Research Report

IMPLEMENTING PROLOG

- compiling predicate logic programs

Volume 2

by

David H D Warren

D.A.I. Research Report No. 40

Department of Artificial Intelligence
University of Edinburgh

IMPLEMENTING PROLOG
- compiling predicate logic programs

Volume 2

by

David H D Warren

D.A.I. Research Report No. 40

May 1977

Volume 2 - Appendices

1.0 PLM REGISTERS, DATA AREAS AND DATA STRUCTURES

Here we summarise the state of the PLM during unification. Recall that the machine is attempting to match the head of the current clause against the current goal. A failure to unify will cause backtracking to the latest choice point where the parent goal will be reconsidered.

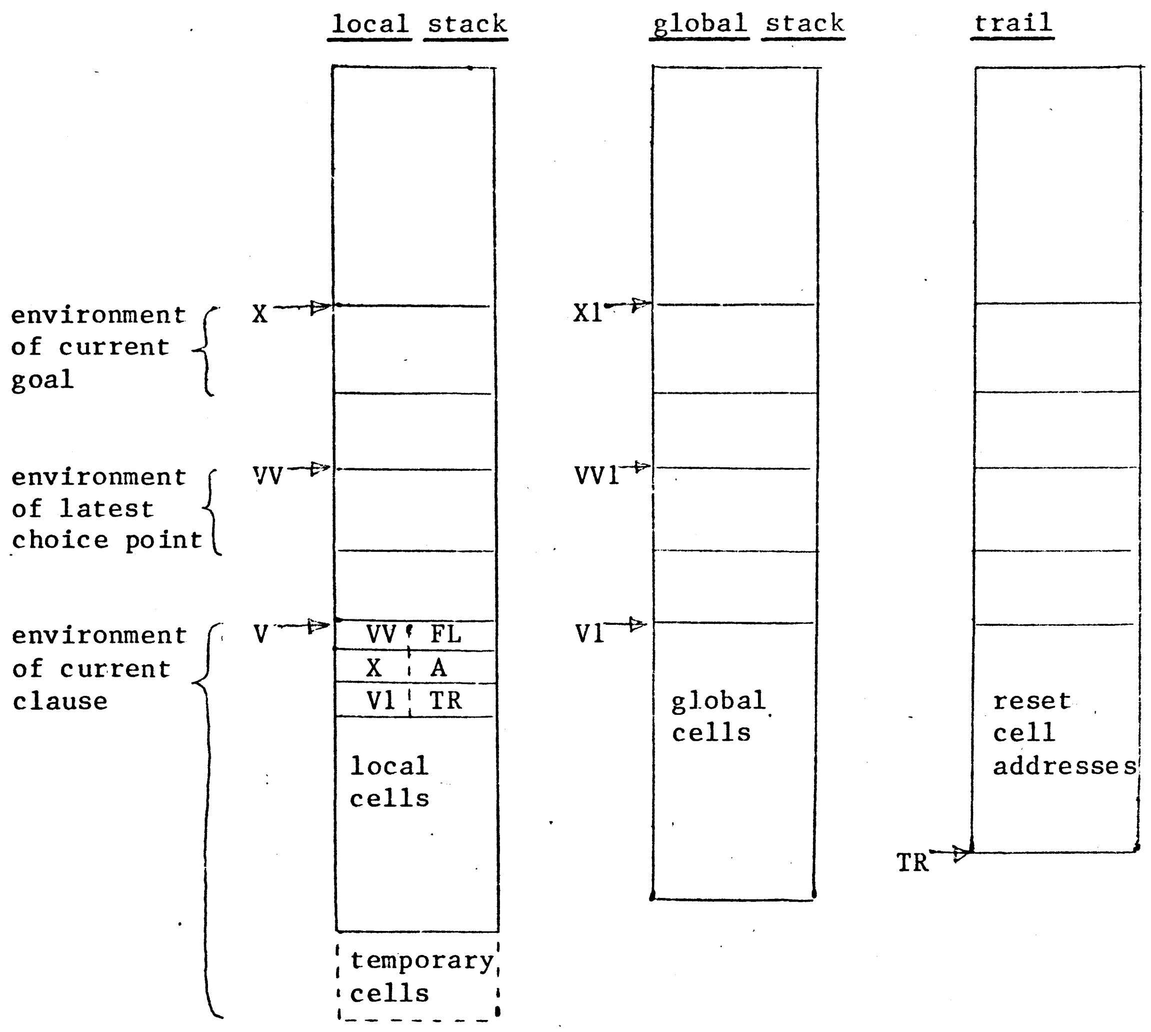
Registers

V	top of local stack = local frame for current clause
V1	top of global stack = global frame for current clause
X	local frame for current goal
X1	global frame for current goal
VV	local frame for latest choice point
VV1	global frame for latest choice point
TR	pushdown list pointer for the trail
PC	current instruction
A	arguments and continuation of current goal
B	a skeleton involved in unification
Y	the global frame corresponding to B

Other registers used in the DEC10 implementation

FL	failure label, but only when VV=V
T	construct passed as argument to a unification routine
B1	construct passed as argument to a unification routine
C	return address for a runtime routine
R1	temporary results
R2	temporary results

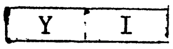
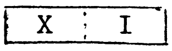
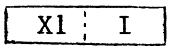
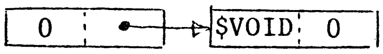
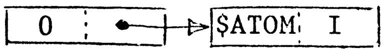
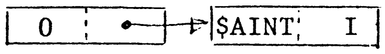
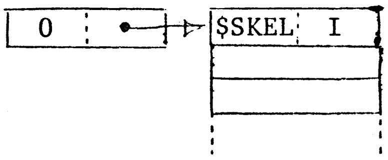
Data areas and environment layout

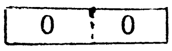
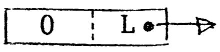
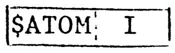
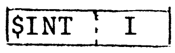
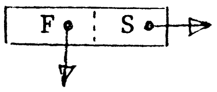


Fields of an environment

A	parent goal's arguments and continuation
X	parent goal's local frame
V1	global frame corresponding to this local frame
TR	state of TR when parent goal was invoked
FL	failure label, if any, for parent goal; ie. an alternative clause
VV	local frame for the choice point prior to the parent goal

Representations for source and constructed terms

Source term (literal)	DEC10 form
var(I)	
local(I)	
global(I)	
void	
[atom(I)]	
[int(I)]	
[fn(I),...]	

Constructed term (cell value)	DEC10 form
undef	
ref(L)	
atom(I)	
int(I)	
mol(S,F)	

2.0 PLM INSTRUCTIONS AND LITERALS

2.1 Summary

literals

var(I)	local(I)
atom(I)	global(I)
int(I)	void
fn(I)	

unification

uvar(N,F,I)	uvarl(N,F,I)
uref(N,F,I)	urefl(N,F,I)
uatom(N,I)	uatoml(N,I)
uint(N,I)	uintl(N,I)
uskel(N,S)	uskell(N,S)
uskeld(N,I)	
uskelc(N,S)	init(I,J)
	localinit(I,J)

control transfer

ifdone(L)
call(L)
try(L)
trylast(L)

"red tape"

enter	cut(I)
neck(I,J)	neckcut(I,J)
foot(N)	neckcutfoot(J,N)
neckfoot(J,N)	fail

extra instructions for clause indexing

gsect	switch(N)
ssect(L,C)	case(L)
ssectlast(L)	
endssect	ifatom(I,L)
	ifint(I,L)
ugvar(I)	iffn(I,L)
tryatom(I,C)	goto(L)
tryint(I,C)	notlast
tryskel(S,C)	
trylastatom(I,C)	
trylastint(I,C)	
trylastskel(S,C)	

2.2 var(I)

Use: An occurrence of a variable in a skeleton. I is the number of the global variable.

Example: 'var(2)' for:-

```
     reverse(cons(X,L1),L2,L3) :- reverse(L1,cons(X,L2),L3).
                                                     **
```

Effect: Serves as a pointer to a construct which is the value of the global variable.

DEC10 form:

```
                WD i(Y)          ;where i=I.
```

2.3 atom(I)

Use: An occurrence of an atom in a skeleton or goal is represented by the address of a literal 'atom(I)' where I identifies the atom.

Example: '[atom(nil)]' for:-

```
     sort(L0,L) :- qsort(L0,L,nil).
                              ***
```

Effect: The address of the atom literal serves as a pointer to a construct representing the atom.

DEC10 form:

```
                WD label

     label:     XWD $ATOM,i      ;where i = functor number of atom.
```

2.4 int(I)

Use: An occurrence of an integer in a skeleton or goal is represented by the address of a literal 'int(I)' where I is the value of the integer.

Example: '[int(29)]' for:-

```
     leapyear(X) :- duration(february,X,29).
                                       **
```

Effect: The address of the integer literal serves as a pointer to a construct representing the integer.

DEC10 form:

```
              WD label

     label:   XWD $INT,i        ;where i=I.
```

2.5 fn(I)

Use: An occurrence of a skeleton term in a goal or in a non-mode '+' position in the head of a clause is represented by the address of a skeleton literal, which commences with a functor literal 'fn(I)' where I identifies the functor of the skeleton.

Example: '[fn(cons),var(0),var(2)]' for:-

```
     reverse(cons(X,L1),L2,L3) :- reverse(L1,cons(X,L2),L3).
                                             **********
```

Effect: The address of the skeleton literal serves as the skeleton component of the molecule which represents the subterm.

DEC10 form:

```
              WD label

     label:   XWD $SKEL,i       ;where i = skeleton's functor number.
              ...               ;inner literals
```

2.6 local(I)

Use: An occurrence of a local variable as an argument of a goal. I is the number of the local variable.

Example: 'local(0)' for:-

```
     reverse(cons(X,L1),L2,L3) :- reverse(L1,cons(X,L2),L3).
                                                    **
```

Effect: Serves as a pointer to a construct which is the value of the local variable.

