411D)66 ECL.6

```
'The following are presumed primitive routines, corresponding to
machine-coded routines, or micro-coded instructions in the
implementation of an ECL system.
';

FLUSHMD(CS\FRAME);

FLUSHMD(CONTROL\STACK);

FLUSHFIX("ERROR");

SIZE\OF ← EXPR(M:MODE, A:ADDRESS; INT) 1;

REAL\EVAL ← EVAL;

ADDRESS\GPROC ←
  EXPR(B\:BOOL, S\:SYMBOL, L\:FORM; ADDRESS)
    BEGIN
      B\ => ERROR();
      S\ # "OF" => ERROR();
      DECL X:ANY BYVAL REAL\EVAL(L\.CAR);
      LIFT(CONST(ADDRESS.UR OF ALLOC(ANY BYVAL X), 0), ADDRESS);
    END;

ADDRESS ←
  QL("ADDRESS", NIL, NIL, NIL, NIL, ADDRESS\GPROC) ::
    STRUCT(V:REF, VP:INT);

INT\OF ←
  EXPR(A:ADDRESS; INT) [) A.VP = 0 => VAL(A.V); VAL(VS[A.VP]) (];

BOOL\OF ←
  EXPR(A:ADDRESS; BOOL)
    [) A.VP = 0 => VAL(A.V); VAL(VS[A.VP]) (];

MODE\OF ←
  EXPR(A:ADDRESS; MODE)
    [) A.VP = 0 => VAL(A.V); VAL(VS[A.VP]) (];

REF\OF ←
  EXPR(A:ADDRESS; REF) [) A.VP = 0 => VAL(A.V); VAL(VS[A.VP]) (];
```

```
SYMBOL\OF ←
  EXPR(A:ADDRESS; SYMBOL)
    [) A.VP = 0 => VAL(A.V); VAL(VS[A.VP]) (];

IS\VS\ADDRESS ← EXPR(A:ADDRESS; BOOL) A.VP # 0;

VS\ADDRESS ←
  EXPR(VP:INT; ADDRESS)
    LIFT(CONST(ADDRESS.UR OF NIL, VP), ADDRESS);

COPY ←
  EXPR(NEW\ADDR:ADDRESS, OLD\ADDR:ADDRESS, N:INT)
    BEGIN
      NEW\ADDR.VP # 0 AND OLD\ADDR.VP # 0 =>
        VS[NEW\ADDR.VP] ← ALLOC(ANY BYVAL VAL(VS[OLD\ADDR.VP]));
      NEW\ADDR.VP # 0 =>
        VS[NEW\ADDR.VP] ← ALLOC(ANY BYVAL VAL(OLD\ADDR.V));
      OLD\ADDR.VP # 0 => VAL(NEW\ADDR.V) ← VAL(VS[OLD\ADDR.VP]);
      VAL(NEW\ADDR.V) ← VAL(OLD\ADDR.V);
    END;

EXECUTE ←
  EXPR(P:REF)
    BEGIN
      DECL L:FORM;
      SHARED NP BY - 1 TO CS[CP].NP
        REPEAT
          OR((A ← NS[NP].MD) = "INT",
             A = "ATOM",
             A = "REAL",
             A = "REF",
             A = "DTPR",
             A = "DDB",
             A = "NONE") => L ← ALLOC(DTPR OF NS[NP].ADDR.V, L);
        END;
      L ← ALLOC(DTPR OF P, L);
      DECL Q:REF LIKE ALLOC(ANY BYVAL REAL\EVAL(L));
      RETURN\RESULT(MD(VAL(Q)), CONST(ADDRESS OF Q));
    END;

IS\PURE ←
  EXPR(; BOOL)
    BEGIN
      NOT IS\VS\ADDRESS(B) => FALSE;
      B GT VS\ADDRESS(CS[CP].CVP) => TRUE;
      FALSE;
    END;
```

```
REAL\GT ← GT;

GT ←
  EXPR(A:ANY, B:ANY; BOOL)
    GENERIC(A, B)
      [ARITH, ARITH] => REAL\GT(A, B);
      [ADDRESS, ADDRESS] => REAL\GT(A.VP, B.VP);
      TRUE => ERROR();
    END;

↑;
```

'The following constitute the ECL model proper. The objective is that it run in any given ECL implementation.

The model is currently grossly incomplete and undebugged.

```
';

ERROR ← EXPR() BREAK("ERROR CALLED");

NS\VAL ← EXPR(NP:INT; ANY) VAL(NS[NP].ADDR.V);

ECL\EVAL ←
  EXPR(F:FORM)
    BEGIN
      F = NIL => NO\RESULT();
      DECL M:MODE LIKE MD(VAL(F));
      M = ATOM =>
        BEGIN
          DECL I:INT LIKE FIND\NAME(NP, F);
          I # 0 => [) A ← NS[I].MD; B ← NS[I].ADDR (];
          F.TLB = NIL => NO\RESULT();
          A ← MD(VAL(F.TLB));
          B.V ← F.TLB;
          B.VP ← 0;
        END;
      M = DTPR =>
        BEGIN
          MD(VAL(F.CAR)) # ATOM => APPLY(F);
          DECL S:SYMBOL LIKE F.CAR;
          S = "BEGIN" => EVAL\BLOCK(F);
          S = "EXPR" => RETURN\PTR(F, FORM);
          S = "FOR" => EVAL\FOR(F);
          S = "[" => EVAL\SEL(F);
          S = "." => EVAL\SELQ(F);
          S = "←" => EVAL\ASSIGN(F);
          S = "->" => EVAL\COND(F, TRUE, FALSE);
          S = "=>" => EVAL\COND(F, TRUE, TRUE);
          S = "+>" => EVAL\COND(F, FALSE, FALSE);
          S = "#>" => EVAL\COND(F, FALSE, TRUE);
          S = "DECL" => EVAL\DECL(F);
          S = "LIFT" => EVAL\LIFT(F);
          S = "LOWER" => EVAL\LOWER(F);
          S = "GENERIC" => EVAL\GENERIC(F);
          S = "CONST" => EVAL\CONST(F);
          S = "ALLOC" => EVAL\ALLOC(F);
          S = "PROC" => EVAL\PROC(F);
          S = "STRUCT" => EVAL\STRUCT(F);
          S = "PTR" => EVAL\PTR(F);
          S = "SEQ" => EVAL\SEQ(F);
          S = "VECTOR" => EVAL\VECTOR(F);
          S = "ONEOF" => EVAL\ONEOF(F);
```

```
S = "SPECIAL" =>
  [) A ← VAL(F.CDR).A; B ← VAL(F.CDR).B (];
APPLY(F);
```

```
        END;
      M = REAL OR M = INT =>
        RETURN\RESULT(M, CONST(ADDRESS OF VAL(F)));
      M = REF =>
        RETURN\RESULT(MD(VAL(VAL(F))),
                      CONST(ADDRESS OF VAL(VAL(F))));
      M = DDB => RETURN\PTR(F, MODE);
    END;

RETURN\PTR ←
  EXPR(R:FORM, M:MODE)
    BEGIN
      A ← M;
      B ← VS\ADDRESS(VP + 1);
      COPY(B, CONST(ADDRESS OF R), 1);
      VP ← VP + 1;
    END;

FIND\NAME ←
  EXPR(NS\INDEX:INT, NAME:SYMBOL; INT)
    BEGIN
      DECL IX:INT;
      FOR I FROM NS\INDEX BY - 1 TO 1
        REPEAT NS[I].NAME = NAME => IX ← I END;
      IX;
    END;

NO\RESULT ← EXPR() [) A ← NIL; B ← NO\ADDRESS (];

EVAL\DECL ←
  EXPR(F:FORM)
    BEGIN
      DECL MUST\COPY:BOOL;
      VP ← CS[CP].CVP;
      EVAL\CONST(F ← F.CDR);
      INSTALL();
      MUST\COPY -> RETURN\RESULT(A, CONST(ADDRESS OF VAL(B.V)));
      DECL ID\LIST:FORM BYVAL F.CAR;
      ID\LIST = NIL => NOTHING;
      REPEAT
        NS[NP].NAME ← ID\LIST.CAR;
        CS[CP].CVP ← VP;
        ID\LIST.CDR = NIL => NOTHING;
        ID\LIST ← ID\LIST.CDR;
        INSTALL();
      END;
    END;
```

```
INSTALL ← EXPR() NS[NP ← NP + 1] ← CONST(NS\FRAME OF NIL, A, B);

EVAL\CONST ←
  EXPR(F:FORM)
    BEGIN
      ECL\EVAL(F.CAR);
      A # MODE => ERROR();
      DECL M:MODE BYVAL MODE\OF(B);
      VP ← CS[CP].CVP;
      DECL BC:SYMBOL BYVAL F.CDR.CAR;
      M.NO\GEN\FLG AND M = A => [) BC = "BYVAL" => PURIFY() (];
      DECL USER\GEN\FN:REF BYVAL
        [) M.UFN # NIL => M.UFN[5]; NIL (];
      BC = "SHARED" OR BC = "LIKE" OR BC = "BYVAL" =>
        BEGIN
          ECL\EVAL(F.CDR.CDR.CAR);
          DECL LOCATION:ADDRESS BYVAL B;
          USER\GEN\FN # NIL ->
            BEGIN
              DECL F:FORM BYVAL
                ALLOC(DTPR OF
                      VAL(USER\GEN\FN),
                      ALLOC(DTPR OF
                            BC,
                            ALLOC(DTPR OF
                                  ALLOC(DTPR OF
                                        "SPECIAL",
                                        ALLOC(ANY BYVAL
                                              ALLOC(AB\PAIR BYVAL
                                                    CONST(AB\PAIR OF
                                                          A, B)))),
                                  ALLOC(DTPR OF M, NIL))));
              APPLY(F);
              BC = "SHARED" AND B # LOCATION => ERROR();
            END;
          COVERS(M, A) => [) BC = "BYVAL" => PURIFY() (];
          DECL USER\CONV\FN:REF BYVAL
            [) A.UFN = NIL => NIL; A.UFN[1] (];
          BEGIN
            USER\CONV\FN # NIL =>
              BEGIN
                DECL F:FORM BYVAL
                  ALLOC(DTPR OF
                        VAL(USER\CONV\FN),
                        ALLOC(DTPR OF
                              ALLOC(DTPR OF
                                    "SPECIAL",
                                    ALLOC(ANY BYVAL
                                          ALLOC(AB\PAIR BYVAL
                                                CONST(AB\PAIR OF
                                                      A, B)))),
               ALLOC(DTPR OF M, NIL)));
  APPLY(F);
END;
```

```
        SYS\CONV(M);
      END;
      NOT COVERS(M, A) => ERROR();
      BC = "BYVAL" => PURIFY();
    END;
  DECL CVP:INT BYVAL CS[CP].CVP;
  DECL M1:MODE;
  DECL N:INT;
  BC = NIL OR BC = "SIZE" =>
    BEGIN
      DECL V:SEQ(INT) BYVAL
        EVALUATE\DOPE\VECTOR([) M.LRFLG => 0; M.SZ (],
                            BEGIN
                              BC = NIL => NIL;
                              F.CDR.CDR.CAR;
                            END);
      M.CLASS = "ROW" =>
        BEGIN
          M1 ← M.D.SBMD;
          DECL IS\SEQ:BOOL BYVAL M.D.RLPT = - 1;
          CS[CP].CVP ← VP ← VP + [) IS\SEQ => 1; 0 (];
          N ←
            BEGIN
              IS\SEQ AND LENGTH(V) # 0 => V[1];
              NOT IS\SEQ => M.D.RLPT;
              0;
            END;
          N LT 0 => ERROR();
          BEGIN
            NOT M1.LRFLG =>
              BEGIN
                N GT 0 ->
                  BEGIN
                    EV\CONST(M1,
                             "SIZE",
                             SUB\VECTOR(V,
                                        [) IS\SEQ => 2; 1 (],
                                        LENGTH(V)));
                    SLIDE\VS(CS[CP].CVP);
                    CS[CP].CVP ← VP;
                  END;
                TO N - 1
                  REPEAT
                    COPY(VS\ADDRESS(VP + 1),
                         B,
                         SIZE\OF(A, B));
                    VP ← VP + SIZE\OF(A, B);
                  END;
                CS[CP].CVP ← VP;
              END;
            CS[CP].CVP ← VP ← VP + M1.SZ;
            N GT 0 -> EV\CONST(M1);
```

```
FOR J TO N
  REPEAT
    COPY(LOCATE(M, VS\ADDRESS(CVP)), B, M1.SZ);
```

```
            END;
          VP ← CS[CP].CVP;
        END;
        IS\SEQ ->
          BEGIN
            SEQ\HEADER.N\COMP ← N;
            SEQ\HEADER.N\WORDS ← VP - CVP + 1;
            COPY(VS\ADDRESS(CVP),
                 CONST(ADDRESS OF SEQ\HEADER),
                 1);
          END;
      END;
    M.CLASS = "STRUCT" =>
      BEGIN
        CS[CP].CVP ← VP ← VP + FIXED\SIZE(M);
        DECL I:INT;
        FOR J TO LENGTH(VAL(M.D))
          REPEAT
            EV\CONST(M.D[J].TYPE,
                     "SIZE",
                     SUB\VECTOR(V,
                                I,
                                BEGIN
                                  M.D[J].TYPE.LRFLG => 0;
                                  M.D[J].TYPE.SZ;
                                END));
            M.D[J].TYPE.LRFLG ->
              BEGIN
                COPY(LOCATE(M, J, VS\ADDRESS(CVP)),
                     B,
                     SIZE\OF(A, B));
                VP ← CS[CP].CVP;
              END;
            NOT M.D[J].TYPE.LRFLG ->
              BEGIN
                SET\INTERNAL\POINTER(VS\ADDRESS(CVP),
                                     M,
                                     J,
                                     VS\ADDRESS(CS[CP].CVP +
                                                1));
                SLIDE\VS(CS[CP].CVP);
                CS[CP].CVP ← VP;
                I ← I + M.D[J].TYPE.SZ;
              END;
          END;
      END;
    M.CLASS = "PTR" =>
      RETURN\RESULT(M, CONST(ADDRESS OF NIL));
    M.CLASS = "BASIC" => DEFAULT\VALUE(M);
  END;
BC = "OF" =>
  BEGIN
```

```
M.CLASS = "ROW" AND M.D.RLPT = - 1 =>
  BEGIN
    N ← LIST\LENGTH(V);
```

```
      CS[CP].CVP ← VP ← VP + 1;
      EVAL\COMPONENTS(N);
      SEQ\HEADER.N\COMP ← N;
      SEQ\HEADER.N\WORDS ← VP - CVP + 1;
      COPY(VS\ADDRESS(CVP),
           CONST(ADDRESS OF SEQ\HEADER),
           1);
    END;
  M.CLASS = "ROW" =>
    BEGIN
      DECL NS:INT BYVAL LIST\LENGTH(V);
      N ← M.D.RLPT;
      EVAL\COMPONENTS([) NS LT N => NS; N (]);
      DECL D:SEQ(INT) BYVAL DOPE\VECTOR\OF(A, B);
      TO N - NS
        REPEAT
          EV\CONST(M1, "SIZE", D);
          DOPE\VECTOR\OF(A, B) # D => ERROR();
          SLIDE\VS(CS[CP].CVP);
          CS[CP].CVP ← VP;
        END;
    END;
  M.CLASS = "STRUCT" =>
    BEGIN
      DECL NS:INT BYVAL LIST\LENGTH(V);
