",1);
       c :- new driver (c.obj,c.cdrp,none,none,none,
            false, p.level);
       entvirt (q,c,true,acs)
   end CCVP;
```

```
procedure CDVP(c,index,slc,acs);
   integer index;
   ref (object) c,acs; ref (driver) slc;
   begin ref (prototype) p;
      p :-  c.PP.progaddr(index) qua prototype;
      if p == none then error ("CDVP",1);
      slc :- new driver (c,slc,none,none,none,
                         false,p.level-1);
      slc.con := true;
      entvirt (p,slc,true,acs);
   end CDVP;
```

7.1.3 Formal procedures

A formal procedure may only be normal, not remote or connected.

The dynamic parameter descriptor for an actual procedure or <type> procedure corresponding to a formal parameter specified as procedure or <type> procedure must contain the following information:

1. Pointer to the prototype for the procedure.
2. Pointer to driver of object where procedure is declared.

The number of parameters and their type given in the call will be matched against the parameter requirements of the procedure parameters.

Type differences will be tolerated only for value parameters with type compatible with the declared type.

7.2 Arithmetic and Boolean expressions

These follow the rules as defined in ALGOL 60 with the exception of <factor>↑<term> which is of type real.

7.3 Ref_expressions

7.3.1 Validity

The qualification of a reference may be divided into two parts: a static qualification (the class) and a dynamic qualification (the object instance where the class is local).

The dynamic qualification is not in general known at compile time. The need to check the dynamic qualification would be a heavy burden on the runtime system in the general case.

The implementation is greatly simplified by the following restrictions in the Common Base:

1. Quantities declared within a class containing local class declarations may not be accessed by dot-notation.

2. Synonomous classes whose apparent block levels are different, are assumed by the compiler to have different dynamic qualifications.

These two restrictions imply that the dynamic qualification may be completely checked at compile time, except in the case of ref parameters to a formal or virtual procedure. In these exceptional cases, the dynamic qualification must be checked at procedure entry time.

The check is performed by comparing BL prior to entry of the procedure with BLF, where BL is the apparent block level of the qualification of the actual parameter and BLF the one for the formal parameter.

In addition, the qualification compatibility check normally performed at compile time must be performed at runtime when a formal or virtual procedure is called. The check ensures that one of the qualifying classes includes the other.

It follows that the static qualification as well as the associated apparent block level for each actual and formal parameter must be available to the runtime system at the time of entering a formal or virtual procedure.

The preceding paragraphs also apply for ref array parameters.

In the following four cases, class membership of a reference value must be compared and checked against the static qualification of a variable:

1. Explicit reference assignment, case 2.

2. Reference parameter transmission to non-formal, non-virtual procedures in default mode, case 2.

3. Name parameter to the left of denotes. This check is required even in case 1, since the qualification of the actual parameter may be a subclass of the qualification of the formal one. Since the qualification of the actual parameter is not known at compile time, this check belongs in a system soubroutine which interprets the store operation (SFP).

4. Name parameter in reference expression. A check similar to that of 3 above must be performed in the subroutine which interprets the load operation (LFP) because the formal qualification may be a subclass of the actual one.

7.3.2 Generating reference

The generating reference new C(A1,....,An) is assumed to create the following in-line coding in the compiled program:

1. A call on a "begin class" procedure to generate the C object, create the necessary drivers and prepare to accept parameters as in-line coding.

2. In-line coding to compute parameters and store the values in the C object.

3. A call on the subroutine ENTER which will enter the coding for declarations in the outermost prefix.

During the parameter evaluation, the object has two drivers:

The first one is the ordinary driver for an attached object, the second is a thunk driver which is deleted when all parameters have been computed and stored.

If the number of parameters is zero, the second driver need not be created.

The contents of the thunk driver are as follows:

obj	pointer to block B where generating expression is found.
drp	equal to static link (drp) of block b.
pex	none.
drex	pointer to object driver.
acs	none.
md	false

When all parameters have been evaluated, the second driver is discarded and display is updated. The declarations local to the class are then performed starting with the outermost prefix.

The program of the outermost prefix is then entered.

The driver for the object is given DISPLAY (BL).mdp as static link (drp) where BL is either the level of the block containing the declaration of the class C (normal case) or the level of the connection of the class D where the class C is declared.

Note:

Since the class or block where C is declared must have local class declarations (C is one of these), it must always have a master driver.

The procedures in the formal description relating to generating references are:

BC	begin class
BCR	begin class return
ECB	end class body

In addition, the procedures "detach", "resume" and "attach" may be considered related to generating references.

ECB has been previously described.