DEC10 form:

```
                 WD i(X)          ;where i=I+3.
```

2.7 global(I)

Use: An occurrence of a global variable as an argument of a goal. I is the number of the global variable.

Example: 'global(1)' for:-

```
     reverse(cons(X,L1),L2,L3) :- reverse(L1,cons(X,L2),L3).
                                           **
```

Effect: Serves as a pointer to a construct which is the value of the global variable.

DEC10 form:

```
                 WD i(X1)         ;where i=I.
```

2.8 void

Use: An occurrence of a void variable (ie. the variable occurs nowhere else) as an argument of a goal.

Example: 'void' for:-

```
employed(X) :- employs(Y,X).
                       *
```

Effect: Any instruction which attempts to unify against this outer literal behaves as a (successful) no-operation.

DEC10 form:

```
          WD label

label:    XWD $VOID,0
```

2.9 uvar(N,F,I)

Use: Argument N in the head of a clause is the first occurrence of a variable of type F (local or global), number I. (A temporary variable will have F=local.)

Example: 'uvar(1,global,2)' for:-

```
     reverse(cons(X,L1),L2,L3) :- reverse(L1,cons(X,L2),L3).
                          **
```

Effect: The outer literal representing argument N of the current goal is accessed via register A and the dereferenced result is assigned to cell I in frame F of the current environment, unless the result is a local reference and F is global. In the latter case, a reference to cell I in frame E is assigned to the incoming reference, and the assignment is trailed if necessary.

DEC10 form:

```
              MOVE T,@n(A)        ;where n=N.
              TLNN T,$1MA
              JSP C,$UVAR
              MOVEM T,i(reg)      ;where i=I+3 and reg=V if F=local
                                  ;or    i=I and reg=V1 if F=global.
```

If N<9 and fastcode is not required, this is condensed to:-

```
              JSP C,routine
              MOVEM T,i(reg)

     routine: MOVE T,@n(A)
              TLNN T,$1MA
              JSP C1,...
              JRST 0(C)
```

2.10 uvarl(N,F,I)

Use: Argument N of a skeleton at level 1 in the head of a clause is the first occurrence of a variable of type F (local or global), number I. The instruction is not needed if the skeleton is in a mode '-' position.

Example: 'uvarl(1,local,0)' for:-

```
:-mode reverse(+,+,?).
reverse(cons(X,L1),L2,L3) :- reverse(L1,cons(X,L2),L3).
              **
```

Effect: The inner literal representing argument N of the matching skeleton is accessed via register B and the dereferenced result is assigned to cell I in frame F of the current environment. Note: if the result is a reference, it must refer to a global cell, which will therefore be at least as senior as the cell assigned.

DEC10 form:

```
              MOVE T,@n(B)       ;where n=N+1.
              TLNN T,$1MA
              JSP C,$UVAR1
              MOVEM T,i(reg)     ;where i=I+3 and reg=V if F=local
                                 ;or    i=I and reg=V1 if F=global.
```

If N<5 and fastcode is not required, this is condensed to:-

```
              JSP C,routine
              MOVEM T,i(reg)

     routine: MOVE T,@n(B)
              TLNN T,$1MA
              JSP C1,...
              JRST 0(C)
```

2.11 uref(N,F,I)

Use: Argument N in the head of a clause is a subsequent occurrence of a variable of type F (local or global), number I. (A temporary variable will have F=local.)

Example: 'uref(2,local,0)' for:-

```
     reverse(nil,L,L).
                   *
```

Effect: The outer literal representing argument N of the current goal is accessed via register A and the dereferenced result is unified with the dereferenced value of cell I in frame F of the current environment.

DEC10 form:

```
        MOVE B,@n(A)      ;where n=N.
        MOVE B1,i(reg)    ;where i,reg are as for 'uvar'.
        JSP C,$UREF
```

If N<5 this is condensed to:-

```
        MOVE B1,i(reg)
        JSP C,routine

routine: MOVE B,@n(A)
        ...
```

2.12 urefl(N,F,I)

Use: Argument N of a skeleton at level 1 in the head of a clause is a subsequent occurrence of a variable of type F (local or global), number I. The instruction is not needed if the skeleton is in a mode '-' position.

Example: 'urefl(0,global,0)' for:-

```
     concatenate(cons(X,L1),L2,cons(X,L3)) :- concatenate(L1,L2,L3).
                                 *
```

Effect: The inner literal representing argument N of the matching skeleton is accessed via register B and the dereferenced result is unified with the dereferenced value of cell I in frame F of the current environment.

DEC10 form:

```
          MOVE T,@n(B)      ;where n=N+1.
          MOVE B1,i(reg)    ;where i,reg are as for 'uvar'.
          JSP C,$UREF1
```

If N<3 this is condensed to:-

```
          MOVE B1,i(reg)
          JSP C,routine

 routine: MOVE T,@n(B)
          ...
```

2.13 uatom(N,I)

Use: Argument N in the head of a clause is an atom, identified by I.

Example: 'uatom(1,september)' for:-

```
     month(9,september).
             *********
```

Effect: The outer literal representing argument N of the current goal is accessed via register A and the dereferenced result is unified with atom I.

DEC10 form:

```
               MOVE T,@n(A)          ;where n=N.
               JSP C,$UATOM
               XWD $ATOM,i           ;where i = functor number of atom.

     $UATOM:   TLNN T,$1MAS
               JRST ...
               CAME T,0(C)
               JRST $FAIL
               JRST 1(C)
```

If N<8 this is condensed to:-

```
               JSP C,routine
               XWD $ATOM,i

     routine:  MOVE T,@n(A)
               TLNN T,$1MAS
               JSP C1,...
               CAME T,0(C)
               JRST $FAIL
               JRST 1(C)
```

2.14 uatoml(N,I)

Use: Argument N of a skeleton at level 1 in the head of a clause is an atom, identified by I.

Example: 'uatoml(1,nil)' for:-

```
    singleton(cons(X,nil)).
                     ***
```

Effect: The inner literal representing argument N of the matching skeleton is accessed via register B and the dereferenced result is unified with atom I.

DEC10 form:

```
            MOVE T,@n(B)        ;where n=N+1.
            JSP C,$UATOM
            XWD $ATOM,i         ;where i = functor number of atom.
```

If N<5 this is condensed to:-

```
            JSP C,routine
            XWD $ATOM,i

    routine: MOVE T,@n(B)
            ...
```

2.15 uint(N,I)

Use: Argument N in the head of a clause is an integer, value I.

Example: 'uint(0,9)' for:-

```
month(9,september).
     *
```

Effect: The outer literal representing argument N of the current goal is accessed via register A and the dereferenced result is unified with integer I.

DEC10 form:

```
                MOVE T,@n(A)        ;where n=N.
                JSP C,$UATOM
                XWD $INT,i          ;where i = value of the integer.
```

If N<8 this is condensed to:-

```
                JSP C,routine
                XWD $INT,i

       routine: MOVE T,@n(A)
                TLNN T,$1MAS
                JSP C1,...
                CAME T,0(C)
                JRST $FAIL
                JRST 1(C)
```

2.16 uintl(N,I)

Use: Argument N of a skeleton at level 1 in the head of a clause is an integer, value I.

Example: 'uintl(1,2)' for:-

```
differentiate(square(X),X,*(X,2)).
                           *
```

Effect: The inner literal representing argument N of the matching skeleton is accessed via register B and the dereferenced result is unified with nteger I.

DEC10 form:

```
                MOVE T,@n(B)        ;where n=N+1.
                JSP C,$UATOM
                XWD $INT,i          ;where i = value of the integer.
```

If N<5 this is condensed to:-

```
                JSP C,routine
                XWD $INT,i

     routine:   MOVE T,@n(B)
                ...
```

2.17 uskel(N,S)

Use: Argument N in the head of a clause is a skeleton term for which S is the address of a corresponding skeleton literal. (Not used for a mode '+' or mode '-' position.)

Example: 'uskel(2,[fn(cons),var(0),var(2)])' for:-

```
     concatenate(cons(X,L1),L2,cons(X,L3)) :- concatenate(L1,L2,L3).
                            **********
```

Effect: The outer literal representing argument N of the current goal is accessed via register A and dereferenced. If the result is a reference, a molecule is assigned to the cell referenced, the assignment is trailed if necessary and register Y is set to 'undef'. The molecule is constructed from S and the address of the current global frame given by register V1. If the result of the dereferencing is not a reference, a failure occurs unless the result is a molecule with the same functor as S. In the latter case register B is set to the address of the skeleton part of the matching molecule and register Y to the address of its (global) frame.

DEC10 form:

```
              MOVE B,@n(A)      ;where n=N.
              JSP C,$USK
              WD address        ;of literal S.

     $USK:    HLRZ Y,B          ;load type of B into Y.
              CAIGE Y,$MOLS     ;if B isn't a molecule
              JRST @table(Y)    ;  switch on Y.
              MOVE R1,0(B)      ;load functor of B.
              CAME R1,@0(C)     ;if different from functor of S
              JRST $FAIL        ;  then fail.
              JRST 1(C)         ;return to in-line code.
```

If N<5 this is condensed to:-

```
              JSP C,routine
              WD address

     routine: MOVE B,@n(A)
              ...
```

2.18 uskell(N,S)

Use: Argument N of a skeleton at level 1 in the head of a clause is another skeleton term for which S is the address of a corresponding skeleton literal.

Example: 'uskell(0,[fn(int),var(0)])' for:-

```
     expr(cons(int(N),S),S,N).
               ******
```

Effect: The inner literal representing argument N of the matching skeleton is accessed via register B and the dereferenced result is unified with the molecule formed from S and the global frame address in register Y.

DEC10 form:

```
               MOVE T,@n(B)        ;where n=N+1.
               JSP C,$USK1
               WD address          ;of literal S.

     $USK1:    HLRZ R1,T
               CAIGE R1,$MOLS
               JRST @table(R1)
               MOVE R2,@0(C)
               CAME R2,0(T)
               JRST $FAIL
```

If N<3 this is condensed to:-

```
               JSP C,routine
               WD address

     routine:  MOVE T,@n(B)
               ...
```

2.19 uskeld(N,I)

Use: Argument N in the head of a clause is a skeleton term, and this position has mode '+'. I identifies the functor of the skeleton term.