      N ← LENGTH(VAL(M.D));
      CS[CP].CVP ← VP ← VP + FIXED\SIZE(M);
      FOR J TO [) NS LT N => NS; N (]
        REPEAT
          EV\CONST(M.D[J].TYPE, "BYVAL", V.CAR);
          V ← V.CDR;
          M.D[J].TYPE.LRFLG ->
            BEGIN
              COPY(LOCATE(M, J, VS\ADDRESS(CVP)),
                   B,
                   SIZE\OF(A, B));
              VP ← CS[CP].CVP;
            END;
          NOT M.D[J].TYPE.LRFLG ->
            BEGIN
              SET\INTERNAL\POINTER(VS\ADDRESS(CVP),
                                   M,
                                   J,
                                   VS\ADDRESS(CS[CP].CVP));
              SLIDE\VS(CS[CP].CVP);
              CS[CP].CVP ← VP;
            END;
        END;
      N GT NS ->
        FOR J FROM NS + 1 TO N
          REPEAT
            EV\CONST(M.D[J].TYPE, NIL, NIL);
```

```
M.D[J].TYPE.LRFLG ->
  BEGIN
    COPY(LOCATE(M, J, VS\ADDRESS(CVP)),
                                B,
                                SIZE\OF(A, B));
                           VP ← CS[CP].CVP;
                         END;
                       NOT M.D[J].TYPE.LRFLG ->
                         BEGIN
                           SET\INTERNAL\POINTER(VS\ADDRESS(CVP),
                                                M,
                                                J,
                                                VS\ADDRESS(CS[CP].CVP));
                           SLIDE\VS(CS[CP].CVP);
                           CS[CP].CVP ← VP;
                         END;
                     END;
               END;
             ERROR();
           END;
       END;

   EVAL\COMPONENTS ←
     EXPR(N:INT)
       BEGIN
         ECL\EVAL(V.CAR);
         SLIDE\VS(CS[CP].CVP);
         PURIFY();
         CS[CP].CVP ← VP;
         DECL D:SEQ(INT) BYVAL DOPE\VECTOR\OF(A, B);
         TO N - 1
           REPEAT
             V ← V.CDR;
             ECL\EVAL(V.CAR);
             SLIDE\VS(CS[CP].CVP);
             PURIFY();
             CS[CP].CVP ← VP;
             D # DOPE\VECTOR\OF(A, B) => ERROR();
           END;
       END;

   LIST\LENGTH ←
     EXPR(L:FORM BYVAL; INT)
       BEGIN
         L = NIL => 0;
         DECL R:INT BYVAL 1;
         REPEAT L ← L.CDR; L = NIL => R; R ← R + 1 END;
       END;
```

```
PURIFY ←
  EXPR()
    BEGIN
      IS\PURE() => NOTHING;
      RETURN\RESULT(A, CONST(ADDRESS OF VAL(B.V)));
      SLIDE\VS(CS[CP].CVP);
    END;

EV\CONST ←
  EXPR(M:MODE, BC:SYMBOL, V:ANY)
    BEGIN
      DECL F:FORM BYVAL
        ALLOC(DTPR OF
              M, ALLOC(DTPR OF BC, ALLOC(DTPR OF V, NIL)));
      EVAL\CONST(F);
    END;

APPLY ←
  EXPR(F:FORM)
    BEGIN
      ECL\EVAL(F.CAR);
      DECL P:REF BYVAL REF\OF(B);
      DECL TYPE:SYMBOL BYVAL
        BEGIN
          P = NIL => ERROR();
          MD(VAL(P)) = DTPR AND P.CAR = "EXPR" => "EXPR";
          MD(VAL(P)) = CEXPR => "CEXPR";
          MD(VAL(P)) = SUBR => "SUBR";
          ERROR();
        END;
      CS[CP ← CP + 1] ←
        CONST(CS\FRAME OF
              VP,
              VP,
              NP,
              TYPE,
              P,
              [) TYPE = "EXPR" => P.CDR.CDR.CDR.CAR; NIL (]);
      BIND\FORMALS(F.CDR, P);
      EVAL\DECL(NIL,
                [) TYPE = "EXPR" => P.CDR.CDR.CAR; P.RETYPE (],
                "LIKE",
                [) TYPE = "EXPR" => CS[CP].LC; EXECUTE(P) (]);
      A ← NS[NP].MD;
      B ← NS[NP].ADDR;
      SLIDE\VS(CS[CP].VP);
      NP ← CS[CP].NP;
      CP ← CP - 1;
    END;
```

```
SLIDE\VS ←
  EXPR(TOP\VP:INT)
    BEGIN
      IS\VS\ADDRESS(B) AND B GT VS\ADDRESS(TOP\VP) =>
        BEGIN
          DECL N:INT BYVAL SIZE\OF(A, B);
          DECL NEW\ADDR:ADDRESS BYVAL VS\ADDRESS(TOP\VP + 1);
          COPY(NEW\ADDR, B, N);
          B ← NEW\ADDR;
          VP ← TOP\VP + N;
        END;
      VP ← CS[CP].VP;
    END;

BIND\FORMALS ←
  EXPR(ARG:FORM BYVAL, P:REF)
    BEGIN
      MD(VAL(P)) = DTPR =>
        BEGIN
          DECL FML:FORM BYVAL P.CDR.CAR;
          REPEAT
            FML = NIL => NOTHING;
            DECL ARG\VAL:FORM BYVAL
              [) ARG # NIL => ARG.CAR; NIL (];
            DECL BC:SYMBOL BYVAL FML.CAR.CDR.CDR.CAR;
            BC = "LISTED" OR BC = "UNEVAL" ->
              BEGIN
                ECL\EVAL(FML.CAR.CDR.CAR);
                A # MODE => ERROR();
                CONST(ADDRESS OF FORM) # B => ERROR();
                BC = "LISTED" -> [) ARG\VAL ← ARG; ARG ← NIL (];
                NS[NP ← NP + 1] ←
                  CONST(NS\FRAME OF
                        FML.CAR.CAR,
                        FORM,
                        CONST(ADDRESS OF ARG\VAL));
              END;
            BC = "LIKE" OR BC = "SHARED" OR BC = "BYVAL" ->
              EVAL\DECL(ALLOC(DTPR BYVAL
                              CONST(DTPR OF FML.CAR.CAR, NIL)),
                        FML.CAR.CDR.CAR,
                        BC,
                        ARG\VAL);
            FML ← FML.CDR;
            ARG # NIL -> ARG ← ARG.CDR;
          END;
        END;
      ERROR();
    END;
```

```
RETURN\RESULT ←
  EXPR(M:MODE, ADDR:ADDRESS)
    BEGIN
      A ← M;
      DECL N:INT BYVAL SIZE\OF(M, ADDR);
      B ← VS\ADDRESS(VP + 1);
      COPY(B, ADDR, N);
      VP ← VP + N;
    END;

EVAL\FOR ←
  EXPR(F:FORM BYVAL)
    BEGIN
      CS[CP ← CP + 1] ←
        CONST(CS\FRAME OF VP, VP, NP, "FOR", F, F ← F.CDR);
      DECL FLAG:BOOL                    /* 'TRUE IF NO OPTIONS';
      DECL EXIT:BOOL                    /* 'TRUE IF EXIT CONDITIONAL WORKS';
      DECL LC:FORM SHARED CS[CP].LC;
      DECL CNP:INT BYVAL CS[CP].NP;
      DECL CVP:INT SHARED CS[CP].CVP;
      DECL ID:SYMBOL;
      DECL NXT:SYMBOL;
      DECL STEP, FV:INT;
      DECL TEST:BOOL;
      DECL NEXT:ROUTINE LIKE EXPR() NXT ← (LC ← LC.CDR).CAR;
      DECL EVAL\INT:ROUTINE LIKE
        EXPR()
          [) ECL\EVAL((LC ← LC.CDR).CAR); NEXT(); INT\OF(B) (];
      NXT ← LC.CAR;
      NXT # "REPEAT" ->
        BEGIN
          NXT = "FOR" -> [) NEXT(); ID ← NXT; NEXT() (];
          EV\CONST(INT,
                   BEGIN
                     NXT = "FROM" => "BYVAL";
                     OR(NXT = "BYVAL",
                        NXT = "LIKE",
                        NXT = "SHARED") => NXT;
                     NIL;
                   END,
                   (LC ← LC.CDR).CAR);
          DECL V:INT SHARED NS\VAL(NP);
          CNP ← NP;
          NOT FLAG -> V ← 1;
          STEP ← 1;
          NEXT();
          NXT = "BY" -> STEP ← EVAL\INT();
          NXT = "TO" -> [) FV ← EVAL\INT(); TEST ← STEP GE 0 (];
        END;
      DECL LOOP:FORM BYVAL LC;
      NO\RESULT();
```

```
REPEAT
  EXIT OR STEP # 0 AND (V = FV OR V GT FV = TEST) =>
    NOTHING;
```

```
        LC ← LOOP;
        REPEAT
          (LC ← LC.CDR) = NIL OR EXIT => NOTHING;
          VP ← CVP;
          ECL\EVAL(LC.CAR);
          NP ← CNP;
        END;
      END;
      SLIDE\VS(CVP);
      NP ← CS[CP].NP;
      CP ← CP - 1;
    END;

EVAL\BLOCK ←
  EXPR(F:FORM)
    BEGIN
      CS[CP ← CP + 1] ←
        CONST(CS\FRAME OF VP, VP, NP, "BLOCK", F, F.CDR);
      NO\RESULT();
      DECL EXIT:BOOL;
      DECL CVP:INT SHARED CS[CP].CVP;
      DECL LC:FORM SHARED CS[CP].LC;
      REPEAT
        EXIT OR LC = NIL => NOTHING;
        VP ← CVP;
        ECL\EVAL(LC.CAR);
        LC ← LC.CDR;
      END;
      SLIDE\VS(CS[CP].VP);
      NP ← CS[CP].NP;
      CP ← CP - 1;
    END;

EVAL\COND ← QL();

EVAL\ASSIGN ←
  EXPR(F:FORM)
    BEGIN
      ECL\EVAL(F.CDR.CAR);
      DECL USER\ASSIGN\FN:REF BYVAL
        [) A.UFN # NIL => A.UFN[2]; NIL (];
      USER\ASSIGN\FN # NIL =>
        BEGIN
          DECL P:FORM BYVAL
            ALLOC(DTPR OF
                  VAL(USER\ASSIGN\FN),
                  ALLOC(DTPR OF
                        ALLOC(DTPR OF
                              "SPECIAL",
                              ALLOC(ANY BYVAL
                                    ALLOC(AB\PAIR BYVAL
                                          CONST(AB\PAIR OF A, B)))),
          ALLOC(DTPR BYVAL
               CONST(DTPR BYVAL VAL(F.CDR.CDR)))));
APPLY(P);
```

```
          END;
        DECL DEST:AB\PAIR BYVAL CONST(AB\PAIR OF A, B);
        ECL\EVAL(F.CDR.CDR.CAR);
        NOT COVERS(DEST.A, A) => ERROR();
        DECL DEST\N:INT BYVAL SIZE\OF(DEST.A, DEST.B);
        DECL N:INT BYVAL SIZE\OF(A, B);
        N # DEST\N => ERROR();
        COPY(DEST.B, B, N);
        A ← DEST.A;
        B ← DEST.B;
      END;

EVAL\LIFT ←
  EXPR(F:FORM)
    BEGIN
      ECL\EVAL(F.CDR.CDR.CAR);
      A # MODE => ERROR();
      DECL M, M1:MODE BYVAL MODE\OF(B);
      VP ← CS[CP].CVP;
      ECL\EVAL(F.CDR.CAR);
      REPEAT
        M1.UR = NIL => ERROR();
        M1.UR = A => A ← M;
        M1 ← M1.UR;
      END;
    END;

EVAL\LOWER ←
  EXPR(F:FORM)
    [) ECL\EVAL(F.CDR.CAR); A.UR = NIL => ERROR(); A ← A.UR (];

SEQ\HEADER ← SEQ\HEADER :: STRUCT(N\COMP:INT, N\WORDS:INT);

NSPFN ←
  EXPR(X:NS\FRAME)
    BEGIN
      PRINT(X.NAME);
      PRINT(%:);
      PRINT(X.MD);
      PRINT(' = ');
      PRINT(VAL(X.ADDR));
      PRINT('
');
    END;
```

```
VSPFN ← EXPR(X:REF) [) PRINT(VAL(X)); PRINT('
') (];

CSPFN ←
  EXPR(X:CS\FRAME)
    BEGIN
      PRINT(X.VP);
      PRINT(%         );
      PRINT(X.CVP);
      PRINT(%         );
      PRINT(X.NP);
      PRINT(%         );
      PRINT(X.TYPE);
      PRINT('
');
    END;

NS\FRAME ←
  QL("NS\FRAME", NIL, NIL, NIL, NSPFN) ::
    STRUCT(NAME:SYMBOL, MD:MODE, ADDR:ADDRESS);

NAME\STACK ← NAME\STACK :: SEQ(NS\FRAME);

QL("CS\FRAME", NIL, NIL, NIL, CSPFN) ←
  CS\FRAME ::
    STRUCT(VP:INT,
           CVP:INT,
           NP:INT,
           TYPE:SYMBOL,
           FM:REF,
           LC:FORM);

CONTROL\STACK ← CONTROL\STACK :: SEQ(CS\FRAME);

AB\PAIR ← AB\PAIR :: STRUCT(A:MODE, B:ADDRESS);

EVAL\SEL ←
  EXPR(F:FORM)
    BEGIN
      ECL\EVAL(F.CDR.CAR);
      DECL USER\SEL\FN:REF BYVAL
        [) A.UFN # NIL => A.UFN[3]; NIL (];
      DECL OBJ:AB\PAIR BYVAL CONST(AB\PAIR OF A, B);
      USER\SEL\FN # NIL =>
        BEGIN
          DECL P:FORM BYVAL
            ALLOC(DTPR OF
                  VAL(USER\SEL\FN),
                  ALLOC(DTPR OF
                        ALLOC(DTPR OF
                              "SPECIAL",
                        ALLOC(AB\PAIR BYVAL OBJ)),
                ALLOC(DTPR OF F.CDR.CDR.CAR, NIL)));
APPLY(P);
```

```
        END;
      ECL\EVAL(F.CDR.CDR.CAR);
      DECL IX:INT BYVAL
        BEGIN
          OBJ.A.CLASS = "STRUCT" AND
            (A = SYMBOL OR A = PTR(ATOM)) =>
            FIELD\TO\IX(OBJ.A, SYMBOL\OF(B));
          EV\CONST(INT,
                   "LIKE",
                   ALLOC(DTPR OF
                         "SPECIAL",
                         ALLOC(AB\PAIR BYVAL
                               CONST(AB\PAIR OF A, B))));
          INT\OF(B);
        END;
      LOCATE(OBJ.A, OBJ.B, IX);
    END;

DEFAULT\VALUE ←
  EXPR(M:MODE)
    BEGIN
      M.CLASS = "BASIC" =>
        RETURN\RESULT(M,
                      CONST(ADDRESS OF
                            BEGIN
                              M = INT => 0;
                              M = REAL => 0.0;
                              M = BOOL => FALSE;
                              M = CHAR => INT\CHAR(0);
                              M = NONE => NOTHING;
                            END));
      ERROR();
    END;

↑;
```

```
'The following constitutes an environment for. interpretation of
an ECL process.