```
procedure BC (x,slx,acs);
   ref (prototype) x; ref (object) slx;
   ref (object) acs;
   begin ref (driver) y; ref (prototype) q;
      comment begin class;
      y :- new driver (new object (x),slx.MDP,exit,CD,acs,
                                                 true,x.level);
      y.ob := true; y.obj.MDP :- y;
      if x.nrp ≠ 0 then
         begin y :- new driver (CD.obj,CD.drp,none,y,none,
                                                 false,CD.level);
            y.con := CD.con;
            y.cdrp :- CD.cdrp;
            CD :- y;
            DDISPLAY[CD.level] :- CD;
            go to exit
         end else CD :- y;
            update display;
            go to CD.obj.PP.prefix[0].declare;
      end BC;
```

ENTER has been described previously (section 7.2.1).

BC is entered to create data storage for the instance of the class declaration, make necessary drivers, transfer parameters and enter local declarations of the outermost prefix.

The parameters have been checked by the compiler, and they are stored into the class body by in-line coding in the compiled program. A temporary (thunk) driver is used during this evaluation. Return to the runtime system after the parameter evaluation is by the procedure ENTER.

DISPLAY is updated, and the declaration coding in outermost prefix is entered.

The coding for declarations in each prefix will end by a call on BCR, for a prefixed block BPBR.

```
procedure BCR (q); integer q;
   begin ref (prototype) x,y;
      comment begin class return;
      x :- cd.obj.PP;
      y :- x.prefix[q+1];
      if y =/= none then go to y.declare;
      go to x.prefix[0].statements;
end BCR;
```

The subroutine BCR is used to locate the next class where declarations should be performed. When declarations in the innermost class have been processed, the outermost prefix is located and the first statement entered.

DISPLAY is updated prior to entry of the declaration part.

Return from declarations in a prefixed block is through BPBR.

7.3.3 Instantaneous qualification

An instantaneous qualification (X qua C) will always result in a runtime check. It is verified that X is of class C or a subclass of C.

The case of X being none is handled by compiler generated coding as usual.

The subroutine CIQ (check instantaneous qualification) is used to check the validity of the instantaneous qualification.

The subroutine either gives an errormessage or acts as a ref(C) procedure whose value is X.

```
ref procedure CIQ(x,c);
   ref (object) x; ref (prototype) c;
      begin ref (prototype) d;
         d :- x.PP;
         if d.plev < c.plev then error ("ciq",1);
         if d.prefix[c.plev] =/= c then error ("ciq",2);
         CIQ :- x
   end CIQ;
```

Possible error: the object X is not of class C or a subclass of C.

The subroutine will conceptually follow the prefix chain in the prototypes starting on the prototype for the class of X. If the prototype for the class C is not found during this scan, a runtime error condition exists.

The procedure value is a reference to the block instance.

7.4 Character_expressions

Character expressions are in the language only as character constants, character variables and character procedures.

The character procedure "char" is system defined.

The character procedure "getchar" is defined local to the type text.

7.5 Text_expressions

Text expressions are in the language as text constants, text variables and text procedures.

The text procedures "copy" and "blanks" are system defined.

The text procedures, "sub", "main" and "strip" are attributes of any text.