Example: 'uskeld(0,cons)' for:-

```
:-mode concatenate(+,+,-).
concatenate(cons(X,L1),L2,cons(X,L3)) :- concatenate(L1,L2,L3).
            ****
```

Effect: cf. 'uskel'. The result of dereferencing the matching outer literal is guaranteed to be a non-reference. A failure occurs unless it is a molecule with functor as indicated by I. Register B is set to the address of the skeleton part of the molecule and register Y to the address of the (global) frame.

DEC10 form:

```
              MOVE B,@n(A)        ;where n=N.
              JSP C,$USKD
              XWD $SKEL,i         ;cf. fn(I).

     $USKD:   HLRZ Y,B            ;load type of B into Y.
              CAIGE Y,$MOLS       ;if B isn't a molecule
              JRST @table(Y)      ;  switch on Y.
              MOVE R1,0(B)        ;load functor of B.
              CAME R1,0(C)        ;if different from fn(I)
              JRST $FAIL          ;  then fail.
              JRST 1(C)           ;return to in-line code.
```

If N<5 this is condensed to:-

```
              JSP C,routine
              XWD $SKEL,i

     routine: MOVE B,@n(A)
```

2.20 uskelc(N,S)

Use: Argument N in the head of a clause is a skeleton term, and this position has mode '-'. S is the address of a corresponding skeleton literal.

Example: 'uskelc(2,[fn(cons),var(0),var(1)])' for:-

```
:-mode concatenate(+,+,-).
concatenate(cons(X,L1),L2,cons(X,L3)) :- concatenate(L1,L2,L3).
                              **********
```

Effect: cf. 'uskel'. The result of dereferencing the matching outer literal is guaranteed to be a reference. A molecule formed from S with global frame address from V1 is assigned to the cell referenced and the assignment is trailed if necessary.

DEC10 form:

```
                MOVE B,@n(A)        ;where n=N.
                JSP C,$USKC
                WD address          ;of literal S.

    $USKC:      JUMPE B,UNDO        ;if B=undef goto UNDO.
    CONTINUE:                       CAILE B,$MAXREF ;if B is not a
 reference
                JRST 1(C)           ;  then it's a void so return.
                SKIPN R1,0(B)       ;if B is fully dereferenced
                JRST ASSIGN         ;  then goto ASSIGN.
                ...                 ;else continue dereferencing.

    UNDO:       MOVEI B,@-2(C)      ;undo initial dereference step.
    ASSIGN:     ...                 ;proceed with assignment.
```

If N<8 this is condensed to:-

```
                JSP C,routine       ;call special subroutine.
                WD address          ;address of skeleton literal S.

    routine:    SKIPE B,@n(A)       ;deref.arg.N into B unless undef
                JRST CONTINUE       ;  goto CONTINUE.
                MOVEI B,@n(A)       ;load addr.of undef cell into B.
                JRST ASSIGN         ;goto ASSIGN
```

2.21 init(I,J)

Use: The instuction is used (a) following a 'uskel' or 'uskelc', or (b) preceding a 'uskell' which is an argument of a 'uskeld' instruction, or (c) preceding a 'neck'. I to J-1 inclusive are the numbers of global variables having their first occurrences in, respectively, (a) the level 1 skeleton or (b) the level 2 skeleton or (c) the body of the clause concerned. The instuction is omitted if there are no such variables (ie. I=J).

Example: The three different cases are illustrated by the use of 'init(1,2)' for each of:-

```
(a)  concatenate(cons(X,L1),L2,cons(X,L3)) :- concatenate(L1,L2,L3).
                                             *
```

```
(b)  :-mode lookup(?,+,?).
     lookup(X,tree(_,pair(X,Y),_),Y).
                               *
```

```
(c)  member(X,L) :- concatenate(L1,cons(X,L2),L).
                  *
```

Effect: The cells for the global variables I through J-1 are initialised to 'undef'.

DEC10 form:

```
                SETZM n(V1)       ;for each n from I
                ...               ;            to J-1
```

If J-I > 2 this is condensed to:-

```
                MOVEI R1,j(V1)    ;where j=J.
                JSP C,routine

       routine: SETZM -i(R1)      ;where i=I.
                SETZM 1-i(R1)
                ...
                SETZM -1(R1)
                JRST 0(C)
```

2.22 <u>localinit(I,J)</u>

<u>Use</u>: Precedes a 'neck' instruction. The local variables which have their first occurrences within the body of the clause are numbered from I to J-1 inclusive. The instruction is omitted if there are no such variables (ie. I=J). Note if both an 'init' and a 'localinit' precede a 'neck' instruction, the order of the two is not important.

<u>Example</u>: 'localinit(1,2)' for:-

```
     member(X,L) :- concatenate(L1,cons(X,L2),L).
                  *
```

<u>Effect</u>: The cells for the local variables I through J-1 are initialised to '<u>undef</u>'.

<u>DEC10 form</u>:

```
                SETZM n(V)         ;for each n from I+3
                ...                ;            to J+2
```

If J-I > 2 this is condensed to:-

```
                MOVEI R1,j(V)      ;where j=J+3.
                JSP C,routine

     routine:   SETZM -i(R1)       ;where i=I+3.
                SETZM 1-i(R1)
                ...
                SETZM -1(R1)
                JRST 0(C)
```

2.23 ifdone(L)

Use: Precedes the instructions for the arguments of a level 1 skeleton (not occurring in a mode '+' or mode '-' position). L is the address following the last argument instruction.

Example: 'ifdone(label1)' for (cf. Section $$):-

```
     member(X,cons(X,L)).
                    *
```

Effect: If register Y contains 'undef', indicating that the skeleton has matched against a reference, control is transferred to label L, thereby skipping the argument instructions.

DEC10 form:

```
                JUMPE Y,label     ;where label=L.
```

2.24 call(L)

Use: Corresponds to the predicate of a goal in the body of a clause. L is the address of the procedure code for the predicate.

Example: 'call(reverse)' for:-

```
     reverse(cons(X,L1),L2,L3) :- reverse(L1,cons(X,L2),L3).
                                  *******
```

Effect: The address of the outer literals and continuation which follows the 'call' instruction is assigned to register A and control is transferred to L.

DEC10 form:

```
                JSP A,label       ;where label=L.
```

2.25 try(L)

Use: (a) In unindexed procedure code, each clause in the procedure is represented by an instruction 'try(L)' where 'L' is the address of the clause's code. These instructions are ordered as the corresponding clauses in the source program.
(b) The 'try' instruction is also used in indexed procedure code.

Effect: The address of the following instruction is stored in the FL field of the current environment and control is transferred to 'L'. (In our DEC10 implementation, the address is saved in register FL and is only stored in the FL field if and when the 'neck' instruction is reached.)

DEC10 form:

```
        JSP FL,label        ;where label=L.
```

2.26 trylast(L)

Use: (a) In unindexed procedure code, it replaces the 'try(L)' instruction for the last clause in the procedure.
(b) The instruction is also used in indexed procedure code.

Effect: Registers VV and VV1 are reset to the values they held at the time the current goal was invoked. Control is transferred to 'L'.

DEC10 form:

```
        HLRZ VV,0(V)        ;VV:=VV field of current env.
        HLRZ VV1,2(VV)      ;VV1:=V1 field of the VV env.
        JRST label          ;where label=L.
```

2.27 enter

Use: The first instruction in the procedure code for a predicate. It is executed immediately after a 'call' instruction.

Effect: The instruction is responsible for initialising the control information in a new environment. The VV,X,A,V1,TR fields in the local frame are set from the VV,X,A,V1,TR registers. Registers VV and VV1 are then set to the values of registers V and V1 respectively.

DEC10 form:

```
            JSP C,$ENTER

$ENTER:     HRLZM VV,0(V)       ;VV field set.
            HRLI A,(X)
            MOVEM A,1(V)        ;X,A fields set.
            HLRZM TR,R1
            HRLI R1,(V1)
            MOVEM R1,2(V)       ;V1,TR fields set.
            MOVEI VV,(V)        ;VV:=V.
            MOVEI VV1,(V1)      ;VV1:=V1.
            MOVEM TR,$TR0       ;save TR in location $TR0.
            JRST 0(C)           ;return. .
```

2.28 neck(I,J)

Use: Precedes the body of a non-unit clause having I local variables (excluding temporaries) and J global variables.

Example: 'neck(1,1)' for:-

```
     rterm(T,N,N,wd(Atom)) :- flagatom(T,Atom).
                            **
```

Effect: Registers X and X1 are set from registers V and V1 respectively. The contents of registers V and V1 are then incremented by the sizes of (the non-temporary part of) the local frame and of the global frame respectively. Both stacks are checked to ensure a sufficient margin of free space.

DEC10 form:

```
         JSP C,$NECK
         WD i(V)           ;where i=I+3.
         WD j(V1)          ;where j=J.

$NECK:   HRRM FL,0(V)      ;set FL field in local frame.
         MOVEI X,(V)       ;X:=V.
         MOVEI X1,(V1)     ;X1:=V1.
         MOVEI V,@0(C)     ;V:=V+i.
         MOVEI V1,@1(C)    ;V1:=V1+j.
         CAMLE V,$VMAX     ;if insufficient local freespace
         JSP R1,..         ;  call subroutine.
         CAMLE V1,$V1MAX   ;if insufficient global freespace
         JSP R1,...        ;  call subroutine.
         JRST 2(C)         ;return to in-line code.
```

If J=0 this is condensed to:-

```
         JSP C,$NECK1
         WD i(V)

$NECK1:  HRRM FL,0(V)
         MOVEI X,(V)
         MOVEI V,@0(C)
         CAMLE V,$VMAX
         JSP R1,...
         JRST 1(C)
```

If J=0 and I<5 this is further condensed to:-

```
         JSP C,routine

routine: HRRM FL,0(V)
         MOVEI X,(V)
         MOVEI V,i(V)      ;where i=I+3.
         CAMLE V,$VMAX
         JSP R1,...
         JRST 0(C)
```

2.29 foot(N)

Use: At the end of a non-unit clause for a predicate of arity N.