';

A ← CONST(MODE);

VS ← CONST(SEQ(REF) SIZE 100);

NS ← CONST(NAME\STACK SIZE 100);

CS ← CONST(CONTROL\STACK SIZE 100);

NP ← 0;

CP ← 0;

VP ← 0;

NO\ADDRESS ← LIFT(CONST(ADDRESS.UR OF NIL, 0), ADDRESS);

B ← NO\ADDRESS;

EVAL\COND <-
  EXPR(F:FORM BYVAL, TEST:BOOL, CLAUSE:BOOL)
    BEGIN
      CLAUSE AND (CS[CP].TYPE = "BLOCK" OR CS[CP].TYPE = "FOR") #>
        ERROR();
      ECL\EVAL((F ← F.CDR).CAR);
      A # BOOL => ERROR();
      BOOL\OF(B) # TEST => NO\RESULT;
      ECL\EVAL(F.CDR.CAR);
      CLAUSE -> EXIT ← TRUE;
    END;

FS <-
  EXPR(OLD:FORM UNEVAL, NEW:FORM UNEVAL, L:FORM BYVAL; FORM)
    BEGIN
      DECL OUT:FORM;
      DECL Q:FORM LIKE
        CONS("RA",
             CONS(CONS("<*", CONS(OLD, CONS(NEW))),
                  CONS(EXPR(F:FORM) F)));
      REPEAT
        L = NIL => NOTHING;
        DECL S:FORM LIKE L.CAR;
        MVAL(S) = ATOM AND
          (MVAL(S.TLB) = FORM OR
            MVAL(S.TLB) = ROUTINE AND MVAL(VAL(S.TLB)) = DTPR) ->
          BEGIN
            OUT ← CONS(S, OUT);
            DECL P:PTR(STRING) LIKE ALLOC(STRING SIZE 50);
```

```
POPORT ← MAKEPF(P, "OUT");
UNPARSE(Q);
UNPARSE(QUOTE(EXIT));
```

```
            CLOSE(POPORT);
            POPORT ← NIL;
            DECL CI:PORT BYVAL CIPORT;
            CIPORT ← MAKEPF(P);
            EDIT(VAL(S.TLB));
            PIPORT ← NIL;
            CIPORT ← CI;
          END;
        L ← L.CDR;
      END;
      OUT;
    END;

MVAL = NOTHING -> LOADB "AIDS";
```

sum(u v) → f(u v)

λ(u v).[issum e → f(u, v)] [

] args[e] args$_2$[e]

issum e → λuv. f

map → f args[e] args[e]

```
COMMENT    VALID 00054 PAGES
C REC  PAGE   DESCRIPTION
C00001 00001
C00005 00002
C00008 00003
C00010 00004
C00011 00005
C00013 00006        INT\OF ← EXPR(S:SITE INT) VAL([) S.V # NIL => S.V VS (][S.VP])
C00015 00007        IS\VS\SITE ← EXPR(A:SITE BOOL) A.V = NIL
C00017 00008        IS\PURE ← EXPR( BOOL) [) NOT IS\VS\SITE(B) => FALSE B GT VS\SITE(CS[CP].C
VP) => TRUE FALSE (]
C00018 00009        SELECT ←
C00020 00010        EV\MVAL ←
C00022 00011        SET\INTERNAL\POINTER ←
C00024 00012        EV\HASH <-
C00027 00013        'The following constitute the ECL model proper.  The objective is
C00029 00014        NO\RESULT ← EXPR() OBJECT ← CONST(BIGREF)
C00030 00015        EVAL\CONST ←
C00033 00016              DECL OBJ:BIGREF BYVAL CONST(BIGREF OF M, VS\SITE(CS[CP].CVP + 1))
C00036 00017                  M.CLASS = "STRUCT" =>
C00038 00018        "       BINDCLASS "OF" OR USER BINDCLASS
C00040 00019                M.CLASS = "STRUCT" =>
C00042 00020        EVAL\COMPONENTS ←
C00044 00021        EVAL\VECTOR ←
C00046 00022        LIST\LENGTH ←
C00048 00023        APPLY ←
C00050 00024        BIND\FORMALS ←
C00052 00025        RETURN\RESULT ←
C00053 00026        EVAL\FOR ←
C00056 00027        EVAL\BLOCK ←
C00058 00028        EVAL\ASSIGN ←
C00060 00029        EVAL\LOWER ← EXPR(F:FORM) [) ECL\EVAL(F.CAR) A.UR = NIL => ERROR() A ←
A.UR (]
C00062 00030        VS\FRAME ← QL("VS\FRAME", NIL, NIL, NIL, VSPFN) :: REF
C00064 00031        FIELD\TO\IX ←
C00066 00032        EVALUATE\DOPE\VECTOR ←
C00067 00033        '
C00069 00034        GETDDB <-
C00072 00035        PTYPE <-
C00074 00036        EVVECT <-
C00075 00037        EVPTR <-
C00077 00038        EVPROC <-
C00079 00039        CMPDDB <-
C00081 00040        FINISH1 <-
C00083 00041        MODECOMPILER <-
C00084 00042        EVDBCL <-
C00087 00043        BLDSSF <-
C00090 00044              'ACCUMULATE DOPE VECTOR LENGTH FOR ALL LENGTH'
C00093 00045              'COUNT WORDS USED FOR INTERNAL PTRS AND BYTE ITEMS'
C00095 00046        EVSTR <-
C00098 00047        '       CONVERSION ROUTINES
C00100 00048        '       USER ROUTINES
```

```
C00101 00049     'The following constitutes an environment for interpretation of
C00103 00050     STBL ← MAKEHASH(SYMBOL, ECL\ATOM, 25)
C00105 00051     '
C00107 00052             ECL\DDB ::
C00108 00053     INIT <-
C00111 00054
C00112 ENDMK
C;
```

```
'
"       ECL MODEL (written in ECL)
"
"               (attributations)
"
"
"       The objectives of the model are:
"
"               1. To provide a semantic description of both the EL1
"                  language and the ECL system, using an interpreter
"                  model written in ECL.
"
"               2. To provide a guide to ECL implementation, to within
"                  choices of representation of data objects and other
"                  primitive constructs in a given implementation.
"
"       The model proper should run on any ECL implementation.  The
"       first set of routines in the following provide a set of
"       primitive constructs allowing the model to run on an ECL
"       machine.  The second set of routines is the model proper, while
"       the third set provides an environment, debugging aids, and
"       unclassified routines (that is, in unsettled status as to
"       whether they are primitive or part of the model proper) for
"       use of the model.
"
';

↑;
```

```
'       The model proper makes use of modes which are explained in
"       the primitive routines and definitions. The main notions
"       are embodied in the modes: BIGREF and SITE.
"
"           A BIGREF denotes a data object. It has two components
"       (and is the counterpart of a REF in EL1): A is the mode of
"       the object, and B is its site. SITE is itself a mode which
"       generalizes the notions of address and byte pointer in the
"       PDP-10 implementation of ECL.
"
"           A SITE serves to locate the value of an object, either
"       on the value stack or in the heap. Objects are generated as
"       one or more consecutive entries on the value stack. The action
"       of ALLOC is to simulate allocation in the heap by packaging the
"       corresponding value stack entries up as a PTR(SEQ(REF))
"       and storing the result in a component of the SITE. Specifically,
"       SITE has two components, V and VP. Iv V is NIL, the object
"       starts at location VS[VP] on the value stack. Otherwise, the
"       object starts at V[VP]. The mode of the object can be known
"       in general only from the mode component of a BIGREF whose other
"       component is the site of the object.
"
';

↑;
```

```
NIL;

'       Load GHM utility routines, if necessary
';

UX = NOTHING -> LOADB "AIDS";

SET\UP(100);

↑;
```

```
'       Here begin the "primitive" constructs enabling
"       the model to run in an ECL system.
';

MVAL ← EXPR(X:ANY; MODE) MD(VAL(X));

SIZE\OF ←
  EXPR(P:BIGREF; INT)
    BEGIN
      DECL M:MODE SHARED P.A;
      M.CLASS = "BASIC" OR M.CLASS = "PTR" => 1;
      M.LRFLG => M.SZ;
      DECL A:SITE SHARED P.B;
      [) A.V # NIL => A.V; VS (][A.VP].N\WORDS;
    END;

REAL\EVAL ← EVAL;

SITE\GPROC ←
  EXPR(B\:BOOL, S\:SYMBOL, L\:FORM; SITE)
    BEGIN
      B\ => ERROR();
      S\ # "OF" => ERROR();
      DECL X:ANY BYVAL REAL\EVAL(L\.CAR);
      LIFT(CONST(SITE.UR OF ALLOC(SEQ(REF) OF ALLOC(ANY BYVAL X)), 1), SITE);
    END;

SITE ← QL("SITE", NIL, NIL, NIL, NIL, SITE\GPROC) :: STRUCT(V:REF, VP:INT);

ADDR <- EXPR(F:FORM; FORM) F;
```

```
INT\OF ← EXPR(S:SITE; INT) VAL([) S.V # NIL => S.V; VS (][S.VP]);

BOOL\OF ← EXPR(S:SITE; BOOL) VAL([) S.V # NIL => S.V; VS (][S.VP]);

MODE\OF ← EXPR(S:SITE; MODE) VAL([) S.V # NIL => S.V; VS (][S.VP]);

REF\OF ← EXPR(S:SITE; REF) VAL([) S.V # NIL => S.V; VS (][S.VP]);

SYMBOL\OF ← EXPR(S:SITE; SYMBOL) VAL([) S.V # NIL => S.V; VS (][S.VP]);

DOPE\VECTOR\OF ←
  EXPR(P:BIGREF; SEQ(INT))
    BEGIN
      DECL M:MODE LIKE P.A;
      DECL N:INT LIKE [) M.LRFLG => 0; M.SZ (];
      DECL S:SEQ(INT) LIKE CONST(SEQ(INT) SIZE N);
      N = 0 => S;
      DECL I:INT;
      DECL DV:ROUTINE LIKE
        EXPR(P:BIGREF BYVAL)
          BEGIN
            DECL M:MODE SHARED P.A;
            M.LRFLG => NOTHING;
            M.CLASS = "PTR" => REPEAT P ← EV\VAL(P); M.CLASS # "PTR" => NOTHING END;
            DECL L:INT LIKE EV\LENGTH(P);
            M.CLASS = "STRUCT" => FOR I TO L REPEAT DV(SELECT(P, I)) END;
            M.D.LENGTH LT 0 => [) S(I ← I + 1) ← L; I = N -> RETURN() (];
            L = 0 => DV([) EV\CONST(M.D.TYPE); OBJECT (]);
            DV(SELECT(P, 1));
          END;
      << DV(P);
      S;
    END;
```

```
IS\VS\SITE ← EXPR(A:SITE; BOOL) A.V = NIL;

VS\SITE ← EXPR(VP:INT; SITE) LIFT(CONST(SITE.UR OF NIL, VP), SITE);

COPY ←
  EXPR(NEW:SITE, OLD:SITE, N:INT)
    BEGIN
      DECL T:ANY LIKE
        BEGIN
          NEW.V = NIL AND NEW.VP # 0 => VS;
          NEW.VP # 0 => VAL(NEW.V);
          NEW.VP ← 1;
          VAL(NEW.V ← ALLOC(SEQ(REF) SIZE N));
        END;
      DECL S:ANY SHARED [) OLD.V = NIL => VS; VAL(OLD.V) (];
      DECL I:INT BYVAL [) OLD.VP = 0 => 1; OLD.VP (];
      DECL J:INT BYVAL NEW.VP;
      TO N REPEAT T[J] ← ALLOC(ANY BYVAL VAL(S[I])); I ← I + 1; J ← J + 1 END;
    END;

MKFM ←
  EXPR(X:ANY; FORM)
    BEGIN
      DECL M:MD(X);
      OR(M = INT, M = REAL, M = CHAR, M = REF, M = DTPR, M = DDB, M = NONE) => ALLOC(ANY BY
VAL X);
      ALLOC(REF BYVAL ALLOC(ANY BYVAL X));
    END;

EXECUTE ←
  EXPR(P:REF; ANY)
    BEGIN
      MVAL(P) = ECL\SUBR => [) REAL\EVAL(P.BODY); MKFM(VAL(GETREF(OBJECT))) (];
      DECL L:FORM;
      SHARED NP BY - 1 TO CS[CP - 1].NP + 1
        REPEAT DECL X:ANY LIKE NS\VAL(NP); L ← ALLOC(DTPR OF MKFM(X), L) END;
      MKFM(REAL\EVAL(ALLOC(DTPR OF F.CAR, L)));
    END;
```

```
IS\PURE ← EXPR(; BOOL) [) NOT IS\VS\SITE(B) => FALSE; B GT VS\SITE(CS[CP].CVP) => TRUE; FALS
E (];

REAL\GT = NOTHING -> "REAL\GT".TLB ← ALLOC(ANY BYVAL GT);

GT ←
  EXPR(A:ANY, B:ANY; BOOL)
    GENERIC(A, B)
      [ARITH, ARITH] => REAL\GT(A, B);
      [SITE, SITE] => REAL\GT(A.VP, B.VP);
      TRUE => ERROR();
    END;

ERROR ← EXPR(S:STRING) BREAK([) S = '' => 'ERROR CALLED'; S (]);

NS\VAL ← EXPR(NP:INT; ANY) VAL(GETREF(NS[NP].OBJECT));

NIL;
```

```
SELECT ←
  EXPR(P:BIGREF, I:INT; BIGREF)
    BEGIN
      DECL M:MODE BYVAL P.A;
      DECL A:MODE;
      DECL V:SITE BYVAL P.B;
      M = MODE =>
        BEGIN
          DECL ATM:ECL\ATOM LIKE FINDHASH(STBL, M.NAME);
          ATM = NIL OR ATM.SBLK = NIL OR ATM.SBLK.CONSTF = NIL => ERROR('SELECTION ERROR');
          DECL X:ANY LIKE ATM.SBLK.CONSTF[I];
          CONST(BIGREF OF MD(X), CONST(SITE OF X));
        END;
      M = SYMBOL =>
        BEGIN
          DECL FIELD:ANY LIKE
            BEGIN
              DECL F:BOOL LIKE TRUE;
              DECL S:SYMBOL LIKE VAL(GETREF(P));
              DECL T:ANY LIKE FINDHASH(STBL, S, F);
              F => T;
              S;
            END[I];
          CONST(BIGREF OF MD(FIELD), CONST(SITE OF FIELD));
        END;
      M = DTPR => [) DECL B:ANY BYVAL GETREF(P)[I]; CONST(BIGREF OF MD(B), CONST(SITE OF B)
) (];
      DECL J:INT LIKE
        BEGIN
          M.CLASS = "STRUCT" => [) A ← M.D[I].TYPE; FINDSTR(M).T[I] (];
          M.CLASS = "ROW" =>
            [) A ← M.D.TYPE; (I - 1) * SIZE\OF(CONST(BIGREF OF A, V)) + [) M.LRFLG => 0; 1 (] (];
          ERROR('SELECTION ERROR');
        END;
      DECL Q:BIGREF LIKE CONST(BIGREF OF A, V);
      Q.B.VP ← V.VP + J;
      A.LRFLG OR M.CLASS # "STRUCT" => Q;
      Q.B.VP ← V.VP + VAL(GETREF(Q));
      Q;
    END;
```

```
EV\MVAL ←
  EXPR(P:BIGREF; MODE)
    BEGIN
      DECL R:REF LIKE GETREF(P);
      MVAL(R) = FORM => MVAL(VAL(R));
      MVAL(R) = MODE => DDB;
      MD(VAL(R)) # BIGREF => ERROR();
      R.A;
    END;

HP\SITE ← EXPR(X:REF; SITE) LIFT(CONST(SITE.UR OF X, 1), SITE);

EV\VAL ←
  EXPR(P:BIGREF; BIGREF)
    BEGIN
      P.A.CLASS # "PTR" => ERROR('TYPE FAULT');
      DECL R:REF LIKE GETREF(P);
      MVAL(R) = BIGREF => VAL(R);
      CONST(BIGREF OF MVAL(VAL(R)), CONST(SITE OF VAL(VAL(R))));
    END;

FIXED\SIZE ← EXPR(M:MODE; INT) [) M.CLASS # "STRUCT" => ERROR(); FINDSTR(M).F (];

FINDSTR ←
  EXPR(M:MODE; STRTAB)
    BEGIN
      DECL FLAG:BOOL;
      DECL E:STRTAB LIKE FINDHASH(HSTRTAB, M, FLAG);
      FLAG => E;
      E.L ← LENGTH(M.D);
      E.F ← [) M.LRFLG => 0; 1 (];
      E.T ← ALLOC(SEQ(INT) SIZE E.L);
      FOR I TO E.L
        REPEAT
          DECL M1:MODE LIKE M.D[I].TYPE;
          E.T[I] ← E.F;
          E.F ← E.F + [) NOT M1.LRFLG => 1; M1.CLASS = "PTR" => 1; 0 ← M1.SZ (];
        END;
      E;
    END;
```

```
SET\INTERNAL\POINTER ←
  EXPR(OBJECT:BIGREF, IX:INT, COMPONENT:SITE)
    BEGIN
      DECL M:MODE LIKE OBJECT.A;
      DECL B:SITE LIKE OBJECT.B;
      DECL X:INT LIKE FINDSTR(M).T[IX];
      VS[B.VP + X] ← ALLOC(INT BYVAL COMPONENT.VP - B.VP);
    END;

COPY\TO\HEAP ←
  EXPR(P:BIGREF; SITE)
    [) DECL Q:SITE; COPY(Q, P.B, SIZE\OF(P)); CONST(SITE OF CONST(BIGREF OF P.A, Q)) (];

STRTAB ← STRTAB :: STRUCT(L:INT, F:INT, T:PTR(SEQ(INT)));

BIGREF ← BIGREF :: STRUCT(A:MODE, B:SITE);

ECL\ATOM ← ECL\ATOM :: STRUCT(TLB:BIGREF, SBLK:PTR(SBLOCK), LINK:PTR("ECL\ATOM"), PNAME
:SYMBOL);

ECL\SUBR ← ECL\SUBR :: STRUCT(BODY:FORM, RETYPE:MODE, PRMD:MODE, FORMALS:SEQ(FDS));

ECL\SYMBOL ← ECL\SYMBOL :: PTR(ECL\ATOM);

PSYS ← PSYS :: PTR(SYSTEM);

EV\LENGTH <-
  EXPR(OBJ:BIGREF BYVAL; INT)
    BEGIN
      DECL M:MODE SHARED OBJ.A;
      DECL V:SITE SHARED OBJ.B;
      M.CLASS = "PTR" -> REPEAT OBJ ← EV\VAL(OBJ); M.CLASS # "PTR" => NOTHING END;
      BEGIN
        M = NONE => 0;
        M.CLASS = "BASIC" => 1;
        M.LRFLG #> GETREF(OBJ).N\COMP;
        M.CLASS = "STRUCT" => LENGTH(M.D);
        M.D.LENGTH;
      END;
    END;
```

```
EV\HASH <-
  EXPR(S:STRING; ECL\SYMBOL)
    BEGIN
      DECL SY:SYMBOL LIKE HASH(S);
      DECL F:BOOL;
      DECL X:ECL\SYMBOL LIKE FINDHASH(STBL, SY, F);
      X # NIL => X;
      X ← ALLOC(ECL\ATOM);
      X.TLB ← CONST(BIGREF OF NONE, NO\SITE);
      X.PNAME ← SY;
      X.LINK ← ATOM\LIST;
      ATOM\LIST ← X;
    END;

EV\TLB <-
  EXPR(F:FORM; BIGREF)
    BEGIN
      DECL P:BIGREF LIKE
        SELECT(CONST(BIGREF OF SYMBOL, CONST(SITE OF F)), FIELD\TO\IX(ATOM, "TLB"));
      DECL TLB:ANY LIKE VAL(GETREF(P));
      MD(TLB) = BIGREF => TLB;
      CONST(BIGREF OF MVAL(TLB), CONST(SITE OF VAL(TLB)));
    END;

REAL\APPLY <- EXPR(F:FORM UNEVAL, G:FORM; ANY) REAL\EVAL(ALLOC(DTPR OF F, G));

GETREF <- EXPR(P:BIGREF; REF) [) DECL Q:SITE SHARED P.B; [) Q.V # NIL => Q.V; VS (][Q.VP] (];

↑;
```

'The following constitute the ECL model proper. The objective is
that it run in any given ECL implementation.

';

```
ECL\EVAL ←
  EXPR(F:FORM)
    BEGIN
      DECL A:MODE SHARED OBJECT.A;
      DECL B:SITE SHARED OBJECT.B;
      F = NIL => NO\RESULT();
      DECL M:MODE LIKE MD(VAL(F));
      M = ATOM =>
        [) DECL I:INT LIKE FIND\NAME(NP, F); I # 0 => OBJECT ← NS[I].OBJECT; OBJECT ← EV\TLB(F)
 (];
      M = DTPR =>
        BEGIN
          MD(VAL(F.CAR)) # ATOM => APPLY(F);
          DECL S:SYMBOL LIKE F.CAR;
          S = "SPECIAL" => [) A ← VAL(F.CDR).A; B ← VAL(F.CDR).B (];
          OBJECT ← EV\TLB(F.CAR);
          A = PSYS => REF\OF(B)(F.CDR);
          APPLY(F);
        END;
      M = REAL OR M = INT => RETURN\RESULT(M, CONST(SITE OF VAL(F)));
      M = REF => RETURN\RESULT(MD(VAL(VAL(F))), CONST(SITE OF VAL(VAL(F))));
      M = DDB => RETURN\PTR(F, MODE);
    END;

RETURN\PTR ←
  EXPR(R:FORM, M:MODE) [) A ← M; B ← VS\SITE(VP + 1); COPY(B, CONST(SITE OF R), 1); VP ← VP +
 1 (];

FIND\NAME ←
  EXPR(NS\INDEX:INT, NAME:SYMBOL; INT)
    [) DECL IX:INT; FOR I FROM NS\INDEX BY - 1 TO 1 REPEAT NS[I].NAME = NAME => IX ← I END; IX
 (];
NO\RESULT ← EXPR() OBJECT ← CONST(BIGREF);

EVAL\DECL ←
  EXPR(F:FORM BYVAL)
    BEGIN
      DECL ID\LIST:FORM BYVAL F.CAR;
      EVAL\CONST(F.CDR);
      INSTALL();
      CS[CP].CVP ← VP;
      ID\LIST = NIL => NOTHING;
      REPEAT
        NS[NP].NAME ← ID\LIST.CAR;
        ID\LIST.CDR = NIL => NOTHING;
        ID\LIST ← ID\LIST.CDR;
        'CAREFUL HERE ... MUST COPY IF A PURE VALUE';
        INSTALL();
        CS[CP].CVP ← VP;
      END;
    END;

INSTALL ← EXPR() NS[NP ← NP + 1] ← CONST(NS\FRAME OF NIL, OBJECT);

EVAL\DOUBLE\COLON ← EXPR(F:FORM) APPLY(F);

EVAL\ALLOC ← EXPR(F:FORM) [) EVAL\CONST(F); B ← COPY\TO\HEAP(OBJECT); A ← REF (];
```

```
EVAL\CONST ←
  EXPR(F:FORM)
    BEGIN
      ECL\EVAL(F.CAR);
      A # MODE => ERROR();
      DECL M:MODE BYVAL MODE\OF(B);
      VP ← CS[CP].CVP;
      DECL BC:SYMBOL BYVAL [) F.CDR = NIL => NIL; F.CDR.CAR (];
      M = A AND M.GENFN = ANY.GENFN => [) BC = "BYVAL" => PURIFY() (];
      DECL USER\GEN\FN:REF BYVAL [) M.UFN # NIL => M.UFN[5]; NIL (];
      BC = "SHARED" OR BC = "LIKE" OR BC = "BYVAL" =>
        BEGIN
          ECL\EVAL(F.CDR.CDR.CAR);
          DECL LOCATION:SITE BYVAL B;
          USER\GEN\FN # NIL ->
            BEGIN
              DECL F:FORM BYVAL
                ALLOC(DTPR OF
                      VAL(USER\GEN\FN),
                      ALLOC(DTPR OF
                            BC,
                            ALLOC(DTPR OF
                                  ALLOC(DTPR OF
                                        "SPECIAL",
                                        ALLOC(ANY BYVAL ALLOC(BIGREF BYVAL CONST(BIGREF OF A, B)))
),
                                  ALLOC(DTPR OF M, NIL))));
              APPLY(F);
              BC = "SHARED" AND B # LOCATION => ERROR();
            END;
          COVERS(M, A) => [) BC = "BYVAL" => PURIFY() (];
          DECL USER\CONV\FN:REF BYVAL [) A.UFN = NIL => NIL; A.UFN[1] (];
          BEGIN
            USER\CONV\FN # NIL =>
              BEGIN
                DECL F:FORM BYVAL
                  ALLOC(DTPR OF
                        VAL(USER\CONV\FN),
                        ALLOC(DTPR OF
                              ALLOC(DTPR OF
                                    "SPECIAL",
                                    ALLOC(ANY BYVAL ALLOC(BIGREF BYVAL CONST(BIGREF OF A, B)))),
                              ALLOC(DTPR OF M, NIL)));
                APPLY(F);
              END;
            SYS\CONV(M);
          END;
          NOT COVERS(M, A) => ERROR();
          BC = "BYVAL" => PURIFY();
        END;
        DECL OBJ:BIGREF BYVAL CONST(BIGREF OF M, VS\SITE(CS[CP].CVP + 1));
        DECL M1:MODE;
        DECL N:INT;
        BC = NIL OR BC = "SIZE" =>
          BEGIN
            M.CLASS = "PTR" => RETURN\RESULT(M, CONST(SITE OF NIL));