Example: 'foot(3)' for:-

```
    concatenate(cons(X,L1),L2,cons(X,L3)) :- concatenate(L1,L2,L3).
                                                                  *
```

Effect: If the register VV indicates a point on the local stack earlier than register X, V is assigned the contents of X. Thus a determinate exit from the current procedure results in all the local storage used during the process being recovered. A, X and X1 are reset from the corresponding field in the parent local frame pointed to by X, and control is transferred to the parent's continuation.

DEC10 form:

```
            JSP C,$FOOT
            WD n(A)           ;where n=N.

   $FOOT:   CAILE X,(VV)      ;if X>VV
            MOVEI V,(X)       ;  then V:=X.
            MOVE A,1(V)       ;reset A from parent.
            HLRZM A,X         ;reset X from parent.
            HLRZ X1,2(X)      ;reset X1 from parent.
            JRST @0(C)        ;goto parent's continuation.
```

If N<9 this is condensed to:-

```
            JRST routine

   routine: CAILE X,(VV)
            MOVEI V,(X)
            MOVE A,1(X)
            HLRZM A,X
            HLRZ X1,2(X)
            JRST n(A)         ;where n=N.
```

2.30 neckfoot(J,N)

Use: The last instruction for a unit clause. It replaces a 'neck(0,J)' followed by a 'foot(N)' where 'J' is the number of global variables and N is the arity of the predicate of the clause. (Note that a unit clause has no non-temporary local variables.)

Example: 'neckfoot(0,3)' for:-

```
     concatenate(nil,L,L).
                         *
```

Effect: The instruction combines the effect of the 'neck' and 'foot' instructions it replaces. However considerable computation is saved; registers A, X and X1 do not have to be modified. Registers V1 and (if a non-determinate exit) V are incremented to take account of the new global and local frames, and control is transferred to the parent's continuation.

DEC10 form:

```
              MOVEI V1,j(V1)    ;where j=J.
              JSP C,$NKFT
              WD n(A)           ;where n=N.

     $NKFT:   CAILE V,(VV)      ;if V>VV (ie. determinate exit)
              JRST BELOW        ;  then skip to BELOW.
              HRRM FL,0(V)      ;set FL field in local frame.
              MOVEI V,3(V)      ;V:=V+3.
              CAMLE V,$VMAX     ;if insufficient local freespace
              JSP R1,...        ;  call subroutine.
     BELOW:   CAMLE V1,$V1MAX   ;if insufficient global freespace
              JSP R1,...        ;  call subroutine.
              JRST @0(C)        ;goto parent's continuation.
```

If J=0 this is condensed to:-

```
              CAIG V,(VV)       ;if V =< VV (ie. non-determinate)
              JSP C,$NKFT0      ;  then call subroutine $NKFT0
              JRST n(A)         ;  else goto parent's continuation.

     $NKFT0:  HRRM FL,0(V)
              MOVEI V,3(V)
              CAMLE V,$VMAX
              JSP R1,...
              JRST @0(C)
```

If J=0 and N<9 this is further condensed to:-

```
              JRST routine

     routine: CAIG V,(VV)
              JSP C,$NKFT0
              JRST n(A)
```

2.31 cut(I)

Use: Corresponds to an occurrence of the cut symbol. I is the number of local variables (excluding temporaries) in the clause, as for the instruction 'neck(I,J)'.

Example: 'cut(2)' for:-

```
compile(S,C) :- translate(S,C),!,assemble(C).
                                *
```

Effect: Any remaining local frames created since the environment of the current clause are discarded by resetting register V to point at the end of the current local frame. Registers VV and VV1 are reset to the backtrack environment of the parent. The portion of the trail created since the parent goal is "tidied up" by discarding references to variables if they don't belong to environments before the backtrack environment.

DEC10 form:

```
                  MOVEI V,i(V)        ;V:=V+i where i=I+3.
                  JSP C,$CUT

        $CUT:     CAILE X,(VV)        ;if no alternatives to cut
                  JRST 0(C)           ;  then return.
                  HLRZ VV,0(X)        ;reset VV from parent.
                  HLRZ VV1,2(VV)      ;reset VV1 from parent.
                  HRRE P,2(X)         ;P:= lh of TR for parent.
                  ADD P,$TRTOP        ;P:= rh of TR for parent.
                  CAIN P,(TR)         ;if no change to TR
                  JRST 0(C)           ;  then return.
                  MOVEI P1,(TR)       ;P1:=rh of TR for parent.
                  MOVEI R1,(TR)       ;
                  SUBI R1,(P)         ;R1:=delta=increase in trail size.
                  HRLI R1,(R1)        ;reset TR to its original value:-
                  SUB TR,R1           ;  TR:=TR-(delta,delta).
        CYCLE:    MOVE R1,1(P)        ;load one of the new trail entries.
                  CAIL R1,(VV)        ;if refers after VV
                  JRST CONTINUE       ;  then continue with next entry.
                  CAIGE R1,(V1)       ;if refers after V1 (ie. is local)
                  CAIGE R1,(VV1)      ;or before VV1
                  PUSH TR,(R1)        ;  then restore it to trail.
        CONTINUE: CAIE P1,1(P)        ;if more trail entries to consider
                  AOJA P,CYCLE        ;  then P:=P+1, goto CYCLE.
                  JRST 0(C)           ;return.
```

If I<10 this is condensed to:-

```
                  JSP C,routine

        routine:  MOVEI V,i(V)
                  JRST $CUT
```

2.32 neckcut(I,J)

Use: Corresponds to a cut symbol which is the first "goal" in the body of a non-unit clause. It replaces a 'neck(I,J)' followed by a 'cut(I)' where 'I' and 'J' are the numbers of local and global variables respectively.

Example: 'neckcut(0,0)' for:-

```
     divide(X,0,Y) :-!, error('division by 0').
                     ***
```

Effect: The instruction combines the effects of the corresponding 'neck' and 'cut' instructions in a straightforward way.

DEC10 form:

```
                JSP C,$NCUT
                WD i(V)          ;where i=I+3.
                WD j(V1)         ;where j=J.
```

If J=0 this is condensed to:-

```
                JSP C,$NCUT1
                WD i(V)
```

If J=0 and I<5 this is further condensed to:-

```
                JSP C,routine

      routine:  MOVEI X,(V)
                MOVEI V,i(V)     ;where i=I+3.
                JRST ...
```

2.33 neckcutfoot(J,N)

Use: Corresponds to a cut symbol which is the only "goal" in the body of a clause. It replaces instructions 'neck(0,J)' followed by 'cut(0)' followed by 'foot(N)' where 'J' is the number of global variables and N is the arity of the predicate of the clause.

Example: 'neckcutfoot(0,2)' for:-

```
     factorial(0,1):-!.
                     ****
```

Effect: Combines the effect of the three instructions it replaces. As with 'neckfoot', considerable computation is saved since registers A, X and X1 do not have to be modified. Register V1 is incremented to take account of the new global frame, registers VV and VV1 are reset to their states prior to invoking the parent goal and trail entries are discarded where possible. Finally control is transferred to the parent's continuation.

DEC10 form:

```
                 MOVEI V1,j(V1)     ;where j=J.
                 JSP C,$NCTF
                 WD n(A)            ;where n=N.

     $NCTF:      CAMLE V1,$V1MAX    ;if insufficient local freespace
                 JSP R1,...         ;  call subroutine.
                 CAILE V,(VV)       ;if V>VV (already determinate)
                 JRST @0(C)         ;  then goto parent's continuation.
     $NCTF0:     HLRZ VV,0(V)       ;reset VV.
                 HLRZ VV1,2(VV)     ;reset VV1.
                 .                  ;
                 .                  ;perform rest of cut.
                 .                  ;
                 JRST @0(C)         ;goto parent's continuation.
```

If J=0 this is condensed to:-

```
                 CAIG V,(VV)        ;if V =< VV (not already determinate)
                 JSP C,$NCTF0       ;  then call subroutine $NCTF0.
                 JRST n(A)          ;goto parent's continuation.
```

2.34 fail

Use: Corresponds to a goal 'fail' in the body of a clause. This goal is defined to be unsolvable and instigates (deep) backtracking.

Example: 'fail' for:-

```
    unknown(X) :- known(X),!,fail.
                             ****
```

Effect: Registers V, V1, A, X and X1 are reset to the values they had prior to the most recent goal which is non-determinate (ie. one for which there are still further choices available). Register TR is also restored to its earlier value by popping entries off the trail and resetting the cells referenced to 'undef'. Finally control is transferred to the clause which is the next choice for the earlier goal.

DEC10 form:

```
                JRST $EFAIL

      $EFAIL:   MOVEI V,(VV)       ;V:=VV
                MOVEI V1,(VV1)     ;V1:=VV1
                HRRZ FL,0(V)       ;FL:=next clause for earlier goal.
                MOVE A,1(V)        ;reset A
                HLRZM A,X          ;reset X
                HLRZ X1,2(X)       ;reset X1
                HRRE R1,2(V)       ;R1:=lh of earlier TR.
                ADD R1,$TRTOP      ;R1:=rh of earlier TR.
                CAIN R1,(TR)       ;if no change to TR
                JRST EXIT          ;  then goto EXIT.
      CYCLE:    POP TR,R2          ;pop an entry off the trail.
                SETZM (R2)         ;set cell refd. to 'undef'.
                CAIE R1,(TR)       ;if more trail entries to consider
                JRST CYCLE         ;  then goto CYCLE.
      EXIT:     MOVEM TR,$TRO      ;save TR in location $TRO
                JRST @FL           ;goto next clause.
```

Note in passing that a failure in a unification instruction causes control to be transferred to a routine $FAIL which instigates shallow backtracking:-

```
      $FAIL:    CAIE V,(VV)        ;if V = VV (no other choices)
                JRST $EFAIL        ;  then deep backtracking.
                CAMN TR,$TRO       ;if no trail entries from this unifn.,
                JRST @FL           ;  then goto next clause.
      CYCLE1:   POP TR,R2          ;pop an entry off the trail.
                SETZM (R2)         ;set the cell refd. to 'undef'.
                CAME TR,$TRO       ;if more trail entries to consider
                JRST CYCLE1        ;  then goto CYCLE1.
                JRST @FL           ;goto next clause.
```

2.35 gsect

Use: Precedes a general section of clauses having a variable at position 0 in the head.

Effect: The outer literal representing argument 0 of the current goal is accessed via register A and the dereferenced result is assigned to cell 0 in the current local frame.

DEC10 form:

```
                JSP C,$GS

     $GS:       MOVE B,@0(A)        ;B := arg. 0
                HLRZM B,Y           ;Y := type of arg. 0
                CAIG Y,$SKEL        ;if arg. 0 is not a molecule
                JRST @table(Y)      ;  then switch on type.
                MOVEM B,3(V)        ;local cell 0 := arg. 0
                JRST 0(C)           ;return.
```

In practice the code is optimised by
(1) coalescing the code for ′enter′ immediately followed by ′gsect′,
(2) ′ssect′ initialises local cell 0 as a side effect so that ′gsect′ doesn′t have to be called if no clauses in the special section are entered,
(3) ′endssect′ performs the work of ′gsect′ if the matching term is a reference so ′gsect′ only needs to handle the non-reference case.

2.36 ssect(L,C)

Use: Precedes a special section of clauses having a non-variable at position 0 in the head. L is the address of the reference code for the section and C is the address of the section which follows.

Effect: If the dereferenced value of argument 0 in the current goal is a reference, control is transferred to L. Otherwise register FL is set to C and control passes to the non-reference code which follows the 'ssect' instruction.

DEC10 form:

```
                JSP C,$SS
                WD refcode          ;where refcode=L
                WD nextsection      ;where nextsection=C

      $SS:      MOVE B,3(V)         ;B := arg.0 from local cell 0
                HLRZM B,Y           ;Y := type of arg. 0
                JUMPE Y,@0(C)       ;if arg.0 is a ref goto refcode.
                MOVE FL,1(C)        ;FL := nextsection
                MOVEM B,R2          ;R2 := arg. 0
                CAIL Y,$MOLS        ;if arg.0 is a molecule
                MOVE R2,0(B)        ;  then R2 := functor of arg.0
                JRST 2(C)           ;return to non-reference code.
```

The above is an optimisation, used only if it is not the first section in the procedure. 'enter' immediately followed by 'ssect' is treated as a special case. Register R2 is set to the address of the atom, integer or functor literal for argument 0. If argument 0 is a reference, this is trailed once and for all to avoid repeated "trailing" for each of the clauses in the section.

2.37 ssectlast(L)

Use: Precedes a special section which is the last section of a procedure. L is the address of the reference code for the section.

Effect: If the dereferenced value of argument 0 in the current goal is a reference, control is transferred to L. Otherwise registers VV and VV1 are reset to the values they held at the time the current goal was invoked and control passes to the reference code which follows the 'ssectlast' instruction.

DEC10 form:

```
                JSP C,$SS1
                WD refcode          ;where refcode=L

     $SS1:      MOVE B,3(V)         ;B := arg.0 from local cell 0.
                HLRZM B,Y           ;Y := type of arg.0
                JUMPE Y,@0(C)       ;if arg.0 is a ref. goto refcode.
                HLRZ VV,0(V)        ;VV := VV field of current env.
                HLRZ VV1,2(VV)      ;VV1 := V1 field of the VV env.
                MOVEM B,R2          ;R2 := arg.0
                CAIL Y,$MOLS        ;if arg.0 is a molecule
                MOVE R2,0(B)        ;  then R2 := functor of arg.0.
                JRST 1(C)           ;return to non-reference code.
```

2.38 endssect

Use: Terminates the reference code at the end of a special section.

Effect: The reference passed as argument 0 is recovered from the trail and stored in local cell 0. The following 'gsect' instruction is skipped.

DEC10 form:

```
                JSP C,$ENDRC

     $ENDRC:    POP TR,R1           ;pop last trail entry into R1.
                MOVEM TR,$TR0       ;TR0 := TR.
                SOS 2(V)            ;correct TR field of current env.
                SETZM (R1)          ;set cell referenced to undef.
                MOVEM R1,3(V)       ;local cell 0 := the reference.
                JRST 1(C)           ;return, skipping one instruction.
```

2.39 switch(N)

Use: Precedes the non-reference code in a special section if there is a sufficient number of clauses in the section (currently 5 or more). N is the number of 'case' instructions which follow and is a power of 2 chosen depending on the number of clauses in the section.

Effect: A key, determined by the principal functor of argument 0 of the current goal, is "anded" with N-1 to give a value M. Control is then transferred to the (M+1)th. 'case' instruction.

DEC10 form:

```
                MOVEI R1,(R2)      ;R1 := key
                ANDI R1,n-1        ;R1 := key/(N-1)
                JRST @NEXT(R1)     ;goto case (R1)
     NEXT:
```

2.40 case(L)

Use: A 'switch(N)' instruction is followed by N 'case' instructions. The parameter L is the address of the code for the subset of the section's clauses corresponding to that case.

Effect: Control is transferred to address L by the preceding 'switch' instruction.

DEC10 form:

```
                WD label           ;where label=L.
```

2.41 ifatom

Use: In the non-reference code of a special section, the clause(s) for the atom identified by I is indicated by address L.

Effect: If argument 0 of the current goal is atom I, control is transferred to address L.

DEC10 form:

```
        CAMN R2,atom        ;where atom = addr. of atom I literal.
        JRST label          ;where label=L.
```

2.42 ifint(I,L)

Use: In the non-reference code of a special section, the clause(s) for integer I is indicated by address L.

Effect: If argument 0 of the current goal is integer I, control is transferred to address L.

DEC10 form:

```
        CAMN R2,int         ;where int = integer I literal
        JRST label          ;where label=L.
```

2.43 iffn(I,L)

Use: In the non-reference code of a special section, the clause(s) for the functor identified by I is indicated by address L.

Effect: If the principal functor of argument 0 of the current goal is functor I, registers B and Y are set according to this molecule and control is transferred to address L.

DEC10 form:

```
            CAMN R2,functor  ;where functor = addr of fn I.literal.
            JRST label       ;where label=L.
```

Note that in the actual implementation, registers B and Y are set by 'ssect' or else by the following preceding the CAMN:-

```
            JSP C,$RLDSK

   $RLDSK:  MOVE B,@0(A)     ;B := arg.0
            HLRZ Y,B         ;Y := type of arg.0
            CAIL Y,MOLS      ;if arg.0 is a molecule
            JRST 0(C)        ;  then return.
            JUMPE Y,DEREF    ;if arg.0 is a ref. goto DEREF.
            MOVEI B,@0(A)    ;B := skel. literal in the goal.
            HRLI B,(X1)      ;lh. of B := X1.
            MOVEI Y,(X1)     ;Y := X1.
            JRST 0(C)        ;return.

   DEREF:   MOVE B,0(B)      ;B := deref B.
            HLRZ Y,B         ;Y := type of B.
            JUMPE Y,DEREF    ;if B is a ref. goto DEREF.
            JRST 0(C)        ;return
```

2.44 goto(L)

Use: (1) Following a sequence of 'if' instructions or a sequence of 'try' instructions in a special section, L is the address of the following section.

Effect: Control is transferred to address L.

DEC10 form:

```
                JRST label        ;where label=L.
```

2.45 notlast

Use: If there is more than one clause for a particular functor in a special section, the 'try' instructions are preceded by a 'notlast' instruction.

Effect: Registers VV and VV1 are reset from V and V1 respectively to indicate the current environment.

DEC10 form:

```
                JSP C,$NLAST

       $NLAST:  MOVEI VV,(V)      ;VV:=V
                MOVEI VV1,(V1)    ;VV1:=V1
                MOVEM             TR,$TR0 ;TR0:=TR
                JRST              0(C) ;return
```

2.46 ugvar(I)

Use: If argument 0 of the head of a clause is a global variable (and the procedure code is to be indexed), this term is represented by 'ugvar(I)' where I is the number of the global variable.

Effect: The construct which has been assigned to local cell 0 (by the corresponding 'gsect' instruction) is also assigned to global cell I unless the construct is a local reference. In the latter case global cell I is initialised to undef and a reference to global cell I is assigned to the local reference. This assignment is trailed if necessary.

DEC10 form:

```
                JSP C,$GTER1
                MOVEM T,i(V1)       ;global cell I := T

     $GTER1:    MOVE T,3(V)         ;T := local cell 0
                CAIG T,MAXREF       ;if T not a reference
                CAIGE T,@0(C)       ;or T < global cell I
                JRST 0(C)           ;  then return.
                MOVEI R1,@0(C)      ;R1 := addr. of global cell I
                SETZM (R1)          ;global cell I := undef
                MOVEM R1,(T)        ;cell T := R1
                CAIGE T,(VV)        ;if T < VV
                PUSH TR,T           ;  then push T onto the trail
                JRST 1(C)           ;return, skipping 1 instr.
```

2.47 tryatom(I,C)

Use: In the reference code of a special section, a clause with atom I as argument 0 of the head is represented by the instruction 'tryatom(I,C)'. C is the address of the clause's code.

Effect: The atom is assigned to the matching reference and the assignment is trailed. Register FL is set to the address of the following instruction and control is transferred to C.

DEC10 form:

```
                JSP C,$RVAT
                XWD clause,atom   ;clause=C,atom= atom I.

     $RVAT:     MOVEI FL,1(C)     ;FL := next PLM instr.
                MOVE R1,0(C)      ;R1 := (clause,atom)
                HLRZM R1,C        ;C := clause.
                MOVE R1,0(R1)     ;R1 := atom.
                MOVEM R1,@0(TR)   ;trailed ref. := atom.
                JRST 0(C)         ;goto clause.
```

Note that the matching reference has already been trailed by 'ssect'.