            M.CLASS = "BASIC" => DEFAULT\VALUE(M);
            DECL V:SEQ(INT) BYVAL
              EVALUATE\DOPE\VECTOR([) M.LRFLG => 0; 0 ← M.SZ (], [) BC = NIL => NIL; F.CDR.CDR (])
;
            M.CLASS = "ROW" =>
              BEGIN
                M1 ← M.D.TYPE;
                DECL IS\SEQ:BOOL BYVAL M.D.LENGTH = - 1;
                CS[CP].CVP ← VP ← VP + [) IS\SEQ => 1; 0 (];
                N ← [) IS\SEQ AND LENGTH(V) # 0 => V[1]; NOT IS\SEQ => M.D.LENGTH; 0 (];
                N LT 0 => ERROR();
                BEGIN
                  NOT M1.LRFLG =>
                    BEGIN
                      N GT 0 ->
                        BEGIN
                          EV\CONST(M1, "SIZE", SUB\VECTOR(V, [) IS\SEQ => 2; 1 (], LENGTH(V)));
                          SLIDE\VS(CS[CP].CVP);
                          VP ← CS[CP].CVP;
                        END;
                      DECL M1SZ:INT LIKE SIZE\OF(OBJECT);
                      TO N - 1 REPEAT COPY(VS\SITE(VP + 1), B, M1SZ); VP ← VP + M1SZ END;
                      CS[CP].CVP ← VP;
                    END;
                  N GT 0 -> EV\CONST(M1);
                  DECL M1SZ:INT BYVAL SIZE\OF(OBJECT);
                  VP ← M1SZ + CS[CP].CVP;
                  FOR J FROM 2 TO N REPEAT COPY(SELECT(OBJ, J).B, B, M1SZ); VP ← VP + M1SZ END;
                END;
                IS\SEQ ->
                  BEGIN
                    DECL SHDR:SEQ\HEADER;
                    SHDR.N\COMP ← N;
                    SHDR.N\WORDS ← [) M1.LRFLG => 0 ← M1.SZ; VS[OBJ.B.VP + 1].N\WORDS (] * N + 1
;
                    COPY(OBJ.B, CONST(SITE OF SHDR), 1);
                  END;
                OBJECT ← OBJ;
                CS[CP].CVP ← VP;
              END;
```

```
  M.CLASS = "STRUCT" =>
    BEGIN
      CS[CP].CVP ← VP ← VP + FIXED\SIZE(M);
      DECL I:INT;
      FOR J TO LENGTH(VAL(M.D))
        REPEAT
          EV\CONST(M.D[J].TYPE,
                   "SIZE",
                   SUB\VECTOR(V,
                              I + 1,
                              I + [) M.D[J].TYPE.LRFLG => 0; 0 ← M.D[J].TYPE.SZ (]));
          SLIDE\VS(CS[CP].CVP);
          BEGIN
            M.D[J].TYPE.LRFLG => COPY(SELECT(OBJ, J).B, B, SIZE\OF(OBJECT));
            SET\INTERNAL\POINTER(OBJ, J, B);
            I ← I + (0 ← M.D[J].TYPE.SZ);
          END;
        END;
      NOT M.LRFLG ->
        BEGIN
          DECL SHDR:SEQ\HEADER;
          SHDR.N\COMP ← LENGTH(M.D);
          DECL N:INT LIKE SHDR.N\WORDS;
          N ← FIXED\SIZE(M);
          TO V[1] REPEAT N ← N + VS[OBJ.B.VP + N].N\WORDS END;
          COPY(OBJ.B, CONST(SITE OF SHDR), 1);
        END;
      OBJECT ← OBJ;
      SLIDE\VS(OBJ.B.VP - 1);
      CS[CP].CVP ← OBJ.B.VP - 1;
    END;
END;
,
"       BINDCLASS "OF" OR USER BINDCLASS
";
      DECL V:FORM BYVAL F.CDR.CDR;
      BEGIN
        DECL CVP:INT BYVAL VP;
        M.CLASS = "ROW" AND M.D.LENGTH = - 1 =>
          BEGIN
            N ← LIST\LENGTH(V);
            CS[CP].CVP ← VP ← VP + 1;
            EVAL\COMPONENTS(N);
            DECL SHDR:SEQ\HEADER LIKE CONST(SEQ\HEADER OF N, VP - CVP);
            COPY(OBJ.B, CONST(SITE OF SHDR), 1);
          END;
        M.CLASS = "ROW" =>
          BEGIN
            DECL M1:MODE LIKE M.D.TYPE;
            DECL LL:INT BYVAL LIST\LENGTH(V);
            N ← M.D.LENGTH;
            EVAL\COMPONENTS([) LL LT N => LL; N (]);
            DECL D:SEQ(INT) BYVAL DOPE\VECTOR\OF(OBJECT);
            TO N - LL
              REPEAT
                EV\CONST(M1, "SIZE", D);
                DOPE\VECTOR\OF(OBJECT) # D => ERROR();
                SLIDE\VS(CS[CP].CVP);
                CS[CP].CVP ← VP;
              END;
          END;
```

```
    M.CLASS = "STRUCT" =>
      BEGIN
        DECL LL:INT BYVAL LIST\LENGTH(V);
        N ← LENGTH(VAL(M.D));
        CS[CP].CVP ← VP ← VP + FIXED\SIZE(M);
        FOR J TO [) LL LT N => LL; N (]
          REPEAT
            DECL CVP:INT BYVAL VP;
            EV\CONST(M.D[J].TYPE, "BYVAL", V.CAR);
            V ← V.CDR;
            M.D[J].TYPE.LRFLG =>
              [) COPY(SELECT(OBJ, J).B, B, SIZE\OF(OBJECT)); VP ← CS[CP].CVP (];
            SLIDE\VS(CVP);
            SET\INTERNAL\POINTER(OBJ, J, B);
          END;
        N GT LL ->
          FOR J FROM LL + 1 TO N
            REPEAT
              EV\CONST(M.D[J].TYPE, NIL, NIL);
              M.D[J].TYPE.LRFLG ->
                [) COPY(SELECT(OBJ, J).B, B, SIZE\OF(OBJECT)); VP ← CS[CP].CVP (];
              NOT M.D[J].TYPE.LRFLG ->
                [) SLIDE\VS(CS[CP].CVP); SET\INTERNAL\POINTER(OBJ, J, B); CS[CP].CVP ← VP (];
            END;
        NOT M.LRFLG =>
          BEGIN
            DECL SHDR:SEQ\HEADER;
            SHDR.N\COMP ← LENGTH(M.D);
            DECL N:INT LIKE SHDR.N\WORDS;
            N ← FIXED\SIZE(M);
            REPEAT OBJ.B.VP + N GE VP => NOTHING; N ← N + VS[OBJ.B.VP + N].N\WORDS END;
            COPY(OBJ.B, CONST(SITE OF SHDR), 1);
          END;
      END;
  END;
  OBJECT ← OBJ;
END;
```

```
EVAL\COMPONENTS ←
  EXPR(N:INT)
    BEGIN
      ECL\EVAL(V.CAR);
      SLIDE\VS(CS[CP].CVP);
      PURIFY();
      CS[CP].CVP ← VP;
      DECL D:SEQ(INT) BYVAL DOPE\VECTOR\OF(OBJECT);
      TO N - 1
        REPEAT
          V ← V.CDR;
          ECL\EVAL(V.CAR);
          SLIDE\VS(CS[CP].CVP);
          PURIFY();
          CS[CP].CVP ← VP;
          D # DOPE\VECTOR\OF(OBJECT) => ERROR();
        END;
    END;

EVAL\PTR ←
  EXPR(F:FORM)
    BEGIN
      DECL M:MODE LIKE REAL\APPLY(PTR, F);
      VS[VP ← VP + 1] ← ALLOC(ANY BYVAL M);
      RETURN\RESULT(MODE, VS\SITE(VP));
    END;

EVAL\SEQ ←
  EXPR(F:FORM)
    BEGIN
      DECL M:MODE LIKE REAL\APPLY(SEQ, F);
      VS[VP ← VP + 1] ← ALLOC(ANY BYVAL M);
      RETURN\RESULT(MODE, VS\SITE(VP));
    END;
```

```
EVAL\VECTOR ←
  EXPR(F:FORM)
    BEGIN
      DECL M:MODE LIKE REAL\APPLY(VECTOR, F);
      VS[VP ← VP + 1] ← ALLOC(ANY BYVAL M);
      RETURN\RESULT(MODE, VS\SITE(VP));
    END;

NIL;

EVAL\STRUCT ←
  EXPR(F:FORM)
    BEGIN
      DECL M:MODE LIKE REAL\APPLY("STRUCT", F);
      VS[VP ← VP + 1] ← ALLOC(ANY BYVAL M);
      RETURN\RESULT(MODE, VS\SITE(VP));
    END;

EVAL\ONEOF ←
  EXPR(F:FORM)
    BEGIN
      DECL M:MODE LIKE REAL\APPLY(ONEOF, F);
      VS[VP ← VP + 1] ← ALLOC(ANY BYVAL M);
      RETURN\RESULT(MODE, VS\SITE(VP));
    END;

EVAL\PROC ←
  EXPR(F:FORM)
    BEGIN
      DECL M:MODE LIKE REAL\APPLY("PROC", F);
      VS[VP ← VP + 1] ← ALLOC(ANY BYVAL M);
      RETURN\RESULT(MODE, VS\SITE(VP));
    END;
```

```
LIST\LENGTH ←
  EXPR(L:FORM BYVAL; INT)
    [) L = NIL => 0; DECL R:INT BYVAL 1; REPEAT L ← L.CDR; L = NIL => R; R ← R + 1 END (];

PURIFY ← EXPR() [) IS\PURE() => NOTHING; RETURN\RESULT(A, B); SLIDE\VS(CS[CP].CVP) (];

EV\CONST ←
  EXPR(M:MODE, BC:SYMBOL, V:ANY)
    BEGIN
      DECL F:FORM BYVAL ALLOC(DTPR OF M);
      BC # NIL ->
        BEGIN
          F.CDR ←
            ALLOC(DTPR OF
                  BC,
                  BEGIN
                    BC = "SIZE" =>
                      BEGIN
                        DECL S:SEQ(INT) LIKE V;
                        DECL N:INT LIKE LENGTH(S);
                        DECL F:FORM LIKE ALLOC(DTPR);
                        N = 0 => F;
                        DECL G:FORM BYVAL F;
                        F.CDR;
                      END;
                    BC = "OF" => V;
                    ALLOC(DTPR OF V);
                  END);
        END;
      EVAL\CONST(F);
    END;
```

```
APPLY ←
  EXPR(F:FORM)
    BEGIN
      ECL\EVAL(F.CAR);
      DECL P:REF BYVAL REF\OF(B);
      DECL TYPE:SYMBOL BYVAL
        [) P = NIL => ERROR(); MD(VAL(P)) = DTPR AND P.CAR = "EXPR" => "EXPR"; MD(VAL(P)).NAM
E (];
      CS[CP ← CP + 1] ←
        CONST(CS\FRAME OF VP, VP, NP, TYPE, P, [) TYPE = "EXPR" => P.CDR.CDR.CDR.CAR; NIL (]);
      BIND\FORMALS(F.CDR, P);
      EV\CONST([) TYPE = "EXPR" => P.CDR.CDR.CAR; P.RETYPE (],
               "LIKE",
               [) TYPE = "EXPR" => CS[CP].LC; EXECUTE(P) (]);
      INSTALL();
      OBJECT ← NS[NP].OBJECT;
      SLIDE\VS(CS[CP].VP);
      NP ← CS[CP].NP;
      CP ← CP - 1;
    END;

SLIDE\VS ←
  EXPR(TOP\VP:INT)
    BEGIN
      IS\VS\SITE(B) AND B GT VS\SITE(TOP\VP) =>
        BEGIN
          DECL CVP:INT BYVAL VP;
          DECL N:INT BYVAL SIZE\OF(OBJECT);
          DECL NEW\PTR:SITE BYVAL VS\SITE(TOP\VP + 1);
          COPY(NEW\PTR, B, N);
          B ← NEW\PTR;
          VP ← TOP\VP + N;
          FOR I FROM VP + 1 TO CVP REPEAT VS[I] ← NIL END;
        END;
      VP ← CS[CP].VP;
    END;
```

```
BIND\FORMALS ←
  EXPR(ARG:FORM BYVAL, P:REF)
    BEGIN
      DECL M:MODE LIKE MD(VAL(P));
      M = DTPR =>
        BEGIN
          DECL CNP:INT BYVAL NP;
          DECL FML:FORM BYVAL P.CDR.CAR;
          REPEAT
            FML = NIL => NOTHING;
            ECL\EVAL(FML.CAR.CDR.CAR);
            A # MODE => ERROR();
            BFORML(MODE\OF(B), FML.CAR.CDR.CDR.CAR, ARG);
            FML ← FML.CDR;
            ARG # NIL -> ARG ← ARG.CDR;
          END;
          FML ← P.CDR.CAR;
          NP ← CNP;