2.48 tryint(I,C)

Exactly analagous to 'tryatom'. The DEC10 form uses routine $RVAT also.

2.49 tryskel(S,C)

Use: In the reference code of a special section, a clause with a skeleton as argument 0 in the head is represented by 'tryskel(S,C)'. S is the address of the skeleton literal and C of the clause's code.

Effect: A molecule is formed from S with current global frame address from register V1 and assigned to the matching reference. The assignment is trailed. Register Y is set to 'undef' and register FL to the address of the following instruction. Control is transferred to C.

DEC10 form:

```
                JSP C,$RVSK
                XWD clause,skeleton ;clause=C,skeleton=S.

      $RVSK:    MOVEI Y,0         ;Y := undef.
                MOVEI FL,1(C)     ;FL := next PLM instr.
                MOVE R1,0(C)      ;R1 := (clause,skeleton).
                HLRZM R1,C        ;C := clause.
                HRLI R1,(V1)      ;R1 := (V1,skeleton).
                MOVEM R1,@0(TR)   ;trailed ref. := (V1,skel.).
                JRST 0(C)         ;goto clause.
```

Note that the matching reference has already been trailed by 'ssect'.

2.50 trylastatom(I,C)

Use: If the final clause in a procedure has atom I as argument 0 of its head, the clause is represented in the reference code by the instruction 'trylastatom(I,C)', where C is the address of the clause's code. No 'endssect' is needed at the end of the section.

Effect: The atom is assigned to the matching reference but the assignment need not be trailed. Registers VV and VV1 are reset to indicate the previous backtrack point. Control is transferred to C.

DEC10 form:

```
                JSP C,$RVAT1
                XWD clause,atom   ;clause=C, atom = atom I.

     $RVAT1:    HLRZ VV,0(V)      ;VV := VV field of current env.
                HLRZ VV1,2(VV)    ;VV1 := V1 field of the VV env.
                MOVE R1,0(C)      ;R1 := (clause,atom).
                HLRZM R1,C        ;C := clause.
                MOVE R1,0(R1)     ;R1 := atom.
                MOVEM R1,@0(TR)   ;trailed ref. := atom.
                JRST 0(C)         ;goto clause.
```

Note that the matching reference has already been trailed by 'ssect'.

2.51 trylastint(I,C)

Exactly analogous to 'trylastatom'. The DEC10 form uses routine $RVAT1 also.

2.52 trylastskel(S,C)

Use: If the final clause in a procedure has a skeleton as argument 0 of its head, the clause is represented in the reference code by an instruction 'trylastskel(S,C)', where S is the address of the skeleton literal and C is the address of the clause's code. No 'endssect' is needed at the end of the section.

Effect: A molecule is formed from S with the current global frame address from register V1 and assigned to the matching reference. The assignment need not be trailed. Register Y is set to 'undef'. Registers VV and VV1 are reset to indicate the previous backtrack point. Control is then transferred to C.

DEC10 form:

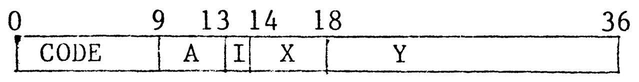
```
                JSP C,$RVSK1
                XWD clause,skeleton ;clause=C,skeleton=S.

      $RVSK1:   MOVEI Y,0          ;Y := undef.
                HLRZ VV,0(V)       ;VV := VV field of current env.
                HLRZ VV1,2(VV)     ;VV1 := V1 field of the VV env.
                MOVE R1,0(C)       ;R1 := (clause,skeleton).
                HLRZM R1,C         ;C := clause.
                HRLI R1,(V1)       ;R1 := (V1,skeleton).
                MOVEM R1,@0(TR)    ;trailed ref. := (V1,skeleton).
                JRST 0(C)          ;goto clause.
```

Note that the matching reference has already been trailed by 'ssect'.

3.0 <u>SYNOPSIS OF THE DEC SYSTEM 10</u>

The machine has 36 bit words which can accommodate two 18 bit addresses. Addresses 0 to 15 refer to fast registers which are used as accumulators and (for 1 to 15 only) as index registers. Signed integers are represented as "2s complement" bit patterns. The instruction format is:-



where	CODE	=	instruction code,
	A	=	accumulator address,
	I	=	indirection bit,
	X	=	index register address,
	Y	=	main address.

An instruction with I=1 is written symbolically in the form:-

```
CODE A,@Y(X)
```

If I=0 the '@' is omitted. If A=0 it can be omitted along with the comma. If X=0 it can be omitted along with the brackets. If Y=0 it can be omitted.

A fundamental mechanism is the "<u>effective address calculation</u>" which is the first step in the execution of each and every instruction. It computes an effective address E depending on I, X and Y. If X is nonzero, the contents of index register X is added to Y to produce a modified address M (modulo 2 to the power 18). If I=0 then simply E=M. If I=1, the addressing is <u>indirect</u> and E is derived by treating the I, X and Y fields of the word stored at M in exactly the same way. The process continues until a referenced location has I=0 and then E is calculated according to the X and Y fields of this location.

Two instructions, PUSH and POP, access pushdown lists which are stored in main memory. A pushdown list is referenced via a pushdown list pointer held in an accumulator. The right half of this word is the address of the current last item in the list. The left half (normally) contains the negative quantity M-L where M is the maximum size of the list and L is the current size.

The instructions referred to in this paper are summarised below. In all cases A is the accumulator address and E is the effective address computed as above. We write:-

(X)	for "the contents of location X",
X.L	for "the left half of location X"
X.R	for "the right half of location X",
(X,Y)	for "the word with left half X and right half Y",
sign X	for "-1 if the top bit of X is 1 or 0 otherwise",
Y:=X	for "location Y is assigned the value X".
<u>skip</u>	for "skip the next instruction"

<u>Instruction</u>	<u>Effect</u>
MOVE A,E	A:=(E)
MOVEI A,E	A:=E
MOVEM A,E	E:=(A)
SETZM E	E:=0
ADD A,E	A:=(A)+(E)
SUB A,E	A:=(A)-(E)
SUBI A,E	A:=(A)-E
AOS E	E:=(E)+1
SOS E	E:=(E)-1
HLRZ A,E	A:=(0,(E.L))
HRRZ A,E	A:=(0,(E.R))
HLRZM A,E	E:=(0,(A.L))
HRLZM A,E	E:=((A.R),0)
HRLI A,E	A.L:=E
HRRM A,E	E.R:=(A.R)
HRRE A,E	A:=(sign (E.R), E.R)
CAIE A,E	if (A) = (0,E) then <u>skip</u>
CAIN A,E	if (A) = (0,E) then <u>skip</u>
CAME A,E	if (A) = (E) then <u>skip</u>
CAMN A,E	if (A) = (E) then <u>skip</u>
CAIG A,E	if (A) > (0,E) then <u>skip</u>
CAILE A,E	if (A) =< (0,E) then <u>skip</u>
CAIGE A,E	if (A) >= (0,E) then <u>skip</u>
CAMLE A,E	if (A) =< (E) then <u>skip</u>

SKIPE A,E	if A = 0 then A:=(E), if (E)=0 then skip
SKIPN A,E	if A = 0 then A:=(E), if (E)=0 then $skip
TLNN A,E	if (A.L)/e = 0 then skip
JRST E	goto E
JSP A,E	A:=(flags,address of next instruction), goto E
JUMPE A,E	if (A)=0 then goto E
AOJA A,E	A:=(A)+1, goto E
PUSH A,E	A:=(A)+(1,1); (A.R):=(E); if (A.L)=0 then interrupt
POP A,E	E:=((A.R)); A:=(A)-(1,1); if (A.L)=0 then interrupt
WD E	a non-executable address word with CODE=0
XWD X,Y	a non-executable data word containing (X,Y)

Constant Symbols

$VOID=1
$SKEL=2
$ATOM=4
$INT=5
$MOLS=16
$MAXREF=777777base8
$1MA=777764base8
$1MAS=777766base8

4.0 TIMING DATA FOR PLM INSTRUCTIONS ON DEC10

The times given below are the minimum times to complete the PLM instruction successfully. Cases where a failure to match occurs are not counted. Certain infrequent but faster special cases are also discounted (for example matching against a void).

The times relate to a KI10 processor and have been calculated from the data given on pages D-4 and D-5 of [DEC 1974]. An extra 1.02 microseconds has been allowed for each indirection and 0.89 microseconds for a control transfer or test instruction. All other factors (such as indexing) have been ignored.