          REPEAT FML = NIL => NOTHING; NS[NP ← NP + 1].NAME ← FML.CAR.CAR; FML ← FML.CDR E
ND;
        END;
      DECL FMLS:ANY LIKE P.FORMALS;
      FOR I TO LENGTH(FMLS)
        REPEAT
          ECL\EVAL(FMLS[I].TYPE);
          BFORML(MODE\OF(B), FMLS[I].SYM, ARG);
          ARG # NIL -> ARG ← ARG.CDR;
        END;
    END;

BFORML ←
  EXPR(M:MODE, BC:SYMBOL, ARG:FORM)
    BEGIN
      DECL ARG\VAL:FORM BYVAL [) ARG # NIL => ARG.CAR; NIL (];
      OR(BC = "LIKE", BC = "SHARED", BC = "BYVAL") =>
        [) EV\CONST(M, BC, ARG\VAL); INSTALL(); CS[CP].CVP ← VP (];
      M # FORM => ERROR();
      BC = "LISTED" -> [) ARG\VAL ← ARG; ARG ← NIL (];
      NS[NP ← NP + 1] ← CONST(NS\FRAME OF NIL, CONST(BIGREF OF FORM, CONST(SITE OF ARG\V
AL)));
    END;
```

```
RETURN\RESULT ←
  EXPR(M:MODE, PTR:SITE BYVAL)
    BEGIN
      A ← M;
      B ← VS\SITE(VP + 1);
      DECL N:INT BYVAL SIZE\OF(CONST(BIGREF OF M, PTR));
      COPY(B, PTR, N);
      VP ← VP + N;
    END;
```

```
EVAL\FOR ←
  EXPR(F:FORM BYVAL)
    BEGIN
      CS[CP ← CP + 1] ← CONST(CS\FRAME OF VP, VP, NP, "FOR", F, F);
      DECL TEST:PROC(INT, INT; BOOL) BYVAL EXPR(; BOOL) FALSE;
      DECL FLAG, EXIT:BOOL;
      DECL LC:FORM SHARED CS[CP].LC;
      DECL OPTIONS:BOOL LIKE LC.CAR # "REPEAT";
      DECL CVP:INT SHARED CS[CP].CVP;
      DECL ID, NXT:SYMBOL;
      DECL STEP, FV:INT;
      DECL NEXT:ROUTINE LIKE EXPR() NXT ← (LC ← LC.CDR).CAR;
      DECL EVAL\INT:ROUTINE LIKE EXPR(; INT) [) ECL\EVAL((LC ← LC.CDR).CAR); NEXT(); INT\OF(B) (
];
      NXT ← LC.CAR;
      STEP ← 1;
      NXT = "FOR" -> [) NEXT(); ID ← NXT; NEXT() (];
      NXT = "FROM" -> NXT ← "BYVAL";
      EV\CONST(INT,
               BEGIN
                 OR(NXT = "BYVAL", NXT = "LIKE", NXT = "SHARED") => [) LC ← LC.CDR; NXT (];
                 FLAG ← TRUE;
                 NIL;
               END,
               LC.CAR);
      INSTALL();
      NS[NP].NAME ← ID;
      SLIDE\VS(CVP);
      CVP ← VP;
      OPTIONS -> NEXT();
      CS[CP].NP ← NP;
      DECL CNP:INT BYVAL CS[CP].NP;
      DECL V:INT SHARED NS\VAL(NP);
      DECL VV:INT BYVAL V;
      FLAG -> VV ← 1;
      NXT = "BY" -> STEP ← EVAL\INT();
      NXT = "TO" -> [) FV ← EVAL\INT(); STEP LT 0 => TEST ← LT; TEST ← GT (];
      DECL LOOP:FORM BYVAL LC;
      NO\RESULT();
      REPEAT
        EXIT OR TEST(VV, FV) => NOTHING;
        V ← VV;
        LC ← LOOP;
        REPEAT (LC ← LC.CDR) = NIL OR EXIT => NOTHING; VP ← CVP; ECL\EVAL(LC.CAR); NP ← CNP
END;
        OPTIONS -> VV ← V + STEP;
      END;
      SLIDE\VS(CVP);
      NP ← CS[CP].NP;
      CP ← CP - 1;
    END;
```

```
EVAL\BLOCK ←
  EXPR(F:FORM)
    BEGIN
      CS[CP ← CP + 1] ← CONST(CS\FRAME OF VP, VP, NP, "BLOCK", F, F);
      NO\RESULT();
      DECL EXIT:BOOL;
      DECL CVP:INT SHARED CS[CP].CVP;
      DECL LC:FORM SHARED CS[CP].LC;
      REPEAT EXIT OR LC = NIL => NOTHING; VP ← CVP; ECL\EVAL(LC.CAR); LC ← LC.CDR END;
      SLIDE\VS(CS[CP].VP);
      NP ← CS[CP].NP;
      CP ← CP - 1;
    END;

EVAL\CONDFF ← EXPR(F:FORM) EVAL\COND(F, FALSE, FALSE);

EVAL\CONDFT ← EXPR(F:FORM) EVAL\COND(F, FALSE, TRUE);

EVAL\CONDTF ← EXPR(F:FORM) EVAL\COND(F, TRUE, FALSE);

EVAL\CONDTT ← EXPR(F:FORM) EVAL\COND(F, TRUE, TRUE);

EVAL\COND ←
  EXPR(F:FORM BYVAL, TEST:BOOL, CLAUSE:BOOL)
    BEGIN
      CLAUSE AND NOT (CS[CP].TYPE = "BLOCK" OR CS[CP].TYPE = "FOR") => ERROR();
      ECL\EVAL(F.CAR);
      A # BOOL => ERROR();
      BOOL\OF(B) # TEST => NO\RESULT();
      ECL\EVAL(F.CDR.CAR);
      CLAUSE -> EXIT ← TRUE;
    END;
```

```
EVAL\ASSIGN ←
  EXPR(F:FORM)
    BEGIN
      ECL\EVAL(F.CAR);
      DECL USER\ASSIGN\FN:REF BYVAL [) A.UFN # NIL => A.UFN[2]; NIL (];
      USER\ASSIGN\FN # NIL =>
        BEGIN
          DECL P:FORM BYVAL
            ALLOC(DTPR OF
                  VAL(USER\ASSIGN\FN),
                  ALLOC(DTPR OF
                        ALLOC(DTPR OF
                              "SPECIAL", ALLOC(ANY BYVAL ALLOC(BIGREF BYVAL CONST(BIGREF OF A, B
)))),
                        ALLOC(DTPR BYVAL CONST(DTPR BYVAL VAL(F.CDR)))));
          APPLY(P);
        END;
      DECL DEST:BIGREF BYVAL OBJECT;
      ECL\EVAL(F.CDR.CAR);
      SYS\CONV(DEST.A);
      DECL DEST\N:INT BYVAL SIZE\OF(DEST);
      DECL N:INT BYVAL SIZE\OF(OBJECT);
      N # DEST\N => ERROR('ASSIGN ERROR - DIFFERENT DOPE VECTORS');
      COPY(DEST.B, B, N);
      OBJECT ← DEST;
    END;

EVAL\LIFT ←
  EXPR(F:FORM)
    BEGIN
      ECL\EVAL(F.CAR);
      A # MODE => ERROR();
      DECL M, M1:MODE BYVAL MODE\OF(B);
      VP ← CS[CP].CVP;
      ECL\EVAL(F.CDR.CAR);
      REPEAT M1.UR = NIL => ERROR(); M1.UR = A => A ← M; M1 ← M1.UR END;
    END;
```

```
EVAL\LOWER ← EXPR(F:FORM) [) ECL\EVAL(F.CAR); A.UR = NIL => ERROR(); A ← A.UR (];

SEQ\HEADER ← SEQ\HEADER :: STRUCT(N\COMP:INT, N\WORDS:INT);

NSPFN ←
  EXPR(X:NS\FRAME)
    BEGIN
      X.OBJECT.A = NIL => NOTHING;
      PRINT(X.NAME);
      PRINT(%:);
      PRINT(X.OBJECT.A);
      PRINT(' = ');
      DECL OBJ:BIGREF BYVAL X.OBJECT;
      FOR I TO SIZE\OF(X.OBJECT)
        REPEAT PRINT('                    '); PRINT(VAL(GETREF(OBJ))); OBJ.B.VP ← OBJ.B.VP + 1; PRIN
T('
') END;
      PRINT('
');
    END;

VSPFN ← EXPR(X:REF) [) PRINT(VAL(X)); PRINT('
') (];

CSPFN ←
  EXPR(X:CS\FRAME)
    BEGIN
      PRINT(X.VP);
      PRINT(%         );
      PRINT(X.CVP);
      PRINT(%         );
      PRINT(X.NP);
      PRINT(%         );
      PRINT(X.TYPE);
      PRINT('
');
    END;
```

```
VS\FRAME ← QL("VS\FRAME", NIL, NIL, NIL, VSPFN) :: REF;

NS\FRAME ← QL("NS\FRAME", NIL, NIL, NIL, NSPFN) :: STRUCT(NAME:SYMBOL, OBJECT:BIGREF);

VALUE\STACK ← VALUE\STACK :: SEQ(VS\FRAME);

NAME\STACK ← NAME\STACK :: SEQ(NS\FRAME);

CS\FRAME ←
  QL("CS\FRAME", NIL, NIL, NIL, CSPFN) ←
    CS\FRAME :: STRUCT(VP:INT, CVP:INT, NP:INT, TYPE:SYMBOL, FM:REF, LC:FORM);

CONTROL\STACK ← CONTROL\STACK :: SEQ(CS\FRAME);

EVAL\SEL ← EXPR(F:FORM) EV\SEL(F);

EVAL\SELQ ← EXPR(F:FORM) EV\SEL(F, TRUE);

EV\SEL ←
  EXPR(F:FORM, FLAG:BOOL)
    BEGIN
      DECL UFN:REF;
      ECL\EVAL(F.CAR);
      REPEAT
        A.UFN # NIL AND (UFN ← A.UFN[3]) # NIL OR A.CLASS # "PTR" => NOTHING;
        OBJECT ← EV\VAL(OBJECT);
      END;
      DECL OBJ:BIGREF BYVAL OBJECT;
      BEGIN
        FLAG => [) VS[VP ← VP + 1] ← ALLOC(FORM BYVAL F.CDR.CAR); A ← SYMBOL; B ← VS\SITE(V
P) (];
        ECL\EVAL(F.CDR.CAR);
      END;
      DECL IX:INT LIKE [) A = INT => INT\OF(B); FIELD\TO\IX(OBJ.A, SYMBOL\OF(B)) (];
      OBJECT ← SELECT(OBJ, IX);
    END;
```

```
FIELD\TO\IX ←
  EXPR(M:MODE, S:SYMBOL; INT)
    BEGIN
      DECL I:INT LIKE 1;
      DECL D:ANY LIKE VAL(M.D);
      DECL L:INT LIKE LENGTH(D);
      SHARED I TO L REPEAT D[I].SYM = S => NOTHING END;
      I LE L => I;
      ERROR();
    END;

DEFAULT\VALUE ←
  EXPR(M:MODE)  .
    BEGIN
      M.CLASS = "BASIC" =>
        RETURN\RESULT(M,
                      CONST(SITE OF
                            BEGIN
                              M = INT => 0;
                              M = REAL => 0.0;
                              M = BOOL => FALSE;
                              M = CHAR => INT\CHAR(0);
                              M = NONE => NOTHING;
                            END));
      ERROR();
    END;
```

```
EVALUATE\DOPE\VECTOR ←
  EXPR(L:INT, F:FORM BYVAL; SEQ(INT))
    BEGIN
      DECL DV:SEQ(INT) LIKE CONST(SEQ(INT) SIZE L);
      L # LIST\LENGTH(F) => ERROR();
      FOR I TO L REPEAT ECL\EVAL(F.CAR); DV[I] ← INT\OF(B); F ← F.CDR; VP ← VP - 1 END;
      DV;
    END;

SUB\VECTOR ←
  EXPR(V:SEQ(INT), I:INT, L:INT; FORM)
    [) DECL VV:FORM; FOR J FROM L BY - 1 TO I REPEAT VV ← ALLOC(DTPR OF MKFM(V[J]), VV) EN
D; VV (];

↑;
```

```
'
"      THE MODE COMPILER
"
i"             All mode compiler routines are entered into the
"              symbol table (STBL) and are hence interpreted
"              by ECL\EVAL, even though they are part of the system
"              proper.
"
';

EVFMS <-
  EXPR(F:FORM; FORM)
    BEGIN
      DECL T:ANY BYVAL EVAL(F);
      MD(T).CLASS # "PTR" -> ERROR('TYPE FAULT');
      'IF ALREADY A MODE, THEN RETURN IT.';
      MD(VAL(T)) = DDB => T;
      'IF A SYMBOL, THEN TRY TO OBTAIN A MODE';
      MD(VAL(T)) = ATOM => SYTOM(T);
      ERROR('TYPE FAULT');
    END;

SYTOM <-
  EXPR(T:SYMBOL; FORM)
    BEGIN
      'IF NO ASSOCIATED MODE, THEN RETURN SYMBOL';
      T.SBLK = NIL OR MD(VAL(T.SBLK.CONSTF)) # DDB => T;
      'OTHERWISE RETURN ASSOCIATED MODE';
      T.SBLK.CONSTF;
    END;
```

```
GETDDB <-
  EXPR(N:SYMBOL, D:REF, CLS:SYMBOL; MODE)
    BEGIN
      DECL NEWMD:MODE;
      'IF MODE ASSOCIATED WITH NAME, RETURN IT';
      N.SBLK # NIL AND MD(VAL(N.SBLK.CONSTF)) = DDB => N.SBLK.CONSTF;
      'OTHERWISE GET A NEW DDB AND SET NAME, CLASS, D FIELDS';
      NEWMD <- ALLOC(DDB);
      NEWMD.D <- D;
      NEWMD.CLASS <- CLS;
      NEWMD.NAME <- N;
      'MAKE SURE NAME IS NOW ASSOCIATED WITH THIS MODE';
      N.SBLK = NIL -> N.SBLK <- ALLOC(SBLOCK);
      N.SBLK.CONSTF <- NEWMD;
    END;

DEFDDB <-
  EXPR(NEWM:MODE; MODE)
    BEGIN
      DECL FIN, BND:BOOL BYVAL TRUE;
      'CALL CMPDDB TO ATTEMPT COMPLETION OF THE MODE.
  IF SUCCESSFUL, EXIT TO THE MODE COMPILER';
      CMPDDB(NEWM) => MODCOMPILER(NEWM);
      'OTHERWISE THE NEW MODE%'S GENERATION FUNCTION IS SET TO A COMPLETE ME FUNCTION
 WHICH WILL ATTEMPT TO';
      'COMPLETE THE MODE WHEN AN OBJECT IS GENERATED';
      NEWM.GENFN <- ADDR(CMPFN);
      NEWM;
    END;

CONCAT <-
  EXPR(P:PTR(STRING), Q:STRING; PTR(STRING))
    BEGIN
      DECL LP:INT LIKE LENGTH(P);
      DECL LQ:INT LIKE LENGTH(Q);
      DECL R:PTR(STRING) LIKE ALLOC(STRING SIZE LP + LQ);
      DECL S:STRING SHARED VAL(R);
      DECL X:STRING SHARED VAL(P);
      FOR I TO LP REPEAT S[I] ← X[I] END;
      FOR I FROM 1 TO LQ REPEAT S[LP + I] ← Q[I] END;
      R;
    END;
```

```
PTYPE <-
  EXPR(F:FORM; STRING)
    BEGIN
      DECL PS:PTR(STRING) BYVAL ALLOC(STRING BYVAL '"');
      DECL S:SYMBOL;
      DECL MDV:MODE BYVAL MD(VAL(F));
      'GET SYMBOL WHOSE PRINT NAME WILL BE USED BELOW';
      MDV = ATOM -> S <- F;
      MDV = DDB -> S <- F.NAME;
      'IF NOT SHORTNAME MODE, JUST USE PRINTNAME';
      MDV # ATOM AND F.UR = NIL => BASIC\STR(S);
      'OTHERWISE USE QUOTED PRINTNAME';
      PS <- CONCAT(PS, BASIC\STR(S));
      VAL(CONCAT(PS, %"));
    END;

EVSEQ <-
  EXPR(F:FORM; MODE)
    BEGIN
      DECL NAME:SYMBOL;
      DECL DEF:REF BYVAL ALLOC(STRTE OF EVFMS(F), - 1);
      DECL RESULT:MODE;
      BEGIN
        DECL PS:PTR(STRING) BYVAL ALLOC(STRING BYVAL 'SEQ(');
        PS <- CONCAT(PS, PTYPE(DEF.TYPE));
        PS <- CONCAT(PS, %));
        NAME <- HASH(VAL(PS));
      END;
      RESULT <- GETDDB(NAME, DEF, "ROW");
      RESULT.FINFLG => RESULT;
      DEFDDB(RESULT);
    END;
```

```
EVVECT <-
  EXPR(LEN:INT, F:FORM; MODE)
    BEGIN
      DECL NAME:SYMBOL;
      DECL DEF:REF BYVAL ALLOC(STRTE OF EVFMS(F), LEN);
      DECL RESULT:MODE;
      BEGIN
        DECL PS:PTR(STRING) BYVAL ALLOC(STRING BYVAL 'VECTOR(');
        PS <- CONCAT(PS, BASIC\STR(LEN));
        PS <- CONCAT(PS, %,);
        PS <- CONCAT(PS, PTYPE(DEF.TYPE));
        PS <- CONCAT(PS, %));
        NAME <- HASH(VAL(PS));
      END;
      RESULT <- GETDDB(NAME, DEF, "ROW");
      RESULT.FINFLG => RESULT;
      DEFDDB(RESULT);
    END;
```

```
EVPTR <-
  EXPR(F:FORM; MODE)
    BEGIN
      DECL NAME:SYMBOL;
      DECL LEN:INT BYVAL LISTLENGTH(F);
      DECL DEF:REF BYVAL ALLOC(SEQ(FORM) SIZE LEN);
      DECL RESULT:MODE;
      LEN = 1 -> DEF[1] <- EVFMS(F);
      LEN # 1 -> [) FOR I TO LEN REPEAT DEF[I] <- EVFMS(F.CAR); F <- F.CDR END (];
      BEGIN
        DECL PS:PTR(STRING) BYVAL ALLOC(STRING BYVAL 'PTR(');
        FOR I TO LEN REPEAT PS <- CONCAT(PS, PTYPE(DEF[I])); PS <- CONCAT(PS, %,) END;
        PS[LENGTH(PS)] <- %);
        NAME <- HASH(VAL(PS));
      END;
      RESULT <- GETDDB(NAME, DEF, "PTR");
      RESULT.FINFLG => RESULT;
      DEFDDB(RESULT);


    END;

EVRNY <-
  EXPR(F:FORM; MODE)
    BEGIN
      DECL NAME:SYMBOL;
      DECL LEN:INT BYVAL LISTLENGTH(F);
      DECL DEF:REF BYVAL ALLOC(SEQ(FORM) SIZE LEN);
      DECL RESULT:MODE;
      LEN = 1 -> DEF[1] <- EVFMS(F);
      LEN # 1 -> [) FOR I TO LEN REPEAT DEF[I] <- EVFMS(F.CAR); F <- F.CDR END (];
      BEGIN
        DECL PS:PTR(STRING) BYVAL ALLOC(STRING BYVAL 'ONEOF(');
        FOR I TO LEN REPEAT PS <- CONCAT(PS, PTYPE(DEF[I])); PS <- CONCAT(PS, %,) END;
        PS[LENGTH(PS)] <- %);
        NAME <- HASH(VAL(PS));
      END;
      RESULT <- GETDDB(NAME, DEF, "GENERIC");
      RESULT.FINFLG => RESULT;
      DEFDDB(RESULT);
    END;
```

```
EVPROC <-
  EXPR(F:FORM; MODE)
    BEGIN
      DECL FRMLS:FORM BYVAL F.CAR;
      DECL RTYPE:FORM BYVAL EVFMS(F.CDR.CAR);
      DECL LEN:INT BYVAL LISTLENGTH(FRMLS);
      DECL PDEF:REF BYVAL ALLOC(PDESC SIZE LEN);
      DECL RESULT:MODE;
      DECL NAME:SYMBOL;
      FOR I TO LEN
        REPEAT
          PDEF.FORMALS[I].TYPE <- EVFMS(FRMLS.CAR.CAR);
          PDEF.FORMALS[I].SYM <- FRMLS.CAR.CDR.CAR;
          FRMLS <- FRMLS.CDR;
        END;
      PDEF.RETYPE <- RTYPE;
      BEGIN
        DECL PS:PTR(STRING) BYVAL ALLOC(STRING BYVAL 'PROC(');
        FOR I TO LEN
          REPEAT
            PS <- CONCAT(PS, PTYPE(PDEF.FORMALS[I].TYPE));
            PS <- CONCAT(PS, % );
            NAME <- PDEF.FORMALS[I].SYM;
            PS <- CONCAT(PS, BASIC\STR(NAME));
            PS <- CONCAT(PS, %,);
          END;
        PS[LENGTH(PS)] <- %;;
        PS <- CONCAT(PS, PTYPE(RTYPE));
        PS <- CONCAT(PS, %));
        NAME <- HASH(VAL(PS));
      END;
      RESULT <- GETDDB(NAME, ROUTINE.D, "PTR");
      RESULT.FINFLG => RESULT;
      RESULT.PROCD <- PDEF;
      RESULT.PROCFLG <- TRUE;
      DEFDDB(RESULT);
    END;
```

```
CMPDDB <-
  EXPR(M:MODE; BOOL)
    BEGIN
      DECL DEPTH:INT;
      DECL CRCFLG, UCMFLG, BOUND:BOOL;
      M.FINFLG => TRUE;
      BOUND <- TRUE;
      BIND(M);
      UCMFLG => ERROR("UNRESOLVED MODE IN MODE DEF");
      CRCFLG => ERROR("CIRCULAR MODE DEF");
      BOUND -> FINISH(M);
      BOUND;
    END;

BIND <-
  EXPR(M:MODE)
    BEGIN
      M.FINFLG => NOTHING;
      M.CYCFLG => DEPTH = M.DEPTH -> CRCFLG <- TRUE;
      M.CYCFLG <- TRUE;
      M.DEPTH <- DEPTH;
      M.CLASS = "PTR" -> DEPTH <- DEPTH + 1;
      ITERATE(M, BIND1);
      M.CYCFLG <- FALSE;
      DEPTH <- M.DEPTH;
    END;

BIND1 <-
  EXPR(F:FORM SHARED, M:MODE)
    BEGIN
      MD(VAL(F)) = ATOM AND MD(VAL(F <- EVFMS(F))) = ATOM => BOUND <- FALSE;
      F.CLASS = "GENERIC" AND M.PROCD = NIL -> UCMFLG <- TRUE;
      BIND(F);
    END;

FINISH <- EXPR(M:MODE) [) ITERATE(M, FINISH2); M.FINFLG => NOTHING; FINISH1(M) (];
```

```
FINISH1 <-
  EXPR(M:MODE BYVAL)
    BEGIN
      REPEAT
        M.FINFLG OR [) M.FINFLG <- TRUE; BUILD\MODE\FUNCTIONS(M); (M <- M.UR) = NIL (] => NOT
HING;
      END;
    END;

FINISH2 <-
  EXPR(F:FORM SHARED, M:MODE) [) F.FINFLG => NOTHING; F.CLASS = "PTR" -> FINISH1(F); FINISH(F
) (];

ITERAT <-
  EXPR(A:MODE, B:ROUTINE)
    BEGIN
      DECL MT:INT;
      'SET MT= -1 FOR PROC, 1 FOR PTR(M1, M2...)';
      A.CLASS = "ROW" => B(A.D.SBMD);
      A.CLASS = "GENERIC" => FOR I TO LENGTH(VAL(A.D)) REPEAT B(A.D[I]) END;
      ';MUST BE A POINTER';
      'IF PROC MODE, ITERATE OVER PROCD COMPONENTS';
      A.PROCFLG =>
        BEGIN
          MT <- - 1;
          B(A.PROCD.RETYPE);
          FOR I TO LENGTH(A.PROCD.FORMALS) REPEAT B(A.PROCD.FORMALS[I].TYPE) END;
        END;
      'IF A SIMPLE   POINTER, CHECK ONE MODE GIVEN BY D FIELD';
      MT <- 1;
      MD(VAL(A.D)) = DDB => B(A.D);
      'OTHERWISE CHECK ALL COMPONENT MODES';
      FOR I TO LENGTH(VAL(A.D)) REPEAT B(A.D[I]) END;
    END;
```

```
MODECOMPILER <-
  EXPR(AB:MODE; MODE)
    BEGIN
      DECL WRDSIZ:INT BYVAL 36;
      DECL IPTRSZ:INT BYVAL 18;
      A.CLASS = "ROW" => BLDRF();
      A.CLASS = "STRUCT" => BLDSF();
      A.CLASS = "PTR" => BLDPF();
      BLDOF();
    END;

BLDSF <-
  EXPR(AB:MODE; MODE) [) DECL NLUI, DVL, BITSZ, CNSTG:INT; BLDSSF(); BLDSAF(); BLDSGF(); AB (]
;

BLDSAF <-
  EXPR()
    BEGIN
      NOT AB.LRFLG => [) AB.SAFLG => ALAF(8); ALAF(6) (];
      NOT AB.WDFLG => ALAF(1);
      AB.SZ = 1 => ALAF(2);
      ALAF(3);
    END;

BLDSGF <- EXPR() [) AB.WDFLG => ALGF(8); ALGF(9) (];
```

```
EVDBCL <-
  EXPR(F:FORM, RHS:MODE; MODE)
    BEGIN
      DECL UFM:FORM;
      DECL RESULT:MODE;
      'IF LEFT HAND SIDE IS NEITHER A SYMBOL OR A MODE,';
      'THEN EVALUATE IT TO A SYMBOL OR A MODE AND';
      'POSSIBLY A LIST OF USER DEFINED MODE FUNCTIONS';
      NOT (MD(VAL(F)) = ATOM OR MD(VAL(F)) = DDB) ->
        [) F <- EVAL(F); MD(VAL(F)) = ATOM => NOTHING; UFM <- F.CDR; F <- EVAL(F.CAR) (];
      'ATTEMPT TO ASSOCIATE THE LHS WITH A MODE';
      MD(VAL(F)) = ATOM -> F <- SYTOM(F);
      'IF A MODE, LHS UR FIELD SHOULD EQUAL RHS MODE';
      MD(VAL(F)) = DDB -> [) F.UR = RHS => RESULT <- F; ERROR('ILLEGAL MODE DEF') (];
      'IF LHS IS AN ATOM, THEN MAKE A NEW MODE RELATED';
      'TO THE RHS MODE BY COPYING AND THEN MODIFYING';
      'RHS MODE%'S DDB';
      MD(VAL(F)) = ATOM ->
        BEGIN
          RESULT <- ALLOC(DDB BYVAL VAL(RHS));
          RHS.UFN # NIL -> RESULT.UFN <- ALLOC(SEQ(REF) BYVAL VAL(RHS.UFN));
          RESULT.UR <- RHS;
          RESULT.NAME <- F;
          RESULT.NAME.SBLK = NIL -> RESULT.NAME.SBLK <- ALLOC(SBLOCK);
          RESULT.NAME.SBLK.CONSTF <- RESULT;
          RESULT.PFLG <- TRUE;
        END;
      'IF NO USER FUNCTIONS, THEN RETURN NEW OR OLD MODE';
      UFM = NIL => RESULT;
      'IF ANY NEW USER FUNCTIONS, THEN SET THEM INTO';
      'THE RESULT%'S UFN TABLE WHETHER MODE IS NEW OR NOT';
      RESULT.UFN = NIL -> RESULT.UFN <- ALLOC(SEQ(REF) SIZE 5);
      FOR I TO LISTLENGTH(UFM) REPEAT RESULT.UFN[I] <- EVFRTN(UFM.CAR); UFM <- UFM.CDR; NO
THING END;
      RESULT;
    END;
```

```
BLDSSF <-
  EXPR()
    BEGIN
      DECL TMP:MODE;
      DECL WFLG:BOOL;
      DECL J, LINK, LINK1:INT;
      DECL LEN:INT BYVAL LENGTH(AB.D);
      DECL STR:REF BYVAL ALLOC(SEQ(STRTE) SIZE LEN);
      DECL WRK:VECTOR(LEN, INT);
      DECL CHAINS:VECTOR(WRDSIZ + 2, INT);
      DECL CLRI:INT BYVAL WRDSIZ + 1;
      DECL CLUI:INT BYVAL WRDSIZ + 2;
      'CALCULATE THE NUMBER OF LENGTH UNRESOLVED ITEMS';
      'AND SET LRFLG, SAFLG, AND EPFLG.';
      FOR I TO LEN
        REPEAT
          WRK[I] <- WRDSIZ;
          TMP <- AB.D[I].TYPE;
          NOT TMP.LRFLG -> [) NLUI <- NLUI + 1; AB.SAFLG <- TMP.SAFLG (];
          TMP.EPFLG -> AB.EPFLG <- TRUE;
        END;
      NLUI = 0 -> AB.LRFLG <- TRUE;
      NLUI GT 1 -> AB.SAFLG <- FALSE;
      'RESERVE SPACE FOR TOTAL LENGTH IF ANY LUIS';
      NLUI # 0 -> WRK[1] <- WRK[1] - IPTRSZ;
      'PUT MODES FROM STRUCTURE DEFINITION INTO STRUCT TABLE';
      'CHAIN ALL BYTE ITEMS BY SIZE AND ALL LENGTH RESOLVED';
      'AND LENGTH UNRESOLVED ITEMS. CHAINS[LINK]';
      'CONTAINS THE FIRST ENTRY FOR A CHAIN OF SIZE LINK-1';
      'IF LE WRDSIZ.  OTHERWISE, IT MARKS THE CHAIN FOR LR';
      'ITEMS OR LU ITEMS.';
      FOR I TO LEN
        REPEAT
          TMP <- STR[I].SBMD <- AB.D[I].TYPE;
          LINK <- CLUI;
          TMP.LRFLG ->
            BEGIN
              TMP.WDFLG => [) WFLG <- TRUE; LINK <- CLRI (];
              LINK <- TMP.SZ + 1;
              BITSZ <- BITSZ + TMP.SZ;
            END;
          STR[I].RLPT <- CHAINS[LINK];
          CHAINS[LINK] <- I;
        END;
      BITSZ GE WRDSIZ -> WFLG <- TRUE;
```

```
'ACCUMULATE DOPE VECTOR LENGTH FOR ALL LENGTH';
'UNRESOLVED ITEMS.  MAKE INTERNAL POINTERS FOR ALL BUT';
'FIRST ITEM WHICH DOES NOT NEED ONE.';
(LINK <- CHAINS[CLUI]) # 0 ->
  BEGIN
    DVL <- STR[LINK].SBMD.SZ;
    LINK <- STR[LINK].RLPT;
    REPEAT
      LINK = 0 => NOTHING;
      DVL <- DVL + (1 <- STR[LINK].SBMD.SZ);