Instruction	microsecs.	Remarks
uvar,uvarl	4.85	v. atom, integer or molecule
uref,urefl	15.88	undef v. molecule
uatom,uatoml, uint,uintl	8.68	v. atom or integer
uskel	12.22	v. molecule
uskell	26.49 +	28.26 per argument v. molecule, with mol. v. ref. for each arg.
uskeld	11.20	v.molecule
uskelc	15.60	v. reference
init,localinit	.95	per cell initialised
ifdone	1.45	
call	1.34	
try	1.34	
trylast	3.75	
enter	9.67	
neck	12.77	general case
foot	7.55	
neckfoot	2.74	no globals, determinate exit
cut	14.32 +	9.24 per trail entry examined general case, assumes each trail entry retained
neckcut	15.53 +	9.24 per trail entry examined no globals
neckcutfoot	12.10 +	9.24 per trail entry examined no globals
fail	13.14 +	5.85 per trail entry examined
"shallow" fail	6.08 +	6.66 per trail entry examined

5.0 BENCHMARK TESTS

Times in milliseconds

Procedure	Data	Prolog-10	Lisp	Pop-2	Prolog-M	Prolog-10I
nreverse	list30	53.7	34.6	203	1156	1160
qsort	list50	75.0	43.8	134	1272	1344
deriv	times10	3.00	5.21	11.2	86.4	76.2
	divide10	2.94	7.71	15.9	90.6	84.4
	log10	1.92	2.19	8.56	61.6	49.2
	ops8	2.24	2.94	5.25	61.2	63.7
serialise	palin25	40.2	19.76	-	711	602
dbquery	-	185	-	300	9970	8888

Time ratios

Procedure	Data	Prolog-10	Lisp	Pop-2	Prolog-M	Prolog-10I
nreverse	list30	1	.64	3.8	22	22
qsort	list50	1	.58	1.8	17	18
deriv	times10	1	1.7	3.7	29	25
	divide10	1	2.6	5.4	31	29
	log10	1	1.1	4.5	32	26
	ops8	1	1.3	2.3	27	28
serialise	palin25	1	.49	-	18	15
dbquery	-	1	-	1.6	54	48

Notes

The above table, giving average figures for actual CPU time on a DECsystem-10 (KI processor), compares compiled Prolog (our implementation, "Prolog-10"), compiled Lisp (Stanford with the NOUUO option), compiled Pop-2 and interpreted Prolog (both the Marseille

Fortran implementation, "Prolog-M", and our implementation in Prolog, "Prolog-10I"). The data was obtained by timing (via "control-T") a large number of iterations of each test. The figures include garbage collection times for Lisp and Pop-2. No garbage collection was needed for Prolog since the stack mechanism recovers storage after each iteration. Test iterations were achieved in the following ways:-

Prolog

```
tests(N) :- read(_),from(1,N,I),test,fail.
tests(N) :- read(_),test.

from(I,I,I):-!.
from(L,N,I) :- N1 is (L+N)/2, from(L,N1,I).
from(L,N,I) :- L1 is (L+N)/2+1, from(L1,N,I).
```

Lisp

```
(DEFPROP TESTS (LAMBDA (N)
   (PROG (I RESULT)
      (READ)
      (SETQ I 0)
LAB   (SETQ RESULT TEST)
      (COND (LESSP I N) (GO LAB))
      (READ)
      (RETURN RESULT)))
EXPR)
```

Pop-2

```
FUNCTION TESTS N;
  VARS I RESULT;
   ERASE(ITEMREAD());
   FORALL I 1 1 N;
      TEST -> RESULT
   CLOSE;
   ERASE(ITEMREAD());
   RESULT
END
```

The dummy "reads" serve to interrupt the execution of each test so that "control-T" timings can be taken. The Prolog form of each benchmark test is listed below, together with the Lisp and Pop-2 versions selected for comparison. Note that in the Prolog examples a more convenient syntactic form is used for lists. Thus '[]' stands for the empty list and '[X,..L]' denotes a list whose head is X and tail is L. A list of two elements 'a' followed by 'b' is written '[a,b]'. Apart from the syntax, such lists are treated no differently from other terms. The timing data would be exactly the same if, say, 'nil' and 'cons(X,L)' were used.

5.1 reverse

```
list30 = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,
                                  21,22,23,24,25,26,27,28,29,30]
```

Prolog : nreverse(list30,X)

```
:-mode nreverse(+,-).
:-mode concatenate(+,+,-).

nreverse([X,..L0],L) :- nreverse(L0,L1), concatenate(L1,[X],L).
nreverse([],[]).

concatenate([X,..L1],L2,[X,..L3]) :- concatenate(L1,L2,L3).
concatenate([],L,L).
```

Lisp : (NREVERSE list30)

```
(DEFPROP NREVERSE (LAMBDA (L)
   (COND ((NULL L) NIL)
      (T (CONCATENATE (NREVERSE (CDR L)) (CONS (CAR L) NIL)))))
EXPR)

(DEFPROP CONCATENATE (LAMBDA (L1 L2)
   (COND ((NULL L1) L2)
      (T (CONS (CAR L1) (CONCATENATE (CDR L1 L2)))))
EXPR)
```

Pop-2 : NREVERSE(list30)

```
FUNCTION NREVERSE LIST;
   IF NULL(LIST) THEN NIL
   ELSE CONCATENATE(NREVERSE(TL(LIST)),HD(LIST)::NIL)
   CLOSE
END;

FUNCTION CONCATENATE LIST1 LIST2;
   IF NULL(LIST1) THEN LIST2
   ELSE HD(LIST1)::CONCATENATE(TL(LIST1),LIST2)
   CLOSE
END;
```

5.2 qsort

```
list50 = [27,74,17,33,94,18,46,83,65, 2,
          32,53,28,85,99,47,28,82, 6,11,
          55,29,39,81,90,37,10, 0,66,51,
           7,21,85,27,31,63,75, 4,95,99,
          11,28,61,74,18,92,40,53,59, 8]
```

Prolog : qsort(list50,X,[])

```
:-mode qsort(+,-,+).
:-mode partition(+,+,-,-).

qsort([X,..L],R,R0) :-
   partition(L,X,L1,L2),
   qsort(L2,R1,R0),
   qsort(L1,R,[X,..R1]).
qsort([],R,R).

partition([X,..L],Y,[X,..L1],L2) :- X =< Y, !,
   partition(L,Y,L1,L2).
partition([X,..L],Y,L1,[X,..L2]) :-
   partition(L,Y,L1,L2).
partition([],_,[],[]).
```

Lisp : (QSORT list50 NIL)

```
(DEFPROP QSORT (LAMBDA (L R)
   (COND ((NULL L) R)
      (T ((LAMBDA (P)
           (QSORT (CAR P) (CONS (CAR L) (QSORT (CDR P) R))))
          (PARTITION (CDR L) (CAR L))))))
EXPR)

(DEFPROP PARTITION (LAMBDA (L X)
   (COND ((NULL L) (CONS NIL NIL))
      (T ((LAMBDA (P)
          (COND ((LESSP (CAR L) X)
                 (CONS (CONS (CAR L) (CAR P)) (CDR P)))
             (T (CONS (CAR P) (CONS (CAR L) (CDR P))))))
          (PARTITION (CDR L) X)))))
EXPR)
```

Pop-2 : QSORT(list50)

```
FUNCTION QSORT LIST;
VARS Y Z Q QQV QQW QQS;
   0;
L2:IF NULL(LIST) OR NULL(TL(LIST)) THEN GOTO SPLIT CLOSE;
   NIL->QQS; NIL->Y; NIL->Z;
   HD(LIST)->QQW;
L1:HD(LIST)->QQV; TL(LIST)->LIST;
   IF QQW>QQV THEN QQV::QQS->QQS
   ELSEIF QQW<QQV THEN QQV::Z->Z
   ELSE QQV::Y->Y
   CLOSE;
   IF NULL(LIST) THEN Z;Y;1; QQS->LIST; GOTO L2 ELSE GOTO L1 CLOSE;
SPLIT: ->Q; IF Q=0 THEN LIST EXIT
   ->Y;
   IF Q=1 THEN ->Z; LIST<>Y;2; Z->LIST; GOTO L2 CLOSE;
   Y<>LIST->LIST;
   GOTO SPLIT
END;
```

5.3 deriv

```
times10 = ((((((((x*x)*x)*x)*x)*x)*x)*x)*x)*x
divide10 = ((((((((x/x)/x)/x)/x)/x)/x)/x)/x)/x
log10 = log(log(log(log(log(log(log(log(log(log(x))))))))))
ops8 = (x+1)*(x~2+2)*(x~3+3)
```

Prolog : d(expr,x,Y)

```
:-mode d(+,+,-).
:-op(300,xfy,~).

d(U+V,X,DU+DV) :-!, d(U,X,DU),d(V,X,DV).
d(U-V,X,DU-DV) :-!, d(U,X,DU),d(V,X,DV).
d(U*V,X,DU*V+U*DV) :-!, d(U,X,DU),d(V,X,DV).
d(U/V,X,(DU*V-U*DV)/V~2) :-!, d(U,X,DU),d(V,X,DV).
d(U~N,X,DU*N*U~N1) :-!, integer(N), N1 is N-1, d(U,X,DU).
d(-U,X,-DU) :-!, d(U,X,DU).
d(exp(U),X,exp(U)*DU) :-!, d(U,X,DU).
d(log(U),X,DU/U) :-!, d(U,X,DU).
d(X,X,1):-!.
d(C,X,0).
```

Lisp : (DERIV expr (QUOTE X))

```
(DEFPROP DERIV (LAMBDA (E X)
   (COND ((ATOM E) (COND ((EQ E X) 1) (T 0)))
      ((OR (EQ (CAR E) (QUOTE PLUS)) (EQ (CAR E) (QUOTE DIFFERENCE)))
         (LIST (CAR E) (DERIV (CADR E) X) (DERIV (CADDR E) X)))
      ((EQ (CAR E) (QUOTE TIMES))
         (LIST (QUOTE PLUS)
            (LIST (CAR E) (CADDR E) (DERIV (CADR E) X))
            (LIST (CAR E) (CADR E) (DERIV (CADDR E) X))))
      ((EQ (CAR E) (QUOTE QUOTIENT))
         (LIST (CAR E)
            (LIST (QUOTE DIFFERENCE)
               (LIST (QUOTE TIMES) (CADDR E) (DERIV (CADR E) X))
               (LIST (QUOTE TIMES) (CADR E) (DERIV (CADDR E) X)))
            (LIST (QUOTE TIMES) (CADDR E) (CADDR E))))
      ((AND (EQ (CAR E) (QUOTE EXPT)) (NUMBERP (CADDR E)))
         (LIST (QUOTE TIMES)
            (LIST (QUOTE TIMES) (CADDR E)
               (LIST (CAR E) (CADR E) (SUB1 (CADDR E))))
            (DERIV (CADR E) X)))
      ((EQ (CAR E) (QUOTE MINUS))
         (LIST (CAR E) (DERIV (CADR E) X)))
      ((EQ (CAR E) (QUOTE EXP))
         (LIST (QUOTE TIMES) E (DERIV (CADR E) X)))
      ((EQ (CAR E) (QUOTE LOG))
         (LIST (QUOTE QUOTIENT) (DERIV (CADR E) X) (CADR E)))
      (T NIL)))
EXPR)
```

Pop-2 : DERIV(expr,X)

```
VARS SUM1 SUM2 DESTSUM OPERATION 4 ++;
RECORDFNS("SUM",[0 0])->SUM1->SUM2->DESTSUM->NONOP ++;
VARS DIFC1 DIFC2 DESTDIFC OPERATION 4 --;
RECORDFNS("DIFC",[0 0])->DIFC1->DIFC2->DESTDIFC->NONOP --;
VARS PROD1 PROD2 DESTPROD OPERATION 3 **;
RECORDFNS("PROD",[0 0])->PROD1->PROD2->DESTPROD->NONOP **;
VARS QUOT1 QUOT2 DESTQUOT OPERATION 3 ///;
RECORDFNS("QUOT",[0 0])->QUOT1->QUOT2->DESTQUOT->NONOP ///;
VARS POWR1 POWR2 DESTPOWR OPERATION 2 ~~;
RECORDFNS("POWR",[0 0])->POWR1->POWR2->DESTPOWR->NONOP ~~;
VARS MINUS1 DESTMINUS MINUS;
RECORDFNS("MINUS",[0])->MINUS1->DESTMINUS->MINUS;
VARS EXPF1 DESTEXPF EXPF;
RECORDFNS("EXPF",[0])->EXPF1->DESTEXPF->EXPF;
VARS LOGF1 DESTLOGF LOGF;
RECORDFNS("LOGF",[0])->LOGF1->DESTLOGF->LOGF;

FUNCTION DERIV E X;
   IF E.ISNUMBER  THEN 0
   ELSEIF E.ISWORD THEN IF E=X THEN 1 ELSE 0 CLOSE
   ELSEIF E.DATAWORD="SUM" THEN DERIV(SUM1(E),X)++DERIV(SUM2(E),X)
   ELSEIF E.DATAWORD="DIFC" THEN DERIV(DIFC1(E),X)--DERIV(DIFC2(E),X)
   ELSEIF E.DATAWORD="PROD" THEN
      DERIV(PROD1(E),X)**PROD2(E)++PROD1(E)**DERIV(PROD2(E),X)
   ELSEIF E.DATAWORD="QUOT" THEN
      (DERIV(QUOT1(E),X)**QUOT2(E)--QUOT1(E)**DERIV(QUOT2(E),X))
         ///QUOT2(E)~~2
   ELSEIF E.DATAWORD="POWR" AND POWR2(E).ISNUMBER THEN
      DERIV(POWR1(E),X)**POWR2(E)**POWR1(E)~~(POWR2(E)-1)
   ELSEIF E.DATAWORD="MINUS" THEN MINUS(DERIV(MINUS1(E),X))
   ELSEIF E.DATAWORD="EXPF" THEN E**DERIV(EXPF1(E),X)
   ELSEIF E.DATAWORD="LOGF" THEN DERIV(LOGF1(E),X)///LOGF1(E)
   ELSE "ERROR"
   CLOSE
END;
```

5.4 serialise

palin25 = "ABLE WAS I ERE I SAW ELBA"

ie. a list of 25 numbers representing the character codes.

Result = [2,3,6,4,1,9,2,8,1,5,1,4,7,4,1,5,1,8,2,9,1,4,6,3,2]

Prolog : serialise(palin25,X)

```
:-mode serialise(+,-).
:-mode pairlists(+,-,-).
:-mode arrange(+,-).
:-mode split(+,+,-,-).
:-mode before(+,+).
:-mode numbered(+,+,-).

serialise(L,R) :-
   pairlists(L,R,A),
   arrange(A,T),
   numbered(T,1,N).

pairlists([X,..L],[Y,..R],[pair(X,Y),..A]) :- pairlists(L,R,A).
pairlists([],[],[]).

arrange([X,..L],tree(T1,X,T2)) :-
   split(L,X,L1,L2),
   arrange(L1,T1),
   arrange(L2,T2).
arrange([],void).

split([X,..L],X,L1,L2) :-!, split(L,X,L1,L2).
split([X,..L],Y,[X,..L1],L2) :- before(X,Y),!, split(L,Y,L1,L2).
split([X,..L],Y,L1,[X,..L2]) :- before(Y,X),!, split(L,Y,L1,L2).
split([],_,[],[]).

before(pair(X1,Y1),pair(X2,Y2)) :- X1<X2.

numbered(tree(T1,pair(X,N1),T2),N0,N) :-
   numbered(T1,N0,N1),
   N2 is N1+1,
   numbered(T2,N2,N).
numbered(void,N,N).
```

Lisp : (SERIALISE palin25)

```
(DEFPROP SERIALISE (LAMBDA (L)
   (PROG (R)
      (SETQ R (DUPLICATE L))
      (NUMBERTREE 1 (ARRANGE (CELLS R)))
      (RETURN R)))
EXPR)

(DEFPROP DUPLICATE (LAMBDA (L)
   (COND ((NULL L) NIL)
      (T (CONS (CAR L) (DUPLICATE (CDR L))))))
EXPR)

(DEFPROP CELLS (LAMBDA (L)
   (COND ((NULL L) NIL)
      (T (CONS L (CELLS (CDR L))))))
EXPR)

(DEFPROP ARRANGE (LAMBDA (L)
   (COND ((NULL L) NIL)
      (T (CONS (CONS (CAR L) (MIDDLEPART (CAAR L) (CDR L)))
               (CONS (ARRANGE (LOWERPART (CAAR L) (CDR L)))
                     (ARRANGE (UPPERPART (CAAR L) (CDR L))))))))
EXPR)

(DEFPROP MIDDLEPART (LAMBDA (X L)
   (COND ((NULL L) NIL)
      ((EQ (CAAR L) X) (CONS (CAR L) (MIDDLEPART X (CDR L))))
      (T (MIDDLEPART X (CDR L)))))
EXPR)

(DEFPROP LOWERPART (LAMBDA (X L)
   (COND ((NULL L) NIL)
      ((LESSP (CAAR L) X) (CONS (CAR L) (LOWERPART X (CDR L))))
      (T (LOWERPART X (CDR L)))))
EXPR)

(DEFPROP UPPERPART (LAMBDA (X L)
   (COND ((NULL L) NIL)
      ((GREATERP (CAAR L) X) (CONS (CAR L) (UPPERPART X (CDR L))))
      (T (UPPERPART X (CDR L)))))
EXPR)

(DEFPROP NUMBERTREE (LAMBDA (N TREE)
   (COND ((NULL TREE) N)
      (T (NUMBERTREE
            (NUMBERLIST
               (NUMBERTREE N
                  (CADR TREE))
               (CAR TREE))
            (CDDR TREE)))))
EXPR)

(DEFPROP NUMBERLIST (LAMBDA (N LO)
   (PROG (L)
      (SETQ L LO)
LOOP  (RPLACA (CAR L) N)
      (SETQ L (CDR L))
      (COND ((NOT (NULL L)) (GO LOOP)))
      (RETURN (ADD1 N))))
EXPR)
```

5.5 query

The solutions to a database query to find countries of similar population density are

```
   [indonesia, 223, pakistan,    219]
   [uk,        650, w_germany,   645]
   [italy,     477, philippines,461]
   [france,    246, china,       244]
   [ethiopia,   77, mexico,       76]
```

Prolog : query([C1,D1,C2,D2])

```
query([C1,D1,C2,D2]):-
   density(C1,D1),
   density(C2,D2),
   D1>D2,
   20*D1<21*D2.

density(C,D) :- pop(C,P), area(C,A), D is (P*100)/A.

/* populations in 100000s,     areas in 1000s of sq. miles. */

pop(china,       8250).          area(china,       3380).
pop(india,       5863).          area(india,       1139).
pop(ussr,        2521).          area(ussr,        8708).
pop(usa,         2119).          area(usa,         3609).
pop(indonesia,   1276).          area(indonesia,    570).
pop(japan,       1097).          area(japan,        148).
pop(brazil,      1042).          area(brazil,      3288).
pop(bangladesh,   750).          area(bangladesh,    55).
pop(pakistan,     682).          area(pakistan,     311).
pop(w_germany,    620).          area(w_germany,     96).
pop(nigeria,      613).          area(nigeria,      373).
pop(mexico,       581).          area(mexico,       764).
pop(uk,           559).          area(uk,            86).
pop(italy,        554).          area(italy,        116).
pop(france,       525).          area(france,       213).
pop(philippines, 415).           area(philippines,  90).
pop(thailand,     410).          area(thailand,     200).
pop(turkey,       383).          area(turkey,       296).
pop(egypt,        364).          area(egypt,        386).
pop(spain,        352).          area(spain,        190).
pop(poland,       337).          area(poland,       121).
pop(s_korea,      335).          area(s_korea,       37).
pop(iran,         320).          area(iran,         628).
pop(ethiopia,     272).          area(ethiopia,     350).
pop(argentina,    251).          area(argentina,   1080).
```

Pop-2 : QUERY(N)

[N is the number of times the test is to be iterated. The strips COUNTRY, POPULATION, AREA are initialised with the appropriate data.]

```
VARS COUNTRY POPULATION AREA;
INIT(25)->COUNTRY;
INIT(25)->POPULATION;
INIT(25)->AREA;

FUNCTION DENSITY I; SUBSCR(I,POPULATION)*100/SUBSCR(I,AREA) END;

FUNCTION QUERY N;
VARS I C1 C2 D1 D2;
   ERASE(ITEMREAD());
   N+1->N;
   FORALL I 1 1 N;
      IF I=N THEN ERASE(ITEMREAD()) CLOSE;
      FORALL C1 1 1 25;
         DENSITY(C1)->D1;
         FORALL C2 1 1 25;
            DENSITY(C2)->D2;
            IF D1>D2 AND 20*D1<21*D2 AND I=N
               THEN PR([% SUBSCR(C1,COUNTRY),D1,
                          SUBSCR(C2,COUNTRY),D2 %]);NL(1)
            CLOSE
         CLOSE
      CLOSE
   CLOSE
END;
```