      SHARED J ← 0 REPEAT WRK[J + 1] GE IPTRSZ => NOTHING END;
      WRK[J + 1] <- WRK[J + 1] - IPTRSZ;
      LINK1 <- STR[LINK].RLPT;
      'MAKE AN INTERNAL POINTER FETCHING INSTRUCTION';
      STR[LINK].RLPT <- MAKIPT(J);
      LINK <- LINK1;
    END;
  END;
'PROCESS BYTE ITEMS FROM LARGEST TO SMALLEST,';
'ALL OF A GIVEN SIZE AT THE SAME TIME';
'WRK[J+1] GIVES NUMBER OF FREE BITS IN J TH WORD';
'OF STRUCT BEING LAID OUT';
FOR I TO WRDSIZ
  REPEAT
    DECL BYTSIZ:INT BYVAL WRDSIZ - I;
    LINK <- CHAINS[BYTSIZ + 1];
    REPEAT
      LINK = 0 => NOTHING;
      SHARED J ← 0 REPEAT WRK[J + 1] GE BYTSIZ => NOTHING END;
      WRK[J + 1] <- WRK[J + 1] - BYTSIZ;
      LINK1 <- STR[LINK].RLPT;
      'MAKE A BYTE POINTER FOR THIS ENTRY USING COMPONENT';
      'SIZE, LEFT TO RIGHT PACKING, AND NUMBER OF BITS LEFT';
      'IN J TH WORD TO DETERMINE POSITION';
      STR[LINK].RLPT <- MAKBYT(J);
      LINK <- LINK1;
    END;
  END;
  'COUNT WORDS USED FOR INTERNAL PTRS AND BYTE ITEMS';
  SHARED J ← 0 REPEAT WRK[J + 1] = WRDSIZ => NOTHING END;
  'NOW PROCESS WORD ITEMS WITH J = FIRST FREE ADDRESS';
  LINK <- CHAINS[CLRI];
  REPEAT
    LINK = 0 => NOTHING;
    LINK1 <- STR[LINK].RLPT;
    STR[LINK].RLPT <- J;
    J <- J + (1 <- STR[LINK].SBMD.SZ);
    LINK <- LINK1;
  END;
  'SET THE STRUCTURE TABLE INTO THE DDB';
  'AND SET THE CONSTANT STORAGE SIZE.';
  AB.STRTB <- STR;
  CONSTG <- J;
  'NOW SET SZ AND WDFLG AND PICK A SELECTION FUNCTION';
  NLUI GT 0 => [) STR[CHAINS[CLUI]].RLPT <- J; AB.SZ <- DVL; AB.WDFLG <- TRUE; ALSF(15) (];
  J = 0 => ALSF(16);
  NOT (AB.WDFLG <- WFLG) => [) AB.SZ <- BITSZ; ALSF(16) (];
  AB.SZ <- J + 1;
  ALSF(15);
END;
```

```
EVSTR <-
  EXPR(L:FORM LISTED; MODE)
    BEGIN
      DECL NAME:SYMBOL;
      DECL RESULT:MODE;
      DECL LEN:INT BYVAL LIST\LENGTH(L);
      DECL DEF:REF BYVAL ALLOC(SEQ(FDS) SIZE LEN);
      'LOAD THE STRUCTURE DEFINITION TABLE WITH NAMES AND';
      'MODES (POSSIBLY SYMBOLIC) OF THE COMPONENTS';
      FOR I TO LEN
        REPEAT DEF[I].SYM <- L.CAR.CAR; DEF[I].TYPE <- EVFMS(L.CAR.CDR.CAR); L <- L.CDR END;
      'CONSTRUCT THE NAME OF THE MODE BY GOING THROUGH THE';
      'D FIELD COMPONENT BY COMPONENT.';
      BEGIN
        DECL PSTRING:PTR(STRING) BYVAL ALLOC(STRING BYVAL 'STRUCT(');
        FOR I TO LEN
          REPEAT
            PSTRING <- CONCAT(PSTRING, BASIC\STR(DEF[I].SYM));
            PSTRING <- CONCAT(PSTRING, %:);
            PSTRING <- CONCAT(PSTRING, PTYPE(DEF[I].TYPE));
            PSTRING <- CONCAT(PSTRING, %,);
          END;
        PSTRING[LENGTH(PSTRING)] <- %);
        NAME <- HASH(VAL(PSTRING));
      END;
      'SEE IF AN ALREADY DEFINED MODE ASSOCIATED WITH';
      'THE NAME.  IF NOT, THEN OBTAIN A NEW DDB AND';
      'SET THE NAME, D, AND CLASS FIELDS';
      RESULT <- GETDDB(NAME, DEF, "STRUCT");
      'IF FINFLG ALREADY TRUE, THEN RETURN OLD MODE';
      'OTHERWISE ATTEMPT TO COMPLETE NEW PARTIAL DDB';
      RESULT.FINFLG => RESULT;
      DEFDDB(RESULT);
    END;

↑;
```

```
'       CONVERSION ROUTINES
';

CONVPT <-
  EXPR(M:MODE)
    BEGIN
      NOT CMPDDB(M) => ERROR('CAN%'T COMPLETE MODE');
      DECL MM:MODE LIKE EV\MVAL(OBJECT);
      VAL(GETREF(OBJECT)) = NIL -> MM ← NONE;
      DECL CASE:INT LIKE CONST(VECTOR(2, BOOL) OF M.HFLG, M.SFLG);
      OR(CASE = 0,
         CASE = 3 AND MM = M.D,
         FOR I TO LENGTH(M.D) REPEAT M.D[I] = MM => TRUE; FALSE END) => RETURN\RESULT(M, B
);
      BREAK('TYPE FAULT');
    END;

CKSINT <- EXPR(M:MODE; BOOL) AND(M.CLASS = "ROW", M.LRFLG, M.D.TYPE = BOOL, M.D.LENGTH LE
 36);

SYS\CONV <-
  EXPR(M:MODE)
    BEGIN
      COVERS(M, A) => A ← M;
      DECL UCFN:REF BYVAL [) M.UFN # NIL => M.UFN[2]; NIL (];
      UCFN # NIL => SAPPLY(UCFN, OBJECT, M);
      M.CLASS = "PTR" AND A.CLASS = "PTR" => CONVPT(M);
      M = NONE => OBJECT ← CONST(POINTER);
      M = INT => [) BREAK(INT) (];
      M = REAL => [) BREAK(REAL) (];
      CKSINT(M) AND A = INT => [) BREAK(SINT) (];
      BREAK('TYPE FAULT');
    END;

↑;
```

```
'       USER ROUTINES
';

HASH\SUBR <-
  EXPR(S:STRING; ECL\SYMBOL)
    BEGIN
      DECL X:ECL\SYMBOL LIKE EV\HASH(VAL(GETREF(NS[NP - 1].OBJECT)), VAL(GETREF(NS[NP].OB
JECT)));
      RETURN\RESULT(ECL\SYMBOL, CONST(SITE OF X));
      X;
    END;

LENGTH\SUBR <- EXPR(X:ANY; INT) RETURN\RESULT(INT, CONST(SITE OF EV\LENGTH(NS[NP].OBJEC
T)));

VAL\SUBR <- EXPR(X:ANY; ANY) [) DECL Y:ANY LIKE EV\VAL(NS[NP].OBJECT); RETURN\RESULT(Y.A,
 Y.B) (];

MD\SUBR <- EXPR(X:ANY; MODE) RETURN\RESULT(MODE, CONST(SITE OF NS[NP].OBJECT.A));

EVAL\SUBR <- EXPR(F:FORM; ANY) ECL\EVAL(VAL(GETREF(NS[NP].OBJECT)));

↑;
```

```
'The following constitutes an environment for interpretation of
an ECL process.
';

VS ← CONST(VALUE\STACK SIZE 100);

NS ← CONST(NAME\STACK SIZE 100);

CS ← CONST(CONTROL\STACK SIZE 100);

NP ← 0;

CP ← 1;

CS[1].TYPE ← "TOP";

VP ← 0;

OBJECT ← CONST(BIGREF);

NO\SITE ← OBJECT.B;

CLEAR <-
  EXPR()
    BEGIN
      SHARED NP BY - 1 TO 1 REPEAT NS[NP] ← CONST(NS\FRAME) END;
      SHARED VP BY - 1 TO 1 REPEAT VS[VP] ← NIL END;
      NP ← VP ← 0;
      SHARED CP BY - 1 TO 1 REPEAT CS[CP] ← CONST(CS\FRAME) END;
      CS[CP ← 1].TYPE ← "TOP";
      OBJECT ← CONST(BIGREF);
      HSTRTAB ← MAKEHASH(MODE, STRTAB, 25);
      STBL ← MAKEHASH(SYMBOL, ECL\SYMBOL, 25);
      INIT();
      RESET();
    END;

ATOM\LIST ← CONST(PTR("ECL\ATOM"));

MAKEHASH = NOTHING -> LOADB "SYS:HASH";
```

```
STBL ← MAKEHASH(SYMBOL, ECL\ATOM, 25);

HSTRTAB ← MAKEHASH(MODE, STRTAB, 25);

ATOMS <- CONST(FORM);

DATA ←
  EXPR()
    BEGIN
      ATOM\LIST ← NIL;
      DECL L:FORM BYVAL ATOMS;
      DECL F:FORM;
      REPEAT
        L = NIL => NOTHING;
        DECL N:FORM BYVAL (F ← L.CAR).CAR;
        MVAL(N) = REF -> N ← VAL(VAL(N));
        DECL FLAG:BOOL;
        DECL M:MODE LIKE REAL\EVAL((F ← F.CDR).CAR);
        DECL V:ANY BYVAL REAL\EVAL(F.CDR.CAR);
        DECL ATM:ECL\SYMBOL LIKE FINDHASH(STBL, N, FLAG);
        NOT FLAG -> ATM ← ALLOC(ECL\ATOM);
        BEGIN
          M = SUBR =>
            BEGIN
              V.CAR # "EXPR" => ERROR("MAKE\SUBR INAPPROPRIATELY APPLIED");
              DECL N:INT BYVAL LIST\LENGTH((V ← V.CDR).CAR);
              DECL S:PTR(ECL\SUBR) LIKE ALLOC(ECL\SUBR SIZE N);
              DECL P:FORM BYVAL V.CAR;
              FOR I TO N
                REPEAT
                  S.FORMALS[I].SYM ← P.CAR.CDR.CDR.CAR;
                  S.FORMALS[I].TYPE ← P.CAR.CDR.CAR;
                  P ← P.CDR;
                END;
              S.RETYPE ← REAL\EVAL((V ← V.CDR).CAR);
              S.BODY ← V.CDR.CAR;
              ATM.TLB ← CONST(BIGREF OF ROUTINE, CONST(SITE OF S));
            END;
          ATM.TLB ← CONST(BIGREF OF M, CONST(SITE OF V));
        END;
        DECL ATM:ECL\SYMBOL LIKE FINDHASH(STBL, N);
        ATM.LINK ← ATOM\LIST;
        ATOM\LIST ← ATM;
        L ← L.CDR;
      END;
    END;

↑;
```

```
'
"       THE FOLLOWING ARE FOR DEBUGGING AND/OR UNCLASSIFIED
"       ROUTINES (AS TO PRIMITIVE, MODEL, OR ENVIRONMENTAL).
';

Q <- QUOTE(ECL\EVAL(QUOTE(STRUCT(A:REAL, B:REAL))));

MODES <-
  QUOTE(BEGIN
          SITE :: STRUCT(V:REF, VP:INT);
          ECL\REF :: STRUCT(A:MODE, B:"SITE");
          ECL\SUBR :: STRUCT(BODY:FORM, RETYPE:"ECL\MODE", PRMD:"ECL\MODE", FORMALS:SEQ("
ECL\FDS"));
          ECL\BASIC :: ONEOF(NONE, BOOL, CHAR, INT, REAL, "ECL\MODE", "ECL\SYMBOL", "ECL\STRIN
G");
          ECL\ARITH :: ONEOF(INT, REAL);
          ECL\FORM :: PTR(INT, REAL, "ECL\REF", "ECL\DDB", "ECL\ATOM", "ECL\DTPR");
          ECL\DTPR :: STRUCT(CAR:"ECL\FORM", CDR:"ECL\FORM");
          ECL\MODE :: PTR("ECL\DDB");
          ECL\ROUTINE :: PTR("ECL\DTPR", "ECL\CEXPR", "ECL\SUBR");
          ECL\STRING :: SEQ(CHAR);
          ECL\SYMBOL :: PTR("ECL\ATOM");
          ECL\FDS :: STRUCT(TYPE:"ECL\FORM", SYM:"ECL\SYMBOL");
          ECL\HWD :: VECTOR(18, BOOL);
          ECL\PDPTR :: PTR("ECL\PDESC");
          ECL\PDESC :: STRUCT(FORMALS:SEQ("ECL\FDS"), RETYPE:"ECL\FORM");
          ECL\SBLOCK ::
            STRUCT(SINFO:"ECL\HWD", PLIST:"ECL\FORM", RMTCH:"ECL\FORM", CONSTF:"ECL\FORM");
          ECL\ATOM ::
            STRUCT(TLB:"ECL\REF", SBLK:PTR("ECL\SBLOCK"), LINK:PTR("ECL\ATOM"), PNAME:"ECL\S
TRING");
          ECL\STRTE :: STRUCT(JUNK:"ECL\HWD", TYPE:"ECL\FORM", LENGTH:INT);
```

```
 ECL\DDB ::
   STRUCT(SFLG:BOOL,
          HFLG:BOOL,
          PROCFLG:BOOL,
          EPFLG:BOOL,
          WDFLG:BOOL,
          LRFLG:BOOL,
          FINFLG:BOOL,
          SAFLG:BOOL,
          BNDFLG:BOOL,
          CRCFLG:BOOL,
          CYCFLG:BOOL,
          PFLG:BOOL,
          BIFLG:BOOL,
          QMWFLG:BOOL,
          NAME:"ECL\SYMBOL",
          PROCD:"ECL\PDPTR",
          CLASS:"ECL\SYMBOL",
          UR:"ECL\MODE",
          DO:PTR(SEQ(INT)),
          STRTB:PTR(SEQ("ECL\STRTE")),
          UFN:PTR(SEQ("ECL\REF")),
          SFN:"ECL\FORM",
          AFN:"ECL\FORM",
          GENFN:"ECL\FORM",
          TRFN:"ECL\FORM",
          SZ:"ECL\HWD",
          D:"ECL\REF");
END);
```

```
INIT <-
  EXPR()
    BEGIN
      ATOM\LIST ← NIL;
      STBL ← MAKEHASH(SYMBOL, ECL\SYMBOL, 50);
      DECL A:FORM BYVAL DATA;
      REPEAT
        DECL CASE:FORM BYVAL A.CAR;
        DECL M:MODE BYVAL CASE.CAR;
        CASE ← CASE.CDR;
        REPEAT
          DECL C:FORM BYVAL CASE.CAR;
          DECL S:SYMBOL BYVAL [) DECL C:FORM LIKE C.CAR; MVAL(C) = ATOM => C; VAL(VAL(C)) (]
;
          DECL ATM:ECL\SYMBOL LIKE EV\HASH(BASIC\STR(S));
          DECL TOP:BIGREF SHARED ATM.TLB;
          TOP ← CONST(BIGREF OF NONE, NO\SITE);
          ATM.PNAME ← S;
          BEGIN
            M = SUBR =>
              BEGIN
                DECL V:ANY BYVAL VAL(C.CDR.CAR.TLB);
                V.CAR # "EXPR" -> ERROR('ECL\SUBR FOR WRONG FORM');
                DECL N:INT BYVAL LIST\LENGTH((V ← V.CDR).CAR);
                DECL SBR:PTR(ECL\SUBR) LIKE ALLOC(ECL\SUBR SIZE N);
                DECL P:FORM BYVAL V.CAR;
                FOR I TO N
                  REPEAT
                    SBR.FORMALS[I].SYM ← P.CAR.CDR.CDR.CAR;
                    SBR.FORMALS[I].TYPE ← P.CAR.CDR.CAR;
                    P ← P.CDR;
                  END;
                SBR.RETYPE ← REAL\EVAL((V ← V.CDR).CAR);
                SBR.BODY ← V.CDR.CAR;
                TOP.A ← ROUTINE;
                TOP.B ← CONST(SITE OF SBR);
              END;
            TOP.A ← M;
            TOP.B ← CONST(SITE OF [) C.CDR = NIL => VAL(S.TLB); VAL(C.CDR.CAR.TLB) (]);
          END;
          (CASE ← CASE.CDR) = NIL => NOTHING;
        END;
        (A ← A.CDR) = NIL => NOTHING;
      END;
    END;
```